Extended Structural Equation Modelling

Facilitates treatment of statistical model specifications as things that can be generated and manipulated programmatically. Structural equation models may be specified with reticular action model matrices or paths, linear structural relations matrices or paths, or directly in matrix algebra. Fit functions include full information maximum likelihood, maximum likelihood, and weighted least squares. Example models include confirmatory factor, multiple group, mixture distribution, categorical threshold, modern test theory, differential equations, state space, and many others.

GNU make

Apache License (== 2.0) | file LICENSE

RcppEigen, StanHeaders, BH

R (>= 3.0.2), digest, MASS, methods, parallel

Matrix, mvtnorm, numDeriv, roxygen2 (>= 3.1), Rmpi, rpf (>= 0.36), snowfall

yes

yes

0ClassUnion.R cache.R 'MxBaseNamed.R' 'MxData.R' 'MxDataWLS.R' 'DefinitionVars.R' 'MxReservedNames.R' 'MxNamespace.R' 'MxSearchReplace.R' 'MxFlatSearchReplace.R' 'MxUntitled.R' 'MxAlgebraFunctions.R' 'MxExponential.R' 'MxMatrix.R' 'DiagMatrix.R' 'FullMatrix.R' 'IdenMatrix.R' 'LowerMatrix.R' 'SdiagMatrix.R' 'StandMatrix.R' 'SymmMatrix.R' 'UniMatrix.R' 'ZeroMatrix.R' 'MxMatrixFunctions.R' 'MxAlgebra.R' 'MxCycleDetection.R' 'MxDependencies.R' 'MxAlgebraConvert.R' 'MxAlgebraTransform.R' 'MxSquareBracket.R' 'MxEval.R' 'MxRename.R' 'MxPath.R' 'MxObjectiveMetaData.R' 'MxRAMMetaData.R' 'MxExpectation.R' 'MxExpectationNormal.R' 'MxExpectationRAM.R'
R topics documented:

'MxExpectationLISREL.R' 'MxFitFunction.R' 'MxFitFunctionAlgebra.R' 'MxFitFunctionML.R'
'MxFitFunctionMultigroup.R' 'MxFitFunctionRow.R' 'MxFitFunctionWLS.R'
'MxRAMObjective.R' 'MxLISRELObjective.R' 'MxFIMLObjective.R' 'MxMLEObjective.R'
'MxRowObjective.R' 'MxAlgebraObjective.R' 'MxBounds.R' 'MxConstraint.R' 'MxInterval.R'
'MxTypes.R' 'MxCompute.R' 'MxModel.R' 'MxRAMModel.R' 'MxLISRELModel.R'
'MxModelDisplay.R' 'MxFlatModel.R' 'MxMultiModel.R' 'MxModelFunctions.R'
'MxModelParameters.R' 'MxUnitTesting.R' 'MxApply.R' 'MxRun.R' 'MxRunHelperFunctions.R'
'MxSummary.R' 'MxCompare.R' 'MxSwift.R' 'MxOptions.R' 'MxThreshold.R' 'OriginalMx.R'
'MxGraph.R' 'MxGraphviz.R' 'MxDeparse.R' 'MxCommunication.R' 'MxRestore.R'
'MxVersion.R' 'MxPPML.R' 'MxRAMtoML.R' 'MxDiff.R' 'MxErrorHandling.R'
'omxBrownie.R' 'omxConstrainThresholds.R' 'omxGetNPSOL.R' 'MxFitFunctionGREML.R' 'MxRPC.R'
'MxExpectationBA81.R' 'MxFitFunctionGREML.R' 'MxML.R' 'MxIII.R'
'zzz.R'

Version 2.2.4

R topics documented:

BaseCompute-class .................................................. 7
Bollen .............................................................. 8
cvectorize ......................................................... 9
demoOneFactor .....................................................10
demoTwoFactor .....................................................11
diag2vec ..........................................................12
dztData ...........................................................13
dzmData ...........................................................14
dzoData ...........................................................16
eigenvec .......................................................... 17
example1 ..........................................................18
example2 ..........................................................19
expm ..............................................................20
factorExample1 .....................................................20
factorScaleExample1 ..............................................21
factorScaleExample2 ...............................................22
genericFitDependencies,MxBaseFitFunction-method ...........23
HS.ability.data ....................................................24
imxAddDependency ............................................... 26
imxCheckMatrices ............................................... 26
imxCheckVariables .............................................. 27
imxConDecMatrixSlots ........................................... 27
imxConstraintRelations .......................................... 27
imxConvertIdentifier ............................................ 28
imxConvertLabel ................................................ 28
imxConvertSubstitution ......................................... 29
imxCreateMatrix ............................................... 29
imxDataTypes ...................................................30
imxDefaultGetSlotDisplayNames ................................30
### R topics documented:

- `imxDeparse` ........................................ 31
- `imxDependentModels` .............................. 31
- `imxDetermineDefaultOptimizer` ........................ 31
- `imxDiff` ........................................ 32
- `imxDmvnorm` ........................................ 32
- `imxEvalByName` ...................................... 33
- `imxExtractMethod` .................................. 33
- `imxExtractNames` ................................... 34
- `imxExtractReferences` .............................. 34
- `imxExtractSlot` ..................................... 34
- `imxFilterDefinitionVariables` ....................... 35
- `imxFlattenModel` .................................. 35
- `imxFreezeModel` .................................... 35
- `imxGenerateLabels` .................................. 36
- `imxGenerateNamespace` ................................ 36
- `imxGenericModelBuilder` ............................ 36
- `imxGenSwift` ....................................... 37
- `imxGetSlotDisplayNames` ........................... 37
- `imxHasNPSOL` ...................................... 38
- `imxHasOpenMP` ...................................... 38
- `imxIdentifier` ....................................... 39
- `imxIndependentModels` ................................ 39
- `imxInitModel` ....................................... 39
- `imxIsDefinitionVariable` ............................. 40
- `imxIsPath` .......................................... 40
- `imxLocateFunction` .................................. 40
- `imxLocateIndex` .................................... 41
- `imxLocateLabel` ..................................... 41
- `imxLog` ........................................... 42
- `imxLookupSymbolTable` .............................. 42
- `imxModelBuilder` .................................... 42
- `imxModelTypes` ...................................... 43
- `imxMpiWrap` ........................................ 43
- `imxOriginalMx` ...................................... 44
- `imxPPML` ........................................... 44
- `imxPPML.Test.Battery` ................................ 45
- `imxPPML.Test.Test` .................................. 46
- `imxPreprocessModel` ................................ 46
- `imxReplaceMethod` .................................. 47
- `imxReplaceModels` ................................... 47
- `imxReplaceSlot` ..................................... 48
- `imxReservedNames` ................................... 48
- `imxReverseIdentifier` ................................ 49
- `imxSameType` ....................................... 49
- `imxSeparatorChar` ................................... 49
- `imxSfClient` ....................................... 50
- `imxSimpleRAMPredicate` ............................. 50
- `imxSparseInvert` .................................... 50
R topics documented:

- imxSquareMatrix .......................................................... 51
- imxSymmetricMatrix ....................................................... 51
- imxTypeName .................................................................. 51
- imxUntitledName ............................................................. 52
- imxUntitledNumber .......................................................... 52
- imxUntitledNumberReset ................................................... 52
- imxUpdateModelValues ...................................................... 53
- imxVariableTypes ............................................................. 53
- imxVerifyMatrix ............................................................... 53
- imxVerifyModel ............................................................... 53
- imxVerifyName ............................................................... 54
- imxVerifyReference ........................................................ 54
- imxWlsChiSquare ............................................................. 55
- imxWlsStandardErrors ...................................................... 55
- jointdata .................................................................... 57
- latentMultipleRegExample1 ............................................... 58
- latentMultipleRegExample2 ............................................... 59
- logm ........................................................................ 60
- multiData1 ................................................................... 60
- mxAlgebra .................................................................. 61
- MxAlgebra-class .............................................................. 64
- MxAlgebraFormula-class .................................................. 65
- mxAlgebraObjective ......................................................... 65
- MxBaseExpectation-class .................................................. 67
- MxBaseFitFunction-class .................................................. 67
- MxBaseNamed-class .......................................................... 67
- MxBaseObjectiveMetaData-class ....................................... 68
- mxBounds .................................................................... 68
- MxBounds-class .............................................................. 69
- MxCharOrList-class .......................................................... 70
- MxCharOrNumber-class .................................................... 70
- mxCheckIdentification ..................................................... 70
- mxCI ........................................................................ 72
- MxCI-class ..................................................................... 74
- mxCompare ................................................................. 75
- MxCompute-class ............................................................ 78
- mxComputeConfidenceInterval ......................................... 78
- mxComputeDefault .......................................................... 79
- mxComputeEM ................................................................. 79
- mxComputeGradientDescent ............................................... 81
- mxComputeHessianQuality ................................................ 82
- mxComputeIterate ........................................................... 83
- mxComputeNewtonRaphson .............................................. 83
- mxComputeNothing ........................................................ 84
- mxComputeNumericDeriv .................................................. 84
- mxComputeOnce .............................................................. 85
- mxComputeReportDeriv .................................................... 86
- mxComputeSequence ......................................................... 87
### R topics documented:

<table>
<thead>
<tr>
<th>Class/Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxComputeStandardError</td>
<td>87</td>
</tr>
<tr>
<td>mxConstraint</td>
<td>88</td>
</tr>
<tr>
<td>MxConstraint-class</td>
<td>89</td>
</tr>
<tr>
<td>mxData</td>
<td>90</td>
</tr>
<tr>
<td>MxData-class</td>
<td>93</td>
</tr>
<tr>
<td>mxDataDynamic</td>
<td>94</td>
</tr>
<tr>
<td>MxDataFrameOrMatrix-class</td>
<td>95</td>
</tr>
<tr>
<td>MxDataStatic-class</td>
<td>95</td>
</tr>
<tr>
<td>mxDataWLS</td>
<td>95</td>
</tr>
<tr>
<td>MxDirectedGraph-class</td>
<td>97</td>
</tr>
<tr>
<td>mxEval</td>
<td>97</td>
</tr>
<tr>
<td>MxExpectation-class</td>
<td>99</td>
</tr>
<tr>
<td>mxExpectationBA81</td>
<td>99</td>
</tr>
<tr>
<td>mxExpectationGREML</td>
<td>101</td>
</tr>
<tr>
<td>MxExpectationGREML-class</td>
<td>103</td>
</tr>
<tr>
<td>mxExpectationLISREL</td>
<td>104</td>
</tr>
<tr>
<td>mxExpectationNormal</td>
<td>109</td>
</tr>
<tr>
<td>mxExpectationRAM</td>
<td>111</td>
</tr>
<tr>
<td>mxExpectationStateSpace</td>
<td>114</td>
</tr>
<tr>
<td>mxExpectationStateSpaceContinuousTime</td>
<td>118</td>
</tr>
<tr>
<td>mxFactor</td>
<td>124</td>
</tr>
<tr>
<td>mxFIMLObjective</td>
<td>125</td>
</tr>
<tr>
<td>MxFitFunction-class</td>
<td>128</td>
</tr>
<tr>
<td>mxFitFunctionAlgebra</td>
<td>128</td>
</tr>
<tr>
<td>mxFitFunctionGREML</td>
<td>130</td>
</tr>
<tr>
<td>MxFitFunctionGREML-class</td>
<td>131</td>
</tr>
<tr>
<td>mxFitFunctionML</td>
<td>132</td>
</tr>
<tr>
<td>mxFitFunctionMultigroup</td>
<td>134</td>
</tr>
<tr>
<td>mxFitFunctionR</td>
<td>135</td>
</tr>
<tr>
<td>mxFitFunctionRow</td>
<td>136</td>
</tr>
<tr>
<td>mxFitFunctionWLS</td>
<td>138</td>
</tr>
<tr>
<td>MxFlatModel-class</td>
<td>140</td>
</tr>
<tr>
<td>mxGenerateData</td>
<td>140</td>
</tr>
<tr>
<td>mxGetExpected</td>
<td>142</td>
</tr>
<tr>
<td>mxGREMLDataHandler</td>
<td>143</td>
</tr>
<tr>
<td>MxInterval-class</td>
<td>145</td>
</tr>
<tr>
<td>mxKalmanScores</td>
<td>145</td>
</tr>
<tr>
<td>MxLISRELModel-class</td>
<td>147</td>
</tr>
<tr>
<td>mxLISRELObjective</td>
<td>147</td>
</tr>
<tr>
<td>MxListOrNull-class</td>
<td>151</td>
</tr>
<tr>
<td>mxMakeNames</td>
<td>151</td>
</tr>
<tr>
<td>mxMatrix</td>
<td>152</td>
</tr>
<tr>
<td>MxMatrix-class</td>
<td>154</td>
</tr>
<tr>
<td>mxMI</td>
<td>156</td>
</tr>
<tr>
<td>mxMLObjective</td>
<td>158</td>
</tr>
<tr>
<td>mxModel</td>
<td>160</td>
</tr>
<tr>
<td>MxModel-class</td>
<td>163</td>
</tr>
<tr>
<td>mxOption</td>
<td>165</td>
</tr>
</tbody>
</table>
R topics documented:

<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxOptionalChar-class</td>
<td>168</td>
</tr>
<tr>
<td>MxOptionalCharOrNumber-class</td>
<td>168</td>
</tr>
<tr>
<td>MxOptionalLogical-class</td>
<td>168</td>
</tr>
<tr>
<td>MxOptionalMatrix-class</td>
<td>169</td>
</tr>
<tr>
<td>MxOptionalNumeric-class</td>
<td>169</td>
</tr>
<tr>
<td>mxPath</td>
<td>169</td>
</tr>
<tr>
<td>MxRAMGraph-class</td>
<td>172</td>
</tr>
<tr>
<td>MxRAMMetaData-class</td>
<td>172</td>
</tr>
<tr>
<td>MxRAMModel-class</td>
<td>172</td>
</tr>
<tr>
<td>mxRAMObjective</td>
<td>173</td>
</tr>
<tr>
<td>mxRename</td>
<td>175</td>
</tr>
<tr>
<td>mxRestore</td>
<td>176</td>
</tr>
<tr>
<td>mxRObjective</td>
<td>178</td>
</tr>
<tr>
<td>mxRowObjective</td>
<td>179</td>
</tr>
<tr>
<td>mxRun</td>
<td>181</td>
</tr>
<tr>
<td>mxSave</td>
<td>183</td>
</tr>
<tr>
<td>mxSetDefaultOptions</td>
<td>184</td>
</tr>
<tr>
<td>mxSimplify2Array</td>
<td>185</td>
</tr>
<tr>
<td>mxStandardizeRAMpaths</td>
<td>185</td>
</tr>
<tr>
<td>mxThreshold</td>
<td>188</td>
</tr>
<tr>
<td>mxTryHard</td>
<td>189</td>
</tr>
<tr>
<td>mxTypes</td>
<td>191</td>
</tr>
<tr>
<td>mxVersion</td>
<td>192</td>
</tr>
<tr>
<td>myAutoregressiveData</td>
<td>193</td>
</tr>
<tr>
<td>myFADataRaw</td>
<td>194</td>
</tr>
<tr>
<td>myGrowthKnownClassData</td>
<td>194</td>
</tr>
<tr>
<td>myGrowthMixtureData</td>
<td>195</td>
</tr>
<tr>
<td>myLongitudinalData</td>
<td>196</td>
</tr>
<tr>
<td>myRegData</td>
<td>197</td>
</tr>
<tr>
<td>myRegDataRaw</td>
<td>198</td>
</tr>
<tr>
<td>myTwinData</td>
<td>199</td>
</tr>
<tr>
<td>mzfData</td>
<td>200</td>
</tr>
<tr>
<td>mzmData</td>
<td>201</td>
</tr>
<tr>
<td>Named-entity</td>
<td>203</td>
</tr>
<tr>
<td>nuclear_twin_design_data</td>
<td>203</td>
</tr>
<tr>
<td>numHess1</td>
<td>204</td>
</tr>
<tr>
<td>numHess2</td>
<td>205</td>
</tr>
<tr>
<td>omxAllInt</td>
<td>205</td>
</tr>
<tr>
<td>omxApply</td>
<td>207</td>
</tr>
<tr>
<td>omxAssignFirstParameters</td>
<td>208</td>
</tr>
<tr>
<td>omxBrownie</td>
<td>209</td>
</tr>
<tr>
<td>omxCheckCloseEnough</td>
<td>210</td>
</tr>
<tr>
<td>omxCheckEquals</td>
<td>211</td>
</tr>
<tr>
<td>omxCheckError</td>
<td>212</td>
</tr>
<tr>
<td>omxCheckIdentical</td>
<td>213</td>
</tr>
<tr>
<td>omxCheckNamespace</td>
<td>214</td>
</tr>
<tr>
<td>omxCheckSetEquals</td>
<td>214</td>
</tr>
<tr>
<td>omxCheckTrue</td>
<td>215</td>
</tr>
</tbody>
</table>
This is an internal class and should not be used directly.
Bollen Data on Industrialization and Political Democracy

Description

Data set used in some of OpenMx’s examples, for instance WLS. The data were reported in Bollen (1989, p. 428, Table 9.4) This set includes data from 75 developing countries each assessed on four measures of democracy measured twice (1960 and 1965), and three measures of industrialization measured once (1960).

Usage

data("Bollen")

Format

A data frame with 75 observations on the following 11 numeric variables.

y1 Freedom of the press, 1960
y2 Freedom of political opposition, 1960
y3 Fairness of elections, 1960
y4 Effectiveness of elected legislature, 1960
y5 Freedom of the press, 1965
y6 Freedom of political opposition, 1965
y7 Fairness of elections, 1965
y8 Effectiveness of elected legislature, 1965
x1 GNP per capita, 1960
x2 Energy consumption per capita, 1960
x3 Percentage of labor force in industry, 1960

Details

Variables y1-y4 and y5-y8 are typically used as indicators of the latent trait of “political democracy” in 1960 and 1965 respectively. x1-x3 are used as indicators of industrialization (1960).

Source

The sem package (in turn, via pers. comm Bollen to Fox)
References


Examples

data(Bollen)
str(Bollen)
plot(y1 ~ y2, data = Bollen)

---

cvectorize

Vectorize By Column

description

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

Usage

cvectorize(x)

Arguments

x an input matrix.

See Also

rvectorize, vech, vechs

Examples

cvectorize(matrix(1:9, 3, 3))
cvectorize(matrix(1:12, 3, 4))
Demonstration data for a one factor model

Description
Data set used in some of OpenMx’s examples.

Usage
```
data("demoOneFactor")
```

Format
A data frame with 500 observations on the following 5 numeric variables.

```
x1
x2
x3
x4
x5
```

Details
Variables x1-x5 are typically used as indicators of the latent trait.

Source
Simulated.

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

Examples
```
data(demoOneFactor)
cov(demoOneFactor)
cor(demoOneFactor)
```
Demonstration data for a two factor model

demoTwoFactor

Description

Data set used in some of OpenMx’s examples.

Usage

data("demoTwoFactor")

Format

A data frame with 500 observations on the following 10 numeric variables.

x1
x2
x3
x4
x5
y1
y2
y3
y4
y5

Details

Variables x1-x5 are typically used as indicators of one latent trait. Variables y1-y5 are typically used as indicators of another latent trait.

Source

Simulated.

References

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

Examples

data(demoTwoFactor)
cov(demoTwoFactor)
cor(demoTwoFactor)
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 (i.e., it doesn’t have `diag()`’s functions of returning an Identity matrix from an `nrow` specification, nor to return a matrix wrapped around a diagonal if provided with a vector). To get `vector2matrix` functionality, call `vec2diag`.

### See Also

- `vec2diag`

### Examples

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

diag2vec(matrix(1:12, nrow=3, ncol=4))
#  [,1]
# [1,]  1
# [2,]  5
# [3,]  9
```
\textit{dzfData} \hspace{1cm} \textit{DZ female data}

\begin{itemize}
  \item \textbf{Description} \\
    Data for extended twin example ETC88.R
  \item \textbf{Usage} \\
    \texttt{data("dzfData")}
  \item \textbf{Format} \\
    A data frame with 2007 observations on the following 37 variables.
    \begin{itemize}
      \item \texttt{famid} a numeric vector
      \item \texttt{e1} a numeric vector
      \item \texttt{e2} a numeric vector
      \item \texttt{e3} a numeric vector
      \item \texttt{e4} a numeric vector
      \item \texttt{e5} a numeric vector
      \item \texttt{e6} a numeric vector
      \item \texttt{e7} a numeric vector
      \item \texttt{e8} a numeric vector
      \item \texttt{e9} a numeric vector
      \item \texttt{e10} a numeric vector
      \item \texttt{e11} a numeric vector
      \item \texttt{e12} a numeric vector
      \item \texttt{e13} a numeric vector
      \item \texttt{e14} a numeric vector
      \item \texttt{e15} a numeric vector
      \item \texttt{e16} a numeric vector
      \item \texttt{e17} a numeric vector
      \item \texttt{e18} a numeric vector
      \item \texttt{a1} a numeric vector
      \item \texttt{a2} a numeric vector
      \item \texttt{a3} a numeric vector
      \item \texttt{a4} a numeric vector
      \item \texttt{a5} a numeric vector
      \item \texttt{a6} a numeric vector
    \end{itemize}
\end{itemize}
Examples

```r
data(dzmData)
str(dzmData)
```

---

**dzmData**  

*DZ Male data*

**Description**

Data for extended twin example ETC88.R

**Usage**

```r
data("dzmData")
```

**Format**

A data frame with 1990 observations on the following 37 variables.

- `famid` a numeric vector
- `e1` a numeric vector
- `e2` a numeric vector
- `e3` a numeric vector
- `e4` a numeric vector
- `e5` a numeric vector
- `e6` a numeric vector
- `e7` a numeric vector
- `e8` a numeric vector
- `a7` a numeric vector
- `a8` a numeric vector
- `a9` a numeric vector
- `a10` a numeric vector
- `a11` a numeric vector
- `a12` a numeric vector
- `a13` a numeric vector
- `a14` a numeric vector
- `a15` a numeric vector
- `a16` a numeric vector
- `a17` a numeric vector
- `a18` a numeric vector
- `aQP` a numeric vector
- `aQQ` a numeric vector
- `aQR` a numeric vector
- `aQS` a numeric vector
- `aQT` a numeric vector
- `aQU` a numeric vector
- `aQV` a numeric vector
- `aQW` a numeric vector
- `aQX` a numeric vector
\textit{dzmData}

e9  a numeric vector
e10  a numeric vector
e11  a numeric vector
e12  a numeric vector
e13  a numeric vector
e14  a numeric vector
e15  a numeric vector
e16  a numeric vector
e17  a numeric vector
e18  a numeric vector
a1  a numeric vector
a2  a numeric vector
a3  a numeric vector
a4  a numeric vector
a5  a numeric vector
a6  a numeric vector
a7  a numeric vector
a8  a numeric vector
a9  a numeric vector
a10  a numeric vector
a11  a numeric vector
a12  a numeric vector
a13  a numeric vector
a14  a numeric vector
a15  a numeric vector
a16  a numeric vector
a17  a numeric vector
a18  a numeric vector

\textbf{Examples}

data(dzmData)
str(dzmData)
**dzoData**  
* DZ opposite sex data

**Description**
Data for extended twin example ETC88.R

**Usage**
data("dzoData")

**Format**
A data frame with 3981 observations on the following 37 variables.
- `famid` a numeric vector
- `e1` a numeric vector
- `e2` a numeric vector
- `e3` a numeric vector
- `e4` a numeric vector
- `e5` a numeric vector
- `e6` a numeric vector
- `e7` a numeric vector
- `e8` a numeric vector
- `e9` a numeric vector
- `e10` a numeric vector
- `e11` a numeric vector
- `e12` a numeric vector
- `e13` a numeric vector
- `e14` a numeric vector
- `e15` a numeric vector
- `e16` a numeric vector
- `e17` a numeric vector
- `e18` a numeric vector
- `a1` a numeric vector
- `a2` a numeric vector
- `a3` a numeric vector
- `a4` a numeric vector
- `a5` a numeric vector
- `a6` a numeric vector
eigenvec

a7 a numeric vector
a8 a numeric vector
a9 a numeric vector
a10 a numeric vector
a11 a numeric vector
a12 a numeric vector
a13 a numeric vector
a14 a numeric vector
a15 a numeric vector
a16 a numeric vector
a17 a numeric vector
a18 a numeric vector

Examples

data(dzoData)
str(dzoData)

eigenvec

Eigenvec/Eigenvalue Decomposition

Description

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

Usage

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

Arguments

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

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

See Also

eigen

Examples

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

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

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

---

**example1**  
Bivariate twin data example from Classic Mx Manual

Description

Data set used in some of OpenMx’s examples.

Usage

data("example1")

Format

A data frame with 400 observations on the following variables.

- **IDNum** Twin pair ID
- **Zygosity**  Zygosity of the twin pair
- **X1** X variable for twin 1
- **Y1** Y variable for twin 1
- **X2** X variable for twin 2
- **Y2** Y variable for twin 2
**Details**

Same as `example2` but in wide format instead of tall.

**Source**

Classic Mx Manual.

**References**

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

**Examples**

```r
data(example1)
plot(X2 ~ X1, data=example1)
```

---

**Description**

Data set used in some of OpenMx’s examples.

**Usage**

```r
data("example2")
```

**Format**

A data frame with 800 observations on the following variables.

- `IDNum` ID number
- `TwinNum` Twin ID number
- `Zygosity` Zygosity of the twin
- `X` X variable for twins 1 and 2
- `Y` Y variable for twins 1 and 2

**Details**

Same as `example1` but in tall format instead of wide.

**Source**

Classic Mx Manual.

**References**

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

```r
data(example2)
plot(Y ~ X, data=example2)
```

---

expm

*Matrix exponential*

Description

Matrix exponential

Usage

```r
expm(x)
```

Arguments

- **x**  
  matrix

---

factorExample1

*Example Factor Analysis Data*

Description

Data set used in some of OpenMx’s examples.

Usage

```r
data("factorExample1")
```

Format

A data frame with 500 observations on the following variables.

- x1
- x2
- x3
- x4
- x5
- x6
- x7
- x8
- x9
factorScaleExample1

Details
This appears to be a three factor model, but perhaps with an odd loading structure.

Source
Simulated

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

Examples
data(factorExample1)
round(cor(factorExample1), 2)

factanal(covmat=cov(factorExample1), factors=3, rotation="promax")

factorScaleExample1 Example Factor Analysis Data for Scaling the Model

Description
Data set used in some of OpenMx’s examples.

Usage
data("factorScaleExample1")

Format
A data frame with 200 observations on the following variables.
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
Details

This appears to be a three factor model with factor 1 loading on X1-X4, factor 2 on X5-X8, and factor 3 on X9-X12.

Source

Simulated

References

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

Examples

data(factorScaleExample1)
round(cor(factorScaleExample1), 2)

factorScaleExample2  Example Factor Analysis Data for Scaling the Model

Description

Data set used in some of OpenMx’s examples.

Usage

data("factorScaleExample2")

Format

A data frame with 200 observations on the following variables.

  X1
  X2
  X3
  X4
  X5
  X6
  X7
  X8
  X9
  X10
  X11
  X12
Details

This appears to be a three factor model with factor 1 loading on X1-X4, factor 2 on X5-X8, and factor 3 on X9-X12. It differs from factorScaleExample1 in the scaling of the variables.

Source

Simulated

References

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

Examples

data(factorScaleExample2)
round(cor(factorScaleExample2), 2)

data(factorScaleExample2)
plot(sapply(factorScaleExample1, var), type='l', ylim=c(0, 6), lwd=3)
lines(1:12, sapply(factorScaleExample2, var), col='blue', lwd=3)

genericFitDependencies,MxBaseFitFunction-method

Add dependencies

Description

If there is an expectation, then the fitfunction should always depend on it. Hence, subclasses that implement this method must ignore the passed-in dependencies and use "dependencies <- callNextMethod()" instead.

Usage

## S4 method for signature 'MxBaseFitFunction'
genericFitDependencies(.Object, flatModel, dependencies)

Arguments

/Object fit function object
flatModel flat model that lives with .Object
dependencies accumulated dependency relationships
Description

This classic data set contains intelligence-test scores from 301 children on 26 distinct tests. The data are also available in the MBESS package.

The tests cover mental speed, memory, mathematical-ability, spatial, and verbal ability as listed below.

Usage

data("HS.ability.data")

Format

A data frame with 301 observations on the following 2 variables.

id   student ID number (int)
Gender Sex (Factor w/ 2 levels “Female”, “Male”)
grade Grade in school (integer 7 or 8)
agey Age in years (integer)
agem Age in months (integer)
school School attended (Factor w/2 levels “Grant-White” and “Pasteur”)
addition A speed test (numeric)
code A speed test (numeric)
counting A speed test (numeric)
straight A speed test (numeric)
wordr A memory subtest
numberr A memory subtest
figurer A memory subtest
object A memory subtest
numberf A memory subtest
figurew A memory subtest
deduct A mathematical subtest
numeric A mathematical subtest
problemr A mathematical subtest
series A mathematical subtest
arithmet A mathematical subtest
visual A spatial subtest
**Details**

The data are from children who differ in grade (seventh- and eighth-grade) and are nested in one of two schools (Pasteur and Grant-White). You will see it in use elsewhere, both in R (lavaan, MBESS), and in Joreskog (1969) reporting a cfa on the Grant-White school subject subset).

The last two tests are substitute versions for other tests. paperrev (a paper form board test) can substitute for paper and flagssub for the lozenges test flags.

**Source**


**References**


**Examples**

data(HS.ability.data)
str(HS.ability.data)
levels(HS.ability.data$school)
plot(flags ~ flagssub, data = HS.ability.data)
imxAddDependency  

_add a dependency_

Description

The dependency tracking system ensures that algebra and fit functions are not recomputed if their inputs have not changed. Dependency information is computed prior to handing the model off to the optimizer to reduce overhead during optimization.

Usage

imxAddDependency(source, sink, dependencies)

Arguments

source  
a character vector of the names of the computation sources (inputs)

sink  
the name of the computation sink (output)

dependencies  
the dependency graph

Details

Each free parameter keeps track of all the objects that store that free parameter and the transitive closure of all algebras and fit functions that depend on that free parameter. Similarly, each definition variable keeps track of all the objects that store that free parameter and the transitive closure of all the algebras and fit functions that depend on that free parameter. At each iteration of the optimization, when the free parameter values are updated, all of the dependencies of that free parameter are marked as dirty (see omxFitFunction.repopulateFun). After an algebra or fit function is computed, omxMarkClean() is called to indicate that the algebra or fit function is updated. Similarly, when definition variables are populated in FIML, all of the dependencies of the definition variables are marked as dirty. Particularly for FIML, the fact that non-definition-variable dependencies remain clean is a big performance gain.

imxCheckMatrices  
imxCheckMatrices

Description

This is an internal function exported for those people who know what they are doing.

Usage

imxCheckMatrices(model)

Arguments

model  
model
**imxCheckVariables**

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxCheckVariables(flatModel, namespace)

**Arguments**
- flatModel: flatModel
- namespace: namespace

**imxConDecMatrixSlots**

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxConDecMatrixSlots(object)

**Arguments**
- object: of class MxMatrix

**imxConstraintRelations**

**Description**
A string vector of valid constraint binary relations.

**Usage**
imxConstraintRelations

**Format**
chr [1:3] "<" "==" ">"
**imxConvertIdentifier**

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

`imxConvertIdentifier(identifiers, modelname, namespace)`

**Arguments**

- `identifiers`  
- `modelname`  
- `namespace`

---

**imxConvertLabel**

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

`imxConvertLabel(label, modelname, dataname, namespace)`

**Arguments**

- `label`  
- `modelname`  
- `dataname`  
- `namespace`
imxConvertSubstitution

Description
This is an internal function exported for those people who know what they are doing.

Usage
imxConvertSubstitution(substitution, modelname, namespace)

Arguments
substitution  substitution
modelname    modelname
namespace     namespace

imxCreateMatrix  Create a matrix

Description
This is an internal function exported for those people who know what they are doing.

Usage
imxCreateMatrix(.Object, labels, values, free, lbound, ubound, nrow, ncol,
                 byrow, name, condenseSlots, persist, ...)

Arguments
.Object         the matrix
labels         labels
values         values
free           free
lbound         lbound
ubound         ubound
nrow           nrow
ncol           ncol
byrow          byrow
name           name
condenseSlots  condenseSlots
persist        persist
...            Not used.
### imxDataTypes

**Valid types of data that can be contained by MxData**

#### Description

Valid types of data that can be contained by MxData

#### Usage

```r
imxDataTypes
d
```

#### Format

```r
chr [1:5] "raw" "cov" "cor" "sscp" "acov"
```

---

### imxDefaultGetSlotDisplayNames

#### Description

Returns a list of display-friendly object slot names. This is an internal function exported for those people who know what they are doing.

#### Usage

```r
imxDefaultGetSlotDisplayNames(x, pattern = ".*")
```

#### Arguments

- `x` The object from which to get slot names
- `pattern` Initial pattern to match (default of `.*` matches any)
**imxDeparse**  
*Deparse for MxObjects*

**Description**  
Deparse for MxObjects

**Usage**  
```r
imxDeparse(object, indent = "\t")
```

**Arguments**  
- `object`  
- `indent`  

**imxDependentModels**  
*Are submodels dependence?*

**Description**  
Are submodels dependence?

**Usage**  
```r
imxDependentModels(model)
```

**Arguments**  
- `model`  

**imxDetermineDefaultOptimizer**  
*imxDetermineDefaultOptimizer*

**Description**  
This is an internal function exported for those people who know what they are doing.

**Usage**  
```r
imxDetermineDefaultOptimizer()
```

**Details**  
- Returns a character, the default optimizer
imxDiff

Set difference on regular types or S4 objects

Description

Set difference on regular types or S4 objects

Usage

imxDiff(a, b, slots = c("setequal", "intersect"))

Arguments

a
b
slots

imxDmvnorm

A C implementation of dmvnorm

Description

This API is visible to permit testing. Please do not use.

Usage

imxDmvnorm(loc, mean, sigma)

Arguments

loc
mean
sigma
**imxEvalByName**

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxEvalByName(name, model, compute = FALSE, show = FALSE)
```

**Arguments**
- `name`
- `model`
- `compute`
- `show`

**Details**
This function should not be used in MxSummary. All summary information should be extracted from `runstate`.

**imxExtractMethod**

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxExtractMethod(model, index)
```

**Arguments**
- `model`
- `index`
imxExtractNames

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxExtractNames(lst)
```

**Arguments**

- `lst` : lst

---

imxExtractReferences

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxExtractReferences(lst)
```

**Arguments**

- `lst` : lst

---

imxExtractSlot

**Description**

Checks for and extracts a slot from the object. This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxExtractSlot(x, name)
```

**Arguments**

- `x` : The object
- `name` : the name of the slot
**imxFilterDefinitionVariables**

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxFilterDefinitionVariables(defVars, dataName)
```

**Arguments**

- `defVars` (defVars)
- `dataName` (dataName)

**imx FlattenModel**  
*Remove hierarchical structure from model*

**Description**

Remove hierarchical structure from model.

**Usage**

```r
imx FlattenModel(model, namespace)
```

**Arguments**

- `model` (model)
- `namespace` (namespace)

**imx FreezeModel**  
*Freeze model*

**Description**

Remove free parameters and fit function from model.

**Usage**

```r
imx FreezeModel(model)
```

**Arguments**

- `model` (model)
imxGenerateLabels

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```r
imxGenerateLabels(model)
```

**Arguments**
- `model`    model

imxGenerateNamespace

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```r
imxGenerateNamespace(model)
```

**Arguments**
- `model`    model

imxGenericModelBuilder

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```r
imxGenericModelBuilder(model, lst, name, manifestVars, latentVars, submodels, remove, independent)
```
**Arguments**

- `model` model
- `lst` lst
- `name` name
- `manifestVars` manifestVars
- `latentVars` latentVars
- `submodels` submodels
- `remove` remove
- `independent` independent

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```
imxGenSwift(tc, sites, sfile)
```

**Arguments**

- `tc` tc
- `sites` sites
- `sfile` sfile

**Description**

`imxGetSlotDisplayNames` returns a list of display-friendly object slot names. This is an internal function exported for those people who know what they are doing.

**Usage**

```
imxGetSlotDisplayNames(object, pattern = ".*", slotList = slotNames(object), showDots = FALSE, showEmpty = FALSE)
```
Arguments

- **object**: The object from which to get slot names
- **pattern**: Initial pattern to match (default of ‘.*’ matches any)
- **slotList**: List of slots for which to get display names (default = slotNames(object), i.e., all)
- **showDots**: Include slots whose names start with ‘.’ (default FALSE)
- **showEmpty**: Include slots with length-zero contents (default FALSE)

---

**imxHasNPSOL**  
**imxHasNPSOL**

Description

**imxHasNPSOL**

Usage

`imxHasNPSOL()`

Value

Returns TRUE if the NPSOL proprietary optimizer is compiled and linked with OpenMx. Otherwise FALSE.

---

**imxHasOpenMP**  
**imxHasOpenMP**

Description

This is an internal function exported for those people who know what they are doing.

Usage

`imxHasOpenMP()`
**imxIdentifier**

| **imxIdentifier** | **imxIdentifier** |

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```
imxIdentifier(namespace, name)
```

**Arguments**

- namespace
- name

---

**imxIndependentModels**

| **Are submodels independent?** |

**Description**

Are submodels independent?

**Usage**

```
imxIndependentModels(model)
```

**Arguments**

- model

---

**imxInitModel**

| **imxInitModel** |

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```
imxInitModel(model)
```

**Arguments**

- model
Description

This is an internal function exported for those people who know what they are doing.

Usage

imxIsDefinitionVariable(name)

Arguments

name name

Description

This is an internal function exported for those people who know what they are doing.

Usage

imxIsPath(value)

Arguments

value value

Description

This is an internal function exported for those people who know what they are doing.

Usage

imxLocateFunction(function_name)

Arguments

function_name function_name
**imxLocateIndex**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is an internal function exported for those people who know what they are doing.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>imxLocateIndex(model, name, referant)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>referant</td>
</tr>
</tbody>
</table>

---

**imxLocateLabel**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is an internal function exported for those people who know what they are doing.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>imxLocateLabel(label, model, parameter)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
</tr>
<tr>
<td>model</td>
</tr>
<tr>
<td>parameter</td>
</tr>
</tbody>
</table>
\texttt{imxLog} \hspace{1cm} \textit{Test thread-safe output code}

\subsection*{Description}

This is the code that the backend uses to write diagnostic information to standard error. This function should not be called from R. We make it available only for testing.

\subsection*{Usage}

\begin{verbatim}
imxLog(str)
\end{verbatim}

\subsection*{Arguments}

\begin{itemize}
\item \texttt{str} \hspace{1cm} the character string to output
\end{itemize}

\texttt{imxLookupSymbolTable} \hspace{1cm} \textit{imxLookupSymbolTable}

\subsection*{Description}

This is an internal function exported for those people who know what they are doing.

\subsection*{Usage}

\begin{verbatim}
imxLookupSymbolTable(name)
\end{verbatim}

\subsection*{Arguments}

\begin{itemize}
\item \texttt{name} \hspace{1cm} name
\end{itemize}

\texttt{imxModelBuilder} \hspace{1cm} \textit{imxModelBuilder}

\subsection*{Description}

This is an internal function exported for those people who know what they are doing.

\subsection*{Usage}

\begin{verbatim}
imxModelBuilder(model, lst, name, manifestVars, latentVars, submodels, remove, independent)
\end{verbatim}
Arguments

- model
- lst
- name
- manifestVars
- latentVars
- submodels
- remove
- independent

Details

TODO: It probably makes sense to split this into separate methods. For example, modelAddVariables and modelRemoveVariables could be their own methods. This would reduce some cut & paste duplication.

---

**imxModelTypes**

Description

A list of supported model types

Usage

imxModelTypes

Format

list()

---

**imxMpiWrap**

Description

This is an internal function exported for those people who know what they are doing.

Usage

imxMpiWrap(fun)

Arguments

- fun
**imxOriginalMx**

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxOriginalMx(mx.filename, output.directory)
```

**Arguments**
- `mx.filename`  
- `output.directory`

**imxPPML**

**Description**
Potentially enable the PPML optimization for the given model.

**Usage**
```
imxPPML(model, flag = TRUE)
```

**Arguments**
- `model`  the MxModel to evaluate
- `flag` whether to potentially enable PPML
Description

PPML can be applied to a number of special cases. This function will test the given model for all of these special cases.

Usage

imxPPML.Test.Battery(model, verbose = FALSE, testMissingness = TRUE, testPermutations = TRUE, testEstimates = TRUE, testFakeLatents = TRUE, tolerances = c(0.001, 0.001, 0.001))

Arguments

model          the model to test
verbose        whether to print diagnostics
testMissingness try with missingness
testPermutations try with permutations
testEstimates   examine estimates
testFakeLatents try with fake latents
tolerances      a vector of tolerances

Details

Requirements for model passed to this function: - Path-specified - Means vector must be present - Covariance data (with data means vector) - (Recommended) All error variances should be specified on the diagonal of the S matrix, and not as a latent with a loading only on to that manifest

Function will test across all permutations of: - Covariance vs Raw data - Means vector present vs Means vector absent - Path versus Matrix specification - All orders of permutations of latents with manifests
**imxPreprocessModel**

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxPreprocessModel(model)
```

**Arguments**

- `model` : model
**imxReplaceMethod**

---

### Description

This is an internal function exported for those people who know what they are doing.

### Usage

```r
imxReplaceMethod(x, name, value)
```

### Arguments

- **x**
  - the thing
- **name**
  - name
- **value**
  - value

---

**imxReplaceModels**

---

### Description

Replace parts of a model

### Usage

```r
imxReplaceModels(model, replacements)
```

### Arguments

- **model**
  - model
- **replacements**
  - replacements
### imxReplaceSlot

**Description**

Checks for and replaces a slot from the object. This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxReplaceSlot(x, name, value, check = TRUE)
```

**Arguments**

- `x`: Object
- `name`: The name of the slot
- `value`: Replacement value
- `check`: Check replacement value for validity (default TRUE)

### imxReservedNames

**Description**

Vector of reserved names.

**Usage**

```r
imxReservedNames
```

**Format**

```r
chr [1:6] "data" "objective" "likelihood" "fitfunction" ...
```
imxReverseIdentifier

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxReverseIdentifier(model, name)
```

**Arguments**

- `model`
- `name`

---

imxSameType

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxSameType(a, b)
```

**Arguments**

- `a`
- `b`

---

imxSeparatorChar

**Description**

The character between the model name and the named entity inside the model.

**Usage**

```r
imxSeparatorChar
```

**Format**

```r
chr "." 
```
**imxSfClient**

**Description**

As of snowfall 1.84, the snowfall supervisor process stores an internal state information in a variable named ".sfOption" that is located in the "snowfall" namespace. The snowfall client processes store internal state information in a variable named ".sfOption" that is located in the global namespace.

**Usage**

```cpp
imxSfClient()
```

**Details**

As long as the previous statement is true, then the current process is a snowfall client if-and-only-if exists(".sfOption").

---

**imxSimpleRAMPredicate**

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```cpp
imxSimpleRAMPredicate(model)
```

**Arguments**

- `model` - model

---

**imxSparseInvert**

**Sparse symmetric matrix invert**

**Description**

This API is visible to permit testing. Please do not use.

**Usage**

```cpp
imxSparseInvert(mat)
```

**Arguments**

- `mat` - the matrix to invert
**imxSquareMatrix**

---

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxSquareMatrix(.Object)
```

**Arguments**
- `.Object` .Object

---

**imxSymmetricMatrix**

---

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxSymmetricMatrix(.Object)
```

**Arguments**
- `.Object` .Object

---

**imxTypeName**

---

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
```
imxTypeName(model)
```

**Arguments**
- `model` model
### imxUntitledName

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxUntitledName()

**Details**
Returns a character, the name of the next untitled entity

### imxUntitledNumber

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxUntitledNumber()

**Details**
Increments the untitled number counter and returns its value

### imxUntitledNumberReset

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxUntitledNumberReset()

**Details**
Resets the imxUntitledNumber counter
### imxUpdateModelValues

**Description**

Deprecated. This function does not handle parameters with equality constraints. Do not use.

**Usage**

```r
imxUpdateModelValues(model, flatModel, values)
```

**Arguments**

- `model` model
- `flatModel` flat model
- `values` values to update

### imxVariableTypes

**Description**

This is an internal function exported for those people who know what they are doing.

**Usage**

```r
imxVariableTypes
```

**Format**

```r
chr(0)
```

**Details**

The acceptable variable types
imxVerifyMatrix

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxVerifyMatrix(.Object)

**Arguments**
- .Object

---
imxVerifyModel

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxVerifyModel(model)

**Arguments**
- model

---
imxVerifyName

**Description**
This is an internal function exported for those people who know what they are doing.

**Usage**
imxVerifyName(name, stackNumber)

**Arguments**
- name
- stackNumber
**imxVerifyReference**  

**Description**  
This is an internal function exported for those people who know what they are doing.

**Usage**  
`imxVerifyReference(reference, stackNumber)`

**Arguments**
- `reference`: reference  
- `stackNumber`: stackNumber

---

**imxWlsChiSquare**  

**Calculate Chi Square for a WLS Model**

**Description**  
This is an internal function used to calculate the Chi Square distributed fit statistic for weighted least squares models.

**Usage**  
`imxWlsChiSquare(model, J=NA)`

**Arguments**
- `model`: An MxModel object with acov (WLS) data
- `J`: Optional precomputed Jacobian matrix

**Details**
The Chi Square fit statistic for models fit with maximum likelihood depends on the difference in model fit in minus two log likelihood units between the saturated model and the more restricted model under investigation. For models fit with weighted least squares a different expression is required. If $J$ is the first derivative (Jacobian) of the mapping from the free parameters to the unique elements of the expected covariance, means, and thresholds, $J_c$ is the orthogonal complement of $J$, $W$ is the inverse of the full weight matrix, and $e$ is the difference between the sample-estimated and model-implied covariance, means, and thresholds, then the Chi Square fit statistic is

$$
\chi^2 = e' J_c (J_c' W J_c)^{-1} J_c' e
$$

with $e'$ indicating the transpose of $e$. This Equation 2.20a from Browne (1984) where he showed that this statistic is chi-square distributed with the conventional degrees of freedom.
Value
A named list with components

Chi numeric value of the Chi Square fit statistic.
ChiDoF degrees of freedom for the Chi Square fit statistic.

References

---

**imxWlsStandardErrors**  
*Calculate Standard Errors for a WLS Model*

Description
This is an internal function used to calculate standard errors for weighted least squares models.

Usage

`imxWlsStandardErrors(model)`

Arguments

model An MxModel object with acov (WLS) data

Details
The standard errors for models fit with maximum likelihood are related to the second derivative (Hessian) of the likelihood function with respect to the free parameters. For models fit with weighted least squares a different expression is required. If $J$ is the first derivative (Jacobian) of the mapping from the free parameters to the unique elements of the expected covariance, means, and thresholds, $V$ is the weight matrix used, $W$ is the inverse of the full weight matrix, and $U = VJ(J'VJ)^{-1}$, then the asymptotic covariance matrix of the free parameters is

$$Acov(\theta) = U'WU$$

with $U'$ indicating the transpose of $U$.

Value
A named list with components

SE The standard errors of the free parameters
Cov The full covariance matrix of the free parameters. The square root of the diagonal elements of Cov equals SE.
Jac The Jacobian computed to obtain the standard errors.
References


Joint Ordinal and continuous variables to be modeled together

Description

Data set used in some of OpenMx’s examples.

Usage

data("jointdata")

Format

A data frame with 250 observations on the following variables.

- z1 Continuous variable
- z2 Ordinal variable with 2 levels (0, 1)
- z3 Continuous variable
- z4 Ordinal variable with 4 levels (0, 1, 2, 3)
- z5 Ordinal variable with 3 levels (0, 1, 3)

Details

Data generated to test the joint ML algorithm thoroughly.

Source

Simulated.

References

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

Examples

data(jointdata)
par(mfrow=c(2, 3))
h <- lapply(jointdata, hist)
par(mfrow=c(1, 1))

plot(z2 ~ z1, jointdata)
Example data for multiple regression among latent variables

Description
Data set used in some of OpenMx's examples.

Usage
data("latentMultipleRegExample1")

Format
A data frame with 200 observations on the following variables.
X1  Factor 1 indicator
X2  Factor 1 indicator
X3  Factor 1 indicator
X4  Factor 1 indicator
X5  Factor 2 indicator
X6  Factor 2 indicator
X7  Factor 2 indicator
X8  Factor 2 indicator
X9  Factor 3 indicator
X10 Factor 3 indicator
X11 Factor 3 indicator
X12 Factor 3 indicator

Details
Factor 1 strongly predicts factor 3. Factor 2 weakly predicts factor 3.

Source
Simulated.

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

Examples
data(latentMultipleRegExample1)
round(cor(latentMultipleRegExample1), 2)
Data set used in some of OpenMx’s examples.

Usage

data("latentMultipleRegExample2")

Format

A data frame with 200 observations on the following variables.

xQ Factor 1 indicator
xR Factor 1 indicator
xS Factor 1 indicator
xT Factor 1 indicator
xU Factor 2 indicator
xV Factor 2 indicator
xW Factor 2 indicator
xX Factor 2 indicator
xY Factor 3 indicator
xQP Factor 3 indicator
xQQ Factor 3 indicator
xQR Factor 3 indicator

Details

Factor 1 strongly predicts factor 3. Factor 2 weakly predicts factor 3. Very similar to latentMultipleRegExample1.

Source

Simulated.

References

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

Examples

data(latentMultipleRegExample2)
round(cor(latentMultipleRegExample2), 2)
**logm**

*Matrix logarithm*

---

**Description**

Matrix logarithm

**Usage**

```r
logm(x, tol = .Machine$double.eps)
```

**Arguments**

- `x` matrix
- `tol` tolerance

---

**multiData1**

*Data for multiple regression*

---

**Description**

Data set used in some of OpenMx’s examples.

**Usage**

```r
data("multiData1")
```

**Format**

A data frame with 500 observations on the following variables.

- `x1`
- `x2`
- `x3`
- `x4`
- `y`

**Details**

`x1`-`x4` are predictor variables, and `y` is the outcome.

**Source**

Simulated.
mxAlgebra

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

Examples

```r
data(multiData1)
summary(lm(y ~ ., data=multiData1))
#results can be replicated in OpenMx.
```

mxAlgebra

Create MxAlgebra Object

Description
This function creates a new MxAlgebra object.

Usage

```r
mxAlgebra(expression, name = NA, dimnames = NA, ..., fixed = FALSE)
```

Arguments

- `expression`: An R expression of OpenMx-supported matrix operators and matrix functions.
- `name`: An optional character string indicating the name of the object.
- `dimnames`: list. The dimnames attribute for the algebra: a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.
- `...`: Not used. Forces argument ‘fixed’ to be specified by name.
- `fixed`: If TRUE, this algebra will not be recomputed automatically when things it depends on change. `mxComputeOnce` can be used to force it to recompute.

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 mxFitFunctionAlgebra function. The mxFitFunctionAlgebra function must reference by name the MxAlgebra object to be evaluated.

The following operators and functions are supported in mxAlgebra:

- `solve()` Inversion
- `t()` Transposition
- `^` Elementwise powering
%\% Kronecker powering
+ Addition
- Subtraction
%\% Matrix Multiplication
* Elementwise product
/ Elementwise division
%\% Kronecker product
%\&% Quadratic product

Functions

cov2cor Convert covariance matrix to correlation matrix
cov Cholesky Decomposition
cbind Horizontal adhesion
cbind Vertical adhesion
det Determinant
tr Trace
sum Sum
prod Product
max Maximum
min Min
abs Absolute value
sin Sine
sinh Hyperbolic sine
cos Cosine
cosh Hyperbolic cosine	an Tangent	anh Hyperbolic tangent
exp Exponent
log Natural Logarithm
sqrt Square root
p2z Standard-normal quantile
lgamma Log-gamma function
eigenval Eigenvalues of a square matrix. Usage: eigenval(x); eigenvect(x); ieigenval(x); ieigenvect(x)
rvectorize Vectorize by row
cvectorize Vectorize by column
vech Half-vectorization
vechs Strict half-vectorization
mxAlgebra

vech2full  Inverse half-vectorization
vechs2full  Inverse strict half-vectorization
vec2diag  Create matrix from a diagonal vector (similar to diag)
diag2vec  Extract diagonal from matrix (similar to diag)
expm  Matrix Exponential
logm  Matrix Logarithm
omxExponential  Matrix Exponential
omxMnor  Multivariate Normal Integration
omxAllInt  All cells Multivariate Normal Integration
omxNot  Perform unary negation on a matrix
omxAnd  Perform binary and on two matrices
omxOr  Perform binary or on two matrices
omxGreaterThan  Perform binary greater on two matrices
omxLessThan  Perform binary less than on two matrices
omxApproxEquals  Perform binary equals to (within a specified epsilon) on two matrices

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. mxFitFunctionAlgebra for an objective function which takes an MxAlgebra or MxMatrix object as the function to be minimized. MxMatrix and mxMatrix for objects which may be entered in the expression argument and the function that creates them. More information about the OpenMx package may be found here.

Examples

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

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

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

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

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

# Numbers in mxAlgebras are upgraded to 1x1 matrices
# Example of Kronecker powering (%%) and multiplication (%%)
A <- mxMatrix(type="Full", nrow=3, ncol=3, values=c(1:9), name="A")
ml <- mxModel(model="kron", A, mxAlgebra(A %*% 2, name="KroneckerPower"))
mxRun(ml)$KroneckerPower

# Running kron
# mxAlgebra 'KroneckerPower'
# $formula: A %*% 2
# $result:
# [,1] [,2] [,3]
# [1,]  1  16  49
# [2,]  4  25  64
# [3,]  9  36  81

---

**MxAlgebra-class**

**MxAlgebra Class**

**Description**

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

**Details**

The MxAlgebra class has the following slots:

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

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

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

The ‘result’ slot is used to hold the results of computing the expression in the ‘formula’ slot. If the containing model has not been executed, then the ‘result’ slot will hold a 0 x 0 matrix. Otherwise the slot will store the computed value of the algebra using the final estimates of the free parameters.
Slots may be referenced with the $ symbol. See the documentation for Classes and the examples in the mxAlgebra document for more information.

References

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

See Also

mxAlgebra, mxMatrix, MxMatrix

mxAlgebraObjective

DEPRECATED: Create MxAlgebraObjective Object

Description

WARNING: Objective functions have been deprecated as of OpenMx 2.0.

Please use MxFitFunctionAlgebra() instead. As a temporary workaround, MxAlgebraObjective returns a list containing a NULL MxExpectation object and an MxFitFunctionAlgebra object.

All occurrences of

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

Should be changed to

mxFitFunctionAlgebra(algebra, numObs = NA, numStats = NA)

Arguments

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

numObs (optional) An adjustment to the total number of observations in the model.

numStats (optional) An adjustment to the total number of observed statistics in the model.
mxAlgebraObjective

Details

NOTE: THIS DESCRIPTION IS DEPRECATED. Please change to using mxFitFunctionAlgebra as shown in the example below.

Fit functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. While the other fit functions in OpenMx require an expectation function for the model, the mxAlgebraObjective function uses the referenced MxAlgebra or MxMatrix object as the function to be minimized.

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

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

Value

Returns a list containing a NULL MxExpectation object and an MxFitFunctionAlgebra object. MxFitFunctionAlgebra objects should be included with models with referenced MxAlgebra and MxMatrix objects.

References

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

See Also

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

Examples

# Create and fit a very simple model that adds two numbers using mxFitFunctionAlgebra

library(OpenMx)

# Create a matrix 'A' with no free parameters
A <- mxMatrix('Full', nrow = 1, ncol = 1, values = 1, 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 <- mxFitFunctionAlgebra('B')

# Place the algebra, its associated matrix and
MxBaseExpectation-class

Description

The virtual base class for all expectations. Expectations contain enough information to generate simulated data. This is an internal class and should not be used directly.

See Also

mxExpectationNormal, mxExpectationRAM, mxExpectationLISREL, mxExpectationStateSpace, mxExpectationBA81

MxBaseFitFunction-class

Description

The virtual base class for all fit functions. This is an internal class and should not be used directly.

See Also

mxFitFunctionAlgebra, mxFitFunctionML, mxFitFunctionMultigroup, mxFitFunctionR, mxFitFunctionWLS, mxFitFunctionRow, mxFitFunctionGREML

MxBaseNamed-class

Description

This is an internal class and should not be used directly. It is the base class for named entities. Fit functions, expectations, and computes contain this class.
### MxBaseObjectiveMetaData-class

**MxBaseObjectiveMetaData**

**Description**

This is an internal class and should not be used directly. It is the virtual base class for all objective functions meta-data.

### `mxBounds` Create MxBounds Object

**Description**

This function creates a new MxBounds object.

**Usage**

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

**Arguments**

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

**Details**

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

Minima and maxima may be specified as scalar numeric values.

**Value**

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

**References**

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

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)

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.
mxCheckIdentification

Description

Use the dimension of the null space of the Jacobian to determine whether or not a model is identified local to its current parameter values. The output is a list of the the identification status, the Jacobian, and which parameters are not identified.

Usage

mxCheckIdentification(model, details=TRUE)

Arguments

model A MxModel object or list of MxModel objects.
details logical.

Details

The mxCheckIdentification function is used to check that a model is identified. That is, the function will tell you if the model has a unique solution in parameter space. The function is most useful when applied to either (a) a model that has been run and had some NA standard errors, or (b) a model that has not been run but has reasonable starting values. In the former situation, mxCheckIdentification is used as a diagnostic after a problem was indicated. In the latter situation, mxCheckIdentification is used as a sanity check.

The method uses the Jacobian of the model expected means and the unique elements of the expected covariance matrix with respect to the free parameters. It is the first derivative of the mapping between the free parameters and the sufficient statistics for the Normal distribution. The method
does not depend on data, but does depend on the current values of the free parameters. Thus, it only provides local identification, not global identification. Because the method does not depend on data, the model still could be empirically unidentified due to missing data.

The Jacobian is evaluated numerically and generally takes a few seconds, but much less than a minute.

The identification may not be accurate for model where definition variables are used. Currently, only the first row of the definition variable is evaluated.

When TRUE, the 'details' argument provides the names of the non-identified parameters. Otherwise, only the status and Jacobian are returned.

Value

A named list with components

status logical. TRUE if the model is locally identified; otherwise FALSE.

jacobian matrix. The numerically evaluated Jacobian.

non_identified_parameters vector. The free parameter names that are not identified

References


See Also

mxModel

Examples

require(OpenMx)

data(demoOneFactor)
manifests <- names(demoOneFactor)
latents <- "G1"
model2 <- mxModel(model="One Factor", type="RAM",
  manifestVars = manifests,
  latentVars = latents,
  mxPath(from = latents[1], to=manifests[1:5]),
  mxPath(from = manifests, arrows = 2, lbound=1e-6),
  mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
  mxData(cov(demoOneFactor), type = "cov", numObs=500)
)
fit2 <- mxRun(model2)

id2 <- mxCheckIdentification(fit2)
id2$status
# The model is locally identified

# Build a model from the solution of the previous one
# but now the factor variance is also free
model2n <- mxModel(fit2, name="Non Identified Two Factor",
  mxPath(from=latents[1], arrows=2, free=TRUE, values=1)
)

mid2 <- mxCheckIdentification(model2n)
mid2$non_identified_parameters
# The factor loadings and factor variance
# are not identified.

---

## mxCI

### Create mxCI Object

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

A typical use case is when a parameter estimate is obtained that is at or near a lower bound. In this case, there is no point in computing the lower part of the CI. Only the upper bound is needed. In all
cases, a two-sided hypothesis test is assumed. Therefore, the upper bound will exclude 2.5% (for interval=0.95) even though only one bound is requested. To obtain a one-sided CI for a one-sided hypothesis test, interval=0.90 will obtain a 95% confidence interval.

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

Estimation of likelihood-based confidence intervals begins after optimization has been completed, with each parameter moved in the direction(s) specified in the ‘type’ argument until the specified increase in -2 log likelihood is reached. All other free parameters are left free for this stage of optimization. This process repeats until all confidence intervals have been calculated. The calculation of likelihood-based confidence intervals can be computationally intensive, and may add a significant amount of time to model estimation when many confidence intervals are requested.

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

Confidence interval estimation may result in model non-convergence at the confidence limit. Separate optimizer messages may be passed for each confidence limit. This has no impact on the parameter estimates themselves, but may indicate a problem with the referenced confidence limit. Model non-convergence for a particular confidence limit may indicate parameter interdependence or the influence of a parameter boundary.

These error messages and their meanings are listed in the help for mxSummary.

The validity of a confidence limit can be checked by running a model with the appropriate parameter fixed at the confidence limit in question. If the confidence limit is valid, the -2 log likelihoods of these two models should differ by the specified chi-squared criterion (as set using the ‘lowerdelta’ or ‘upperdelta’ slots in the MxCI object (you can choose which of these to set via the type parameter of mxCI).

Value

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

References

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

MxCI for the S4 class created by mxCI. mxComputeConfidenceInterval is the internal compute plan that implements the algorithm. MxMatrix and mxMatrix for free parameter specification. More information about the OpenMx package may be found here.

Examples

```r
library(OpenMx)

# generate data
covariance <- matrix(c(1.0, 0.5, 0.5, 1.0),
                      nrow=2,
                      dimnames=list(c("a", "b"), c("a", "b")))
data <- mxData(covariance, "cov", numObs=100)

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

# request 95 percent confidence intervals
ci <- mxCI(c("var1", "cov12", "var2"))

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

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

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

# view all results
summary(results)
```

---

**MxCI-class**

**MxCI Class**

**Description**

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

The MxCI class has the following slots:

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

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

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

Likelihood-based confidence intervals may be specified by including one or more MxCI objects in an MxModel object. Estimation of confidence intervals requires model optimization using the mxRun function with the ‘intervals’ argument set to TRUE. The calculation of likelihood-based confidence intervals can be computationally intensive, and may add a significant amount of time to model estimation when many confidence intervals are requested.

References

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

See Also

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

mxCompare Assign Model Parameters

Description

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

Usage

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

<table>
<thead>
<tr>
<th>base</th>
<th>A MxModel object or list of MxModel objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison</td>
<td>A MxModel object or list of MxModel objects.</td>
</tr>
<tr>
<td>...</td>
<td>Not used. Forces remaining arguments to be specified by name.</td>
</tr>
<tr>
<td>all</td>
<td>A boolean value on whether to compare all bases with all comparisons. Defaults to FALSE.</td>
</tr>
</tbody>
</table>

details

The mxCompare function is used to compare the fit of one or more MxMatrix objects with output to one or more comparison models. Fit statistics for the comparison model or models are subtracted from the fit statistics for the base model or models. All models included in the ‘base’ argument are also listed without comparison (compared to a <NA> model) to present their raw fit statistics.

Model comparisons are made by subtracting the fit of the comparison model from the fit of a base model. To make sure that the differences between models are positive and yield p-values for likelihood ratio tests, the model or models listed in the ‘base’ argument should be more saturated (i.e., more estimated parameters and fewer degrees of freedom) than models listed in the ‘comparison’ argument. If a comparison is made where the comparison model has a higher minus 2 log likelihood (-2LL) than the base model, then the difference in their -2LLs will be negative. P-values for likelihood ratio tests will not be reported when either the -2LL or degrees of freedom for the comparison are negative.

When multiple models are included in both the ‘base’ and ‘comparison’ arguments, then comparisons are made between the two lists of models based on the value of the ‘all’ argument. If ‘all’ is set to FALSE (default), then the first model in the ‘base’ list is compared to the first model in the ‘comparison’ list, second with second, and so on. If there are an unequal number of ‘base’ and ‘comparison’ models, then the shorter list of models is repeated to match the length of the longer list. For example, comparing base models ‘B1’ and ‘B2’ with comparison models ‘C1’, ‘C2’ and ‘C3’ will yield three comparisons: ‘B1’ with ‘C1’, ‘B2’ with ‘C2’, and ‘B1’ with ‘C3’. Each of those comparisons are prefaced by a comparison between the base model and a missing comparison model to present the fit of the base model.

If ‘all’ is set to TRUE, all possible comparisons between base and comparison models are made, and one entry is made for each base model. All comparisons involving the first model in ‘base’ are made first, followed by all comparisons with the second ‘base’ model, and so on. When there are multiple models in either the ‘base’ or ‘comparison’ arguments but not both, then the ‘all’ argument does not affect the set of comparisons made.

The following columns appear in the output:

<table>
<thead>
<tr>
<th>base</th>
<th>Name of the base model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison</td>
<td>Name of the comparison model. Is &lt;NA&gt; for the first</td>
</tr>
<tr>
<td>ep</td>
<td>Estimated parameters of the comparison model.</td>
</tr>
<tr>
<td>minus2LL</td>
<td>Minus 2<em>log-likelihood of the comparison model. If the comparison model is &lt;NA&gt;, then the minus 2</em>log-likelihood of the base model is given.</td>
</tr>
<tr>
<td>df</td>
<td>Degrees in freedom of the comparison model. If the comparison model is &lt;NA&gt;, then the degrees of freedom of the base model is given.</td>
</tr>
</tbody>
</table>
AIC  Akaike’s Information Criterion for the comparison model. If the comparison model is <NA>,
then the AIC of the base model is given.

diffLL  Difference in minus 2\*log-likelihoods of the base and comparison models. Will be positive
when base model -2LL is higher than comparison model -2LL.

diffdf  Difference in degrees of freedoms of the base and comparison models. Will be positive
when base model DF is lower than comparison model DF (base model estimated parameters
is higher than comparison model estimated parameters)

p  P-value for likelihood ratio test based on diffLL and diffdf values.

The mxCompare function will give a p-value for any comparison in which both ‘diffLL’ and ‘diffdf’
are non-negative. However, this p-value is based on the assumptions of the likelihood ratio test,
specifically that the two models being compared are nested. The likelihood ratio test and associated
p-values are not valid when the comparison model is not nested in the referenced base model.

Use options(‘digits’ = N) to set the minimum number of significant digits to be printed in values.
The mxCompare function does not directly accept a digits argument, and depends on the value of
the ‘digits’ option.

See Also

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

Examples

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

fit1 <- mxRun(model1)

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

fit2 <- mxRun(model2)
mxCompare(fit1, fit2)

# vary precision of the output
oldPrecision = as.numeric(options('digits'))
options('digits' = 1)
mxCompare(fit1, fit2)
options('digits' = oldPrecision)

---

**MxCompute-class**  
*MxCompute*

**Description**

This is an internal class and should not be used directly.

---

**mxComputeConfidenceInterval**  
*Find likelihood-based confidence intervals*

**Description**

There are various ways to pose an equivalent profile likelihood problem. For good performance, it is essential to tailor the problem to the abilities of the optimizer. The problem can be posed without the use of constraints. This is how the code worked in version 2.1 and prior. Although this way of posing the problem creates an ill-conditioned Hessian, NPSOL is somehow able to isolate the poor conditioning from the rest of the problem and optimize it quickly. However, SLSQP is not so clever and exhibits very poor performance. For SLSQP, good performance is contingent on posing the problem using an inequality constraint on the fit.

**Usage**

```
mxComputeConfidenceInterval(plan, ..., freeset = NA_character_,
                           verbose = 0L, engine = NULL, fitfunction = "fitfunction",
                           tolerance = NA_real_, constraintType = "ineq")
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>compute plan to optimize the model</td>
</tr>
<tr>
<td>...</td>
<td>Not used. Forces remaining arguments to be specified by name.</td>
</tr>
<tr>
<td>freeset</td>
<td>names of matrices containing free variables</td>
</tr>
<tr>
<td>verbose</td>
<td>level of debugging output</td>
</tr>
<tr>
<td>engine</td>
<td>deprecated</td>
</tr>
<tr>
<td>fitfunction</td>
<td>The deviance function to constrain with an inequality constraint.</td>
</tr>
<tr>
<td>tolerance</td>
<td>deprecated</td>
</tr>
<tr>
<td>constraintType</td>
<td>one of c('ineq', 'eq', 'both', 'none')</td>
</tr>
</tbody>
</table>
**Details**

Geometrically, SLSQP performs best on smooth likelihood surfaces with smooth derivatives. In the profile CI problem, the distance limit on the deviance is like a wall. Walls do not have smooth derivatives but are more like a step function. The point of `mxConstraint` is to isolate the parts of a problem that are geometrically non-smooth. Constraints are dealt with specially in SLSQP to best accommodate their sharp geometry.

For the default compute plan, the choice of `constraintType` is determined by which optimizer is selected.

**References**


---

**mxComputeDefault**

*Default compute plan*

**Description**

The default compute plan is approximately as follows: `mxComputeSequence(list(mxComputeGradientDescent(), mxComputeConfidenceInterval(), mxComputeNumericDeriv(), mxComputeStandardError(), mxComputeReportDeriv()))`

**Usage**

`mxComputeDefault(freeSet = NA_character_())`

**Arguments**

- `freeSet` names of matrices containing free variables

---

**mxComputeEM**

*Fit a model using DLR’s (1977) Expectation-Maximization (EM) algorithm*

**Description**

The EM algorithm constitutes the following steps: Start with an initial parameter vector. Predict the missing data to form a completed data model. Optimize the completed data model to obtain a new parameter vector. Repeat these steps until convergence criteria are met.

**Usage**

`mxComputeEM(expectation, predict, mstep, observedFit = "fitfunction", ..., `maxIter = 500L, tolerance = 1e-09, verbose = 0L, `freeSet = NA_character_, accel = "varadhan2008", `information = NA_character_, infoArgs = list())`
Arguments

expectation a vector of expectation names
predict what to predict from the observed data (available options depend on the expectation)
mstep a compute plan to optimize the completed data model
observedFit the name of the observed data fit function (defaults to "fitfunction")
... Not used. Forces remaining arguments to be specified by name.
maxIter maximum number of iterations
tolerance optimization is considered converged when the maximum relative change in fit is less than tolerance
verbose level of diagnostic output
freeSet names of matrices containing free variables
accel name of acceleration method ("varadhan2008" or "ramsay1975")
information name of information matrix approximation method
infoArgs arguments to control the information matrix method

Details

This compute plan does not work with any and all expectations. It requires a special kind of expectation that can predict its missing data to create a completed data model.

The EM algorithm does not produce a parameter covariance matrix for standard errors. S-EM, an implementation of Meng & Rubin (1991), is included.

References


**mxComputeGradientDescent**

*Optimize parameters using a gradient descent optimizer*

**Description**

This optimizer does not require analytic derivatives of the fit function. The open-source version of OpenMx only offers 1 choice, SLSQP (from the NLOPT collection). The proprietary version of OpenMx offers the choice of two optimizers, SLSQP and NPSOL.

**Usage**

```r
mxComputeGradientDescent(freeSet = NA_character_, ..., engine = NULL, 
fitfunction = "fitfunction", verbose = 0L, tolerance = NA_real_, 
useGradient = NULL, warmStart = NULL, nudgeZeroStarts = TRUE, 
maxMajorIter = NULL, gradientAlgo = mxOption(NULL, "Gradient algorithm"), 
gradientIterations = 2, gradientStepSize = 1e-05)
```

**Arguments**

- **freeset**: names of matrices containing free variables
- ...: Not used. Forces remaining arguments to be specified by name.
- **engine**: specific NPSOL or SLSQP
- **fitfunction**: name of the fitfunction (defaults to 'fitfunction')
- **verbose**: level of debugging output
- **tolerance**: how close to the optimum is close enough (also known as the optimality tolerance)
- **useGradient**: whether to use the analytic gradient (if available)
- **warmStart**: a Cholesky factored Hessian to use as the NPSOL Hessian starting value (pre-conditioner)
- **nudgeZeroStarts**: whether to nudge any zero starting values prior to optimization (default TRUE)
- **maxMajorIter**: maximum number of major iterations
- **gradientAlgo**: one of c('forward','central')
- **gradientIterations**: number of Richardson iterations to use for the gradient (default 2)
- **gradientStepSize**: the step size for the gradient (default 1e-5)
One of the most important options for SLSQP is `gradientalgo`. By default, the `forward` method is used. This method requires `gradientiterations` function evaluations per parameter per gradient. This method often works well enough but can result in imprecise gradient estimations that may not allow SLSQP to fully optimize a given model. If code red is reported then you are encouraged to try the `central` method. The central method requires 2 times `gradientiterations` function evaluations per parameter per gradient, but it can be much more accurate.

---

### References


### Examples

```r
# Load the data and define the model

data(demoOneFactor)
factorModel <- mxModel(name = "One Factor",
  mxMatrix(type="Full", nrow=5, ncol=1, free=FALSE, values=0.2, name="A"),
  mxMatrix(type="Symm", nrow=1, ncol=1, free=FALSE, values=1, name="L"),
  mxMatrix(type="Diag", nrow=5, ncol=5, free=TRUE, values=1, name="U"),
  mxAlgebra(expression=A %*% L %*% t(A) + U, name="R"),
  mxExpectationNormal(covariance="R", dimnames=names(demoOneFactor)),
  mxFitFunctionML()
)

# Fit the model and compute the Hessian quality

dataModelFit <- mxRun(factorModel)
dataModelFit$output$hessianQuality
```

---

### Description

Tests whether the Hessian is positive definite (model$output$infoDefinite) and, if so, computes the condition number (model$output$conditionNumber). See Luenberger & Ye (2008) Second Order Test (p. 190) and Condition Number (p. 239).

### Usage

```r
mxComputeHessianQuality(freeSet = NA_character_)
```

### Arguments

- **freeSet**: names of matrices containing free variables
References

mxComputeIterate
Repeatedly invoke a series of compute objects until change is less than tolerance

Description
One step (typically the last) must compute the fit or maxAbsChange.

Usage
mxComputeIterate(steps, ..., maxIter = 500L, tolerance = 1e-09,
                   verbose = 0L, freeSet = NA_character_)

Arguments
steps a list of compute objects
... Not used. Forces remaining arguments to be specified by name.
maxIter the maximum number of iterations
tolerance iterates until maximum relative change is less than tolerance
verbose level of debugging output
freeSet Names of matrices containing free variables.

mxComputeNewtonRaphson
Optimize parameters using the Newton-Raphson algorithm

Description
This optimizer requires analytic 1st and 2nd derivatives of the fit function. Comprehensive diagnostics are available by increasing the verbose level.

Usage
mxComputeNewtonRaphson(freeSet = NA_character_, ...,
                        fitfunction = "fitfunction", maxIter = 100L, tolerance = 1e-12,
                        verbose = 0L)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeSet</td>
<td>names of matrices containing free variables</td>
</tr>
<tr>
<td>...</td>
<td>Not used. Forces remaining arguments to be specified by name.</td>
</tr>
<tr>
<td>fitfunction</td>
<td>name of the fitfunction (defaults to 'fitfunction')</td>
</tr>
<tr>
<td>maxIter</td>
<td>maximum number of iterations</td>
</tr>
<tr>
<td>tolerance</td>
<td>optimization is considered converged when the maximum relative change in fit is less than tolerance</td>
</tr>
<tr>
<td>verbose</td>
<td>level of debugging output</td>
</tr>
</tbody>
</table>

References


mxComputeNothing

*Compute nothing*

Description

Note that this compute plan actually does nothing whereas mxComputeOnce("expectation", "nothing") may remove the prediction of an expectation.

Usage

mxComputeNothing()

mxComputeNumericDeriv

*Numerically estimate Hessian using Richardson extrapolation*

Description

For N free parameters, Richardson extrapolation requires (iterations * (N^2 + N)) function evaluations.

Usage

mxComputeNumericDeriv(freeSet = NA_character_, ..., 
fitfunction = "fitfunction", parallel = TRUE, stepSize = 1e-04, 
iterations = 4L, verbose = 0L, knownHessian = NULL)
mxComputeOnce

Arguments

- **freeset**: names of matrices containing free variables
- **...**: Not used. Forces remaining arguments to be specified by name.
- **fitfunction**: name of the fitfunction (defaults to 'fitfunction')
- **parallel**: whether to evaluate the fitfunction in parallel (defaults to TRUE)
- **stepSize**: starting set size (defaults to 0.0001)
- **iterations**: number of Richardson extrapolation iterations (defaults to 4L)
- **verbose**: Level of debugging output.
- **knownHessian**: an optional matrix of known Hessian entries

Details

The implementation is closely based on the numDeriv R package.

Examples

```r
library(OpenMx)
data(demoOneFactor)
factorModel <- mxModel(name = "One Factor",
mxMatrix(type = "Full", nrow = 5, ncol = 1, free = FALSE, values = .2, name = "A"),
mxMatrix(type = "Symm", nrow = 1, ncol = 1, free = FALSE, values = 1, name = "L"),
mxMatrix(type = "Diag", nrow = 5, ncol = 5, free = TRUE, values = 1, name = "U"),
mxAlgebra(A %*% L %*% t(A) + U, name = "R"),
mxExpectationNormal(covariance = "R", dimnames = names(demoOneFactor)),
mxFitFunctionML(),
mxData(cov(demoOneFactor), type = "cov", numObs = 500),
mxComputeSequence(;
list(mxComputeNumericDeriv(), mxComputeReportDeriv())
)
)
factorModelFit <- mxRun(factorModel)
factorModelFit$output$hessian
```

**mxComputeOnce**

*Compute something once*

Description

Some models are optimized for a sparse Hessian. Therefore, it can be much more efficient to compute the inverse Hessian in comparison to computing the Hessian and then inverting it.

Usage

```r
mxComputeOnce(from, what = "nothing", how = NULL, ...,
freeSet = NA_character_, verbose = 0L, .is.bestfit = FALSE)
```
Arguments

from the object to perform the computation (a vector of expectation or fit function names)
what what to compute (default is "nothing")
how to compute it (optional)
... Not used. Forces remaining arguments to be specified by name.
freeSet names of matrices containing free variables
verbose the level of debugging output
.is.bestfit do not use; for backward compatibility

Details

The information matrix is only valid when parameters are at the maximum likelihood estimate. The information matrix is returned in model$output$hessian. You cannot request both the information matrix and the Hessian. The information matrix is invariant to the sign of the log likelihood scale whereas the Hessian is not. Use the how parameter to specify which approximation to use (one of "default", "hessian", "sandwich", "bread", and "meat").

Examples

data(demoOneFactor)
factorModel <- mxModel(name = "One Factor",
  mxMatrix(type="Full", nrow=5, ncol=1, free=TRUE, values=0.2, name="A"),
  mxMatrix(type="Symm", nrow=1, ncol=1, free=FALSE, values=1, name="L"),
  mxMatrix(type="Diag", nrow=5, ncol=5, free=TRUE, values=1, name="U"),
  mxAlgebra(expression=A %*% L %*% t(A) + U, name="R"),
  mxFitFunctionML(),mxExpectationNormal(covariance="R", dimnames=names(demoOneFactor)),
  mxData(observed=var(demoOneFactor), type="cov", numObs=500),
  mxComputeOnce('fitfunction', 'fit'))
factorModelFit <- mxRun(factorModel)
factorModelFit$output$fit # 972.15

mxComputeReportDeriv Report derivatives

Description

Copy the internal gradient and Hessian back to R.

Usage

mxComputeReportDeriv(freeSet = NA_character_)

Arguments

freeSet names of matrices containing free variables
**mxComputeSequence**

Invoke a series of compute objects in sequence

### Description

Invoke a series of compute objects in sequence

### Usage

```r
mxComputeSequence(steps = list(), ..., freeSet = NA_character_,
                   independent = FALSE)
```

### Arguments

- **steps**: a list of compute objects
- **...**: Not used; forces argument 'freeSet' to be specified by name.
- **freeSet**: Names of matrices containing free parameters.
- **independent**: Whether the steps could be executed out-of-order.

---

**mxComputeStandardError**

Compute standard errors given the Hessian or inverse Hessian

### Description

Compute standard errors given the Hessian or inverse Hessian

### Usage

```r
mxComputeStandardError(freeSet = NA_character_)
```

### Arguments

- **freeSet**: names of matrices containing free variables
Create MxConstraint Object

Description

This function creates a new MxConstraint object.

Usage

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

Arguments

expression  An R expression of matrix operators and matrix functions.
name        An optional character string indicating the name of the object.
...         Not used. Helps OpenMx catch bad input to argument ‘expression’.

Details

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

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

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

Value

Returns an MxConstraint object.
MxConstraint-class

References

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

See Also

MxConstraint for the S4 class created by mxConstraint.

Examples

library(OpenMx)

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

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

fit <- mxRun(model)
fit$matrices$K$values

# [ ,1] [ ,2]
# [ ,1]  1  3
# [ ,2]  2  4

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

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

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

---

MxConstraint-class    MxConstraint Class

Description

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

The MxConstraint class has the following slots:

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

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

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

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

References

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

See Also

mxConstraint for the function that creates MxConstraint objects.

mxData

Create MxData Object

Description

This function creates a new MxData object.

Usage

mxData(observed, type, means = NA, numObs = NA, acov=NA, fullWeight=NA, 
thresholds=NA, ..., sort=TRUE)

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”, or “cor”. |
| 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”. |
| acov           | Asymptotic covariance matrix of observed, means, and thresholds. Used for weighted least squares at weight matrix. |
fullWeight  Full asymptotic covariance matrix of observed, means, and thresholds. Used for weighted least squares in standard error and quasi-chi-squared calculation.

thresholds  Observed thresholds. Used for weighted least squares with ordinal data.

...  Not used. Forces remaining arguments to be specified by name.

sort  Whether to sort raw data prior to use (default TRUE)

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 expectation functions in `MxModel` objects. Four different data types are supported (a fifth, sscp, is not yet implemented):

- **raw** The contents of the ‘observed’ argument are treated as raw data. Missing values are permitted and must be designated as the system missing value. The ‘means’ and ‘numObs’ arguments cannot be specified, as the ‘means’ argument is not relevant and the ‘numObs’ argument is automatically populated with the number of rows in the data. Data of this type may use fit functions such as `mxFitFunctionML` function in `MxModel` objects, which will automatically use covariance estimation under full-information maximum likelihood for this data type.

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

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

- **acov** The contents of the ‘observed’ argument are treated as the polychoric correlation matrix of the ordinal variables. The ‘means’ argument is not required, but may be included for estimations involving means. The ‘thresholds’ argument is not required, but may be included for estimations involving thresholds and ordinal variables. The ‘numObs’ argument is required, which should reflect the number of observations or rows in the data described by the polychoric correlation matrix. Data of this type may use the fit functions such as `mxFitFunctionWLS` functions, depending on the specified model.

`MxData` objects may not be included in `MxAlgebra` objects or use the `mxFitFunctionAlgebra` 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 `mxFitModel` 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 or correlation 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 expectation 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 and correlation 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**

```r
library(OpenMx)

#Create a covariance matrix
covMatrix <- matrix( c(0.77642931, 0.39590663,
                        0.39590663, 0.49115615),
                       nrow = 2, ncol = 2, byrow = TRUE)
covNames <- c("x", "y")
dimList <- list(covNames, covNames)
dimnames(covMatrix) <- dimList

#Create an MxData object including that covariance matrix
testData <- mxData( observed=covMatrix, type="cov", numObs = 100)

testModel <- mxModel(model="testModel",
                      mxMatrix(type="Symm", nrow=2, ncol=2, values=c(0.2,1.2),
                               free=TRUE, name="expCov", dimnames=dimList),
                      mxExpectationNormal(covariance="expCov", dimnames=covNames),
                      mxFitFunctionML(),
                      testData)

outModel <- mxRun(testModel)

summary(outModel)
```
MxData-class

Description

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

Details

The MxNonNullData class has the following slots:

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

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

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

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

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

- **raw**: The contents of the 'observed' slot are treated as raw data. Missing values are permitted and must be designated as the system missing value. The 'vector' and 'numObs' slots cannot be specified, as the 'vector' argument is not relevant and the 'numObs' argument is automatically populated with the number of rows in the data. Data of this type may use the `mxFitFunctionML` function as its fit function in MxModel objects, which can deal 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 fit functions such as the `mxFitFunctionML`, 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 fit functions such as the `mxFitFunctionML`, 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' or 'cor', 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 mxFitFunctionAlgebra 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 or correlation 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.

mxDataDynamic Create dynamic data

Description

Create dynamic data

Usage

mxDataDynamic(type, ..., expectation, verbose = 0L)

Arguments

type type of data

... Not used. Forces remaining arguments to be specified by name.

expectation the name of the expectation to provide the data

verbose Increase runtime debugging output
MxDataFrameOrMatrix-class

MxDataFrameOrMatrix

Description

Internal class that is the union of data.frame and matrix.

Details

Not to be used.

MxDataStatic-class

Create static data

Description

Internal static data class.

Details

Not to be used.

mxDataWLS

Create MxData Object for Least Squares (WLS, DLS, ULS) Analyses

Description

This function creates a new MxData object of type “ULS” (unweighted least squares), “WLS” (weighted least squares) or “DLS” (diagonally-weighted least squares). The appropriate fit function to include with these models is mxFitFunctionWLS

Usage

mxDataWLS(data, type = "WLS", useMinusTwo = TRUE, returnInverted = TRUE, debug = FALSE, fullWeight = TRUE)
Arguments

data    A matrix or data.frame which provides raw data to be used for WLS.
type    A character string 'WLS' (default), 'DLS', or 'ULS' for weighted, diagonal, or unweighted least squares
useMinusTwo Logical indicating whether to use -2LL (default) or -LL.
returnInverted Logical indicating whether to return the information matrix (default) or the co-variance matrix.
debug   Logical to set debugging on or off (default)
fullWeight Logical determining if the full weight matrix is returned (default). Needed for standard error and quasi-chi-squared calculation.

Details

The mxDataWLS function creates an MxData object, which can be used in MxModel objects. This function takes raw data and returns an MxData object to be used in a model to fit with weighted least squares.

Both Ordinal and continuous data are supported. A combination of these data types succeeds without error, but when using 'WLS' or 'DLS' the answers appear incorrect. The 'ULS' estimates for joint ordinal and continuous data appear accurate. Consequently, do not use this function for joint problems unless type='ULS'.

Value

Returns a new MxData object.

References

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

See Also

mxFitFunctionWLS, 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 and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionWLS
library(OpenMx)

# Simulate some data
x=rnorm(1000, mean=0, sd=1)
y= 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)
```r
wdata <- mxDataWLS(tmpFrame)

# Define the matrices

S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values = c(1, 0, 0, 1),
               free = c(TRUE, FALSE, FALSE, TRUE), labels = c("Vx", NA, NA, "Vy"), name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values = c(0, 1, 0, 0),
               free = c(FALSE, TRUE, FALSE, FALSE), labels = c(NA, "b", NA, NA), name = "A")
I <- mxMatrix(type = "Iden", nrow = 2, ncol = 2, name = "I")

# Define the expectation

expCov <- mxAlgebra(solve(I - A) %*% S %*% t(solve(I - A)), name = "expCov")
expFunction <- mxExpectationNormal(covariance = "expCov", dimnames = tmpNames)

# Choose a fit function

fitFunction <- mxFitFunctionWLS()

# Define the model

tmpModel <- mxModel(model = "exampleModel", S, A, I, expCov, expFunction, fitFunction,
                    wdata)

# Fit the model and print a summary

tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)
```

---

**MxDirectedGraph-class**  
*MxDirectedGraph*

**Description**

This is an internal class and should not be used directly. It is a class for directed graphs.

---

**mxEval**  
*Evaluate Values in MxModel*

**Description**

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

mxEval(expression, model, compute = FALSE, show = FALSE, defvar.row = 1, cache = new.env(parent = emptyenv()), cacheBack = FALSE)

mxEvalByName(name, model, compute = FALSE, show = FALSE, defvar.row = 1, cache = new.env(parent = emptyenv()), cacheBack = 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.
defvar.row The row number for definition variables when compute=TRUE.
cache An R environment of matrix values used to speedup computation.
cacheBack If TRUE then return the list pair (value, cache).
name The character name of an object to evaluate.

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).

The mxEvalByName function is a wrapper around mxEval that takes a character instead of an R expression.

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

The ‘cache’ is used to speedup calculation by storing previously computing values. The cache is a list of matrices, such that names(cache) must all be of the form “modelname.entityname”. Setting ‘cacheBack’ to TRUE will return the pair list(value, cache) where value is the result of the mxEval() computation and cache is the updated cache.

References

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

See Also

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

library(OpenMx)

# Set up a 1x1 matrix
matrixA <- mxMatrix("Full", nrow = 1, ncol = 1, values = 1, name = "A")

# Set up an algebra
algebraB <- mxAlgebra(A + A, name = "B")

# Put them both in a model
testModel <- mxModel(model="testModel", matrixA, algebraB)

# Even though the model has not been run, we can evaluate the algebra
# given the starting values in matrixA.
mxEval(B, testModel, compute=TRUE)

# If we just print the algebra, we can see it has not been evaluated
testModel$B

MxExpectation-class  MxExpectation

Description

This is an internal class and should not be used directly.

mxExpectationBA81  Create a Bock & Aitkin (1981) expectation

Description

When a two-tier covariance matrix is recognized, this expectation automatically enables analytic
dimension reduction (Cai, 2010).

Usage

mxExpectationBA81(ItemSpec, item = "item", ..., qpoints = 49L, qwidth = 6,
mean = "mean", cov = "cov", verbose = 0L, weightColumn = NA_integer_,
EstepItem = NULL, debugInternal = FALSE)
Arguments

ItemSpec  a single item model (to replicate) or a list of item models in the same order as the column of ItemParam

item  the name of the mxMatrix holding item parameters with one column for each item model with parameters starting at row 1 and extra rows filled with NA

...  Not used. Forces remaining arguments to be specified by name.

qpoints  number of points to use for equal interval quadrature integration (default 49L)

qwidth  the width of the quadrature as a positive Z score (default 6.0)

mean  the name of the mxMatrix holding the mean vector

cov  the name of the mxMatrix holding the covariance matrix

verbose  the level of runtime diagnostics (default 0L)

weightColumn  the name of the column in the data containing the row weights (default NA)

EstepItem  a simple matrix of item parameters for the E-step. This option is mainly of use for debugging derivatives.

debugInternal  when enabled, some of the internal tables are returned in $debug. This is mainly of use to developers.

Details

The standard Normal distribution of the quadrature acts like a prior distribution for difficulty. It is not necessary to impose any additional Bayesian prior on difficulty estimates (Baker & Kim, 2004, p. 196).

References


See Also

RPF
mxExpectationGREML  

Create MxExpectationGREML Object

Description

This function creates a new MxExpectationGREML object.

Usage

mxExpectationGREML(V, yvars=character(0), xvars=list(), addOnes=TRUE, blockByPheno=TRUE, staggerZeroes=TRUE, dataset.is.yX=FALSE, casesToRemoveFromV=integer(0))

Arguments

V  Character string: the name of the MxAlgebra or MxMatrix to serve as the ’V’ matrix (the model-expected covariance matrix).
yvars, xvars, addOnes, blockByPheno, staggerZeroes  Passed to mxGremldataHandler().
dataset.is.yX  Logical; defaults to FALSE. If TRUE, then the first column of the raw dataset is taken as-is to be the ’y’ phenotype vector, and the remaining columns are taken as-is to be the ’X’ matrix of covariates. In this case, mxGremldataHandler() is never internally called at runtime, and all other arguments besides V and casesToRemoveFromV are ignored.
casesToRemoveFromV  Integer vector. Its elements are the numbers of the rows and columns of covariance matrix ’V’ to be dropped at runtime, usually because they correspond to rows of ’y’ or ’X’ that contained missing observations. By default, no cases are dropped from ’V.’ Ignored unless dataset.is.yX=TRUE.

Details

"GREML" stands for "genomic-relatedness-matrix restricted maximum-likelihood." In the strictest sense of the term, it refers to genetic variance-component estimation from matrices of subjects’ pairwise degree of genetic relatedness, as calculated from genome-wide marker data. It is from this original motivation that some of the terminology originates, such as calling ’y’ the "phenotype" vector. However, OpenMx’s implementation of GREML is applicable for analyses from any subject-matter domain, and in which the following assumptions are reasonable:

1. Conditional on ’X’ (the covariates), the phenotype vector (response variable) ’y’ is a single realization from a multivariate-normal distribution having (in general) a dense covariance matrix, ’V’.
2. The parameters of the covariance matrix, such as variance components, are of primary interest.
3. The random effects are normally distributed.
4. Weighted least-squares regression, using the inverse of ’V’ as a weight matrix, is an adequate model for the phenotypic means. Note that the regression coefficients are not actually free parameters to be numerically optimized.
Computationally, the chief distinguishing feature of an OpenMx GREML analysis is that the phenotype vector, 'y,' is a single realization of a random vector that, in general, cannot be partitioned into independent subvectors. For this reason, definition variables are not compatible (and should be unnecessary with) GREML expectation. GREML expectation can still be used if the covariance matrix is sparse, but as of this writing, OpenMx does not take advantage of the sparseness to improved performance. Because of the limitations of restricted maximum likelihood, GREML expectation is presently incompatible with ordinal variables.

Value

Returns a new object of class `MxExpectationGREML`.

References

One of the first uses of the acronym "GREML":

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

See Also

See `MxExpectationGREML` for the S4 class created by `mxExpectationGREML()`. More information about the OpenMx package may be found here.

Examples

dat <- cbind(rnorm(100),rep(1,100))
colnames(dat) <- c("y","x")
ge <- mxExpectationGREML(V="y",yvars="y",Xvars=list("X"),addOnes=FALSE)
gff <- mxFitFunctionGREML(dv=c(ve="I"))
plan <- mxComputeSequence(steps=list(  mxComputeNewtonRaphson(freeSet=c("Ve"),fitfunction="fitfunction"),  mxComputeOnce('fitfunction',  c("fit","gradient","hessian","ihessian"),freeSet=c("Ve")),  mxComputeStandardError(freeSet=c("Ve")),  mxComputeReportDeriv(freeSet=c("Ve"))
))
testmod <- mxModel("GREMLtest",  mxData(observed = dat, type="raw"),  mxMatrix(type = "Full", nrow = 1, ncol=1, free=TRUE,  values = 1, labels = "ve", lboun = 0.0001, name = "Ve"),  mxMatrix("Iden",nrow=100,name="I",condenseSlots=TRUE),  mxAlgebra(1 %x% Ve, names="V"),  ge,  gff,  plan )
MxExpectationGREML-class

MxExpectationGREML-class

Class "MxExpectationGREML"

Description

MxExpectationGREML is a type of expectation class. It contains the necessary elements for specifying a GREML model. For more information, see mxExpectationGREML().

Objects from the Class

Objects can be created by calls of the form mxExpectationGREML(V, yvars, xvars, addOnes, blockByPheno,

Slots

V: Object of class "MxCharOrNumber". Identifies the MxAlgebra or MxMatrix to serve as the 'V' matrix.
yvars: Character vector. Each string names a column of the raw dataset, to be used as a phenotypes.
xvars: A list of data column names, specifying the covariates to be used with each phenotype.
addOnes: Logical; pertains to data-handling at runtime.
blockByPheno: Logical; pertains to data-handling at runtime.
staggerZeroes: Logical; pertains to data-handling at runtime.
dataset.is.yX: Logical; pertains to data-handling at runtime.
y: Object of class "MxData". Its observed slot will contain the phenotype vector, 'y.'
X: A matrix, to contain the 'X' matrix of covariates.
yXcolnames: Character vector; used to store the column names of 'y' and 'X.'
casesToDrop: Integer vector, specifying the rows and columns of the 'V' matrix to be removed at runtime.
b: A matrix, to contain the vector of regression coefficients calculated at runtime.
bCov: A matrix, to contain the sampling covariance matrix of the regression coefficients calculated at runtime.
numFixEff: Integer number of covariates in 'X.'
dims: Object of class "character".
definitionVars: Object of class "list".
numStats: Numeric; number of observed statistics.
dataColumns: Object of class "numeric".
name: Object of class "character".
data: Object of class "MxCharOrNumber".
subModels: Object of class "MxOptionalCharOrNumber".
container: Object of class "MxOptionalCharOrNumber".
.runDims: Object of class "character".
mxExpectationLISREL

Extends
Class "MxBaseExpectation", directly. Class "MxBaseNamed", by class "MxBaseExpectation", distance 2. Class "MxExpectation", by class "MxBaseExpectation", distance 2.

Methods
No methods defined with class "MxExpectationGREML" in the signature.

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

See Also
See mxExpectationGREML() for creating MxExpectationGREML objects, and for more information generally concerning GREML analyses, including a complete example. More information about the OpenMx package may be found here.

Examples
showClass("MxExpectationGREML")

mxExpectationLISREL Create MxExpectationLISREL Object

Description
This function creates a new MxExpectationLISREL object.

Usage
mxExpectationLISREL(LX=NA, LY=NA, BE=NA, GA=NA, PH=NA, PS=NA, TD=NA, TE=NA, TH=NA, TX = NA, TY = NA, KA = NA, AL = NA, 
     dimnames = NA, thresholds = NA, threshnames = dimnames)

Arguments
LX An optional character string indicating the name of the 'LX' matrix.
LY An optional character string indicating the name of the 'LY' matrix.
BE An optional character string indicating the name of the 'BE' matrix.
GA An optional character string indicating the name of the 'GA' matrix.
PH An optional character string indicating the name of the 'PH' matrix.
PS An optional character string indicating the name of the 'PS' matrix.
TD An optional character string indicating the name of the 'TD' matrix.
TE An optional character string indicating the name of the 'TE' matrix.
An optional character string indicating the name of the 'TH' matrix.

An optional character string indicating the name of the 'TX' matrix.

An optional character string indicating the name of the 'TY' matrix.

An optional character string indicating the name of the 'KA' matrix.

An optional character string indicating the name of the 'AL' matrix.

An optional character vector that is currently ignored.

An optional character string indicating the name of the thresholds matrix.

An optional character vector to be assigned to the column names of the thresholds matrix.

Details

Expectation functions define the way that model expectations are calculated. The mxExpectationLISREL calculates the expected covariance and means of a given MxData object given a LISREL model. This model is defined by Linear Structural RELations (LISREL; Jöreskog & Sörbom, 1982, 1996). Arguments 'LX' through 'AL' must refer to MxMatrix objects with the associated properties of their respective matrices in the LISREL modeling approach.

The full LISREL specification has 13 matrices and is sometimes called the extended LISREL model. It is defined by the following equations.

$$\eta = \alpha + B\eta + \Gamma\xi + \zeta$$
$$y = \tau_y + \Lambda_y\eta + \epsilon$$
$$x = \tau_x + \Lambda_x\xi + \delta$$

The table below is provided as a quick reference to the numerous matrices in LISREL models. Note that NX is the number of manifest exogenous (independent) variables, the number of Xs. NY is the number of manifest endogenous (dependent) variables, the number of Ys. NK is the number of latent exogenous variables, the number of Ksis or Xis. NE is the number of latent endogenous variables, the number of etas.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Word</th>
<th>Abbreviation</th>
<th>Dimensions</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Lambda_x)</td>
<td>Lambda x</td>
<td>LX</td>
<td>NX x NK</td>
<td></td>
<td>Exogenous Factor Loading Matrix</td>
</tr>
<tr>
<td>(\Lambda_y)</td>
<td>Lambda y</td>
<td>LY</td>
<td>NY x NE</td>
<td></td>
<td>Endogenous Factor Loading Matrix</td>
</tr>
<tr>
<td>(B)</td>
<td>Beta</td>
<td>BE</td>
<td>NE x NE</td>
<td></td>
<td>Regressions of Latent Endogenous Variables</td>
</tr>
<tr>
<td>(\Gamma)</td>
<td>Gamma</td>
<td>GA</td>
<td>NE x NK</td>
<td></td>
<td>Regressions of Latent Exogenous Variables</td>
</tr>
<tr>
<td>(\Phi)</td>
<td>Phi</td>
<td>PH</td>
<td>NK x NK</td>
<td>(\text{cov}(\xi))</td>
<td>Covariance Matrix of Latent Exogenous Variables</td>
</tr>
<tr>
<td>(\Psi)</td>
<td>Psi</td>
<td>PS</td>
<td>NE x NE</td>
<td>(\text{cov}(\zeta))</td>
<td>Residual Covariance Matrix of Latent Endogenous Variables</td>
</tr>
<tr>
<td>(\Theta_\delta)</td>
<td>Theta delta</td>
<td>TD</td>
<td>NX x NX</td>
<td>(\text{cov}(\delta))</td>
<td>Residual Covariance Matrix of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>(\Theta_\epsilon)</td>
<td>Theta epsilon</td>
<td>TE</td>
<td>NY x NY</td>
<td>(\text{cov}(\epsilon))</td>
<td>Residual Covariance Matrix of Manifest Endogenous Variables</td>
</tr>
<tr>
<td>(\Theta_{\delta\epsilon})</td>
<td>Theta delta epsilon</td>
<td>TH</td>
<td>NX x NY</td>
<td>(\text{cov}(\delta, \epsilon))</td>
<td>Residual Covariance Matrix of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>(\tau_x)</td>
<td>tau x</td>
<td>TX</td>
<td>NX x 1</td>
<td></td>
<td>Residual Means of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>(\tau_y)</td>
<td>tau y</td>
<td>TY</td>
<td>NY x 1</td>
<td></td>
<td>Residual Means of Manifest Endogenous Variables</td>
</tr>
<tr>
<td>(\kappa)</td>
<td>kappa</td>
<td>KA</td>
<td>NK x 1</td>
<td>(\text{mean}(\xi))</td>
<td>Means of Latent Exogenous Variables</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>alpha</td>
<td>AL</td>
<td>NE x 1</td>
<td></td>
<td>Residual Means of Latent Endogenous Variables</td>
</tr>
</tbody>
</table>
From the extended LISREL model, several submodels can be defined. Subtypes of the LISREL model are defined by setting some of the arguments of the LISREL expectation function to NA. Note that because the default values of each LISREL matrix is NA, setting a matrix to NA can be accomplished by simply not giving it any other value.

The first submodel is the LISREL model without means.

\[
\eta = B\eta + \Gamma\xi + \zeta \\
y = \Lambda_y\eta + \epsilon \\
x = \Lambda_x\xi + \delta
\]

The LISREL model without means requires 9 matrices: LX, LY, BE, GA, PH, PS, TD, TE, and TH. Hence this LISREL model has TX, TY, KA, and AL as NA. This can be accomplished by leaving these matrices at their default values.

The TX, TY, KA, and AL matrices 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 TX, TY, KA, and AL matrices are ignored and the model without means is estimated.

A second submodel involves only endogenous variables.

\[
\eta = B\eta + \zeta \\
y = \Lambda_y\eta + \epsilon
\]

The endogenous-only LISREL model requires 4 matrices: LY, BE, PS, and TE. The LX, GA, PH, TD, and TH must be NA in this case. However, means can also be specified, allowing TY and AL if the data are raw or if observed means are provided.

Another submodel involves only exogenous variables.

\[
x = \Lambda_x\xi + \delta
\]

The exogenous-model model requires 3 matrices: LX, PH, and TD. The LY, BE, GA, PS, TE, and TH matrices must be NA. However, means can also be specified, allowing TX and KA if the data are raw or if observed means are provided.

The model that is run depends on the matrices that are not NA. If all 9 matrices are not NA, then the full model is run. If only the 4 endogenous matrices are not NA, then the endogenous-only model is run. If only the 3 exogenous matrices are not NA, then the exogenous-only model is run. If some endogenous and exogenous matrices are not NA, but not all of them, then appropriate errors are thrown. Means are included in the model whenever their matrices are provided.

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

Like the mxExpectationRAM, the mxExpectationLISREL evaluates with respect to an MxData object. The MxData object need not be referenced in the mxExpectationLISREL function, but must be included in the MxModel object. mxExpectationLISREL requires that the ’type’ argument in the associated MxData object be equal to ’cov’, ’cor’, or ’raw’.
mxExpectationLISREL

To evaluate, place `mxExpectationLISREL` 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 `MxExpectationLISREL` object. One and only one `MxExpectationLISREL` object can be included with models using one and only one fit function object (e.g., `MxFitFunctionML`) and with referenced `MxAlgebra`, `MxData` and `MxMatrix` objects.

References


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

See Also

demo("LISRELJointFactorModel")

Examples

```R
library(OpenMx)

vNames <- paste("v",as.character(1:6),sep="")
dimList <- list(vNames, vNames)
covData <- matrix(c(0.9223899, 0.1862938, 0.4374359, 0.8959973, 0.9928430, 0.5320662,
                    0.1862938, 0.2889364, 0.3927790, 0.3321639, 0.3371594, 0.4476898,
                    0.4374359, 0.3927790, 1.0069552, 0.6918755, 0.7482155, 0.9013952,
                    0.8959973, 0.3321639, 0.6918755, 1.8059956, 1.6142005, 0.8040448,
                    0.9928430, 0.3371594, 0.7482155, 1.6142005, 1.9223567, 0.8777786,
                    0.5320662, 0.4476898, 0.3913952, 0.8040448, 0.8777786, 1.3997558),
nrow=6, ncol=6, byrow=TRUE, dimnames=dimList)

mLX <- mxMatrix("Full", values=c(.5, .6, .8, rep(0, 6), .4, .7, .5),
                     name="LX", nrow=6, ncol=2,
                     free=c(TRUE,TRUE,TRUE,rep(FALSE, 6),TRUE,TRUE),
                     dimnames=list(vNames, c("x1","x2")))

mTD <- mxMatrix("Diag", values=c(rep(.2, 6)),
                      name="TD", nrow=6, ncol=6, free=TRUE,
                      dimnames=dimList)
```

```
mPH <- mxMatrix("Symm", values=c(1, .3, 1),
    name="PH", nrow=2, ncol=2, free=c(FALSE, TRUE, FALSE),
    dimnames=list(c("x1","x2"), c("x1","x2")))

# Create a LISREL expectation with LX, TD, and PH matrix names
expFunction <- mxExpectationLISREL(LX="LX", TD="TD", PH="PH")

# Create fit function and data
tmpData <- mxData( observed=observed, type="cov", numObs=100)
fitFunction <- mxFitFunctionML()

# Create the model, fit it, and print a summary.
tmpModel <- mxModel(model="exampleModel",
    LX, TD, PH, expFunction, fitFunction, tmpData)
tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

#--------------------------------------------------------------
# Fit factor model with means
require(OpenMx)
data(demoOneFactor)
nvar <- ncol(demoOneFactor)
varnames <- colnames(demoOneFactor)

factorMeans <- mxMatrix("Zero", 1, 1, name="Kappa",
    dimnames=list("F1", NA))

xIntercepts <- mxMatrix("Full", nvar, 1, free=TRUE, name="TauX",
    dimnames=list(varnames, NA))

factorLoadings <- mxMatrix("Full", nvar, 1, TRUE, .6, name="LambdaX",
    labels=paste("lambda", 1:nvar, sep=""),
    dimnames=list(varnames, "F1"))

factorCovariance <- mxMatrix("Diag", 1, 1, FALSE, 1, name="Phi")

xResidualVariance <- mxMatrix("Diag", nvar, nvar, TRUE, .2, name="ThetaDelta",
    labels=paste("theta", 1:nvar, sep=""))

liModel <- mxModel(model="LISREL Factor Model",
    factorMeans, xIntercepts, factorLoadings, factorCovariance, xResidualVariance,
    mxExpectationLISREL(LX="LambdaX", PH="Phi",
    TD="ThetaDelta", TX="TauX", KA="Kappa"),
    mxFitFunctionML(),
    mxData(cov(demoOneFactor), "cov",
    means=colMeans(demoOneFactor), numObs=nrow(demoOneFactor))
  )

liRun <- mxRun(liModel)
```
**mxExpectationNormal**

Create **MxExpectationNormal** Object

**Description**

This function creates an MxExpectationNormal object.

**Usage**

```r
mxExpectationNormal(covariance, means, dimnames = NA, thresholds = NA, threshnames = dimnames)
```

**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.
- `threshnames`: An optional character vector to be assigned to the column names of the thresholds matrix.

**Details**

Expectation functions define the way that model expectations are calculated. The `mxExpectationNormal` function uses the algebra defined by the 'covariance' and 'means' arguments to define the expected covariance and means under the assumption of multivariate normality. 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 is used to assign the dimnames of the means vector as well as the row and columns dimnames of the covariance matrix.

- `thresholds`: The name of the thresholds matrix. When needed (for modelling ordinal data), this matrix should be created using `mxMatrix()`. The thresholds matrix must have as many columns as there are ordinal variables in the model, and number of rows equal to one fewer than the maximum number of levels found in the ordinal variables. The starting values of this matrix must also be set to reasonable values. Fill each column with a set of ordered start thresholds, one for each level of this column’s factor levels minus 1. These thresholds may be free if you wish them to be estimated, or fixed. The unused rows in each column, if any, can be set to any value including NA.

- `threshnames`: A character vector consisting of the variables in the thresholds matrix, i.e., the names of ordinal variables in a model. This is necessary for OpenMx to map the thresholds matrix columns onto the variables in your data. If you set the `dimnames` of the columns in the thresholds matrix then `threshnames` is not needed.
Usage Notes: \texttt{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 \texttt{dimnames}.

\texttt{mxExpectationNormal} evaluates with respect to an \texttt{MxData} object. The \texttt{MxData} object need not be referenced in the \texttt{mxExpectationNormal} function, but must be included in the \texttt{MxModel} object. When the 'type' argument in the associated \texttt{MxData} object is equal to 'raw', missing values are permitted in the associated \texttt{MxData} object.

To evaluate, place an \texttt{mxExpectationNormal} object, the \texttt{mxData} object for which the expected covariance approximates, referenced \texttt{MxAlgebra} and \texttt{MxMatrix} objects, optional \texttt{MxBounds} or \texttt{MxConstraint} objects, and an \texttt{mxFitFunction} such as \texttt{mxFitFunctionML} in an \texttt{MxModel} object. This model may then be evaluated using the \texttt{mxRun} function.

The results of the optimization can be reported using the \texttt{summary} function, or accessed directly in the 'output' slot of the resulting model (i.e., \texttt{modelName$output}). Components of the output may be referenced using the \texttt{Extract} functionality.

\textbf{Value}

Returns an \texttt{MxExpectationNormal} object.

\textbf{References}

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

\textbf{Examples}

\begin{verbatim}
# Create and fit a model using \texttt{mxMatrix}, \texttt{mxAlgebra},
# \texttt{mxExpectationNormal}, and \texttt{mxFitFunctionML}

library(OpenMx)

# Simulate some data
x=rnorm(1000, mean=0, sd=1)
y= 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices

M <- mxMatrix(type = "Full", nrow = 1, ncol = 2, values=c(0,0),
              free=c(TRUE,TRUE), labels=c("Mx", "My"), name = "M")
S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1,0,0,1),
              free=c(TRUE,FALSE,TRUE,TRUE), labels=c("Vx", NA, NA, "Vy"),
              name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,1,0,0),
              free=c(FALSE,TRUE,FALSE,FALSE), labels=c(NA, "b", NA, NA),
              name = "A")
I <- mxMatrix(type="Iden", nrow=2, ncol=2, name="I")
\end{verbatim}
# Define the expectation

```r
expCov <- mxAlgebra(solve(I-A) %*% S %*% t(solve(I-A)), name="expCov")
expFunction <- mxExpectationNormal(covariance="expCov", means="M", dimnames=tmpNames)
```

# Choose a fit function

```r
fitFunction <- mxFitFunctionML()
```

# Define the model

```r
tmpModel <- mxModel(model="exampleModel", M, S, A, I,
                     expCov, expFunction, fitFunction,
                     mxData(observed=tmpFrame, type="raw"))
```

# Fit the model and print a summary

```r
tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)
```

---

### mxExpectationRAM

Create an MxExpectationRAM Object

**Description**

This function creates an MxExpectationRAM object.

**Usage**

```r
mxExpectationRAM(A="A", S="S", F="F", M = NA, dimnames = NA, thresholds = NA,
                 threshnames = dimnames)
```

**Arguments**

- **A**: A character string indicating the name of the 'A' matrix.
- **S**: A character string indicating the name of the 'S' matrix.
- **F**: A character string indicating the name of the 'F' matrix.
- **M**: An optional character string indicating the name of the 'M' matrix.
- **dimnames**: An optional character vector to be assigned to the column names of the 'F' and 'M' matrices.
- **thresholds**: An optional character string indicating the name of the thresholds matrix.
- **threshnames**: An optional character vector to be assigned to the column names of the thresholds matrix.
mxExpectationRAM

Details

Expectation functions define the way that model expectations are calculated. The mxExpectationRAM calculates the expected covariance and means of a given MxData object given a RAM model. 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 mxExpectationRAM will not return an error for incorrect specification, but incorrect specification will likely lead to estimation problems or errors in the mxRun function.

mxExpectationRAM evaluates with respect to an MxData object. The MxData object need not be referenced in the mxExpectationRAM function, but must be included in the MxModel object.

To evaluate, place mxExpectationRAM 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 MxExpectationRAM object. mxExpectationRAM 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
# Create and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionML
library(OpenMx)

# Simulate some data
x <- rnorm(1000, mean=0, sd=1)
y <- 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices
matrixS <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1,0,0,1),
                   free=c(TRUE,FALSE,FALSE,TRUE), labels=c("Vx", NA, NA, "Vy"),
                   name = "S")
matrixA <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,1,0,0),
                   free=c(FALSE,TRUE,FALSE,FALSE), labels=c(NA, "b", NA, NA),
                   name = "A")
matrixF <- mxMatrix(type="Iden", nrow=2, ncol=2, name="F")
matrixM <- mxMatrix(type = "Full", nrow = 1, ncol = 2, values=c(0,0),
                   free=c(TRUE,TRUE), labels=c("Mx", "My"), name = "M")

# Define the expectation
expFunction <- mxExpectationRAM(M="M", dimnames = tmpNames)

# Choose a fit function
fitFunction <- mxFitFunctionML()

# Define the model
tmpModel <- mxModel(model="exampleRAMModel",
                    matrixA, matrixS, matrixF, matrixM,
                    expFunction, fitFunction,
                    mxData( observed=tmpFrame, type="raw")

# Fit the model and print a summary
tmpModelOut <- mxRun(tmpModel)
```
mxExpectationStateSpace

Create an MxExpectationStateSpace Object

Description

This function creates a new MxExpectationStateSpace object.

Usage

mxExpectationStateSpace(A, B, C, D, Q, R, x0, P0, u,
                           dimnames = NA, thresholds = NA, threshnames = dimnames,
                           ..., t = NA)

Arguments

A       A character string indicating the name of the 'A' matrix.
B       A character string indicating the name of the 'B' matrix.
C       A character string indicating the name of the 'C' matrix.
D       A character string indicating the name of the 'D' matrix.
Q       A character string indicating the name of the 'Q' matrix.
R       A character string indicating the name of the 'R' matrix.
x0      A character string indicating the name of the 'x0' matrix.
P0      A character string indicating the name of the 'P0' matrix.
u       A character string indicating the name of the 'u' matrix.
dimnames An optional character vector to be assigned to the row names of the 'C' matrix.
thresholds Not Yet Implemented. An optional character string indicating the name of the thresholds matrix.
threshnames Not Yet Implemented. An optional character vector to be assigned to the column names of the thresholds matrix.
...     Unused. Requires further arguments to be named.
t       Not to be used.

Details

Expectation functions define the way that model expectations are calculated. When used in conjunction with the mxFitFunctionML, the mxExpectationStateSpace uses maximum likelihood prediction error decomposition (PED) to obtain estimates of free parameters in a model of the raw MxData object. State space expectations treat the raw data as a multivariate time series of equally spaced times with each row corresponding to a single occasion. This is not a model of the block Toeplitz lagged autocovariance matrix. State space expectations implement a classical Kalman filter to produce expectations.
The hybrid Kalman filter (combination of classical Kalman and Kalman-Bucy filters) for continuous latent time with discrete observations is implemented and is available as `mxExpectationStateSpace-ContinuousTime`. The following alternative filters are not yet implemented: square root Kalman filter (in Cholesky or singular value decomposition form), extended Kalman filter for linear approximations to nonlinear state space models, unscented Kalman filter for highly nonlinear state space models, and Rauch-Tung-Striebel smoother for updating forecast state estimates after a complete forward pass through the data has been made.

Missing data handling is implemented in the same fashion as full information maximum likelihood for partially missing rows of data. Additionally, completely missing rows of data are handled by only using the prediction step from the Kalman filter and omitting the update step.

This model uses notation for the model matrices commonly found in engineering and control theory. The 'A', 'B', 'C', 'D', 'Q', 'R', 'x0', and 'P0' arguments must be the names of `MxMatrix` or `MxAlgebra` objects with the associated properties of the A, B, C, D, Q, R, x0, and P0 matrices in the state space modeling approach.

The state space expectation is defined by the following model equations.

\[
x_t = Ax_{t-1} + Bu_t + q_t \\
y_t = Cx_t + Du_t + r_t
\]

with \(q_t\) and \(r_t\) both independently and identically distributed random Gaussian (normal) variables with mean zero and covariance matrices \(Q\) and \(R\), respectively.

The first equation is called the state equation. It describes how the latent states change over time. Also, the state equation in state space modeling is directly analogous to the structural model in LISREL structural equation modeling.

The second equation is called the output equation. It describes how the latent states relate to the observed states at a single point in time. The output equation shows how the observed output is produced by the latent states. Also, the output equation in state space modeling is directly analogous to the measurement model in LISREL structural equation modeling.

Note that the covariates, \(u_t\), have "instantaneous" effects on both the state and output equations. If lagged effects are desired, then the user must create a lagged covariate by shifting their observed variable to the desired lag.

The state and output equations, together with some minimal assumptions and the Kalman filter, imply a new expected covariance matrix and means vector for every row of data. The expected covariance matrix of row \(t\) is

\[
S_t = C(AP_{t-1}A^T + Q)C^T + R
\]

The expected means vector of row \(t\) is

\[
\hat{y}_t = Cx_t + Du_t
\]

The 'dimnames' arguments takes an optional character vector.

The 'A' argument refers to the \(A\) matrix in the State Space approach. This matrix consists of time regressive coefficients from the latent variable in column \(j\) at time \(t - 1\) to the latent variable in row
$i$ at time $t$. Entries in the diagonal are autoregressive coefficients. Entries in the off-diagonal are cross-lagged regressive coefficients. If the $A$ and $B$ matrices are zero matrices, then the state space model reduces to a factor analysis. The $A$ matrix is sometimes called the state-transition model.

The 'B' argument refers to the $B$ matrix in the State Space approach. This matrix consists of regressive coefficients from the input (manifest covariate) variable $j$ at time $t$ to the latent variable in row $i$ at time $t$. Note that the covariate effect is contemporaneous: the covariate at time $t$ has influence on the latent state also at time $t$. A lagged effect can be created by lagged the observed variable. The $B$ matrix is sometimes called the control-input model.

The 'C' argument refers to the $C$ matrix in the State Space approach. This matrix consists of contemporaneous regression coefficients from the latent variable in column $j$ to the observed variable in row $i$. This matrix is directly analogous to the factor loadings matrix in LISREL and Mplus models. The $C$ matrix is sometimes called the observation model.

The 'D' argument refers to the $D$ matrix in the State Space approach. This matrix consists of contemporaneous regressive coefficients from the input (manifest covariate) variable $j$ to the observed variable in row $i$. The $D$ matrix is sometimes called the feedthrough or feedforward matrix.

The 'Q' argument refers to the $Q$ matrix in the State Space approach. This matrix consists of residual covariances among the latent variables. This matrix must be symmetric. As a special case, it is often diagonal. The $Q$ matrix is the covariance of the process noise. Just as in factor analysis and general structural equation modeling, the scale of the latent variables is usually set by fixing some factor loadings in the $C$ matrix, or fixing some factor variances in the $Q$ matrix.

The 'R' argument refers to the $R$ matrix in the State Space approach. This matrix consists of residual covariances among the observed (manifest) variables. This matrix must be symmetric. As a special case, it is often diagonal. The $R$ matrix is the covariance of the observation noise.

The 'x0' argument refers to the $x_0$ matrix in the State Space approach. This matrix consists of the column vector of the initial values for the latent variables. The state space expectation uses the $x_0$ matrix as the starting point to recursively estimate the latent variables’ values at each time. These starting values can be difficult to pick, however, for sufficiently long time series they often do not greatly impact the estimation.

The 'P0' argument refers to the $P_0$ matrix in the State Space approach. This matrix consists of the initial values of the covariances of the error in the initial latent variable estimates given in $x_0$. That is, the $P_0$ matrix gives the covariance of $x_0 - x_{true_0}$ where $x_{true_0}$ is the vector of true initial values. $P_0$ is a measure of the accuracy of the initial latent state estimates. The Kalman filter uses this initial covariance to recursively generated a new covariance for each time point based on the previous time point. The Kalman filter updates this covariance so that it is as small as possible (minimum trace). Similar to the $x_0$ matrix, these starting values are often difficult to choose.

The 'u' argument refers to the $u$ matrix in the State Space approach. This matrix consists of the inputs or manifest covariates of the state space expectation. The $u$ matrix must be a column vector with the same number of rows as the $B$ and $D$ matrices have columns. If no inputs are desired, $u$ can be a zero matrix. If time-varying inputs are desired, then they should be included as columns in the MxData object and referred to in the labels of the $u$ matrix as definition variables. There is an example of this below.

The MxMatrix objects included as arguments may be of any type, but should have the properties described above. The mxExpectationStateSpace will not return an error for incorrect specification, but incorrect specification will likely lead to estimation problems or errors in the mxRun function.
object. `mxExpectationStateSpace` requires that the 'type' argument in the associated `MxData` object be equal to 'raw'. Neighboring rows of the `MxData` object are treated as adjacent, equidistant time points increasing from the first to the last row.

To evaluate, place `mxExpectationStateSpace` 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 `MxExpectationStateSpace` object. `mxExpectationStateSpace` 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.

See Also

`mxExpectationStateSpaceContinuousTime`

Examples

```r
# Create and fit a model using mxMatrix, mxExpectationStateSpace, and mxFitFunctionML
require(OpenMx)
data(demoOneFactor)
nvar <- ncol(demoOneFactor)
varnames <- colnames(demoOneFactor)
ssModel <- mxModel(model="State Space Manual Example",
                   mxMatrix("Full", 1, 1, TRUE, .3, name="A"),
                   mxMatrix("Zero", 1, 1, name="B"),
                   mxMatrix("Full", nvar, 1, TRUE, .6, name="C", dimnames=list(varnames, "F1")),
                   mxMatrix("Zero", nvar, 1, name="D"),
                   mxMatrix("Diag", 1, 1, FALSE, 1, name="Q"),
                   mxMatrix("Diag", nvar, nvar, TRUE, .2, name="R"),
                   mxMatrix("Zero", 1, 1, name="x0"),
                   mxMatrix("Diag", 1, 1, FALSE, 1, name="P0"),
                   mxMatrix("Zero", 1, 1, name="u"),
                   mxData( observed=demoOneFactor[1:100,], type="raw"),#fewer rows = fast
                   mxExpectationStateSpace())
mxRun(ssModel)
```
mxExpectationStateSpaceContinuousTime

create and fit a model with inputs using mxMatrix, mxExpectationStateSpace, and mxFitFunctionML

ssRun <- mxRun(ssModel)
summary(ssRun)

# Note the freely estimated Autoregressive parameter (A matrix)
# is near zero as it should be for the independent rows of data
# from the factor model.

# Create and fit a model with inputs using mxMatrix, mxExpectationStateSpace, and mxFitFunctionML
require(OpenMx)
data(demoOneFactor)
nvar <- ncol(demoOneFactor)
varnames <- colnames(demoOneFactor)
demoOneFactorInputs <- cbind(demoOneFactor, V1=rep(1, nrow(demoOneFactor)))
ssModel <- mxModel(model="State Space Inputs Manual Example",
  mxMatrix("Full", 1, 1, TRUE, .3, name="A"),
  mxMatrix("Full", 1, 1, TRUE, values=1, name="B"),
  mxMatrix("Full", nvar, 1, TRUE, .6, name="C", dimnames=list(varnames, "F1")),
  mxMatrix("Zero", nvar, 1, name="D"),
  mxMatrix("Diag", 1, 1, FALSE, 1, name="Q"),
  mxMatrix("Diag", nvar, nvar, TRUE, .2, name="R"),
  mxMatrix("Zero", 1, 1, name="x0"),
  mxMatrix("Diag", 1, 1, FALSE, 1, name="P0"),
  mxMatrix("Full", 1, 1, FALSE, labels="data.V1", name="u"),
  mxData(observed=demoOneFactorInputs[,1:100,], type="raw"),
  mxExpectationStateSpace("A", "B", "C", "D", "Q", "R", "x0", "P0", u="u"),
  mxFitFunctionML()
)
ssRun <- mxRun(ssModel)
summary(ssRun)

# Note the freely estimated Autoregressive parameter (A matrix)
# and the freely estimated Control-Input parameter (B matrix)
# are both near zero as they should be for the independent rows of data
# from the factor model that does not have inputs, covariates,
# or exogenous variables.

mxExpectationStateSpaceContinuousTime

Create an MxExpectationStateSpace Object

Description
This function creates a new MxExpectationStateSpace object.

Usage
mxExpectationStateSpaceContinuousTime(A, B, C, D, Q, R, x0, P0, u, t = NA,
Arguments

- **A**: A character string indicating the name of the 'A' matrix.
- **B**: A character string indicating the name of the 'B' matrix.
- **C**: A character string indicating the name of the 'C' matrix.
- **D**: A character string indicating the name of the 'D' matrix.
- **Q**: A character string indicating the name of the 'Q' matrix.
- **R**: A character string indicating the name of the 'R' matrix.
- **x0**: A character string indicating the name of the 'x0' matrix.
- **P0**: A character string indicating the name of the 'P0' matrix.
- **u**: A character string indicating the name of the 'u' matrix.
- **t**: A character string indicating the name of the 't' matrix.
- **dimnames**: An optional character vector to be assigned to the row names of the 'C' matrix.
- **thresholds**: Not Yet Implemented. An optional character string indicating the name of the thresholds matrix.
- **threshnames**: Not Yet Implemented. An optional character vector to be assigned to the column names of the thresholds matrix.

Details

The *mxExpectationStateSpaceContinuousTime* and *mxExpectationSSCT* functions are identical. The latter is simply an abbreviated name. When using the former, tab completion is strongly encouraged to save tedious typing. Both of these functions are wrappers for the *mxExpectationStateSpace* function, which could be used for both discrete and continuous time modeling. However, there is a strong possibility of misunderstanding the model parameters when switching between discrete time and continuous time. The expectation matrices have the same names, but mean importantly different things so caution is warranted. The best practice is to use *mxExpectationStateSpace* for discrete time models, and *mxExpectationStateSpaceContinuousTime* for continuous time models.

Expectation functions define the way that model expectations are calculated. That is to say, expectation functions define how a set of model matrices get turned into expectations for the data. When used in conjunction with the *mxFitFunctionML*, the *mxExpectationStateSpace* uses maximum likelihood prediction error decomposition (PED) to obtain estimates of free parameters in a model of the raw MxData object. Continuous time state space expectations treat the raw data as a multivariate time series of possibly unevenly spaced times with each row corresponding to a single occasion. Continuous time state space expectations implement a hybrid Kalman filter to produce expectations. The hybrid Kalman filter uses a Kalman-Bucy filter for the prediction step and the classical Kalman filter for the update step. It is a hybrid between the classical Kalman filter used for the discrete (but possibly unequally spaced) measurement occasions and the continuous time Kalman-Bucy filter for latent variable predictions.
Missing data handling is implemented in the same fashion as full information maximum likelihood for partially missing rows of data. Additionally, completely missing rows of data are handled by only using the prediction step from the Kalman-Bucy filter and omitting the update step. This model uses notation for the model matrices commonly found in engineering and control theory. The \( A \), \( B \), \( C \), \( D \), \( Q \), \( R \), \( x_0 \), and \( P_0 \) arguments must be the names of MxMatrix or MxAlgebra objects with the associated properties of the \( A \), \( B \), \( C \), \( D \), \( Q \), \( R \), \( x_0 \), and \( P_0 \) matrices in the state space modeling approach. The \( t \) matrix must be a 1x1 matrix using definition variables that gives the times at which measurements occurred.

The state space expectation is defined by the following model equations.

\[
\frac{d}{dt}x(t) = Ax(t) + Bu_t + q(t)
\]
\[
y_t = Cx_t + Du_t + r_t
\]

with \( q(t) \) and \( r_t \) both independently and identically distributed random Gaussian (normal) variables with mean zero and covariance matrices \( Q \) and \( R \), respectively. Subscripts or square brackets indicate discrete indices; parentheses indicate continuous indices. The derivative of \( x(t) \) with respect to \( t \) is \( \frac{d}{dt} x(t) \).

The first equation is called the state equation. It describes how the latent states change over time with a first-order linear differential equation.

The second equation is called the output equation. It describes how the latent states relate to the observed states at a single point in time. The output equation shows how the observed output is produced by the latent states. Also, the output equation in state space modeling is directly analogous to the measurement model in LISREL structural equation modeling.

Note that the covariates, \( u \), have "instantaneous" effects on both the state and output equations. If lagged effects are desired, then the user must create a lagged covariate by shifting their observed variable to the desired lag.

The state and output equations, together with some minimal assumptions and the Kalman filter, imply a new expected covariance matrix and means vector for every row of data. The expected covariance matrix of row \( t \) is

\[
S_t = C(AP_{t-1}A^T + Q)C^T + R
\]

The expected means vector of row \( t \) is

\[
\hat{y}_t = Cx_t + Du_t
\]

The 'dimnames' arguments takes an optional character vector.

The \( A \) argument refers to the \( A \) matrix in the State Space approach. This matrix gives the dynamics. Entries in the diagonal give the strength of the influence of a variable’s position on its slope. Entries in the off-diagonal give the coupling strength from one variable to another. The \( A \) matrix is sometimes called the state-transition model.

The \( B \) argument refers to the \( B \) matrix in the State Space approach. This matrix consists of exogenous forces that influence the dynamics. Note that the covariate effect is contemporaneous: the covariate at time \( t \) has influence on the slope of the latent state also at time \( t \). A lagged effect
can be created by lagged the observed variable. The $B$ matrix is sometimes called the control-input model.

The 'C' argument refers to the $C$ matrix in the State Space approach. This matrix consists of contemporaneous regression coefficients from the latent variable in column $j$ to the observed variable in row $i$. This matrix is directly analogous to the factor loadings matrix in LISREL and Mplus models. The $C$ matrix is sometimes called the observation model.

The 'D' argument refers to the $D$ matrix in the State Space approach. This matrix consists of contemporaneous regressive coefficients from the input (manifest covariate) variable $j$ to the observed variable in row $i$. The $D$ matrix is sometimes called the feedthrough or feedforward matrix.

The 'Q' argument refers to the $Q$ matrix in the State Space approach. This matrix gives the covariance of the process noise. The dynamic noise can be thought of as unmeasured covariate inputs active at all times. This matrix must be symmetric, diagonal, or zero. As a special case, it is often diagonal. The $Q$ matrix is the covariance of the process noise. Just as in factor analysis and general structural equation modeling, the scale of the latent variables is usually set by fixing some factor loadings in the $C$ matrix, or fixing some factor variances in the $Q$ matrix.

The 'R' argument refers to the $R$ matrix in the State Space approach. This matrix consists of residual covariances among the observed (manifest) variables. This matrix must be symmetric. As a special case, it is often diagonal. The $R$ matrix is the covariance of the observation noise.

The 'x0' argument refers to the $x_0$ matrix in the State Space approach. This matrix consists of the column vector of the initial values for the latent variables. The state space expectation uses the $x_0$ matrix as the starting point to recursively estimate the latent variables’ values at each time. These starting values can be difficult to pick, however, for sufficiently long time series they often do not greatly impact the estimation.

The 'P0' argument refers to the $P_0$ matrix in the State Space approach. This matrix consists of the initial values of the covariances of the error in the initial latent variable estimates given in $x_0$. That is, the $P_0$ matrix gives the covariance of $x_0 - x_{true_0}$ where $x_{true_0}$ is the vector of true initial values. $P_0$ is a measure of the accuracy of the initial latent state estimates. The Kalman filter uses this initial covariance to recursively generated a new covariance for each time point based on the previous time point. The Kalman filter updates this covariance so that it is as small as possible (minimum trace). Similar to the $x_0$ matrix, these starting values are often difficult to choose.

The 'u' argument refers to the $u$ matrix in the State Space approach. This matrix consists of the inputs or manifest covariates of the state space expectation. The $u$ matrix must be a column vector with the same number of rows as the $B$ and $D$ matrices have columns. If no inputs are desired, $u$ can be a zero matrix. If time-varying inputs are desired, then they should be included as columns in the MxData object and referred to in the labels of the $u$ matrix as definition variables. There is an example of this below.

The 't' argument refers to the $t$ matrix in the State Space approach. This matrix should be 1x1 (1 row and 1 column) and not free. The label for the element of this matrix should be 'data.YourTimeVariable'. The 'data' part does not change, but 'YourTimeVariable' should be a name in your data set that gives the times at which measurement happened. The units of time are up to you. Your choice of time units will influence of the values of the parameters you estimate. Also, recall that the model is given $x_0$ and $P_0$. These always happen at $t = 0$. So the first row of data happens some amount of time after zero.

The MxMatrix objects included as arguments may be of any type, but should have the properties described above. The mxExpectationStateSpace will not return an error for incorrect specification, but incorrect specification will likely lead to estimation problems or errors in the mxRun function.
mxExpectationStateSpaceContinuousTime evaluates with respect to an MxData object. The MxData object need not be referenced in the mxExpectationStateSpace function, but must be included in the MxModel object. mxExpectationStateSpace requires that the 'type' argument in the associated MxData object be equal to 'raw'. Neighboring rows of the MxData object are treated as adjacent, equidistant time points increasing from the first to the last row.

To evaluate, place an mxExpectationStateSpaceContinuousTime object, 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 MxExpectationStateSpace object. mxExpectationStateSpace 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.

See Also

mxExpectationStateSpace

Examples

#---------------------------------------------------------------
# Example 1
# Undamped linear oscillator, i.e. a noisy sine wave.
# Measurement error, but no dynamic error, single indicator.
# This example works great.
#
#---------------------------------------------------------------

require(OpenMx)

set.seed(405)
tlen <- 200
t <- seq(1.2, 50, length.out = tlen)

freqParam <- .5
initialCond <- matrix(c(2.5, 0))
x <- initialCond[1,1]*cos(freqParam*t)
plot(t, x, type='l')

measVar <- 1.5
y <- cbind(obs=x+rnorm(tlen, sd=sqrt(measVar)), tim=t)
plot(t, y[,1], type='l')

#---------------------------------------------------------------
# Model Specification
#Note: the bounds are here only to keep SLSQP from
# stepping too far off a cliff. With the bounds in
# place, SLSQP finds the right solution. Without
# the bounds, SLSQP goes crazy.

cdim <- list('obs', c('ksi', 'ksiDot'))
amat <- mxMatrix('Full', 2, 2, c(FALSE, TRUE, FALSE, TRUE), c(0, -.1, 1, -.2),
         name='A', lbound=-10)
bmat <- mxMatrix('Zero', 2, 1, name='B')
cmat <- mxMatrix('Full', 1, 2, FALSE, c(1, 0), name='C', dimnames=cdim)
dmat <- mxMatrix('Zero', 1, 1, name='D')
qmat <- mxMatrix('Diag', 2, 2, name='Q')
rmat <- mxMatrix('Diag', 1, 1, TRUE, .4, name='R', lbound=1e-6)
xmat <- mxMatrix('Full', 2, 1, TRUE, c(0, 0), name='x0', lbound=-10, ubound=10)
pmat <- mxMatrix('Diag', 2, 2, FALSE, 1, name='P0')
umat <- mxMatrix('Zero', 1, 1, name='u')
tmat <- mxMatrix('Full', 1, 1, name='time', labels='data.tim')
osc <- mxModel("LinearOscillator",
amat, bmat, cmat, dmat, qmat, rmat, xmat, pmat, umat, tmat,
mxExpectationSSCT('A', 'B', 'C', 'D', 'Q', 'R', 'x0', 'P0', 'u', 'time'),
mxFitFunctionML(),
mxData(y, 'raw'))

oscr <- mxRun(osc)

#---------------------------------------------------------------
# Results Examination

summary(oscr)

(ssFreqParam <- mxEval(sqrt(-A[2,1]), oscr))
freqParam
mxFactor

Fail-safe Factors

Description

This is a wrapper for the R function factor.

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

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

Usage

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

Arguments

x

either a vector of data or a data.frame object.

levels

a mandatory vector of the values that ’x’ might have taken.

labels

either an optional vector of labels for the levels, or a character string of length 1.

exclude

a vector of values to be excluded from the set of levels.

ordered

logical flag to determine if the levels should be regarded as ordered (in the order given). Required to be TRUE.

collapse

logical flag to determine if duplicate labels should collapsed into a single level
Details

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

References

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

Examples

```r
myVar <- c('s', 't', 'a', 't', 'i', 's', 't', 'i', 'c', 's')
ff <- mxFactor(myVar, levels=letters)
# Note: letters is a built in list of all lowercase letters of the alphabet
# [1] statistics
# Levels: a < b < c < d < e < f < g < h < i < j < k < l < m < n < o < p < q <
# r < s < t < u < v < w < x < y < z
as.integer(ff) # the internal codes
factor(ff)     # NOTE: drops the levels that do not occur.
                # mxFactor prevents you doing this unintentionally.

# This example works on a dataframe
foo <- data.frame(x=c(1:3), y=c(4:6), z=c(7:9))

# Applies one set of levels to all three columns
mxFactor(foo, c(1:9))

# Apply unique sets of levels to each variable
mxFactor(foo, list(c(1:3), c(4:6), c(7:9)))

mxFactor(foo, c(1:9), labels=c(1,1,2,2,2,3,3), collapse=TRUE)
```

---

### mxFIMLObjective

**DEPRECATED: Create MxFIMLObjective Object**

### Description

WARNING: Objective functions have been deprecated as of OpenMx 2.0.

Please use `mxExpectationNormal()` and `mxFitFunctionML()` instead. As a temporary workaround, `mxFIMLObjective` returns a list containing an `MxExpectationNormal` object and an `MxFitFunctionML` object.
All occurrences of

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

Should be changed to

```r
mxExpectationNormal(covariance, means, dimnames = NA, thresholds = NA, threshnames = dim-
names) mxFitFunctionML(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.
- **threshnames**: An optional character vector to be assigned to the column names of the thresh-
  olds matrix.

**Details**

NOTE: THIS DESCRIPTION IS DEPRECATED. Please change to using `mxExpectationNormal` and `mxFitFunctionML` as shown in the example below.

Objective functions were functions for which free parameter values are chosen such that the value of the objective function is minimized. The `mxFIMLObjective` function used 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 is used to assign the dimnames of the means vector as well as 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.

**thresholds**: The name of the thresholds matrix. When needed (for modelling ordinal data), this matrix should be created using `mxMatrix()`. The thresholds matrix must have as many columns as there are ordinal variables in the model, and number of rows equal to one fewer than the maximum number of levels found in the ordinal variables. The starting values of this matrix must also be set to reasonable values. Fill each column with a set of ordered start thresholds, one for each level of this column’s factor levels minus 1. These thresholds may be free if you wish them to be estimated, or fixed. The unused rows in each column, if any, can be set to any value including NA.

**threshnames**: A character vector consisting of the variables in the thresholds matrix, i.e., the names of ordinal variables in a model. This is necessary for OpenMx to map the thresholds matrix columns onto the variables in your data. If you set the `dimnames` of the columns in the thresholds matrix then threshnames is not needed.
Usage Notes: 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.

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.

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 reported using the summary function, or accessed directly in the 'output' slot of the resulting model (i.e., modelName$output). Components of the output may be referenced using the Extract functionality.

Value

Returns a list containing an MxExpectationNormal object and an MxFitFunctionML object.

References

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

Examples

# Create and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionML
library(OpenMx)

# Simulate some data
x=rnorm(1000, mean=0, sd=1)
y= 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices
M <- mxMatrix(type = "Full", nrow = 1, ncol = 2, values=c(0,0),
              free=c(TRUE,TRUE), labels=c("Mx", "My"), name = "M")
S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1,0,0,1),
              free=c(TRUE,FALSE,FALSE,TRUE), labels=c("Vx", NA, NA, "Vy"), name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,1,0,0),
              free=c(FALSE,TRUE,FALSE,FALSE), labels=c(NA, "b", NA, NA), name = "A")
I <- mxMatrix(type="Iden", nrow=2, ncol=2, name="I")

# Define the expectation
expCov <- mxAlgebra(solve(I-A) %*% S %*% t(solve(I-A)), name="expCov")
expFunction <- mxExpectationNormal(covariance="expCov", means="M", dimnames=tmpNames)
# Choose a fit function

fitFunction <- mxFitFunctionML()

# Define the model

tmpModel <- mxModel(model="exampleModel", M, S, A, I, expCov, expFunction, fitFunction, 
mxData( observed=tmpFrame, type="raw")

# Fit the model and print a summary

tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

---

**MxFitFunction-class**  
**MxFitFunction**

**Description**

This is an internal class and should not be used directly.

---

**mxFitFunctionAlgebra**  
**Create MxFitFunctionAlgebra Object**

**Description**

mxFitFunctionAlgebra returns an MxFitFunctionAlgebra object.

**Usage**

mxFitFunctionAlgebra(algebra, numObs = NA, numStats = NA, ..., gradient = 
NA_character_, hessian = NA_character_, verbose = 0L, units="-2lnL")

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algebra</td>
<td>A character string indicating the name of an MxAlgebra or MxMatrix object to use for optimization.</td>
</tr>
<tr>
<td>numObs</td>
<td>(optional) An adjustment to the total number of observations in the model.</td>
</tr>
<tr>
<td>numStats</td>
<td>(optional) An adjustment to the total number of observed statistics in the model.</td>
</tr>
<tr>
<td>...</td>
<td>Not used. Forces remaining arguments to be specified by name.</td>
</tr>
<tr>
<td>gradient</td>
<td>(optional) A character string indicating the name of an MxAlgebra object.</td>
</tr>
<tr>
<td>hessian</td>
<td>(optional) A character string indicating the name of an MxAlgebra object.</td>
</tr>
<tr>
<td>verbose</td>
<td>(optional) An integer to increase the level of runtime log output.</td>
</tr>
<tr>
<td>units</td>
<td>(optional) The units of the fit statistic.</td>
</tr>
</tbody>
</table>
Details

Fit functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. While the other fit functions in OpenMx require an expectation function for the model, the mxAlgebraObjective function uses the referenced MxAlgebra or MxMatrix object as the function to be minimized.

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

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

First and second derivatives can be provided with the algebra fit function. The dimnames on the gradient and hessian MxAlgebras are matched against names of free variables. Names that do not match are ignored. If you are working in log likelihood units, the customary -2 scaling factor is not applied automatically. You have to take care of multiplying by -2 yourself.

Value

Returns an MxFitFunctionAlgebra object. MxFitFunctionAlgebra objects should be included with models with referenced MxAlgebra and MxMatrix objects.

References

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

See Also

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

Examples

# Create and fit a very simple model that adds two numbers using mxFitFunctionAlgebra
library(OpenMx)

# Create a matrix 'A' with no free parameters
A <- mxMatrix(’Full’, nrow = 1, ncol = 1, values = 1, 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'
onjective <- mxFitFunctionAlgebra(’B’)

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

# Evaluate the algebra
tmpModelOut <- mxRun(tmpModel)

# View the results
tmpModelOut$output$minimum

mxFitFunctionGREML  

Create MxFitFunctionGREML Object

Description

This function creates a new MxFitFunctionGREML object.

Usage

mxFitFunctionGREML(dV=character(0))

Arguments

dV  Vector of character strings; defaults to a character vector of length zero. If a value of non-zero length is provided, it must be a named character vector. This vector's names must be the labels of free parameters in the model. The vector's elements (i.e., the character strings themselves) must be the names of MxAlgebra or MxMatrix object(s), each of which equals the first partial derivatives of the 'V' matrix with respect to the corresponding free parameter.

Details

Making effective use of argument dV will usually require a custom mxComputeSequence(). The derivatives of the REML loglikelihood function with respect to parameters can be internally computed from the derivatives of the 'V' matrix supplied via dV. These loglikelihood derivatives will be valid as long as (1) the derivatives of 'V' evaluate to symmetric matrices the same size as 'V,' and (2) the model contains no MxConstraints.

Value

Returns a new object of class MxFitFunctionGREML.

References

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

See `mxFitFunctionGREML` for the S4 class created by `mxFitFunctionGREML()`. For more information generally concerning GREML analyses, including a complete example, see `mxExpectationGREML()`. More information about the OpenMx package may be found here.

Examples

```r
gff <- mxFitFunctionGREML()
str(gff)
```

Description

`mxFitFunctionGREML` is the fitfunction class for GREML analyses.

Objects from the Class

Objects can be created by calls of the form `mxFitFunctionGREML(dV)`.

Slots

- `dV`: Object of class "MxCharOrNumber". Identifies the `MxAlgebra` or `MxMatrix` object(s) to serve as the derivatives of 'V' with respect to free parameters.
- `dVnames`: Vector of character strings; names of the free parameters corresponding to slot `dV`.
- `info`: Object of class "list".
- `dependencies`: Object of class "integer".
- `expectation`: Object of class "integer".
- `vector`: Object of class "logical".
- `result`: Object of class "matrix".
- `name`: Object of class "character".
- `numObs`: Object of class "integer".

Extends

Class "MxBaseFitFunction", directly. Class "MxBaseNamed", by class "MxBaseFitFunction", distance 2. Class "MxFitFunction", by class "MxBaseFitFunction", distance 2.

Methods

No methods defined with class "MxFitFunctionGREML" in the signature.
References

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

See Also

See `mxFitFunctionGREML()` for creating MxFitFunctionGREML objects. See `mxExpectationGREML()` for creating MxExpectationGREML objects, and for more information generally concerning GREML analyses, including a complete example. More information about the OpenMx package may be found here.

Examples

```r
showClass("MxFitFunctionGREML")
```

---

### `mxFitFunctionML` Create MxFitFunctionML Object

**Description**

This function creates a new MxFitFunctionML object.

**Usage**

```r
mxFitFunctionML(vector = FALSE, rowDiagostics = FALSE)
```

**Arguments**

- **vector**: A logical value indicating whether the objective function result is the likelihood vector.
- **rowDiagostics**: A logical value indicating whether the row-wise results of the objective function should be returned as an attribute of the fit function.

**Details**

Fit functions are functions for which free parameter values are optimized such that the value of a cost function is minimized. The `mxFitFunctionML` function computes -2*(log likelihood) of the data given the current values of the free parameters and the expectation function (e.g., `mxExpectationNormal` or `mxExpectationRAM`) selected for the model.

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.

The 'rowDiagostics' argument is either TRUE or FALSE, and determines whether the row likelihoods are returned as an attribute of the fit function. It is sometimes useful to inspect the likelihoods for outliers, diagnostics, or other anomalies.

When vector=FALSE and rowDiagostics=TRUE, fitfunction can be referenced in the model and included in algebras as a scalar. The row likelihoods are an attribute of the fit function but are not accessible in the model.
Usage Notes:
The results of the optimization can be reported using the `summary` function, or accessed directly in the 'output' slot of the resulting model (i.e., `modelName$output`). Components of the output may be referenced using the `Extract` functionality.

Value
Returns a new MxFitFunctionML object. One and only one MxFitFunctionML object should be included in each model along with an associated `mxExpectationNormal` or `mxExpectationRAM` object.

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

Examples

```r
# Create and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionML
library(OpenMx)

# Simulate some data
x <- rnorm(1000, mean=0, sd=1)
y <- 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices
M <- mxMatrix(type = "Full", nrow = 1, ncol = 2, values=c(0,0),
               free=c(TRUE,TRUE), labels=c("Mx", "My"), name = "M")
S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,0,0,0),
              free=c(TRUE,FALSE,FALSE,TRUE), labels=c("Vx", NA, NA, "Vy"), name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,0,0,0),
              free=c(FALSE,TRUE,FALSE,FALSE), labels=c("A", "b", NA, NA), name = "A")
I <- mxMatrix(type="Iden", nrow=2, ncol=2, name="I")

# Define the expectation
expCov <- mxAlgebra(solve(I-A) %*% S %*% t(solve(I-A)), name="expCov")
expFunction <- mxExpectationNormal(covariance="expCov", means="M", dimnames=tmpNames)

# Choose a fit function
fitFunction <- mxFitFunctionML()

# Define the model
tmpModel <- mxModel(model="exampleModel", M, S, A, I, expCov, expFunction, fitFunction,
                    mxData(observed=tmpFrame, type="raw"))
```
mxFitFunctionMultigroup

Aggregate fit statistics from multiple submodels

Description

mxFitFunctionMultigroup creates a fit function consisting of the sum of the fit statistics from a list of submodels provided.

This is conceptually similar to creating an mxAlgebra consisting of the sum of the subModel objectives and also creating an algebra fit function to optimize the model based on this aggregate value.

This call to mxFitFunctionMultigroup:
mxFitFunctionMultigroup(c("model1", "model2"))
then, is almost equivalent to the following pair of statements:
mxAlgebra(model1.objective + model2.objective, name="myAlgebra")
mxFitFunctionAlgebra("myAlgebra")

In addition to being more compact and readable, using mxFitFunctionMultigroup has additional side effects which are valuable for multi-group modeling.

Firstly, it aggregates analytic derivative calculations. Secondly, it allows mxRefModels to compute saturated models for raw data, as this function can learn which are the constituent submodels.

Note: You can refer to the algebra generated by mxFitFunctionMultigroup when used in a group "modelName" as:
modelName.fitfunction

Usage

mxFitFunctionMultigroup(groups, ..., verbose = 0L)

Arguments

- groups: vector of fit function names (strings)
- ...: Not used. Forces subsequent arguments to be specified by name.
- verbose: the level of debugging output

Examples

require("OpenMx")
mxFitFunctionMultigroup(c("model1", "model2")) # names of sub-models to be jointly optimised
Description

mxFitFunctionR returns an MxFitFunctionR object.

Usage

```r
mxFitFunctionR(fitfun, ..., units="-2lnL")
```

Arguments

- `fitfun` A function that accepts two arguments.
- `...` The initial state information to the objective function.
- `units` (optional) The units of the fit statistic.

Details

The `mxFitFunctionR` function evaluates a user-defined R function called the 'fitfun'. `mxFitFunctionR` is useful in defining new `mxFitFunctions`, since any calculation that can be performed in R can be treated as an `mxFitFunction`.

The 'fitfun' 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.

Throwing an exception (via stop) from inside `fitfun` may result in unpredictable behavior. You may want to wrap your code in `tryCatch` while experimenting.

Value

Returns an MxFitFunctionR object.

References

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

# Create and fit a model using mxFitFunctionR
library(OpenMx)

A <- mxMatrix(nrow = 2, ncol = 2, values = c(1:4), free = TRUE, name = 'A')
squared <- function(x) { x ^ 2 }

# Define the objective function in R
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))
}

# Define the expectation function
fitFunction <- mxFitFunctionR(objFunction)

# Define the model
tmpModel <- mxModel(model="exampleModel", A, fitFunction)

# Fit the model and print a summary
tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

mxFitFunctionRow

Create an MxFitFunctionRow Object

Description

mxFitFunctionRow returns an MxFitFunctionRow object.

Usage

mxFitFunctionRow(rowAlgebra, reduceAlgebra, dimnames,
                  rowResults = "rowResults", filteredDataRow = "filteredDataRow",
                  existenceVector = "existenceVector", units="-2lnL")

Arguments

rowAlgebra A character string indicating the name of the algebra to be evaluated row-wise.
reduceAlgebra A character string indicating the name of the algebra that collapses the row results into a single number which is then optimized.
dimnames A character vector of names corresponding to columns be extracted from the data set.
rowResults The name of the auto-generated "rowResults" matrix. See details.
filteredDataRow The name of the auto-generated "filteredDataRow" matrix. See details.
eexistenceVector The name of the auto-generated "existenceVector" matrix. See details.
units (optional) The units of the fit statistic.

Details
Fit functions are functions for which free parameter values are optimized such that the value of a cost function is minimized. The mxFitFunctionRow function evaluates a user-defined MxAlgebra object called the 'rowAlgebra' in a row-wise fashion. It then stores results of the row-wise evaluation in another MxAlgebra object called the 'rowResults'. Finally, the mxFitFunctionRow function collapses the row results into a single number which is then used for optimization. The MxAlgebra object named by the 'reduceAlgebra' collapses the row results into a single number.
The ‘filteredDataRow’ is populated in a row-by-row fashion with all the non-missing data from the current row. You cannot assume that the length of the filteredDataRow matrix remains constant (unless you have no missing data). The ‘existenceVector’ is populated in a row-by-row fashion with a value of 1.0 in column j if a non-missing value is present in the data set in column j, and a value of 0.0 otherwise. Use the functions omxSelectRows, omxSelectCols, and omxSelectRowsAndCols to shrink other matrices so that their dimensions will be conformable to the size of ‘filteredDataRow’.

Value
Returns a new MxFitFunctionRow object. Only one MxFitFunction object should be included in each model. There is no need for an MxExpectation object when using mxFitFunctionRow.

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

Examples
# Model that adds two data columns row-wise, then sums that column
# Notice no optimization is performed here.
library(OpenMx)

xdat <- data.frame(a=rnorm(10), b=1:10) # Make data set
amod <- mxModel(model="example1",
               mxData(observed=xdat, type='raw'),
               mxAlgebra(sum(filteredDataRow), name = 'rowAlgebra'),
               mxAlgebra(sum(rowResults), name = 'reduceAlgebra'),
               mxFitFunctionRow(
               rowAlgebra='rowAlgebra',
               reduceAlgebra='reduceAlgebra',
               dimnames=c('a','b'))
mxFitFunctionWLS

Create MxFitFunctionWLS Object

Description

This function creates a new MxFitFunctionWLS object.

Usage

mxFitFunctionWLS(weights = "ULS")

Arguments

weights Ignored. Uses weights from mxData

Details

Fit functions are functions for which free parameter values are optimized such that the value of a cost function is minimized. The mxFitFunctionWLS function computes the weighted least squares difference between the data and the model-implied expectations for the data based on the free parameters and the expectation function (e.g., mxExpectationNormal or mxExpectationRAM) selected for the model.

The 'weights' argument is ignored. Rather the weights are provided in the mxData object, often generated by the mxDataWLS function.

Usage Notes:
The results of the optimization can be reported using the `summary` function, or accessed directly in the 'output' slot of the resulting model (i.e., `modelName$output`). Components of the output may be referenced using the `Extract` functionality.

**Value**

Returns a new `MxFitFunctionWLS` object. One and only one `MxFitFunctionWLS` object should be included in each model along with an associated `mxExpectationNormal` or `mxExpectationRAM` object.

**References**

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

**Examples**

```r
# Create and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionWLS
library(OpenMx)

# Simulate some data
x <- rnorm(1000, mean=0, sd=1)
y <- 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)
wdx <- mxDataWLS(tmpFrame)

# Define the matrices
S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1, 0, 0, 1),
              free=c(TRUE, FALSE, FALSE, TRUE), labels=c("Vx", NA, NA, "Vy"), name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0, 1, 0, 0),
              free=c(FALSE, TRUE, FALSE, FALSE), labels=c(NA, "b", NA, NA), name = "A")
I <- mxMatrix(type = "Iden", nrow=2, ncol=2, name = "I")

# Define the expectation
expCov <- mxAlgebra(solve(I-A) %*% S %*% t(solve(I-A)), name="expCov")
expFunction <- mxExpectationNormal(covariance="expCov", dimnames=tmpNames)

# Choose a fit function
fitFunction <- mxFitFunctionWLS()

# Define the model
tmpModel <- mxModel(model="exampleModel", S, A, I, expCov, expFunction, fitFunction, wdata)
```
# Fit the model and print a summary

tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

---

### MxFlatModel-class  
**MxFlatModel**

**Description**

This is an internal class and should not be used.

---

### mxGenerateData  
**Generate data based on an MxModel object**

**Description**

This function creates a randomly sampled data set based on the model.

**Usage**

mxGenerateData(model, nrows)

**Arguments**

- **model**  
  An MxModel object upon which the data are generated.

- **nrows**  
  Numeric. The number of rows of data to generate.

**Details**

This function looks inside the MxModel object to extract the model-implied means and covariance. It then generates data with the same mean and covariance. Data can be generated based on Normal (mxExpectationNormal), RAM (mxExpectationRAM), and LISREL (mxExpectationLISREL) models.

Thresholds and ordinal data are implemented by generating continuous data and then using cut and mxFactor to break the continuous data at the thresholds into an ordered factor.

For state space models (i.e. models with an mxExpectationStateSpace expectation), the data are generated based on the autoregressive structure of the model. The rows of data in a state space model are not independent replicates of a stationary process. Rather, they are the result of a latent (possibly non-stationary) autoregressive process. For state space models different rows of data often correspond to different times.

**Value**

A matrix or data.frame with nrows rows.
References

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

Examples

#--------
# Create data based on state space model.
require(OpenMx)
nvar <- 5
varnames <- paste("x", 1:nvar, sep=""")
ssModel <- mxModel(model="State Space Manual Example",
  mxMatrix("Full", 1, 1, TRUE, .3, name="A"),
  mxMatrix("Zero", 1, 1, name="B"),
  mxMatrix("Full", nvar, 1, TRUE, .6, name="C", dimnames=list(varnames, "F1")),
  mxMatrix("Zero", nvar, 1, name="D"),
  mxMatrix("Diag", 1, 1, FALSE, 1, name="Q"),
  mxMatrix("Diag", nvar, nvar, TRUE, .2, name="R"),
  mxMatrix("Zero", 1, 1, name="x0"),
  mxMatrix("Diag", 1, 1, FALSE, 1, name="P0"),
  mxMatrix("Zero", 1, 1, name="u"),
  mxExpectationStateSpace("A", "B", "C", "D", "Q", "R", "x0", "P0", "u"),
  mxFitFunctionML())

ssData <- mxGenerateData(ssModel, 200) # 200 time points

# Add simulated data to model
ssModel <- mxModel(ssModel, mxData(ssData, 'raw'))

# Fit model to simulated data
ssRun <- mxRun(ssModel)

# Compare parameters estimated from random data to
# their true generating values
cbind(Rand=omxGetParameters(ssRun), Gen=omxGetParameters(ssModel))
# Note the parameters should be "close" (up to sampling error)
# to the generating values

#--------
require(OpenMx)
manifests <- paste("x", 1:5, sep=""")
latents <- c("G")
factorModel <- mxModel("One Factor",
  type="RAM",
  manifestVars = manifests,
  latentVars = latents,
  mxPath(from=latents, to=manifests, values=.8),
  mxPath(from=manifests, arrows=2, values=.2),
  mxPath(from=latents, arrows=2,
        free=FALSE, values=1.0),
mxGetExpected

Extract the component from a model’s expectation

Description
This function extracts the expected means, covariance, or thresholds from a model.

Usage
mxGetExpected(model, component)
imxGetExpectationComponent(model, component)

Arguments
model MxModel object from which to extract the expectation component.
component A character. The name of the component to extract.

Details
The expected means, covariance, or thresholds can be extracted from Normal (mxExpectationNormal), RAM (mxExpectationRAM), and LISREL (mxExpectationLISREL) models.

Value
A matrix, giving the expectation component requested.

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

Examples

#--------
require(OpenMx)
manifests <- paste("x", 1:5, sep="")
latents <- c("G")
factorModel <- mxModel("One Factor",
    type="RAM", 
    mxPath(from = 'one', to = manifests))

factorData <- mxGenerateData(factorModel, 100)
factorModel <- mxModel(factorModel, mxData(factorData, 'raw'))
factorRun <- mxRun(factorModel)
cbind(Rand=omxGetParameters(factorRun), Gen=omxGetParameters(factorModel))
mxGREMLDataHandler

Helper Function for Structuring GREML Data

Description

This function takes a dataframe or matrix and uses it to setup the 'y' and 'X' matrices for a GREML analysis; this includes trimming out NAs from 'X' and 'y.' The result is a matrix the first column of which is the 'y' vector, and the remaining columns of which constitute 'X.'

Usage

mxGREMLDataHandler(data, yvars=character(0), Xvars=list(), addOnes=TRUE, blockByPheno=TRUE, staggerZeroes=TRUE)

Arguments

data Either a dataframe or matrix, with column names, containing the variables to be used as phenotypes and covariates in 'y' and 'X,' respectively.
yvars Character vector. Each string names a column of the raw dataset, to be used as a phenotype.
Xvars A list of data column names, specifying the covariates to be used with each phenotype. The list should have the same length as argument yvars.
addOnes Logical; should lead columns of ones (for the regression intercepts) be adhered to the covariates when assembling the 'X' matrix? Defaults to TRUE.
blockByPheno Logical; relevant to polyphenotype analyses. If TRUE (default), then the resulting 'y' will contain phenotype #1 for individuals 1 thru n, phenotype #2 for individuals 1 thru n, ... If FALSE, then observations are "blocked by individual", and the resulting 'y' will contain individual #1's scores on phenotypes 1 thru p, individual #2's scores on phenotypes 1 thru p, ... Note that in either case, 'X' will be structured appropriately for 'y.'
staggerZeroes Logical; relevant to polyphenotype analyses. If TRUE (default), then each phenotype's covariates in 'X' are "staggered," and 'X' is padded out with zeroes. If FALSE, then 'X' is formed simply by stacking the phenotypes' covariates;
this requires each phenotype to have the same number of covariates (i.e., each character vector in xvars must be of the same length). The default (TRUE) is intended for instances where the multiple phenotypes truly are different variables, whereas staggerZeros=FALSE is intended for instances where the multiple "phenotypes" actually represent multiple observations on the same variable. One example of the latter case is longitudinal data where the multiple "phenotypes" are repeated measures on a single phenotype.

Details

For a monophenotype analysis (only), argument Xdata can be a character vector. In a polyphenotype analysis, if the same covariates are to be used with all phenotypes, then Xdata can be a list of length 1.

Note the synergy between the output of mxGREMLDataHandler() and arguments dataset.is.yX and casesToDropFromV to mxExpectationGREML().

If the dataframe or matrix supplied for argument data has n rows, and argument yvars is of length p, then the resulting 'y' and 'X' matrices will have np rows. Then, if either matrix contains any NA's, the rows containing the NA's are trimmed from both 'X' and 'y' before being returned in the output (in which case they will obviously have fewer than np rows). Function mxGREMLDataHandler() reports which rows of the full-size 'X' and 'y' were trimmed out due to missing observations. These row indices can be provided as argument casesToDropFromV to mxExpectationGREML().

Value

A list with these two components:

yX Numeric matrix. The first column is the phenotype vector, 'y,' while the remaining columns constitute the 'X' matrix of covariates. If this matrix is used as the raw dataset for a model, then the model's GREML expectation can be constructed with dataset.is.yX=TRUE in mxExpectationGREML().

casesToDrop Numeric vector. Contains the indices of the rows of the 'y' and 'X' that were dropped due to containing NA's. Can be provided as as argument casesToDropFromV to mxExpectationGREML().

References

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

See Also

For more information generally concerning GREML analyses, including a complete example, see mxExpectationGREML(). More information about the OpenMx package may be found here.

Examples

dat <- cbind(rnorm(100),rep(1,100))
colnames(dat) <- c("y","x")
dat[42,1] <- NA
dat[57,2] <- NA
MxInterval-class

MxInterval

Description

This is an internal class and should not be used directly.

See Also

mxCI

mxKalmanScores

Estimate Kalman scores and error covariance matrices

Description

This function creates the Kalman predicted, Kalman updated, and Rauch-Tung-Striebel smoothed latent state and error covariance estimates for an MxModel object that has an MxExpectationStateSpace object.

Usage

mxKalmanScores(model, data=NA)

Arguments

model An MxModel object with an MxExpectationStateSpace.

data An optional data.frame or matrix.

Details

This is a helper function that computes the results of the classical Kalman filter. In particular, for every row of data there is a predicted latent score, an error covariance matrix for the predicted latent scores that provides an estimate of the predictions precision, an updated latent score, and an updated error covariance matrix for the updated lated scores. Additionally, the Rauch-Tung-Striebel (RTS) smoothed latent scores and error covariance matrices are returned.
Value

A list with components xPredicted, PPredicted, xUpdated, PUpdated, xSmoothed, PSmoothed, m2ll, and L. The rows of xPredicted, xUpdated, and xSmoothed correspond to different time points. The columns are the different latent variables. The third index of PPredicted, PUpdated, and PSmoothed corresponds to different times. This works nicely with the R default print method for arrays. At each time there is a covariance matrix of the latent variable scores. For all items listed below, the first element goes with the zeroth time point (See example).

- **xPredicted** matrix of Kalman predicted scores
- **PPredicted** array of Kalman predicted error covariances
- **xUpdated** matrix of Kalman updated scores
- **PUpdated** array of Kalman updated error covariances
- **xSmoothed** matrix of RTS smoothed scores
- **PSmoothed** array of RTS smoothed error covariances
- **m2ll** minus 2 log likelihood
- **L** likelihood

References


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

See Also

- **mxExpectationStateSpace**

Examples

```r
# Create and fit a model using mxMatrix, mxExpectationStateSpace, and mxFitFunctionML
require(OpenMx)
data(demoOneFactor)
# Use only first 50 rows, for speed of example
data <- demoOneFactor[1:50,]
nvar <- ncol(demoOneFactor)
varnames <- colnames(demoOneFactor)
ssModel <- mxModel(model="State Space Manual Example",
  mxMatrix("Full", 1, 1, TRUE, .3, name="A"),
  mxMatrix("Zero", 1, 1, name="B"),
  mxMatrix("Full", nvar, 1, TRUE, .6, name="C", dimnames=list(varnames, "F1")),
  mxMatrix("Zero", nvar, 1, name="D"),
  mxMatrix("Diag", 1, 1, FALSE, 1, name="Q"),
  mxExpectationStateSpace(
    observed = "data",
    numObs = nrow(data),
    numStates = nvar,
    numTimes = 50,
    stateVarNames = varnames,
    stateDimnames = "F1",
    stateCovMatrices = c("A", "B", "C", "D", "Q"),
    stateCovMeasMatrices = c("A", "B", "C", "D", "Q"),
    stateTransMatrices = c("A", "B", "C", "D", "Q"),
    stateTransMeasMatrices = c("A", "B", "C", "D", "Q"),
    stateCovMeasData = diag(nvar),
    stateTransData = diag(nvar),
    stateCovTransData = diag(nvar),
    stateTransMeasData = diag(nvar),
    meu = rep(0, nvar),
    me = rep(0, nvar),
    from = "Df1",
    to = "Df1",
    type = "full"),
  mxFitFunctionML()
)
ssModelSummary <- mxRun(ssModel)
summary(ssModelSummary)
```

"mxKalmanScores"
```r
mxMatrix("Diag", nvar, nvar, TRUE, .2, name="R"),
mxMatrix("Zero", 1, 1, name="x0"),
mxMatrix("Diag", 1, 1, FALSE, 1, name="P0"),
mxMatrix("Zero", 1, 1, name="u"),
mxData(observed=data, type="raw"),
mxExpectationStatespace("A", "B", "C", "D", "Q", "R", "x0", "P0", "u"),
mxFitFunctionML()
)
ssRun <- mxRun(ssModel)
summary(ssRun)
# Note the freely estimated Autoregressive parameter (A matrix)
# is near zero as it should be for the independent rows of data
# from the factor model.
ssScores <- mxKalmanScores(ssRun)
cor(cbind(ssScores$xPredicted[,1], ssScores$xUpdated[,1], ssScores$xSmoothed[,1]))
# Because the autoregressive dynamics are near zero, the predicted and updated scores
# correlate minimally, and the updated and smoothed latent state estimates
# are extremely close.

# The first few latent predicted scores
head(ssScores$xPredicted)

# The predicted latent score for time 10
ssScores$xPredicted[10+1,]

# The error covariance of the predicted score at time 10
ssScores$PPredicted[10+1]
```

---

**MxLISRELModel-class**  

**Description**

This is an internal class and should not be used directly.

---

**mxEvaluationObjective**  

Create MxLISRELObjective Object

**Description**

This function creates a new MxLISRELObjective object.
mxLISRELObjective

Usage

mxLISRELObjective(LX=NA, LY=NA, BE=NA, GA=NA, PH=NA, PS=NA, TD=NA, TE=NA, TH=NA, TX = NA, TY = NA, KA = NA, AL = NA, 
dimnames = NA, thresholds = NA, vector = FALSE, threshnames = dimnames)

Arguments

LX An optional character string indicating the name of the 'LX' matrix.
LY An optional character string indicating the name of the 'LY' matrix.
BE An optional character string indicating the name of the 'BE' matrix.
GA An optional character string indicating the name of the 'GA' matrix.
PH An optional character string indicating the name of the 'PH' matrix.
PS An optional character string indicating the name of the 'PS' matrix.
TD An optional character string indicating the name of the 'TD' matrix.
TE An optional character string indicating the name of the 'TE' matrix.
TH An optional character string indicating the name of the 'TH' matrix.
TX An optional character string indicating the name of the 'TX' matrix.
TY An optional character string indicating the name of the 'TY' matrix.
KA An optional character string indicating the name of the 'KA' matrix.
AL An optional character string indicating the name of the 'AL' matrix.
dimnames An optional character vector that is currently ignored
thresholds An optional character string indicating the name of the thresholds matrix.
vector A logical value indicating whether the objective function result is the likelihood vector.
threshnames An optional character vector to be assigned to the column names of the thresholds matrix.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxLISRELObjective provides maximum likelihood estimates of free parameters in a model of the covariance of a given MxData object. This model is defined by Linear Structural RELations (LISREL; Jöreskog & Sörbom, 1982, 1996). Arguments 'LX' through 'AL' must refer to MxMatrix objects with the associated properties of their respective matrices in the LISREL modeling approach.

The full LISREL specification has 13 matrices and is sometimes called the extended LISREL model. It is defined by the following equations.

\[
\eta = \alpha + B\eta + \Gamma\xi + \zeta
\]
\[
y = \tau_y + \Lambda_y\eta + \epsilon
\]
\[
x = \tau_x + \Lambda_x\xi + \delta
\]
The table below is provided as a quick reference to the numerous matrices in LISREL models. Note that NX is the number of manifest exogenous (independent) variables, the number of Xs. NY is the number of manifest endogenous (dependent) variables, the number of Ys. NK is the number of latent exogenous variables, the number of Ksis or Xis. NE is the number of latent endogenous variables, the number of etas.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Word</th>
<th>Abbreviation</th>
<th>Dimensions</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda_x$</td>
<td>Lambda x</td>
<td>LX</td>
<td>NX x NK</td>
<td></td>
<td>Exogenous Factor Loading Matrix</td>
</tr>
<tr>
<td>$\Lambda_y$</td>
<td>Lambda y</td>
<td>LY</td>
<td>NY x NE</td>
<td></td>
<td>Endogenous Factor Loading Matrix</td>
</tr>
<tr>
<td>$B$</td>
<td>Beta</td>
<td>BE</td>
<td>NE x NE</td>
<td></td>
<td>Regressions of Latent Endogenous Variables Predicting Endogenous Variables</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>Gamma</td>
<td>GA</td>
<td>NE x NK</td>
<td></td>
<td>Regressions of Latent Exogenous Variables Predicting Endogenous Variables</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Phi</td>
<td>PH</td>
<td>NK x NK</td>
<td>$\text{cov}(\xi)$</td>
<td>Covariance Matrix of Latent Exogenous Variables</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>Psi</td>
<td>PS</td>
<td>NE x NE</td>
<td>$\text{cov}(\zeta)$</td>
<td>Residual Covariance Matrix of Latent Endogenous Variables</td>
</tr>
<tr>
<td>$\Theta_\delta$</td>
<td>Theta delta</td>
<td>TD</td>
<td>NX x NX</td>
<td>$\text{cov}(\delta)$</td>
<td>Residual Covariance Matrix of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>$\Theta_\epsilon$</td>
<td>Theta epsilon</td>
<td>TE</td>
<td>NY x NY</td>
<td>$\text{cov}(\epsilon)$</td>
<td>Residual Covariance Matrix of Manifest Endogenous Variables</td>
</tr>
<tr>
<td>$\Theta_{\delta\epsilon}$</td>
<td>Theta delta epsilon</td>
<td>TH</td>
<td>NX x NY</td>
<td>$\text{cov}(\delta, \epsilon)$</td>
<td>Residual Covariance Matrix of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>$\tau_x$</td>
<td>tau x</td>
<td>TX</td>
<td>NX x 1</td>
<td></td>
<td>Residual Means of Manifest Exogenous Variables</td>
</tr>
<tr>
<td>$\tau_y$</td>
<td>tau y</td>
<td>TY</td>
<td>NY x 1</td>
<td></td>
<td>Residual Means of Manifest Endogenous Variables</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>kappa</td>
<td>KA</td>
<td>NK x 1</td>
<td>$\text{mean}(\xi)$</td>
<td>Means of Latent Exogenous Variables</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>alpha</td>
<td>AL</td>
<td>NE x 1</td>
<td></td>
<td>Residual Means of Latent Endogenous Variables</td>
</tr>
</tbody>
</table>

From the extended LISREL model, several submodels can be defined. Subtypes of the LISREL model are defined by setting some of the arguments of the LISREL objective to NA. Note that because the default values of each LISREL matrix is NA, setting a matrix to NA can be accomplished by simply not giving it any other value.

The first submodel is the LISREL model without means.

$$\eta = B\eta + \Gamma\xi + \zeta$$

$$y = \Lambda_y\eta + \epsilon$$

$$x = \Lambda_x\xi + \delta$$

The LISREL model without means requires 9 matrices: LX, LY, BE, GA, PH, PS, TD, TE, and TH. Hence this LISREL model has TX, TY, KA, and AL as NA. This can be accomplished by leaving these matrices at their default values.

The TX, TY, KA, and AL matrices 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 TX, TY, KA, and AL matrices are ignored and the model without means is estimated.

A second submodel involves only endogenous variables.

$$\eta = B\eta + \zeta$$

$$y = \Lambda_y\eta + \epsilon$$

The endogenous-only LISREL model requires 4 matrices: LY, BE, PS, and TE. The LX, GA, PH, TD, and TH must be NA in this case. However, means can also be specified, allowing TY and AL if the data are raw or if observed means are provided.
Another submodel involves only exogenous variables.

\[ x = \Lambda \xi + \delta \]

The exogenous-model model requires 3 matrices: LX, PH, and TD. The LY, BE, GA, PS, TE, and TH matrices must be NA. However, means can also be specified, allowing TX and KA if the data are raw or if observed means are provided.

The model that is run depends on the matrices that are not NA. If all 9 matrices are not NA, then the full model is run. If only the 4 endogenous matrices are not NA, then the endogenous-only model is run. If only the 3 exogenous matrices are not NA, then the exogenous-only model is run. If some endogenous and exogenous matrices are not NA, but not all of them, then appropriate errors are thrown. Means are included in the model whenever their matrices are provided.

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

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

To evaluate, place MxLISRELObjective objects, the mxData object for which the expected covariance approximates, referenced MxAlgebra and MxMatrix objects, and optional MxBound 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 MxLISRELObjective object. MxLISRELObjective 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
###---------------------------###
### Factor Model
mLX <- mxMatrix("Full", values=c(.5, .6, .8, rep(0, 6), .4, .7, .5),
    name="LX", nrow=6, ncol=2,
    free=c(TRUE,TRUE,TRUE,rep(FALSE, 6),TRUE,TRUE,TRUE))

mTD <- mxMatrix("Diag", values=c(rep(.2, 6)), name="TD", nrow=6, ncol=6,
    values=c(1, 1, 1, 1, 1, 1))

mLISREL <- mxModel("LISREL", mLX, mTD)
```
MxListOrNull-class

An optional list

Description

An optional list

mxMakeNames

mxMakeNames

Description

Adjust a character vector so that it can be used as MxMatrix column or row names. OpenMx is (much) more restrictive than base R's make.names.

Usage

mxMakeNames(names, unique = FALSE)

Arguments

names a character vector

unique whether the pass the result through make.unique

See Also

make.names

Examples

demo <- c("", "103", "data", "foo.bar[3,2]", "+!", "!+")
mxMakeNames(demo, unique=TRUE)
Create MxMatrix Object

This function creates a new MxMatrix object.

Usage

```
mxMatrix(type = "Full", nrow = NA, ncol = NA,
    free = FALSE, values = NA, labels = NA, lbounds = NA,
    ubounds = NA, byrow = getOption("mxByRow"), dimnames = NA, name = NA,
    condenseSlots = getOption("mxCondenseMatrixSlots"))
```

Arguments

type  A character string indicating the matrix type, where type indicates the range
    of values and equalities in the matrix. Must be one of: ‘Diag’, ‘Full’, ‘Iden’,

nrow  Integer; the desired number of rows. One or both of ‘nrow’ and ‘ncol’ is re-
    quired when ‘values’, ‘free’, ‘labels’, ‘lbounds’, and ‘ubounds’ arguments are not
    matrices, depending on the desired MxMatrix type.

ncol  Integer; the desired number of columns. One or both of ‘nrow’ and ‘ncol’ is
    required when ‘values’, ‘free’, ‘labels’, ‘lbounds’, and ‘ubounds’ arguments are
    not matrices, depending on the desired MxMatrix 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.

lbounds  A vector or matrix of numeric lower bounds. Default bounds are specified with
    an NA.

ubounds  A vector or matrix of numeric upper bounds. Default bounds are specified with
    an NA.

byrow  Logical; defaults to value of global option ‘mxByRow’. If FALSE (default),
    the ‘values’, ‘free’, ‘labels’, ‘lbounds’, and ‘ubounds’ 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.
condenseSlots Logical; defaults to value of global option 'mxByRow'. If TRUE, then the resulting MxMatrix will "condense" its 'labels', 'free', 'lbound', and 'ubound' down to 1x1 matrices if they contain only FALSE (‘free’) or NA (the other three). If FALSE, those four matrices and the ‘values’ matrix will all be of equal dimensions.

Details

The mxMatrix function creates MxMatrix objects, which consist of five 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.

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 principal 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 main 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 the diagonal.
- ‘Symm’ matrices are symmetric matrices (see ‘Symm’), with 1’s along the main diagonal.
- ‘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 ‘Lower’ or ‘Symm’, then the arguments to ‘free’, ‘values’, ‘labels’, ‘lbound’, or ‘ubound’ may be vectors of length N * (N + 1)/2, where N is the number of rows and columns of the matrix. When ‘type’ is ‘Sdiag’ or ‘Stand’, then the arguments to ‘free’, ‘values’, ‘labels’, ‘lbound’, or ‘ubound’ may be vectors of length N * (N - 1)/2.

Value

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

References

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

Description

MxMatrix is a virtual S4 class that comprises the nine types of matrix objects used by OpenMx (see \texttt{mxMatrix()} for details). An MxMatrix object is a named entity. New instances of this class can be created using the function \texttt{mxMatrix()}. MxMatrix objects may be used as arguments in other functions from the OpenMx package, including \texttt{mxAlgebra()}, \texttt{mxConstraint()}, and \texttt{mxModel()}. 

Objects from the Class

All nine types of object that the class comprises can be created via \texttt{mxMatrix()}. 

Slots

name: Character string; the name of the MxMatrix object. Note that this is the object’s "Mx name" (so to speak), which identifies it in OpenMx’s internal namespace, rather than the symbol identifying it in R’s workspace. Use of MxMatrix objects in an \texttt{mxAlgebra} or \texttt{mxConstraint} function requires reference by name.
values: Numeric matrix of 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.

labels: Matrix of character strings which provides 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.

free: Logical matrix specifying whether each element is free versus fixed. An element 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 a fitfunction when that MxMatrix object is included in an MxModel object and evaluated using the `mxRun()` function.

lbound: Numeric matrix of lower bounds on free parameters.

ubound: Numeric matrix of upper bounds on free parameters.

.sqbrackets: Logical matrix; used internally by OpenMx. Identifies which elements have labels with square brackets in them.

.persist: Logical; used internally by OpenMx. Governs how `mxRun()` handles the MxMatrix object when it is inside the MxModel being run.

.condenseSlots: Logical; used internally by OpenMx. If FALSE, then the matrices in the 'values', 'labels', 'free', 'lbound', and 'ubound' slots are all of equal dimensions. If TRUE, then the last four of those slots will "condense" a matrix consisting entirely of FALSE or NA down to 1x1.

display: Character string; used internally by OpenMx when parsing MxAlgebras.

dependencies: Integer; used internally by OpenMx when parsing MxAlgebras.

Methods

$ signature(x = "MxMatrix"):

$<- signature(x = "MxMatrix"):

[ signature(x = "MxMatrix"):

<- signature(x = "MxMatrix"):

dim signature(x = "MxMatrix"):

dimnames signature(x = "MxMatrix"):

dimnames<- signature(x = "MxMatrix"):

length signature(x = "MxMatrix"):

names signature(x = "MxMatrix"):

ncol signature(x = "MxMatrix"):

nrow signature(x = "MxMatrix"):

print signature(x = "MxMatrix"):

show signature(object = "MxMatrix"):

Note that some methods are documented separately (see below, under "See Also").
References

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

See Also

mxMatrix() for creating MxMatrix objects. Note that functions imxCreateMatrix(), imxDeparse(), imxSquareMatrix(), imxSymmetricMatrix(), and imxVerifyMatrix() are separately documented methods for this class. More information about the OpenMx package may be found here.

Examples

showClass("MxMatrix")

mxMI

Estimate Modification Indices for MxModel Objects

Description

This function estimates the change in fit function value resulting from freeing currently fixed parameters.

Usage

mxMI(model, matrices=NA, full=TRUE)

Arguments

model An MxModel for which modification indices are desired.
matrices Character vector. The names of the matrices in which to search for modification
full Logical. Whether or not to return the full modification index in addition to the restricted.

Details

Modification indices provide an estimate of how much the fit function value would change if a parameter that is currently fixed was instead freely estimated. There are two versions of this estimate: a restricted version and an full version. The restricted version is reported as the MI and is much faster to compute. The full version is reported as MI.Full. The full version accounts for the total change in fit function value resulting from the newly freed parameter. The restricted version only accounts for the change in the fit function due to the movement of the new free parameter. In particular, the restricted version does not account for the change in fit function value due to the other free parameters moving in response to the new parameter.

The algorithm respects fixed parameter labels. That is, when a fixed parameter has a label and occurs in more than one spot, then that fixed parameter is freed in all locations in which it occurs to evaluate the modification index for that fixed parameter.
When the fit function is in minus two log likelihood units (e.g. `mxFitFunctionML`), then the MI will be approximately chi squared distributed with 1 degree of freedom. Using a p-value of 0.01 has been suggested. Hence, a MI greater than `qchisq(p=0.01, df=1)`, or 6.63, is suggestive of a modification.

Users should be cautious in their use of modification indices. If a model was created with the aid of MIs, then it should always be reported. Do not pretend that you have a theoretical reason for part of a model that was put there because it was suggested by a modification index. This is fraud.

When using modification indices there are two options for best practices. First, you can report the analyses as exploratory. Document all the explorations that you did, and know that your results may or may not generalize. Second, you can use cross-validation. Reserve part of your data for exploration, and use the remaining data to test if the exploratory model generalizes to new data.

**Value**

A named list with components

- **MI** The restricted modification index.
- **MI.Full** The full modification index.
- **plusOneParamModels** A list of models with one additional free parameter

**References**


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

**Examples**

```r
# Create a model
require(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
lats <- c("G")
factorModel <- mxModel("One Factor",
  type="RAM",
  manifestVars = manifests,
  latentVars = lats,
  mxPath(from=lats, to=manifests),
  mxPath(from=manifests, arrows=2),
  mxPath(from=lats, arrows=2,
    free=FALSE, values=1.0),
  mxPath(from = 'one', to = manifests),
  mxData(demoOneFactor[1:200,], type="raw")) #Not full data
#No SEs for speed
factorModel <- mxOption(factorModel, 'Standard Errors', 'No')
factorRun <- mxRun(factorModel)

# See if it should be modified
# Notes
# Using full=FALSE for faster performance
# Using matrices='A' and 'S' to not get MIs for
```
mxMLObjective

Description

WARNING: Objective functions have been deprecated as of OpenMx 2.0. Please use mxExpectationNormal() and mxFitFunctionML() instead. As a temporary workaround, mxMLObjective returns a list containing an MxExpectationNormal object and an MxFitFunctionML object.

mxMLObjective(covariance, means = NA, dimnames = NA, thresholds = NA) All occurrences of mxMLObjective(covariance, means = NA, dimnames = NA, thresholds = NA) Should be changed to

mxExpectationNormal(covariance, means = NA, dimnames = NA, thresholds = NA, threshnames = dimnames) mxFitFunctionML(vector = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>covariance</td>
<td>A character string indicating the name of the expected covariance algebra.</td>
</tr>
<tr>
<td>means</td>
<td>An optional character string indicating the name of the expected means algebra.</td>
</tr>
<tr>
<td>dimnames</td>
<td>An optional character vector to be assigned to the dimnames of the covariance and means algebras.</td>
</tr>
<tr>
<td>thresholds</td>
<td>An optional character string indicating the name of the thresholds matrix.</td>
</tr>
</tbody>
</table>

Details

NOTE: THIS DESCRIPTION IS DEPRECATED. Please change to using mxExpectationNormal and mxFitFunctionML as shown in the example below.

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

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

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

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

Value

Returns a list containing an MxExpectationNormal object and an MxFitFunctionML object.

References

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

Examples

# Create and fit a model using mxMatrix, mxAlgebra, mxExpectationNormal, and mxFitFunctionML
library(OpenMx)

# Simulate some data
x=rnorm(1000, mean=0, sd=1)
y=0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices
S <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1,0,0,1),
free=c(TRUE,FALSE,FALSE,TRUE), labels=c("Vx", NA, NA, "Vy"), name = "S")
A <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,1,0,0),
free=c(FALSE,TRUE,FALSE,FALSE), labels=c(NA, "b", NA, NA), name = "A")
I <- mxMatrix(type="Iden", nrow=2, ncol=2, name="I")

# Define the expectation
expCov <- mxAlgebra(solve(I-A) %*% S %*% t(solve(I-A)), name="expCov")
expFunction <- mxEval(expCov, dimnames=tmpNames)

# Choose a fit function
fitFunction <- mxFitFunctionML()

# Define the model

tmpModel <- mxModel(model="exampleModel", S, A, I, expCov, expFunction, fitFunction,
  mxD ata (observed= cov (tmpFrame), type="cov", numObs= dim (tmpFrame)[1]))

# Fit the model and print a summary

tmpModelOut <- mxRun (tmpModel)
summary(tmpModelOut)

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 Mx-
  Model object, then all elements of that model are placed in the resulting Mx-
  Model object. If 'model' is a string, then a new model is created with the string
  as its name. If 'model' is either unspecified or 'model' is a named entity, data
  source, or MxPath object, then a new model is created.

... An arbitrary number of mxMatrix, mxPath, mxData, and other functions such
  as mxConstraints and mxCI. These will all be added or removed from the model
  as specified in the 'model' argument, based on the 'remove' argument.

manifestVars For RAM-type models, A list of manifest variables to be included in the model.

latentVars For RAM-type models, A list of latent variables to be included in the model.

remove logical. If TRUE, elements listed in this statement are removed from the original
  model. If FALSE, elements listed in this statement are added to the original
  model.

independent logical. If TRUE then the model is evaluated independently of other models.

type character vector. The model type to assign to this model. Defaults to op-
  tions("mxDefaultType"). See below for valid types

name An optional character vector indicating the name of the object.
Details

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

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

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

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

The model type is determined by a character vector supplied to the 'type' argument. The type of a model is a dynamic property, 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 the $ symbol. For instance, for an MxModel named "yourModel" containing an MxMatrix named "yourMatrix", the contents of "yourMatrix" can be accessed as yourModel$yourMatrix. Slots (i.e., matrices, algebras, etc.) in an MxMatrix may also be referenced with the $ symbol (e.g., yourModel$matrices or yourModel$algebras). See the documentation for Classes and the examples in Classes for more information.

Value

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

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

See Also

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

Examples

library(OpenMx)

# At the simplest, you can create an empty model,
# placing it in an object, and add to it later
emptyModel <- mxModel(model="IAMEmpty")

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

# Update 'firstdraft', and rename the model 'finaldraft'
finalModel <- mxModel(model=firstModel,
  mxMatrix(type='Symm', nrow = 3, ncol = 3, name = "S"),
  mxMatrix(type='Iden', nrow = 3, name = "F"),
  name = "finaldraft")

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

# Two ways to view the matrix named "A" in MxModel object 'model'
finalModel@A
finalModel$matrices@A

# A working example using OpenMx Path Syntax
data(HS.ability.data) # load the data

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

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

HSModel <- mxModel(model="Holzinger_and_Swineford_1939", type="RAM",
manifestVars = manifests, # list the measured variables (boxes)
latentVars = latents, # list the latent variables (circles)
# factor loadings from latents to manifests
mxPath(from="vis", to= Spatial), # factor loadings
mxPath(from="math", to= Math), # factor loadings
mxPath(from="text", to= Verbal), # factor loadings

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

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

fitModel <- mxRun(HSmodel) # run the model
summary(fitModel) # examine the output: Fit statistics and path loadings

MxModel-class

MxModel Class

Description

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

Details

The MxModel class has the following slots:

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

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

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

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

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

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

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

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

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

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

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

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

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

Named entities in MxModel objects may be viewed and referenced by name using the $ symbol. For instance, for an MxModel named "yourModel" containing an MxMatrix named "yourMatrix", the contents of "yourMatrix" can be accessed as yourModel$yourMatrix. Slots (i.e., matrices, algebras, etc.) in an mxMatrix may also be referenced with the $ symbol (e.g., yourModel$matrices or yourModel$algebras). See the documentation for Classes and the examples in mxModel for more information.

References

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

See Also

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

mxOption

Set or Clear an Optimizer Option

Description

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

Usage

mxOption(model, key, value, reset = FALSE)
**Arguments**

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

**Details**

`mxOption` is used to set, clear, or query an option (given in the ‘key’ argument) in the back-end optimizer. Valid option keys are listed below.

Use value = NULL to remove an existing option. Leaving value blank will return the current value of the option specified by ‘key’.

To reset all options to their default values, use ‘reset = TRUE’. When reset = TRUE, ‘key’ and ‘value’ are ignored.

If the ‘model’ argument is set to NULL, the default optimizer option (i.e. those applying to all models by default) will be set.

To see the defaults, use `getOption('mxOptions')`.

Before the model is submitted to the back-end, all keys and values are converted into strings using the `as.character` function.

The maximum number of major iterations (the option “Major iterations”) for optimization for NPSOL can be specified either by using a numeric value (such as 50, 1000, etc) or by specifying a user-defined function. The user-defined function should accept two arguments as input, the number of parameters and the number of constraints, and return a numeric value as output.

**OpenMx options**

- **Number of Threads**: \(i\) the number of processor cores to use. Use `dectectCores()` to find how many are available.
- **Calculate Hessian**: [Yes | No] calculate the Hessian explicitly after optimization.
- **Standard Errors**: [Yes | No] return standard error estimates from the explicitly calculate hessian.
- **Default optimizer**: [NPSOL | SLSQP] the gradient descent optimizer to use.
- **Number of Threads**: [0|1|2|...|10|...] number of threads used for optimization. This is how parallelism works. Default value of 0 uses `detectCores()`.
- **Feasibility tolerance**: \(r\) the maximum acceptable absolute violations in linear and nonlinear constraints.
- **Optimality tolerance**: \(r\) the maximum acceptable difference in fit.
- **Gradient algorithm**: see list finite difference method, either ‘forward’ or ‘central’.
- **Gradient iterations**: 1:4 the number of Richardson extrapolation iterations

**SLSQP** is our new, experimental optimizer.

**NPSOL-specific options**

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

Infinite bound size \( r \) if \( r > 0 \) defines the "infinite" bound `bigbnd`.

Major iterations \( i \) or a function the maximum number of major iterations before termination.

Verify level [\(-1\):\(3 \mid \text{Yes} \mid \text{No} \)] see NPSOL manual.

Line search tolerance \( r \) controls the accuracy with which a step is taken.

Derivative level [0-3] see NPSOL manual.

Hessian [\( \text{Yes} \mid \text{No} \)] return the Hessian (Yes) or the transformed Hessian (No).

Checkpointing options

- **Always Checkpoint** [\(\text{Yes} \mid \text{No} \)] whether to checkpoint all models during optimization.
- **Checkpoint Directory** \( \text{path} \) the directory into which checkpoint files are written.
- **Checkpoint Prefix** \( \text{string} \) the string prefix to add to all checkpoint filenames.
- **Checkpoint Fullpath** \( \text{path} \) overrides the directory and prefix (useful to output to /dev/fd/2).
- **Checkpoint Units** see list the type of units for checkpointing: 'minutes', 'iterations', or 'evaluations'.
- **Checkpoint Count** \( i \) the number of units between checkpoint intervals.

Model transformation options

- **Error Checking** [\(\text{Yes} \mid \text{No} \)] whether model consistency checks are performed in the OpenMx front-end.
- **No Sort Data** character vector of model names for which FIML data sorting is not performed.
- **RAM Inverse Optimization** [\(\text{Yes} \mid \text{No} \)] whether to enable \( \text{solve}(I - A) \) optimization.
- **RAM Max Depth** \( i \) the maximum depth to be used when \( \text{solve}(I - A) \) optimization is enabled.

Multivariate normal integration parameters

- **mvnMaxPointsA** \( i \) base number of integration points
- **mvnMaxPointsB** \( i \) number of integration points per row
- **mvnMaxPointsC** \( i \) number of integration points per rows^2
- **mvnAbsEps** \( i \) absolute tolerance
- **mvnRelEps** \( i \) relative tolerance

Value

If a model is provided, it is returned with the optimizer option either set or cleared. If value is empty, the current value is returned.

References

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

See Also

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

```r
# set the Number of Threads (cores to use)
mxOption(NULL, "Number of Threads", detectCores() - 1)

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

testModel <- mxOption(testModel, "Function precision", 1e-5) # set precision
testModel <- mxOption(testModel, "Function precision", NULL) # clear precision
# N.B. This is model-specific precision (defaults to global setting)

# may optimize for speed
# at cost of not getting standard errors
testModel <- mxOption(testModel, "Calculate Hessian", "No")
testModel <- mxOption(testModel, "Standard Errors", "No")

testModel$options # see the list of options you set
```

---

**MxOptionalChar-class**  
*An optional character*

**Description**

An optional character

---

**MxOptionalCharOrNumber-class**  
*A character, integer, or NULL*

**Description**

A character, integer, or NULL

---

**MxOptionalLogical-class**  
*An optional logical*

**Description**

This is an internal class, the union of NULL and logical.
MxOptionalMatrix-class

An optional matrix

Description
An optional matrix

MxOptionalNumeric-class

An optional numeric

Description
An optional numeric

mxPath

Create List of Paths

Description
This function creates a list of paths.

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>character vector. These are the sources of the new paths.</td>
</tr>
<tr>
<td>to</td>
<td>character vector. These are the sinks of the new paths.</td>
</tr>
<tr>
<td>connect</td>
<td>String. Specifies the type of source to sink connection: &quot;single&quot;, &quot;all.pairs&quot;, &quot;all.bivariate&quot;, &quot;unique.pairs&quot;, &quot;unique.bivariate&quot;. Default value is &quot;single&quot;.</td>
</tr>
<tr>
<td>arrows</td>
<td>numeric value. Must be either 1 (for single-headed) or 2 (for double-headed arrows).</td>
</tr>
<tr>
<td>free</td>
<td>boolean vector. Indicates whether paths are free or fixed.</td>
</tr>
<tr>
<td>values</td>
<td>numeric vector. The starting values of the parameters.</td>
</tr>
<tr>
<td>labels</td>
<td>character vector. The names of the paths.</td>
</tr>
<tr>
<td>lbound</td>
<td>numeric vector. The lower bounds of free parameters.</td>
</tr>
<tr>
<td>ubound</td>
<td>numeric vector. The upper bounds of free parameters.</td>
</tr>
<tr>
<td>...</td>
<td>Not used. Allows OpenMx to catch the use of the deprecated ‘all’ argument.</td>
</tr>
</tbody>
</table>
Details

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

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

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

"all.pairs": (a,a), (a,b), (a,c), (b,a), (b,b), (b,c), (c,a), (c,b), (c,c).
"unique.pairs": (a,a), (a,b), (a,c), (b,b), (b,c), (c,c).
"all.bivariate": (a,b), (a,c), (b,a), (b,c), (c,a), (c,b).
"unique.bivariate": (a,b), (a,c), (b,c).
"single": (a,a), (b,b), (c,c).

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

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

The ‘arrows’ argument specifies the type of paths created. A value of 1 indicates a one-headed arrow representing regression. This path represents a regression of the ‘to’ variable on the ‘from’ variable, such that the arrow points to the ‘to’ variable in a path diagram. A value of 2 indicates a two-headed arrow, representing a covariance or variance. If multiple paths are created in the same mxPath function, then the ‘arrows’ argument may take a vector of 1s and 2s to be applied to the set of created paths.

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

Value

Returns a list of paths.

Note

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

References


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

See Also

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

Examples

```r
# A simple Example: 1 factor Confirmatory Factor Analysis

library(OpenMx)

data(demoOneFactor)

manifests <- names(demoOneFactor)

latents <- c("G")

factorModel <- mxModel(model="One Factor", type="RAM",
                       manifestVars = manifests,
                       latentVars = latents,
                       mxPath(from=latents, to=manifests),
                       mxPath(from=manifests, arrows=2),
                       mxPath(from=latents, arrows=2,free=FALSE, values=1.0),
                       mxData(cov(demoOneFactor), type="cov",numObs=500))

factorFit <- mxRun(factorModel)

summary(factorFit)

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

# list of 100 variable names: "01" "02" "03"
myManifest <- sprintf("%02d", c(1:100))

# the latent variables for the model
myLatent <- c("G1", "G2", "G3", "G4", "G5")

# Start building the model:
# Define its type, and add the manifest and latent variable name lists

testModel <- mxModel(model="testModel", type = "RAM",
                     manifestVars = myManifest, latentVars = myLatent)

# Create covariances between the latent variables and add to the model
# Here we use combn to create the covariances
# nb: To create the variances and covariances in one operation you could use
# expand.grid(myLatent,myLatent) to specify from and to
```
uniquePairs <- combn(myLatent, 2)
covariances <- mxPath(from = uniquePairs[,1],
                      to = uniquePairs[,2], arrows = 2, free = TRUE, values = 1)
testModel <- mxModel(model = testModel, covariances)

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

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

# add single-headed paths to the model
testModel <- mxModel(model = testModel, singles)
**mxRAMObjective**

**DEPRECATED: Create MxRAMObjective Object**

### Description

**WARNING:** Objective functions have been deprecated as of OpenMx 2.0.

Please use `mxExpectationRAM()` and `mxFitFunctionML()` instead. As a temporary workaround, `mxRAMObjective` returns a list containing an `MxExpectationNormal` object and an `MxFitFunctionML` object.

All occurrences of

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

Should be changed to

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

### 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 vector to be assigned to the column names of the thresholds matrix.

- **vector**
  - A logical value indicating whether the objective function result is the likelihood vector.

- **threshnames**
  - An optional character vector to be assigned to the column names of the thresholds matrix.

### Details

**NOTE:** THIS DESCRIPTION IS DEPRECATED. Please change to using `mxExpectationRAM` and `mxFitFunctionML` as shown in the example below.

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

The 'dimnames' arguments takes an optional character vector. If this argument is not a single NA, then this vector be assigned to be the column names of the 'F' matrix and optionally to the 'M' matrix, if the 'M' matrix exists.
The 'A' argument refers to the A or asymmetric matrix in the RAM approach. This matrix consists of all of the asymmetric paths (one-headed arrows) in the model. A free parameter in any row and column describes a regression of the variable represented by that row regressed on the variable represented in that column.

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

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

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

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

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

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 list containing an MxExpectationRAM object and an MxFitFunctionML object.

References


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

Examples

```r
# Create and fit a model using mxMatrix, mxAlgebra,
```
# mxExpectationNormal, and mxFitFunctionML

library(OpenMx)

# Simulate some data

x = rnorm(1000, mean=0, sd=1)
y = 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

# Define the matrices

matrixS <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(1,0,0,1),
                   free=c(TRUE,FALSE,FALSE,TRUE), labels=c("Vx", NA, NA, "Vy"),
                   name = "S")
matrixA <- mxMatrix(type = "Full", nrow = 2, ncol = 2, values=c(0,1,0,0),
                   free=c(FALSE,TRUE,FALSE,FALSE), labels=c(NA, "b", NA, NA),
                   name = "A")
matrixF <- mxMatrix(type="Iden", nrow=2, ncol=2, name="F")
matrixM <- mxMatrix(type = "Full", nrow = 1, ncol = 2, values=c(0,0),
                   free=c(TRUE,TRUE), labels=c("Mx", "My"), name = "M")

# Define the expectation

expFunction <- mxExpectationRAM(M="M", dimnames = tmpNames)

# Choose a fit function

fitFunction <- mxFitFunctionML()

# Define the model

tmpModel <- mxModel(model="exampleRAMModel",
                   matrixA, matrixS, matrixF, matrixM,
                   expFunction, fitFunction,
                   mxData(observed=tmpFrame, type="raw"))

# Fit the model and print a summary

tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

mxRename

---

**Description**

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

mxRename(model, newname, oldname = NA)

Arguments

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

Value

Return a mxModel object with the target model renamed.

References

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

Examples

```r
library(OpenMx)

# create two empty models
modelA <- mxModel(model='modelA')
modelB <- mxModel(model='modelB')

# create a parent model with two submodels
modelC <- mxModel(model='modelC', modelA, modelB)

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

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

model1
```

mxRestore

Description

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

Usage

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

Arguments

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

Details

In general, the arguments ‘chkpt.directory’ and ‘chkpt.prefix’ should be identical to the mxOption: ‘Checkpoint Directory’ and ‘Checkpoint Prefix’ that were specified on the model before execution.
Alternatively, the checkpoint file can be manually loaded as a data.frame in R. Use read.table with the options header=TRUE, sep="\t", stringsAsFactors=FALSE, check.names=FALSE.

Value

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

References

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

Examples

library(OpenMx)

# Simulate some data

x=rnorm(1000, mean=0, sd=1)
y= 0.5*x + rnorm(1000, mean=0, sd=1)
tmpFrame <- data.frame(x, y)
tmpNames <- names(tmpFrame)

data <- mxData(cov(tmpFrame), type="cov", numObs = 1000)
expCov <- mxMatrix(type="Symm", nrow=2, ncol=2, values=c(.2,.1,.2), free=TRUE, name="expCov")
expFunction <- mxEvaluationNormal(covariance="expCov", dimnames=tmpNames)
fitFunction <- mxFitFunctionML()
testModel <- mxModel(model="testModel", expCov, data, expFunction, fitFunction)

#Use mxRun to optimize the free parameters in the expected covariance matrix
modelOut <- mxRun(testModel, checkpoint = TRUE)
modelOut$expCov

#Use mxRestore to load the last checkpoint saved state of the model
modelRestore <- mxRestore(testModel)
modelRestore$expCov
DEPRECATED: Create MxRObjective Object

Description

WARNING: Objective functions have been deprecated as of OpenMx 2.0.
Please use mxFitFunctionR() instead. As a temporary workaround, mxRObjective returns a list containing a NULL MxExpectation object and an MxFitFunctionR object.

All occurrences of
mxRObjective(-fitfun, ...)
Should be changed to
mxFitFunctionR(fitfun, ...)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objfun</td>
<td>A function that accepts two arguments.</td>
</tr>
<tr>
<td>...</td>
<td>The initial state information to the objective function.</td>
</tr>
</tbody>
</table>

Details

NOTE: THIS DESCRIPTION IS DEPRECATED. Please change to using mxExpectationNormal and mxFitFunctionML as shown in the example below.

The fitfun 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.

Throwing an exception (via stop) from inside fitfun may result in unpredictable behavior. You may want to wrap your code in tryCatch while experimenting.

Value

Returns a list containing a NULL mxExpectation object and an MxFitFunctionR object.

References

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

# Create and fit a model using mxFitFunctionR

library(OpenMx)

A <- mxMatrix(nrow = 2, ncol = 2, values = c(1:4), free = TRUE, name = 'A')
squared <- function(x) { x ^ 2 }

# Define the objective function in R

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))
}

# Define the expectation function

fitFunction <- mxFitFunctionR(objFunction)

# Define the model

tmpModel <- mxModel(model="exampleModel", A, fitFunction)

# Fit the model and print a summary

tmpModelOut <- mxRun(tmpModel)
summary(tmpModelOut)

---

mxRowObjective

DEPRECATED: Create MxRowObjective Object

Description

WARNING: Objective functions have been deprecated as of OpenMx 2.0.

Please use mxFitFunctionRow() instead. As a temporary workaround, mxRowObjective returns a list containing a NULL MxExpectation object and an MxFitFunctionRow object.

All occurrences of

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

Should be changed to

mxFitFunctionRow(rowAlgebra, reduceAlgebra, dimnames, rowResults = "rowResults", filteredDataRow = "filteredDataRow", existenceVector = "existenceVector")
Arguments

rowAlgebra A character string indicating the name of the algebra to be evaluated row-wise.
reduceAlgebra A character string indicating the name of the algebra that collapses the row results into a single number which is then optimized.
dimnames A character vector of names corresponding to columns be extracted from the data set.
rowResults The name of the auto-generated "rowResults" matrix. See details.
filteredDataRow The name of the auto-generated "filteredDataRow" matrix. See details.
existenceVector The name of the auto-generated "existenceVector" matrix. See details.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxRowObjective function evaluates a user-defined MxAlgebra object called the 'rowAlgebra' in a row-wise fashion. It then stores results of the row-wise evaluation in another MxAlgebra object called the 'rowResults'. Finally, the mxRowObjective function collapses the row results into a single number which is then used for optimization. The MxAlgebra object named by the 'reduceAlgebra' collapses the row results into a single number.
The ‘filteredDataRow’ is populated in a row-by-row fashion with all the non-missing data from the current row. You cannot assume that the length of the filteredDataRow matrix remains constant (unless you have no missing data). The ‘existenceVector’ is populated in a row-by-row fashion with a value of 1.0 in column j if a non-missing value is present in the data set in column j, and a value of 0.0 otherwise. Use the functions omxSelectRows, omxSelectCols, and omxSelectRowsAndCols to shrink other matrices so that their dimensions will be conformable to the size of ‘filteredDataRow’.

Value

Please use mxFitFunctionRow() instead. As a temporary workaround, mxRowObjective returns a list containing a NULL MxExpectation object and an MxFitFunctionRow object.

References

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

Examples

# Model that adds two data columns row-wise, then sums that column
# Notice no optimization is performed here.

library(OpenMx)

xdat <- data.frame(a=rnorm(10), b=1:10) # Make data set
amod <- mxModel(model="example1",
    mxData(observed=xdat, type='raw'),
    mxAlgebra(sum(filteredDataRow), name = 'rowAlgebra'),
    mxAlgebra(sum(rowResults), name = 'reduceAlgebra'),
    mxRowObjective(rowAlgebra = "rowAlgebra", reduceAlgebra = "reduceAlgebra", dimnames = dimnames(xdat), rowResults = "rowResults", filteredDataRow = "filteredDataRow", existenceVector = "existenceVector")
    )
mxRun

mxFitFunctionRow(
    rowAlgebra='rowAlgebra',
    reduceAlgebra='reduceAlgebra',
    dimnames=c('a','b'))
)

amodOut <- mxRun(amod)
mxEval(rowResults, model=amodOut)
mxEval(reduceAlgebra, model=amodOut)

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

bmod <- mxModel(model="example2",
    mxData(observed=xdat, type='raw'),
    mxMatrix(values=.75, ncol=1, nrow=1, free=TRUE, name='B'),
    mxAlgebra((filteredDataRow - B) ^ 2, name='rowAlgebra'),
    mxAlgebra(sum(rowResults), name='reduceAlgebra'),
    mxFitFunctionRow(
        rowAlgebra='rowAlgebra',
        reduceAlgebra='reduceAlgebra',
        dimnames=c('a'))
)

bmodOut <- mxRun(bmod)
mxEval(B, model=bmodOut)
mxEval(reduceAlgebra, model=bmodOut)
mxEval(rowResults, model=bmodOut)

---

mxRun **Send a Model to the Optimizer**

**Description**

This function begins optimization on the top-level model.

**Usage**

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

**Arguments**

- `model` A `MxModel` object to be optimized.
- `...` Not used. Forces remaining arguments to be specified by name.
- `intervals` A boolean indicating whether to compute the specified confidence intervals.
- `silent` A boolean indicating whether to print status to terminal.
- `suppressWarnings` A boolean indicating whether to suppress warnings.
mxRun

unsafe  A boolean indicating whether to ignore errors.
checkpoint  A boolean indicating whether to periodically write parameter values to a file.
useSocket  A boolean indicating whether to periodically write parameter values to a socket.
onlyFrontend  A boolean indicating whether to run only front-end model transformations.
useOptimizer  A boolean indicating whether to run only the log-likelihood of the current free parameter values but not move any of the free parameters.

Details

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

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

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

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

Free parameters are estimated or updated based on the expectation and fit functions. 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 expectation and fit functions in both submodels, as well as an additional fit function describing how the results of the first two should be combined.

Value

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

References

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

Examples

# Create and run the 1-factor CFA on the openmx.psyc.virginia.edu front page

library(OpenMx)

data(demoOneFactor) # load the demoOneFactor dataframe

manifests <- names(demoOneFactor) # set the manifest to the 5 demo variables

latents <- c("G") # define 1 latent variable

model <- mxModel(model="One Factor", type="RAM",
manifestVars = manifests,
latentVars = latents,
...
mxSave

mxPath(from=latents, to=manifests, labels=paste("b", 1:5, sep="")),
mxPath(from=manifests, arrows=2, labels=paste("u", 1:5, sep="")),
mxPath(from=latents, arrows=2, free=FALSE, values=1.0),
mxData(cov(demoOneFactor), type="cov", numObs=500)
)
model <- mxRun(model) # Run the model, returning the result into model
summary(model) # Show summary of the fitted model

---

mxSave

Save End State to Checkpoint File

Description

The function saves the last state of a model to a checkpoint file.

Usage

mxSave(model, chkpt.directory = ".", chkpt.prefix = "")

Arguments

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

Details

In general, the arguments ‘chkpt.directory’ and ‘chkpt.prefix’ should be identical to the mxOption: ‘Checkpoint Directory’ and ‘Checkpoint Prefix’ that were specified on the model before execution. Alternatively, the checkpoint file can be manually loaded as a data.frame in R. Use `read.table` with the options `header=TRUE, sep="\\t", stringsAsFactors=FALSE, check.names=FALSE`.

Value

Returns a logical indicating the succes of writing the checkpoint file to the checkpoint directory.

References

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

library(OpenMx)

# Simulate some data

x = rnorm(1000, mean=0, sd=1)
y = 0.5*x + rnorm(1000, mean=0, sd=1)
tmpframe <- data.frame(x, y)
tmpnames <- names(tmpframe)

# Create a model that includes an expected covariance matrix, an expectation function, a fit function, and an observed covariance matrix

data <- mxData(cov(tmpframe), type="cov", numObs = 1000)
expCov <- mxMatrix(type="Symm", nrow=2, ncol=2, values=c(2,1,1,2), free=TRUE, name="expCov")
expFunction <- mxExpectationNormal(covariance="expCov", dimnames=tmpnames)
fitFunction <- mxFitFunctionML()

# Use mxRun to optimize the free parameters in the expected covariance matrix

modelOut <- mxRun(testModel)
modelOut$expCov

# Save the ending state of modelOut in a checkpoint file

mxSave(modelOut)

# Restore the saved model from the checkpoint file

modelSaved <- mxRestore(testModel)
modelSaved$expCov

# Imagine how much time you saved by not having to re-run the model that took hours or days to run.

---

mxSetDefaultOptions  Reset global options to the default

Description

Reset global options to the default

Usage

mxSetDefaultOptions()
**mxSimplify2Array**

Like `simplify2array` but works with vectors of different lengths

**Description**

Vectors are filled column-by-column into a matrix. Shorter vectors are padded with NAs to fill whole columns.

**Usage**

```r
mxSimplify2Array(x, higher = FALSE)
```

**Arguments**

- `x` a list of vectors
- `higher` whether to produce a higher rank array (defaults to `FALSE`)

**Examples**

```r
ev1 <- 1:3
ev2 <- 4:5
ev3 <- 6:10
mxSimplify2Array(list(v1, v2, v3))
```

```
  [,1] [,2] [,3]
# [1,]  1   4   6
# [2,]  2   5   7
# [3,]  3  NA   8
# [4,] NA  NA   9
# [5,] NA  NA  10
```

---

**mxStandardizeRAMpaths**

Standardize RAM models’ path coefficients

**Description**

Provides a dataframe containing the standardized values of all nonzero path coefficients appearing in the A and S matrices of models that use RAM expectation (either of type="RAM" or containing an explicit `mxExpectationRAM()` statement). These standardized values are what the path coefficients would be if all variables in the analysis–both manifest and latent–were standardized to unit variance. Can optionally include asymptotic standard errors for those standardized coefficients, computed via the delta method.

**Usage**

```r
mxStandardizeRAMpaths(model, SE=FALSE)
```
### Arguments

- **model**
  - An `mxModel` object, that either uses RAM expectation or contains at least one submodel that does.

- **SE**
  - Logical. Should standard errors be included with the standardized point estimates? Defaults to FALSE. Certain conditions are required for use of SE=TRUE; see "Details" below.

### Details

Matrix A contains the Asymmetric paths, i.e. the single-headed arrows. Matrix S contains the Symmetric paths, i.e. the double-headed arrows. The function will work even if `mxMatrix` objects named "A" and "S" are absent from the model, since it identifies which matrices in the model have been assigned the roles of A and S in the `mxExpectationRAM` statement. Note that, in models of type="RAM", the necessary matrices and expectation statement are automatically assembled from the `mxPath` objects.

If `model` contains any submodels with independent=TRUE that use RAM expectation, `mxStandardizeRAMpaths()` automatically applies itself recursively over those submodels.

Use of SE=TRUE requires that package numDeriv be installed. It also requires that `model` contain no `mxConstraint` statements, and have a nonempty hessian element in its output slot. There are three common reasons why the latter condition may not be met. First, the model may not have been run yet, i.e. it was not output by `mxRun()`. Second, `mxOption"Hessian"` might be set to "No". Third, computing the Hessian matrix might possibly have been skipped per a user-defined `mxCompute*` statement (if any are present in the model). If `model` contains RAM-expectation submodels with independent=TRUE, these conditions are checked separately for each such submodel.

In any event, using these standard errors for hypothesis-testing or forming confidence intervals is not generally advised. Instead, it is considered best practice to conduct likelihood-ratio tests or compute likelihood-based confidence intervals (from `mxCI()`), as in examples below.

The user should note that `mxStandardizeRAMpaths()` only cares whether an element of A or S is nonzero, and not whether it is a fixed or free parameter. So, for instance, if the function is used on a model not yet run, any free parameters in A or S initialized at zero will not appear in the function’s output.

The user is warned to interpret the output of `mxStandardizeRAMpaths()` cautiously if any elements of A or S depend upon definition variables.

### Value

If argument `model` is a single-group model that uses RAM expectation, then `mxStandardizeRAMpaths()` returns a dataframe, with one row for each nonzero path coefficient in A and S, and with the following columns:

- **name**
  - Character strings that uniquely identify each nonzero path coefficient in terms of the model name, the matrix ("A" or "S"), the row number, and the column number.

- **label**
  - Character labels for those path coefficients that are labeled elements of an `mxMatrix` object, and NA for those that are not. Note that path coefficients having the same
label (and therefore the same UNstandardized value) can have different
standardized values, and therefore the same label may appear more than once in this
dataframe.

matrix Character strings of "A" or "S", depending on which matrix contains the given
path coefficient.

row Character. The rownames of the matrix containing each path coefficient; row
numbers are used instead if the matrix has no rownames.

col Character. The colnames of the matrix containing each path coefficient; column
numbers are used instead if the matrix has no colnames.

Raw.Value Numeric values of the raw (i.e., UNstandardized) path coefficients.

Raw.SE Numeric values of the asymptotic standard errors of the raw path coefficients if
if se=true, or NA otherwise.

Std.Value Numeric values of the standardized path coefficients.

Std.SE Numeric values of the asymptotic standard errors of the standardized path coef-
ficients if SE=TRUE, or NA otherwise.

If model is a multi-group model containing at least one submodel with RAM expectation, then
mxStandardizeRAMpaths() returns a list. The list has a number of elements equal to the number
of submodels that either have RAM expectation or contain a submodel that does. List elements cor-
responding to RAM-expectation submodels contain a dataframe, as described above. List elements
corresponding to "container" submodels are themselves lists, of the kind described here.

Examples

library(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents  <- c("G")
factorModel <- mxModel(model="One Factor", type="RAM",
  manifestVars = manifests,
  latentVars   = latents,
  mxPath(from=latents, to=manifests),
  mxPath(from=manifests, arrows=2, values=0.1),
  mxPath(from=latents, arrows=2,free=FALSE, values=1.0),
  mxData(cov(demoOneFactor), type="cov",numObs=500)
)
factorFit <- mxRun(factorModel)
summary(factorFit)$parameters
mxStandardizeRAMpaths(model=factorFit,SE=FALSE)

## Likelihood ratio test of variable x1's factor loading:
factorModelNull <- omxSetParameters(factorModel,labels="One Factor.A[1,6]",
  values=0,free=FALSE)
factorFitNull <- mxRun(factorModelNull)
mxCompare(factorFit,factorFitNull)[2,"p"] #<--p-value

## Confidence intervals for all standardized paths:
factorModel2 <- mxModel(model=factorModel,
  mxMatrix(type="Iden",nrow=nrow(factorModel$A),name="I"),
mxThreshold

Create List of Thresholds

Description
This function creates a list of thresholds.

Usage
mxThreshold(vars, nThresh=NA,
free=FALSE, values=NA, labels=NA,
lbound=NA, ubound=NA)

Arguments
vars character vector. These are the variables for which thresholds are to be specified.
nThresh numeric vector. These are the number of thresholds for each variables listed in 'vars'.
free boolean vector. Indicates whether threshold parameters are free or fixed.
values numeric vector. The starting values of the parameters.
labels character vector. The names of the parameters.
lbound numeric vector. The lower bounds of free parameters.
ubound numeric vector. The upper bounds of free parameters.

Details
The mxPath function creates MxThreshold objects. These consist of a list of ordinal variables and the thresholds that define the relationship between the observed ordinal variable and the continuous latent variable assumed to underly it. This function directly mirrors the usage of mxPath, but is used to specify thresholds rather than means, variances and bivariate relationships.

The ‘vars’ argument specifies which variables you wish to specify thresholds for. Variables are referenced by name, and these names must appear in the ‘manifestVar’ argument of the mxModel function if thresholds are to be correctly processed. Additionally, variables for which thresholds are specified must be specified as ordinal factors in whatever data is included in the model.
The ‘nThresh’ argument specifies how many thresholds are to be specified for the variable or variables included in the ‘vars’ argument. The number of thresholds for a particular variable should be one fewer than the number of categories specified for that variable.

The ‘free’ argument specifies whether the thresholds created by the mxThreshold 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 thresholds.

The ‘values’ is a numeric vectors containing the starting values of the created thresholds. ‘values’ gives a starting value for estimation. The ‘labels’ argument specifies the names of the parameters in the resulting MxThreshold object. The ‘lbound’ and ‘ubound’ arguments specify lower and upper bounds for the created threshold parameters.

Thresholds for multiple variables may be specified simultaneously by including a vector of variable names to the ‘vars’ argument. When multiple variables are included in the ‘vars’ argument, the length of the ‘vars’ argument must be evenly divisible by the length of the ‘nThresh’ argument. All subsequent arguments (‘free’ through ‘ubound’) should have their lengths be a factor of the total number of thresholds specified for all variables.

If four variables are included in the ‘vars’ argument, then the ‘nThresh’ argument should contain either one, two or four elements. If the ‘nThresh’ argument specifies two thresholds for each variable, then ‘free’, ‘values’, and all subsequent arguments should specify eight values by including one, two, four or eight elements. Whenever fewer values are specified than are required (e.g., specify two values for eight thresholds), then the entire vector of values is repeated until the required number of values is reached, and will return an error if the correct number of values cannot be achieved by repeating the entire vector.

**Value**

Returns a list of thresholds.

**References**

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

**See Also**

demo("mxThreshold")

mxPath for comparable specification of paths. mxMatrix for a matrix-based approach to thresholds specification; mxModel for the container in which mxThresholds are embedded. More information about the OpenMx package may be found here.
Description

Makes multiple attempts to fit an MxModel object with mxRun() until the optimizer yields an acceptable solution or the maximum number of attempts is reached. Each attempt uses the parameter estimates of the previous attempt as start values, except they are each multiplied by random draws from a uniform distribution. From among its attempts, the function returns the fitted, post-mxRun() model with the smallest fit-function value, and prints to the console the start values it used for that model.

Usage

mxTryHard(model, extraTries = 10, greenOK = FALSE, loc = 1,
        scale = 0.25, checkHess = TRUE, fit2beat = Inf, paste = TRUE,
        iterationSummary=FALSE, bestInitsOutput=TRUE, showInits=FALSE,
        ...)  

Arguments

model The model to be run; object of class MxModel.

extraTries The number of attempts to run the model in addition to the first. In effect, is the maximum number of attempts mxTryHard() will make, since the function will stop once an acceptable solution is reached. Defaults to 10, in which case a maximum of 11 total attempts will be made.

greenOK Logical; is a solution with Mx status GREEN (npsolstatus=1) acceptable? Defaults to FALSE.

loc, scale The location and scale parameters of the uniform (rectangular) distribution from which random values are drawn to disturb start values between attempts. The location parameter is the distribution’s median, and the scale parameter is the half-width of the rectangle (that is, the absolute difference between the median and the extrema). Defaults to a uniform distribution on the interval (0.75, 1.25).

checkHess Logical; is a positive-definite Hessian a requirement for an acceptable solution? Defaults to TRUE.

fit2beat An upper limit to the objective-function value that an acceptable solution may have. Useful if a nested submodel of model has already been fitted, since model, with its additional free parameters, should not yield a fit-function value any greater than that of the submodel.

paste Logical. If TRUE (default), start values for the returned fitted model are printed to console as a comma-separated string. This is useful if the user wants to copy-paste these values into an R script, say, in an omxSetParameters() statement. If FALSE, the vector of start values is printed as-is. Note that this vector, from omxGetParameters(), has names corresponding to the free parameters; these names are not displayed when paste=TRUE.

iterationSummary Logical. If TRUE, displays parameter estimates and fit values for every fit attempt. Defaults to FALSE.

bestInitsOutput Logical. If TRUE, outputs starting values that resulted in best fit, according to format specified by paste argument. Defaults to TRUE.
showInits Logical. If TRUE, displays starting values for every fit attempt. Defaults to FALSE.

Additional arguments to be passed to \texttt{mxRun()} (for example, intervals=TRUE).

Note that \texttt{mxTryHard()} always internally invokes \texttt{mxRun()} with argument suppressWarnings=TRUE.

\section*{Value}

Usually, \texttt{mxTryHard()} returns a post-\texttt{mxRun()} \texttt{MxModel} object. Specifically, this will be the fitted model having the smallest fit-function value found by \texttt{mxTryHard()} during its attempts. The start values used to obtain this fitted model are printed to console.

If every attempt at running \texttt{model} fails, \texttt{mxTryHard()} returns an object of class 'try-error', and the start values from the last attempt are printed to console.

\texttt{mxTryHard()} throws a warning if the returned \texttt{MxModel} object has a nonzero npsolstatus.

\section*{See Also}

\texttt{mxRun()}

\section*{Examples}

\begin{verbatim}
library(OpenMx)
data(demoOneFactor) # load the demoOneFactor dataframe

manifests <- names(demoOneFactor) # set the manifest to the 5 demo variables
latents <- c("G")  # define 1 latent variable
model <- mxModel(model="One Factor", type="RAM",
  manifestVars = manifests,
  latentVars = latents,
  mxPath(from=latents, to=manifests, labels=paste("b", 1:5, sep="")),
  mxPath(from=manifests, arrows=2, labels=paste("u", 1:5, sep="")),
  mxPath(from=latents, arrows=2, free=FALSE, values=1.0),
  mxData(cov(demoOneFactor), type="cov", numObs=500)
)
model <- mxTryHard(model) # Run the model, returning the result into model
summary(model) # Show summary of the fitted model
\end{verbatim}
mxVersion

Value
Returns a character vector of type names.

Examples
mxTypes()

mxVersion

Returns Current Version String

Description
This function returns a string with the current version number of OpenMx. Optionally (with verbose = TRUE (the default)), it prints a message containing the version of R, the platform, and the optimiser.

Usage
mxVersion(model = NULL, verbose = TRUE)

Arguments
model optional MxModel to request optimizer from (default = NULL)
verbose Whether to print version information to the console (default = TRUE)

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

Examples

# Print useful version information.
mxVersion()
# If you just want the version, use this call.
x = mxVersion(verbos=false)

library(OpenMx)
data(demoOneFactor) # load the demoOneFactor dataframe
manifests <- names(demoOneFactor) # set the manifest to the 5 demo variables
latents <- c("G") # define 1 latent variable
model <- mxModel(model = "One Factor", type = "RAM",
   manifestVars = manifests,
   latentVars = latents,
   mxPath(from = latents, to = manifests, labels = paste("b", 1:5, sep = "")),
   mxPath(from = manifests, arrows = 2 , labels = paste("u", 1:5, sep = "")),
   mxPath(from = latents , arrows = 2 , free = FALSE, values = 1.0),
   mxData(cov(demoOneFactor), type = "cov", numObs = 500)
**myAutoregressiveData**

Example data with autoregressively related columns

### Description

Data set used in some of OpenMx’s examples.

### Usage

```r
data("myAutoregressiveData")
```

### Format

A data frame with 100 observations on the following variables.

- x1 x variable and time 1
- x2 x variable and time 2
- x3 x variable and time 3
- x4 x variable and time 4
- x5 x variable and time 5

### Details

The rows are independently and identically distributed, but the columns are and auto-correlation structure.

### Source

Simulated.

### References

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

### Examples

```r
data(myAutoregressiveData)
round(cor(myAutoregressiveData), 2)
# note the sub-diagonal correlations (lag 1)
# x1-x2, x2-x3, x3-x4, x4-x5
# and the second sub-diagonal correlations (lag 2)
# x1-x3, x2-x4, x3-x5
```
myFADataRaw

Example 500-row dataset with 12 generated variables

Description
Twelve columns of generated numeric data: x1 x2 x3 x4 x5 x6 y1 y2 y3 z1 z2 z3.

Usage
data(myFADataRaw)

Details
The x variables intercorrelate around .6 with each other.
The y variables intercorrelate around .5 with each other, and correlate around .3 with the X vars.
There are three ordinal variables, z1, z2, and z3.
The data are used in some OpenMx examples, especially confirmatory factor analysis.
There are no missing data.

Examples
data(myFADataRaw)
str(myFADataRaw)

myGrowthKnownClassData
Data for a growth mixture model with the true class membership

Description
Data set used in some of OpenMx’s examples.

Usage
data(“myGrowthKnownClassData”)

Format
A data frame with 500 observations on the following variables.
x1  x variable and time 1
x2  x variable and time 2
x3  x variable and time 3
x4  x variable and time 4
x5  x variable and time 5
c  Known class membership variable
myGrowthMixtureData

Details

The same as myGrowthMixtureData, but with the class membership variable.

Source

Simulated.

References

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

Examples

data(myGrowthKnownClassData)

# plot the observed trajectories
# blue lines are class 1, green lines are class 2
colSel <- c('blue', 'green')[myGrowthKnownClassData$c]
matplot(t(myGrowthKnownClassData[, -6]), type='l', lty=1, col=colSel)

myGrowthMixtureData  Data for a growth mixture model

Description

Data set used in some of OpenMx’s examples.

Usage

data("myGrowthMixtureData")

Format

A data frame with 500 observations on the following variables.

x1  x variable and time 1
x2  x variable and time 2
x3  x variable and time 3
x4  x variable and time 4
x5  x variable and time 5

Details

The same as myGrowthKnownClassData, but without the class membership variable.
myLongitudinalData

Source
Simulated.

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

Examples
data(myGrowthMixtureData)
matplot(t(myGrowthMixtureData), type='l', lty=1)
data(myGrowthKnownClassData)
all(myGrowthKnownClassData[, -6] == myGrowthMixtureData)

---

myLongitudinalData  Data for a linear latent growth curve model

Description
Data set used in some of OpenMx’s examples.

Usage
data("myLongitudinalData")

Format
A data frame with 500 observations on the following variables.
  x1  x variable and time 1
  x2  x variable and time 2
  x3  x variable and time 3
  x4  x variable and time 4
  x5  x variable and time 5

Details
Linear growth model with mean intercept around 10, and slope of about 1.5.

Source
Simulated.

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

**Examples**
```
data(myLongitudinalData)
matplot(t(myLongitudinalData), type='l', lty=1)
```

**Description**
Data set used in some of OpenMx’s examples.

**Usage**
```
data("myRegData")
```

**Format**
A data frame with 100 observations on the following variables.
- **w** Predictor variable
- **x** Predictor variable
- **y** Predictor variable
- **z** Outcome variable

**Details**
w, x, and y are predictors of z. x and y are correlated.

**Source**
Simulated.

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

**Examples**
```
data(myRegData)
summary(lm(z ~ ., data=myRegData))
```
myRegDataRaw

Example regression data with correlated predictors

Description
Data set used in some of OpenMx’s examples.

Usage
data("myRegDataRaw")

Format
A data frame with 100 observations on the following variables.

- w  Predictor variable
- x  Predictor variable
- y  Predictor variable
- z  Outcome variable

Details
w, x, and y are predictors of z. x and y are correlated. Equal to myRegData.

Source
Simulated.

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

Examples
data(myRegData)
data(myRegDataRaw)

all(myRegDataRaw == myRegData)
myTwinData

Twin data on weight and height

Description
Data set used in some of OpenMx’s examples.

Usage
data("myTwinData")

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

- fam  Family ID variable
- age  Age of the twin pair. Range: 17 to 88.
- zyg  Integer codes for zygosity and gender combinations
- part
- wt1  Weight in kilograms for twin 1
- wt2  Weight in kilograms for twin 2
- ht1  Height in meters for twin 1
- ht2  Height in meters for twin 2
- htwt1 Product of ht and wt for twin 1
- htwt2 Product of ht and wt for twin 2
- bmi1  Body Mass Index for twin 1
- bmi2  Body Mass Index for twin 2

Details
Height and weight are highly correlated, and each individually highly heritable. These data present
and opportunity for multivariate behavior genetics modeling.

Source
Timothy Bates

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

Examples

data(myTwinData)

plot( ht1 ~ wt1, myTwinData)
mzfData

MZ female example twin data

Description
Data for extended twin example ETC88.R

Usage
data("mzfData")

Format
A data frame with 3099 observations on the following 37 variables.

famid  a numeric vector
e1    a numeric vector
e2    a numeric vector
e3    a numeric vector
e4    a numeric vector
e5    a numeric vector
e6    a numeric vector
e7    a numeric vector
e8    a numeric vector
e9    a numeric vector
e10   a numeric vector
e11   a numeric vector
e12   a numeric vector
e13   a numeric vector
e14   a numeric vector
e15   a numeric vector
e16   a numeric vector
e17   a numeric vector
e18   a numeric vector
a1    a numeric vector
a2    a numeric vector
a3    a numeric vector
a4    a numeric vector
a5    a numeric vector
a6    a numeric vector
mzmData

a7 a numeric vector
a8 a numeric vector
a9 a numeric vector
a10 a numeric vector
a11 a numeric vector
a12 a numeric vector
a13 a numeric vector
a14 a numeric vector
a15 a numeric vector
a16 a numeric vector
a17 a numeric vector
a18 a numeric vector

Examples

data(mzfData)
str(mzfData)

Description

Data for extended twin example ETC88.R

Usage

data("mzmData")

Format

A data frame with 3019 observations on the following 37 variables.

famid a numeric vector
e1 a numeric vector
e2 a numeric vector
e3 a numeric vector
e4 a numeric vector
e5 a numeric vector
e6 a numeric vector
e7 a numeric vector
e8 a numeric vector
e9  a numeric vector
e10  a numeric vector
e11  a numeric vector
e12  a numeric vector
e13  a numeric vector
e14  a numeric vector
e15  a numeric vector
e16  a numeric vector
e17  a numeric vector
e18  a numeric vector
a1   a numeric vector
a2   a numeric vector
a3   a numeric vector
a4   a numeric vector
a5   a numeric vector
a6   a numeric vector
a7   a numeric vector
a8   a numeric vector
a9   a numeric vector
a10  a numeric vector
a11  a numeric vector
a12  a numeric vector
a13  a numeric vector
a14  a numeric vector
a15  a numeric vector
a16  a numeric vector
a17  a numeric vector
a18  a numeric vector

Examples

data(mzmData)
str(mzmData)
Named-entity

<table>
<thead>
<tr>
<th>Description</th>
<th>A named entity is an S4 object that can be referenced by name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
<td>Every named entity is guaranteed to have a slot called &quot;name&quot;. Within a model, the named entities of that model can be accessed using the $ operator. 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. The following S4 classes are named entities in the OpenMx library: MxAlgebra, MxConstraint, MxMatrix, MxModel, MxData, and MxObjective.</td>
</tr>
</tbody>
</table>
| Examples    | library(OpenMx)

# Create a model, add a matrix to it, and then access the matrix by name.  
testModel <- mxModel(model="anEmptyModel")

testMatrix <- mxMatrix(type="Full", nrow=2, ncol=2, values=c(1,2,3,4), name="yourMatrix")

yourModel <- mxModel(testModel, testMatrix, name="noLongerEmpty")

yourModel$yourMatrix |

| nuclear_twin_design_data | Twin data from a nuclear family design |

<table>
<thead>
<tr>
<th>Description</th>
<th>Data set used in some of OpenMx’s examples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>data(&quot;nuclear_twin_design_data&quot;)</td>
</tr>
</tbody>
</table>
Format
A data frame with 1743 observations on the following variables.

- Twin1
- Twin2
- Father
- Mother
- zyg  Zygosity of the twin pair

Details
This is a wide format data set. A single variable has values for different member of the same nuclear family.

Source
Likely simulated.

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

Examples
```r
data(nuclear_twin_design_data)
cor(nuclear_twin_design_data[,5], use="pairwise.complete.obs")
```

---

**numHess1**  
**numeric Hessian data 1**

Description

data file used by the HessianTest.R script

Usage
```r
data("numHess1")
```

Format
A 12 by 12 data frame containing Hessian (numeric variables a-l)

Examples
```r
data(numHess1)
str(numHess1)
```
numHess2  numeric Hessian data 2

Description

data file used by the HessianTest.R script

Usage

data("numHess2")

Format

A 12 by 12 data frame containing Hessian matrix (numeric variables a-l)

Examples

data(numHess2)
str(numHess2)

omxAllInt  All Interval Multivariate Normal Integration

Description

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

Usage

omxAllInt(covariance, means, ...)

Arguments

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

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

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

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

All further arguments are considered threshold matrices.

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

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

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

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

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

See Also

`omxMnor`

Examples

```r
data(myFADataRaw)

covariance <- cov(myFADataRaw[,1:5])
means <- colMeans(myFADataRaw[,1:5])

# Integrate from -Infinity to 0 and 0 to 1 on first variable
thresholdForColumn1 <- cbind(c(-Inf, 0, 1))
# Note: The first variable will never be calculated from 1 to +Infinity.
```
# These columns will be integrated from -Inf to -1, -1 to 0, etc.
thresholdsForColumn2 <- cbind(c(-Inf, -1, 0, 1, Inf))
thresholdsForColumns3and4 <- cbind(c(-Inf, 1.96, 2.326, Inf),
                                  c(-Inf, -1.96, 2.326, Inf))

# The integration
omxAllInt(covariance, means,
           thresholdsForColumn1, thresholdsForColumn2,
           thresholdsForColumns3and4, thresholdsForColumn2)
# Notice that columns 2 and 5 are assigned identical thresholds.

# An alternative specification of the same calculation follows
covariance <- cov(myFADataRaw[,1:5])
means <- colMeans(myFADataRaw[,1:5])

# Note NAs to indicate the end of the sequence of thresholds.
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(covariance, means, thresholds)

---

omxApply

On-Demand Parallel Apply

Description

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

Usage

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

Arguments

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

See Also

omxLapply, omxSapply
Examples

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

Description

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

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

Usage

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

Arguments

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

See Also

`omxGetParameters`, `omxSetParameters`

Examples

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

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

model$matrices$A

# $labels
# [1,] [,1] [,2] [,3]
```
Description

This function returns a brownie recipe.

Usage

```r
omxBrownie(quantity=1, walnuts=TRUE)
```

Arguments

- **quantity**: Number of batches of brownies desired. Defaults to one.
- **walnuts**: Logical. Indicates whether walnuts are to be included in the brownies. Defaults to TRUE.

Details

Returns a brownie recipe. Alter the `quantity` variable to make more pans of brownies. Ingredients, equipment and procedure are listed, but neither ingredients nor equipment are provided.

Value

Returns a brownie recipe.

References

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

See Also

More information about the OpenMx package may be found here.

Examples

```r
# Return a brownie recipe
omxBrownie()
```
omxCheckCloseEnough

Approximate Equality Testing Function

Description

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

Usage

omxCheckCloseEnough(a, b, epsilon = 10^(-15), na.action = na.fail)

Arguments

a  a numeric vector or matrix
b  a numeric vector or matrix
epsilon  a non-negative tolerance threshold
na.action  either na.fail (default) or na.pass. Use of na.omit or na.exclude is not recommended.

Details

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

When na.action is set to na.pass, a and b are expected to have identical missingness patterns.

References

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

See Also

omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9, 3.0), epsilon = 0.5)
omxCheckCloseEnough(matrix(3, 3), matrix(4, 3, 3), epsilon = 2)
# Throws an error
try(omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9, 3.0), epsilon = 0.01))
omxCheckEquals

Equality Testing Function

Description

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

Usage

omxCheckEquals(a, b)

Arguments

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

Details

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

References

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

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckSetEquals, omxCheckTrue, omxCheckIdentical

Examples

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

omxCheckEquals(FALSE, FALSE)

# Throws an error
try(omxCheckEquals(c(1, 2, 3), c(2, 1, 3)))
omxCheckError  Correct Error Message Function

Description

This function tests whether the correct error message is thrown.

Usage

omxCheckError(expression, message)

Arguments

expression an R expression that produces an error
message a character string with the desired error message

Details

Arguments ‘expression’ and ‘message’ give the expression that generates the error and the message that is supposed to be generated, respectively.

References

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

See Also

omxCheckWarning omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

A <- mxMatrix('Full', 1, 1, labels = 'data.foo', free = TRUE, name = 'A')
model <- mxModel('model', A)
omxCheckError(mxRun(model),
paste("The definition variable 'data.foo'",
"has been assigned to a",
"free parameter in matrix 'A'"))
omxCheckCloseEnough(matrix(3, 3, 3), matrix(4, 3, 3), epsilon = 2)
# Throws error, check the message
tmsg <- paste("In omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9, 3), 0.01)
"=": not equal to within 0.01 : '1 2 3' and '1.1 1.9 3'")
omxCheckError(omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9, 3.0), .01), tmsg)
omxCheckIdentical 

Exact Equality Testing Function

Description

This function tests whether two objects are equal.

Usage

omxCheckIdentical(a, b)

Arguments

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

Details

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

References

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

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

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

omxCheckIdentical(FALSE, FALSE)

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

Description

This is an internal function exported for those people who know what they are doing.

Usage

omxCheckNamespace(model, namespace)

Arguments

model
namespace

Details

This function checks that the named entities in the model are valid.

omxCheckSetEquals

Set Equality Testing Function

Description

This function tests whether two vectors contain the same elements.

Usage

omxCheckSetEquals(a, b)

Arguments

a
b

the first vector to compare.
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.
omxCheckTrue

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckIdentical, omxCheckTrue, omxCheckEquals

Examples

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

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

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

omxCheckTrue

Boolean Equality Testing Function

Description

This function tests whether an object is equal to TRUE.

Usage

omxCheckTrue(a)

Arguments

a

the value to test.

Details

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

References

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

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckEquals
Examples

    omxCheckTrue(1 + 1 == 2)

    omxCheckTrue(matrix(TRUE, 3, 3))

    # Throws an error
    try(omxCheckTrue(FALSE))

omxCheckWarning  Correct Warning Message Function

Description

This function tests whether the correct warning message is thrown.

Usage

    omxCheckWarning(expression, message)

Arguments

    expression  an R expression that produces a warning
    message     a character string with the desired warning message

Details

Arguments ‘expression’ and ‘message’ give the expression that generates the warning and the message that is supposed to be generated, respectively.

References

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

See Also

    omxCheckError omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

    msg <- paste("Objective functions like mxFIMLObjective()",
                  "have been deprecated in favor of expectation and fit functions.
                  Please use mxExpectationNormal(covariance=, means=, ...) instead,
                  and add a call to mxFitFunctionML()",
                  "See examples at help(mxExpectationNormal)")
    foo <- omxCheckWarning(mxFIMLObjective('cov', 'mean'), msg)
omxCheckWithinPercentError

Approximate Percent Equality Testing Function

Description

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

Usage

omxCheckWithinPercentError(a, b, percent = 0.1)

Arguments

a  
a numeric vector or matrix.

b  
a numeric vector or matrix.

percent  
a non-negative percentage.

Details

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

References

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

See Also

omxCheckCloseEnough, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

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

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

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

Description
Add constraint to ML model to keep thresholds in order

Usage
omxConstrainMLThresholds(model, dist = 0.1)

Arguments
- model: the MxModel to which constraints should be added
- dist: unused

Details
This function adds a nonlinear constraint to an ML model. The constraint keeps the thresholds in order. Constraints often slow model estimation, however, keeping the thresholds in increasing order helps ensure the likelihood function is well-defined. If you're having problems with ordinal data, this is one of the things to try.

Value
A new MxModel object with the constraints added

See Also
demo("omxConstrainMLThresholds")

omxDetectCores

Description
Detects the number of cores on the local machine

Usage
omxDetectCores()

Arguments
- ...: unused
**omxGetNPSOL**

**Description**
Get the non-CRAN version of OpenMx from the OpenMx website.

**Usage**

```r
omxGetNPSOL()
```

**Details**
This function

**Value**
Invisible NULL

**Examples**

```r
## Not run: omxGetNPSOL()
```

---

**omxGetParameters**

**Fetch Model Parameters**

**Description**
Return a vector of the chosen parameters from the model.

**Usage**

```r
omxGetParameters(model, indep = FALSE, free = c(TRUE, FALSE, NA),
      fetch = c('values', 'free', 'lbound', 'ubound', 'all'))
```

**Arguments**

- `model` a MxModel object
- `indep` fetch parameters from independent submodels.
- `free` fetch either free parameters (TRUE), or fixed parameters or both types. Default value is TRUE.
- `fetch` which attribute of the parameters to fetch. Default choice is ‘values’.
Details

The argument ‘free’ dictates whether to return only free parameters or only fixed parameters or both free and fixed parameters. The function can return unlabelled free parameters (parameters with a label of NA). These anonymous free parameters will be identified as ‘modelname.matrixname[row,col]’. It will not return fixed parameters that have a label of NA. No distinction is made between ordinary labels, definition variables, and square bracket constraints. The function will return either a vector of parameter values, or free/fixed designations, or lower bounds, or upper bounds, depending on the ‘fetch’ argument. Using fetch with ‘all’ returns a data frame that is populated with all of the attributes.

See Also

omxSetParameters, omxLocateParameters, omxAssignFirstParameters

Examples

library(OpenMx)

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

# Request all free parameters in model
omxGetParameters(model)

# A11  A12 model.A[2,2]
# 1  2  4

# Request fixed parameters from model
omxGetParameters(model, free = FALSE)

# A21
# 3

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

A$free
# [,1] [,2]
# [1,] TRUE TRUE
# [2,] FALSE TRUE

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

# Example using un-labelled parameters
# Read in some demo data
data(demoOneFactor)
# Grab the names for manifestVars
manifestVars <- names(demoOneFactor)
nVar = length(manifestVars) # 5 variables
factorModel <- mxModel("One Factor",
  mxMatrix(name="A", type="Full", nrow=nVar, ncol=1, values=0.2, free=TRUE,
    lbound = 0.0, labels=letters[1:nVar]),
  mxMatrix(name="L", type="Symm", nrow=1, ncol=1, values=1, free=FALSE),
  # the "U" matrix has nVar (5) anonymous free parameters
  mxMatrix(name="U", type="Diag", nrow=nVar, ncol=nVar, values=1, free=TRUE),
  mxAlgebra(expression=A %&% L + U, name="R"),
  mxExpectationNormal(covariance="R", dimnames=manifestVars),
  mxFitFunctionML(),
  mxData( observed= cov(demoOneFactor), type= "cov", numObs=500)
)

# Get all free parameters
params <- omxGetParameters(factorModel)
lbound <- omxGetParameters(factorModel, fetch="lbound")
# Set new values for these params, saving them in a new model
newFactorModel <- omxSetParameters(factorModel, names(params), values = 1:10)
# Read out the values from the new model
newParams <- omxGetParameters(newFactorModel)

---

**omxGetRAMDepth**

### Description

Get the potency of a matrix for inversion speed-up

### Usage

omxGetRAMDepth(A, maxdepth = nrow(A) - 1)

### Arguments

- **A**: MxMatrix object
- **maxdepth**: Numeric. maximum depth to check

### Details

This function is used internally by the `mxExpectationRAM` function to determine how far to expand 
\((I - A)^{-1} = I + A + A^2 + A^3 + \ldots\). It is similarly used by `mxExpectationLISREL` in expanding 
\((I - B)^{-1} = I + B + B^2 + B^3 + \ldots\). In many situations \(A^2\) is a zero matrix (nilpotent of order 2).
So when \(A\) has large dimension it is much faster to compute \(I + A\) than \((I - A)^{-1}\).
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.

On-Demand Parallel Lapply

Description

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

Usage

omxLapply(x, fun, ...)

Arguments

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

See Also

omxApply, omxSapply

Examples

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

omxLocateParameters  Summarize Model Parameters

Description

Return a data.frame object summarizing the free parameters in the model.

Usage

omxLocateParameters(model, labels = NULL, indep = FALSE)

Arguments

model  a MxModel object
labels  optionally specify which free parameters to retrieve.
indep  fetch parameters from independent submodels.

Details

Invoking the function with the default value for the ‘labels’ argument retrieves all the free parameters. The ‘labels’ argument can be used to select a subset of the free parameters. Note that ‘NA’ is a valid possible input to the ‘labels’ argument.

See Also

omxGetParameters, omxSetParameters, omxAssignFirstParameters

Examples

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

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

# Request all free parameters in model
omxLocateParameters(model)
omxLogical

Logical mxAlgebra() operators

Description

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

Usage

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

Arguments

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

Examples

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

model <- mxModel(A, B, EPSILON, name = 'model')
mxEval(omxNot(A), model)
mxEval(omxGreaterThan(A,B), model)
mxEval(omxLessThan(B,A), model)
mxEval(omxOr(omxNot(A),B), model)
mxEval(omxAnd(omxNot(B), A), model)
mxEval(omxApproxEquals(A,B, EPSILON), model)
Description

omxCbind columnwise binding of two or more MxMatrices. omxRbind rowwise binding of two or more MxMatrices. omxTranspose transpose of MxMatrix.

Usage

omxCbind(..., allowUnlabeled =
    getOption("mxOptions")[["Allow Unlabeled"]],
    dimnames = NA, name = NA)
omxRbind(..., allowUnlabeled =
    getOption("mxOptions")[["Allow Unlabeled"]],
    dimnames = NA, name = NA)
omxTranspose(matrix, allowUnlabeled =
    getOption("mxOptions")[["Allow Unlabeled"]],
    dimnames = NA, name = NA)

Arguments

... two or more MxMatrix objects
matrix MxMatrix input
allowUnlabeled whether or not to accept free parameters with NA labels
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

Description

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

Usage

omxMnor(covariance, means, lbound, ubound)
Arguments

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

Details

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

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

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

Examples

```r
data(myFADatRaw)
covariance <- cov(myFADatRaw[,1:3])
means <- colMeans(myFADatRaw[,1:3])
lbound <- c(-Inf, 0, 1)  # Integrate from -Infinity to 0 on first variable
ubound <- c(0, Inf, 2.5)  # From 0 to +Infinity on second, and from 1 to 2.5 on third
omxMnor(covariance, means, lbound, ubound)
# 0.0005995

# An alternative specification of the bounds follows
# Integrate from -Infinity to 0 on first variable
v1bound = c(-Inf, 0)
# From 0 to +Infinity on second
v2bound = c(0, Inf)
# and from 1 to 2.5 on third
v3bound = c(1, 2.5)
bounds <- cbind(v1bound, v2bound, v3bound)
lbound <- bounds[1,]
ubound <- bounds[2,]
omxMnor(covariance, means, lbound, ubound)
```

Description

Assign new names to the unnamed parameters
Usage

omxNameAnonymousParameters(model, indep = FALSE)

Arguments

model       the MxModel
indep       whether models are independent

Value

a list with components for the new MxModel with named parameters, and the new names.

Description

Get quantiles from a normal distribution

Usage

omxNormalQuantiles(nBreaks, mean = 0, sd = 1)

Arguments

nBreaks     the number of thresholds, or a vector of the number of thresholds
mean        the mean of the underlying normal distribution
sd          the standard deviation of the underlying normal distribution

Value

a vector of quantiles

Examples

omxNormalQuantiles(3)
omxNormalQuantiles(3, mean=7)
omxNormalQuantiles(2, mean=1, sd=3)
Description

Create parallel models for parallel confidence intervals

Usage

omxParallelCI(model, run = TRUE)

Arguments

model an MxModel with confidence intervals in it
run whether to run the model or just return the parallelized interval models

Value

an MxModel object

Examples

require(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
lats <- c("G")
factorModel <- mxModel("One Factor",
    type="RAM",
    manifestVars=manifests,
    latentVars=lats,
    mxPath(from=lats, to=manifests),
    mxPath(from=manifests, arrows=2),
    mxPath(from=lats, arrows=2, free=FALSE, values=1.0),
    mxData( observed= cov(demoOneFactor), type="cov",
    numObs=500),
    # add confidence intervals for free params in A and S matrices
    mxCI(c('A', 'S'))) 
factorRun <- mxRun(factorModel)
factorCI <- omxParallelCI(factorRun) # Run CIs in parallel
omxQuotes

Description
Quote helper function, often for error messages.

Usage
omxQuotes(name)

Arguments
name a character vector

Details
This is a helper function for creating a nicely put together formatted string.

Value
a character string

Examples
omxQuotes(c("Oh", "blah", "dee", "Oh", "blah", "da"))
omxQuotes(c("A", "S", "F"))
omxQuotes("Hello World")

omxRAMtoML

Description
Convert a RAM model to an ML model

Usage
omxRAMtoML(model)

Arguments
model the MxModel

Details
This is a legacy function that was once used to convert RAM models to ML models in the old (1.0 release of OpenMx) objective function style.
Value

an ML model with an ML objective

---

### omxRMSEA

Get the RMSEA with confidence intervals from model

---

### Description

This function calculates the Root Mean Square Error of the Approximation (RMSEA) for a model and computes confidence intervals for that fit statistic.

### Usage

```r
omxRMSEA(model, lower=.025, upper=.975, null=.05, ...)
```

### Arguments

- `model`: An MxModel object for which the RMSEA is desired
- `lower`: The lower confidence bound for the confidence interval
- `upper`: The upper confidence bound for the confidence interval
- `null`: Value of RMSEA used to test for close fit
- `...`: Further named arguments passed to summary

### Details

To help users obtain fit statistics related to the RMSEA, this function computes confidence intervals and a test for close fit. The user determines how close the fit is required to be by setting the `null` argument to the value desired for comparison.

### Value

A named vector with elements `lower`, `est.rmsea`, `upper`, `null`, and `Prob(x <= null)`.

### References

Examples
require(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents <- c("G")
factorModel <- mxModel("One Factor",
  type="RAM",
  manifestVars=manifests,
  latentVars=latents,
  mxPath(from=latents, to=manifests),
  mxPath(from=manifests, arrows=2),
  mxPath(from=latents, arrows=2, free=FALSE, values=1.0),
  mxData( observed= cov(demoOneFactor), type="cov", numObs=500))
factorRun <- mxRun(factorModel)
factorSat <- mxRefModels(factorRun, run=TRUE)
summary(factorRun, refModels=factorSat)
# Gives RMSEA with 95% confidence interval
omxRMSEA(factorRun, .05, .95, refModels=factorSat)
# Gives RMSEA with 90% confidence interval
# and probability of 'close enough' fit

omxSapply

On-Demand Parallel Sapply

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

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

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

See Also
omxApply, omxLapply
omxSaturatedModel

Create Reference (Saturated and Independence) Models

Description

This function creates and optionally runs saturated and independence (null) models of a base model or data set for use with mxSummary to obtain more fit indices.

Usage

mxRefModels(x, run=FALSE)

Arguments

x A MxModel object, data frame, or matrix.
run logical. If TRUE runs the models before returning; otherwise returns built models without running.

Details

For typical structural equation models the saturated model is the free-est possible model. All covariances and, when possible, all means are estimated. In the case of ordinal data, the ordinal means are fixed to zero and the thresholds are estimated. When the ordinal data are binary, those variances are also constrained to one. This is the free-est possible model, only constrained for model identification. The saturated model is used to create the RMSEA, and Chi-squared fit indices.

The independence model, sometimes called the null model, is a model of each variable being completely independent of every other variable. As such, all the variances and, when possible, all means are estimated. However, covariances are set to zero. Ordinal variables are handled the same for the independence and saturated models. The independence model is used, along with the saturated model, to create CFI and TLI fit indices.

When the mxFitFunctionMultigroup fit function is used, mxRefModels creates the appropriate multigroup saturated and independence models. Saturated and independence models are created separately for each group. Each group has its own saturated and independence model. The multigroup saturated model is a multigroup model where each group has its own saturated model, and similarly for the independence model.

One potentially important limitation of the mxRefModels function is for behavioral genetics models. If variables ‘x’, ‘y’, and ‘z’ are measured on twins 1 and 2 creating the modeled variables ‘x1’ , ‘y1’, ‘z1’, ‘x2’, ‘y2’, ‘z2’, then this function may not create the intended saturated or independence models. In particular, the means of ‘x1’ and ‘x2’ are estimated separately. Similarly, the covariance of

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
omxSapply(x, quantile)
```
'x1' with 'y1' and 'x2' with 'y2' are allowed be be distinct: \( \text{cov}(x1, y1) \neq \text{cov}(x2, y2) \). Moreover, the cross-twin covariances are estimated: e.g. \( \text{cov}(x1, y2) = 0 \).

Another potential misuse of this function is for models with definition variables. If definition variables are detect, a warning message is passed because the saturated and independence model may not be correct because they do not account for the definition variables.

When an MxModel has been run, some effort is made to make the reference models for only the variables used in the model. For covariance data, all variables are modeled by default. For raw data when the model has been run, only the modeled variables are used in the reference models. This matches the behavior of mxModel.

In general, it is best practice to give mxRefModels a model that has already been run.

For IFA models (mxExpectationBA81), the independence model preserves equality constraints among item parameters from the original model.

References

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

Examples

```r
require(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents <- c("G")
factorModel <- mxModel("One Factor",
  type="RAM",
  manifestVars=manifests,
  latentVars=latents,
  mxPath(from=latents, to=manifests),
  mxPath(from=manifests, arrows=2),
  mxPath(from=latents, arrows=2, free=FALSE, values=1.0),
  mxData( observed= cov(demoOneFactor), type="cov", numObs=500))
summary(factorRun <- mxRun(factorModel))
factorSat <- mxRefModels(factorRun, run=TRUE)
summary(factorRun, refModels=factorSat)
```

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

**Description**

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

**Usage**

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

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

Details

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

Value

Returns a new matrix with the filtered data.

References

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

Examples

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

---

omxSetParameters Assign Model Parameters

Description

Modify the attributes of parameters in a model. This function cannot modify parameters that have NA labels. Often you will want to call omxAssignFirstParameters after using this, to force the starting values of equated parameters to the same value (otherwise the model cannot begin to be evaluated)
Usage

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

Arguments

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

See Also

omxGetParameters, omxAssignFirstParameters

Examples

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

# set value of cells labelled "a" and "b" to 1 and 2 respectively
model <- omxSetParameters(model, c('a', 'b'), values = c(1, 2))

# set label of cell labelled "a" to "b" and vice versa
model <- omxSetParameters(model, c('a', 'b'), newlabels = c('b', 'a'))

# set label of cells labelled "a" to "b"
model <- omxSetParameters(model, c('a'), newlabels = 'b')

# ensure initial values are the same for each instance of a labeled parameter
model <- omxAssignFirstParameters(model)

omxSymbolTable  Internal OpenMx algebra operations

Description

This is an internal table used in the OpenMx backend.
Description

OpenMx is a package for structural equation modeling, matrix algebra optimization and other statistical estimation problems. Try the example below. We try and have useful help files: for instance help(mxRun) to learn more. Also the reference manual

Details

OpenMx solves algebra optimization and statistical estimation problems using matrix algebra. Most users use it for Structural equation modeling.

The core function is mxModel, which makes a model. Models are containers for data, matrices, mxPaths algebras, bounds and constraints. Models most often have an expectation function (e.g., mxExpectationNormal) to calculate the expectations for the model. Models need a fit function. Several of these are built-in (e.g., mxFitFunctionML) OpenMx also allows user-defined fit functions for purposes not covered by the built-in functions. (e.g., mxFitFunctionR or mxFitFunctionAlgebra).

Once built, the resulting mxModel can be run (i.e., optimized) using mxRun. This returns the fitted model.

You can see the resulting parameter estimates, algebra evaluation etc using summary(yourModel)

The user’s manual is online (see reference below), but functions mxRun, mxModel, mxMatrix all have working examples to get you started as well.

The main OpenMx functions are: mxAlgebra, mxBounds, mxCI, mxConstraint, mxData, mxMatrix, mxModel, and mxPath

Expectation functions include mxExpectationNormal, mxExpectationRAM, mxExpectationLISREL, and mxExpectationStateSpace;

Fit functions include mxFitFunctionML, mxFitFunctionAlgebra, mxFitFunctionRow and mxFitFunctionR.

OpenMx comes with several useful datasets built-in. Access them using data(package="OpenMx")

To cite package 'OpenMx' in publications use:


ordinalTwinData  Data for ordinal twin model

Description
Example data for ordinal twin-data modelling. Three variables measured in each twin.

Usage
data("ordinalTwinData")

Format
A data frame with 139 observations on the following 7 variables.

zyg  a numeric vector
var1_twin1  a numeric vector
var2_twin1  a numeric vector
rvectorize

var3_twin1 a numeric vector
var1_twin2 a numeric vector
var2_twin2 a numeric vector
var3_twin2 a numeric vector

Examples

data(ordinalTwinData)
str(ordinalTwinData)

rvectorize Vectorize By Row

Description

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

Usage

rvectorize(x)

Arguments

x an input matrix.

See Also

cvectorize, vech, vechs

Examples

rvectorize(matrix(1:9, 3, 3))
rvectorize(matrix(1:12, 3, 4))
summary-MxModel

Model Summary

Description
This function returns summary statistics of a model after it has been run.

Usage
summary(object, ...)

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

Details
mxSummary allows the user to set or override the following parameters of the model:

- **numObs** Numeric. Specify the total number of observations for the model.
- **numStats** Numeric. Specify the total number of observed statistics for the model.
- **refModels** List of MxModel objects. Specify a saturated and independence likelihoods in single argument for testing.
- **SaturatedLikelihood** Numeric or MxModel object. Specify a saturated likelihood for testing.
- **SaturatedDoF** Numeric. Specify the degrees of freedom of the saturated likelihood for testing.
- **IndependenceLikelihood** Numeric or MxModel object. Specify an independence likelihood for testing.
- **IndependenceDoF** Numeric. Specify the degrees of freedom of the independence likelihood for testing.
- **indep** Logical. Set to FALSE to ignore independent submodels in summary.
- **verbose** logical. Changes the printing style for summary (see Details).

The `verbose` argument changes the printing style for the summary of a model. When `verbose=FALSE`, a relatively minimal amount of information is printed: the free parameters, the likelihood, and a few fit indices. When more information is available, more is printed. For example, when the model has a saturated likelihood, several additional fit indices are printed. On the other hand, when `verbose=TRUE`, the compute plan, the data summary, and additional timing information are always printed. Moreover, available fit indices are printed regardless of whether or not they are defined. The undefined fit indices are printed as NA. Running a saturated model and including it with the call to `summary` will define these fit indices and they will display meaningful values. It should be noted that the `verbose` argument only changes the printing style, all of the same information is calculated and exists in the output of `summary`. More information is displayed when `verbose=TRUE`, and less when `verbose=FALSE`.

verbose
The information criteria (AIC, BIC) are reported in a table. The table shows different versions of the information criteria. Each entry in the table is an AIC or BIC obtained using different penalties. In particular, the entries of the table do not show different penalties, but rather different versions of AIC and BIC. For example the AIC is reported with both a Parameters Penalty and a Degrees of Freedom Penalty. AIC generally takes the form $-2LL + 2k$. With the Parameters Penalty $k$ is the number of free parameters. With the Degrees of Freedom Penalty, $k$ is the model degrees of freedom. BIC is defined similarly: $-2LL + k \log(N)$ where $k$ is either the number of free parameters or the degrees of freedom. The Sample-Size Adjusted BIC is only defined for the parameters penalty: $-2LL + k \log((N + 2)/24)$.

The `refModels`, `SaturatedLikelihood`, `SaturatedDoF`, `IndependenceLikelihood`, and `IndependenceDoF` arguments can be used to obtain further fit statistics (RMSEA, CFI, TLI, Chi-Squared). For covariance data, saturated and independence models are fitted automatically so all fit indices are reported. For raw data, these reference models are not estimated to save computational time. For an easy way to make reference models for most cases is provided by the `mxRefModels` function. When the `SaturatedLikelihood` or `IndependenceLikelihood` arguments are used, the appropriate degrees of freedom are attempted to be calculated by OpenMx. However, depending on the model, it may sometimes behoove the user to also explicitly provide the corresponding `SaturatedDoF` and/or `IndependenceDoF`. Again, for the vast majority of cases, the `mxRefModels` function handles these situations effectively and conveniently.

The summary function can report Error codes as follows:

- **1**: The final iterate satisfies the optimality conditions to the accuracy requested, but the sequence of iterates has not yet converged. NPSOL was terminated because no further improvement could be made in the merit function (Mx status GREEN).
- **2**: The linear constraints and bounds could not be satisfied. The problem has no feasible solution.
- **3**: The nonlinear constraints and bounds could not be satisfied. The problem may have no feasible solution.
- **4**: The major iteration limit was reached (Mx status BLUE).
- **6**: The model does not satisfy the first-order optimality conditions to the required accuracy, and no improved point for the merit function could be found during the final linesearch (Mx status RED).
- **7**: The function derivates returned by funcon or funobj appear to be incorrect.
- **9**: An input parameter was invalid.

For many raw data models, OpenMx does not automatically report the absolute fit indices (Chi-Squared, CFI, TLI, and RMSEA). They are available once you fit reference models. See the example given in `mxRefModels`.

OpenMx does not recommend using some fit indices. These are GFI, AGFI, NFI, and SRMR. The Goodness of Fit Index (GFI) and Adjusted Goodness of Fit Index (AGFI) are not recommended because they are strongly influenced by sample size and have rather high Type I error rates (Sharma, Mukherjee, Kumar, & Dillon, 2005). The Normed Fit Index (NFI) has no penalty for model complexity. That is, adding more parameters to a model always improves the NFI, regardless of how useful those parameters are. Because the Non-Normed Fit Index (NNFI), also known as the Tucker-Lewis Index (TLI), does adjust for model complexity it is used instead. Lastly, the Standardized
Root Mean Square Residual (SRMR) is not reported because it (1) only applies to covariance models, having no direct extension to missing data, (2) has no penalty for model complexity, similar to the NFI, and (3) is positively biased (Hu & Bentler, 1999).

References

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


Examples

```r
library(OpenMx)
data(demoOneFactor)  # load the demoOneFactor dataframe
manifests <- names(demoOneFactor)  # set the manifest to the 5 demo variables
latents <- c("G")  # define 1 latent variable
model <- mxModel(model="One Factor", type="RAM",

  manifestVars = manifests,
  latentVars = latents,
  mxPath(from = latents, to=manifests, labels = paste("b", 1:5, sep="")),
  mxPath(from = manifests, arrows = 2, labels = paste("u", 1:5, sep="")),
  mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
  mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
model <- mxRun(model)  # Run the model, returning the result into model

# Show summary of the fitted model
summary(model)

# Compute the summary and store in the variable "statistics"
statistics <- summary(model)

# Access components of the summary
statistics$parameters
statistics$saturatedLikelihood

# Specify a saturated likelihood for testing
summary(model, SaturatedLikelihood = -3000)

# Add a CI and view it in the summary
model = mxRun(mxModel(model=model, mxCI("b5")), intervals = TRUE)
summary(model)
```
**Description**

This function returns the trace of an n-by-n square matrix x, defined to be the sum of the elements on the main diagonal (the diagonal from the upper left to the lower right).

**Usage**

\(\text{tr}(x)\)

**Arguments**

- \(x\) an input matrix. Must be square

**Details**

The input matrix must be square.

**See Also**

vech, rvectorize, cvectorize

**Examples**

\[
\text{tr}(\text{matrix}(1:9, 3, 3))
\]
\[
\text{tr}(\text{matrix}(1:12, 3, 4))
\]

---

**twinData**

*Australian twin sample biometric data.*

**Description**

Australian twin data with 3808 observations on the 12 variables including body mass index (BMI) assessed in both MZ and DZ twins.

Questionnaires were mailed to 5967 pairs age 18 years and over. These data consist of completed questionnaires returned by both members of 3808 (64 percent) pairs. There are two cohort blocks in the data: a younger group (zyg 1:5), and an older group (zyg 6:10)

It is a wide dataset, with two individuals per line. Data include zygosity (zyg), along with heights in metres, weights in kg, and the derived variables BMI in kg/m^2 (stored as “htwt1” and “htwt2”), as well as the log of this variable, stored here as bm1 and bm2. The logged values are more closely normally distributed.
fam is a family identifier. Age is entered only once, as the both twins in each pair share a common age.

fam  a numeric vector
age  a numeric vector
zyg  a numeric vector
part a numeric vector
wt1  a numeric vector
wt2  a numeric vector
ht1  a numeric vector
ht2  a numeric vector
htwt1 a numeric vector
htwt2 a numeric vector
bmi1 a numeric vector
bmi2 a numeric vector

Usage

data(twinData)

Format

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

fam  a numeric vector of family IDs
age  a numeric vector of ages (years)
zyg  a numeric vector of zygosity (see below for important details)
part a numeric vector
wt1  a numeric vector of weights in kg (twin 1)
wt2  a numeric vector of weights in kg (twin 2)
ht1  a numeric vector of heights in kg (twin 1)
ht2  a numeric vector of heights in kg (twin 2)
htwt1 a numeric vector of kg/m^2 twin 1
htwt2 a numeric vector of kg/m^2 twin 2
bmi1 a numeric vector of log BMI for twin 1
bmi2 a numeric vector of log BMI for twin 2

Details

Zygosity is coded as follows 1 == MZFF (i.e MZ females) 2 == MZMM (i.e MZ males) 3 == DZFF
4 == DZMM 5 == DZOS opposite sex pairs

Note: Zygosity 6:10 is the same, for an older cohort in the sample. So: 6 == MZFF (i.e MZ females)
7 == MZMM (i.e MZ males) 8 == DZFF 9 == DZMM 10 == DZOS opposite sex pairs
References


Examples

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

### Description

Data set used in some of OpenMx’s examples.

### Usage

```r
data("twin_NA_dot")
```

### Format

A data frame with 3808 observations on the following variables.

- `fam` Family ID variable
- `age` Age of the twin pair. Range: 17 to 88.
- `zyg` Integer codes for zygosity and gender combinations
- `part`
- `wt1` Weight in kilograms for twin 1
- `wt2` Weight in kilograms for twin 2
- `ht1` Height in meters for twin 1
- `ht2` Height in meters for twin 2
- `htwt1` Product of ht and wt for twin 1
- `htwt2` Product of ht and wt for twin 2
- `bmi1` Body Mass Index for twin 1
- `bmi2` Body Mass Index for twin 2

### Details

Same as `myTwinData` but has . as the missing data value instead of NA.
vec2diag

Source

Timothy Bates

References

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

Examples

data(twin_NA_dot)
summary(twin_NA_dot)
# Note that all variables are treated as factors because of the missing data coding.

vec2diag  Create Diagonal Matrix From Vector

Description

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

Usage

vec2diag(x)

Arguments

x a row or column vector.

Details

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

See Also

diag2vec

Examples

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

**Half-vectorization**

**Description**

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

**Usage**

vech(x)

**Arguments**

- **x**: an input matrix.

**Details**

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

**See Also**

- vech2full, vechs, rvectorize, cvectorize

**Examples**

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

vech2full  

**Inverse Half-vectorization**

**Description**

This function returns the symmetric matrix constructed from a half-vectorization.

**Usage**

vech2full(x)

**Arguments**

- **x**: an input single column or single row 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. The inverse half-vectorization takes a vector and reconstructs a symmetric matrix such that \( \text{vech2full}(\text{vech}(x)) \) is identical to \( x \) if \( x \) is symmetric.

Note that very few vectors have the correct number of elements to construct a symmetric matrix. For example, vectors with 1, 3, 6, 10, and 15 elements can be used to make a symmetric matrix, but none of the other numbers between 1 and 15 can. An error is thrown if the number of elements in \( x \) cannot be used to make a symmetric matrix.

See Also

\texttt{vechs2full}, \texttt{vech}, \texttt{vechs}, \texttt{rvectorize}, \texttt{cvectorize}

Examples

\begin{verbatim}
vech2full(1:10)
matrix(1:16, 4, 4)
vech(matrix(1:16, 4, 4))
vech2full(vech(matrix(1:16, 4, 4)))
\end{verbatim}

\begin{tabular}{l}
\hline
\texttt{vechs} & \textit{Strict Half-vectorization} \\
\hline
\end{tabular}

Description

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

Usage

\texttt{vechs(x)}

Arguments

\begin{verbatim}
x  an input matrix.
\end{verbatim}

Details

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

vech, rvectorize, cvectorize

Examples

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

---

**Inverse Strict Half-vectorization**

**Description**

This function returns the symmetric matrix constructed from a strict half-vectorization.

**Usage**

```r
vechs2full(x)
```

**Arguments**

- `x`: an input single column or single row matrix.

**Details**

The strict 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. The inverse strict half-vectorization takes a vector and reconstructs a symmetric matrix such that `vechs2full(vechs(x))` is equal to `x` with zero along the diagonal if `x` is symmetric.

Note that very few vectors have the correct number of elements to construct a symmetric matrix. For example, vectors with 1, 3, 6, 10, and 15 elements can be used to make a symmetric matrix, but none of the other numbers between 1 and 15 can. An error is thrown if the number of elements in `x` cannot be used to make a symmetric matrix.

**See Also**

vech2full, vech, vechs, rvectorize, cvectorize
Examples

vechs2full(1:10)

matrix(1:16, 4, 4)
vechs(matrix(1:16, 4, 4))
vechs2full(vechs(matrix(1:16, 4, 4)))
Index

*Topic classes

MxExpectationGREML-class, 103
MxFitFunctionGREML-class, 131
MxMatrix-class, 154

*Topic datasets

Bollen, 8
demoOneFactor, 10
demoTwoFactor, 11
dzfData, 13
dzmData, 14
dzoData, 16
example1, 18
example2, 19
factorExample1, 20
factorScaleExample1, 21
factorScaleExample2, 22
HS.ability.data, 24
imxConstraintRelations, 27
imxDataTypes, 30
imxModelTypes, 43
imxReservedNames, 48
imxSeparatorChar, 49
imxVariableTypes, 53
jointdata, 57
latentMultipleRegExample1, 58
latentMultipleRegExample2, 59
multiData1, 60
myAutoregressiveData, 193
myFADatraw, 194
myGrowthKnownClassData, 194
myGrowthMixtureData, 195
myLongitudinalData, 196
myRegData, 197
myRegDatraw, 198
myTwinData, 199
mzfData, 200
mzmData, 201
nuclear_twin_design_data, 203
numHess1, 204

numHess2, 205
omxSymbolTable, 235
ordinalTwinData, 237
twin_NA_dot, 244
twinData, 242

[, MxMatrix-method (MxMatrix-class), 154
[<-, MxMatrix-method (MxMatrix-class), 154

[][, MxFlatModel-method
(MxFlatModel-class), 140
[[, MxModel-method (MxModel-class), 163
[<-, MxFlatModel-method
(MxFlatModel-class), 140
[<-, MxLISRELModel-method
(MxLISRELModel-class), 147
[[<-, MxModel-method (MxModel-class), 163
[[<-, MxRAMModel-method
(MxRAMModel-class), 172

$, BaseCompute-method
(BaseCompute-class), 7
$, MxAlgebra-method (MxAlgebra-class), 64
$, MxBaseExpectation-method
(MxBaseExpectation-class), 67
$, MxBaseFitFunction-method
(MxBaseFitFunction-class), 67
$, MxConstraint-method
(MxConstraint-class), 89
$, MxData-method (MxData-class), 93
$, MxFlatModel-method
(MxFlatModel-class), 140

$, MxInterval-method (MxInterval-class), 145
$, MxMatrix-method (MxMatrix-class), 154
$, MxModel-method (MxModel-class), 163
$, MxPath-method (MxPath), 169
$, MxThreshold-method (MxThreshold), 188
$<-, BaseCompute-method
(BaseCompute-class), 7
$<-, MxAlgebra-method (MxAlgebra-class), 250
INDEX

64
$<$-, MxBaseExpectation-method
(MxBaseExpectation-class), 67
$<$-, MxBaseFitFunction-method
(MxBaseFitFunction-class), 67
$<$-, MxConstraint-method
(MxConstraint-class), 89
$<$-, MxData-method (MxData-class), 93
$<$-, MxFlatModel-method
(MxFlatModel-class), 140
$<$-, MxInterval-method
(MxInterval-class), 145
$<$-, MxLISRELModel-method
(MxLISRELModel-class), 147
$<$-, MxMatrix-method (MxMatrix-class), 154
$<$-, MxModel-method (MxModel-class), 163
$<$-, MxPath-method (MxPath), 169
$<$-, MxRAMModel-method
(MxRAMModel-class), 172
$<$-, MxThreshold-method (MxThreshold-class), 188
%% (mxAlebra), 61
%*% (mxAlebra), 61
apply, 207
as.character, 166
as.list, 207, 222, 231
BaseCompute-class, 7
Bollen, 8
Classes, 65, 69, 90, 161, 163
cut, 140
cvectorize, 9, 62, 238, 242, 246–248
data.frame, 92, 94, 96
demoOneFactor, 10
demoTwoFactor, 11
detectCores, 166
diag, 12, 63, 245
diag2vec, 12, 63, 245
DiagMatrix-class (MxMatrix-class), 154
dim, MxMatrix-method (MxMatrix-class), 154
dimnames, 109, 126
dimnames, MxAlebra-method
(MxAlebra-class), 64
dimnames, MxMatrix-method
(MxMatrix-class), 154
dimnames<-, MxAlebra-method
(MxAlebra-class), 64
dimnames<-, MxMatrix-method
(MxMatrix-class), 154
dzfData, 13
dzmData, 14
dzoData, 16
eigen, 18
eigenval, 62
eigenval (eigenvec), 17
eigenvec, 17
example1, 18, 19
example2, 19, 19
expm, 20, 63
Extract, 73, 75, 110, 127, 133, 139
factor, 124, 125
factorExample1, 20
factorScaleExample1, 21, 23
factorScaleExample2, 22
FullMatrix-class (MxMatrix-class), 154
genericFitDependencies, MxBaseFitFunction-method,
23
here, 63, 66, 69, 74, 75, 92, 94, 96, 102, 104,
129, 131, 132, 144, 154, 156, 162,
165, 171, 189, 209
HS.ability.data, 24
HS.fake.data (HS.ability.data), 24
IdenMatrix-class (MxMatrix-class), 154
eigenval (eigenvec), 17
eigenvec (eigenvec), 17
imxAddDependency, 26
imxCheckMatrices, 26
imxCheckVariables, 27
imxConDecMatrixSlots, 27
imxConDecMatrixSlots, MxMatrix-method
(imxConDecMatrixSlots), 27
imxConstraintRelations, 27
imxConvertIdentifier, 28
imxConvertLabel, 28
imxConvertSubstitution, 29
imxCreateMatrix, 29, 156
imxCreateMatrix, DiagMatrix-method
(imxCreateMatrix), 29
imxCreateMatrix, FullMatrix-method
(imxCreateMatrix), 29
imxCreateMatrix, IdenMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, LowerMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, MxMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, SdiagMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, StandMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, SymmMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, UnitMatrix-method
  (imxCreateMatrix), 29
imxCreateMatrix, ZeroMatrix-method
  (imxCreateMatrix), 29
imxDataTypes, 30
imxDefaultGetSlotDisplayNames, 30
imxDeparse, IdenMatrix-method
  (imxDeparse), 31
imxDeparse, matrix-method (imxDeparse),
  31
imxDeparse, MxAlgebra-method
  (imxDeparse), 31
imxDeparse, MxConstraint-method
  (imxDeparse), 31
imxDeparse, MxData-method (imxDeparse),
  31
imxDeparse, MxMatrix-method
  (imxDeparse), 31
imxDeparse, UnitMatrix-method
  (imxDeparse), 31
imxDeparse, ZeroMatrix-method
  (imxDeparse), 31
imxIndependentModels, 31
imxDiff, 32
imxDmvnorm, 32
imxEvalByName, 33
imxExtractMethod, 33
imxExtractNames, 34
imxExtractReferences, 34
imxExtractSlot, 34
imxFilterDefinitionVariables, 35
imxFlattenModel, 35
imxFreezeModel, 35
imxGenerateLabels, 36
imxGenerateNamespace, 36
imxGenericModelBuilder, 36
imxGenSwift, 37
imxGetExpectationComponent
  (mxGetExpected), 142
imxGetSlotDisplayNames, 37
imxHasNPSOL, 38
imxHasOpenMP, 38
imxIdentifier, 39
imxIndependentModels, 39
imxInitModel, 39
imxInitModel, MxLISRELModel-method
  (imxInitModel), 39
imxInitModel, MxModel-method
  (imxInitModel), 39
imxInitModel, MxRAMModel-method
  (imxInitModel), 39
imxIsDefinitionVariable, 40
imxIsPath, 40
imxLocateFunction, 40
imxLocateIndex, 41
imxLocateLabel, 41
imxLog, 42
imxLookupSymbolTable, 42
imxModelBuilder, 42
imxModelBuilder, MxLISRELModel-method
  (imxModelBuilder), 42
imxModelBuilder, MxModel-method
  (imxModelBuilder), 42
imxModelBuilder, MxRAMModel-method
  (imxModelBuilder), 42
imxModelTypes, 43
imxMpiWrap, 43
imxOriginalMx, 44
imxPPML, 44
imxPPML.Test.Battery, 45
imxPPML.Test.Test, 46
imxPreprocessModel, 46
imxReplaceMethod, 47
imxReplaceModels, 47
imxReplaceSlot, 48
imxReservedNames, 48
imxReverseIdentifier, 49
imxSameType, 49
imxSeparatorChar, 49
imxSfClient, 50
imxSimpleRAMPredicate, 50
imxSparseInvert, 50
imxSquareMatrix, 51, 56
imxSquareMatrix, DiagMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, IdemMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, LowerMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, MxMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, SdiagMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, StandMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, SymmMatrix-method (imxSquareMatrix), 51
imxSquareMatrix, UnitMatrix-method (imxSquareMatrix), 51
imxVerifyMatrix, ZeroMatrix-method (imxVerifyMatrix), 54
imxVerifyModel, 54
imxVerifyModel, MxLISRELModel-method (imxVerifyModel), 54
imxVerifyModel, MxModel-method (imxVerifyModel), 54
imxVerifyModel, MxRAMModel-method (imxVerifyModel), 54
imxVerifyName, 54
imxVerifyReference, 55
imxWLSChisquare, 55
imxWLSStandardErrors, 56
jointdata, 57
lapply, 222
latentMultipleRegExample1, 58, 59
latentMultipleRegExample2, 59
length, MxMatrix-method (MxMatrix-class), 154
logm, 60, 63
LowerMatrix-class (MxMatrix-class), 154
make.names, 151
make.unique, 151
matrix, 92, 94, 96
Mod, 17
multiData, 60
MxAlgebra (MxAlgebra-class), 64
mxAlgebra, 61, 64–66, 91, 94, 98, 129, 153, 154, 161, 236
MxAlgebra-class, 64
MxAlgebraFormula (MxAlgebraFormula-class), 65
MxAlgebraFormula-class, 65
mxAlgebraObjective, 65, 161
MxAlgebras, 73, 75, 164
MxBaseExpectation-class, 67
MxBaseFitFunction-class, 67
MxBaseNamed (MxBaseNamed-class), 67
MxBaseNamed-class, 67
MxBaseObjectiveMetaData
  (MxBaseObjectiveMetaData-class), 68
MxBaseObjectiveMetaData-class, 68
MxBounds (MxBounds-class), 69
mxBounds, 68, 69, 153, 161, 236
MxBounds-class, 69
MxCharOrList-class, 70
MxCharOrNumber-class, 70
mxCheckIdentification, 70
MxCI, 72–74, 164
MxCI (MxCI-class), 74
mxCI, 72, 73–75, 145, 150–152
MxCI-class, 74
mxCmpare, 75
MxCompute (MxCompute-class), 78
MxCompute-class, 78
mxCmpComputeConfidenceInterval, 74, 78
MxComputeConfidenceInterval-class
  (mxCmpComputeConfidenceInterval), 78
mxCmpComputeDefault, 79
MxComputeDefault-class
  (mxCmpComputeDefault), 79
mxCmpComputeEM, 79
MxComputeEM-class (mxCmpComputeEM), 79
mxCmpComputeGradientDescent, 81
MxComputeGradientDescent-class
  (mxCmpComputeGradientDescent), 81
mxCmpComputeHessianQuality, 82
MxComputeHessianQuality-class
  (mxCmpComputeHessianQuality), 82
mxCmpComputeIterate, 83
MxComputeIterate-class
  (mxCmpComputeIterate), 83
mxCmpComputeNewtonRaphson, 83
MxComputeNewtonRaphson-class
  (mxCmpComputeNewtonRaphson), 83
mxCmpComputeNothing, 84
mxCmpComputeNumericDeriv, 84
MxComputeNumericDeriv-class
  (mxCmpComputeNumericDeriv), 84
mxCmpComputeOnce, 84
MxComputeOnce-class (mxCmpComputeOnce), 85
mxCmpComputeReportDeriv, 85
MxComputeReportDeriv-class
  (mxCmpComputeReportDeriv), 86
mxCmpComputeSequence, 87, 130
MxComputeSequence-class
  (mxCmpComputeSequence), 87
mxCmpComputeStandardError, 87
MxComputeStandardError-class
  (mxCmpComputeStandardError), 87
MxConstraint (MxConstraint-class), 89
mxCmConstraint, 64, 79, 88, 89, 90, 153, 154, 160, 161, 186, 236
MxConstraint-class, 89
MxData (MxData-class), 93
mxCmpData, 90, 93, 94, 107, 110, 112, 117, 122, 127, 138, 150, 159–162, 174, 236
MxData-class, 93
mxCmpDataDynamic, 94
MxDataDynamic-class (mxCmpDataDynamic), 94
MxDataFrameOrMatrix
  (MxDataFrameOrMatrix-class), 95
MxDataFrameOrMatrix-class, 95
MxDataStatic (MxDataStatic-class), 95
MxDataStatic-class, 95
mxCmpDataWLS, 95, 138
MxDirectedGraph
  (MxDirectedGraph-class), 97
MxDirectedGraph-class, 97
mxCmpEval, 66, 97, 107, 112, 117, 122, 129, 150, 159, 174
mxCmpEvalByName (mxCmpEval), 97
MxExpectation (MxExpectation-class), 99
MxExpectation-class, 99
mxCmpExpectationBA81, 67, 99, 233
MxExpectationBA81-class
  (mxCmpExpectationBA81), 99
MxExpectationGREML, 101, 102
INDEX

MxExpectationGREML (MxExpectationGREML-class), 103
mxFactor, 124, 140
mxFIMLObjective, 125, 161
MxFitFunction (MxFitFunction-class), 128
MxFitFunction-class, 128
mxFitFunctionAlgebra, 63, 66, 67, 91, 94, 128, 236
MxFitFunctionAlgebra-class (mxFitFunctionAlgebra), 128
MxFitFunctionGREML (MxFitFunctionGREML-class), 130, 131
MxFitFunctionGREML, 130, 131
MxFitFunctionGREML-class, 131
MxFitFunctionML, 67, 91, 93, 110, 114, 119, 126, 132, 157, 158, 173, 178, 236
MxFitFunctionML-class (mxFitFunctionML), 132
MxFitFunctionMultigroup, 67, 134, 232
MxFitFunctionMultigroup-class (mxFitFunctionMultigroup), 134
MxFitFunctionR, 67, 135, 236

MxFitfunctionR-class (mxFitFunctionR), 135
mxFitFunctionRow, 67, 136, 236
MxFitFunctionRow-class (mxFitFunctionRow), 136
mxFitFunctionWLS, 67, 91, 95, 96, 138
MxFitFunctionWLS-class (mxFitFunctionWLS), 138
MxFlatModel-class, 140
mxGenerateData, 140
mxGetExpected, 142
mxGREMLDataHandler, 101, 143
MxInterval-class, 145
mxKalmanScores, 145
MxLISRELModel-class, 147
mxLISRELObjective, 147
MxListOrNull-class, 151
mxMakeNames, 151
MxMatrices, 73, 75, 164
MxMatrix (MxMatrix-class), 154
mxMatrix, 63, 65, 69, 74, 91, 94, 109, 126, 152, 154, 156, 160–162, 171, 186, 189, 236
MxMatrix-class, 154
mxMI, 156
mxMLObjective, 158, 161
MxModel (MxModel-class), 163
MxModel-class, 163
MxNonNullable-class (MxData-class), 93
mxOption, 162, 165, 165, 177, 183, 186
MxOptionalChar-class, 168
MxOptionalCharOrNumber-class, 168
MxOptionalLogical-class, 168
MxOptionalMatrix-class, 169
MxOptionalNumeric-class, 169
INDEX

MxPath, 91, 94, 164, 170
mxPath, 160–162, 164, 169, 186, 188, 189, 236
MxPath-class (mxPath), 169
MxRAMGraph (MxRAMGraph-class), 172
MxRAMGraph-class, 172
MxRAMMetaData-class, 172
MxRAMModel-class, 172
mxRAMObjective, 150, 161, 173
mxRefModels, 134, 240
mxRefModels (omxSaturatedModel), 232
mxRename, 175
mxRestore, 176
mxRObjective, 178
mxSave, 183
mxSetDefaultOptions, 184
mxSimplify2Array, 185
mxStandardizeRAMpaths, 185, 186
mxSummary, 73, 232
mxSummary (summary-MxModel), 239
MxThreshold, 188, 189
MxThreshold-class (MxThreshold), 188
mxTryHard, 189
mxTypes, 161, 191
mxVersion, 192
myAutoregressiveData, 193
myFADataRaw, 194
myGrowthKnownClassData, 194, 195
myGrowthMixtureData, 195, 195
myLongitudinalData, 196
myRegData, 197, 198
myRegDataRaw, 198
myTwinData, 199, 244
mzfData, 200
mzmData, 201

Named entities, 161, 165
named entity, 64, 74, 89, 93, 154, 163
Named-entities (Named-entity), 203
named-entities, 161
named-entities (Named-entity), 203
Named-entity, 203
named-entity (Named-entity), 203
names, 231

names, MxFlatModel-method
(MxFlatModel-class), 140
names, MxMatrix-method (MxMatrix-class), 154
names, MxModel-method (MxModel-class), 163
ncol, MxMatrix-method (MxMatrix-class), 154
nrow, MxMatrix-method (MxMatrix-class), 154
nuclear_twin_design_data, 203
NULL, 93
numHess1, 204
numHess2, 205
omxAllInt, 63, 205
omxAnd, 63
omxAnd (omxLogical), 224
omxApply, 207, 223, 231
omxApproxEquals, 63
omxApproxEquals (omxLogical), 224
omxAssignFirstParameters, 208, 220, 223, 234, 235
omxBrownie, 209
omxBind (omxMatrixOperations), 225
omxCheckCloseEnough, 210, 211, 213, 215, 217
omxCheckEquals, 210, 211, 212, 213, 215–217
omxCheckError, 212, 216
omxCheckIdentical, 210–212, 213, 215–217
omxCheckNamespace, 214
omxCheckSetEquals, 210–213, 214, 215–217
omxCheckTrue, 210–213, 215, 216, 217
omxCheckWarning, 212, 216
omxCheckWithinPercentError, 210–213, 215, 216, 217
omxConstrainMLThresholds, 218
omxDetectCores, 218
omxExponential, 63
omxExponential (expm), 20
omxGetNPSOL, 219
omxGetParameters, 190, 208, 219, 223, 235
omxGetRAMDepth, 221
omxGraphviz, 222
omxGreaterThan, 63
omxGreaterThan (omxLogical), 224
omxLapply, 207, 222, 231
omxLessThan, 63
show.mXDataDynamic-method
  (mXDataDynamic), 94
show.mXDataStatic-method
  (mXDataStatic-class), 95
show.mXExpectationBA81-method
  (mXExpectationBA81), 99
show.mXExpectationLISREL-method
  (mXExpectationLISREL), 104
show.mXExpectationNormal-method
  (mXExpectationNormal), 109
show.mXExpectationRAM-method
  (mXExpectationRAM), 111
show.mXExpectationStateSpace-method
  (mXExpectationStateSpace), 114
show.mXFitFunctionAlgebra-method
  (mXFitFunctionAlgebra), 128
show.mXFitFunctionML-method
  (mXFitFunctionML), 132
show.mXFitFunctionR-method
  (mXFitFunctionR), 135
show.mXFitFunctionRow-method
  (mXFitFunctionRow), 136
show.mXFitFunctionWLS-method
  (mXFitFunctionWLS), 138
show.mXFlatModel-method
  (mXFlatModel-class), 140
show.mXInterval-method
  (mXInterval-class), 145
show.mXMatrix-method (mXMatrix-class), 154
show.mXModel-method (mXModel-class), 163
show.mXNonNullData-method
  (mXData-class), 93
show.mXPath-method (mXPath), 169
show.mXThreshold-method (mXThreshold), 188
StandMatrix-class (mXMatrix-class), 154
summary, 72, 110, 127, 133, 139
summary (summary-MxModel), 239
summary, mXModel-method
  (summary-MxModel), 239
summary-MxModel, 239
SymmMatrix-class (mXMatrix-class), 154

tr, 242
twin.na.dot, 244
twinData, 242

UnitMatrix-class (mXMatrix-class), 154

vec2diag, 12, 63, 245
vech, 9, 62, 238, 242, 246, 247, 248
vech2full, 63, 246, 246, 248
vechs, 9, 62, 238, 246, 247, 247, 248
vechs2full, 63, 247, 248

ZeroMatrix-class (mXMatrix-class), 154