

# Package ‘mdatools’

May 8, 2026

**Version** 0.15.0

**Title** Multivariate Data Analysis for Chemometrics

**Date** 2026-03-06

**Description** Projection based methods for preprocessing, exploring and analysis of multivariate data used in chemometrics. S. Kucheryavskiy (2020) <[doi:10.1016/j.chemolab.2020.103937](https://doi.org/10.1016/j.chemolab.2020.103937)>.

**Encoding** UTF-8

**License** MIT + file LICENSE

**Imports** methods, graphics, grDevices, stats, spam, pcv

**RoxygenNote** 7.3.3

**Suggests** testthat

**NeedsCompilation** no

**Depends** R (>= 3.5.0)

**URL** <https://mda.tools>

**BugReports** <https://github.com/svkucheryavski/mdatools/issues>

**Author** Sergey Kucheryavskiy [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-3145-7244>>)

**Maintainer** Sergey Kucheryavskiy <[svkucheryavski@gmail.com](mailto:svkucheryavski@gmail.com)>

**Repository** CRAN

**Date/Publication** 2026-03-06 10:10:10 UTC

## Contents

arr2int . . . . .	12
as.data.frame.ddsimcares . . . . .	12
as.matrix.classres . . . . .	13
as.matrix.ddsimcares . . . . .	14
as.matrix.ldecomp . . . . .	14
as.matrix.plsdares . . . . .	15
as.matrix.plsres . . . . .	15
as.matrix.regcoeffs . . . . .	16

as.matrix.regres . . . . .	16
as.matrix.simcamres . . . . .	17
as.matrix.simcares . . . . .	17
asjson . . . . .	18
asjson.ddsimca . . . . .	18
asjson.pca . . . . .	19
asjson.pls . . . . .	19
asvector . . . . .	20
asvector.pca . . . . .	20
asvector.pls . . . . .	21
capitalize . . . . .	21
carbs . . . . .	22
categorize . . . . .	23
categorize.pca . . . . .	23
categorize.pls . . . . .	24
chisq.crit . . . . .	25
chisq.prob . . . . .	25
clamp.dof . . . . .	26
classify . . . . .	26
classify.plsda . . . . .	27
classify.simca . . . . .	28
classmodel.processRefValues . . . . .	28
classres . . . . .	29
classres.getPerformance . . . . .	30
cleanLabels . . . . .	31
confint.regcoeffs . . . . .	31
constraint . . . . .	32
constraintAngle . . . . .	32
constraintClosure . . . . .	33
constraintNonNegativity . . . . .	33
constraintNorm . . . . .	34
constraints.list . . . . .	34
constraintUnimod . . . . .	34
create_categories . . . . .	35
crossval . . . . .	35
crossval.getParams . . . . .	36
crossval.regmodel . . . . .	36
crossval.simca . . . . .	37
crossval.str . . . . .	37
dd.crit . . . . .	38
ddmoments.param . . . . .	38
ddrobust.param . . . . .	39
ddsimca . . . . .	39
ddsimca.fromjson . . . . .	42
ddsimca.readJSON . . . . .	43
ddsimcares . . . . .	43
ellipse . . . . .	45
employ.constraint . . . . .	46

employ.prep . . . . .	46
extractArray . . . . .	47
extractBlock . . . . .	47
extractPrep . . . . .	48
extractStringArray . . . . .	48
extractValue . . . . .	49
fprintf . . . . .	49
genhash . . . . .	50
getCalibrationData . . . . .	50
getCalibrationData.pca . . . . .	51
getCalibrationData.simcam . . . . .	51
getConfidenceEllipse . . . . .	52
getConfusionMatrix . . . . .	52
getConfusionMatrix.classres . . . . .	53
getConvexHull . . . . .	53
getDataLabels . . . . .	54
getImplementedConstraints . . . . .	54
getImplementedPrepMethods . . . . .	54
getLabelsAsIndices . . . . .	55
getLabelsAsValues . . . . .	55
getMainTitle . . . . .	55
getPlotColors . . . . .	56
getProbabilities . . . . .	56
getProbabilities.pca . . . . .	57
getProbabilities.simca . . . . .	57
getPureVariables . . . . .	58
getRegcoeffs . . . . .	58
getRegcoeffs.regmodel . . . . .	59
getRes . . . . .	60
getSelectedComponents . . . . .	60
getSelectivityRatio . . . . .	61
getSelectivityRatio.pls . . . . .	61
getVariance.mcr . . . . .	62
getVIPScores . . . . .	62
getVIPScores.pls . . . . .	63
hotelling.crit . . . . .	63
hotelling.prob . . . . .	64
imshow . . . . .	64
ipls . . . . .	65
ipls.backward . . . . .	67
ipls.forward . . . . .	68
jm.crit . . . . .	69
jm.prob . . . . .	69
ldcomp . . . . .	70
ldcomp.getDistances . . . . .	71
ldcomp.getLimitsCoordinates . . . . .	71
ldcomp.getLimParams . . . . .	72
ldcomp.getQLimits . . . . .	72

ldecomp.getT2Limits . . . . .	73
ldecomp.getVariances . . . . .	73
ldecomp.plotDistances . . . . .	74
ldecomp.plotResiduals . . . . .	75
mcr . . . . .	76
mcrals . . . . .	76
mcrals.cal . . . . .	81
mcrals.fcnpls . . . . .	82
mcrals.npls . . . . .	83
mcrals.ols . . . . .	83
mcrpure . . . . .	84
mda.cbind . . . . .	87
mda.data2im . . . . .	87
mda.df2mat . . . . .	88
mda.exclcols . . . . .	88
mda.exclrows . . . . .	89
mda.getattr . . . . .	89
mda.getexclind . . . . .	90
mda.im2data . . . . .	90
mda.inclcols . . . . .	91
mda.inclrows . . . . .	91
mda.purge . . . . .	92
mda.purgeCols . . . . .	92
mda.purgeRows . . . . .	92
mda.rbind . . . . .	93
mda.setattr . . . . .	93
mda.setimbg . . . . .	94
mda.show . . . . .	94
mda.subset . . . . .	95
mda.t . . . . .	95
mdaplot . . . . .	96
mdaplot.areColors . . . . .	99
mdaplot.formatValues . . . . .	99
mdaplot.getColors . . . . .	100
mdaplot.getXAxisLim . . . . .	101
mdaplot.getXTickLabels . . . . .	101
mdaplot.getXTicks . . . . .	102
mdaplot.getYAxisLim . . . . .	102
mdaplot.getYTickLabels . . . . .	103
mdaplot.getYTicks . . . . .	103
mdaplot.plotAxes . . . . .	104
mdaplot.prepareColors . . . . .	105
mdaplot.showColorbar . . . . .	105
mdaplot.showLines . . . . .	106
mdaplotg . . . . .	106
mdaplotg.getLegend . . . . .	109
mdaplotg.getXLim . . . . .	109
mdaplotg.getYLim . . . . .	110

mdaplotg.prepareData . . . . .	111
mdaplotg.processParam . . . . .	111
mdaplotg.showLegend . . . . .	112
mdaplotyy . . . . .	113
mdatools . . . . .	115
paste1 . . . . .	116
pca . . . . .	116
pca.cal . . . . .	121
pca.fromjson . . . . .	122
pca.getB . . . . .	122
pca.mvreplace . . . . .	123
pca.nipals . . . . .	124
pca.readJSON . . . . .	125
pca.run . . . . .	125
pca.svd . . . . .	126
pca.syncResAliases . . . . .	126
pcares . . . . .	127
pellets . . . . .	129
people . . . . .	129
pinv . . . . .	130
plot.classres . . . . .	130
plot.ddsimca . . . . .	131
plot.ddsimcares . . . . .	131
plot.ipls . . . . .	132
plot.mcr . . . . .	132
plot.pca . . . . .	133
plot.pcares . . . . .	133
plot.pls . . . . .	134
plot.plsda . . . . .	135
plot.plsdares . . . . .	135
plot.plsres . . . . .	136
plot.randtest . . . . .	136
plot.regcoeffs . . . . .	137
plot.regres . . . . .	138
plot.simca . . . . .	138
plot.simcam . . . . .	139
plot.simcamres . . . . .	139
plotAcceptance . . . . .	140
plotAcceptance.ddsimca . . . . .	140
plotAcceptance.ddsimcares . . . . .	141
plotAliens . . . . .	142
plotAliens.ddsimcares . . . . .	142
plotBars . . . . .	143
plotBiplot . . . . .	144
plotBiplot.pca . . . . .	144
plotConfidenceEllipse . . . . .	145
plotContributions . . . . .	146
plotContributions.mcr . . . . .	147

plotConvexHull . . . . .	147
plotCooman . . . . .	148
plotCooman.simcam . . . . .	148
plotCooman.simcamres . . . . .	149
plotCorr . . . . .	150
plotCorr.randtest . . . . .	151
plotCumVariance . . . . .	151
plotCumVariance.ldecomp . . . . .	152
plotCumVariance.mcr . . . . .	152
plotCumVariance.pca . . . . .	153
plotDensity . . . . .	153
plotDiscriminationPower . . . . .	154
plotDiscriminationPower.simcam . . . . .	154
plotDistances . . . . .	155
plotDistances.ddsimca . . . . .	156
plotDistances.ddsimcares . . . . .	157
plotDistances.ldecomp . . . . .	158
plotDistances.pca . . . . .	159
plotDistDoF . . . . .	160
plotEigenvalues . . . . .	161
plotEigenvalues.pca . . . . .	161
plotErrorbars . . . . .	162
plotExtreme . . . . .	163
plotExtreme.ddsimca . . . . .	163
plotExtreme.ddsimcares . . . . .	164
plotExtreme.pca . . . . .	164
plotExtremes . . . . .	165
plotExtremes.ddsimca . . . . .	165
plotExtremes.ddsimcares . . . . .	166
plotExtremes.pca . . . . .	167
plotFoM . . . . .	168
plotFoM.ddsimcares . . . . .	168
plotFoMs . . . . .	169
plotFoMs.ddsimcares . . . . .	169
plotHist . . . . .	170
plotHist.randtest . . . . .	171
plotHotellingEllipse . . . . .	171
plotLines . . . . .	172
plotLoadings . . . . .	173
plotLoadings.pca . . . . .	174
plotMisclassified . . . . .	174
plotMisclassified.classmodel . . . . .	175
plotMisclassified.classres . . . . .	175
plotModelDistance . . . . .	176
plotModelDistance.simcam . . . . .	176
plotModellingPower . . . . .	178
plotPerformance . . . . .	179
plotPerformance.classmodel . . . . .	179

plotPerformance.classres . . . . .	180
plotPointsShape . . . . .	181
plotPredictions . . . . .	182
plotPredictions.classmodel . . . . .	182
plotPredictions.classres . . . . .	183
plotPredictions.regmodel . . . . .	184
plotPredictions.regres . . . . .	185
plotPredictions.simcam . . . . .	186
plotPredictions.simcamres . . . . .	186
plotProbabilities . . . . .	187
plotProbabilities.classres . . . . .	187
plotPurity . . . . .	188
plotPurity.mcrpure . . . . .	189
plotPuritySpectra . . . . .	189
plotPuritySpectra.mcrpure . . . . .	190
plotQDoF . . . . .	191
plotRegcoeffs . . . . .	191
plotRegcoeffs.regmodel . . . . .	192
plotRegressionLine . . . . .	192
plotResiduals . . . . .	193
plotResiduals.ldecomp . . . . .	193
plotResiduals.pca . . . . .	194
plotResiduals.regres . . . . .	194
plotRMSE . . . . .	195
plotRMSE.ipls . . . . .	195
plotRMSE.regmodel . . . . .	196
plotRMSE.regres . . . . .	197
plotRMSERatio . . . . .	198
plotRMSERatio.regmodel . . . . .	198
plotScatter . . . . .	199
plotScores . . . . .	200
plotScores.ldecomp . . . . .	200
plotScores.pca . . . . .	201
plotSelection . . . . .	202
plotSelection.ipls . . . . .	202
plotSelectivityArea . . . . .	203
plotSelectivityArea.ddsimcares . . . . .	204
plotSelectivityRatio . . . . .	205
plotSelectivityRatio.pls . . . . .	205
plotSensitivity . . . . .	206
plotSensitivity.classmodel . . . . .	206
plotSensitivity.classres . . . . .	207
plotSensitivity.ddsimca . . . . .	207
plotSensitivity.ddsimcares . . . . .	208
plotseries . . . . .	209
plotSpecificity . . . . .	210
plotSpecificity.classmodel . . . . .	210
plotSpecificity.classres . . . . .	211

plotSpectra . . . . .	211
plotSpectra.mcr . . . . .	212
plotT2DoF . . . . .	212
plotVariance . . . . .	213
plotVariance.ldecomp . . . . .	214
plotVariance.mcr . . . . .	214
plotVariance.pca . . . . .	215
plotVariance.pls . . . . .	216
plotVariance.plsres . . . . .	217
plotVIPScores . . . . .	217
plotVIPScores.pls . . . . .	218
plotWeights . . . . .	218
plotWeights.pls . . . . .	219
plotXCumVariance . . . . .	220
plotXCumVariance.pls . . . . .	220
plotXCumVariance.plsres . . . . .	221
plotXLoadings . . . . .	221
plotXLoadings.pls . . . . .	222
plotXResiduals . . . . .	222
plotXResiduals.pls . . . . .	223
plotXResiduals.plsres . . . . .	224
plotXScores . . . . .	225
plotXScores.pls . . . . .	225
plotXScores.plsres . . . . .	226
plotXVariance . . . . .	227
plotXVariance.pls . . . . .	227
plotXVariance.plsres . . . . .	228
plotXYLoadings . . . . .	228
plotXYLoadings.pls . . . . .	229
plotXYResiduals . . . . .	229
plotXYResiduals.pls . . . . .	230
plotXYResiduals.plsres . . . . .	231
plotXYScores . . . . .	232
plotXYScores.pls . . . . .	232
plotXYScores.plsres . . . . .	233
plotYCumVariance . . . . .	234
plotYCumVariance.pls . . . . .	234
plotYCumVariance.plsres . . . . .	235
plotYResiduals . . . . .	235
plotYResiduals.plsres . . . . .	236
plotYResiduals.regmodel . . . . .	236
plotYVariance . . . . .	237
plotYVariance.pls . . . . .	237
plotYVariance.plsres . . . . .	238
pls . . . . .	239
pls.cal . . . . .	245
pls.fromjson . . . . .	246
pls.getLimitsCoordinates . . . . .	246

pls.getpredictions	247
pls.getxdecomp	248
pls.getxscores	249
pls.getydecomp	249
pls.getyscores	250
pls.getZLimits	251
pls.readJSON	251
pls.run	252
pls.simpls	252
pls.syncResAliases	253
plsda	253
plsdares	257
plsres	259
predict.ddsimca	262
predict.mcrals	263
predict.mcrpure	264
predict.pca	264
predict.pls	265
predict.plsda	265
predict.simca	266
predict.simcam	267
prep	267
prep.alsbasecorr	268
prep.alsbasecorr.asjson	269
prep.alsbasecorr.fromjson	270
prep.apply	270
prep.asjson	271
prep.autoscale	271
prep.center	272
prep.center.asjson	272
prep.center.fromjson	273
prep.center.params	273
prep.emsc	274
prep.emsc.asjson	274
prep.emsc.fromjson	275
prep.emsc.params	275
prep.fit	276
prep.fromjson	276
prep.generic	277
prep.list	277
prep.msc	277
prep.norm	278
prep.norm.asjson	279
prep.norm.fromjson	280
prep.norm.params	280
prep.ref2km	281
prep.savgol	281
prep.savgol.asjson	282

prep.savgol.fromjson . . . . .	282
prep.savgol.params . . . . .	283
prep.scale . . . . .	284
prep.scale.asjson . . . . .	284
prep.scale.fromjson . . . . .	285
prep.scale.params . . . . .	285
prep.snv . . . . .	286
prep.spikes . . . . .	287
prep.spikes.asjson . . . . .	287
prep.spikes.fromjson . . . . .	288
prep.transform . . . . .	288
prep.varsel . . . . .	289
prep.varsel.asjson . . . . .	290
prep.varsel.fromjson . . . . .	290
preparePlotData . . . . .	291
prepCalData . . . . .	291
print.classres . . . . .	292
print.ddsimca . . . . .	292
print.ddsimcares . . . . .	293
print.ipls . . . . .	293
print.ldecomp . . . . .	294
print.mcrals . . . . .	294
print.mcrpure . . . . .	295
print.pca . . . . .	295
print.pcares . . . . .	296
print.pls . . . . .	296
print.plsda . . . . .	297
print.plsdares . . . . .	297
print.plsres . . . . .	298
print.prepmodel . . . . .	298
print.randtest . . . . .	299
print.regcoeffs . . . . .	299
print.regmodel . . . . .	300
print.regres . . . . .	300
print.simca . . . . .	301
print.simcam . . . . .	301
print.simcamres . . . . .	302
print.simcares . . . . .	302
processLimType . . . . .	303
processMembers . . . . .	303
processStrangers . . . . .	304
randtest . . . . .	304
readJSON . . . . .	306
regcoeffs . . . . .	307
regcoeffs.getStats . . . . .	308
regres . . . . .	308
regres.bias . . . . .	309
regres.err . . . . .	309

regres.r2 . . . . .	310
regres.rmse . . . . .	310
regres.slope . . . . .	310
regress.addattrs . . . . .	311
repmat . . . . .	311
selectCompNum . . . . .	312
selectCompNum.pca . . . . .	312
selectCompNum.pls . . . . .	313
selratio . . . . .	314
setDistanceLimits . . . . .	314
setDistanceLimits.pca . . . . .	315
setDistanceLimits.pls . . . . .	316
setParams . . . . .	317
setParams.ddsimca . . . . .	317
showDistanceLimits . . . . .	318
showLabels . . . . .	318
showPredictions . . . . .	319
showPredictions.classres . . . . .	319
simca . . . . .	320
simcam . . . . .	323
simcam.getPerformanceStats . . . . .	325
simcamres . . . . .	325
simcares . . . . .	327
simdata . . . . .	329
splitExcludedData . . . . .	330
splitPlotData . . . . .	330
summary.classres . . . . .	331
summary.ddsimca . . . . .	331
summary.ddsimcares . . . . .	332
summary.ipls . . . . .	332
summary.ldecomp . . . . .	333
summary.mcrals . . . . .	333
summary.mcrpure . . . . .	334
summary.pca . . . . .	334
summary.pcares . . . . .	335
summary.pls . . . . .	335
summary.plsda . . . . .	336
summary.plsdares . . . . .	336
summary.plsres . . . . .	337
summary.prepmodel . . . . .	337
summary.randtest . . . . .	338
summary.regcoeffs . . . . .	338
summary.regmodel . . . . .	339
summary.regres . . . . .	339
summary.simca . . . . .	340
summary.simcam . . . . .	340
summary.simcamres . . . . .	341
summary.simcares . . . . .	341

unmix.mcrpure . . . . .	342
vipscores . . . . .	342
writeCSV . . . . .	343
writeCSV.ddsimcares . . . . .	344
writeCSV.pcares . . . . .	345
writeCSV.plsres . . . . .	345
writeJSON . . . . .	346
writeJSON.ddsimca . . . . .	346
writeJSON.pca . . . . .	347
writeJSON.pls . . . . .	347
writeJSON.prepmodel . . . . .	348

<b>Index</b>	<b>349</b>
--------------	------------

---

arr2int	<i>Convert a vector of integers to a compact interval string</i>
---------	--

---

### Description

Takes a vector of integer values, sorts them, groups consecutive values into intervals and returns a human-readable string. For example, `c(1, 5, 6, 7, 8, 20, 23, 24, 25)` becomes "1, 5-8, 20, 23-25".

### Usage

```
arr2int(x)
```

### Arguments

`x` a numeric vector of integer values.

### Value

a character string with the compact representation.

---

as.data.frame.ddsimcares	<i>Creates a data frame from DD-SIMCA classification results.</i>
--------------------------	---

---

### Description

The data frame contains object label, class label (if provided), role, decision, as well as values for distances ( $h/h_0$ ,  $q/q_0$ ,  $f$ ) for every object from dataset used to create the results object.

**Usage**

```
## S3 method for class 'ddsimcares'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  ncomp = x$ncomp.selected,
  limType = "classic"
)
```

**Arguments**

x	classification results (object of class ddsimcares).
row.names	NULL or a character vector giving the row names for the data frame (not used).
optional	logical, if TRUE, setting row names is optional (not used).
...	other arguments
ncomp	model complexity (number of components) to compute the classification results for.
limType	limit type to use ('classic' or 'robust')

**Value**

a data frame.

---

as.matrix.classres      *as.matrix method for classification results*

---

**Description**

Generic as.matrix function for classification results. Returns matrix with performance values for specific class.

**Usage**

```
## S3 method for class 'classres'
as.matrix(x, ncomp = NULL, nc = 1, ...)
```

**Arguments**

x	classification results (object of class plsdares, simcamres, etc.).
ncomp	model complexity (number of components) to show the parameters for.
nc	if there are several classes, which class to show the parameters for.
...	other arguments

---

as.matrix.ddsimcares *Creates a matrix from DD-SIMCA classification results.*

---

### Description

The matrix contains results characteristics computed for different number of components, including number of objects accepted/rejected by the model, explained and cumulative explained variance, as well as (if reference classes are provided) main figures of merits (TP, FP, TN, FN, sensitivity, specificity, efficiency and selectivity).

If dataset used to create the results contained only target class members or only objects from alternative classes, the FoMs will be reduced accordingly.

### Usage

```
## S3 method for class 'ddsimcares'
as.matrix(x, limType = "classic", ...)
```

### Arguments

x	classification results (object of class ddsimcares).
limType	limit type to use ('classic' or 'robust')
...	other arguments

### Value

a matrix.

---

as.matrix.ldecomp *as.matrix method for ldecomp object*

---

### Description

Generic as.matrix function for linear decomposition. Returns a matrix with information about the decomposition.

### Usage

```
## S3 method for class 'ldecomp'
as.matrix(x, ncomp = NULL, ...)
```

### Arguments

x	object of class ldecomp
ncomp	number of components to get the result for (if NULL will return for each available)
...	other arguments

---

as.matrix.plsdares      *as.matrix method for PLS-DA results*

---

### Description

Returns a matrix with model performance statistics for PLS-DA results

### Usage

```
## S3 method for class 'plsdares'  
as.matrix(x, ncomp = NULL, nc = 1, ...)
```

### Arguments

x	PLS-DA results (object of class plsdares)
ncomp	number of components to calculate the statistics for (if NULL gets for all components)
nc	for which class to calculate the statistics for
...	other arguments

---

as.matrix.plsres      *as.matrix method for PLS results*

---

### Description

Returns a matrix with model performance statistics for PLS results

### Usage

```
## S3 method for class 'plsres'  
as.matrix(x, ncomp = NULL, ny = 1, ...)
```

### Arguments

x	PLS results (object of class plsres)
ncomp	number of components to calculate the statistics for
ny	for which response variable calculate the statistics for
...	other arguments

---

as.matrix.regcoeffs     *as.matrix method for regression coefficients class*

---

### Description

returns matrix with regression coefficients for given response number and amount of components

### Usage

```
## S3 method for class 'regcoeffs'
as.matrix(x, ncomp = 1, ny = 1, ...)
```

### Arguments

x	regression coefficients object (class regcoeffs)
ncomp	number of components to return the coefficients for
ny	number of response variable to return the coefficients for
...	other arguments

---

as.matrix.regres     *as.matrix method for regression results*

---

### Description

Returns a matrix with model performance statistics for regression results

### Usage

```
## S3 method for class 'regres'
as.matrix(x, ncomp = NULL, ny = 1, ...)
```

### Arguments

x	regression results (object of class regres)
ncomp	model complexity (number of components) to calculate the statistics for (can be a vector)
ny	for which response variable calculate the statistics for
...	other arguments

---

as.matrix.simcamres     *as.matrix method for SIMCAM results*

---

### Description

Generic as.matrix function for SIMCAM results. Returns matrix with performance values for specific class.

### Usage

```
## S3 method for class 'simcamres'  
as.matrix(x, nc = seq_len(x$nclasses), ...)
```

### Arguments

x	classification results (object of class simcamres).
nc	vector with classes to use.
...	other arguments

---

as.matrix.simcares     *as.matrix method for SIMCA classification results*

---

### Description

Generic as.matrix function for classification results. Returns matrix with performance values for specific class.

### Usage

```
## S3 method for class 'simcares'  
as.matrix(x, ncomp = NULL, ...)
```

### Arguments

x	classification results (object of class simcares).
ncomp	model complexity (number of components) to show the parameters for.
...	other arguments

---

asjson	<i>S3 implementation of asjson() method</i>
--------	---

---

**Description**

S3 implementation of asjson() method

**Usage**

```
asjson(obj, ...)
```

**Arguments**

obj	object of any class, e.g. pca
...	other parameters relevant for the method

---

asjson.ddsimca	<i>Converts object with DD-SIMCA model to JSON string compatible with web-application.</i>
----------------	--

---

**Description**

Converts object with DD-SIMCA model to JSON string compatible with web-application.

**Usage**

```
## S3 method for class 'ddsimca'
asjson(
  obj,
  limType = "classic",
  alpha = obj$alpha,
  gamma = obj$gamma,
  ncomp = obj$ncomp.selected,
  ...
)
```

**Arguments**

obj	Object with DD-SIMCA model (from <a href="#">ddsimca</a> ).
limType	type of critical limits to use ('classic' or 'robust').
alpha	significance level for decision boundary.
gamma	significance level for outlier detection boundary.
ncomp	number of selected components.
...	other arguments

**Value**

stringified JSON

---

asjson.pca	<i>Converts object with PCA model to JSON string compatible with web-application.</i>
------------	---

---

**Description**

Converts object with PCA model to JSON string compatible with web-application.

**Usage**

```
## S3 method for class 'pca'
asjson(obj, ...)
```

**Arguments**

obj	Object with PCA model (from <a href="#">pca</a> ).
...	other arguments

**Value**

stringified JSON

---

asjson.pls	<i>Converts object with PLS model to JSON string compatible with web-application.</i>
------------	---

---

**Description**

Converts object with PLS model to JSON string compatible with web-application.

**Usage**

```
## S3 method for class 'pls'
asjson(obj, ...)
```

**Arguments**

obj	Object with PLS model (from <a href="#">pls</a> ).
...	other arguments

**Value**

stringified JSON

---

asvector	<i>S3 implementation of as.vector() method</i>
----------	--

---

**Description**

S3 implementation of as.vector() method

**Usage**

```
asvector(obj)
```

**Arguments**

obj            object of any class, e.g. pca

---

asvector.pca	<i>Converts object with PCA model to numeric vector compatible with web-application.</i>
--------------	--

---

**Description**

Converts object with PCA model to numeric vector compatible with web-application.

**Usage**

```
## S3 method for class 'pca'  
asvector(obj)
```

**Arguments**

obj            Object with PCA model (from [pca](#)).

**Value**

vector with values.

---

asvector.pls	<i>Converts object with PLS model to numeric vector compatible with web-application.</i>
--------------	--

---

**Description**

Converts object with PLS model to numeric vector compatible with web-application.

**Usage**

```
## S3 method for class 'pls'  
asvector(obj)
```

**Arguments**

obj            Object with PLS model (from [pls](#)).

**Value**

vector with values.

---

capitalize	<i>Capitalize text or vector with text values</i>
------------	---

---

**Description**

Capitalize text or vector with text values

**Usage**

```
capitalize(str)
```

**Arguments**

str            text or vector with text values

---

carbs

*Raman spectra of carbohydrates*

---

### Description

The dataset consists of Raman spectra of fructose, lactose, and ribose as well as spectra of their mixtures.

### Usage

```
data(simdata)
```

### Format

The data is a list (carbs) with the following fields:

- \$D a matrix (21x1401) with spectral values for the mixtures.
- \$S a matrix (1401x3) with spectral values for the pure components.
- \$C a matrix (21x3) with concentration of the pure components.

### Details

The dataset consists of Raman spectra of fructose, lactose, and ribose as well as spectra of their mixtures. The original spectra were downloaded from publicly available SPECARB library [1], created by S.B. Engelsen. The spectra were truncated to the range from 200 to 1600 cm<sup>-1</sup>.

The spectra of mixtures were created by linear combinations of the original spectra:

$$D = CS' + E$$

Concentrations of the components, C, follow a simplex lattice design with four levels. Some noise calculated as a random number uniformly distributed between 0% and 3% of maximum initial intensity (E) was added to each spectrum of the dataset, D, individually.

### References

1. Engelsen S.B., Database on Raman spectra of carbohydrates. Available at: <http://www.models.life.ku.dk/~specarb/specarb> [visited 31.05.2020]

---

categorize	<i>Categorize PCA results</i>
------------	-------------------------------

---

**Description**

Categorize PCA results

**Usage**

```
categorize(obj, ...)
```

**Arguments**

obj	object with PCA model
...	other parameters

---

categorize.pca	<i>Categorize PCA results based on orthogonal and score distances.</i>
----------------	--

---

**Description**

The method compares score and orthogonal distances of PCA results from `res` with critical limits computed for the PCA model and categorizes the corresponding objects as "regular", "extreme" or "outlier".

**Usage**

```
## S3 method for class 'pca'
categorize(obj, res = obj$res$cal, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj	object with PCA model
res	object with PCA results
ncomp	number of components to use for the categorization
...	other parameters

**Details**

The method does not categorize hidden values if any.

**Value**

vector (factor) with results of categorization.

---

categorize.pls	<i>Categorize data rows based on PLS results and critical limits for total distance.</i>
----------------	--

---

### Description

The method uses full distance for decomposition of X-data and squared Y-residuals of PLS results from res with critical limits computed for the PLS model and categorizes the corresponding objects as "regular", "extreme" or "outlier".

### Usage

```
## S3 method for class 'pls'  
categorize(obj, res = obj$res$cal, ncomp = obj$ncomp.selected, ...)
```

### Arguments

obj	object with PLS model
res	object with PLS results
ncomp	number of components to use for the categorization
...	other parameters

### Details

The method does not categorize hidden values if any. It is based on the approach described in [1] and works only if data driven approach is used for computing critical limits.

### Value

vector (factor) with results of categorization.

### References

1. Rodionova O. Ye., Pomerantsev A. L. Detection of Outliers in Projection-Based Modeling. Analytical Chemistry (2020, in publish). doi: 10.1021/acs.analchem.9b04611

---

chisq.crit	<i>Calculates critical limits for distance values using Chi-square distribution</i>
------------	---

---

**Description**

The method is based on Chi-squared distribution with  $DF = 2 * (m(u)/s(u)^2)$

**Usage**

```
chisq.crit(param, alpha = 0.05, gamma = 0.01)
```

**Arguments**

param	matrix with distribution parameters
alpha	significance level for extreme objects
gamma	significance level for outliers

---

chisq.prob	<i>Calculate probabilities for distance values using Chi-square distribution</i>
------------	--

---

**Description**

Calculate probabilities for distance values using Chi-square distribution

**Usage**

```
chisq.prob(u, param)
```

**Arguments**

u	vector with distances
param	vector with distribution parameters

---

<code>clamp.dof</code>	<i>Round and clamp degrees of freedom to valid range [1, 250]</i>
------------------------	---

---

**Description**

Round and clamp degrees of freedom to valid range [1, 250]

**Usage**

```
clamp.dof(x)
```

**Arguments**

`x` vector with raw degrees of freedom values

**Value**

vector with rounded and clamped values

---

<code>classify</code>	<i>Creates classification outcomes for given PCA result objects and distance parameters.</i>
-----------------------	--

---

**Description**

Creates classification outcomes for given PCA result objects and distance parameters.

**Usage**

```
classify(  
  pcares,  
  indices,  
  numbers,  
  qp,  
  hp,  
  nobj.cal,  
  alpha = 0.05,  
  gamma = 0.01,  
  classname,  
  c.ref = NULL  
)
```

**Arguments**

pcares	object of class <code>pcares</code> .
indices	list with indices of members, strangers, and unknown samples as well as information about excluded objects.
numbers	list with numbers of members, strangers, unknown and excluded samples.
qp	distribution parameters for q-distances.
hp	distribution parameters for h-distances.
nobj.cal	number of objects in calibration set the model was built on (needed for outliers detection).
alpha	significance level for decision boundary.
gamma	significance level for outlier detection boundary.
classname	name of the target class.
c.ref	vector with names of the reference classes.

**Value**

a list with classification outcomes for each number of components.

---

classify.plsda	<i>PLS-DA classification</i>
----------------	------------------------------

---

**Description**

Converts PLS predictions of y values to predictions of classes

**Usage**

```
classify.plsda(model, y)
```

**Arguments**

model	a PLS-DA model (object of class <code>plsda</code> )
y	a matrix with predicted y values

**Details**

This is a service function for PLS-DA class, do not use it manually.

**Value**

Classification results (an object of class `classes`)

---

classify.simca	<i>SIMCA classification</i>
----------------	-----------------------------

---

**Description**

Makes classification based on calculated T2 and Q values and corresponding limits

**Usage**

```
classify.simca(obj, pca.res, c.ref = NULL)
```

**Arguments**

obj	a SIMCA model (object of class simca)
pca.res	results of projection data to PCA space
c.ref	vector with class reference values

**Details**

This is a service function for SIMCA class, do not use it manually.

**Value**

vector with predicted class values (c.pred)

---

classmodel.processRefValues
-----------------------------

*Check reference class values and convert it to a factor if necessary*

---

**Description**

Check reference class values and convert it to a factor if necessary

**Usage**

```
classmodel.processRefValues(c.ref, classnames = NULL)
```

**Arguments**

c.ref	class reference values provided by user
classnames	text with class name in case of logical reference values

---

classes	<i>Results of classification</i>
---------	----------------------------------

---

**Description**

classes is used to store results of classification for one or multiple classes.

**Usage**

```
classes(c.pred, c.ref = NULL, p.pred = NULL, ncomp.selected = 1)
```

**Arguments**

c.pred	matrix with predicted values (+1 or -1) for each class.
c.ref	matrix with reference values for each class.
p.pred	matrix with probability values for each class.
ncomp.selected	vector with selected number of components for each class.

**Details**

There is no need to create a classes object manually, it is created automatically when building a classification model (e.g. using [simca](#) or [plsda](#)) or apply the model to new data. For any classification method from `mdatools`, a class using to represent results of classification (e.g. [simcares](#)) inherits fields and methods of classes.

**Value**

c.pred	predicted class values (+1 or -1).
p.pred	predicted class probabilities.
c.ref	reference (true) class values if provided.

The following fields are available only if reference values were provided.

tp	number of true positives.
tn	number of true negatives.
fp	number of false positives.
fn	number of false negatives.
specificity	specificity of predictions.
sensitivity	sensitivity of predictions.
misclassified	ratio of misclassified objects.

**See Also**

Methods for classes class:

<code>showPredictions.classres</code>	shows table with predicted values.
<code>plotPredictions.classres</code>	makes plot with predicted values.
<code>plotSensitivity.classres</code>	makes sensitivity plot.
<code>plotSpecificity.classres</code>	makes specificity plot.
<code>plotMisclassified.classres</code>	makes ms ratio plot.
<code>plotPerformance.classres</code>	makes plot with misclassified ratio, specificity and sensitivity values.

---

`classres.getPerformance`

*Calculation of classification performance parameters*

---

### Description

Calculates and returns performance parameters for classification result (e.g. number of false negatives, false positives, sn, specificity, etc.).

### Usage

```
classres.getPerformance(c.ref, c.pred)
```

### Arguments

<code>c.ref</code>	reference class values for objects (vector with numeric or text values)
<code>c.pred</code>	predicted class values for objects (array nobj x ncomponents x nclasses)

### Details

The function is called automatically when a classification result with reference values is created, for example when applying a `plsda` or `simca` models.

### Value

Returns a list with following fields:

<code>\$fn</code>	number of false negatives (nclasses x ncomponents)
<code>\$fp</code>	number of false positives (nclasses x ncomponents)
<code>\$tp</code>	number of true positives (nclasses x ncomponents)
<code>\$sensitivity</code>	sensitivity values (nclasses x ncomponents)
<code>\$specificity</code>	specificity values (nclasses x ncomponents)
<code>\$misclassified</code>	misclassification ratio values (nclasses x ncomponents)

---

cleanLabels	<i>Clean text labels from extra elements so they are compatible with JSON</i>
-------------	---

---

**Description**

Clean text labels from extra elements so they are compatible with JSON

**Usage**

```
cleanLabels(str)
```

**Arguments**

str	label string (for example produced by expression).
-----	--

---

confint.regcoeffs	<i>Confidence intervals for regression coefficients</i>
-------------------	---

---

**Description**

returns matrix with confidence intervals for regression coefficients for given response number and number of components.

**Usage**

```
## S3 method for class 'regcoeffs'
confint(object, parm = NULL, level = 0.95, ncomp = 1, ny = 1, ...)
```

**Arguments**

object	regression coefficients object (class regcoeffs)
parm	not used, needed for compatibility with general method
level	confidence level
ncomp	number of components (one value)
ny	index of response variable (one value)
...	other arguments

---

constraint	<i>Class for MCR-ALS constraint</i>
------------	-------------------------------------

---

**Description**

Class for MCR-ALS constraint

**Usage**

```
constraint(name, params = NULL, method = NULL)
```

**Arguments**

name	short text with name for the constraint
params	a list with parameters for the constraint method (if NULL - default parameters will be used)
method	method to call when applying the constraint, provide it only for user defined constraints

**Details**

Use this class to create constraints and add them to a list for MCR-ALS curve resolution (see [mcrals](#)). Either provide name and parameters to one of the existing constraint implementations or make your own. See the list of implemented constraints by running `constraints()`

For your own constraint you need to create a method, which takes matrix with values (either spectra or contributions being resolved) as the first argument, does something and then returns a matrix with the same dimension as the result. The method can have any number of optional parameters.

See help for [mcrals](#) or Bookdown tutorial for details.

---

constraintAngle	<i>Method for angle constraint</i>
-----------------	------------------------------------

---

**Description**

Adds a small portion of mean to contributions or spectra to increase contrast

**Usage**

```
constraintAngle(x, d, weight = 0.05)
```

**Arguments**

x	data matrix (spectra or contributions)
d	matrix with the original spectral values
weight	how many percent of mean to add (between 0 and 1)

---

constraintClosure      *Method for closure constraint*

---

**Description**

Force rows of data sum up to given value

**Usage**

```
constraintClosure(x, d, sum = 1)
```

**Arguments**

x	data matrix (spectra or contributions)
d	matrix with the original spectral values
sum	which value the spectra or contributions should sum up to

---

constraintNonNegativity  
*Method for non-negativity constraint*

---

**Description**

Set all negative values in the matrix to 0

**Usage**

```
constraintNonNegativity(x, d)
```

**Arguments**

x	data matrix (spectra or contributions)
d	matrix with the original spectral values

---

constraintNorm	<i>Method for normalization constraint</i>
----------------	--

---

**Description**

Normalize rows of matrix to unit length or area

**Usage**

```
constraintNorm(x, d, type = "length")
```

**Arguments**

x	data matrix (spectra or contributions)
d	matrix with the original spectral values
type	type of normalization ("area", "length" or "sum")

---

constraints.list	<i>Shows information about all implemented constraints</i>
------------------	--

---

**Description**

Shows information about all implemented constraints

**Usage**

```
constraints.list()
```

---

constraintUnimod	<i>Method for unimodality constraint</i>
------------------	--

---

**Description**

forces column of matrix to have one maximum each

**Usage**

```
constraintUnimod(x, d, tol = 0)
```

**Arguments**

x	data matrix (spectra or contributions)
d	matrix with the original spectral values
tol	tolerance (value between 0 and 1) to make the method stable to small fluctuations

---

create_categories	<i>Create a factor with categories (regular, extreme, outlier)</i>
-------------------	--

---

**Description**

Create a factor with categories (regular, extreme, outlier)

**Usage**

```
create_categories(nobj, extremes_ind, outliers_ind)
```

**Arguments**

nobj	number of objects
extremes_ind	logical or numeric indices for extreme objects
outliers_ind	logical or numeric indices for outlier objects

---

crossval	<i>Generate sequence of indices for cross-validation</i>
----------	--

---

**Description**

Generates and returns sequence of object indices for each segment in random segmented cross-validation

**Usage**

```
crossval(cv = 1, nobj = NULL, resp = NULL)
```

**Arguments**

cv	cross-validation settings, can be a number or a list. If cv is a number, it will be used as a number of segments for random cross-validation (if cv = 1, full cross-validation will be performed), if it is a list, the following syntax can be used: cv = list('rand', nseg, nrep) for random repeated cross-validation with nseg segments and nrep repetitions or cv = list('ven', nseg) for systematic splits to nseg segments ('venetian blinds').
nobj	number of objects in a dataset
resp	vector with response values to use in case of venetian blinds

**Value**

matrix with object indices for each segment

---

`crossval.getParams`      *Define parameters based on 'cv' value*

---

**Description**

Define parameters based on 'cv' value

**Usage**

```
crossval.getParams(cv, nobj)
```

**Arguments**

<code>cv</code>	settings for cross-validation provided by user
<code>nobj</code>	number of objects in calibration set

---

`crossval.regmodel`      *Cross-validation of a regression model*

---

**Description**

Does cross-validation of a regression model

**Usage**

```
crossval.regmodel(obj, x, y, cv, cal.fun, pred.fun, cv.scope = "local")
```

**Arguments**

<code>obj</code>	a regression model (object of class <code>regmodel</code> )
<code>x</code>	a matrix with x values (predictors from calibration set)
<code>y</code>	a matrix with y values (responses from calibration set)
<code>cv</code>	number of segments (if <code>cv = 1</code> , full cross-validation will be used)
<code>cal.fun</code>	reference to function for model calibration
<code>pred.fun</code>	reference to function for getting predicted y-values (see description)
<code>cv.scope</code>	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.

**Value**

object of class `plsres` with results of cross-validation

Function 'pred.fun' must take four arguments: autoscaled x-values, array with regression coefficients, vectors for centering and scaling of y-values (if used). The function must return predicted y-values in original units (unscaled and uncentered).

---

crossval.simca	<i>Cross-validation of a SIMCA model</i>
----------------	--

---

**Description**

Does the cross-validation of a SIMCA model

**Usage**

```
crossval.simca(obj, x, cv)
```

**Arguments**

obj	a SIMCA model (object of class simca)
x	a matrix with x values (predictors from calibration set)
cv	number of segments (if cv = 1, full cross-validation will be used)

**Value**

object of class simcares with results of cross-validation

---

crossval.str	<i>String with description of cross-validation method</i>
--------------	---

---

**Description**

String with description of cross-validation method

**Usage**

```
crossval.str(cv)
```

**Arguments**

cv	a list with cross-validation settings
----	---------------------------------------

**Value**

a string with the description text

---

dd.crit	<i>Calculates critical limits for distance values using Data Driven moments approach</i>
---------	--

---

**Description**

Calculates critical limits for distance values using Data Driven moments approach

**Usage**

```
dd.crit(paramQ, paramT2, alpha = 0.05, gamma = 0.01)
```

**Arguments**

paramQ	matrix with parameters for distribution of Q distances
paramT2	matrix with parameters for distribution of T2 distances
alpha	significance level for extreme objects
gamma	significance level for outliers

---

ddmoments.param	<i>Calculates critical limits for distance values using Data Driven moments approach</i>
-----------------	--

---

**Description**

Calculates critical limits for distance values using Data Driven moments approach

**Usage**

```
ddmoments.param(U)
```

**Arguments**

U	matrix or vector with distance values
---	---------------------------------------

---

ddrobust.param	<i>Calculates critical limits for distance values using Data Driven robust approach</i>
----------------	---

---

**Description**

Calculates critical limits for distance values using Data Driven robust approach

**Usage**

```
ddrobust.param(U)
```

**Arguments**

U                    matrix or vector with distance values

---

ddsimca	<i>Data Driven SIMCA</i>
---------	--------------------------

---

**Description**

ddsimca is used to develop DD-SIMCA (Data Driven SIMCA) model for one-class classification.

**Usage**

```
ddsimca(  
  x,  
  classname,  
  ncomp = min(nrow(x) - 1, ncol(x) - 1, 20),  
  center = TRUE,  
  scale = FALSE,  
  pcv = list("ven", 10),  
  alpha = 0.05,  
  gamma = 0.01,  
  exclrows = NULL,  
  exclcols = NULL,  
  prep = NULL,  
  do.round = TRUE,  
  ...  
)
```

**Arguments**

x	a numerical matrix with data values.
classname	short text (up to 20 symbols) with class name.
ncomp	maximum number of components to calculate.
center	logical, do mean centering of data or not.
scale	logical, do standardization of data or not.
pcv	Procrustes cross-validation settings (see details).
alpha	significance level for making the predictions (can be also adjusted when model is applied to data).
gamma	significance level for detection of outliers (can be also adjusted when model is applied to data).
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)
exclcols	columns to be excluded from calculations (numbers, names or vector with logical values)
prep	optional list with preprocessing methods created using <code>'prep'</code> function.
do.round	logical, round or not DoF for distances.
...	any other parameters suitable for <code>pca</code> method.

**Details**

DD-SIMCA is based on PCA model with additional functionality, so `ddsimca` class inherits most of the functionality of `pca` class.

In order to make a decision, DDSIMCA uses score and orthogonal distances to PCA model. It combines the two distances to joint full distance and uses chi-distribution for finding a critical value which is employed as decision rule. More details about DD-SIMCA can be found in [1] (open access).

Procrustes cross-validation (PCV) is used to generate a validation set in order to find optimal model complexity (number of components). The PCV settings are similar to the ones used for conventional cross-validation. The best way is to set `'pcv'` value to a list, for example: `pcv = list('ven', nseg)` for systematic splits or `pcv = list('rand', nseg)` for random splits. In case if full cross-validation must be employed, use `pcv = list('loo')`.

**Value**

Returns an object of `ddsimca` class with following fields:

classname	a short text with class name.
calres	an object of class <code>simcares</code> with classification results for a calibration data.
pvres	an object of class <code>simcares</code> with classification results for a Procrustes validation set.

Fields, inherited from `pca` class:

ncomp	number of components included to the model.
-------	---

<code>ncomp.selected</code>	selected (optimal) number of components.
<code>loadings</code>	matrix with loading values (nvar x ncomp).
<code>eigenvals</code>	vector with eigenvalues for all existent components.
<code>expvar</code>	vector with explained variance for each component (in percent).
<code>cumexpvar</code>	vector with cumulative explained variance for each component (in percent).
<code>info</code>	information about the model, provided by user when build the model.

### Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

### References

1. Kucheryavskiy S, Rodionova O, Pomerantsev A. A comprehensive tutorial on Data-Driven SIMCA: Theory and implementation in web. *Journal of Chemometrics*. 2024; 38(7):e3556. doi:10.1002/cem.3556
2. S. Kucheryavskiy, O. Rodionova, A. Pomerantsev, Procrustes cross-validation of multivariate regression models. *Analytica Chimica Acta*. 2023; 1255:341096. doi:10.1016/j.aca.2023.341096.

### See Also

Methods for ddsimca objects:

<code>print. ddsimca</code>	shows information about the object.
<code>summary. ddsimca</code>	shows summary statistics for the model.
<code>plot. ddsimca</code>	makes an overview of DD-SIMCA model with four plots.
<code>predict. ddsimca</code>	applies DD-SIMCA model to a new data.

Methods, inherited from `classmodel` class:

<code>plotPredictions. classmodel</code>	shows plot with predicted values.
<code>plotSensitivity. classmodel</code>	shows sensitivity plot.
<code>plotSpecificity. classmodel</code>	shows specificity plot.
<code>plotMisclassified. classmodel</code>	shows misclassified ratio plot.

Methods, inherited from `pca` class:

<code>selectCompNum. pca</code>	set number of optimal components in the model
<code>plotScores. pca</code>	shows scores plot.
<code>plotLoadings. pca</code>	shows loadings plot.
<code>plotVariance. pca</code>	shows explained variance plot.
<code>plotCumVariance. pca</code>	shows cumulative explained variance plot.

## Examples

```
## make a SIMCA model for Iris setosa class with full cross-validation
library(mdatools)

data = iris[, 1:4]
class = iris[, 5]

# take first 20 objects of setosa as calibration set
se = data[1:20, ]

# make SIMCA model and apply to test set
model = ddsimca(se, "setosa", pcv = list("ven", 10))
model = selectCompNum(model, 1)

# show information, summary and plot overview
print(model)
summary(model)
plot(model)
```

---

ddsimca.fromjson	<i>Converts JSON string created in mda.tools/ddsimca app to ddsimca object</i>
------------------	--

---

## Description

Converts JSON string created in mda.tools/ddsimca app to ddsimca object

## Usage

```
ddsimca.fromjson(str)
```

## Arguments

str	stringified JSON (from model file)
-----	------------------------------------

## Value

object of `ddsimca` class

---

ddsimca.readJSON	<i>Reads DD-SIMCA model from JSON file made in web-application (mda.tools/ddsimca).</i>
------------------	---

---

**Description**

Reads DD-SIMCA model from JSON file made in web-application (mda.tools/ddsimca).

**Usage**

```
ddsimca.readJSON(fileName)
```

**Arguments**

fileName	file name (or full path) to JSON file.
----------	--

**Value**

list with DD-SIMCA model similar to what ddsimca() creates.

---

ddsimcares	<i>Results of DD-SIMCA one-class classification</i>
------------	---

---

**Description**

@description ddsimcares is used to store results for DD-SIMCA one-class classification. Do not create this object manually, it will be created automatically by applying DD-SIMCA model.

**Usage**

```
ddsimcares(pcares, outcomes, classname, indices, numbers, alpha, c.ref = NULL)
```

**Arguments**

pcares	results of PCA decomposition of data (class pcares).
outcomes	outcomes of DD-SIMCA classification procedure.
classname	short text (up to 20 symbols) with class name.
indices	list with the object indices (members, strangers, unknown).
numbers	list with the object numbers in each subset (members, strangers, unknown).
alpha	significance level used for making the predictions.
c.ref	optional, vector with reference classes.

## Details

Class `ddsimcares` inherits all properties and methods of class `pcare`s, and has additional properties and functions for representing of classification results and other DD-SIMCA outcomes.

Do not create a `ddsimcares` object manually, it is created automatically when a DD-SIMCA model is developed (see `ddsimca`) or when the model is applied to a new data (see `predict.ddsimca`). The object can be used to show summary and plots for the results.

## Value

Returns an object (list) of class `ddsimcares` with the same fields as `pcare`s plus additional field `simcares` which is a list with all DD-SIMCA outcomes and related properties:

## See Also

Methods specific for `ddsimcares` objects:

<code>print.ddsimcares</code>	shows information about the object.
<code>summary.ddsimcares</code>	shows statistics for results of classification.
<code>as.data.frame.ddsimcares</code>	converts DD-SIMCA results into data frame.
<code>as.matrix.ddsimcares</code>	converts summary of DD-SIMCA results into matrix.
<code>writeCSV.ddsimcares</code>	saves DD-SIMCA results into a CSV file.
<code>plotAcceptance.ddsimcares</code>	shows acceptance plot (q/q0 vs h/h0) with decision and outlier boundaries.
<code>plotExtremes.ddsimcares</code>	shows extremes plot.
<code>plotAliens.ddsimcares</code>	shows aliens plot.
<code>plotDistances.ddsimcares</code>	shows plot with individual distances (q, h or f).

Methods, inherited from `ldecomp` class:

<code>plotScores.ldecomp</code>	makes scores plot.
<code>plotVariance.ldecomp</code>	makes explained variance plot.
<code>plotCumVariance.ldecomp</code>	makes cumulative explained variance plot.

Check also `ddsimca` and `pcare`s.

## Examples

```
## make a DD-SIMCA model for Iris setosa class and show results for calibration set
library(mdatools)

data = iris[, 1:4]
class = iris[, 5]

# take every second of first 50 objects (setosa) as calibration set
se = data[seq(1, 50, by = 2), ]

# take the rest as test set
ind.test = c(seq(2, 50, by = 2), 51:150)
```

```
x.test = data[ind.test, ]
c.test = class[ind.test]

# make DD-SIMCA model and set optimal number of components to 1
model = ddsimca(se, 'setosa', scale = TRUE)
model = selectCompNum(model, 1)

# apply model to test set
r = predict(model, x.test, c.test)
print(r)

# show summary
summary(r)

# show plots
par(mfrow = c(2, 2))
plotAcceptance(r)
plotFoMs(r)
plotExtremes(r)
plotSelectivityArea(r)
```

---

ellipse

*Create ellipse on the current plot*

---

### Description

Create ellipse on the current plot

### Usage

```
ellipse(xc = 0, yc = 0, a, b, col = "black", lty = 1, ...)
```

### Arguments

xc	coordinate of center (x)
yc	coordinate of center (y)
a	major axis
b	minor axis
col	color of the ellipse line
lty	type of the ellipse line
...	any argument suitable for lines function

---

employ.constraint	<i>Applies constraint to a dataset</i>
-------------------	--

---

**Description**

Applies constraint to a dataset

**Usage**

```
employ.constraint(obj, x, d, ...)
```

**Arguments**

obj	object with constraint
x	matrix with pure spectra or contributions
d	matrix with original spectral values
...	other arguments

---

employ.prep	<i>Applies a list with preprocessing methods to a dataset</i>
-------------	---

---

**Description**

Applies a list with preprocessing methods to a dataset

**Usage**

```
employ.prep(obj, x, ...)
```

**Arguments**

obj	list with preprocessing methods (created using prep function).
x	matrix with dataset
...	other arguments

---

extractArray	<i>Extract numeric array from JSON string</i>
--------------	---

---

**Description**

Extract numeric array from JSON string

**Usage**

```
extractArray(js, key)
```

**Arguments**

js	stringified JSON.
key	name of the element.

**Value**

array of numbers.

---

extractBlock	<i>Extracts a JSON subset in main JSON structure</i>
--------------	--

---

**Description**

Extracts a JSON subset in main JSON structure

**Usage**

```
extractBlock(js, name)
```

**Arguments**

js	stringified JSON with model
name	name of the JSON block to extract

**Value**

string with part of the JSON string related to preprocessing

---

extractPrep	<i>Extracts JSON related to preprocessing model</i>
-------------	---

---

**Description**

Extracts JSON related to preprocessing model

**Usage**

```
extractPrep(js)
```

**Arguments**

js	stringified JSON with model
----	-----------------------------

**Value**

string with part of the JSON string related to preprocessing

---

extractStringArray	<i>Extract string array from JSON string</i>
--------------------	--

---

**Description**

Extract string array from JSON string

**Usage**

```
extractStringArray(js, key)
```

**Arguments**

js	stringified JSON.
key	name of the element.

**Value**

array of strings.

---

extractValue	<i>Extract single value from JSON string</i>
--------------	--

---

**Description**

Extract single value from JSON string

**Usage**

```
extractValue(js, key)
```

**Arguments**

js	stringified JSON.
key	name of the element.

**Value**

value

---

fprintf	<i>Imitation of fprintf() function</i>
---------	--

---

**Description**

Imitation of fprintf() function

**Usage**

```
fprintf(...)
```

**Arguments**

...	arguments for sprintf function
-----	--------------------------------

---

genhash	<i>Generates unique pseudo-hash number based on current time and date</i>
---------	---

---

**Description**

Generates unique pseudo-hash number based on current time and date

**Usage**

genhash()

**Value**

string with the hash

---

getCalibrationData	<i>Calibration data</i>
--------------------	-------------------------

---

**Description**

Calibration data

**Usage**

getCalibrationData(obj)

**Arguments**

obj            a model object

**Details**

Generic function getting calibration data from a linear decomposition model (e.g. PCA)

---

`getCalibrationData.pca`

*Returns matrix with original calibration data*

---

### **Description**

Returns matrix with original calibration data

### **Usage**

```
## S3 method for class 'pca'  
getCalibrationData(obj)
```

### **Arguments**

`obj`                    object with PCA model

---

`getCalibrationData.simcam`

*Get calibration data*

---

### **Description**

Get data, used for calibration of the SIMCAM individual models and combine to one dataset.

### **Usage**

```
## S3 method for class 'simcam'  
getCalibrationData(obj)
```

### **Arguments**

`obj`                    SIMCAM model (object of class `simcam`)

### **Details**

See examples in help for [simcam](#) function.

---

getConfidenceEllipse *Compute confidence ellipse for a set of points*

---

**Description**

Compute confidence ellipse for a set of points

**Usage**

```
getConfidenceEllipse(points, conf.level = 0.95, n = 100)
```

**Arguments**

points	matrix or data frame with coordinates of the points
conf.level	confidence level for the ellipse
n	number of points in the ellipse coordinates

**Value**

matrix with coordinates of the ellipse points (x and y)

---

getConfusionMatrix *Confusion matrix for classification results*

---

**Description**

Confusion matrix for classification results

**Usage**

```
getConfusionMatrix(obj, ...)
```

**Arguments**

obj	classification results (object of class simcares, simcamres, etc)
...	other parameters.

**Details**

Returns confusion matrix for classification results represented by the object.

---

```
getConfusionMatrix.classres
```

*Confusion matrix for classification results*

---

**Description**

The columns of the matrix correspond to classification results, rows - to the real classes. In case of soft classification with multiple classes (e.g. SIMCAM) sum of values for every row will not correspond to the total number of class members as the same object can be classified as a member of several classes or none of them.

**Usage**

```
## S3 method for class 'classres'
getConfusionMatrix(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj	classification results (object of class simcares, simcamres, etc)
ncomp	number of components to make the matrix for (NULL - use selected for a model).
...	other arguments

**Details**

Returns confusion matrix for classification results represented by the object.

---

```
getConvexHull
```

*Compute coordinates of a closed convex hull for data points*

---

**Description**

Compute coordinates of a closed convex hull for data points

**Usage**

```
getConvexHull(points)
```

**Arguments**

points	matrix or data frame with coordinates of the points
--------	---

---

getDataLabels	<i>Create a vector with labels for plot series</i>
---------------	--

---

**Description**

For scatter plots labels correspond to rows of the data (names, values, indices, etc.). For non-scatter plots labels correspond to the columns (names, indices or max value for each column)

**Usage**

```
getDataLabels(ps, labels = NULL)
```

**Arguments**

ps	'plotseries' object
labels	vector with user defined labels or type of labels to show ("values", "names", "indices")

---

getImplementedConstraints	<i>Shows a list with implemented constraints</i>
---------------------------	--

---

**Description**

Shows a list with implemented constraints

**Usage**

```
getImplementedConstraints()
```

---

getImplementedPrepMethods	<i>Shows a list with implemented preprocessing methods</i>
---------------------------	--

---

**Description**

Shows a list with implemented preprocessing methods

**Usage**

```
getImplementedPrepMethods()
```

---

`getLabelsAsIndices`      *Create labels as column or row indices*

---

**Description**

Create labels as column or row indices

**Usage**

`getLabelsAsIndices(ps)`

**Arguments**

`ps`                      'plotseries' object

---

`getLabelsAsValues`      *Create labels from data values*

---

**Description**

Create labels from data values

**Usage**

`getLabelsAsValues(ps)`

**Arguments**

`ps`                      'plotseries' object

---

`getMainTitle`              *Get main title*

---

**Description**

returns main title for a plot depending on a user choice

**Usage**

`getMainTitle(main, ncomp, default)`

**Arguments**

main	main title of a plot, provided by user
ncomp	number of components to select, provided by user
default	default title for the plot

**Details**

Depending on a user choice it returns main title for a plot

---

<code>getPlotColors</code>	<i>Define colors for plot series</i>
----------------------------	--------------------------------------

---

**Description**

Define colors for plot series

**Usage**

```
getPlotColors(ps, col, opacity, cgroup, colmap)
```

**Arguments**

ps	'plotseries' object
col	color specified by user (if any)
opacity	opacity for the color
cgroup	vector for color grouping (if any)
colmap	name or values for colormap

---

<code>getProbabilities</code>	<i>Get class belonging probability</i>
-------------------------------	--

---

**Description**

Compute class belonging probabilities for classification results.

**Usage**

```
getProbabilities(obj, ...)
```

**Arguments**

obj	an object with classification results (e.g. SIMCA)
...	other parameters

---

getProbabilities.pca *Probabilities for residual distances*

---

### Description

Probabilities for residual distances

### Usage

```
## S3 method for class 'pca'  
getProbabilities(obj, ncomp, q, h, ...)
```

### Arguments

obj	object with PCA model
ncomp	number of components to compute the probability for
q	vector with squared orthogonal distances for given number of components
h	vector with score distances for given number of components
...	other parameters

### Details

Computes p-value for every object being from the same population as calibration set based on its orthogonal and score distances.

---

getProbabilities.simca  
*Probabilities of class belonging for PCA/SIMCA results*

---

### Description

Probabilities of class belonging for PCA/SIMCA results

### Usage

```
## S3 method for class 'simca'  
getProbabilities(obj, ncomp, q, h, ...)
```

### Arguments

obj	object with PCA model
ncomp	number of components to compute the probability for
q	vector with squared orthogonal distances for given number of components
h	vector with score distances for given number of components
...	other parameters

**Details**

Computes p-value for every object being from the same population as calibration set based on its orthogonal and score distances.

---

getPureVariables      *Identifies pure variables*

---

**Description**

The method identifies indices of pure variables using the SIMPLISMA algorithm.

**Usage**

```
getPureVariables(D, ncomp, purevars, offset)
```

**Arguments**

D	matrix with the spectra
ncomp	number of pure components
purevars	user provided values for pure variables (no calculation will be run in this case)
offset	offset (between 0 and 1) for calculation of parameter alpha

**Value**

The function returns a list with the following fields:

ncomp	number of pure components.
purevars	vector with indices for pure variables.
purityspec	matrix with purity values for each resolved components.
purevals	vector with purity values for resolved components.

---

getRegcoeffs      *Get regression coefficients*

---

**Description**

Generic function for getting regression coefficients from PLS model

**Usage**

```
getRegcoeffs(obj, ...)
```

**Arguments**

obj	a PLS model
...	other parameters

---

getRegcoeffs.regmodel *Regression coefficients for PLS model*

---

### Description

Returns a matrix with regression coefficients for the PLS model which can be applied to a data directly

### Usage

```
## S3 method for class 'regmodel'  
getRegcoeffs(  
  obj,  
  ncomp = obj$ncomp.selected,  
  ny = 1,  
  full = FALSE,  
  alpha = 0.05,  
  ...  
)
```

### Arguments

obj	a PLS model (object of class pls)
ncomp	number of components to return the coefficients for
ny	if y is multivariate which variables you want to see the coefficients for
full	if TRUE the method also shows p-values and t-values as well as confidence intervals for the coefficients (if available)
alpha	significance level for confidence intervals (a number between 0 and 1, e.g. 0.05)
...	other parameters

### Details

The method recalculates the regression coefficients found by the PLS algorithm taking into account centering and scaling of predictors and responses, so the matrix with coefficients can be applied directly to original data ( $yp = Xb$ ).

If number of components is not specified, the optimal number, selected by user or identified by a model will be used.

If Jack-knifing method was used to get statistics for the coefficient the method returns all statistics as well (p-value, t-value, confidence interval). In this case user has to specified a number of y-variable (if there are many) to get the statistics and the coefficients for. The confidence interval is computed for unstandardized coefficients.

### Value

A matrix with regression coefficients and (optionally) statistics.

---

getRes	<i>Return list with valid results</i>
--------	---------------------------------------

---

**Description**

Return list with valid results

**Usage**

```
getRes(res, classname = "ldecomp")
```

**Arguments**

res	list with results
classname	name of class (for result object) to look for

---

getSelectedComponents	<i>Get selected components</i>
-----------------------	--------------------------------

---

**Description**

returns number of components depending on a user choice

**Usage**

```
getSelectedComponents(obj, ncomp = NULL)
```

**Arguments**

obj	an MDA model or result object (e.g. pca, pls, simca, etc)
ncomp	number of components to select, provided by user

**Details**

Depending on a user choice it returns optimal number of component for the model (if user did not provide any value) or check the user choice for correctness and returns it back

---

getSelectivityRatio    *Selectivity ratio*

---

**Description**

Generic function for returning selectivity ratio values for regression model (PCR, PLS, etc)

**Usage**

```
getSelectivityRatio(obj, ...)
```

**Arguments**

obj	a regression model
...	other parameters

---

getSelectivityRatio.pls

*Selectivity ratio for PLS model*

---

**Description**

Returns vector with Selectivity ratio values. This function is a proxy for [selratio](#) and will be removed in future releases.

**Usage**

```
## S3 method for class 'pls'  
getSelectivityRatio(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj	a PLS model (object of class pls)
ncomp	number of components to get the values for (if NULL user selected as optimal will be used)
...	other parameters

**Value**

vector with selectivity ratio values

**References**

[1] Tarja Rajalahti et al. Chemometrics and Laboratory Systems, 95 (2009), pp. 35-48.

---

getVariance.mcr	<i>Compute explained variance for MCR case</i>
-----------------	--

---

**Description**

Compute explained variance for MCR case

**Usage**

```
getVariance.mcr(obj, x)
```

**Arguments**

obj	object of class mcr
x	original spectral data

---

getVIPScores	<i>VIP scores</i>
--------------	-------------------

---

**Description**

Generic function for returning VIP scores values for regression model (PCR, PLS, etc)

**Usage**

```
getVIPScores(obj, ...)
```

**Arguments**

obj	a regression model
...	other parameters

---

getVIPScores.pls	<i>VIP scores for PLS model</i>
------------------	---------------------------------

---

**Description**

Returns vector with VIP scores values. This function is a proxy for [vipcores](#) and will be removed in future releases.

**Usage**

```
## S3 method for class 'pls'
getVIPScores(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj	a PLS model (object of class pls)
ncomp	number of components to show
...	other parameters

**Value**

matrix nvar x 1 with VIP score values

---

hotelling.crit	<i>Calculate critical limits for distance values using Hotelling T2 distribution</i>
----------------	--

---

**Description**

Calculate critical limits for distance values using Hotelling T2 distribution

**Usage**

```
hotelling.crit(nobj, ncomp, alpha = 0.05, gamma = 0.01)
```

**Arguments**

nobj	number of objects in calibration set
ncomp	number of components
alpha	significance level for extreme objects
gamma	significance level for outliers

**Value**

vector with four values: critical limits for given alpha and gamma, mean distance and DoF.

---

hotelling.prob	<i>Calculate probabilities for distance values and given parameters using Hotelling T2 distribution</i>
----------------	---

---

**Description**

Calculate probabilities for distance values and given parameters using Hotelling T2 distribution

**Usage**

```
hotelling.prob(u, ncomp, nobj)
```

**Arguments**

u	vector with distances
ncomp	number of components
nobj	number of objects in calibration set

---

imshow	<i>show image data as an image</i>
--------	------------------------------------

---

**Description**

show image data as an image

**Usage**

```
imshow(
    data,
    channels = 1,
    show.excluded = FALSE,
    main = paste0(" ", colnames(data)[channels]),
    colmap = "jet"
)
```

**Arguments**

data	data with image
channels	indices for one or three columns to show as image channels
show.excluded	logical, if TRUE the method also shows the excluded (hidden) pixels
main	main title for the image
colmap	colormap used to show the intensity levels

---

 ipls
 

---



---

*Variable selection with interval PLS*


---

**Description**

Applies iPLS algorithm to find variable intervals most important for prediction.

**Usage**

```

ipls(
  x,
  y,
  glob.ncomp = 10,
  center = TRUE,
  scale = FALSE,
  cv = list("ven", 10),
  exclcols = NULL,
  exclrows = NULL,
  int.ncomp = glob.ncomp,
  int.num = NULL,
  int.width = NULL,
  int.limits = NULL,
  int.niter = NULL,
  ncomp.selcrit = "min",
  method = "forward",
  x.test = NULL,
  y.test = NULL,
  silent = FALSE,
  full = FALSE,
  cv.scope = "local"
)

```

**Arguments**

<code>x</code>	a matrix with predictor values.
<code>y</code>	a vector with response values.
<code>glob.ncomp</code>	maximum number of components for a global PLS model.
<code>center</code>	logical, center or not the data values.
<code>scale</code>	logical, standardize or not the data values.
<code>cv</code>	cross-validation settings (see details).
<code>exclcols</code>	columns of <code>x</code> to be excluded from calculations (numbers, names or vector with logical values).
<code>exclrows</code>	rows to be excluded from calculations (numbers, names or vector with logical values).
<code>int.ncomp</code>	maximum number of components for interval PLS models.

<code>int.num</code>	number of intervals.
<code>int.width</code>	width of intervals.
<code>int.limits</code>	a two column matrix with manual intervals specification.
<code>int.niter</code>	maximum number of iterations (if NULL it will be the smallest of two values: number of intervals and 30).
<code>ncomp.selcrit</code>	criterion for selecting optimal number of components ('min' for minimum of RMSECV).
<code>method</code>	iPLS method ('forward' or 'backward').
<code>x.test</code>	matrix with predictors for test set (by default is NULL, if specified, is used instead of <code>cv</code> ).
<code>y.test</code>	matrix with responses for test set.
<code>silent</code>	logical, show or not information about selection process.
<code>full</code>	logical, if TRUE the procedure will continue even if no improvement is observed.
<code>cv.scope</code>	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.

## Details

The algorithm splits the predictors into several intervals and tries to find a combination of the intervals, which gives best prediction performance. There are two selection methods: "forward" when the intervals are successively included, and "backward" when the intervals are successively excluded from a model. On the first step the algorithm finds the best (forward) or the worst (backward) individual interval. Then it tests the others to find the one which gives the best model in a combination with the already selected/excluded one. The procedure continues until no improvement is observed or the maximum number of iteration is reached.

There are several ways to specify the intervals. First of all either number of intervals (`int.num`) or width of the intervals (`int.width`) can be provided. Alternatively one can specify the limits (first and last variable number) of the intervals manually with `int.limits`.

Cross-validation settings, `cv`, can be a number or a list. If `cv` is a number, it will be used as a number of segments for random cross-validation (if `cv = 1`, full cross-validation will be performed). If it is a list, the following syntax can be used: `cv = list('rand', nseg, nrep)` for random repeated cross-validation with `nseg` segments and `nrep` repetitions or `cv = list('ven', nseg)` for systematic splits to `nseg` segments ('venetian blinds').

## Value

object of 'ipls' class with several fields, including:

<code>var.selected</code>	a vector with indices of selected variables
<code>int.selected</code>	a vector with indices of selected intervals
<code>int.num</code>	total number of intervals
<code>int.width</code>	width of the intervals

int.limits	a matrix with limits for each interval
int.stat	a data frame with statistics for the selection algorithm
glob.stat	a data frame with statistics for the first step (individual intervals)
gm	global PLS model with all variables included
om	optimized PLS model with selected variables

## References

[1] Lars Noergaard et al. Interval partial least-squares regression (iPLS): a comparative chemometric study with an example from near-infrared spectroscopy. *Appl.Spec.* 2000; 54: 413-419

## Examples

```
library(mdatools)

## forward selection for simdata

data(simdata)
Xc = simdata$spectra.c
yc = simdata$conc.c[, 3, drop = FALSE]

# run iPLS and show results
im = ipls(Xc, yc, int.ncomp = 5, int.num = 10, cv = 4, method = "forward")
summary(im)
plot(im)

# show "developing" of RMSECV during the algorithm execution
plotRMSE(im)

# plot predictions before and after selection
par(mfrow = c(1, 2))
plotPredictions(im$gm)
plotPredictions(im$om)

# show selected intervals on spectral plot
ind = im$var.selected
mspectrum = apply(Xc, 2, mean)
plot(simdata$wavelength, mspectrum, type = 'l', col = 'lightblue')
points(simdata$wavelength[ind], mspectrum[ind], pch = 16, col = 'blue')
```

---

ipls.backward

*Runs the backward iPLS algorithm*


---

## Description

Runs the backward iPLS algorithm

**Usage**

```
ipls.backward(x, y, obj, int.stat, glob.stat, full, cv.scope)
```

**Arguments**

x	a matrix with predictor values.
y	a vector with response values.
obj	object with initial settings for iPLS algorithm.
int.stat	data frame with initial interval statistics.
glob.stat	data frame with initial global statistics.
full	logical, if TRUE the procedure will continue even if no improvement is observed.
cv.scope	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.

---

ipls.forward	<i>Runs the forward iPLS algorithm</i>
--------------	--

---

**Description**

Runs the forward iPLS algorithm

**Usage**

```
ipls.forward(x, y, obj, int.stat, glob.stat, full, cv.scope)
```

**Arguments**

x	a matrix with predictor values.
y	a vector with response values.
obj	object with initial settings for iPLS algorithm.
int.stat	data frame with initial interval statistics.
glob.stat	data frame with initial global statistics.
full	logical, if TRUE the procedure will continue even if no improvement is observed.
cv.scope	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.

---

jm.crit	<i>Calculate critical limits for distance values using Jackson-Mudholkar approach</i>
---------	---

---

**Description**

Calculate critical limits for distance values using Jackson-Mudholkar approach

**Usage**

```
jm.crit(residuals, eigenvals, alpha = 0.05, gamma = 0.01)
```

**Arguments**

residuals	matrix with PCA residuals
eigenvals	vector with eigenvalues for PCA components
alpha	significance level for extreme objects
gamma	significance level for outliers

**Value**

vector with four values: critical limits for given alpha and gamma, mean distance and DoF.

---

jm.prob	<i>Calculate probabilities for distance values and given parameters using Hotelling T2 distribution</i>
---------	---

---

**Description**

Calculate probabilities for distance values and given parameters using Hotelling T2 distribution

**Usage**

```
jm.prob(u, eigenvals, ncomp)
```

**Arguments**

u	vector with distances
eigenvals	vector with eigenvalues for PCA components
ncomp	number of components

---

ldecomp	<i>Class for storing and visualising linear decomposition of dataset (<math>X = TP' + E</math>)</i>
---------	---

---

### Description

Creates an object of ldecomp class.

### Usage

```
ldecomp(scores, loadings, residuals, eigenvals, ncomp.selected = ncol(scores))
```

### Arguments

scores	matrix with score values (I x A).
loadings	matrix with loading values (J x A).
residuals	matrix with data residuals (I x J)
eigenvals	vector with eigenvalues for the loadings
ncomp.selected	number of selected components

### Details

ldecomp is a general class for storing results of decomposition of dataset in form  $X = TP' + E$ . Here, X is a data matrix, T - matrix with scores, P - matrix with loadings and E - matrix with residuals. It is used, for example, for PCA results ([pcares](#)), in PLS and other methods. The class also includes methods for calculation of residual distances and explained variance.

There is no need to use the ldecomp class manually. For example, when building a PCA model with [pca](#) or apply it to new data, the results will automatically inherit all methods of ldecomp.

### Value

Returns an object (list) of ldecomp class with following fields:

scores	matrix with score values (I x A).
residuals	matrix with data residuals (I x J).
T2	matrix with score distances (I x A).
Q	matrix with orthogonal distances (I x A).
ncomp.selected	selected number of components.
expvar	explained variance for each component.
cumexpvar	cumulative explained variance.

---

ldecomp.getDistances    *Compute score and residual distances*

---

### Description

Compute orthogonal Euclidean distance from object to PC space (Q, q) and Mahalanobis squared distance between projection of the object to the space and its origin (T2, h).

### Usage

```
ldecomp.getDistances(scores, loadings, residuals, eigenvals)
```

### Arguments

scores	matrix with scores (T).
loadings	matrix with loadings (P).
residuals	matrix with residuals (E).
eigenvals	vector with eigenvalues for the components

### Details

The distances are calculated for every 1:n components, where n goes from 1 to ncomp (number of columns in scores and loadings).

### Value

Returns a list with Q, T2 and tnorm values for each component.

---

ldecomp.getLimitsCoordinates  
*Compute coordinates of lines or curves with critical limits*

---

### Description

Compute coordinates of lines or curves with critical limits

### Usage

```
ldecomp.getLimitsCoordinates(
  Qlim,
  T2lim,
  ncomp,
  norm,
  log,
  show.limits = c(TRUE, TRUE)
)
```

**Arguments**

Qlim	matrix with critical limits for orthogonal distances
T2lim	matrix with critical limits for score distances
ncomp	number of components for computing the coordinates
norm	logical, shall distance values be normalized or not
log	logical, shall log transformation be applied or not
show.limits	vector with two logical values defining if limits for extreme and/or outliers must be shown

**Value**

list with two matrices (x and y coordinates of corresponding limits)

---

ldecomp.getLimParams *Compute parameters for critical limits based on calibration results*

---

**Description**

Compute parameters for critical limits based on calibration results

**Usage**

```
ldecomp.getLimParams(U, do.round = TRUE)
```

**Arguments**

U	matrix with residual distances
do.round	logical, shall Nu values be rounded or not

---

ldecomp.getQLimits *Compute critical limits for orthogonal distances (Q)*

---

**Description**

Compute critical limits for orthogonal distances (Q)

**Usage**

```
ldecomp.getQLimits(lim.type, alpha, gamma, params, residuals, eigenvals)
```

**Arguments**

lim.type	which method to use for calculation of critical limits for residuals
alpha	significance level for extreme limits.
gamma	significance level for outlier limits.
params	distribution parameters returned by ldecomp.getLimParams
residuals	matrix with residuals (E)
eigenvals	eigenvalues for the components used to decompose the data

---

ldecomp.getT2Limits    *Compute critical limits for score distances (T2)*

---

**Description**

Compute critical limits for score distances (T2)

**Usage**

```
ldecomp.getT2Limits(lim.type, alpha, gamma, params)
```

**Arguments**

lim.type	which method to use for calculation ("chisq", "ddmoments", "ddrobust")
alpha	significance level for extreme limits.
gamma	significance level for outlier limits.
params	distribution parameters returned by ldecomp.getLimParams

---

ldecomp.getVariances    *Compute explained variance*

---

**Description**

Computes explained variance and cumulative explained variance for data decomposition.

**Usage**

```
ldecomp.getVariances(scores, loadings, residuals, Q)
```

**Arguments**

scores	matrix with scores (T).
loadings	matrix with loadings (P).
residuals	matrix with residuals (E).
Q	matrix with squared orthogonal distances.

**Value**

Returns a list with two vectors.

---

`ldecomp.plotDistances` *Distance plot for a set of ldecomp objects*

---

**Description**

Shows a plot with score (T2, h) vs orthogonal (Q, q) distances and corresponding critical limits for given number of components.

**Usage**

```
ldecomp.plotDistances(
  res,
  Qlim,
  T2lim,
  ncomp,
  log = FALSE,
  norm = FALSE,
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = c(TRUE, TRUE),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  show.legend = TRUE,
  legend.position = "topright",
  show.excluded = FALSE,
  ...
)
```

**Arguments**

<code>res</code>	list with result objects to show the plot for
<code>Qlim</code>	matrix with critical limits for orthogonal distance
<code>T2lim</code>	matrix with critical limits for score distance
<code>ncomp</code>	how many components to use (by default optimal value selected for the model will be used)
<code>log</code>	logical, apply log transformation to the distances or not (see details)
<code>norm</code>	logical, normalize distance values or not (see details)
<code>cgroup</code>	color grouping of plot points (works only if one result object is available)
<code>xlim</code>	limits for x-axis (if NULL will be computed automatically)

ylim	limits for y-axis (if NULL will be computed automatically)
show.limits	vector with two logical values defining if limits for extreme and/or outliers must be shown
lim.col	vector with two values - line color for extreme and outlier limits
lim.lwd	vector with two values - line width for extreme and outlier limits
lim.lty	vector with two values - line type for extreme and outlier limits
show.legend	logical, show or not a legend on the plot (needed if several result objects are available)
legend.position	if legend must be shown, where it should be
show.excluded	logical, show or hide rows marked as excluded (attribute 'exclrows').
...	other plot parameters (see mdaplotg for details)

### Details

The function is a bit more advanced version of [plotDistances.ldecomp](#). It allows to show distance values for several result objects (e.g. calibration and test set or calibration and new prediction set) as well as display the corresponding critical limits in form of lines or curves.

Depending on how many result objects your model has or how many you specified manually, using the `res` parameter, the plot behaves in a bit different way.

If only one result object is provided, then it allows to colorise the points using `cgroup` parameter. If two or more result objects are provided, then the function shows distances in groups, and adds corresponding legend.

The function can show distance values normalised ( $h/h_0$  and  $q/q_0$ ) as well as with log transformation ( $\log(1 + h/h_0)$ ,  $\log(1 + q/q_0)$ ). The latter is useful if distribution of the points is skewed and most of them are densely located around bottom left corner.

---

`ldecomp.plotResiduals` *Residuals distance plot for a set of ldecomp objects (legacy, use [ldecomp.plotDistances](#) instead).*

---

### Description

Shows a plot with score (T2, h) vs orthogonal (Q, q) distances and corresponding critical limits for given number of components.

### Usage

```
ldecomp.plotResiduals(res, ...)
```

### Arguments

res	list with result objects to show the plot for
...	other parameters

---

mcr *General class for Multivariate Curve Resolution model*

---

### Description

mcr is used to store and visualise general MCR data and results.

### Usage

```
mcr(x, ncomp, method, exclrows = NULL, exclcols = NULL, info = "", ...)
```

### Arguments

x	spectra of mixtures (as matrix or data frame)
ncomp	number of pure components to resolve
method	function for computing spectra of pure components
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)
exclcols	columns to be excluded from calculations (numbers, names or vector with logical values)
info	text with information about the MCR model
...	other parameters related to specific method

---

mcrals *Multivariate curve resolution using Alternating Least Squares*

---

### Description

mcrals allows resolving spectroscopic data to linear combination of individual spectra and contributions using the alternating least squares (ALS) algorithm with constraints.

### Usage

```
mcrals(
  x,
  ncomp,
  cont.constraints = list(),
  spec.constraints = list(),
  spec.ini = matrix(runif(ncol(x) * ncomp), ncol(x), ncomp),
  cont.forced = matrix(NA, nrow(x), ncomp),
  spec.forced = matrix(NA, ncol(x), ncomp),
  cont.solver = mcrals.nnl,
  spec.solver = mcrals.nnl,
```

```

    exclrows = NULL,
    exclcols = NULL,
    verbose = FALSE,
    max.niter = 100,
    tol = 10^-6,
    info = ""
)

```

## Arguments

<code>x</code>	spectra of mixtures (matrix or data frame).
<code>ncomp</code>	number of components to calculate.
<code>cont.constraints</code>	a list with constraints to be applied to contributions (see details).
<code>spec.constraints</code>	a list with constraints to be applied to spectra (see details).
<code>spec.ini</code>	a matrix with initial estimation of the pure components spectra.
<code>cont.forced</code>	a matrix that allows forcing some of the concentration values (see details).
<code>spec.forced</code>	a matrix that allows forcing some of the spectra values (see details).
<code>cont.solver</code>	which function to use as a solver for resolving of pure components contributions (see details).
<code>spec.solver</code>	which function to use as a solver for resolving of pure components spectra (see details).
<code>exclrows</code>	rows to be excluded from calculations (numbers, names or vector with logical values).
<code>exclcols</code>	columns to be excluded from calculations (numbers, names or vector with logical values).
<code>verbose</code>	logical, if TRUE information about every iteration will be shown.
<code>max.niter</code>	maximum number of iterations.
<code>tol</code>	tolerance, when explained variance change is smaller than this value, iterations stop.
<code>info</code>	a short text with description of the case (optional).

## Details

The method implements the iterative ALS algorithm, where, at each iteration, spectra and contributions of each chemical component are estimated and then a set of constraints is applied to each. The method is well described in [1, 2].

The method assumes that the spectral data ( $D$ ) is a linear combination of pure components spectra ( $S$ ) and pure component concentrations ( $C$ ):

$$D = CS' + E$$

So the task is to get  $C$  and  $S$  by knowing  $D$ . In order to do that you need to provide:

1. Constraints for spectra and contributions. The constraints should be provided as a list with name of the constraint and all necessary parameters. You can see which constraints and parameters are

currently supported by running `constraintList()`. See the code examples below or a Bookdown tutorial for more details.

2. Initial estimation of the pure components spectra,  $S$ . By default method uses a matrix with random numbers but you can provide a better guess (for example by running `mcrpure`) as a first step.

3. Which solver to use for resolving spectra and concentrations. There are two built in solvers: `mcrals.nls` (default) and `mcrals.ols`. The first implements non-negative least squares method which gives non-negative (thus physically meaningful) solutions. The second is ordinary least squares and if you want to get non-negative spectra and/or contributions in this case you need to provide a non-negativity constraint.

The algorithm iteratively resolves  $C$  and  $S$  and checks how well  $CS'$  is to  $D$ . The iterations stop either when number exceeds value in `max.niter` or when improvements (difference between explained variance on current and previous steps) is smaller than `tol` value.

Parameters `cont.forced` and `spec.forced` allow you to force some parts of the contributions or the spectra to be equal to particular pre-defined values. In this case you need to provide the parameters (or just one of them) in form of a matrix. For example `cont.forced` should have as many rows as you have in the original spectral data  $x$  and as many columns as many pure components you want to resolve. Fill all values of this matrix with `NA` and the values you want to force with real numbers. For example if you know that in the first measurement concentration of 2 and 3 components was zero, set the corresponding values of `cont.force` to zero. See also the last case in the examples section.

## Value

Returns an object of `mcrals` class with the following fields:

<code>resspec</code>	matrix with resolved spectra.
<code>rescont</code>	matrix with resolved contributions.
<code>cont.constraints</code>	list with contribution constraints provided by user.
<code>spec.constraints</code>	list with spectra constraints provided by user.
<code>expvar</code>	vector with explained variance for each component (in percent).
<code>cumexpvar</code>	vector with cumulative explained variance for each component (in percent).
<code>ncomp</code>	number of resolved components
<code>max.niter</code>	maximum number of iterations
<code>info</code>	information about the model, provided by user when building the model.

More details and examples can be found in the Bookdown tutorial.

## Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

## References

1. J. Jaumot, R. Gargallo, A. de Juan, and R. Tauler, "A graphical user-friendly interface for MCR-ALS: a new tool for multivariate curve resolution in MATLAB", *Chemometrics and Intelligent # Laboratory Systems* 76, 101-110 (2005).

## See Also

Methods for mcrals objects:

`summary.mcrals` shows some statistics for the case.  
`predict.mcrals` computes contributions by projection of new spectra to the resolved ones.

Plotting methods for mcrals objects:

<code>plotSpectra.mcr</code>	shows plot with resolved spectra.
<code>plotContributions.mcr</code>	shows plot with resolved contributions.
<code>plotVariance.mcr</code>	shows plot with explained variance.
<code>plotCumVariance.mcr</code>	shows plot with cumulative explained variance.

## Examples

```
library(mdatools)

# resolve mixture of carbohydrates Raman spectra

data(carbs)

# define constraints for contributions
cc <- list(
  constraint("nonneg")
)

# define constraints for spectra
cs <- list(
  constraint("nonneg"),
  constraint("norm", params = list(type = "area"))
)

# because by default initial approximation is made by using random numbers
# we need to seed the generator in order to get reproducible results
set.seed(6)

# run ALS
m <- mcrals(carbs$D, ncomp = 3, cont.constraints = cc, spec.constraints = cs)
summary(m)
```

```

# plot cumulative and individual explained variance
par(mfrow = c(1, 2))
plotVariance(m)
plotCumVariance(m)

# plot resolved spectra (all of them or individually)
par(mfrow = c(2, 1))
plotSpectra(m)
plotSpectra(m, comp = 2:3)

# plot resolved contributions (all of them or individually)
par(mfrow = c(2, 1))
plotContributions(m)
plotContributions(m, comp = 2:3)

# of course you can do this manually as well, e.g. show original
# and resolved spectra
par(mfrow = c(1, 1))
mdaplotg(
  list(
    "original" = prep.norm(carbs$D, "area"),
    "resolved" = prep.norm(mda.subset(mda.t(m$respec), 1), "area")
  ), col = c("gray", "red"), type = "l"
)

# in case if you have reference spectra of components you can compare them with
# the resolved ones:
par(mfrow = c(3, 1))
for (i in 1:3) {
  mdaplotg(
    list(
      "pure" = prep.norm(mda.subset(mda.t(carbs$$), 1), "area"),
      "resolved" = prep.norm(mda.subset(mda.t(m$respec), 1), "area")
    ), col = c("gray", "red"), type = "l", lwd = c(3, 1)
  )
}

# This example shows how to force some of the contribution values
# First of all we combine the matrix with mixtures and the pure spectra, so the pure
# spectra are on top of the combined matrix
Dplus <- mda.rbind(mda.t(carbs$$), carbs$D)

# since we know that concentration of C2 and C3 is zero in the first row (it is a pure
# spectrum of first component), we can force them to be zero in the optimization procedure.
# Similarly we can do this for second and third rows.

cont.forced <- matrix(NA, nrow(Dplus), 3)
cont.forced[1, ] <- c(NA, 0, 0)
cont.forced[2, ] <- c(0, NA, 0)
cont.forced[3, ] <- c(0, 0, NA)

m <- mcrals(Dplus, 3, cont.forced = cont.forced, cont.constraints = cc, spec.constraints = cs)
plot(m)

```

```
# See bookdown tutorial for more details.
```

---

```
mcrals.cal
```

```
MCR-ALS calibration
```

---

## Description

The method resolves spectral data into pure component spectra and contributions using the Alternating Least Squares algorithm with constraints.

## Usage

```
mcrals.cal(
  D,
  ncomp,
  cont.constraints,
  spec.constraints,
  spec.ini,
  cont.forced,
  spec.forced,
  cont.solver,
  spec.solver,
  max.niter,
  tol,
  verbose
)
```

## Arguments

D	matrix with the spectra
ncomp	number of pure components
cont.constraints	a list with constraints to be applied to contributions (see details).
spec.constraints	a list with constraints to be applied to spectra (see details).
spec.ini	a matrix with initial estimation of the pure components spectra.
cont.forced	a matrix that allows forcing some of the concentration values (see details).
spec.forced	a matrix that allows forcing some of the spectra values (see details).
cont.solver	which function to use as a solver for resolving of pure components contributions (see details).
spec.solver	which function to use as a solver for resolving of pure components spectra (see details).

max.niter	maximum number of iterations.
tol	tolerance, when explained variance change is smaller than this value, iterations stop.
verbose	logical, if TRUE information about every iteration will be shown.

**Value**

The function returns a list with the following fields:

ncomp	number of pure components.
resspec	matrix with resolved spectra.
rescont	matrix with resolved contributions.
cont.constraints	list with contribution constraints provided by user.
spec.constraints	list with spectra constraints provided by user.
max.niter	maximum number of iterations

---

mcrals.fcnnls	<i>Fast combinatorial non-negative least squares</i>
---------------	--

---

**Description**

Fast combinatorial non-negative least squares

**Usage**

```
mcrals.fcnnls(
  D,
  A,
  tol = 10 * .Machine$double.eps * as.numeric(sqrt(crossprod(A[, 1]))) * nrow(A)
)
```

**Arguments**

D	a matrix
A	a matrix
tol	tolerance parameter for algorithm convergence

**Details**

Computes Fast combinatorial NNLS solution for B:  $D = AB'$  subject to  $B \geq 0$ . Implements the method described in [1].

**References**

1. Van Benthem, M.H. and Keenan, M.R. (2004), Fast algorithm for the solution of large scale non-negativity-constrained least squares problems. J. Chemometrics, 18: 441-450. doi:10.1002/cem.889

---

mcrals.npls                      *Non-negative least squares*

---

**Description**

Non-negative least squares

**Usage**

```
mcrals.npls(  
  D,  
  A,  
  tol = 10 * .Machine$double.eps * as.numeric(sqrt(crossprod(A[, 1]))) * nrow(A)  
)
```

**Arguments**

D	a matrix
A	a matrix
tol	tolerance parameter for algorithm convergence

**Details**

Computes NNLS solution for  $B: D = AB'$  subject to  $B \geq 0$ . Implements the active-set based algorithm proposed by Lawson and Hanson [1].

**References**

1. Lawson, Charles L.; Hanson, Richard J. (1995). Solving Least Squares Problems. SIAM.

---

mcrals.ols                      *Ordinary least squares*

---

**Description**

Ordinary least squares

**Usage**

```
mcrals.ols(D, A)
```

**Arguments**

D	a matrix
A	a matrix

**Details**

Computes OLS solution for  $D = AB'$  (or  $D' = AB'$ ), where D, A are known

---

mcrpure	<i>Multivariate curve resolution based on pure variables</i>
---------	--

---

**Description**

mcrpure allows resolving spectroscopic data to linear combination of individual spectra and contributions using the pure variables approach.

**Usage**

```
mcrpure(
  x,
  ncomp,
  purevars = NULL,
  offset = 0.05,
  exclrows = NULL,
  exclcols = NULL,
  info = ""
)
```

**Arguments**

x	spectra of mixtures (matrix or data frame).
ncomp	maximum number of components to calculate.
purevars	vector with indices for pure variables (optional, if you want to provide the variables directly).
offset	offset for correcting noise in computing maximum angles (should be value within [0, 1)).
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values).
exclcols	columns to be excluded from calculations (numbers, names or vector with logical values).
info	a short text with description of the case (optional).

**Details**

The method estimates purity of each variable and then uses the purest ones to decompose the spectral data into spectra ('resspec') and contributions ('rescont') of individual chemical components by ordinary least squares.

The pure variables are identified using stepwise maximum angle calculations and described in detail in [1]. So the purity of a spectral variable (wavelength, wavenumber) is actually an angle (measured in degrees) between the variable and vector of ones for the first component; and between the variable and space formed by previously found pure variables for the other components.

**Value**

Returns an object of `mcrpure` class with the following fields:

<code>resspec</code>	matrix with resolved spectra.
<code>rescont</code>	matrix with resolved contributions.
<code>purevars</code>	indices of the selected pure variables.
<code>purevals</code>	purity values for the selected pure variables.
<code>purityspect</code>	purity spectra (matrix with purity values for each variable and component).
<code>expvar</code>	vector with explained variance for each component (in percent).
<code>cumexpvar</code>	vector with cumulative explained variance for each component (in percent).
<code>offset</code>	offset value used to compute the purity
<code>ncomp</code>	number of resolved components
<code>info</code>	information about the model, provided by user when building the model.

More details and examples can be found in the Bookdown tutorial.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**References**

1. Willem Windig, Neal B. Gallagher, Jeremy M. Shaver, Barry M. Wise. A new approach for interactive self-modeling mixture analysis. *Chemometrics and Intelligent Laboratory Systems*, 77 (2005) 85–96. DOI: 10.1016/j.chemolab.2004.06.009

**See Also**

Methods for `mcrpure` objects:

<code>summary.mcrpure</code>	shows some statistics for the case.
<code>unmix.mcrpure</code>	makes unmixing of new set of spectra.
<code>predict.mcrpure</code>	computes contributions by projection of new spectra to the resolved ones.

Plotting methods for `mcrpure` objects:

<code>plotPurity.mcrpure</code>	shows plot with maximum purity of each component.
<code>plotPuritySpectra.mcrpure</code>	shows plot with purity spectra.
<code>plotSpectra.mcr</code>	shows plot with resolved spectra.
<code>plotContributions.mcr</code>	shows plot with resolved contributions.
<code>plotVariance.mcr</code>	shows plot with explained variance.
<code>plotCumVariance.mcr</code>	shows plot with cumulative explained variance.

**Examples**

```

library(mdatools)

# resolve mixture of carbohydrates Raman spectra

data(carbs)
m = mcrpure(carbs$D, ncomp = 3)

# examples for purity spectra plot (you can select which components to show)
par(mfrow = c(2, 1))
plotPuritySpectra(m)
plotPuritySpectra(m, comp = 2:3)

# you can do it manually and combine e.g. with original spectra
par(mfrow = c(1, 1))
mdaplotg(
  list(
    "spectra" = prep.norm(carbs$D, "area"),
    "purity" = prep.norm(mda.subset(mda.t(m$respec), 1), "area")
  ), col = c("gray", "red"), type = "l"
)

# show the maximum purity for each component
par(mfrow = c(1, 1))
plotPurity(m)

# plot cumulative and individual explained variance
par(mfrow = c(1, 2))
plotVariance(m)
plotCumVariance(m)

# plot resolved spectra (all of them or individually)
par(mfrow = c(2, 1))
plotSpectra(m)
plotSpectra(m, comp = 2:3)

# plot resolved contributions (all of them or individually)
par(mfrow = c(2, 1))
plotContributions(m)
plotContributions(m, comp = 2:3)

# of course you can do this manually as well, e.g. show original
# and resolved spectra
par(mfrow = c(1, 1))
mdaplotg(
  list(
    "original" = prep.norm(carbs$D, "area"),
    "resolved" = prep.norm(mda.subset(mda.t(m$respec), 1), "area")
  ), col = c("gray", "red"), type = "l"
)

# in case if you have reference spectra of components you can compare them with

```

```

# the resolved ones:
par(mfrow = c(3, 1))
for (i in 1:3) {
  mdaplotg(
    list(
      "pure" = prep.norm(mda.subset(mda.t(carbs$S), 1), "area"),
      "resolved" = prep.norm(mda.subset(mda.t(m$respec), 1), "area")
    ), col = c("gray", "red"), type = "l", lwd = c(3, 1)
  )
}

# See bookdown tutorial for more details.

```

---

mda.cbind	<i>A wrapper for cbind() method with proper set of attributes</i>
-----------	---

---

**Description**

A wrapper for cbind() method with proper set of attributes

**Usage**

```
mda.cbind(...)
```

**Arguments**

... datasets (data frames or matrices) to bind

**Value**

the merged datasets

---

mda.data2im	<i>Convert data matrix to an image</i>
-------------	--

---

**Description**

Convert data matrix to an image

**Usage**

```
mda.data2im(data)
```

**Arguments**

data data matrix

---

mda.df2mat	<i>Convert data frame to a matrix</i>
------------	---------------------------------------

---

**Description**

The function converts data frame to a numeric matrix.

**Usage**

```
mda.df2mat(x, full = FALSE)
```

**Arguments**

x	a data frame
full	logical, if TRUE number of dummy variables for a factor will be the same as number of levels, otherwise by one smaller

**Details**

If one or several columns of the data frame are factors they will be converted to a set of dummy variables. If any columns/rows were hidden in the data frame they will remain hidden in the matrix. If there are factors among the hidden columns, the corresponding dummy variables will be hidden as well.

All other attributes (names, axis names, etc.) will be inherited.

**Value**

a numeric matrix

---

mda.exclcols	<i>Exclude/hide columns in a dataset</i>
--------------	--

---

**Description**

Exclude/hide columns in a dataset

**Usage**

```
mda.exclcols(x, ind)
```

**Arguments**

x	dataset (data frame or matrix).
ind	indices of columns to exclude (numbers, names or logical values)

**Details**

The method assigns attribute 'exclcols', which contains number of columns, which should be excluded/hidden from calculations and plots (without removing them physically). The argument `ind` should contain column numbers (excluding already hidden), names or logical values.

**Value**

dataset with excluded columns

---

<code>mda.exclrows</code>	<i>Exclude/hide rows in a dataset</i>
---------------------------	---------------------------------------

---

**Description**

Exclude/hide rows in a dataset

**Usage**

```
mda.exclrows(x, ind)
```

**Arguments**

<code>x</code>	dataset (data frame or matrix).
<code>ind</code>	indices of rows to exclude (numbers, names or logical values)

**Details**

The method assigns attribute 'exclrows', which contains number of rows, which should be excluded/hidden from calculations and plots (without removing them physically). The argument `ind` should contain rows numbers (excluding already hidden), names or logical values.

**Value**

dataset with excluded rows

---

<code>mda.getattr</code>	<i>Get data attributes</i>
--------------------------	----------------------------

---

**Description**

Returns a list with important data attributes (name, xvalues, excluded rows and columns, etc.)

**Usage**

```
mda.getattr(x)
```

**Arguments**

<code>x</code>	a dataset
----------------	-----------

---

<code>mda.getexclind</code>	<i>Get indices of excluded rows or columns</i>
-----------------------------	--

---

**Description**

Get indices of excluded rows or columns

**Usage**

```
mda.getexclind(excl, names, n)
```

**Arguments**

<code>excl</code>	vector with excluded values (logical, text or numbers)
<code>names</code>	vector with names for rows or columns
<code>n</code>	number of rows or columns

---

<code>mda.im2data</code>	<i>Convert image to data matrix</i>
--------------------------	-------------------------------------

---

**Description**

Convert image to data matrix

**Usage**

```
mda.im2data(img)
```

**Arguments**

<code>img</code>	an image (3-way array)
------------------	------------------------

---

mda.inclcols	<i>Include/unhide the excluded columns</i>
--------------	--

---

**Description**

include columns specified by user (earlier excluded using mda.exclcols)

**Usage**

```
mda.inclcols(x, ind)
```

**Arguments**

x	dataset (data frame or matrix).
ind	number of excluded columns to include

**Value**

dataset with included columns.

---

mda.inclrows	<i>include/unhide the excluded rows</i>
--------------	---

---

**Description**

include rows specified by user (earlier excluded using mda.exclrows)

**Usage**

```
mda.inclrows(x, ind)
```

**Arguments**

x	dataset (data frame or matrix).
ind	number of excluded rows to include

**Value**

dataset with included rows

---

mda.purge	<i>Removes excluded (hidden) rows and columns from data</i>
-----------	---

---

**Description**

Removes excluded (hidden) rows and columns from data

**Usage**

```
mda.purge(data)
```

**Arguments**

data	data frame or matrix with data
------	--------------------------------

---

mda.purgeCols	<i>Removes excluded (hidden) columns from data</i>
---------------	--

---

**Description**

Removes excluded (hidden) columns from data

**Usage**

```
mda.purgeCols(data)
```

**Arguments**

data	data frame or matrix with data
------	--------------------------------

---

mda.purgeRows	<i>Removes excluded (hidden) rows from data</i>
---------------	---

---

**Description**

Removes excluded (hidden) rows from data

**Usage**

```
mda.purgeRows(data)
```

**Arguments**

data	data frame or matrix with data
------	--------------------------------

---

mda.rbind	<i>A wrapper for rbind() method with proper set of attributes</i>
-----------	---

---

**Description**

A wrapper for rbind() method with proper set of attributes

**Usage**

```
mda.rbind(...)
```

**Arguments**

... datasets (data frames or matrices) to bind

**Value**

the merged datasets

---

mda.setattr	<i>Set data attributes</i>
-------------	----------------------------

---

**Description**

Set most important data attributes (name, xvalues, excluded rows and columns, etc.) to a dataset

**Usage**

```
mda.setattr(x, attrs, type = "all")
```

**Arguments**

x	a dataset
attrs	list with attributes
type	a text variable telling which attributes to set ('all', 'row', 'col')

---

mda.setimbg	<i>Remove background pixels from image data</i>
-------------	---

---

**Description**

Remove background pixels from image data

**Usage**

```
mda.setimbg(data, bgpixels)
```

**Arguments**

data	a matrix with image data
bgpixels	vector with indices or logical values corresponding to background pixels

---

mda.show	<i>Wrapper for show() method</i>
----------	----------------------------------

---

**Description**

Wrapper for show() method

**Usage**

```
mda.show(x, n = 50)
```

**Arguments**

x	data set
n	number of rows to show

---

mda.subset	<i>A wrapper for subset() method with proper set of attributes</i>
------------	--

---

**Description**

A wrapper for subset() method with proper set of attributes

**Usage**

```
mda.subset(x, subset = NULL, select = NULL)
```

**Arguments**

x	dataset (data frame or matrix)
subset	which rows to keep (indices, names or logical values)
select	which columns to select (indices, names or logical values)

**Details**

The method works similar to the standard subset() method, with minor differences. First of all it keeps (and corrects, if necessary) all important attributes. If only columns are selected, it keeps all excluded rows as excluded. If only rows are selected, it keeps all excluded columns. If both rows and columns are selected it removes all excluded elements first and then makes the subset.

The parameters subset and select may each be a vector with numbers or names without excluded elements, or a logical expression.

**Value**

a data with the subset

---

mda.t	<i>A wrapper for t() method with proper set of attributes</i>
-------	---

---

**Description**

A wrapper for t() method with proper set of attributes

**Usage**

```
mda.t(x)
```

**Arguments**

x	dataset (data frames or matrices) to transpose
---	--

**Value**

the transposed dataset

---

mdaplot

*Plotting function for a single set of objects*

---

**Description**

mdaplot is used to make different kinds of plot for one set of data objects.

**Usage**

```
mdaplot(  
  data = NULL,  
  ps = NULL,  
  type = "p",  
  pch = 16,  
  col = NULL,  
  bg = par("bg"),  
  bwd = 0.8,  
  border = NA,  
  lty = 1,  
  lwd = 1,  
  cex = 1,  
  cgroup = NULL,  
  xlim = NULL,  
  ylim = NULL,  
  colmap = "default",  
  labels = NULL,  
  main = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  show.labels = FALSE,  
  show.colorbar = !is.null(cgroup),  
  show.lines = FALSE,  
  show.grid = TRUE,  
  grid.lwd = 0.5,  
  grid.col = "lightgray",  
  show.axes = TRUE,  
  xticks = NULL,  
  yticks = NULL,  
  xticklabels = NULL,  
  yticklabels = NULL,  
  xlas = 0,  
  ylas = 0,  
  lab.col = "darkgray",
```

```

    lab.cex = 0.65,
    show.excluded = FALSE,
    col.excluded = "#C0C0C0",
    nbins = 60,
    force.x.values = NA,
    opacity = 1,
    pch.colinv = FALSE,
    ...
)

```

### Arguments

data	a vector, matrix or a data.frame with data values.
ps	'plotseries' object, if NULL will be created based on the provided data values
type	type of the plot ("p", "d", "l", "b", "o", "h", "e").
pch	a character for markers (same as plot parameter).
col	a color for markers or lines (same as plot parameter).
bg	background color for scatter plots with 'pch=21:25'.
bwd	a width of a bar as a percent of a maximum space available for each bar.
border	color for border of bars (if barplot is used)
lty	line type
lwd	line width
cex	scale factor for the marker
cgroup	a vector with values to use for make color groups.
xlim	limits for the x axis (if NULL, will be calculated automatically).
ylim	limits for the y axis (if NULL, will be calculated automatically).
colmap	a colormap to use for coloring the plot items.
labels	a vector with text labels for data points or one of the following: "names", "indices", "values".
main	an overall title for the plot (same as plot parameter).
xlab	a title for the x axis (same as plot parameter).
ylab	a title for the y axis (same as plot parameter).
show.labels	logical, show or not labels for the data objects.
show.colorbar	logical, show or not colorbar legend if color grouping is on.
show.lines	vector with two coordinates (x, y) to show horizontal and vertical line cross the point.
show.grid	logical, show or not a grid for the plot.
grid.lwd	line thickness (width) for the grid.
grid.col	line color for the grid.
show.axes	logical, make a normal plot or show only elements (markers, lines, bars) without axes.

<code>xticks</code>	values for x ticks.
<code>yticks</code>	values for y ticks.
<code>xticklabels</code>	labels for x ticks.
<code>yticklabels</code>	labels for y ticks.
<code>xlas</code>	orientation of xticklabels.
<code>ylas</code>	orientation of yticklabels.
<code>lab.col</code>	color for data point labels.
<code>lab.cex</code>	size for data point labels.
<code>show.excluded</code>	logical, show or hide rows marked as excluded (attribute <code>'exclrows'</code> ).
<code>col.excluded</code>	color for the excluded objects (rows).
<code>nbins</code>	if scatter density plot is shown, number of segments to split the plot area into. (see also <code>?smoothScatter</code> )
<code>force.x.values</code>	vector with corrected x-values for a bar plot (do not specify this manually).
<code>opacity</code>	opacity for plot colors (value between 0 and 1).
<code>pch.colinv</code>	allows to swap values for <code>'col'</code> and <code>'bg'</code> for scatter plots with <code>'pch'</code> values from 21 to 25.
<code>...</code>	other plotting arguments.

## Details

Most of the parameters are similar to what are used with standard `plot` function. The differences are described below.

The function makes a plot of one set of objects. It can be a set of points (scatter plot), bars, lines, scatter-lines, errorbars or an image. The data is organized as a data frame, matrix or vector. For scatter plots only the first two columns will be used, for bar plot only values from the first row. It is recommended to use `mda.subset` method if plot should be made only for a subset of the data, especially if you have any excluded rows or columns or other special attributed, described in the Bookdown tutorial.

If data is a data frame and contains one or more factors, they will be converted to a dummy variables (using function `mda.df2mat`) and appears at the end (last columns) if line or bar plot is selected.

The function allows to colorize lines and points according to values of a parameter `cgroup`. The parameter must be a vector with the same elements as number of objects (rows) in the data. The values are divided into up to eight intervals and for each interval a particular color from a selected color scheme is assigned. Parameter `show.colorbar` allows to turn off and on a color bar legend for this option.

The used color scheme is defined by the `colmap` parameter. The default scheme is based on color brewer ([colorbrewer2.org](http://colorbrewer2.org)) diverging scheme with eight colors. There is also a gray scheme (`colmap = "gray"`) and user can define its own just by specifying the needed sequence of colors (e.g. `colmap = c("red", "yellow", "green")`, two colors is minimum). The scheme will then be generated automatically as a gradient among the colors.

Besides that the function allows to change tick values and corresponding tick labels for x and y axis, see Bookdown tutorial for more details.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**See Also**

[mdaplotg](#) - to make plots for several sets of data objects (groups of objects).

**Examples**

```
# See all examples in the tutorial.
```

---

mdaplot.areColors	<i>Check color values</i>
-------------------	---------------------------

---

**Description**

Checks if elements of argument are valid color values

**Usage**

```
mdaplot.areColors(palette)
```

**Arguments**

palette	vector with possibly color values (names, RGB, etc.)
---------	--

---

mdaplot.formatValues	<i>Format vector with numeric values</i>
----------------------	--

---

**Description**

Format vector with values, so only significant decimal numbers are left.

**Usage**

```
mdaplot.formatValues(data, round.only = FALSE, digits = 3)
```

**Arguments**

data	vector or matrix with values
round.only	logical, do formatting or only round the values
digits	how many significant digits take into account

**Details**

Function takes into account difference between values and the values themselves.

**Value**

matrix with formatted values

---

mdaplot.getColors	<i>Color values for plot elements</i>
-------------------	---------------------------------------

---

**Description**

Generate vector with color values for plot objects (lines, points, bars), depending on number of groups for the objects.

**Usage**

```
mdaplot.getColors(  
  ngroups = NULL,  
  cgroup = NULL,  
  colmap = "default",  
  opacity = 1,  
  maxsplits = 64  
)
```

**Arguments**

ngroups	number of colors to create.
cgroup	vector of values, used for color grouping of plot points or lines.
colmap	which colormap to use ('default', 'gray', 'old', or user defined in form c('col1', 'col2', ...)).
opacity	opacity for colors (between 0 and 1)
maxsplits	if continuous values are used for color grouping - how many groups to create?

**Value**

Returns vector with generated color values

---

mdaplot.getXAxisLim     *Calculate limits for x-axis.*

---

### Description

Calculates limits for x-axis depending on data values that have to be plotted, extra plot elements that have to be shown and margins.

### Usage

```
mdaplot.getXAxisLim(  
  ps,  
  xlim,  
  show.labels = FALSE,  
  show.lines = FALSE,  
  show.excluded = FALSE,  
  bwd = 0.8  
)
```

### Arguments

ps	'plotseries' object.
xlim	limits provided by user
show.labels	logical, will data labels be shown on the plot
show.lines	logical or numeric with line coordinates to be shown on the plot.
show.excluded	logical, will excluded values be shown on the plot
bwd	if limits are computed for bar plot, this is a bar width (otherwise NULL)

### Value

Returns a vector with two limits.

---

mdaplot.getXTickLabels     *Prepare xticklabels for plot*

---

### Description

Prepare xticklabels for plot

### Usage

```
mdaplot.getXTickLabels(xticklabels, xticks, excluded_cols)
```

**Arguments**

xticklabels	xticklabels provided by user (if any)
xticks	xticks (provided or computed)
excluded_cols	columns excluded from plot data (if any)

---

mdaplot.getXTicks      *Prepare xticks for plot*

---

**Description**

Prepare xticks for plot

**Usage**

```
mdaplot.getXTicks(xticks, xlim, x_values = NULL, type = NULL)
```

**Arguments**

xticks	xticks provided by user (if any)
xlim	limits for x axis
x_values	x values for the plot data object
type	type of the plot

---

mdaplot.getYAxisLim      *Calculate limits for y-axis.*

---

**Description**

Calculates limits for y-axis depending on data values that have to be plotted, extra plot elements that have to be shown and margins.

**Usage**

```
mdaplot.getYAxisLim(
  ps,
  ylim,
  show.lines = FALSE,
  show.excluded = FALSE,
  show.labels = FALSE,
  show.colorbar = FALSE
)
```

**Arguments**

ps	'plotseries' object.
ylim	limits provided by user
show.lines	logical or numeric with line coordinates to be shown on the plot.
show.excluded	logical, will excluded values be shown on the plot
show.labels	logical, will data labels be shown on the plot
show.colorbar	logical, will colorbar be shown on the plot

**Value**

Returns a vector with two limits.

---

mdaplot.getYTickLabels  
*Prepare yticklabels for plot*

---

**Description**

Prepare yticklabels for plot

**Usage**

```
mdaplot.getYTickLabels(yticklabels, yticks)
```

**Arguments**

yticklabels	yticklabels provided by user (if any)
yticks	yticks (provided or computed)

---

mdaplot.getYTicks *Prepare yticks for plot*

---

**Description**

Prepare yticks for plot

**Usage**

```
mdaplot.getYTicks(yticks, ylim, y_values = NULL, type = NULL)
```

**Arguments**

yticks	yticks provided by user (if any)
ylim	limits for y axis
y_values	y values for the plot data object
type	type of the plot

---

mdaplot.plotAxes      *Create axes plane*

---

### Description

Creates an empty axes plane for given parameters

### Usage

```
mdaplot.plotAxes(  
  xticklabels = NULL,  
  yticklabels = NULL,  
  xlim = xlim,  
  ylim = ylim,  
  xticks = NULL,  
  yticks = NULL,  
  main = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  xlas = 0,  
  ylas = 0,  
  show.grid = TRUE,  
  grid.lwd = 0.5,  
  grid.col = "lightgray"  
)
```

### Arguments

xticklabels	labels for x ticks
yticklabels	labels for y ticks
xlim	vector with limits for x axis
ylim	vector with limits for y axis
xticks	values for x ticks
yticks	values for y ticks
main	main title for the plot
xlab	label for x axis
ylab	label for y axis
xlas	orientation of xticklabels
ylas	orientation of yticklabels
show.grid	logical, show or not axes grid
grid.lwd	line thickness (width) for the grid
grid.col	line color for the grid

---

mdaplot.prepareColors *Prepare colors based on palette and opacity value*

---

**Description**

Prepare colors based on palette and opacity value

**Usage**

```
mdaplot.prepareColors(palette, ncolors, opacity)
```

**Arguments**

palette	vector with main colors for current palette
ncolors	number of colors to generate
opacity	opacity for the colors (one value or individual for each color)

**Value**

vector with colors

---

mdaplot.showColorbar *Plot colorbar*

---

**Description**

Shows a colorbar if plot has color grouping of elements (points or lines).

**Usage**

```
mdaplot.showColorbar(  
  cgroup,  
  colmap = "default",  
  lab.col = "darkgray",  
  lab.cex = 0.65  
)
```

**Arguments**

cgroup	a vector with values used to make color grouping of the elements
colmap	a colormap to be used for color generation
lab.col	color for legend labels
lab.cex	size for legend labels

---

mdaplot.showLines      *Plot lines*

---

**Description**

Shows horizontal and vertical lines on a plot.

**Usage**

```
mdaplot.showLines(point, lty = 2, lwd = 0.75, col = rgb(0.2, 0.2, 0.2))
```

**Arguments**

point	vector with two values: x coordinate for vertical point y for horizontal
lty	line type
lwd	line width
col	color of lines

**Details**

If it is needed to show only one line, the other coordinate shall be set to NA.

---

mdaplotg      *Plotting function for several plot series*

---

**Description**

mdaplotg is used to make different kinds of plots or their combination for several sets of objects.

**Usage**

```
mdaplotg(  
  data,  
  groupby = NULL,  
  type = "p",  
  pch = 16,  
  lty = 1,  
  lwd = 1,  
  cex = 1,  
  col = NULL,  
  bwd = 0.8,  
  legend = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  main = NULL,
```

```

    labels = NULL,
    ylim = NULL,
    xlim = NULL,
    colmap = "default",
    legend.position = "topright",
    show.legend = TRUE,
    show.labels = FALSE,
    show.lines = FALSE,
    show.grid = TRUE,
    grid.lwd = 0.5,
    grid.col = "lightgray",
    xticks = NULL,
    xticklabels = NULL,
    yticks = NULL,
    yticklabels = NULL,
    show.excluded = FALSE,
    lab.col = "darkgray",
    lab.cex = 0.65,
    xlas = 1,
    ylas = 1,
    opacity = 1,
    ...
)

```

### Arguments

<code>data</code>	a matrix, data frame or a list with data values (see details below).
<code>groupby</code>	one or several factors used to create groups of data matrix rows (works if data is a matrix)
<code>type</code>	type of the plot ('p', 'l', 'b', 'h', 'e').
<code>pch</code>	a character for markers (same as plot parameter).
<code>lty</code>	the line type (same as plot parameter).
<code>lwd</code>	the line width (thickness) (same as plot parameter).
<code>cex</code>	the cex factor for the markers (same as plot parameter).
<code>col</code>	colors for the plot series
<code>bwd</code>	a width of a bar as a percent of a maximum space available for each bar.
<code>legend</code>	a vector with legend elements (if NULL, no legend will be shown).
<code>xlab</code>	a title for the x axis (same as plot parameter).
<code>ylab</code>	a title for the y axis (same as plot parameter).
<code>main</code>	an overall title for the plot (same as plot parameter).
<code>labels</code>	what to use as labels ('names' - row names, 'indices' - row indices, 'values' - values).
<code>ylim</code>	limits for the y axis (if NULL, will be calculated automatically).
<code>xlim</code>	limits for the x axis (if NULL, will be calculated automatically).

colmap	a colormap to generate colors if col is not provided
legend.position	position of the legend ('topleft', 'topright', 'top', 'bottomleft', 'bottomright', 'bottom').
show.legend	logical, show or not legend for the data objects.
show.labels	logical, show or not labels for the data objects.
show.lines	vector with two coordinates (x, y) to show horizontal and vertical line cross the point.
show.grid	logical, show or not a grid for the plot.
grid.lwd	line thickness (width) for the grid
grid.col	line color for the grid
xticks	tick values for x axis.
xticklabels	labels for x ticks.
yticks	tick values for y axis.
yticklabels	labels for y ticks.
show.excluded	logical, show or hide rows marked as excluded (attribute 'exclrows')
lab.col	color for data point labels.
lab.cex	size for data point labels.
xlas	orientation of xticklabels
ylas	orientation of yticklabels
opacity	opacity for plot colors (value between 0 and 1)
...	other plotting arguments.

## Details

The `mdaplotg` function is used to make a plot with several sets of objects. Simply speaking, use it when you need a plot with legend. For example to show line plot with spectra from calibration and test set, scatter plot with height and weight values for women and men, and so on.

Most of the parameters are similar to `mdaplot`, the difference is described below.

The data should be organized as a list, every item is a matrix (or data frame) with data for one set of objects. Alternatively you can provide data as a matrix and use parameter `groupby` to create the groups. See tutorial for more details.

There is no color grouping option, because color is used to separate the sets. Marker symbol, line style and type, etc. can be defined as a single value (one for all sets) and as a vector with one value for each set.

## Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

---

mdaplotg.getLegend      *Create and return vector with legend values*

---

**Description**

Create and return vector with legend values

**Usage**

```
mdaplotg.getLegend(ps, data.names, legend = NULL)
```

**Arguments**

ps	list with plot series
data.names	names of the data sets
legend	legend values provided by user

**Value**

vector of text values for the legend

---

mdaplotg.getXLim      *Compute x-axis limits for mdaplotg*

---

**Description**

Compute x-axis limits for mdaplotg

**Usage**

```
mdaplotg.getXLim(  
  ps,  
  xlim,  
  show.excluded,  
  show.legend,  
  show.labels,  
  legend.position,  
  bwd = NULL  
)
```

**Arguments**

ps	list with plotseries
xlim	limits provided by user
show.excluded	logical, will excluded values also be shown
show.legend	will legend be shown on the plot
show.labels	will labels be shown on the plot
legend.position	position of legend on the plot (if shown)
bwd	size of bar for bar plot

**Value**

vector with two values

---

mdaplotg.getYLim	<i>Compute y-axis limits for mdaplotg</i>
------------------	---

---

**Description**

Compute y-axis limits for mdaplotg

**Usage**

```
mdaplotg.getYLim(
  ps,
  ylim,
  show.excluded,
  show.legend,
  legend.position,
  show.labels
)
```

**Arguments**

ps	list with plotseries
ylim	limits provided by user
show.excluded	logical, will excluded values also be shown
show.legend	will legend be shown on the plot
legend.position	position of legend on the plot (if shown)
show.labels	logical, will data point labels also be shown

**Value**

vector with two values

---

mdaplotg.prepareData *Prepare data for mdaplotg*

---

**Description**

Prepare data for mdaplotg

**Usage**

```
mdaplotg.prepareData(data, type, groupby)
```

**Arguments**

data	datasets (in form of list, matrix or data frame)
type	vector with type for dataset
groupby	factor or data frame with factors - used to split data matrix into groups

**Value**

list of datasets

The method should prepare data as a list of datasets (matrices or data frames). One list element will be used to create one plot series.

If 'data' is matrix or data frame and no 'groupby' parameter is provided, then every row will be taken as separate set. This option is available only for line or bar plots.

---

mdaplotg.processParam *Check mdaplotg parameters and replicate them if necessary*

---

**Description**

Check mdaplotg parameters and replicate them if necessary

**Usage**

```
mdaplotg.processParam(param, name, is.type, ngroups)
```

**Arguments**

param	A parameter to check
name	name of the parameter (needed for error message)
is.type	function to use for checking parameter type
ngroups	number of groups (plot series)

---

mdaplotg.showLegend    *Show legend for mdaplotg*

---

### Description

Shows a legend for plot elements or their groups.

### Usage

```
mdaplotg.showLegend(  
  legend,  
  col,  
  pt.bg = NA,  
  pch = NULL,  
  lty = NULL,  
  lwd = NULL,  
  cex = 1,  
  bty = "o",  
  position = "topright",  
  plot = TRUE,  
  ...  
)
```

### Arguments

legend	vector with text elements for the legend items
col	vector with color values for the legend items
pt.bg	vector with background colors for the legend items (e.g. for pch = 21:25)
pch	vector with marker symbols for the legend items
lty	vector with line types for the legend items
lwd	vector with line width values for the legend items
cex	vector with cex factor for the points
bty	border type for the legend
position	legend position ("topright", "topleft", "bottomright", "bottomleft", "top", "bottom")
plot	logical, show legend or just calculate and return its size
...	other parameters

---

`mdaplotyy`*Create line plot with double y-axis*

---

**Description**

`mdaplotyy` creates line plot for two plot series and uses separate y-axis for each.

**Usage**

```
mdaplotyy(  
  data,  
  type = "l",  
  col = mdaplot.getColors(2),  
  lty = c(1, 1),  
  lwd = c(1, 1),  
  pch = (if (type == "b") c(16, 16) else c(NA, NA)),  
  cex = 1,  
  xlim = NULL,  
  ylim = NULL,  
  main = attr(data, "name"),  
  xlab = attr(data, "xaxis.name"),  
  ylab = rownames(data),  
  labels = "values",  
  show.labels = FALSE,  
  lab.cex = 0.65,  
  lab.col = "darkgray",  
  show.grid = TRUE,  
  grid.lwd = 0.5,  
  grid.col = "lightgray",  
  xticks = NULL,  
  xticklabels = NULL,  
  xlas = 0,  
  ylas = 0,  
  show.legend = TRUE,  
  legend.position = "topright",  
  legend = ylab,  
  ...  
)
```

**Arguments**

<code>data</code>	a matrix or a data.frame with two rows of values.
<code>type</code>	type of the plot ("l" or "b").
<code>col</code>	a color for markers or lines (same as plot parameter) for each series.
<code>lty</code>	line type for each series (two values)
<code>lwd</code>	line width for each series (two values)

pch	a character for markers (same as plot parameter) for each series (two values).
cex	scale factor for the markers
xlim	limits for the x axis (if NULL, will be calculated automatically).
ylim	limits for the y axis, either list with two vectors (one for each series) or NULL.
main	an overall title for the plot (same as plot parameter).
xlab	a title for the x axis (same as plot parameter).
ylab	a title for each of the two y axis (as a vector of two text values).
labels	a vector with text labels for data points or one of the following: "names", "indices", "values".
show.labels	logical, show or not labels for the data objects.
lab.cex	size for data point labels.
lab.col	color for data point labels.
show.grid	logical, show or not a grid for the plot.
grid.lwd	line thickness (width) for the grid.
grid.col	line color for the grid.
xticks	values for x ticks.
xticklabels	labels for x ticks.
xlas	orientation of xticklabels.
ylas	orientation of yticklabels (will be applied to both y axes).
show.legend	logical show legend with name of each plot series or not
legend.position	position of legend if it must be shown
legend	values for the legend
...	other plotting arguments.

### Details

This plot has properties both `mdaplot` and `mdaplotg`, so when you specify color, line properties etc. you have to do it for both plot series.

### Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

### See Also

[mdaplotg](#) - to make plots for several sets of data objects (groups of objects).

### Examples

```
# See all examples in the tutorial.
```

## Description

This package contains classes and functions for most common methods used in Chemometrics. For a complete list of functions, use `library(help = 'mdatools')`.

## Details

The project is hosted on GitHub (<https://svkucheryavski.github.io/mdatools/>), there you can also find a Bookdown user tutorial explaining most important features of the package. There is also a dedicated YouTube channel (<https://www.youtube.com/channel/UCox0H4utfMq4FIu2kymuyTA>) with introductory Chemometric course with examples based on mdatools functionality.

Every method is represented by two classes: a model class for keeping all parameters and information about the model, and a class for keeping and visualising results of applying the model to particular data values.

Every model class, e.g. `pls`, has all needed functionality implemented as class methods, including model calibration, validation (test set and cross-validation), visualisation of the calibration and validation results with various plots and summary statistics.

So far the following modelling and validation methods are implemented:

<code>pca</code> , <code>pcares</code>	Principal Component Analysis (PCA).
<code>pls</code> , <code>plsres</code>	Partial Least Squares regression (PLS).
<code>simca</code> , <code>simcares</code>	Soft Independent Modelling of Class Analogues (SIMCA)
<code>simcam</code> , <code>simcamres</code>	SIMCA for multiple classes case (SIMCA)
<code>plsda</code> , <code>plsdares</code>	Partial Least Squares Discriminant Analysis (PLS-DA).
<code>randtest</code>	Randomization test for PLS-regression.
<code>ipls</code>	Interval PLS variable.
<code>mcrals</code>	Multivariate Curve Resolution with Alternating Least Squares.
<code>mcrpure</code>	Multivariate Curve Resolution with Purity approach.

Methods for data preprocessing:

<code>prep.autoscale</code>	data mean centering and/or standardization.
<code>prep.savgol</code>	Savitzky-Golay transformation.
<code>prep.snv</code>	Standard normal variate.
<code>prep.msc</code>	Multiplicative scatter correction.
<code>prep.norm</code>	Spectra normalization.
<code>prep.alsbasecorr</code>	Baseline correction with Asymmetric Least Squares.

All plotting methods are based on two functions, `mdaplot` and `mdaplotg`. The functions extend the basic functionality of R plots and allow to make automatic legend and color grouping of data points or lines with colorbar legend, automatically adjust axes limits when several data groups are plotted and so on.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

---

paste1	<i>Paste values together with no separator and collapse into a single string</i>
--------	--

---

**Description**

Paste values together with no separator and collapse into a single string

**Usage**

```
paste1(...)
```

**Arguments**

```
...          values to paste
```

---

pca	<i>Principal Component Analysis</i>
-----	-------------------------------------

---

**Description**

pca is used to build and explore a principal component analysis (PCA) model.

**Usage**

```
pca(  
  x,  
  ncomp = min(nrow(x) - 1, ncol(x), 20),  
  center = TRUE,  
  scale = FALSE,  
  exclrows = NULL,  
  exclcols = NULL,  
  x.test = NULL,  
  method = "svd",  
  rand = NULL,  
  lim.type = "ddmoments",  
  alpha = 0.05,  
  gamma = 0.01,  
  info = "",  
  prep = NULL,  
  do.round = FALSE  
)
```

**Arguments**

x	calibration data (matrix or data frame).
ncomp	maximum number of components to calculate.
center	logical, do mean centering of data or not.
scale	logical, do standardization of data or not.
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)
exclcols	columns to be excluded from calculations (numbers, names or vector with logical values)
x.test	test data (matrix or data frame).
method	method to compute principal components ("svd", "nipals").
rand	vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead)
lim.type	which method to use for calculation of critical limits for residual distances (see details)
alpha	significance level for extreme limits for T2 and Q distances.
gamma	significance level for outlier limits for T2 and Q distances.
info	a short text with model description.
prep	optional list with preprocessing methods created using <code>'prep'</code> function.
do.round	logical, round or not DoF for distances.

**Details**

If number of components is not specified, a minimum of number of objects - 1 and number of variables in calibration set is used. One can also specify an optimal number of components, once model is calibrated (`ncomp.selected`). The optimal number of components is used to build a residual distance plot, as well as for SIMCA classification.

If some of rows of calibration set should be excluded from calculations (e.g. because they are outliers) you can provide row numbers, names, or logical values as parameter `exclrows`. In this case they will be completely ignored when the model is calibrated. However, score and residual distances will be computed for these rows as well and then hidden. You can show them on corresponding plots by using parameter `show.excluded = TRUE`.

It is also possible to exclude selected columns from calculations by providing parameter `exclcols` in form of column numbers, names or logical values. In this case loading matrix will have zeros for these columns. This allows computing PCA models for selected variables without removing them physically from a dataset.

Take into account that if you use other packages to make plots (e.g. `ggplot2`) you will not be able to distinguish between hidden and normal objects.

By default loadings are computed for the original dataset using either SVD or NIPALS algorithm. However, for datasets with large number of rows (e.g. hyperspectral images), there is a possibility to run algorithms based on random permutations [1, 2]. In this case you have to define parameter `rand` as a vector with two values: `p` - oversampling parameter and `q` - number of power iterations.

Usually `rand = c(15, 0)` or `rand = c(5, 1)` are good options, which give quite precise solutions but much faster.

There are several ways to calculate critical limits for orthogonal (Q, q) and score (T2, h) distances. In `mdatools` you can specify one of the following methods via parameter `lim.type`: "jm" Jackson-Mudholkar approach [3], "chisq" - method based on chi-square distribution [4], "ddmoments" and "ddrobust" - related to data driven method proposed in [5]. The "ddmoments" is based on method of moments for estimation of distribution parameters (also known as "classical" approach) while "ddrobust" is based on robust estimation.

If `lim.type="chisq"` or `lim.type="jm"` is used, only limits for Q-distances are computed based on corresponding approach, limits for T2-distances are computed using Hotelling's T-squared distribution. The methods utilizing the data driven approach calculate limits for combination of the distances based on chi-square distribution and parameters estimated from the calibration data.

The critical limits are calculated for a significance level defined by parameter 'alpha'. You can also specify another parameter, 'gamma', which is used to calculate acceptance limit for outliers (shown as dashed line on residual distance plot).

You can also recalculate the limits for existent model by using different values for alpha and gamma, without recomputing the model itself. In this case use the following code (it is assumed that you current PCA/SIMCA model is stored in variable `m`): `m = setDistanceLimits(m, lim.type, alpha, gamma)`.

In case of PCA the critical limits are just shown on residual plot as lines and can be used for detection of extreme objects (solid line) and outliers (dashed line). When PCA model is used for classification in SIMCA (see [simca](#)) or DD-SIMCA ([ddsimca](#)) the limits are also employed for classification of objects.

If you provide a list with preprocessing methods, PCA will apply them to the training set before excluding the columns and rows (if specified). The list will be used to train a preprocessing model which becomes a part of the PCA model object. So when you use method 'predict()' the provided dataset will be automatically preprocessed by the preprocessing model.

Any PCA model (with or without preprocessing) developed in this package can be saved as JSON file using method [writeJSON](#) and then be loaded to interactive web-application for PCA available at <https://mda.tools/pca>. Likewise one can develop a model in the app, save it to JSON file and then load it to R by using method [readJSON](#). In this case, however, the model object will not contain calibration/training results, so some of the plots and statistics will not be available.

## Value

Returns an object of `pca` class with following fields:

<code>ncomp</code>	number of components included to the model.
<code>ncomp.selected</code>	selected (optimal) number of components.
<code>loadings</code>	matrix with loading values (nvar x ncomp).
<code>eigenvals</code>	vector with eigenvalues for all existent components.
<code>expvar</code>	vector with explained variance for each component (in percent).
<code>cumexpvar</code>	vector with cumulative explained variance for each component (in percent).
<code>T2lim</code>	statistical limit for T2 distance.
<code>Qlim</code>	statistical limit for Q residuals.

info	information about the model, provided by user when building the model.
prep	trained preprocessing model (if specified)
calres	an object of class <a href="#">pcares</a> with PCA results for a calibration data.
testres	an object of class <a href="#">pcares</a> with PCA results for a test data, if it was provided.

More details and examples can be found in the Bookdown tutorial.

### Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

### References

1. N. Halko, P.G. Martinsson, J.A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53 (2010) pp. 217-288.
2. S. Kucheryavskiy, Blessing of randomness against the curse of dimensionality, *Journal of Chemometrics*, 32 (2018).
3. J.E. Jackson, *A User's Guide to Principal Components*, John Wiley & Sons, New York, NY (1991).
4. A.L. Pomerantsev, Acceptance areas for multivariate classification derived by projection methods, *Journal of Chemometrics*, 22 (2008) pp. 601-609.
5. A.L. Pomerantsev, O.Ye. Rodionova, Concept and role of extreme objects in PCA/SIMCA, *Journal of Chemometrics*, 28 (2014) pp. 429-438.

### See Also

Methods for `pca` objects:

<code>plot.pca</code>	makes an overview of PCA model with four plots.
<code>summary.pca</code>	shows some statistics for the model.
<code>categorize.pca</code>	categorize data rows as "normal", "extreme" or "outliers".
<a href="#">selectCompNum.pca</a>	set number of optimal components in the model
<a href="#">setDistanceLimits.pca</a>	set critical limits for residuals
<a href="#">predict.pca</a>	applies PCA model to a new data.
<a href="#">writeJSON.pca</a>	saves PCA model to a JSON file so it can be used in web-applications.
<a href="#">pca.readJSON</a>	loads PCA model from a JSON file to <code>pca</code> object.

Plotting methods for `pca` objects:

<a href="#">plotScores.pca</a>	shows scores plot.
<a href="#">plotLoadings.pca</a>	shows loadings plot.
<a href="#">plotVariance.pca</a>	shows explained variance plot.
<a href="#">plotCumVariance.pca</a>	shows cumulative explained variance plot.
<a href="#">plotEigenvalues.pca</a>	shows eigenvalues plot.
<a href="#">plotDistances.pca</a>	shows plot for residual distances (Q vs. T2).
<a href="#">plotBiplot.pca</a>	shows bi-plot.

<code>plotExtremes.pca</code>	shows extreme plot.
<code>plotT2DoF</code>	plot with degrees of freedom for score distance.
<code>plotQDoF</code>	plot with degrees of freedom for orthogonal distance.
<code>plotDistDoF</code>	plot with degrees of freedom for both distances.

Most of the methods for plotting data are also available for PCA results (`pcare`s) objects. Also check `pca.mvreplace`, which replaces missing values in a data matrix with approximated using iterative PCA decomposition.

## Examples

```
library(mdatools)
### Examples for PCA class

## 1. Make PCA model for People data with autoscaling

data(people)
model = pca(people, scale = TRUE, info = "Simple PCA model")
model = selectCompNum(model, 4)
summary(model)
plot(model, show.labels = TRUE)

## 2. Show scores and loadings plots for the model

par(mfrow = c(2, 2))
plotScores(model, comp = c(1, 3), show.labels = TRUE)
plotScores(model, comp = 2, type = "h", show.labels = TRUE)
plotLoadings(model, comp = c(1, 3), show.labels = TRUE)
plotLoadings(model, comp = c(1, 2), type = "h", show.labels = TRUE)
par(mfrow = c(1, 1))

## 3. Show residual distance and variance plots for the model
par(mfrow = c(2, 2))
plotVariance(model, type = "h")
plotCumVariance(model, show.labels = TRUE, legend.position = "bottomright")
plotResiduals(model, show.labels = TRUE)
plotResiduals(model, ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

## 4. Use list with preprocessing methods

# get the data (calibration and test set)
data(simdata)
Xc <- simdata$spectra.c
Xt <- simdata$spectra.t

# create a list with two preprocessing methods
p <- list(
  prep("savgol", width = 7, porder = 2, dorder = 2),
  prep("norm", type = "snv")
)
```

```
# build a PCA model with and without preprocessing
m1 <- pca(Xc, 5, prep = p)
m2 <- pca(Xc, 5)

# apply the models to test set
r1 <- predict(m1, Xt)
r2 <- predict(m2, Xt)

# check scores
par(mfrow = c(1, 2))
plotScores(m1, c(1, 2), res = list(cal = m1$calres, test = r1), main = "With preprocessing")
plotScores(m2, c(1, 2), res = list(cal = m2$calres, test = r2), main = "Without preprocessing")
```

---

pca.cal

*PCA model calibration*

---

### **Description**

Calibrates (builds) a PCA model for given data and parameters

### **Usage**

```
pca.cal(x, ncomp, center, scale, method, rand = NULL)
```

### **Arguments**

x	matrix with data values
ncomp	number of principal components to calculate
center	logical, do mean centering or not
scale	logical, do standardization or not
method	algorithm for computing PC space (only 'svd' and 'nipals' are supported so far)
rand	vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead)

### **Value**

an object with calibrated PCA model

---

pca.fromjson	<i>Converts JSON string created in mda.tools/pca app to pca object</i>
--------------	--

---

**Description**

Converts JSON string created in mda.tools/pca app to pca object

**Usage**

```
pca.fromjson(str)
```

**Arguments**

str	stringified JSON (from model file)
-----	------------------------------------

**Value**

object of `pca` class

---

pca.getB	<i>Low-dimensional approximation of data matrix X</i>
----------	---

---

**Description**

Low-dimensional approximation of data matrix X

**Usage**

```
pca.getB(X, k = NULL, rand = NULL, dist = "unif")
```

**Arguments**

X	data matrix
k	rank of X (number of components)
rand	a vector with two values - oversampling parameter (p) and number of iterations (q)
dist	distribution for generating random numbers, 'unif' or 'norm'

---

pca.mvreplace	<i>Replace missing values in data</i>
---------------	---------------------------------------

---

**Description**

pca.mvreplace is used to replace missing values in a data matrix with approximated by iterative PCA decomposition.

**Usage**

```
pca.mvreplace(  
  x,  
  center = TRUE,  
  scale = FALSE,  
  maxncomp = 10,  
  expvarlim = 0.95,  
  covlim = 10^-6,  
  maxiter = 100  
)
```

**Arguments**

x	a matrix with data, containing missing values.
center	logical, do centering of data values or not.
scale	logical, do standardization of data values or not.
maxncomp	maximum number of components in PCA model.
expvarlim	minimum amount of variance, explained by chosen components (used for selection of optimal number of components in PCA models).
covlim	convergence criterion.
maxiter	maximum number of iterations if convergence criterion is not met.

**Details**

The function uses iterative PCA modeling of the data to approximate and impute missing values. The result is most optimal for data sets with low or moderate level of noise and with number of missing values less than 10% for small dataset and up to 20% for large data.

**Value**

Returns the same matrix x where missing values are replaced with approximated.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

## References

Philip R.C. Nelson, Paul A. Taylor, John F. MacGregor. Missing data methods in PCA and PLS: Score calculations with incomplete observations. *Chemometrics and Intelligent Laboratory Systems*, 35 (1), 1996.

## Examples

```
library(mdatools)

## A very simple example of imputing missing values in a data with no noise

# generate a matrix with values
s = 1:6
odata = cbind(s, 2*s, 4*s)

# make a matrix with missing values
mdata = odata
mdata[5, 2] = mdata[2, 3] = NA

# replace missing values with approximated
rdata = pca.mvreplace(mdata, scale = TRUE)

# show all matrices together
show(cbind(odata, mdata, round(rdata, 2)))
```

---

pca.nipals

*NIPALS based PCA algorithm*

---

## Description

Calculates principal component space using non-linear iterative partial least squares algorithm (NIPALS)

## Usage

```
pca.nipals(x, ncomp = min(ncol(x), nrow(x) - 1), tol = 10^-10)
```

## Arguments

x	a matrix with data values (preprocessed)
ncomp	number of components to calculate
tol	tolerance (if difference in eigenvalues is smaller - convergence achieved)

## Value

a list with scores, loadings and eigenvalues for the components

**References**

Geladi, Paul; Kowalski, Bruce (1986), "Partial Least Squares Regression:A Tutorial", *Analytica Chimica Acta* 185: 1-17

---

pca.readJSON	<i>Reads PCA model from JSON file made in web-application (mda.tools/pca).</i>
--------------	--

---

**Description**

Reads PCA model from JSON file made in web-application (mda.tools/pca).

**Usage**

```
pca.readJSON(fileName)
```

**Arguments**

fileName          file name (or full path) to JSON file.

**Value**

list with PCA model similar to what pca() creates.

---

pca.run	<i>Runs one of the selected PCA methods</i>
---------	---

---

**Description**

Runs one of the selected PCA methods

**Usage**

```
pca.run(x, ncomp, method, rand = NULL)
```

**Arguments**

x	data matrix
ncomp	number of components
method	name of PCA methods ('svd', 'nipals')
rand	parameters for randomized algorithm (if not NULL)

---

pca.svd

*Singular Values Decomposition based PCA algorithm*

---

### Description

Computes principal component space using Singular Values Decomposition

### Usage

```
pca.svd(x, ncomp = min(ncol(x), nrow(x) - 1))
```

### Arguments

x                    a matrix with data values (preprocessed)  
ncomp                number of components to calculate

### Value

a list with scores, loadings and eigenvalues for the components

---

pca.syncResAliases    *Sync calres/testres aliases with the canonical res[["cal"]]/res[["test"]] fields.*

---

### Description

Sync calres/testres aliases with the canonical res[["cal"]]/res[["test"]] fields.

### Usage

```
pca.syncResAliases(obj)
```

### Arguments

obj                    model object

### Value

model object with aliases updated

---

pcares *Results of PCA decomposition*

---

**Description**

pcares is used to store and visualise results for PCA decomposition.

**Usage**

```
pcares(...)
```

**Arguments**

... all arguments supported by `ldecomp`.

**Details**

In fact `pcares` is a wrapper for `ldecomp` - general class for storing results for linear decomposition  $X = TP' + E$ . So, most of the methods, arguments and returned values are inherited from `ldecomp`.

There is no need to create a `pcares` object manually, it is created automatically when build a PCA model (see `pca`) or apply the model to a new data (see `predict.pca`). The object can be used to show summary and plots for the results.

It is assumed that data is a matrix or data frame with I rows and J columns.

**Value**

Returns an object (list) of class `pcares` and `ldecomp` with following fields:

<code>scores</code>	matrix with score values (I x A).
<code>residuals</code>	matrix with data residuals (I x J).
<code>T2</code>	matrix with score distances (I x A).
<code>Q</code>	matrix with orthogonal distances (I x A).
<code>ncomp.selected</code>	selected number of components.
<code>expvar</code>	explained variance for each component.
<code>cumexpvar</code>	cumulative explained variance.

**See Also**

Methods for `pcares` objects:

<code>print.pcares</code>	shows information about the object.
<code>summary.pcares</code>	shows statistics for the PCA results.

Methods, inherited from `ldecomp` class:

<code>plotScores.ldecomp</code>	makes scores plot.
<code>plotVariance.ldecomp</code>	makes explained variance plot.
<code>plotCumVariance.ldecomp</code>	makes cumulative explained variance plot.
<code>plotDistances.ldecomp</code>	makes Q vs. T2 distance plot.

Check also [pca](#) and [ldecomp](#).

## Examples

```
### Examples for PCA results class

library(mdatools)

## 1. Make a model for every odd row of People data
## and apply it to the objects from every even row

data(people)
x = people[seq(1, 32, 2), ]
x.new = people[seq(2, 32, 2), ]

model = pca(people, scale = TRUE, info = "Simple PCA model")
model = selectCompNum(model, 4)

res = predict(model, x.new)
summary(res)
plot(res)

## 2. Make PCA model for People data with autoscaling
## and full cross-validation and get calibration results

data(people)
model = pca(people, scale = TRUE, info = "Simple PCA model")
model = selectCompNum(model, 4)

res = model$calres
summary(res)
plot(res)

## 3. Show scores plots for the results
par(mfrow = c(2, 2))
plotScores(res)
plotScores(res, cgroup = people[, "Beer"], show.labels = TRUE)
plotScores(res, comp = c(1, 3), show.labels = TRUE)
plotScores(res, comp = 2, type = "h", show.labels = TRUE)
par(mfrow = c(1, 1))

## 4. Show residuals and variance plots for the results
par(mfrow = c(2, 2))
plotVariance(res, type = "h")
plotCumVariance(res, show.labels = TRUE)
plotResiduals(res, show.labels = TRUE, cgroup = people[, "Sex"])
```

```
plotResiduals(res, ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))
```

---

pellets

*Image data*

---

### Description

Dataset for showing how mdatools works with images. It is an RGB image represented as 3-way array.

### Usage

```
data(people)
```

### Format

a 3-way array (height x width x channels).

### Details

This is an image with pellets of four different colours mixed in a glas volume.

---

people

*People data*

---

### Description

Dataset for exploratory analysis with 32 objects (male and female persons) and 12 variables.

### Usage

```
data(people)
```

### Format

a matrix with 32 observations (persons) and 12 variables.

[, 1]	Height in cm.
[, 2]	Weight in kg.
[, 3]	Hair length (-1 for short, +1 for long).
[, 4]	Shoe size (EU standard).
[, 5]	Age, years.
[, 6]	Income, euro per year.
[, 7]	Beer consumption, liters per year.

- [ , 8] Wine consumption, liters per year.
- [ , 9] Sex (-1 for male, +1 for female).
- [ , 10] Swimming ability (index, based on 500 m swimming time).
- [ , 11] Region (-1 for Scandinavia, +1 for Mediterranean).
- [ , 12] IQ (European standardized test).

### Details

The data was taken from the book [1] and is in fact a small subset of a pan-European demographic survey. It includes information about 32 persons, 16 represent northern Europe (Scandinavians) and 16 are from the Mediterranean regions. In both groups there are 8 male and 8 female persons. The data includes both quantitative and qualitative variables and is particularly useful for benchmarking exploratory data analysis methods.

### Source

1. K. Esbensen. Multivariate Data Analysis in Practice. Camo, 2002.

---

<code>pinv</code>	<i>Pseudo-inverse matrix</i>
-------------------	------------------------------

---

### Description

Computes pseudo-inverse matrix using SVD

### Usage

```
pinv(data)
```

### Arguments

`data` a matrix with data values to compute inverse for

---

<code>plot.classres</code>	<i>Plot function for classification results</i>
----------------------------	---

---

### Description

Generic plot function for classification results. Alias for [plotPredictions.classres](#).

### Usage

```
## S3 method for class 'classres'
plot(x, ...)
```

**Arguments**

x classification results (object of class plsdares, simcamres, etc.).  
... other arguments for plotPredictions() method.

---

plot.ddsimca *Model overview plot for DD-SIMCA*

---

**Description**

Shows a set of plots for DD-SIMCA model.

**Usage**

```
## S3 method for class 'ddsimca'  
plot(x, comp = c(1, 2), ncomp = x$ncomp.selected, ...)
```

**Arguments**

x a DD-SIMCA model (object of class ddsimca)  
comp which components to show on scores and loadings plot  
ncomp how many components to use for residuals plot  
... other arguments

**Details**

See examples in help for [ddsimca](#) function.

---

plot.ddsimcares *Plot method for DD-SIMCA results.*

---

**Description**

Plot method for DD-SIMCA results.

**Usage**

```
## S3 method for class 'ddsimcares'  
plot(x, ...)
```

**Arguments**

x DD-SIMCA results (object of class ddsimcares)  
... other arguments

---

`plot.ipls`*Overview plot for iPLS results*

---

**Description**

Shows a plot for iPLS results.

**Usage**

```
## S3 method for class 'ipls'  
plot(x, ...)
```

**Arguments**

`x` an iPLS object (object of class `ipls`).  
`...` other arguments.

**Details**

See details for [plotSelection.ipls](#).

---

`plot.mcr`*Plot summary for MCR model*

---

**Description**

Plot summary for MCR model

**Usage**

```
## S3 method for class 'mcr'  
plot(x, ...)
```

**Arguments**

`x` mcr model object  
`...` other parameters

---

`plot.pca`*Model overview plot for PCA*

---

**Description**

Shows a set of plots (scores, loadings, residuals and explained variance) for PCA model.

**Usage**

```
## S3 method for class 'pca'
plot(
  x,
  comp = c(1, 2),
  ncomp = x$ncomp.selected,
  show.labels = FALSE,
  show.legend = TRUE,
  ...
)
```

**Arguments**

<code>x</code>	a PCA model (object of class <code>pca</code> )
<code>comp</code>	vector with two values - number of components to show the scores and loadings plots for
<code>ncomp</code>	number of components to show the residuals plot for
<code>show.labels</code>	logical, show or not labels for the plot objects
<code>show.legend</code>	logical, show or not a legend on the plot
<code>...</code>	other arguments

**Details**

See examples in help for [pca](#) function.

---

`plot.pcares`*Plot method for PCA results object*

---

**Description**

Show several plots to give an overview about the PCA results

**Usage**

```
## S3 method for class 'pcares'
plot(x, comp = c(1, 2), ncomp = x$ncomp.selected, show.labels = TRUE, ...)
```

**Arguments**

x	PCA results (object of class pcares)
comp	which components to show the scores plot for (can be one value or vector with two values).
ncomp	how many components to use for showing the residual distance plot
show.labels	logical, show or not labels for the plot objects
...	other arguments

---

plot.pls

*Model overview plot for PLS*


---

**Description**

Shows a set of plots (x residuals, regression coefficients, RMSE and predictions) for PLS model.

**Usage**

```
## S3 method for class 'pls'
plot(x, ncomp = x$ncomp.selected, ny = 1, show.legend = TRUE, ...)
```

**Arguments**

x	a PLS model (object of class pls)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
ny	which y variable to show the summary for (if NULL, will be shown for all)
show.legend	logical, show or not a legend on the plot
...	other arguments

**Details**

See examples in help for [pls](#) function.

---

plot.plsda	<i>Model overview plot for PLS-DA</i>
------------	---------------------------------------

---

**Description**

Shows a set of plots (x residuals, regression coefficients, misclassification ratio and predictions) for PLS-DA model.

**Usage**

```
## S3 method for class 'plsda'
plot(x, ncomp = x$ncomp.selected, nc = 1, show.legend = TRUE, ...)
```

**Arguments**

x	a PLS-DA model (object of class plsda)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
nc	which class to show the plots
show.legend	logical, show or not a legend on the plot
...	other arguments

**Details**

See examples in help for [plsda](#) function.

---

plot.plsdares	<i>Overview plot for PLS-DA results</i>
---------------	---

---

**Description**

Shows a set of plots (x residuals, y variance, classification performance and predictions) for PLS-DA results.

**Usage**

```
## S3 method for class 'plsdares'
plot(x, nc = 1, ncomp = x$ncomp.selected, show.labels = FALSE, ...)
```

**Arguments**

x	PLS-DA results (object of class plsdares)
nc	which class to show the plot for
ncomp	how many components to use
show.labels	logical, show or not labels for the plot objects
...	other arguments

**Details**

See examples in help for [pls](#) function.

---

plot.plsres                      *Overview plot for PLS results*

---

**Description**

Shows a set of plots for PLS results.

**Usage**

```
## S3 method for class 'plsres'
plot(x, ncomp = x$ncomp.selected, ny = 1, show.labels = FALSE, ...)
```

**Arguments**

x	PLS results (object of class plsres)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
ny	which y variable to show the summary for (if NULL, will be shown for all)
show.labels	logical, show or not labels for the plot objects
...	other arguments

**Details**

See examples in help for [plsres](#) function.

---

plot.randtest                      *Plot for randomization test results*

---

**Description**

Makes a bar plot with alpha values for each component.

**Usage**

```
## S3 method for class 'randtest'
plot(x, main = "Alpha", xlab = "Components", ylab = "", ...)
```

**Arguments**

x	results of randomization test (object of class 'randtest')
main	main title for the plot
xlab	label for x axis
ylab	label for y axis
...	other optional arguments

**Details**

See examples in help for [randtest](#) function.

---

plot.regcoeffs	<i>Regression coefficients plot</i>
----------------	-------------------------------------

---

**Description**

Shows plot with regression coefficient values for every predictor variable (x)

**Usage**

```
## S3 method for class 'regcoeffs'
plot(
  x,
  ncomp = 1,
  ny = 1,
  type = (if (x$nvar > 30) "l" else "h"),
  col = c(mdaplot.getColors(1), "lightgray"),
  show.lines = c(NA, 0),
  show.ci = FALSE,
  alpha = 0.05,
  ylab = paste0("Coefficients (", x$respnames[ny], ")"),
  ...
)
```

**Arguments**

x	regression coefficients object (class regcoeffs)
ncomp	number of components to use for creating the plot
ny	index of response variable to make the plot for
type	type of the plot
col	vector with two colors for the plot (one is used to show real coefficient and another one to show confidence intervals)
show.lines	allows to show horizontal line at c(NA, 0)
show.ci	logical, show or not confidence intervals if they are available

alpha	significance level for confidence intervals (a number between 0 and 1, e.g. for 95% alpha = 0.05)
ylab	label for y-axis
...	other arguments for plotting methods (e.g. main, xlab, etc)

---

plot.regres	<i>Plot method for regression results</i>
-------------	---

---

**Description**

Plot method for regression results

**Usage**

```
## S3 method for class 'regres'
plot(x, ...)
```

**Arguments**

x	regression results (object of class regres)
...	other arguments

**Details**

This is a shortcut for [plotPredictions.regres](#)

---

plot.simca	<i>Model overview plot for SIMCA</i>
------------	--------------------------------------

---

**Description**

Shows a set of plots for SIMCA model.

**Usage**

```
## S3 method for class 'simca'
plot(x, comp = c(1, 2), ncomp = x$ncomp.selected, ...)
```

**Arguments**

x	a SIMCA model (object of class simca)
comp	which components to show on scores and loadings plot
ncomp	how many components to use for residuals plot
...	other arguments

**Details**

See examples in help for [simcam](#) function.

---

plot.simcam	<i>Model overview plot for SIMCAM</i>
-------------	---------------------------------------

---

**Description**

Shows a set of plots for SIMCAM model.

**Usage**

```
## S3 method for class 'simcam'  
plot(x, nc = c(1, 2), ...)
```

**Arguments**

x	a SIMCAM model (object of class simcam)
nc	vector with two values - classes (SIMCA models) to show the plot for
...	other arguments

**Details**

See examples in help for [simcam](#) function.

---

plot.simcamres	<i>Model overview plot for SIMCAM results</i>
----------------	---

---

**Description**

Just shows a prediction plot for SIMCAM results.

**Usage**

```
## S3 method for class 'simcamres'  
plot(x, ...)
```

**Arguments**

x	SIMCAM results (object of class simcamres)
...	other arguments

**Details**

See examples in help for [simcamres](#) function.

---

plotAcceptance	<i>Acceptance plot for DDSIMCA model and results (generic function)</i>
----------------	---

---

**Description**

Acceptance plot for DDSIMCA model and results (generic function)

**Usage**

```
plotAcceptance(obj, ...)
```

**Arguments**

obj	model or result object, e.g. ddsimcares
...	other parameters relevant for the plot

---

plotAcceptance.ddsimca	<i>Acceptance plot for DD-SIMCA model.</i>
------------------------	--

---

**Description**

Shows Acceptance plot for calibration of Procrustes validation results (if available).

**Usage**

```
## S3 method for class 'ddsimca'  
plotAcceptance(obj, res = "cal", ...)
```

**Arguments**

obj	DD-SIMCA model (object of class ddsimca).
res	name of the results (either 'cal' or 'pv')
...	any parameters suitable for <a href="#">plotAcceptance.ddsimcares</a> .

---

plotAcceptance.ddsimcares

*Acceptance plot for DD-SIMCA results object.*


---

### Description

Shows a plot with  $h/h_0$  and  $q/q_0$  distance values for given number of components and type of critical limits.

### Usage

```
## S3 method for class 'ddsimcares'
plotAcceptance(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  show = "auto",
  log = FALSE,
  show.legend = TRUE,
  colors = c(regular = "#2679B2", extreme = "#F2B825", outlier = "#D22C2F", alien =
    "#2679B2", external = "#D22C2F", unknown = "#3c6784", excluded = "#00ff00"),
  lim.lty = c(2, 3),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lab.cex = 0.75,
  lab.col = "#808080",
  ylim = NULL,
  xlim = NULL,
  res.name = NULL,
  show.excluded = FALSE,
  ...
)
```

### Arguments

obj	DD-SIMCA results (object of class ddsimcares)
ncomp	number of components in the model to show the plot for.
limType	limit type to show the plot for ('classic' or 'robust')
show	name of subset to show ("members", "strangers", "all", or "auto").
log	logical, apply log transformation or not.
show.legend	logical, show or not colorbar legend on the plot.
colors	named vector with colors for every role
lim.lty	vector with two values - types of the lines showing critical limits (extreme, outlier).

lim.col	vector with two values - colors of the lines showing critical limits (extreme, outlier).
lim.lwd	vector with two values - thickness of the lines showing critical limits (extreme, outlier).
lab.cex	font size for data point labels.
lab.col	color for data point labels.
ylim	limits for y-axis, if NULL will be estimated automatically.
xlim	limits for x-axis, if NULL will be estimated automatically.
res.name	optional, short name for the result object (shown in plot title).
show.excluded	logical, show or not the excluded rows (objects).
...	other parameters compatible with <code>mdaplot</code> method.

---

plotAliens	<i>Aliens plot for DD-SIMCA results (generic function)</i>
------------	--

---

### Description

Aliens plot for DD-SIMCA results (generic function)

### Usage

```
plotAliens(obj, ...)
```

### Arguments

obj	result object, e.g. ddsimcares
...	other parameters relevant for the plot

---

plotAliens.ddsimcares	<i>Aliens plot for DD-SIMCA results.</i>
-----------------------	--

---

### Description

Aliens plot for DD-SIMCA results.

**Usage**

```
## S3 method for class 'ddsimcares'
plotAliens(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  main = sprintf("Aliens (A = %d)", ncomp),
  xlab = "Number of aliens (expected)",
  ylab = "Number of aliens (observed)",
  col = "#2679b2",
  ellipse.col = "#eeeeee",
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	DD-SIMCA results (object of class ddsimcares)
ncomp	number of components in the model to show the plot for.
limType	limit type to show the plot for ('classic' or 'robust')
main	main title.
xlab	label for x-axis.
ylab	label for y-axis.
col	color of the plot points.
ellipse.col	color of the confidence ellipse elements.
show.plot	logical, if FALSE returns only values and does not show the plot.
...	other parameters compatible with <code>mdaplot</code> method.

---

plotBars	<i>Show plot series as bars</i>
----------	---------------------------------

---

**Description**

First row of the data matrix is taken for creating the bar series. In case of barplot color grouping is made based on columns (not rows as for all other plots).

**Usage**

```
plotBars(ps, col = ps$col, bwd = 0.8, border = NA, force.x.values = NA)
```

**Arguments**

ps	'plotseries' object
col	colors of the bars
bwd	width of the bars (as a ratio for max width)
border	color of bar edges
force.x.values	vector with corrected x-values for a bar plot (needed for group plots, do not change manually).

---

plotBiplot	<i>Biplot</i>
------------	---------------

---

**Description**

Biplot

**Usage**

```
plotBiplot(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for biplot

---

plotBiplot.pca	<i>PCA biplot</i>
----------------	-------------------

---

**Description**

Shows a biplot for selected components.

**Usage**

```
## S3 method for class 'pca'
plotBiplot(
  obj,
  comp = c(1, 2),
  pch = c(16, NA),
  col = mdaplot.getColors(2),
  main = "Biplot",
  lty = 1,
  lwd = 1,
  show.labels = FALSE,
  show.axes = TRUE,
  show.excluded = FALSE,
  lab.col = adjustcolor(col, alpha.f = 0.5),
  ...
)
```

**Arguments**

obj	a PCA model (object of class pca)
comp	a value or vector with several values - number of components to show the plot for
pch	a vector with two values - markers for scores and loadings
col	a vector with two colors for scores and loadings
main	main title for the plot
lty	line type for loadings
lwd	line width for loadings
show.labels	logical, show or not labels for the plot objects
show.axes	logical, show or not axes lines crossing origin (0,0)
show.excluded	logical, show or hide rows marked as excluded (attribute 'exclrows')
lab.col	a vector with two colors for scores and loadings labels
...	other plot parameters (see mdaplotg for details)

---

plotConfidenceEllipse *Add confidence ellipse for groups of points on scatter plot*

---

**Description**

The method shows confidence ellipse for groups of points on a scatter plot made using 'mdaplot()' function with 'cgroup' parameter. It will work only if 'cgroup' is a factor.

**Usage**

```
plotConfidenceEllipse(p, conf.level = 0.95, lwd = 1, lty = 1, opacity = 0)
```

**Arguments**

p	plot data returned by function 'mdaplot()'.
conf.level	confidence level to make the ellipse for (between 0 and 1).
lwd	thickness of line used to show the hull.
lty	type of line used to show the hull.
opacity	if opacity is 0 ellipse is transparent otherwise semi-transparent.

**Examples**

```
# adds 90% confidence ellipse with semi-transparent area over two clusters of points

library(mdatools)
data(people)
group <- factor(people[, "Sex"], labels = c("Male", "Female"))

# first make plot and then add confidence ellipse
p <- mdaplot(people, type = "p", cgroup = group)
plotConfidenceEllipse(p, conf.level = 0.90, opacity = 0.2)
```

---

plotContributions      *Plot resolved contributions*

---

**Description**

Plot resolved contributions

**Usage**

```
plotContributions(obj, ...)
```

**Arguments**

obj	object with mcr case
...	other parameters

---

plotContributions.mcr *Show plot with resolved contributions*

---

### Description

Show plot with resolved contributions

### Usage

```
## S3 method for class 'mcr'
plotContributions(
  obj,
  comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  ...
)
```

### Arguments

obj	object of class mcr
comp	vector with number of components to make the plot for
type	type of the plot
col	vector with colors for individual components
...	other parameters suitable for mdaplotg

---

plotConvexHull *Add convex hull for groups of points on scatter plot*

---

### Description

The method shows convex hull for groups of points on a scatter plot made using 'mdaplot()' function with 'cgroup' parameter. It will work only if 'cgroup' is a factor.

### Usage

```
plotConvexHull(p, lwd = 1, lty = 1, opacity = 0)
```

### Arguments

p	plot data returned by function 'mdaplot()'.
lwd	thickness of line used to show the hull.
lty	type of line used to show the hull.
opacity	if opacity is larger than 0 a semi-transparent polygon is shown over points.

**Examples**

```
# adds convex hull with semi-transparent area over two clusters of points

library(mdatools)
data(people)
group <- factor(people[, "Sex"], labels = c("Male", "Female"))

p <- mdaplot(people, type = "p", cgroup = group)
plotConvexHull(p)
```

---

plotCooman

*Cooman's plot*


---

**Description**

Cooman's plot

**Usage**

```
plotCooman(obj, ...)
```

**Arguments**

obj            classification model or result object  
...            other arguments

**Details**

Generic function for Cooman's plot

---

plotCooman.simcam

*Cooman's plot for SIMCAM model*


---

**Description**

Shows a Cooman's plot for a pair of SIMCA models

**Usage**

```
## S3 method for class 'simcam'  
plotCooman(  
  obj,  
  nc = c(1, 2),  
  res = list(cal = obj$res[["cal"]]),  
  groupby = res[[1]]$c.ref,  
  main = "Cooman's plot",  
  show.limits = TRUE,  
  ...  
)
```

**Arguments**

obj	a SIMCAM model (object of class simcam)
nc	vector with two values - classes (SIMCA models) to show the plot for
res	list with results to show the plot for
groupby	factor to use for grouping points on the plot
main	title of the plot
show.limits	logical, show or not critical limits
...	other plot parameters (see mdaplotg for details)

**Details**

Cooman's plot shows squared orthogonal distance from data points to two selected SIMCA models as well as critical limits for the distance (optional). In case if critical limits must be shown they are computed using chi-square distribution regardless which type of limits is employed for classification.

If only one result object is provided (e.g. results for calibration set or new predictions), then the points can be color grouped using 'groupby' parameter (by default reference class values are used to make the groups). In case of multiple result objects, the points are color grouped according to the objects (e.g. calibration set and test set).

---

plotCooman.simcamres *Cooman's plot for SIMCAM results*

---

**Description**

Shows a Cooman's plot for a pair of SIMCA models

**Usage**

```
## S3 method for class 'simcamres'
plotCooman(
  obj,
  nc = c(1, 2),
  main = "Cooman's plot",
  cgroup = obj$c.ref,
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	SIMCAM results (object of class simcamres)
nc	vector with two values - classes (SIMCA models) to show the plot for
main	main plot title
cgroup	vector of values to use for color grouping of plot points
show.plot	logical, show plot or just return plot data
...	other plot parameters (see mdaplotg for details)

**Details**

The plot is similar to [plotCooman.simcam](#) but shows points only for this result object and does not show critical limits (which are part of a model).

---

plotCorr	<i>Correlation plot</i>
----------	-------------------------

---

**Description**

Correlation plot

**Usage**

```
plotCorr(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for correlation plot

---

plotCorr.randtest      *Correlation plot for randomization test results*

---

**Description**

Makes a plot with statistic values vs. coefficient of determination between permuted and reference y-values.

**Usage**

```
## S3 method for class 'randtest'  
plotCorr(  
  obj,  
  ncomp = obj$ncomp.selected,  
  ylim = NULL,  
  xlab = expression(r^2),  
  ylab = "Test statistic",  
  ...  
)
```

**Arguments**

obj	results of randomization test (object of class 'randtest')
ncomp	number of component to make the plot for
ylim	limits for y axis
xlab	label for x-axis
ylab	label for y-axis
...	other optional arguments

**Details**

See examples in help for [randtest](#) function.

---

plotCumVariance      *Variance plot*

---

**Description**

Variance plot

**Usage**

```
plotCumVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting explained variance for data decomposition

---

```
plotCumVariance.ldecomp
```

*Cumulative explained variance plot*

---

**Description**

Shows a plot with cumulative explained variance vs. number of components.

**Usage**

```
## S3 method for class 'ldecomp'
plotCumVariance(obj, type = "b", labels = "values", show.plot = TRUE, ...)
```

**Arguments**

obj	object of ldecomp class.
type	type of the plot
labels	what to show as labels for plot objects
show.plot	logical, shall plot be created or just plot series object is needed
...	most of graphical parameters from <a href="#">mdaplot</a> function can be used.

---

```
plotCumVariance.mcr
```

*Show plot with cumulative explained variance*

---

**Description**

Show plot with cumulative explained variance

**Usage**

```
## S3 method for class 'mcr'
plotCumVariance(
  obj,
  type = "b",
  labels = "values",
  main = "Cumulative variance",
  xticks = seq_len(obj$ncomp),
  ...
)
```

**Arguments**

obj	object of class mcr
type	type of the plot
labels	what to use as data labels
main	title of the plot
xticks	vector with ticks for x-axis
...	other parameters suitable for mdaplot

---

plotCumVariance.pca     *Cumulative explained variance plot for PCA model*

---

**Description**

Shows a plot with cumulative explained variance for components.

**Usage**

```
## S3 method for class 'pca'
plotCumVariance(obj, legend.position = "bottomright", ...)
```

**Arguments**

obj	a PCA model (object of class pca)
legend.position	position of the legend
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pca](#) function.

---

plotDensity     *Show plot series as density plot (using hex binning)*

---

**Description**

Show plot series as density plot (using hex binning)

**Usage**

```
plotDensity(ps, nbins = 60, colmap = ps$colmap)
```

**Arguments**

ps	'plotseries' object
nbins	number of bins in one dimension
colmap	colormap name or values used to create color gradient

---

```
plotDiscriminationPower
      Discrimination power plot
```

---

**Description**

Discrimination power plot

**Usage**

```
plotDiscriminationPower(obj, ...)
```

**Arguments**

obj	a model object
...	other arguments

**Details**

Generic function for plotting discrimination power values for classification model

---

```
plotDiscriminationPower.simcam
      Discrimination power plot for SIMCAM model
```

---

**Description**

Shows a plot with discrimination power of predictors for a pair of SIMCA models

**Usage**

```
## S3 method for class 'simcam'
plotDiscriminationPower(
  obj,
  nc = c(1, 2),
  type = "h",
  main = paste0("Discrimination power: ", obj$classnames[nc[1]], " vs. ",
    obj$classnames[nc[2]]),
  xlab = attr(obj$dispower, "xaxis.name"),
  ylab = "",
  ...
)
```

**Arguments**

obj	a SIMCAM model (object of class simcam)
nc	vector with two values - classes (SIMCA models) to show the plot for
type	type of the plot
main	main plot title
xlab	label for x axis
ylab	label for y axis
...	other plot parameters (see mdaplotg for details)

**Details**

Discrimination power shows an ability of variables to separate classes. The power is computed similar to model distance, using variance of residuals. However in this case instead of summing the variance across all variables, we take the ratio separately for individual variables.

Discrimination power equal to or above 3 is considered as high.

---

plotDistances	<i>Distance plot for model and results (generic function)</i>
---------------	---

---

**Description**

Distance plot for model and results (generic function)

**Usage**

```
plotDistances(obj, ...)
```

**Arguments**

obj	model or result object, e.g. ddsimcares
...	other parameters relevant for the plot

---

`plotDistances.ddsimca` *Show with distance values (score, orthogonal or full) vs object indices for calibration and PV-set results.*

---

### Description

Show with distance values (score, orthogonal or full) vs object indices for calibration and PV-set results.

### Usage

```
## S3 method for class 'ddsimca'
plotDistances(
  obj,
  res = "both",
  ncomp = obj$ncomp.selected,
  limType = "classic",
  distance = "h",
  log = FALSE,
  lim.lty = c(2, 3),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  ylim = NULL,
  xlim = NULL,
  ...
)
```

### Arguments

<code>obj</code>	DD-SIMCA model (object of class <code>ddsimca</code> )
<code>res</code>	name of the results (can be 'cal', 'pv', 'both', or 'diff', in the latter case shows difference of the distance values between the training and the PV-set).
<code>ncomp</code>	number of components to show the plot for.
<code>limType</code>	limit type to show the plot for ('classic' or 'robust')
<code>distance</code>	which distance to show ("h" for score, "q" for orthogonal or "f" for full distance).
<code>log</code>	logical, apply log transformation or not.
<code>lim.lty</code>	vector with two values - types of the lines showing critical limits (extreme, outlier).
<code>lim.col</code>	vector with two values - colors of the lines showing critical limits (extreme, outlier).
<code>lim.lwd</code>	vector with two values - thickness of the lines showing critical limits (extreme, outlier).
<code>ylim</code>	limits for y-axis, if NULL will be estimated automatically.

xlim            limits for x-axis, if NULL will be estimated automatically.  
 ...            other parameters compatible with `plotDistances.ddsimcares` method.

`plotDistances.ddsimcares`

*Show with distance values (score, orthogonal or full) vs object indices for DD-SIMCA results.*

## Description

Show with distance values (score, orthogonal or full) vs object indices for DD-SIMCA results.

## Usage

```
## S3 method for class 'ddsimcares'
plotDistances(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  distance = "h",
  log = FALSE,
  show.legend = TRUE,
  lim.lty = c(2, 3),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  ylim = NULL,
  xlim = NULL,
  show.plot = TRUE,
  show.excluded = FALSE,
  ...
)
```

## Arguments

`obj`            DD-SIMCA results (object of class `ddsimcares`)  
`ncomp`          number of components to show the plot for.  
`limType`        limit type to show the plot for ('classic' or 'robust')  
`distance`       which distance to show ("h" for score, "q" for orthogonal or "f" for full distance).  
`log`            logical, apply log transformation or not.  
`show.legend`   logical, show or not colorbar legend on the plot.  
`lim.lty`        vector with two values - types of the lines showing critical limits (extreme, outlier).  
`lim.col`        vector with two values - colors of the lines showing critical limits (extreme, outlier).

<code>lim.lwd</code>	vector with two values - thickness of the lines showing critical limits (extreme, outlier).
<code>ylim</code>	limits for y-axis, if NULL will be estimated automatically.
<code>xlim</code>	limits for x-axis, if NULL will be estimated automatically.
<code>show.plot</code>	logical, if FALSE returns only values and does not show the plot.
<code>show.excluded</code>	logical, show or not the excluded rows (objects).
<code>...</code>	other parameters compatible with <code>mdaplot</code> method.

---

`plotDistances.ldecomp` *Distance plot*

---

### Description

Shows a plot with orthogonal (Q, q) vs. score (T2, h) distances for data objects.

### Usage

```
## S3 method for class 'ldecomp'
plotDistances(
  obj,
  ncomp = obj$ncomp.selected,
  norm = FALSE,
  log = FALSE,
  show.plot = TRUE,
  ...
)
```

### Arguments

<code>obj</code>	object of <code>ldecomp</code> class.
<code>ncomp</code>	number of components to show the plot for (if NULL, selected by model value will be used).
<code>norm</code>	logical, normalize distance values or not (see details)
<code>log</code>	logical, apply log transformation to the distances or not (see details)
<code>show.plot</code>	logical, shall plot be created or just plot series object is needed
<code>...</code>	most of graphical parameters from <code>mdaplot</code> function can be used.

---

plotDistances.pca      *Distance plot for PCA model*

---

### Description

Shows a plot with score (T2, h) vs orthogonal (Q, q) distances and corresponding critical limits for given number of components.

### Usage

```
## S3 method for class 'pca'
plotDistances(
  obj,
  ncomp = obj$ncomp.selected,
  log = FALSE,
  norm = TRUE,
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = TRUE,
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  res = obj$res,
  show.legend = TRUE,
  ...
)
```

### Arguments

obj	a PCA model (object of class pca)
ncomp	how many components to use (by default optimal value selected for the model will be used)
log	logical, apply log transformation to the distances or not (see details)
norm	logical, normalize distance values or not (see details)
cgroup	color grouping of plot points (works only if one result object is available)
xlim	limits for x-axis
ylim	limits for y-axis
show.limits	logical, show or not lines/curves with critical limits for the distances
lim.col	vector with two values - line color for extreme and outlier limits
lim.lwd	vector with two values - line width for extreme and outlier limits
lim.lty	vector with two values - line type for extreme and outlier limits
res	list with result objects to show the plot for (by default, model results are used)

show.legend      logical, show or not a legend on the plot (needed if several result objects are available)

...                other plot parameters (see mdaplotg for details)

### Details

The function is a bit more advanced version of [plotResiduals.ldecomp](#). It allows to show distance values for several result objects (e.g. calibration and test set or calibration and new prediction set) as well as display the corresponding critical limits in form of lines or curves.

Depending on how many result objects your model has or how many you specified manually, using the `res` parameter, the plot behaves in a bit different way.

If only one result object is provided, then it allows to colorise the points using `cgroup` parameter. If you specify `cgroup = "categories"` then it will show points as three groups: normal, extreme and outliers. If two or more result objects are provided, then the function shows distances in groups, and adds corresponding legend.

The function can show distance values normalised ( $h/h_0$  and  $q/q_0$ ) as well as with log transformation ( $\log(1 + h/h_0)$ ,  $\log(1 + q/q_0)$ ). The latter is useful if distribution of the points is skewed and most of them are densely located around bottom left corner.

See examples in help for [pca](#) function.

---

plotDistDoF

*Degrees of freedom plot for both distances*

---

### Description

Shows a plot with degrees of freedom computed for score and orthogonal distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

### Usage

```
plotDistDoF(
  obj,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  ...
)
```

### Arguments

`obj`                a PCA model (object of class `pca`)

`type`                type of the plot ("b", "l", "h")

`labels`              what to show as data points labels

`xticks`              vector with tick values for x-axis

...                    other plot parameters (see `mdaplotg` for details)

**Details**

Works only if parameter `lim.type` equal to "ddmoments" or "ddrobust".

---

plotEigenvalues	<i>Eigenvalues plot</i>
-----------------	-------------------------

---

**Description**

Eigenvalues plot

**Usage**

```
plotEigenvalues(obj, ...)
```

**Arguments**

obj	a model object
...	other arguments

**Details**

Generic function for plotting eigenvalues vs. number of components

---

plotEigenvalues.pca	<i>Eigenvalues plot for PCA model</i>
---------------------	---------------------------------------

---

**Description**

Shows a plot with eigenvalues vs. number of components, optionally with log or sqrt transformation applied.

**Usage**

```
## S3 method for class 'pca'  
plotEigenvalues(  
  obj,  
  type = "b",  
  labels = "values",  
  transform = "none",  
  xticks = seq_len(obj$ncomp),  
  ylab = NULL,  
  ...  
)
```

**Arguments**

obj	a PCA model (object of class pca)
type	type of the plot ("l", "b", "h")
labels	what to show as labels for plot objects.
transform	transformation to apply to eigenvalues: "none", "log", or "sqrt"
xticks	vector with ticks for x-axis
ylab	label for y-axis
...	other plot parameters (see mdaPlot for details)

**Details**

The eigenvalues represent the variance captured by each principal component. Transformations can be useful for identifying the number of significant components: "log" applies  $\log(\text{eigenvalues})$  and "sqrt" applies  $\sqrt{\text{eigenvalues}}$ .

---

plotErrorbars	<i>Show plot series as error bars</i>
---------------	---------------------------------------

---

**Description**

It is assumed that first row of dataset contains the y-coordinates of points, second row contains size of lower error bar and third - size for upper error bar. If only two rows are provided it is assumed that error bars are symmetric.

**Usage**

```
plotErrorbars(ps, col = ps$col, pch = 16, lwd = 1, cex = 1, ...)
```

**Arguments**

ps	'plotseries' object
col	color for the error bars
pch	marker symbol for the plot
lwd	line width for the error bars
cex	scale factor for the marker
...	other arguments for function 'points()'.

---

plotExtreme	<i>Shows extreme plot for PCA and DD-SIMCA models</i>
-------------	---

---

**Description**

Generic function for creating extreme plot for PCA and DD-SIMCA models

**Usage**

```
plotExtreme(obj, ...)
```

**Arguments**

obj	a PCA or DD-SIMCA model
...	other parameters

---

plotExtreme.ddsimca	<i>A shortcut to <a href="#">plotExtremes.ddsimca</a>.</i>
---------------------	--

---

**Description**

A shortcut to [plotExtremes.ddsimca](#).

**Usage**

```
## S3 method for class 'ddsimca'  
plotExtreme(obj, ...)
```

**Arguments**

obj	DD-SIMCA model (object of class ddsimca).
...	any parameters suitable for <a href="#">plotExtremes.ddsimca</a> .

plotExtreme.ddsimcares

*Extremes plot (shortcut to [plotExtremes.ddsimcares](#)).*

---

### Description

Extremes plot (shortcut to [plotExtremes.ddsimcares](#)).

### Usage

```
## S3 method for class 'ddsimcares'  
plotExtreme(obj, ...)
```

### Arguments

obj	DD-SIMCA results (object of class ddsimcares)
...	any parameter acceptable by <a href="#">plotExtremes.ddsimcares</a> .

---

plotExtreme.pca

*A shortcut to [plotExtremes.pca](#).*

---

### Description

A shortcut to [plotExtremes.pca](#).

### Usage

```
## S3 method for class 'pca'  
plotExtreme(obj, ...)
```

### Arguments

obj	PCA model (object of class pca).
...	any parameters suitable for <a href="#">plotExtremes.pca</a> .

---

plotExtremes	<i>Shows extreme plot for PCA and DD-SIMCA models</i>
--------------	---

---

**Description**

Generic function for creating extreme plot for PCA and DD-SIMCA models

**Usage**

```
plotExtremes(obj, ...)
```

**Arguments**

obj	a PCA or DD-SIMCA model
...	other parameters

---

plotExtremes.ddsimca	<i>Extreme plot</i>
----------------------	---------------------

---

**Description**

Shows a plot with number of expected vs. number of observed extreme objects for different significance levels (alpha values) for calibration and PV-set results.

**Usage**

```
## S3 method for class 'ddsimca'
plotExtremes(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  col = mdaplot.getColors(2),
  pch = c(16, 16),
  legend.position = "topleft",
  ...
)
```

**Arguments**

obj	a DD-SIMCA model (object of class ddsimca)
ncomp	number of components to show the plot for
limType	limit type to show the plot for ('classic' or 'robust').
col	vector with two colors (for calibration and PV-set sensitivity).
pch	vector with two markers (for calibration and PV-set sensitivity).

```

legend.position
                position of the legend
...
                other arguments, suitable for plotExtremes.ddsimcares

```

```
plotExtremes.ddsimcares
```

*Extremes plot.*

### Description

The method generates a sequence of values for significance level (alpha) and then computes how many extremes the model produces by applying it to the target class members. After that the computed number is plotted against the expected number.

The plot also shows a confidence ellipse for the expected values.

### Usage

```

## S3 method for class 'ddsimcares'
plotExtremes(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  main = sprintf("Extremes (A = %d)", ncomp),
  xlab = "Number of extremes (expected)",
  ylab = "Number of extremes (observed)",
  col = "#2679b2",
  pch = 16,
  ellipse.col = "#eeeeee",
  show.plot = TRUE,
  ...
)

```

### Arguments

obj	DD-SIMCA results (object of class ddsimcares)
ncomp	number of components in the model to show the plot for.
limType	limit type to show the plot for ('classic' or 'robust').
main	main title.
xlab	label for x-axis.
ylab	label for y-axis.
col	color of the plot points.
pch	color of the plot points.
ellipse.col	color of the confidence ellipse elements.
show.plot	logical, if FALSE returns only values and does not show the plot.
...	other parameters compatible with <code>mdaplot</code> method.

---

plotExtremes.pca      *Extreme plot*

---

### Description

Shows a plot with number of expected vs. number of observed extreme objects for different significance levels (alpha values)

### Usage

```
## S3 method for class 'pca'
plotExtremes(
  obj,
  res = obj$res[["cal"]],
  comp = obj$ncomp.selected,
  main = "Extreme plot",
  xlab = "Expected",
  ylab = "Observed",
  pch = rep(21, length(comp)),
  bg = mdaplot.getColors(length(comp)),
  col = rep("white", length(comp)),
  lwd = ifelse(pch %in% 21:25, 0.25, 1),
  cex = rep(1.2, length(comp)),
  ellipse.col = "#cceedf",
  legend.position = "bottomright",
  ...
)
```

### Arguments

obj	a PCA model (object of class pca)
res	object with PCA results to show the plot for (e.g. calibration, test, etc)
comp	vector, number of components to show the plot for
main	plot title
xlab	label for x-axis
ylab	label for y-axis
pch	vector with values for pch parameter for each number of components
bg	vector with background color values for series of points (if pch=21:25)
col	vector with color values for series of points
lwd	line width for point symbols
cex	scale factor for data points
ellipse.col	color for tolerance ellipse
legend.position	position of the legend
...	other arguments

---

plotFoM	<i>Show plot with figure of merit vs. number of components (generic function).</i>
---------	--

---

**Description**

Show plot with figure of merit vs. number of components (generic function).

**Usage**

```
plotFoM(obj, ...)
```

**Arguments**

obj	result object, e.g. ddsimcares
...	other parameters relevant for the method

---

plotFoM.ddsimcares	<i>Figure of merit plot.</i>
--------------------	------------------------------

---

**Description**

Figure of merit plot.

**Usage**

```
## S3 method for class 'ddsimcares'
plotFoM(
  obj,
  fom = "sens",
  limType = "classic",
  type = "b",
  ylim = c(0, 1.1),
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	DD-SIMCA results (object of class ddsimcares)
fom	name of the figure of merit to show the plot for ('sens', 'spec', 'sel', 'eff', 'acc').
limType	limit type to show the plot for ('classic' or 'robust').
type	type of plot (can be 'b', 'l', or 'h').
ylim	limits for y-axis.

show.plot      logical, if FALSE returns only values and does not show the plot.  
 ...            any parameters for `mdaplot` method can be provided.  
 Shows the selected figure of merit values vs. number of components. Pay attention, that some of the FoMs may not be available, for example if result object is made only for members of the target class, only sensitivity ('sens') is available.

---

plotFoMs                      *Show plot with several figures of merit vs. number of components (generic function).*

---

### Description

Show plot with several figures of merit vs. number of components (generic function).

### Usage

```
plotFoMs(obj, ...)
```

### Arguments

obj                      result object, e.g. `ddsimcares`  
 ...                      other parameters relevant for the method

---

plotFoMs.ddsimcares      *Figures of merit plot (multiple FoMs).*

---

### Description

Figures of merit plot (multiple FoMs).

### Usage

```
## S3 method for class 'ddsimcares'
plotFoMs(
  obj,
  foms = NULL,
  limType = "classic",
  type = "b",
  ylim = c(0, 1.1),
  col = NULL,
  legend.position = "bottom",
  main = "Figures of merit",
  ...
)
```

**Arguments**

obj	DD-SIMCA results (object of class ddsimcares)
foms	vector with FoM names to show on the plot (any combination of 'sens', 'spec', 'sel', 'eff', 'acc').
limType	limit type to show the plot for ('classic' or 'robust').
type	type of plot (can be 'b', 'l', or 'h').
ylim	limits for y-axis.
col	vector with color values (should contain one value for each fom), if not provided will use default colors.
legend.position	position of legend on the plot.
main	main title.
...	any parameters for <a href="#">mdaplotg</a> method can be provided.

Shows several figures of merit values vs. number of components as a group plot (see details in [mdaplotg](#) method description).

---

plotHist

*Statistic histogram*


---

**Description**

Statistic histogram

**Usage**

```
plotHist(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting statistic histogram plot

---

plotHist.randtest      *Histogram plot for randomization test results*

---

### Description

Makes a histogram for statistic values distribution for particular component, also shows critical value as a vertical line.

### Usage

```
## S3 method for class 'randtest'
plotHist(obj, ncomp = obj$ncomp.selected, bwd = 0.9, ...)
```

### Arguments

obj	results of randomization test (object of class 'randtest')
ncomp	number of component to make the plot for
bwd	width of bars (between 0 and 1)
...	other optional arguments

### Details

See examples in help for [randtest](#) function.

---

plotHotellingEllipse      *Hotelling ellipse*

---

### Description

Add Hotelling ellipse to a scatter plot

### Usage

```
plotHotellingEllipse(p, conf.lim = 0.95, col = "#a0a0a0", lty = 3, ...)
```

### Arguments

p	plot series (e.g. from PCA scores plot)
conf.lim	confidence limit
col	color of the ellipse line
lty	line type (e.g. 1 for solid, 2 for dashed, etc.)
...	any argument suitable for lines function

**Details**

The method is created to be used with PCA and PLS scores plots, so it shows the statistical limits computed using Hotelling  $T^2$  distribution in form of ellipse. The function works similar to `plotConvexHull` and `plotConfidenceEllipse` but does not require grouping of data points. Can be used together with functions `plotScores.pca`, `plotScores.ldecomp`, `plotXScores.pls`, `plotXScores.plsres`.

See examples for more details.

**Examples**

```
# create PCA model for People data
data(people)
m <- pca(people, 4, scale = TRUE)

# make scores plot and show Hotelling ellipse with default settings
p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p)

# make scores plot and show Hotelling ellipse with manual settings
p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p, conf.lim = 0.99, col = "red")

# in case if you have both calibration and test set, 'plotScores()' returns
# plot series data for both, so you have to subset it and take the first series
# (calibration set) as shown below.
ind <- seq(1, 32, by = 4)
xc <- people[-ind, , drop = FALSE]
xt <- people[ind, , drop = FALSE]
m <- pca(xc, 4, scale = TRUE, x.test = xt)

p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p[[1]])
```

---

plotLines

*Show plot series as set of lines*

---

**Description**

Show plot series as set of lines

**Usage**

```
plotLines(
  ps,
  col = ps$col,
  lty = 1,
  lwd = 1,
```

```

    cex = 1,
    col.excluded = "darkgray",
    show.excluded = FALSE,
    ...
)

```

### Arguments

ps	'plotseries' object
col	a color for markers or lines (same as plot parameter).
lty	line type
lwd	line width
cex	scale factor for the marker
col.excluded	color for the excluded lines.
show.excluded	logical, show or not the excluded data points
...	other arguments for function 'lines()'.

---

 plotLoadings

*Loadings plot*


---

### Description

Loadings plot

### Usage

```
plotLoadings(obj, ...)
```

### Arguments

obj	a model or result object
...	other arguments

### Details

Generic function for plotting loadings values for data decomposition

---

plotLoadings.pca      *Loadings plot for PCA model*

---

### Description

Shows a loadings plot for selected components.

### Usage

```
## S3 method for class 'pca'
plotLoadings(
  obj,
  comp = if (obj$ncomp > 1) c(1, 2) else 1,
  type = (if (length(comp) == 2) "p" else "l"),
  show.legend = TRUE,
  show.axes = TRUE,
  ...
)
```

### Arguments

obj	a PCA model (object of class pca)
comp	a value or vector with several values - number of components to show the plot for
type	type of the plot ('b', 'l', 'h')
show.legend	logical, show or not a legend on the plot
show.axes	logical, show or not axes lines crossing origin (0,0)
...	other plot parameters (see mdaplotg for details)

### Details

See examples in help for [pca](#) function.

---

plotMisclassified      *Misclassification ratio plot*

---

### Description

Misclassification ratio plot

### Usage

```
plotMisclassified(obj, ...)
```

**Arguments**

obj            a model or a result object  
 ...            other arguments

**Details**

Generic function for plotting misclassification values for classification model or results

---

plotMisclassified.classmodel  
*Misclassified ratio plot for classification model*

---

**Description**

Makes a plot with misclassified ratio values vs. model complexity (e.g. number of components)

**Usage**

```
## S3 method for class 'classmodel'
plotMisclassified(obj, ...)
```

**Arguments**

obj            classification model (object of class `plsda`, `simca`, etc.).  
 ...            parameters for `plotPerformance.classmodel` function.

**Details**

See examples in description of `plsda`, `simca` or `simcam`.

---

plotMisclassified.classres  
*Misclassified ratio plot for classification results*

---

**Description**

Makes a plot with ms ratio values vs. model complexity (e.g. number of components) for classification results.

**Usage**

```
## S3 method for class 'classres'
plotMisclassified(obj, ...)
```

**Arguments**

obj                    classification results (object of class `plsdares`, `simcamres`, etc.).  
...                    other parameters for `plotPerformance.classes`

**Details**

See examples in description of `plsdares`, `simcamres`, etc.

---

plotModelDistance      *Model distance plot*

---

**Description**

Model distance plot

**Usage**

```
plotModelDistance(obj, ...)
```

**Arguments**

obj                    a model object  
...                    other arguments

**Details**

Generic function for plotting distance from object to a multivariate model

---

plotModelDistance.simcam  
*Model distance plot for SIMCAM model*

---

**Description**

Shows a plot with distance between one SIMCA model and others.

**Usage**

```
## S3 method for class 'simcam'
plotModelDistance(
  obj,
  nc = 1,
  type = "h",
  xticks = seq_len(obj$nclasses),
  xticklabels = obj$classnames,
  main = paste0("Model distance (", obj$classnames[nc], ")"),
  xlab = "Models",
  ylab = "",
  ...
)
```

**Arguments**

obj	a SIMCAM model (object of class simcam)
nc	one value - number of class (SIMCA model) to show the plot for
type	type of the plot ("h", "l" or "b")
xticks	vector with tick values for x-axis
xticklabels	vector with tick labels for x-axis
main	main plot title
xlab	label for x axis
ylab	label for y axis
...	other plot parameters (see mdaPlotg for details)

**Details**

The plot shows similarity between a selected model and the others as a ratio of residual variance using the following algorithm. Let's take two SIMCA/PCA models, m1 and m2, which have optimal number of components A1 and A2. The models have been calibrated using calibration sets X1 and X2 with number of rows n1 and n2. Then we do the following:

1. Project X2 to model m1 and compute residuals, E12
2. Compute variance of the residuals as  $s12 = \text{sum}(E12^2) / n1$
3. Project X1 to model m2 and compute residuals, E21
4. Compute variance of the residuals as  $s21 = \text{sum}(E21^2) / n2$
5. Compute variance of residuals for m1 as  $s1 = \text{sum}(E1^2) / (n1 - A1 - 1)$
6. Compute variance of residuals for m2 as  $s2 = \text{sum}(E2^2) / (n2 - A2 - 1)$

The model distance then can be computed as:  $d = \text{sqrt}((s12 + s21) / (s1 + s2))$

As one can see, if the two models and corresponding calibration sets are identical, then the distance will be  $\text{sqrt}((n - A - 1) / n)$ . For example, if  $n = 25$  and  $A = 2$ , then the distance between the model and itself is  $\text{sqrt}(22/25) = \text{sqrt}(0.88) = 0.938$ . This case is demonstrated in the example section.

In general, if distance between models is below one classes are overlapping. If it is above 3 the classes are well separated.

**Examples**

```
# create two calibration sets with n = 25 objects in each
data(iris)
x1 <- iris[1:25, 1:4]
x2 <- iris[51:75, 1:4]

# create two SIMCA models with A = 2
m1 <- simca(x1, 'setosa', ncomp = 2)
m2 <- simca(x2, 'versicolor', ncomp = 2)

# combine the models into SIMCAM class
m <- simcam(list(m1, m2))

# show the model distance plot with distance values as labels
# note, that distance between setosa and setosa is 0.938
plotModelDistance(m, show.labels = TRUE, labels = "values")
```

---

plotModellingPower      *Modelling power plot*

---

**Description**

Modelling power plot

**Usage**

```
plotModellingPower(obj, ...)
```

**Arguments**

obj	a model object
...	other arguments

**Details**

Generic function for plotting modelling power values for classification model

---

plotPerformance      *Classification performance plot*

---

**Description**

Classification performance plot

**Usage**

```
plotPerformance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting classification performance for model or results

---

plotPerformance.classmodel  
*Performance plot for classification model*

---

**Description**

Makes a plot with sensitivity values vs. model complexity (e.g. number of components)

**Usage**

```
## S3 method for class 'classmodel'  
plotPerformance(  
  obj,  
  nc = 1,  
  param = "misclassified",  
  type = "b",  
  labels = "values",  
  ylab = "",  
  ylim = c(0, 1.15),  
  xticks = seq_len(dim(obj$res$cal$c.pred)[2]),  
  res = obj$res,  
  ...  
)
```

**Arguments**

obj	classification model (object of class plsda, simca, etc.).
nc	class number to make the plot for.
param	which parameter to make the plot for ("specificity", "sensitivity", or "misclassified")
type	type of the plot
labels	what to show as labels for plot objects.
ylab	label for y axis
ylim	vector with two values - limits for y axis
xticks	vector with tick values for x-axis
res	list with result objects to show the plot for
...	most of the graphical parameters from <code>mdaplotg</code> function can be used.

---

plotPerformance.classres

*Performance plot for classification results*

---

**Description**

Makes a plot with classification performance parameters vs. model complexity (e.g. number of components) for classification results.

**Usage**

```
## S3 method for class 'classres'
plotPerformance(
  obj,
  nc = 1,
  type = "b",
  param = c("sensitivity", "specificity", "misclassified"),
  labels = "values",
  ylab = "",
  ylim = c(0, 1.1),
  xticks = seq_len(obj$ncomp),
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	classification results (object of class plsdares, simcamres, etc.).
nc	if there are several classes, which class to make the plot for.
type	type of the plot

param	which performance parameter to make the plot for (can be a vector with several values).
labels	what to show as labels for plot objects.
ylab	label for y axis
ylim	vector with two values - limits for y axis
xticks	vector with x-axis tick values
show.plot	logical, shall plot be created or just plot series object is needed
...	most of the graphical parameters from <a href="#">mdaplot</a> function can be used.

### Details

See examples in description of [plsdares](#), [simcamres](#), etc.

---

plotPointsShape	<i>Add confidence ellipse or convex hull for group of points</i>
-----------------	--

---

### Description

Add confidence ellipse or convex hull for group of points

### Usage

```
plotPointsShape(p, lwd, lty, opacity, shape_function, ...)
```

### Arguments

p	plot data returned by function 'mdaplot()'
lwd	thickness of line used to show the hull
lty	type of line used to show the hull
opacity	if opacity is larger than 0 a semi-transparent polygon is shown over points
shape_function	function which calculates and return coordinates of the shape
...	extra parameters for shape_function

plotPredictions      *Predictions plot*

---

**Description**

Predictions plot

**Usage**

```
plotPredictions(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting predicted values for classification or regression model or results

---

plotPredictions.classmodel  
*Predictions plot for classification model*

---

**Description**

Makes a plot with class predictions for a classification model.

**Usage**

```
## S3 method for class 'classmodel'  
plotPredictions(  
  obj,  
  res.name = NULL,  
  nc = seq_len(obj$nclasses),  
  ncomp = NULL,  
  main = NULL,  
  ...  
)
```

**Arguments**

obj	a classification model (object of class <code>simca</code> , <code>plsda</code> , etc.). if NULL value is specified, the result will be selected automatically by checking the nearest available from test, cv and calibration results.
res.name	name of result object to make the plot for ("test", "cv" or "cal").
nc	vector with class numbers to make the plot for.
ncomp	what number of components to make the plot for.
main	title of the plot (if NULL will be set automatically)
...	most of the graphical parameters from <code>mdaplotg</code> function can be used.

**Details**

See examples in description of [plsda](#), [simca](#) or [simcam](#).

---

plotPredictions.classres

*Prediction plot for classification results*

---

**Description**

Makes a plot with predicted class values for classification results.

**Usage**

```
## S3 method for class 'classres'
plotPredictions(
  obj,
  nc = seq_len(obj$nclasses),
  ncomp = obj$ncomp.selected,
  ylab = "",
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	classification results (object of class <code>plsdares</code> , <code>simcamres</code> , etc.).
nc	vector with classes to show predictions for.
ncomp	model complexity (number of components) to make the plot for.
ylab	label for y axis
show.plot	logical, shall plot be created or just plot series object is needed
...	most of the graphical parameters from <code>mdaplotg</code> or <code>mdaplot</code> function can be used.

**Details**

See examples in description of [plsdares](#), [simcamres](#), etc.

---

```
plotPredictions.regmodel
```

*Predictions plot for regression model*

---

**Description**

Shows plot with predicted vs. reference (measured) y values for selected components.

**Usage**

```
## S3 method for class 'regmodel'
plotPredictions(
  obj,
  ncomp = obj$ncomp.selected,
  ny = 1,
  legend.position = "topleft",
  show.line = TRUE,
  res = obj$res,
  ...
)
```

**Arguments**

<code>obj</code>	a regression model (object of class <code>regmodel</code> )
<code>ncomp</code>	how many components to use (if <code>NULL</code> - user selected optimal value will be used)
<code>ny</code>	number of response variable to make the plot for (if y is multivariate)
<code>legend.position</code>	position of legend on the plot (if shown)
<code>show.line</code>	logical, show or not line fit for the plot points
<code>res</code>	list with result objects
<code>...</code>	other plot parameters (see <code>mdaplotg</code> for details)

---

plotPredictions.regres

*Predictions plot for regression results*


---

### Description

Shows plot with predicted y values.

### Usage

```
## S3 method for class 'regres'
plotPredictions(
  obj,
  ny = 1,
  ncomp = obj$ncomp.selected,
  show.line = TRUE,
  show.stat = FALSE,
  stat.col = "#606060",
  stat.cex = 0.85,
  xlim = NULL,
  ylim = NULL,
  axes.equal = TRUE,
  show.plot = TRUE,
  ...
)
```

### Arguments

obj	regression results (object of class regres)
ny	number of predictor to show the plot for (if y is multivariate)
ncomp	complexity of model (e.g. number of components) to show the plot for
show.line	logical, show or not line fit for the plot points
show.stat	logical, show or not legend with statistics on the plot
stat.col	color of text in legend with statistics
stat.cex	size of text in legend with statistics
xlim	limits for x-axis (if NULL will be computed automatically)
ylim	limits for y-axis (if NULL will be computed automatically)
axes.equal	logical, make limits for x and y axes equal or not
show.plot	logical, show plot or just return plot data
...	other plot parameters (see mdaplot for details)

### Details

If reference values are available, the function shows a scatter plot with predicted vs. reference values, otherwise predicted values are shown vs. object numbers.

plotPredictions.simcam

*Predictions plot for SIMCAM model*

---

### Description

Makes a plot with class predictions for calibration dataset.

### Usage

```
## S3 method for class 'simcam'
plotPredictions(
  obj,
  nc = seq_len(obj$nclasses),
  main = "SIMCAM Predictions (cal)",
  ...
)
```

### Arguments

obj	a SIMCAM model (object of class simcam)
nc	vector with class numbers to make the plot for.
main	plot title.
...	most of the graphical parameters from <a href="#">mdaplotg</a> function can be used.

### Details

See examples in description of [plsda](#), [simca](#) or [simcam](#).

---

plotPredictions.simcamres

*Prediction plot for SIMCAM results*

---

### Description

Makes a plot with predicted class values for classification results.

### Usage

```
## S3 method for class 'simcamres'
plotPredictions(obj, nc = seq_len(obj$nclasses), main = "Predictions", ...)
```

**Arguments**

obj	classification results (object of class <code>simcamres</code> ).
nc	vector with classes to show predictions for.
main	title of the plot
...	most of the graphical parameters from <code>mdaplotg</code> or <code>mdaplot</code> function can be used.

**Details**

See examples in description of `plsdares`, `simcamres`, etc.

---

`plotProbabilities`      *Plot for class belonging probability*

---

**Description**

Makes a plot with class belonging probabilities for each object of the classification results. Works only with classification methods, which compute this probability (e.g. SIMCA).

**Usage**

```
plotProbabilities(obj, ...)
```

**Arguments**

obj	an object with classification results (e.g. SIMCA)
...	other parameters

---

`plotProbabilities.classres`  
*Plot for class belonging probability*

---

**Description**

Makes a plot with class belonging probabilities for each object of the classification results. Works only with classification methods, which compute this probability (e.g. SIMCA).

**Usage**

```
## S3 method for class 'classres'
plotProbabilities(
  obj,
  ncomp = obj$ncomp.selected,
  nc = 1,
  type = "h",
  ylim = c(0, 1.1),
  show.lines = c(NA, 0.5),
  ...
)
```

**Arguments**

obj	classification results (e.g. object of class <code>simcamres</code> ).
ncomp	number of components to use the probabilities for.
nc	if there are several classes, which class to make the plot for.
type	type of the plot
ylim	vector with limits for y-axis
show.lines	shows a horizontal line at $p = 0.5$
...	most of the graphical parameters from <code>mdaplot</code> function can be used.

---

plotPurity

*Plot purity values*

---

**Description**

Plot purity values

**Usage**

```
plotPurity(obj, ...)
```

**Arguments**

obj	object with mcr pure case
...	other parameters

---

plotPurity.mcrpure      *Purity values plot*

---

**Description**

Purity values plot

**Usage**

```
## S3 method for class 'mcrpure'
plotPurity(
  obj,
  xticks = seq_len(obj$ncomp),
  type = "h",
  labels = "values",
  ...
)
```

**Arguments**

obj	mcrpure object
xticks	ticks for x axis
type	type of the plot
labels	what to use as data labels
...	other parameters suitable for mdaplot

The plot shows largest weighted purity value for each component graphically.

---

plotPuritySpectra      *Plot purity spectra*

---

**Description**

Plot purity spectra

**Usage**

```
plotPuritySpectra(obj, ...)
```

**Arguments**

obj	object with mcr pure case
...	other parameters

---

```
plotPuritySpectra.mcrpure
      Purity spectra plot
```

---

## Description

Purity spectra plot

## Usage

```
## S3 method for class 'mcrpure'
plotPuritySpectra(
  obj,
  comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  show.lines = TRUE,
  lines.col = adjustcolor(col, alpha.f = 0.75),
  lines.lty = 3,
  lines.lwd = 1,
  ...
)
```

## Arguments

<code>obj</code>	mcrpure object
<code>comp</code>	vector of components to show the purity spectra for
<code>type</code>	type of the plot
<code>col</code>	colors for the plot (should be a vector with one value for each component in <code>obj</code> )
<code>show.lines</code>	if TRUE show the selected pure variables as vertical lines
<code>lines.col</code>	color for the selected pure variable lines (by default same as for plots but semi-transparent)
<code>lines.lty</code>	line type for the purity lines
<code>lines.lwd</code>	line width for the purity lines
<code>...</code>	other parameters suitable for <code>mdaplotg</code>

The plot shows weighted purity value of each variable separately for each specified component.

---

plotQDoF *Degrees of freedom plot for orthogonal distance (Nq)*

---

### Description

Shows a plot with degrees of freedom computed for orthogonal distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

### Usage

```
plotQDoF(
  obj,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  ylab = "Nq",
  ...
)
```

### Arguments

obj	a PCA model (object of class pca)
type	type of the plot ("b", "l", "h")
labels	what to show as data points labels
xticks	vector with tick values for x-axis
ylab	label for y-axis
...	other plot parameters (see mdaplotg for details)

### Details

Works only if parameter `lim.type` equal to "ddmoments" or "ddrobust".

---

plotRegcoeffs *Regression coefficients plot*

---

### Description

Regression coefficients plot

### Usage

```
plotRegcoeffs(obj, ...)
```

**Arguments**

obj            a model or result object  
 ...            other arguments

**Details**

Generic function for plotting regression coefficients values for a regression model

---

plotRegcoeffs.regmodel

*Regression coefficient plot for regression model*

---

**Description**

Shows plot with regression coefficient values. Is a proxy for [plot.regcoeffs](#) method.

**Usage**

```
## S3 method for class 'regmodel'
plotRegcoeffs(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj            a regression model (object of class regmodel)  
 ncomp         number of components to show the plot for  
 ...            other plot parameters (see [plot.regcoeffs](#) for details)

---

plotRegressionLine    *Add regression line for data points*

---

**Description**

Shows linear fit line for data points.

**Usage**

```
plotRegressionLine(p, col = p$col, ...)
```

**Arguments**

p              plot data returned by function 'mdaplot()'  
 col            color of line  
 ...            other parameters available for 'abline()' function

---

plotResiduals	<i>Residuals plot</i>
---------------	-----------------------

---

**Description**

Residuals plot

**Usage**

```
plotResiduals(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting residual values for data decomposition

---

plotResiduals.ldecomp	<i>Residuals distance plot for a set of ldecomp objects (legacy, use <a href="#">plotDistances.ldecomp</a> instead).</i>
-----------------------	--

---

**Description**

Residuals distance plot for a set of ldecomp objects (legacy, use [plotDistances.ldecomp](#) instead).

**Usage**

```
## S3 method for class 'ldecomp'  
plotResiduals(obj, ...)
```

**Arguments**

obj	object of ldecomp class.
...	other parameters.

---

plotResiduals.pca	<i>Residuals distance plot for PCA model (legacy, use <a href="#">plotResiduals</a> instead).</i>
-------------------	---

---

**Description**

Residuals distance plot for PCA model (legacy, use [plotResiduals](#) instead).

**Usage**

```
## S3 method for class 'pca'
plotResiduals(obj, ...)
```

**Arguments**

obj	a PCA model (object of class pca)
...	any parameter from <a href="#">plotDistances.pca</a>

---

plotResiduals.regres	<i>Residuals plot for regression results</i>
----------------------	--

---

**Description**

Shows plot with Y residuals (difference between predicted and reference values) for selected response variable and complexity (number of components).

**Usage**

```
## S3 method for class 'regres'
plotResiduals(
  obj,
  ny = 1,
  ncomp = obj$ncomp.selected,
  show.lines = c(NA, 0),
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	regression results (object of class regres)
ny	number of predictor to show the plot for (if y is multivariate)
ncomp	complexity of model (e.g. number of components) to show the plot for
show.lines	allows to show the horizontal line at y = 0
show.plot	logical, show plot or just return plot data
...	other plot parameters (see <a href="#">mdaplot</a> for details)

---

 plotRMSE

*RMSE plot*


---

**Description**

RMSE plot

**Usage**

```
plotRMSE(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting RMSE values vs. complexity of a regression model

---

 plotRMSE.ipls

*RMSE development plot*


---

**Description**

Shows how RMSE develops for each iteration of iPLS selection algorithm.

**Usage**

```
## S3 method for class 'ipls'
plotRMSE(
  obj,
  glob.ncomp = obj$gm$ncomp.selected,
  main = "RMSE development",
  xlab = "Iterations",
  ylab = if (is.null(obj$cv)) "RMSEP" else "RMSECV",
  xlim = NULL,
  ylim = NULL,
  ...
)
```

**Arguments**

obj	iPLS results (object of class ipls).
glob.ncomp	number of components for global PLS model with all intervals.
main	main title for the plot.
xlab	label for x-axis.
ylab	label for y-axis.
xlim	limits for x-axis.
ylim	limits for y-axis.
...	other arguments.

**Details**

The plot shows RMSE values obtained at each iteration of the iPLS algorithm as bars. The first bar correspond to the global model with all variables included, second - to the model obtained at the first iteration and so on. Number at the bottom of each bar corresponds to the interval included or excluded at the particular iteration.

**See Also**

[summary.ipls](#), [plotSelection.ipls](#)

---

plotRMSE.regmodel      *RMSE plot for regression model*

---

**Description**

Shows plot with root mean squared error values vs. number of components for PLS model.

**Usage**

```
## S3 method for class 'regmodel'
plotRMSE(
  obj,
  ny = 1,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  res = obj$res,
  ylab = paste0("RMSE (", obj$res$cal$resnames[ny], ")"),
  ...
)
```

**Arguments**

obj	a regression model (object of class regmodel)
ny	number of response variable to make the plot for (if y is multivariate)
type	type of the plot("b", "l" or "h")
labels	what to show as labels (vector or name, e.g. "names", "values", "indices")
xticks	vector with ticks for x-axis values
res	list with result objects
ylab	label for y-axis
...	other plot parameters (see mdaplotg for details)

---

plotRMSE.regres	<i>RMSE plot for regression results</i>
-----------------	---

---

**Description**

Shows plot with RMSE values vs. model complexity (e.g. number of components).

**Usage**

```
## S3 method for class 'regres'
plotRMSE(
  obj,
  ny = 1,
  type = "b",
  xticks = seq_len(obj$ncomp),
  labels = "values",
  show.plot = TRUE,
  ylab = paste0("RMSE (", obj$resnames[ny], ")"),
  ...
)
```

**Arguments**

obj	regression results (object of class regres)
ny	number of predictor to show the plot for (if y is multivariate)
type	type of the plot
xticks	vector with ticks for x-axis
labels	what to use as labels ("names", "values" or "indices")
show.plot	logical, show plot or just return plot data
ylab	label for y-axis
...	other plot parameters (see mdaplot for details)

---

plotRMSERatio      *Plot for ratio RMSEC/RMSECV vs RMSECV*

---

### Description

Plot for ratio RMSEC/RMSECV vs RMSECV

### Usage

```
plotRMSERatio(obj, ...)
```

### Arguments

obj                    object with any regression model  
 ...                    other parameters

---

plotRMSERatio.regmodel  
*RMSECV/RMSEC ratio plot for regression model*

---

### Description

Shows plot with RMSECV/RMSEC values vs. RMSECV for each component.

### Usage

```
## S3 method for class 'regmodel'
plotRMSERatio(
  obj,
  ny = 1,
  type = "b",
  show.labels = TRUE,
  labels = seq_len(obj$ncomp),
  main = paste0("RMSECV/RMSEC ratio (", obj$res$cal$resnames[ny], ")"),
  ylab = "RMSECV/RMSEC ratio",
  xlab = "RMSECV",
  ...
)
```

**Arguments**

obj	a regression model (object of class regmodel)
ny	number of response variable to make the plot for (if y is multivariate)
type	type of the plot (use only "b" or "l")
show.labels	logical, show or not labels for plot points
labels	vector with point labels (by default number of components)
main	main plot title
ylab	label for y-axis
xlab	label for x-axis
...	other plot parameters (see mdaplot for details)

---

plotScatter

*Show plot series as set of points*


---

**Description**

Show plot series as set of points

**Usage**

```
plotScatter(
  ps,
  pch = 16,
  col = ps$col,
  bg = "white",
  lwd = 1,
  cex = 1,
  col.excluded = "lightgray",
  pch.colinv = FALSE,
  show.excluded = FALSE,
  ...
)
```

**Arguments**

ps	'plotseries' object
pch	marker symbol for the points
col	color of the points
bg	background color of the points if 'pch=21:25'
lwd	line width for the error bars
cex	scale factor for the marker
col.excluded	color for excluded values (if must be shown)

pch.colinv	logical, should 'col' and 'bg' be switched if 'pch=21:25' and 'cgroup' is used to create colors.
show.excluded	logical, show or not the excluded data points
...	other arguments for function 'points()'.

---

plotScores	<i>Scores plot</i>
------------	--------------------

---

### Description

Scores plot

### Usage

```
plotScores(obj, ...)
```

### Arguments

obj	a model or result object
...	other arguments

### Details

Generic function for scores values for data decomposition

---

plotScores.ldecomp	<i>Scores plot</i>
--------------------	--------------------

---

### Description

Shows a plot with scores values for data objects.

### Usage

```
## S3 method for class 'ldecomp'
plotScores(
  obj,
  comp = if (obj$ncomp > 1) c(1, 2) else 1,
  type = "p",
  show.axes = TRUE,
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	object of ldecomp class.
comp	which components to show the plot for (can be one value or vector with two values).
type	type of the plot
show.axes	logical, show or not axes lines crossing origin (0,0)
show.plot	logical, shall plot be created or just plot series object is needed
...	most of graphical parameters from <code>mdaplot</code> function can be used.

---

plotScores.pca	<i>Scores plot for PCA model</i>
----------------	----------------------------------

---

**Description**

Shows a scores plot for selected components.

**Usage**

```
## S3 method for class 'pca'
plotScores(
  obj,
  comp = if (obj$ncomp > 1) c(1, 2) else 1,
  type = "p",
  show.axes = TRUE,
  show.legend = TRUE,
  res = obj$res,
  ...
)
```

**Arguments**

obj	a PCA model (object of class pca)
comp	a value or vector with several values - number of components to show the plot for
type	type of the plot ("p", "l", "b", "h")
show.axes	logical, show or not axes lines crossing origin (0,0)
show.legend	logical, show or not a legend on the plot
res	list with result objects to show the scores for
...	other plot parameters (see <code>mdaplotg</code> for details)

**Details**

If plot is created only for one result object (e.g. calibration set), then the behaviour and all settings for the scores plot are identical to [plotScores.ldecomp](#). In this case you can show scores as a scatter, line or bar plot for any number of components.

Otherwise (e.g. if model contains results for calibration and test set) the plot is a group plot created using [mdaplotg](#) method and only scatter plot can be used.

See examples in help for [pca](#) function.

---

plotSelection	<i>Selected intervals plot</i>
---------------	--------------------------------

---

**Description**

Selected intervals plot

**Usage**

```
plotSelection(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting selected intervals or variables

---

plotSelection.ipls	<i>iPLS performance plot</i>
--------------------	------------------------------

---

**Description**

Shows PLS performance for each selected or excluded intervals at the first iteration.

**Usage**

```
## S3 method for class 'ipls'
plotSelection(
  obj,
  glob.ncomp = obj$gm$ncomp.selected,
  main = "iPLS results",
  xlab = obj$xaxis.name,
  ylab = if (is.null(obj$cv)) "RMSEP" else "RMSECV",
  xlim = NULL,
  ylim = NULL,
  ...
)
```

**Arguments**

obj	iPLS results (object of class ipls).
glob.ncomp	number of components for global PLS model with all intervals.
main	main title for the plot.
xlab	label for x-axis.
ylab	label for y-axis.
xlim	limits for x-axis.
ylim	limits for y-axis.
...	other arguments.

**Details**

The plot shows intervals as bars, which height corresponds to RMSECV obtained when particular interval was selected (forward) or excluded (backward) from a model at the first iteration. The intervals found optimal after backward/forward iPLS selection are shown with green color while the other intervals are gray.

See examples in help for [ipls](#) function.

**See Also**

[summary.ipls](#), [plotRMSE.ipls](#)

---

plotSelectivityArea     *Selectivity vs sensitivity plot for DD-SIMCA results (generic function)*

---

**Description**

Selectivity vs sensitivity plot for DD-SIMCA results (generic function)

**Usage**

```
plotSelectivityArea(obj, ...)
```

**Arguments**

```
obj          result object, e.g. ddsimcares
...          other parameters relevant for the plot
```

---

```
plotSelectivityArea.ddsimcares
```

*Selectivity area plot (similar to ROC curve).*

---

**Description**

The method generates a sequence of values for significance level (alpha) and then computes corresponding selectivity (1 - beta) for each value from the sequence using model parameters for given number of components and limit type. Then the values are shown as line plot using coordinates (1 - alpha) vs beta and area under the curve is shown in the plot title.

Current values of alpha and beta are shown on the plot as a marker.

**Usage**

```
## S3 method for class 'ddsimcares'
plotSelectivityArea(
  obj,
  ncomp = obj$ncomp.selected,
  limType = "classic",
  col = "#2679b2",
  xlab = bquote("1 - selectivity, " ~ beta),
  ylab = bquote("Sensitivity, 1 - " ~ alpha),
  ...
)
```

**Arguments**

```
obj          DD-SIMCA results (object of class ddsimcares).
ncomp        number of components in the model to show the plot for.
limType      limit type to show the plot for ('classic' or 'robust').
col          color of the plot elements.
xlab         label for x-axis.
ylab         label for y-axis.
...          other parameters compatible with mdaplot method.
```

---

plotSelectivityRatio *Selectivity ratio plot*

---

**Description**

Generic function for plotting selectivity ratio values for regression model (PCR, PLS, etc)

**Usage**

```
plotSelectivityRatio(obj, ...)
```

**Arguments**

obj	a regression model
...	other parameters

---

plotSelectivityRatio.pls

*Selectivity ratio plot for PLS model*

---

**Description**

Computes and shows a plot for Selectivity ratio values for given number of components and response variable

**Usage**

```
## S3 method for class 'pls'  
plotSelectivityRatio(obj, ny = 1, ncomp = obj$ncomp.selected, type = "l", ...)
```

**Arguments**

obj	a PLS model (object of class pls)
ny	which response to plot the values for (if y is multivariate), can be a vector.
ncomp	number of components to show
type	type of the plot
...	other plot parameters (see mdaPlot for details)

**Details**

See [selratio](#) for more details.

---

plotSensitivity      *Sensitivity plot*

---

**Description**

Sensitivity plot

**Usage**

```
plotSensitivity(obj, ...)
```

**Arguments**

obj                    a model or result object  
 ...                    other arguments

**Details**

Generic function for plotting sensitivity values for classification model or results

---

plotSensitivity.classmodel  
*Sensitivity plot for classification model*

---

**Description**

Makes a plot with sensitivity values vs. model complexity (e.g. number of components)

**Usage**

```
## S3 method for class 'classmodel'
plotSensitivity(obj, legend.position = "bottomright", ...)
```

**Arguments**

obj                    classification model (object of class plsda, simca, etc.).  
 legend.position      position of the legend (as in mdaplotg).  
 ...                    parameters for [plotPerformance.classmodel](#) function.

**Details**

See examples in description of [plsda](#), [simca](#) or [simcam](#).

---

plotSensitivity.classres  
*Sensitivity plot for classification results*

---

### Description

Makes a plot with sensitivity values vs. model complexity (e.g. number of components) for classification results.

### Usage

```
## S3 method for class 'classres'  
plotSensitivity(obj, legend.position = "bottomright", ...)
```

### Arguments

obj                    classification results (object of class `plsdares`, `simcamres`, etc.).  
legend.position        position of the legend (as in `mdaplotg`).  
...                    other parameters for [plotPerformance.classres](#)

### Details

See examples in description of [plsdares](#), [simcamres](#), etc.

---

plotSensitivity.ddsimca  
*Sensitivity plot.*

---

### Description

Sensitivity plot.

### Usage

```
## S3 method for class 'ddsimca'  
plotSensitivity(  
  obj,  
  limType = "classic",  
  col = mdaplot.getColors(2),  
  type = "b",  
  pch = c(16, 16),  
  legend.position = "bottomright",  
  ...  
)
```

**Arguments**

obj	DD-SIMCA model (object of class ddsimca).
limType	limit type to show the plot for ('classic' or 'robust').
col	vector with two colors (for calibration and PV-set sensitivity).
type	type of the plot ("b", "l", or "h").
pch	vector with two markers (for calibration and PV-set sensitivity).
legend.position	position of legend on the plot.
...	any parameters suitable for the <code>plotSensitivity.ddsimcares</code> method.

**Details**

The method shows sensitivity vs. number of components for calibration and PV-set results.

---

```
plotSensitivity.ddsimcares
```

*Sensitivity plot.*

---

**Description**

Sensitivity plot.

**Usage**

```
## S3 method for class 'ddsimcares'
plotSensitivity(
  obj,
  show.ci = FALSE,
  ci.col = "#a0a0a0",
  ci.lty = c(3, 2, 3),
  ...
)
```

**Arguments**

obj	DD-SIMCA results (object of class ddsimcares).
show.ci	logical, show or not the expected sensitivity and 95% confidence interval for it.
ci.col	color for lines showing expected sensitivity and confidence interval.
ci.lty	type of lines (lower, expected, upper) for the expected sensitivity and confidence interval.
...	any parameters suitable for the <code>plotFoM.ddsimcares</code> method.

**Details**

The method is a wrapper for more general `plotFoM.ddsimcares` method with a possibility to show confidence interval for the expected sensitivity.

---

`plotseries`*Create plot series object based on data, plot type and parameters*

---

## Description

The 'plotseries' object contains all necessary parameters to create main plots from data values, including values for x and y, correct handling of excluded rows and columns, color grouping (if any), limits and labels.

If both 'col' and 'cgroup' are specified, 'cgroup' will be ignored.

Labels can be either provided by user or generated automatically based on values, names or indices of data rows and columns. If series is made for scatter plot 'type="p"' then labels are required for each row of the original dataset. Otherwise (for line, bar and errorbar plot) labels correspond to data columns (variables).

The object has the following plotting methods once created: [plotScatter](#) [plotLines](#) [plotBars](#) [plotDensity](#) [plotErrorbars](#)

## Usage

```
plotseries(  
  data,  
  type,  
  cgroup = NULL,  
  col = NULL,  
  opacity = 1,  
  colmap = "default",  
  labels = NULL  
)
```

## Arguments

<code>data</code>	data to make the plot for (vector, matrix or data frame).
<code>type</code>	type of the plot.
<code>cgroup</code>	vector with values used to create a color grouping of the series instances.
<code>col</code>	color to show the series on plot with (user defined).
<code>opacity</code>	opacity of the colors (between 0 and 1).
<code>colmap</code>	colormap name to generate color/colors if they are not specified by user. See <a href="#">mdaplot.getColor</a> s for details.
<code>labels</code>	either vector with labels for the series instances or string ("names", "values", or "indices") if labels should be generated automatically.

---

plotSpecificity      *Specificity plot*

---

**Description**

Specificity plot

**Usage**

```
plotSpecificity(obj, ...)
```

**Arguments**

obj                    a model or result object  
 ...                    other arguments

**Details**

Generic function for plotting specificity values for classification model or results

---

plotSpecificity.classmodel  
*Specificity plot for classification model*

---

**Description**

Makes a plot with specificity values vs. model complexity (e.g. number of components)

**Usage**

```
## S3 method for class 'classmodel'
plotSpecificity(obj, legend.position = "bottomright", ...)
```

**Arguments**

obj                    classification model (object of class plsd, simca, etc.).  
 legend.position      position of the legend (as in mdaplotg).  
 ...                    parameters for [plotPerformance.classmodel](#) function.

**Details**

See examples in description of [plsd](#), [simca](#) or [simcam](#).

---

`plotSpecificity.classres`*Specificity plot for classification results*

---

**Description**

Makes a plot with specificity values vs. model complexity (e.g. number of components) for classification results.

**Usage**

```
## S3 method for class 'classres'  
plotSpecificity(obj, legend.position = "bottomright", ...)
```

**Arguments**

<code>obj</code>	classification results (object of class <code>plsdares</code> , <code>simcamres</code> , etc.).
<code>legend.position</code>	position of the legend (as in <code>mdaplotg</code> ).
<code>...</code>	other parameters for <a href="#">plotPerformance.classres</a>

**Details**

See examples in description of [plsdares](#), [simcamres](#), etc.

---

`plotSpectra`*Plot resolved spectra*

---

**Description**

Plot resolved spectra

**Usage**

```
plotSpectra(obj, ...)
```

**Arguments**

<code>obj</code>	object with mcr case
<code>...</code>	other parameters

---

plotSpectra.mcr      *Show plot with resolved spectra*

---

### Description

Show plot with resolved spectra

### Usage

```
## S3 method for class 'mcr'
plotSpectra(
  obj,
  comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  ...
)
```

### Arguments

obj	object of class mcr
comp	vector with number of components to make the plot for
type	type of the plot
col	vector with colors for individual components
...	other parameters suitable for mdaplotg

---

plotT2DoF      *Degrees of freedom plot for score distance (Nh)*

---

### Description

Shows a plot with degrees of freedom computed for score distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

### Usage

```
plotT2DoF(
  obj,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  ylab = "Nh",
  ...
)
```

**Arguments**

obj	a PCA model (object of class pca)
type	type of the plot ("b", "l", "h")
labels	what to show as data points labels
xticks	vector with tick values for x-axis
ylab	label for y-axis
...	other plot parameters (see mdaPlotg for details)

**Details**

Works only if parameter `lim.type` equal to "ddmoments" or "ddrobust".

---

plotVariance	<i>Variance plot</i>
--------------	----------------------

---

**Description**

Variance plot

**Usage**

```
plotVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting explained variance for data decomposition

---

plotVariance.ldecomp    *Explained variance plot*

---

### Description

Shows a plot with explained variance vs. number of components.

### Usage

```
## S3 method for class 'ldecomp'
plotVariance(
  obj,
  type = "b",
  variance = "expvar",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  show.plot = TRUE,
  ylab = "Explained variance, %",
  ...
)
```

### Arguments

obj	object of ldecomp class.
type	type of the plot
variance	string, which variance to make the plot for ("expvar", "cumexpvar")
labels	what to show as labels for plot objects.
xticks	vector with ticks for x-axis
show.plot	logical, shall plot be created or just plot series object is needed
ylab	label for y-axis
...	most of graphical parameters from <a href="#">mdaplot</a> function can be used.

---

plotVariance.mcr    *Show plot with explained variance*

---

### Description

Show plot with explained variance

**Usage**

```
## S3 method for class 'mcr'  
plotVariance(  
  obj,  
  type = "h",  
  labels = "values",  
  main = "Variance",  
  xticks = seq_len(obj$ncomp),  
  ...  
)
```

**Arguments**

obj	object of class mcr
type	type of the plot
labels	what to use as data labels
main	title of the plot
xticks	vector with ticks for x-axis
...	other parameters suitable for mdaplot

---

plotVariance.pca

*Explained variance plot for PCA model*

---

**Description**

Shows a plot with explained variance or cumulative explained variance for components.

**Usage**

```
## S3 method for class 'pca'  
plotVariance(  
  obj,  
  type = "b",  
  labels = "values",  
  variance = "expvar",  
  xticks = seq_len(obj$ncomp),  
  res = obj$res,  
  ylab = "Explained variance, %",  
  ...  
)
```

**Arguments**

obj	a PCA model (object of class <code>pca</code> )
type	type of the plot ("b", "l", "h")
labels	what to use as labels (if <code>show.labels = TRUE</code> )
variance	which variance to show
xticks	vector with ticks for x-axis
res	list with result objects to show the variance for
ylab	label for y-axis
...	other plot parameters (see <code>mdaplotg</code> for details)

**Details**

See examples in help for `pca` function.

---

plotVariance.pls      *Variance plot for PLS*

---

**Description**

Shows plot with variance values vs. number of components.

**Usage**

```
## S3 method for class 'pls'
plotVariance(
  obj,
  decomp = "xdecomp",
  variance = "expvar",
  type = "b",
  labels = "values",
  res = obj$res,
  ylab = "Explained variance, %",
  ...
)
```

**Arguments**

obj	a PLS model (object of class <code>pls</code> )
decomp	which decomposition to use ("xdecomp" for x or "ydecomp" for y)
variance	which variance to use ("expvar", "cumexpvar")
type	type of the plot("b", "l" or "h")
labels	what to show as labels for plot objects.
res	list with result objects to show the plot for (by default, model results are used)
ylab	label for y-axis
...	other plot parameters (see <code>mdaplotg</code> for details)

**Details**

See examples in help for [pls](#) function.

---

plotVariance.plsres     *Explained X variance plot for PLS results*

---

**Description**

Shows plot with explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'  
plotVariance(obj, decomp = "xdecomp", variance = "expvar", ...)
```

**Arguments**

obj	PLS results (object of class plsres)
decomp	which decomposition to use ("xdecomp" or "ydecomp")
variance	which variance to use ("expvar", "cumexpvar")
...	other plot parameters (see mdaplot for details)

**Details**

See examples in help for [plsres](#) function.

---

plotVIPScores     *VIP scores plot*

---

**Description**

Generic function for plotting VIP scores values for regression model (PCR, PLS, etc)

**Usage**

```
plotVIPScores(obj, ...)
```

**Arguments**

obj	a regression model
...	other parameters

---

plotVIPScores.pls      *VIP scores plot for PLS model*

---

### Description

Shows a plot with VIP scores values for given number of components and response variable

### Usage

```
## S3 method for class 'pls'
plotVIPScores(
  obj,
  ny = 1,
  ncomp = obj$ncomp.selected,
  type = "l",
  vip.type = "individual",
  ...
)
```

### Arguments

obj	a PLS model (object of class pls)
ny	which response to plot the values for (if y is multivariate), can be a vector. Ignored when vip.type = "combined".
ncomp	number of components to show
type	type of the plot
vip.type	type of VIP scores: "individual" or "combined" (see <a href="#">vipcores</a> )
...	other plot parameters (see mdaplot for details)

### Details

See [vipcores](#) for more details.

---

plotWeights      *Plot for PLS weights*

---

### Description

Plot for PLS weights

### Usage

```
plotWeights(obj, ...)
```

**Arguments**

obj            a model or result object  
 ...            other arguments

**Details**

Generic function for weight plot

---

plotWeights.pls            *Weights plot for PLS*

---

**Description**

Shows plot with weight values for selected components.

**Usage**

```
## S3 method for class 'pls'
plotWeights(
  obj,
  comp = 1,
  type = (if (nrow(obj$weights) < 20) "h" else "l"),
  show.axes = TRUE,
  show.legend = TRUE,
  ...
)
```

**Arguments**

obj            a PLS model (object of class pls)  
 comp          which components to show the plot for (one or vector with several values)  
 type          type of the plot  
 show.axes     logical, show or not axes lines crossing origin (0,0)  
 show.legend   logical, show or not a legend  
 ...            other plot parameters (see mdaPlotg for details)

**Details**

See examples in help for [pls](#) function.

plotXCumVariance      *X cumulative variance plot*

---

**Description**

X cumulative variance plot

**Usage**

```
plotXCumVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting cumulative explained variance for decomposition of x data

---

plotXCumVariance.pls      *Cumulative explained X variance plot for PLS*

---

**Description**

Shows plot with cumulative explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'pls'  
plotXCumVariance(obj, type = "b", main = "Cumulative variance (X)", ...)
```

**Arguments**

obj	a PLS model (object of class pls)
type	type of the plot("b", "l" or "h")
main	title for the plot
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

`plotXCumVariance.plsres`*Explained cumulative X variance plot for PLS results*

---

**Description**

Shows plot with cumulative explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'  
plotXCumVariance(obj, main = "Cumulative variance (X)", ...)
```

**Arguments**

<code>obj</code>	PLS results (object of class <code>plsres</code> )
<code>main</code>	main plot title
<code>...</code>	other plot parameters (see <code>mdaplot</code> for details)

**Details**

See examples in help for [plsres](#) function.

---

`plotXLoadings`*X loadings plot*

---

**Description**

X loadings plot

**Usage**

```
plotXLoadings(obj, ...)
```

**Arguments**

<code>obj</code>	a model or result object
<code>...</code>	other arguments

**Details**

Generic function for plotting loadings values for decomposition of x data

---

plotXLoadings.pls      *X loadings plot for PLS*

---

### Description

Shows plot with X loading values for selected components.

### Usage

```
## S3 method for class 'pls'
plotXLoadings(
  obj,
  comp = if (obj$ncomp > 1) c(1, 2) else 1,
  type = "p",
  show.axes = TRUE,
  show.legend = TRUE,
  ...
)
```

### Arguments

obj	a PLS model (object of class pls)
comp	which components to show the plot for (one or vector with several values)
type	type of the plot
show.axes	logical, show or not axes lines crossing origin (0,0)
show.legend	logical, show or not legend on the plot (when it is available)
...	other plot parameters (see mdaplotg for details)

### Details

See examples in help for [pls](#) function.

---

plotXResiduals      *X residuals plot*

---

### Description

X residuals plot

### Usage

```
plotXResiduals(obj, ...)
```

**Arguments**

obj            a model or result object  
 ...            other arguments

**Details**

Generic function for plotting x residuals for classification or regression model or results

---

plotXResiduals.pls      *Residual distance plot for decomposition of X data*

---

**Description**

Shows a plot with orthogonal distance vs score distance for PLS decomposition of X data.

**Usage**

```
## S3 method for class 'pls'
plotXResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  main = sprintf("X-distances (ncomp = %d)", ncomp),
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = c(TRUE, TRUE),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  show.legend = TRUE,
  legend.position = "topright",
  res = obj$res,
  ...
)
```

**Arguments**

obj            a PLS model (object of class pls)  
 ncomp         how many components to use (by default optimal value selected for the model will be used)  
 norm         logical, normalize distance values or not (see details)  
 log           logical, apply log transformation to the distances or not (see details)  
 main         title for the plot

cgroup	color grouping of plot points (works only if one result object is available)
xlim	limits for x-axis
ylim	limits for y-axis
show.limits	vector with two logical values defining if limits for extreme and/or outliers must be shown
lim.col	vector with two values - line color for extreme and outlier limits
lim.lwd	vector with two values - line width for extreme and outlier limits
lim.lty	vector with two values - line type for extreme and outlier limits
show.legend	logical, show or not a legend on the plot (needed if several result objects are available)
legend.position	position of legend (if shown)
res	list with result objects to show the plot for (by default, model results are used)
...	other plot parameters (see <code>mdaplotg</code> for details)

### Details

The function is almost identical to [plotResiduals.pca](#).

---

`plotXResiduals.plsres` *X residuals plot for PLS results*

---

### Description

Shows a plot with Q residuals vs. Hotelling T2 values for PLS decomposition of x data.

### Usage

```
## S3 method for class 'plsres'
plotXResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  main = sprintf("X-distances (ncomp = %d)", ncomp),
  ...
)
```

### Arguments

obj	PLS results (object of class <code>plsres</code> )
ncomp	how many components to use (if NULL - user selected optimal value will be used)
norm	logical, normalize distance values or not (see details)

log            logical, apply log transformation to the distances or not (see details)  
 main          main title for the plot  
 ...          other plot parameters (see mdaPlot for details)

**Details**

See examples in help for [plsres](#) function.

---

plotXScores            *X scores plot*

---

**Description**

X scores plot

**Usage**

```
plotXScores(obj, ...)
```

**Arguments**

obj            a model or result object  
 ...          other arguments

**Details**

Generic function for plotting scores values for decomposition of x data

---

plotXScores.pls            *X scores plot for PLS*

---

**Description**

Shows plot with X scores values for selected components.

**Usage**

```
## S3 method for class 'pls'
plotXScores(
  obj,
  comp = if (obj$ncomp > 1) c(1, 2) else 1,
  show.axes = TRUE,
  main = "Scores (X)",
  res = obj$res,
  ...
)
```

**Arguments**

obj	a PLS model (object of class pls)
comp	which components to show the plot for (one or vector with several values)
show.axes	logical, show or not axes lines crossing origin (0,0)
main	main plot title
res	list with result objects to show the plot for (by default, model results are used)
...	other plot parameters (see mdaPlotg for details)

**Details**

See examples in help for [pls](#) function.

---

plotXScores.plsres      *X scores plot for PLS results*

---

**Description**

Shows plot with scores values for PLS decomposition of x data.

**Usage**

```
## S3 method for class 'plsres'
plotXScores(obj, comp = c(1, 2), main = "Scores (X)", ...)
```

**Arguments**

obj	PLS results (object of class plsres)
comp	which components to show the plot for (one or vector with several values)
main	main plot title
...	other plot parameters (see mdaPlot for details)

**Details**

See examples in help for [plsres](#) function.

---

plotXVariance	<i>X variance plot</i>
---------------	------------------------

---

**Description**

X variance plot

**Usage**

```
plotXVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting explained variance for decomposition of x data

---

plotXVariance.pls	<i>Explained X variance plot for PLS</i>
-------------------	--

---

**Description**

Shows plot with explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'pls'  
plotXVariance(obj, type = "b", main = "Variance (X)", ...)
```

**Arguments**

obj	a PLS model (object of class pls)
type	type of the plot("b", "l" or "h")
main	title for the plot
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

plotXVariance.plsres *Explained X variance plot for PLS results*

---

**Description**

Shows plot with explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'  
plotXVariance(obj, main = "Variance (X)", ...)
```

**Arguments**

obj	PLS results (object of class plsres)
main	main plot title
...	other plot parameters (see mdaplot for details)

**Details**

See examples in help for [plsres](#) function.

---

plotXYLoadings *XY loadings plot*

---

**Description**

XY loadings plot

**Usage**

```
plotXYLoadings(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting loadings values for decomposition of x and y data

---

plotXYLoadings.pls      *XY loadings plot for PLS*

---

**Description**

Shows plot with X and Y loading values for selected components.

**Usage**

```
## S3 method for class 'pls'  
plotXYLoadings(obj, comp = c(1, 2), show.axes = TRUE, ...)
```

**Arguments**

obj	a PLS model (object of class pls)
comp	which components to show the plot for (one or vector with several values)
show.axes	logical, show or not axes lines crossing origin (0,0)
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

plotXYResiduals      *Plot for XY-residuals*

---

**Description**

Plot for XY-residuals

**Usage**

```
plotXYResiduals(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for XY-residuals plot

---

plotXYResiduals.pls     *Residual XY-distance plot*

---

### Description

Shows a plot with full X-distance (f) vs. orthogonal Y-distance (z) for PLS model results.

### Usage

```
## S3 method for class 'pls'
plotXYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  main = sprintf("XY-distances (ncomp = %d)", ncomp),
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = c(TRUE, TRUE),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  show.legend = TRUE,
  legend.position = "topright",
  res = obj$res,
  ...
)
```

### Arguments

obj	a PLS model (object of class pls)
ncomp	how many components to use (by default optimal value selected for the model will be used)
norm	logical, normalize distance values or not (see details)
log	logical, apply log transformation to the distances or not (see details)
main	title for the plot
cgroup	color grouping of plot points (works only if one result object is available)
xlim	limits for x-axis
ylim	limits for y-axis
show.limits	vector with two logical values defining if limits for extreme and/or outliers must be shown
lim.col	vector with two values - line color for extreme and outlier limits
lim.lwd	vector with two values - line width for extreme and outlier limits

lim.lty	vector with two values - line type for extreme and outlier limits
show.legend	logical, show or not a legend on the plot (needed if several result objects are available)
legend.position	position of legend (if shown)
res	list with result objects to show the plot for (by default, model results are used)
...	other plot parameters (see mdaplotg for details)

### Details

The function presents a way to identify extreme objects and outliers based on both full distance for X-decomposition (known as  $f$ ) and squared residual distance for Y-decomposition ( $z$ ). The approach has been proposed in [1].

The plot is available only if data driven methods (classic or robust) have been used for computing of critical limits.

### References

1. Rodionova O. Ye., Pomerantsev A. L. Detection of Outliers in Projection-Based Modeling. Analytical Chemistry (2020, in publish). doi: 10.1021/acs.analchem.9b04611

---

plotXYResiduals.plsres

*Residual distance plot*

---

### Description

Shows a plot with orthogonal (Q, q) vs. score (T2, h) distances for data objects.

### Usage

```
## S3 method for class 'plsres'
plotXYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  show.labels = FALSE,
  labels = "names",
  show.plot = TRUE,
  ...
)
```

**Arguments**

obj	object of plsres class.
ncomp	number of components to show the plot for (if NULL, selected by model value will be used).
norm	logical, normalize distance values or not (see details)
log	logical, apply log transformation to the distances or not (see details)
show.labels	logical, show or not labels for the plot objects
labels	what to show as labels if necessary
show.plot	logical, shall plot be created or just plot series object is needed
...	most of graphical parameters from <code>mdaplot</code> function can be used.

---

plotXYScores	<i>XY scores plot</i>
--------------	-----------------------

---

**Description**

XY scores plot

**Usage**

```
plotXYScores(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting scores values for decomposition of x and y data

---

plotXYScores.pls	<i>XY scores plot for PLS</i>
------------------	-------------------------------

---

**Description**

Shows plot with X vs. Y scores values for selected component.

**Usage**

```
## S3 method for class 'pls'
plotXYScores(obj, ncomp = 1, show.axes = TRUE, res = obj$res, ...)
```

**Arguments**

obj	a PLS model (object of class pls)
ncomp	which component to show the plot for
show.axes	logical, show or not axes lines crossing origin (0,0)
res	list with result objects to show the plot for (by default, model results are used)
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

plotXYScores.plsres    *XY scores plot for PLS results*

---

**Description**

Shows plot with X vs. Y scores values for PLS results.

**Usage**

```
## S3 method for class 'plsres'  
plotXYScores(obj, ncomp = 1, show.plot = TRUE, ...)
```

**Arguments**

obj	PLS results (object of class plsres)
ncomp	which component to show the plot for
show.plot	logical, show plot or just return plot data
...	other plot parameters (see mdaplot for details)

**Details**

See examples in help for [plsres](#) function.

---

plotYCumVariance      *Y cumulative variance plot*

---

**Description**

Y cumulative variance plot

**Usage**

```
plotYCumVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting cumulative explained variance for decomposition of y data

---

plotYCumVariance.pls      *Cumulative explained Y variance plot for PLS*

---

**Description**

Shows plot with cumulative explained Y variance vs. number of components.

**Usage**

```
## S3 method for class 'pls'
plotYCumVariance(obj, type = "b", main = "Cumulative variance (Y)", ...)
```

**Arguments**

obj	a PLS model (object of class pls)
type	type of the plot("b", "l" or "h")
main	title for the plot
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

`plotYCumVariance.plsres`*Explained cumulative Y variance plot for PLS results*

---

**Description**

Shows plot with cumulative explained Y variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'  
plotYCumVariance(obj, main = "Cumulative variance (Y)", ...)
```

**Arguments**

<code>obj</code>	PLS results (object of class <code>plsres</code> )
<code>main</code>	main plot title
<code>...</code>	other plot parameters (see <code>mdaplot</code> for details)

**Details**

See examples in help for [plsres](#) function.

---

`plotYResiduals`*Y residuals plot*

---

**Description**

Y residuals plot

**Usage**

```
plotYResiduals(obj, ...)
```

**Arguments**

<code>obj</code>	a model or result object
<code>...</code>	other arguments

**Details**

Generic function for plotting y residuals for classification or regression model or results

---

plotYResiduals.plsres *Y residuals plot for PLS results*

---

### Description

Shows a plot with Y residuals vs reference Y values for selected component.

### Usage

```
## S3 method for class 'plsres'
plotYResiduals(obj, ncomp = obj$ncomp.selected, ...)
```

### Arguments

obj	PLS results (object of class plsres)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
...	other plot parameters (see mdaplot for details)

### Details

Proxy for [plotResiduals.regres](#) function.

---

plotYResiduals.regmodel  
*Y residuals plot for regression model*

---

### Description

Shows plot with y residuals (predicted vs. reference values) for selected components.

### Usage

```
## S3 method for class 'regmodel'
plotYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  ny = 1,
  show.lines = c(NA, 0),
  res = obj$res,
  ...
)
```

**Arguments**

obj	a regression model (object of class regmodel)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
ny	number of response variable to make the plot for (if y is multivariate)
show.lines	allows to show the horizontal line at 0 level
res	list with result objects
...	other plot parameters (see mdaplotg for details)

---

plotYVariance	<i>Y variance plot</i>
---------------	------------------------

---

**Description**

Y variance plot

**Usage**

```
plotYVariance(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for plotting explained variance for decomposition of y data

---

plotYVariance.pls	<i>Explained Y variance plot for PLS</i>
-------------------	--

---

**Description**

Shows plot with explained Y variance vs. number of components.

**Usage**

```
## S3 method for class 'pls'
plotYVariance(obj, type = "b", main = "Variance (Y)", ...)
```

**Arguments**

obj	a PLS model (object of class pls)
type	type of the plot("b", "l" or "h")
main	title for the plot
...	other plot parameters (see mdaplotg for details)

**Details**

See examples in help for [pls](#) function.

---

plotYVariance.plsres *Explained Y variance plot for PLS results*

---

**Description**

Shows plot with explained Y variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'  
plotYVariance(obj, main = "Variance (Y)", ...)
```

**Arguments**

obj	PLS results (object of class plsres)
main	main plot title
...	other plot parameters (see mdaplot for details)

**Details**

See examples in help for [plsres](#) function.

**Description**

pls is used to calibrate, validate and apply partial least squares (PLS) regression model.

**Usage**

```
pls(  
  x,  
  y,  
  ncomp = min(nrow(x) - 1, ncol(x), 20),  
  center = TRUE,  
  scale = FALSE,  
  cv = NULL,  
  exclcols = NULL,  
  exclrows = NULL,  
  x.test = NULL,  
  y.test = NULL,  
  method = "simpls",  
  info = "",  
  ncomp.selcrit = "min",  
  lim.type = "ddmoments",  
  alpha = 0.05,  
  gamma = 0.01,  
  cv.scope = "local",  
  prep = NULL  
)
```

**Arguments**

x	matrix with predictors.
y	matrix with responses.
ncomp	maximum number of components to calculate.
center	logical, center or not predictors and response values.
scale	logical, scale (standardize) or not predictors and response values.
cv	cross-validation settings (see details).
exclcols	columns of x to be excluded from calculations (numbers, names or vector with logical values)
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)
x.test	matrix with predictors for test set.
y.test	matrix with responses for test set.

method	algorithm for computing PLS model (only 'simpls' is supported so far)
info	short text with information about the model.
ncomp.selcrit	criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
lim.type	which method to use for calculation of critical limits for residual distances (see details)
alpha	significance level for extreme limits for T2 and Q distances.
gamma	significance level for outlier limits for T2 and Q distances.
cv.scope	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.
prep	optional list with preprocessing methods created using 'prep' function.

### Details

So far only SIMPLS method [1] is available. Implementation works both with one and multiple response variables.

Like in [pca](#), pls uses number of components (ncomp) as a minimum of number of objects - 1, number of x variables and the default or provided value. Regression coefficients, predictions and other results are calculated for each set of components from 1 to ncomp: 1, 1:2, 1:3, etc. The optimal number of components, (ncomp.selected), is found using first local minimum, but can be also forced to user defined value using function ([selectCompNum.pls](#)). The selected optimal number of components is used for all default operations - predictions, plots, etc.

Cross-validation settings, cv, can be a number or a list. If cv is a number, it will be used as a number of segments for random cross-validation (if cv = 1, full cross-validation will be performed). If it is a list, the following syntax can be used: `cv = list("rand", nseg, nrep)` for random repeated cross-validation with nseg segments and nrep repetitions or `cv = list("ven", nseg)` for systematic splits to nseg segments ('venetian blinds').

Calculation of confidence intervals and p-values for regression coefficients can be done based on Jack-Knifing resampling. This is done automatically if cross-validation is used. However it is recommended to use at least 10 segments for stable JK result. See help for [regcoeffs](#) objects for more details.

If you provide a list with preprocessing methods, PLS will apply them to the training set before excluding the columns and rows (if specified). The list will be used to train a preprocessing model which becomes a part of the PLS model object. So when you use method 'predict()' the provided dataset will be automatically preprocessed by the preprocessing model.

Any PLS model (with or without preprocessing) developed in this package can be saved as JSON file using method [writeJSON](#) and then be loaded to interactive web-application for PLS available at <https://mda.tools/pls>. Likewise one can develop a model in the app, save it to JSON file and then load it to R by using method [readJSON](#). In this case, however, the model object will not contain calibration/training results, so some of the plots and statistics will not be available.

### Value

Returns an object of pls class with following fields:

ncomp	number of components included in the model.
ncomp.selected	selected (optimal) number of components.
xcenter	vector with values used to center the predictors (x).
ycenter	vector with values used to center the responses (y).
xscale	vector with values used to scale the predictors (x).
yscale	vector with values used to scale the responses (y).
xloadings	matrix with loading values for x decomposition.
yloadings	matrix with loading values for y decomposition.
xeigenvals	vector with eigenvalues of components (variance of x-scores).
yeigenvals	vector with eigenvalues of components (variance of y-scores).
weights	matrix with PLS weights.
coeffs	object of class <code>regcoeffs</code> with regression coefficients calculated for each component.
info	information about the model, provided by user when building the model.
prep	trained preprocessing model (if specified)
cv	information about cross-validation method used (if any).
res	a list with result objects (e.g. calibration, cv, etc.)

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**References**

1. S. de Jong, Chemometrics and Intelligent Laboratory Systems 18 (1993) 251-263.
2. Tarja Rajalahti et al. Chemometrics and Laboratory Systems, 95 (2009), 35-48.
3. Il-Gyo Chong, Chi-Hyuck Jun. Chemometrics and Laboratory Systems, 78 (2005), 103-112.

**See Also**

Main methods for pls objects:

<code>print</code>	prints information about a pls object.
<code>summary.pls</code>	shows performance statistics for the model.
<code>plot.pls</code>	shows plot overview of the model.
<code>pls.simpls</code>	implementation of SIMPLS algorithm.
<code>predict.pls</code>	applies PLS model to a new data.
<code>selectCompNum.pls</code>	set number of optimal components in the model.
<code>setDistanceLimits.pls</code>	allows to change parameters for critical limits.
<code>categorize.pls</code>	categorize data rows similar to <code>categorize.pca</code> .
<code>selratio</code>	computes matrix with selectivity ratio values.
<code>vipscores</code>	computes matrix with VIP scores values.

Plotting methods for pls objects:

<code>plotXScores.pls</code>	shows scores plot for x decomposition.
<code>plotXYScores.pls</code>	shows scores plot for x and y decomposition.
<code>plotXLoadings.pls</code>	shows loadings plot for x decomposition.
<code>plotXYLoadings.pls</code>	shows loadings plot for x and y decomposition.
<code>plotXVariance.pls</code>	shows explained variance plot for x decomposition.
<code>plotYVariance.pls</code>	shows explained variance plot for y decomposition.
<code>plotXCumVariance.pls</code>	shows cumulative explained variance plot for x decomposition.
<code>plotYCumVariance.pls</code>	shows cumulative explained variance plot for y decomposition.
<code>plotXResiduals.pls</code>	shows distance/residuals plot for x decomposition.
<code>plotXYResiduals.pls</code>	shows joint distance plot for x and y decomposition.
<code>plotWeights.pls</code>	shows plot with weights.
<code>plotSelectivityRatio.pls</code>	shows plot with selectivity ratio values.
<code>plotVIPScores.pls</code>	shows plot with VIP scores values.

Methods inherited from `regmodel` object (parent class for `pls`):

<code>plotPredictions.regmodel</code>	shows predicted vs. measured plot.
<code>plotRMSE.regmodel</code>	shows RMSE plot.
<code>plotRMSERatio.regmodel</code>	shows plot for ratio RMSECV/RMSEC values.
<code>plotYResiduals.regmodel</code>	shows residuals plot for y values.
<code>getRegcoeffs.regmodel</code>	returns matrix with regression coefficients.

Most of the methods for plotting data (except loadings and regression coefficients) are also available for PLS results (`plsres`) objects. There is also a randomization test for PLS-regression (`randtest`) and implementation of interval PLS algorithm for variable selection (`ipls`)

## Examples

```
### Examples of using PLS model class
library(mdatools)

## 1. Make a PLS model for concentration of first component
## using full-cross validation and automatic detection of
## optimal number of components and show an overview

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]

model = pls(x, y, ncomp = 8, cv = 1)
summary(model)
plot(model)

## 2. Make a PLS model for concentration of first component
## using test set and 10 segment cross-validation and show overview

data(simdata)
x = simdata$spectra.c
```

```
y = simdata$conc.c[, 1]
x.t = simdata$spectra.t
y.t = simdata$conc.t[, 1]

model = pls(x, y, ncomp = 8, cv = 10, x.test = x.t, y.test = y.t)
model = selectCompNum(model, 2)
summary(model)
plot(model)

## 3. Make a PLS model for concentration of first component
## using only test set validation and show overview

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]
x.t = simdata$spectra.t
y.t = simdata$conc.t[, 1]

model = pls(x, y, ncomp = 6, x.test = x.t, y.test = y.t)
model = selectCompNum(model, 2)
summary(model)
plot(model)

## 4. Show variance and error plots for a PLS model
par(mfrow = c(2, 2))
plotXCumVariance(model, type = 'h')
plotYCumVariance(model, type = 'b', show.labels = TRUE, legend.position = 'bottomright')
plotRMSE(model)
plotRMSE(model, type = 'h', show.labels = TRUE)
par(mfrow = c(1, 1))

## 5. Show scores plots for a PLS model
par(mfrow = c(2, 2))
plotXScores(model)
plotXScores(model, comp = c(1, 3), show.labels = TRUE)
plotXYScores(model)
plotXYScores(model, comp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

## 6. Show loadings and coefficients plots for a PLS model
par(mfrow = c(2, 2))
plotXLoadings(model)
plotXLoadings(model, comp = c(1, 2), type = 'l')
plotXYLoadings(model, comp = c(1, 2), legend.position = 'topleft')
plotRegcoeffs(model)
par(mfrow = c(1, 1))

## 7. Show predictions and residuals plots for a PLS model
par(mfrow = c(2, 2))
plotXResiduals(model, show.label = TRUE)
plotYResiduals(model, show.label = TRUE)
plotPredictions(model)
plotPredictions(model, ncomp = 4, xlab = 'C, reference', ylab = 'C, predictions')
```

```

par(mfrow = c(1, 1))

## 8. Selectivity ratio and VIP scores plots
par(mfrow = c(2, 2))
plotSelectivityRatio(model)
plotSelectivityRatio(model, ncomp = 1)
par(mfrow = c(1, 1))

## 9. Variable selection with selectivity ratio
selratio = getSelectivityRatio(model)
selvar = !(selratio < 8)

xsel = x[, selvar]
modelsel = pls(xsel, y, ncomp = 6, cv = 1)
modelsel = selectCompNum(modelsel, 3)

summary(model)
summary(modelsel)

## 10. Calculate average spectrum and show the selected variables
i = 1:ncol(x)
ms = apply(x, 2, mean)

par(mfrow = c(2, 2))

plot(i, ms, type = 'p', pch = 16, col = 'red', main = 'Original variables')
plotPredictions(model)

plot(i, ms, type = 'p', pch = 16, col = 'lightgray', main = 'Selected variables')
points(i[selvar], ms[selvar], col = 'red', pch = 16)
plotPredictions(modelsel)

par(mfrow = c(1, 1))

## 12. Use list with preprocessing methods

# get the data (calibration and test set)
data(simdata)
Xc <- simdata$spectra.c
Xt <- simdata$spectra.t
yc <- simdata$conc.c[, 3]
yt <- simdata$conc.t[, 3]

# create a list with two preprocessing methods
p <- list(
  prep("savgol", width = 7, porder = 2, dorder = 2),
  prep("norm", type = "snv")
)

# build a PLS model with and without preprocessing
m1 <- pls(Xc, yc, 5, prep = p)
m2 <- pls(Xc, yc, 5)

```

```
# apply the models to test set
r1 <- predict(m1, Xt, yt)
r2 <- predict(m2, Xt, yt)

# check scores
par(mfrow = c(1, 2))
plotPredictions(m1, res = list(cal = m1$calres, test = r1), main = "With preprocessing")
plotPredictions(m2, res = list(cal = m2$calres, test = r2), main = "Without preprocessing")
```

---

pls.cal

*PLS model calibration*

---

### Description

Calibrates (builds) a PLS model for given data and parameters

### Usage

```
pls.cal(x, y, ncomp, center, scale, method = "simpls", cv = FALSE)
```

### Arguments

x	a matrix with x values (predictors)
y	a matrix with y values (responses)
ncomp	number of components to calculate
center	logical, do mean centering or not
scale	logical, do standardization or not
method	algorithm for computing PLS model (only 'simpls' is supported so far)
cv	logical, is model calibrated during cross-validation or not (or cv settings for calibration)

### Value

model an object with calibrated PLS model

---

pls.fromjson                      *Converts JSON string created in mda.tools/pls app to pls object*

---

**Description**

Converts JSON string created in mda.tools/pls app to pls object

**Usage**

pls.fromjson(str)

**Arguments**

str                      stringified JSON (from model file)

**Value**

object of [pls](#) class

---

pls.getLimitsCoordinates  
    *Compute coordinates of lines or curves with critical limits*

---

**Description**

Compute coordinates of lines or curves with critical limits

**Usage**

pls.getLimitsCoordinates(Qlim, T2lim, Zlim, nobj, ncomp, norm, log)

**Arguments**

Qlim	matrix with critical limits for orthogonal distances (X)
T2lim	matrix with critical limits for score distances (X)
Zlim	matrix with critical limits for orthogonal distances (Y)
nobj	number of objects to compute the limits for
ncomp	number of components for computing the coordinates
norm	logical, shall distance values be normalized or not
log	logical, shall log transformation be applied or not

**Value**

list with two matrices (x and y coordinates of corresponding limits)

---

pls.getpredictions      *Compute predictions for response values*

---

### Description

Compute predictions for response values

### Usage

```
pls.getpredictions(  
  x,  
  coeffs,  
  ycenter,  
  yscale,  
  ynames = NULL,  
  y.attrs = NULL,  
  objnames = NULL,  
  compnames = NULL  
)
```

### Arguments

x	matrix with predictors, already preprocessed (e.g. mean centered) and cleaned
coeffs	array with regression coefficients
ycenter	'ycenter' property of PLS model
yscale	'yscale' property of PLS model
ynames	vector with names of the responses
y.attrs	list with response attributes (e.g. from reference values if any)
objnames	vector with names of objects (rows of x)
compnames	vector with names used for components

### Value

array with predicted y-values

---

pls.getxdecomp      *Compute object with decomposition of x-values*

---

### Description

Compute object with decomposition of x-values

### Usage

```
pls.getxdecomp(  
  x,  
  xscores,  
  xloadings,  
  xeigenvals,  
  xnames = NULL,  
  x.attrs = NULL,  
  objnames = NULL,  
  compnames = NULL  
)
```

### Arguments

x	matrix with predictors, already preprocessed (e.g. mean centered) and cleaned
xscores	matrix with X-scores
xloadings	matrix with X-loadings
xeigenvals	matrix with eigenvalues for X
xnames	vector with names of the predictors
x.attrs	list with predictors attributes
objnames	vector with names of objects (rows of x)
compnames	vector with names used for components

### Value

array 'ldecomp' object for x-values

---

pls.getxscores      *Compute matrix with X-scores*

---

**Description**

Compute matrix with X-scores

**Usage**

```
pls.getxscores(x, weights, xloadings)
```

**Arguments**

x	matrix with predictors, already preprocessed and cleaned
weights	matrix with PLS weights
xloadings	matrix with X-loadings

**Value**

matrix with X-scores

---

pls.getydecomp      *Compute object with decomposition of y-values*

---

**Description**

Compute object with decomposition of y-values

**Usage**

```
pls.getydecomp(  
  y,  
  yscores,  
  xscores,  
  yloadings,  
  yeigenvals,  
  ynames = NULL,  
  y.attrs = NULL,  
  x.attrs = NULL,  
  objnames = NULL,  
  compnames = NULL  
)
```

**Arguments**

y	matrix with responses, already preprocessed (e.g. mean centered) and cleaned
yscores	matrix with Y-scores
xscores	matrix with X-scores
yloadings	matrix with Y-loadings
yeigenvals	matrix with eigenvalues for Y
yname	vector with names of the responses
y.attrs	list with response attributes (e.g. from reference values if any)
x.attrs	list with predictors attributes
objnames	vector with names of objects (rows of x)
compnames	vector with names used for components

**Value**

array 'ldecomp' object for y-values (or NULL if y is not provided)

---

pls.getyscores	<i>Compute and orthogonalize matrix with Y-scores</i>
----------------	---

---

**Description**

Compute and orthogonalize matrix with Y-scores

**Usage**

```
pls.getyscores(y, yloadings, xscores, exclrows)
```

**Arguments**

y	matrix with response values, already preprocessed and cleaned
yloadings	matrix with Y-loadings
xscores	matrix with X-scores (needed for orthogonalization)
exclrows	vector with indices of excluded rows if any

**Value**

matrix with Y-scores

---

pls.getZLimits            *Compute critical limits for orthogonal distances (Q)*

---

**Description**

Compute critical limits for orthogonal distances (Q)

**Usage**

```
pls.getZLimits(lim.type, alpha, gamma, params)
```

**Arguments**

lim.type	which method to use for calculation of critical limits for residuals
alpha	significance level for extreme limits.
gamma	significance level for outlier limits.
params	distribution parameters returned by ldecomp.getLimParams

---

pls.readJSON            *Reads PLS model from JSON file made in web-application (mda.tools/pls).*

---

**Description**

Reads PLS model from JSON file made in web-application (mda.tools/pls).

**Usage**

```
pls.readJSON(fileName)
```

**Arguments**

fileName	file name (or full path) to JSON file.
----------	--

**Value**

list with PLS model similar to what pls() creates.

---

pls.run *Runs selected PLS algorithm*

---

**Description**

Runs selected PLS algorithm

**Usage**

```
pls.run(x, y, ncomp = min(nrow(x) - 1, ncol(x)), method = "simpls", cv = FALSE)
```

**Arguments**

x	a matrix with x values (predictors from calibration set)
y	a matrix with y values (responses from calibration set)
ncomp	how many components to compute
method	algorithm for computing PLS model
cv	logical, is this for CV or not

---

pls.simpls *SIMPLS algorithm*

---

**Description**

SIMPLS algorithm for calibration of PLS model

**Usage**

```
pls.simpls(x, y, ncomp, cv = FALSE)
```

**Arguments**

x	a matrix with x values (predictors)
y	a matrix with y values (responses)
ncomp	number of components to calculate
cv	logical, is model calibrated during cross-validation or not

**Value**

a list with computed regression coefficients, loadings and scores for x and y matrices, and weights.

**References**

[1]. S. de Jong. SIMPLS: An Alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18, 1993 (251-263).

---

pls.syncResAliases      *Sync result aliases (calres, cvres, testres) from canonical res list*

---

**Description**

Sync result aliases (calres, cvres, testres) from canonical res list

**Usage**

```
pls.syncResAliases(obj)
```

**Arguments**

obj                      PLS model (object of class pls)

**Value**

model object with aliases updated

---

plsda                      *Partial Least Squares Discriminant Analysis*

---

**Description**

plsda is used to calibrate, validate and apply partial least squares discriminant analysis (PLS-DA) model.

**Usage**

```
plsda(  
  x,  
  c,  
  ncomp = min(nrow(x) - 1, ncol(x), 20),  
  center = TRUE,  
  scale = FALSE,  
  cv = NULL,  
  exclcols = NULL,  
  exclrows = NULL,  
  x.test = NULL,  
  c.test = NULL,  
  method = "simpls",  
  lim.type = "ddmoments",  
  alpha = 0.05,  
  gamma = 0.01,  
  info = "",  
  ncomp.selcrit = "min",
```

```

    classname = NULL,
    cv.scope = "local"
)

```

### Arguments

<code>x</code>	matrix with predictors.
<code>c</code>	vector with class membership (should be either a factor with class names/numbers in case of multiple classes or a vector with logical values in case of one class model).
<code>ncomp</code>	maximum number of components to calculate.
<code>center</code>	logical, center or not predictors and response values.
<code>scale</code>	logical, scale (standardize) or not predictors and response values.
<code>cv</code>	cross-validation settings (see details).
<code>exclcols</code>	columns of <code>x</code> to be excluded from calculations (numbers, names or vector with logical values)
<code>exclrows</code>	rows to be excluded from calculations (numbers, names or vector with logical values)
<code>x.test</code>	matrix with predictors for test set.
<code>c.test</code>	vector with reference class values for test set (same format as calibration values).
<code>method</code>	method for calculating PLS model.
<code>lim.type</code>	which method to use for calculation of critical limits for residual distances (see details)
<code>alpha</code>	significance level for extreme limits for T2 and Q distances.
<code>gamma</code>	significance level for outlier limits for T2 and Q distances.
<code>info</code>	short text with information about the model.
<code>ncomp.selcrit</code>	criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
<code>classname</code>	name (label) of class in case if PLS-DA is used for one-class discrimination model. In this case it is expected that parameter 'c' will be a vector with logical values.
<code>cv.scope</code>	scope for center/scale operations inside CV loop: 'global' — using globally computed mean and std or 'local' — recompute new for each local calibration set.

### Details

The `plsda` class is based on `pls` with extra functions and plots covering classification functionality. All plots for `pls` can be used. E.g. if you want to see the real predicted values (`y` in PLS) instead of classes use `plotPredictions.pls(model)` instead of `plotPredictions(model)`.

Cross-validation settings, `cv`, can be a number or a list. If `cv` is a number, it will be used as a number of segments for random cross-validation (if `cv = 1`, full cross-validation will be performed). If it is a

list, the following syntax can be used: `cv = list('rand', nseg, nrep)` for random repeated cross-validation with `nseg` segments and `nrep` repetitions or `cv = list('ven', nseg)` for systematic splits to `nseg` segments ('venetian blinds').

Calculation of confidence intervals and p-values for regression coefficients are available only by jack-knifing so far. See help for [regcoeffs](#) objects for details.

### Value

Returns an object of `plsda` class with following fields (most inherited from class `pls`):

<code>ncomp</code>	number of components included to the model.
<code>ncomp.selected</code>	selected (optimal) number of components.
<code>xloadings</code>	matrix with loading values for x decomposition.
<code>yloadings</code>	matrix with loading values for y (c) decomposition.
<code>weights</code>	matrix with PLS weights.
<code>coeffs</code>	matrix with regression coefficients calculated for each component.
<code>info</code>	information about the model, provided by user when building the model.
<code>calres</code>	an object of class <a href="#">plsdares</a> with PLS-DA results for a calibration data.
<code>testres</code>	an object of class <a href="#">plsdares</a> with PLS-DA results for a test data, if it was provided.
<code>cvres</code>	an object of class <a href="#">plsdares</a> with PLS-DA results for cross-validation, if this option was chosen.

### Author(s)

Sergey Kucheryavskiy ([svkucheryavski@gmail.com](mailto:svkucheryavski@gmail.com))

### See Also

Specific methods for `plsda` class:

<code>print.plsda</code>	prints information about a <code>plsda</code> object.
<code>summary.plsda</code>	shows performance statistics for the model.
<code>plot.plsda</code>	shows plot overview of the model.
<code>predict.plsda</code>	applies PLS-DA model to a new data.

Methods, inherited from `classmodel` class:

<code>plotPredictions.classmodel</code>	shows plot with predicted values.
<code>plotSensitivity.classmodel</code>	shows sensitivity plot.
<code>plotSpecificity.classmodel</code>	shows specificity plot.
<code>plotMisclassified.classmodel</code>	shows misclassified ratio plot.

See also methods for class [pls](#).

**Examples**

```
### Examples for PLS-DA model class

library(mdatools)

## 1. Make a PLS-DA model with full cross-validation and show model overview

# make a calibration set from iris data (3 classes)
# use names of classes as class vector
x.cal = iris[seq(1, nrow(iris), 2), 1:4]
c.cal = iris[seq(1, nrow(iris), 2), 5]

model = plsda(x.cal, c.cal, ncomp = 3, cv = 1, info = 'IRIS data example')
model = selectCompNum(model, 1)

# show summary and basic model plots
# misclassification will be shown only for first class
summary(model)
plot(model)

# summary and model plots for second class
summary(model, nc = 2)
plot(model, nc = 2)

# summary and model plot for specific class and number of components
summary(model, nc = 3, ncomp = 3)
plot(model, nc = 3, ncomp = 3)

## 2. Show performance plots for a model
par(mfrow = c(2, 2))
plotSpecificity(model)
plotSensitivity(model)
plotMisclassified(model)
plotMisclassified(model, nc = 2)
par(mfrow = c(1, 1))

## 3. Show both class and y values predictions
par(mfrow = c(2, 2))
plotPredictions(model)
plotPredictions(model, res = "cal", ncomp = 2, nc = 2)
plotPredictions(structure(model, class = "regmodel"))
plotPredictions(structure(model, class = "regmodel"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))

## 4. All plots from ordinary PLS can be used, e.g.:
par(mfrow = c(2, 2))
plotXYScores(model)
plotYVariance(model)
plotXResiduals(model)
plotRegcoeffs(model, ny = 2)
par(mfrow = c(1, 1))
```

---

plsdares

*PLS-DA results*

---

### Description

plsdares is used to store and visualize results of applying a PLS-DA model to new data.

### Usage

```
plsdares(plsres, cres)
```

### Arguments

plsres	PLS results for the data.
cres	Classification results for the data.

### Details

Do not use plsdares manually, the object is created automatically when one applies a PLS-DA model to a new data set, e.g. when calibrate and validate a PLS-DA model (all calibration and validation results in PLS-DA model are stored as objects of plsdares class) or use function [predict.plsda](#).

The object gives access to all PLS-DA results as well as to the plotting methods for visualisation of the results. The plsdares class also inherits all properties and methods of [classres](#) and [plsres](#) classes.

If no reference values provided, classification statistics will not be calculated and performance plots will not be available.

### Value

Returns an object of plsdares class with fields, inherited from [classres](#) and [plsres](#).

### See Also

Methods for plsdares objects:

<a href="#">print.plsdares</a>	shows information about the object.
<a href="#">summary.plsdares</a>	shows statistics for results of classification.
<a href="#">plot.plsdares</a>	shows plots for overview of the results.

Methods, inherited from [classres](#) class:

<a href="#">showPredictions.classres</a>	show table with predicted values.
<a href="#">plotPredictions.classres</a>	makes plot with predicted values.
<a href="#">plotSensitivity.classres</a>	makes plot with sensitivity vs. components values.

`plotSpecificity.classres` makes plot with specificity vs. components values.  
`plotPerformance.classres` makes plot with both specificity and sensitivity values.

Methods for `plsres` objects:

<code>print</code>	prints information about a <code>plsres</code> object.
<code>summary.plsres</code>	shows performance statistics for the results.
<code>plot.plsres</code>	shows plot overview of the results.
<code>plotXScores.plsres</code>	shows scores plot for x decomposition.
<code>plotXYScores.plsres</code>	shows scores plot for x and y decomposition.
<code>plotXVariance.plsres</code>	shows explained variance plot for x decomposition.
<code>plotYVariance.plsres</code>	shows explained variance plot for y decomposition.
<code>plotXCumVariance.plsres</code>	shows cumulative explained variance plot for x decomposition.
<code>plotYCumVariance.plsres</code>	shows cumulative explained variance plot for y decomposition.
<code>plotXResiduals.plsres</code>	shows T2 vs. Q plot for x decomposition.
<code>plotYResiduals.plsres</code>	shows residuals plot for y values.

Methods inherited from `regres` class (parent class for `plsres`):

<code>plotPredictions.regres</code>	shows predicted vs. measured plot.
<code>plotRMSE.regres</code>	shows RMSE plot.

See also `plsda` - a class for PLS-DA models, `predict.plsda` applying PLS-DA model for a new dataset.

## Examples

```
### Examples for PLS-DA results class

library(mdatools)

## 1. Make a PLS-DA model with full cross-validation, get
## calibration results and show overview

# make a calibration set from iris data (3 classes)
# use names of classes as class vector
x.cal = iris[seq(1, nrow(iris), 2), 1:4]
c.cal = iris[seq(1, nrow(iris), 2), 5]

model = plsda(x.cal, c.cal, ncomp = 3, cv = 1, info = 'IRIS data example')
model = selectCompNum(model, 1)

res = model$calres

# show summary and basic plots for calibration results
summary(res)
```

```
plot(res)

## 2. Apply the calibrated PLS-DA model to a new dataset

# make a new data
x.new = iris[seq(2, nrow(iris), 2), 1:4]
c.new = iris[seq(2, nrow(iris), 2), 5]

res = predict(model, x.new, c.new)
summary(res)
plot(res)

## 3. Show performance plots for the results
par(mfrow = c(2, 2))
plotSpecificity(res)
plotSensitivity(res)
plotMisclassified(res)
plotMisclassified(res, nc = 2)
par(mfrow = c(1, 1))

## 3. Show both class and y values predictions
par(mfrow = c(2, 2))
plotPredictions(res)
plotPredictions(res, ncomp = 2, nc = 2)
plotPredictions(structure(res, class = "regres"))
plotPredictions(structure(res, class = "regres"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))

## 4. All plots from ordinary PLS results can be used, e.g.:
par(mfrow = c(2, 2))
plotXYScores(res)
plotYVariance(res, type = 'h')
plotXVariance(res, type = 'h')
plotXResiduals(res)
par(mfrow = c(1, 1))
```

---

plsres

*PLS results*

---

## Description

plsres is used to store and visualize results of applying a PLS model to new data.

## Usage

```
plsres(  
  y.pred,  
  y.ref = NULL,  
  ncomp.selected = dim(y.pred)[2],
```

```

    xdecomp = NULL,
    ydecomp = NULL,
    info = ""
)

```

### Arguments

y.pred	predicted y values.
y.ref	reference (measured) y values.
ncomp.selected	selected (optimal) number of components.
xdecomp	PLS decomposition of X data (object of class ldecomp).
ydecomp	PLS decomposition of Y data (object of class ldecomp).
info	information about the object.

### Details

Do not use `plsres` manually, the object is created automatically when one applies a PLS model to a new data set, e.g. when calibrate and validate a PLS model (all calibration and validation results in PLS model are stored as objects of `plsres` class) or use function `predict.pls`.

The object gives access to all PLS results as well as to the plotting methods for visualisation of the results. The `plsres` class also inherits all properties and methods of `regres` - general class for regression results.

If no reference values provided, regression statistics will not be calculated and most of the plots not available. The class is also used for cross-validation results, in this case some of the values and methods are not available (e.g. scores and scores plot, etc.).

All plots are based on `mdaplot` function, so most of its options can be used (e.g. color grouping, etc.).

RPD is ratio of standard deviation of response values to standard error of prediction (SDy/SEP).

### Value

Returns an object of `plsres` class with following fields:

ncomp	number of components included to the model.
ncomp.selected	selected (optimal) number of components.
y.ref	a matrix with reference values for responses.
y.pred	a matrix with predicted values for responses.
rmse	a matrix with root mean squared error values for each response and component.
slope	a matrix with slope values for each response and component.
r2	a matrix with determination coefficients for each response and component.
bias	a matrix with bias values for each response and component.
sep	a matrix with standard error values for each response and component.
rpd	a matrix with RPD values for each response and component.
xdecomp	decomposition of predictors (object of class ldecomp).
ydecomp	decomposition of responses (object of class ldecomp).
info	information about the object.

**See Also**

Methods for plsres objects:

<code>print</code>	prints information about a plsres object.
<code>summary.plsres</code>	shows performance statistics for the results.
<code>plot.plsres</code>	shows plot overview of the results.
<code>plotXScores.plsres</code>	shows scores plot for x decomposition.
<code>plotXYScores.plsres</code>	shows scores plot for x and y decomposition.
<code>plotXVariance.plsres</code>	shows explained variance plot for x decomposition.
<code>plotYVariance.plsres</code>	shows explained variance plot for y decomposition.
<code>plotXCumVariance.plsres</code>	shows cumulative explained variance plot for x decomposition.
<code>plotYCumVariance.plsres</code>	shows cumulative explained variance plot for y decomposition.
<code>plotXResiduals.plsres</code>	shows T2 vs. Q plot for x decomposition.
<code>plotYResiduals.plsres</code>	shows residuals plot for y values.

Methods inherited from regres class (parent class for plsres):

<code>plotPredictions.regres</code>	shows predicted vs. measured plot.
<code>plotRMSE.regres</code>	shows RMSE plot.

See also `pls` - a class for PLS models.

**Examples**

```
### Examples of using PLS result class
library(mdatools)
## 1. Make a PLS model for concentration of first component
## using full-cross validation and get calibration results

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]

model = pls(x, y, ncomp = 8, cv = 1)
model = selectCompNum(model, 2)
res = model$calres

summary(res)
plot(res)

## 2. Make a PLS model for concentration of first component
## and apply model to a new dataset

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]

model = pls(x, y, ncomp = 6, cv = 1)
```

```

model = selectCompNum(model, 2)

x.new = simdata$spectra.t
y.new = simdata$conc.t[, 1]
res = predict(model, x.new, y.new)

summary(res)
plot(res)

## 3. Show variance and error plots for PLS results
par(mfrow = c(2, 2))
plotXCumVariance(res, type = 'h')
plotYCumVariance(res, type = 'b', show.labels = TRUE, legend.position = 'bottomright')
plotRMSE(res)
plotRMSE(res, type = 'h', show.labels = TRUE)
par(mfrow = c(1, 1))

## 4. Show scores plots for PLS results
## (for results plot we can use color grouping)
par(mfrow = c(2, 2))
plotXScores(res)
plotXScores(res, show.labels = TRUE, cgroup = y.new)
plotXYScores(res)
plotXYScores(res, comp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

## 5. Show predictions and residuals plots for PLS results
par(mfrow = c(2, 2))
plotXResiduals(res, show.label = TRUE, cgroup = y.new)
plotYResiduals(res, show.label = TRUE)
plotPredictions(res)
plotPredictions(res, ncomp = 4, xlab = 'C, reference', ylab = 'C, predictions')
par(mfrow = c(1, 1))

```

---

predict.ddsimca

*DD-SIMCA predictions*

---

### Description

Applies DD-SIMCA model to a new data set

### Usage

```

## S3 method for class 'ddsimca'
predict(
  object,
  x,
  c.ref = NULL,
  alpha = object$alpha,

```

```

    gamma = object$gamma,
    ...
  )

```

### Arguments

object	a DD-SIMCA model (object of class ddsimca)
x	a matrix with x values (predictors)
c.ref	a vector with reference class names (same as class names for models)
alpha	significance level for making the predictions.
gamma	significance level for detection of outliers.
...	other optional parameters.

### Details

See examples in help for [ddsimca](#) function.

### Value

DD-SIMCA results (an object of class ddsimcares)

---

predict.mcrals	<i>MCR ALS predictions</i>
----------------	----------------------------

---

### Description

Applies MCR-ALS model to a new set of spectra and returns matrix with contributions.

### Usage

```

## S3 method for class 'mcrals'
predict(object, x, ...)

```

### Arguments

object	an MCR model (object of class mcr).
x	spectral values (matrix or data frame).
...	other arguments.

### Value

Matrix with contributions

---

predict.mcrpure	<i>MCR predictions</i>
-----------------	------------------------

---

**Description**

Applies MCR model to a new set of spectra and returns matrix with contributions.

**Usage**

```
## S3 method for class 'mcrpure'  
predict(object, x, ...)
```

**Arguments**

object	an MCR model (object of class mcr).
x	spectral values (matrix or data frame).
...	other arguments.

**Value**

Matrix with contributions

---

predict.pca	<i>PCA predictions</i>
-------------	------------------------

---

**Description**

Applies PCA model to a new data set.

**Usage**

```
## S3 method for class 'pca'  
predict(object, x, ...)
```

**Arguments**

object	a PCA model (object of class pca).
x	data values (matrix or data frame).
...	other arguments.

**Value**

PCA results (an object of class pcares)

---

predict.pls	<i>PLS predictions</i>
-------------	------------------------

---

**Description**

Applies PLS model to a new data set

**Usage**

```
## S3 method for class 'pls'  
predict(object, x, y = NULL, cv = FALSE, ...)
```

**Arguments**

object	a PLS model (object of class <code>pls</code> )
x	a matrix with x values (predictors)
y	a matrix with reference y values (responses)
cv	logical, shall predictions be made for cross-validation procedure or not
...	other arguments

**Details**

See examples in help for `pls` function.

**Value**

PLS results (an object of class `plsres`)

---

predict.plsda	<i>PLS-DA predictions</i>
---------------	---------------------------

---

**Description**

Applies PLS-DA model to a new data set

**Usage**

```
## S3 method for class 'plsda'  
predict(object, x, c.ref = NULL, ...)
```

**Arguments**

object	a PLS-DA model (object of class <code>plsda</code> )
x	a matrix with x values (predictors)
c.ref	a vector with reference class values (should be a factor)
...	other arguments

**Details**

See examples in help for [plsda](#) function.

**Value**

PLS-DA results (an object of class `plsdares`)

---

predict.simca	<i>SIMCA predictions</i>
---------------	--------------------------

---

**Description**

Applies SIMCA model to a new data set

**Usage**

```
## S3 method for class 'simca'  
predict(object, x, c.ref = NULL, cal = FALSE, ...)
```

**Arguments**

object	a SIMCA model (object of class <code>simca</code> )
x	a matrix with x values (predictors)
c.ref	a vector with reference class names (same as class names for models)
cal	logical, are predictions for calibration set or not
...	other arguments

**Details**

See examples in help for [simca](#) function.

**Value**

SIMCA results (an object of class `simcares`)

---

predict.simcam	<i>SIMCA multiple classes predictions</i>
----------------	---

---

**Description**

Applies SIMCAM model (SIMCA for multiple classes) to a new data set

**Usage**

```
## S3 method for class 'simcam'
predict(object, x, c.ref = NULL, ...)
```

**Arguments**

object	a SIMCAM model (object of class simcam)
x	a matrix with x values (predictors)
c.ref	a vector with reference class names (same as class names in models)
...	other arguments

**Details**

See examples in help for [simcam](#) function.

**Value**

SIMCAM results (an object of class simcamres)

---

prep	<i>Class for preprocessing object/item.</i>
------	---

---

**Description**

Class for preprocessing object/item.

**Usage**

```
prep(name, ...)
```

**Arguments**

name	short text with name for the preprocessing method.
...	a list with named parameters for the method (if empty - default parameters will be used).

## Details

Use this class to create a list with a sequence of preprocessing methods to keep them together in right order and with defined parameters. The list/object can be provided as an extra argument to any modelling function (e.g. `pca`, `pls`, etc), so the optimal model parameters and the optimal preprocessing will be stored together and can be applied to a raw data by using method `predict`.

For your own preprocessing method you need to create a function, which takes matrix with values (dataset) as the first argument, does something and then return a matrix with the same dimension and same attributes as the result. The method can have any number of optional parameters.

See Bookdown tutorial for details.

---

<code>prep.alsbasecorr</code>	<i>Baseline correction using asymmetric least squares</i>
-------------------------------	---

---

## Description

Baseline correction using asymmetric least squares

## Usage

```
prep.alsbasecorr(data, plambda = 5, p = 0.1, max.niter = 10)
```

## Arguments

<code>data</code>	matrix with spectra (rows correspond to individual spectra)
<code>plambda</code>	power of the penalty parameter (e.g. if <code>plambda = 5</code> , $\lambda = 10^5$ )
<code>p</code>	asymmetry ratio (should be between 0 and 1)
<code>max.niter</code>	maximum number of iterations

## Details

The function implements baseline correction algorithm based on Whittaker smoother. The method was first shown in [1]. The function has two main parameters - power of a penalty parameter (usually varies between 2 and 9) and the ratio of asymmetry (usually between 0.1 and 0.001). The choice of the parameters depends on how broad the disturbances of the baseline are and how narrow the original spectral peaks are.

## Value

preprocessed spectra.

**Examples**

```
# take spectra from carbs dataset
data(carbs)
spectra = mda.t(carbs$S)

# apply the correction
pspectra = prep.alsbasecorr(spectra, plambda = 3, p = 0.01)

# show the original and the corrected spectra individually
par(mfrow = c(3, 1))
for (i in 1:3) {
  mdaplotg(list(
    original = mda.subset(spectra, i),
    corrected = mda.subset(pspectra, i)
  ), type = "l", col = c("black", "red"), lwd = c(2, 1), main = rownames(spectra)[i])
}
```

---

```
prep.alsbasecorr.asjson
```

*Converts preprocessing item from 'prep.alsbasecorr' method to JSON elements*

---

**Description**

Converts preprocessing item from 'prep.alsbasecorr' method to JSON elements

**Usage**

```
prep.alsbasecorr.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

```
prep.alsbasecorr.fromjson
```

*Converts JSON elements to preprocessing item for 'prep.alsbasecorr' method*

---

### Description

Converts JSON elements to preprocessing item for 'prep.alsbasecorr' method

### Usage

```
prep.alsbasecorr.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

### Arguments

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

### Value

prep object for the method

---

```
prep.apply
```

*Applies a list with preprocessing methods to a dataset*

---

### Description

Applies a list with preprocessing methods to a dataset

### Usage

```
prep.apply(obj, x)
```

### Arguments

obj	list with preprocessing methods (created using prep or prep.fit function).
x	matrix with dataset

---

```
prep.asjson
```

*Converts preprocessing model to JSON elements.*

---

**Description**

Converts preprocessing model to JSON elements.

**Usage**

```
prep.asjson(obj)
```

**Arguments**

obj                    list with preprocessing methods (created using `prep.fit` function).

**Value**

stringified JSON.

---

```
prep.autoscale
```

*Autoscale values*

---

**Description**

Autoscale (mean center and standardize) values in columns of data matrix.

**Usage**

```
prep.autoscale(data, center = TRUE, scale = FALSE, max.cov = 0)
```

**Arguments**

data                    a matrix with data values  
center                   a logical value or vector with numbers for centering  
scale                    a logical value or vector with numbers for weighting  
max.cov                  columns that have coefficient of variation (in percent) below or equal to 'max.cov' will not be scaled

**Details**

The use of 'max.cov' allows to avoid overestimation of inert variables, which vary very little. Note, that the 'max.cov' value is already in percent, e.g. if 'max.cov = 0.1' it will compare the coefficient of variation of every variable with 0.1 want to use this option simply keep 'max.cov = 0'.

**Value**

data matrix with processed values

---

prep.center	<i>Centering data columns.</i>
-------------	--------------------------------

---

**Description**

Centering data columns.

**Usage**

```
prep.center(data, type = "mean", center = NULL)
```

**Arguments**

data	a matrix with data values.
type	type of statistic to use for centering ('mean', or 'median').
center	do not use, required for training of preprocessing model.

**Value**

preprocessed data matrix

---

prep.center.asjson	<i>Converts preprocessing item from 'prep.center' method to JSON elements</i>
--------------------	---

---

**Description**

Converts preprocessing item from 'prep.center' method to JSON elements

**Usage**

```
prep.center.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

prep.center.fromjson    *Converts JSON elements to preprocessing item for 'prep.center' method*

---

### Description

Converts JSON elements to preprocessing item for 'prep.center' method

### Usage

```
prep.center.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

### Arguments

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

### Value

prep object for the method

---

prep.center.params    *Precomputes parameters for centering*

---

### Description

Precomputes parameters for centering

### Usage

```
prep.center.params(data, type = "mean", center = NULL)
```

### Arguments

data	a matrix with data values.
type	type of statistic to use for centering ('mean', or 'median').
center	vector with precomputed values for centering.

### Value

list with parameter values

---

prep.emsc	<i>Applies Extended Multiplicative Scatter Correction to data rows</i>
-----------	--

---

**Description**

Applies Extended Multiplicative Scatter Correction to data rows

**Usage**

```
prep.emsc(data, degree = 0, mspectrum = NULL, lnorm = NULL, A = NULL)
```

**Arguments**

data	a matrix with data values.
degree	polynomial degree, if 0 then the result will be the same as for conventional MSC.
mspectrum	optional reference spectrum (if not provided, mean spectrum will be used).
lnorm	do not use, required for training of preprocessing model.
A	do not use, required for training of preprocessing model.

---

prep.emsc.asjson	<i>Converts preprocessing item from 'prep.emsc' method to JSON elements</i>
------------------	---

---

**Description**

Converts preprocessing item from 'prep.emsc' method to JSON elements

**Usage**

```
prep.emsc.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

```
prep.emsc.fromjson
```

*Converts JSON elements to preprocessing item for 'prep.emsc' method*

---

**Description**

Converts JSON elements to preprocessing item for 'prep.emsc' method

**Usage**

```
prep.emsc.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

```
prep.emsc.params
```

*Precomputes parameters for EMSC*

---

**Description**

Precomputes parameters for EMSC

**Usage**

```
prep.emsc.params(data, degree = 0, mspectrum = NULL)
```

**Arguments**

data	a matrix with data values.
degree	polynomial degree.
mspectrum	reference spectrum.

**Value**

list with parameter values

`prep.fit`*Fits preprocessing model*

---

**Description**

Fits preprocessing model

**Usage**

```
prep.fit(obj, x)
```

**Arguments**

`obj` list with preprocessing methods (created using `prep` or `prep.fit` function).  
`x` matrix with training set to be used for computing data dependent parameters

**Value**

same list but with updated methods parameters computed based on the training set.

---

`prep.fromjson`*Converts JSON string to preprocessing model*

---

**Description**

Converts JSON string to preprocessing model

**Usage**

```
prep.fromjson(str)
```

**Arguments**

`str` string with JSON

**Value**

list with the methods.

---

prep.generic	<i>Generic function for preprocessing</i>
--------------	---

---

**Description**

Generic function for preprocessing

**Usage**

```
prep.generic(x, f, ...)
```

**Arguments**

x	data matrix to be preprocessed
f	function for preprocessing
...	arguments for the function f

---

prep.list	<i>Shows information about all implemented preprocessing methods.</i>
-----------	---

---

**Description**

Shows information about all implemented preprocessing methods.

**Usage**

```
prep.list()
```

---

prep.msc	<i>Multiplicative Scatter Correction transformation</i>
----------	---

---

**Description**

Applies Multiplicative Scatter Correction (MSC) transformation to data matrix (spectra)

**Usage**

```
prep.msc(data, mspectrum = NULL, ...)
```

**Arguments**

data	a matrix with data values (spectra)
mspectrum	mean spectrum (if NULL will be calculated from spectra)
...	other optional components

**Details**

MSC is used to remove scatter effects (baseline offset and slope) from spectral data, e.g. NIR spectra.

**Value**

preprocessed spectra (calculated mean spectrum is assigned as attribute 'mspectrum')

**Examples**

```
### Apply MSC to spectra from simdata

library(mdatools)
data(simdata)

spectra = simdata$spectra.c
cspectra = prep.msc(spectra)

par(mfrow = c(2, 1))
mdaplot(spectra, type = "l", main = "Before MSC")
mdaplot(cspectra, type = "l", main = "After MSC")
```

---

```
prep.norm
```

```
Normalization
```

---

**Description**

Normalizes signals (rows of data matrix).

**Usage**

```
prep.norm(data, type = "area", col.ind = NULL, ref.spectrum = NULL)
```

**Arguments**

data	a matrix with data values
type	type of normalization "area", "length", "sum", "snv", "is", or "pqn".
col.ind	indices of columns (can be either integer or logical values) for normalization to internal standard peak.
ref.spectrum	reference spectrum for PQN normalization, if not provided a mean spectrum for data is used

**Details**

The "area", "length", "sum" types do preprocessing to unit area (sum of absolute values), length or sum of all values in every row of data matrix. Type "snv" does the Standard Normal Variate normalization. Type "is" does the normalization to internal standard peak, whose position is defined by parameter 'col.ind'. If the position is a single value, the rows are normalized to the height of this peak. If 'col.ind' points to several adjacent values, the rows are normalized to the area under the peak - sum of the intensities.

The "pqn" is Probabilistic Quotient Normalization as described in [1]. In this case you also need to provide a reference spectrum (e.g. mean or median of spectra for some reference samples). If reference spectrum is not provided it will be computed as mean of the spectra to be preprocessed (parameter data).

**Value**

data matrix with normalized values

**References**

1. F. Dieterle, A. Ross, H. Senn. Probabilistic Quotient Normalization as Robust Method to Account for Dilution of Complex Biological Mixtures. Application in 1 H NMR Metabonomics. Anal. Chem. 2006, 78, 4281–4290.

---

prep.norm.asjson	<i>Converts preprocessing item from 'prep.norm' method to JSON elements</i>
------------------	---

---

**Description**

Converts preprocessing item from 'prep.norm' method to JSON elements

**Usage**

```
prep.norm.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

prep.norm.fromjson	<i>Converts JSON elements to preprocessing item for 'prep.norm' method</i>
--------------------	--

---

**Description**

Converts JSON elements to preprocessing item for 'prep.norm' method

**Usage**

```
prep.norm.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

prep.norm.params	<i>Precomputes parameters for normalization</i>
------------------	---

---

**Description**

Precomputes parameters for normalization

**Usage**

```
prep.norm.params(data, type = "area", col.ind = NULL, ref.spectrum = NULL)
```

**Arguments**

data	a matrix with data values.
type	type of normalization.
col.ind	indices of columns for IS normalization.
ref.spectrum	reference spectrum for PQN normalization.

**Value**

list with parameter values

---

`prep.ref2km`*Kubelka-Munk transformation*

---

**Description**

Applies Kubelka-Munk (km) transformation to data matrix (spectra)

**Usage**

```
prep.ref2km(data)
```

**Arguments**

`data` a matrix with spectra values (absolute reflectance values)

**Details**

Kubelka-Munk is useful preprocessing method for diffuse reflection spectra (e.g. taken for powders or rough surface). It transforms the reflectance spectra  $R$  to K/M units as follows:  $(1 - R)^2 / 2R$

**Value**

preprocessed spectra.

---

`prep.savgol`*Savitzky-Golay filter*

---

**Description**

Applies Savitzky-Golay filter to the rows of data matrix

**Usage**

```
prep.savgol(data, width = 3, porder = 1, dorder = 0, w = NULL)
```

**Arguments**

`data` a matrix with data values  
`width` width of the filter window  
`porder` order of polynomial used for smoothing  
`dorder` order of derivative to take (0 - no derivative)  
`w` do not use, required for training of preprocessing model.

**Details**

The function implements algorithm described in [1] which handles the edge points correctly and does not require to cut the spectra.

**References**

1. Peter A. Gorry. General least-squares smoothing and differentiation by the convolution (Savitzky-Golay) method. Anal. Chem. 1990, 62, 6, 570–573, <https://doi.org/10.1021/ac00205a007>.

---

prep.savgol.asjson	<i>Converts preprocessing item from 'prep.savgol' method to JSON elements</i>
--------------------	---

---

**Description**

Converts preprocessing item from 'prep.savgol' method to JSON elements

**Usage**

```
prep.savgol.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

prep.savgol.fromjson	<i>Converts JSON elements to preprocessing item for 'prep.savgol' method</i>
----------------------	--

---

**Description**

Converts JSON elements to preprocessing item for 'prep.savgol' method

**Usage**

```
prep.savgol.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

prep.savgol.params      *Precomputes parameters for Savitzky-Golay*

---

**Description**

Precomputes parameters for Savitzky-Golay

**Usage**

```
prep.savgol.params(data, width = 3, porder = 1, dorder = 0)
```

**Arguments**

data	a matrix with data values.
width	width of the filter window
porder	order of polynomial used for smoothing
dorder	order of derivative to take (0 - no derivative)

**Value**

list with parameter values

---

prep.scale	<i>Scaling data columns.</i>
------------	------------------------------

---

**Description**

Scaling data columns.

**Usage**

```
prep.scale(data, type = "sd", max.cov = 0, scale = NULL)
```

**Arguments**

data	a matrix with data values.
type	type of statistic to use for scaling ('sd', 'iqr', 'range', 'pareto')
max.cov	columns that have coefficient of variation (in percent) below or equal to 'max.cov' will not be scaled.
scale	do not use, required for training of preprocessing model.

**Value**

preprocessed data matrix

---

prep.scale.asjson	<i>Converts preprocessing item from 'prep.scale' method to JSON elements</i>
-------------------	--

---

**Description**

Converts preprocessing item from 'prep.scale' method to JSON elements

**Usage**

```
prep.scale.asjson(params, npred, left = 0, right = 1)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

```
prep.scale.fromjson
```

*Converts JSON elements to preprocessing item for 'prep.scale' method*

---

**Description**

Converts JSON elements to preprocessing item for 'prep.scale' method

**Usage**

```
prep.scale.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

```
prep.scale.params
```

*Precomputes parameters for scaling*

---

**Description**

Precomputes parameters for scaling

**Usage**

```
prep.scale.params(data, type = "sd", max.cov = 0, scale = NULL)
```

**Arguments**

data	a matrix with data values.
type	type of statistic to use for scaling ('sd', 'iqr', 'range', or 'pareto')
max.cov	columns that have coefficient of variation (in percent) below or equal to 'max.cov' will not be scaled.
scale	vector with precomputed values for scaling.

**Value**

list with parameter values

---

prep.snv

*Standard Normal Variate transformation*

---

### Description

Applies Standard Normal Variate (SNV) transformation to the rows of data matrix

### Usage

```
prep.snv(data)
```

### Arguments

data            a matrix with data values

### Details

SNV is a simple preprocessing to remove scatter effects (baseline offset and slope) from spectral data, e.g. NIR spectra.

### Value

data matrix with processed values

### Examples

```
### Apply SNV to spectra from simdata

library(mdatools)
data(simdata)

spectra = simdata$spectra.c
wavelength = simdata$wavelength

cspectra = prep.snv(spectra)

par(mfrow = c(2, 1))
mdaplot(cbind(wavelength, t(spectra)), type = 'l', main = 'Before SNV')
mdaplot(cbind(wavelength, t(cspectra)), type = 'l', main = 'After SNV')
```

---

prep.spikes	<i>Remove spikes from Raman spectra</i>
-------------	---

---

**Description**

Removes spikes from Raman spectra using median absolute deviation computed for signal differences. See [1] for more details

**Usage**

```
prep.spikes(data, width = 5, threshold = 6)
```

**Arguments**

data	a matrix with data values
width	width of the moving median filter
threshold	threshold to compare modified z-score value with to detect the spike.

**Value**

data matrix with processed values

**References**

1. Darren A. Whitaker, Kevin Hayes, A simple algorithm for despiking Raman spectra, *Chemometrics and Intelligent Laboratory Systems*, 179, 2018, pp. 82-84, 10.1016/j.chemolab.2018.06.009.

---

prep.spikes.asjson	<i>Converts preprocessing item from 'prep.spikes' method to JSON elements</i>
--------------------	---

---

**Description**

Converts preprocessing item from 'prep.spikes' method to JSON elements

**Usage**

```
prep.spikes.asjson(params, npred, left = NULL, right = NULL)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

prep.spikes.fromjson	<i>Converts JSON elements to preprocessing item for 'prep.spikes' method</i>
----------------------	--

---

**Description**

Converts JSON elements to preprocessing item for 'prep.spikes' method

**Usage**

```
prep.spikes.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

prep.transform	<i>Transformation</i>
----------------	-----------------------

---

**Description**

Transforms values using any mathematical function (e.g. log).

**Usage**

```
prep.transform(data, fun, ...)
```

**Arguments**

data	a matrix with data values
fun	reference to a transformation function, e.g. 'log' or 'function(x) x^2'.
...	optional parameters for the transformation function

**Value**

data matrix with transformed values

**Examples**

```
# generate a matrix with two columns
y <- cbind(rnorm(100, 10, 1), rnorm(100, 20, 2))

# apply log transformation
py1 = prep.transform(y, log)

# apply power transformation
py2 = prep.transform(y, function(x) x^-1.25)

# show distributions
par(mfrow = c(2, 3))
for (i in 1:2) {
  hist(y[, i], main = paste0("Original values, column #", i))
  hist(py1[, i], main = paste0("Log-transformed, column #", i))
  hist(py2[, i], main = paste0("Power-transformed, column #", i))
}
```

---

prep.varsel

*Variable selection*

---

**Description**

Returns dataset with selected variables

**Usage**

```
prep.varsel(data, var.ind)
```

**Arguments**

`data` a matrix with data values  
`var.ind` indices of variables (columns) to select, can be either numeric or logical

**Value**

data matrix with the selected variables (columns)

---

prep.varsel.asjson	<i>Converts preprocessing item from 'prep.varsel' method to JSON elements</i>
--------------------	---

---

**Description**

Converts preprocessing item from 'prep.varsel' method to JSON elements

**Usage**

```
prep.varsel.asjson(params, npred, left = NULL, right = NULL)
```

**Arguments**

params	model parameters precomputed by using prep.fit()
npred	number of predictors in original dataset
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)

**Value**

list with main elements required for JSON with preprocessing model compatible with preprocessing web-application (ml, mpl, mp, mpl\_new, mp\_new, info)

---

prep.varsel.fromjson	<i>Converts JSON elements to preprocessing item for 'prep.varsel' method</i>
----------------------	--

---

**Description**

Converts JSON elements to preprocessing item for 'prep.varsel' method

**Usage**

```
prep.varsel.fromjson(mp, mp_new, left = 0, right = 1, left.tot = 0)
```

**Arguments**

mp	model parameters from JSON (user defined)
mp_new	model parameters from JSON after training
left	index of first variable after trimming (if any)
right	index of last variable after trimming (if any)
left.tot	total, accumulated shift on the left part

**Value**

prep object for the method

---

preparePlotData	<i>Take dataset and prepare them for plot</i>
-----------------	---

---

**Description**

The function checks that ‘data’ contains correct numeric values, checks for mandatory attributes (row and column names, x- and y-axis values and names, etc.) and adds them if necessary.

Another thing is to remove hidden columns and split the rest to visible and hidden values (if excluded rows are present).

**Usage**

```
preparePlotData(data)
```

**Arguments**

data	dataset (vector, matrix or data frame)
------	--

---

prepCalData	<i>Prepares calibration data</i>
-------------	----------------------------------

---

**Description**

Prepares calibration data

**Usage**

```
prepCalData(x, exclrows = NULL, exclcols = NULL, min.nrows = 1, min.ncols = 2)
```

**Arguments**

x	matrix or data frame with values (calibration set)
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)
exclcols	columns to be excluded from calculations (numbers, names or vector with logical values)
min.nrows	smallest number of rows which must be in the dataset
min.ncols	smallest number of columns which must be in the dataset

---

print.classres	<i>Print information about classification result object</i>
----------------	---

---

### Description

Generic print function for classification results. Prints information about major fields of the object.

### Usage

```
## S3 method for class 'classres'  
print(x, str = "Classification results (class classres)\nMajor fields:", ...)
```

### Arguments

x	classification results (object of class plsdares, simcamres, etc.).
str	User specified text (e.g. to be used for particular method, like PLS-DA, etc).
...	other arguments

---

print.ddsimca	<i>Print method for DD-SIMCA model object</i>
---------------	---

---

### Description

Prints information about the object structure

### Usage

```
## S3 method for class 'ddsimca'  
print(x, ...)
```

### Arguments

x	a DD-SIMCA model (object of class ddsimca)
...	other arguments

---

print.ddsimcares      *Print method for DD-SIMCA results*

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'ddsimcares'  
print(x, ...)
```

**Arguments**

x	DD-SIMCA results (object of class ddsimcares)
...	other arguments

---

print.ipls      *Print method for iPLS*

---

**Description**

Prints information about the iPLS object structure

**Usage**

```
## S3 method for class 'ipls'  
print(x, ...)
```

**Arguments**

x	an iPLS (object of class ipls)
...	other arguments

---

print.ldecomp	<i>Print method for linear decomposition</i>
---------------	--

---

**Description**

Generic print function for linear decomposition. Prints information about the ldecomp object.

**Usage**

```
## S3 method for class 'ldecomp'  
print(x, str = NULL, ...)
```

**Arguments**

x	object of class ldecomp
str	user specified text to show as a description of the object
...	other arguments

---

print.mcrals	<i>Print method for mcrals object</i>
--------------	---------------------------------------

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'mcrals'  
print(x, ...)
```

**Arguments**

x	mcrals object
...	other arguments

---

print.mcrpure	<i>Print method for mcrpure object</i>
---------------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'mcrpure'  
print(x, ...)
```

**Arguments**

x	mcrpure object
...	other arguments

---

print.pca	<i>Print method for PCA model object</i>
-----------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'pca'  
print(x, ...)
```

**Arguments**

x	a PCA model (object of class pca)
...	other arguments

---

print.pcares	<i>Print method for PCA results object</i>
--------------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'pcares'  
print(x, ...)
```

**Arguments**

x	PCA results (object of class pcares)
...	other arguments

---

print.pls	<i>Print method for PLS model object</i>
-----------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'pls'  
print(x, ...)
```

**Arguments**

x	a PLS model (object of class pls)
...	other arguments

---

print.plsda	<i>Print method for PLS-DA model object</i>
-------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'plsda'  
print(x, ...)
```

**Arguments**

x	a PLS-DA model (object of class plsda)
...	other arguments

---

print.plsdares	<i>Print method for PLS-DA results object</i>
----------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'plsdares'  
print(x, ...)
```

**Arguments**

x	PLS-DA results (object of class plsdares)
...	other arguments

print.plsres            *print method for PLS results object*

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'plsres'  
print(x, ...)
```

**Arguments**

x	PLS results (object of class plsres)
...	other arguments

---

print.prepmodel            *Print the information about methods in the preprocessing model.*

---

**Description**

Print the information about methods in the preprocessing model.

**Usage**

```
## S3 method for class 'prepmodel'  
print(x, ...)
```

**Arguments**

x	preprocessing model (created by <a href="#">prep.fit</a> ).
...	potential further arguments (required for Method/Generic reasons).

**Value**

the x argument (invisibly).

---

print.randtest	<i>Print method for randtest object</i>
----------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'randtest'  
print(x, ...)
```

**Arguments**

x	a randomization test results (object of class randtest)
...	other arguments

---

print.regcoeffs	<i>print method for regression coefficients class</i>
-----------------	---

---

**Description**

prints regression coefficient values for given response number and amount of components

**Usage**

```
## S3 method for class 'regcoeffs'  
print(x, ...)
```

**Arguments**

x	regression coefficients object (class regcoeffs)
...	other arguments

---

print.regmodel	<i>Print method for regression model object</i>
----------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'regmodel'  
print(x, ...)
```

**Arguments**

x	a regression model (object of class regmodel)
...	other arguments

---

print.regres	<i>print method for regression results object</i>
--------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'regres'  
print(x, ...)
```

**Arguments**

x	regression results (object of class regres)
...	other arguments

---

print.simca	<i>Print method for SIMCA model object</i>
-------------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'simca'  
print(x, ...)
```

**Arguments**

x	a SIMCA model (object of class simca)
...	other arguments

---

print.simcam	<i>Print method for SIMCAM model object</i>
--------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'simcam'  
print(x, ...)
```

**Arguments**

x	a SIMCAM model (object of class simcam)
...	other arguments

---

print.simcamres	<i>Print method for SIMCAM results object</i>
-----------------	---

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'simcamres'  
print(x, ...)
```

**Arguments**

x	SIMCAM results (object of class simcamres)
...	other arguments

---

print.simcares	<i>Print method for SIMCA results object</i>
----------------	--

---

**Description**

Prints information about the object structure

**Usage**

```
## S3 method for class 'simcares'  
print(x, ...)
```

**Arguments**

x	SIMCA results (object of class simcares)
...	other arguments

---

processLimType	<i>Make correction to limit types names.</i>
----------------	--

---

**Description**

Make correction to limit types names.

**Usage**

```
processLimType(limType)
```

**Arguments**

limType	the limit type provided by user
---------	---------------------------------

---

processMembers	<i>Computes classification outcomes for target class members.</i>
----------------	---

---

**Description**

Computes classification outcomes for target class members.

**Usage**

```
processMembers(res, indMembers)
```

**Arguments**

res	a list with classification outcomes created by method <code>classify</code> (part of it will be filled by this method).
indMembers	a vector with logical values pointing on data items corresponding to class members.

**Value**

the res list where roles vector is filled for class members, plus values for TP and FN.

---

processStrangers	<i>Computes classification outcomes for members of non-target classes.</i>
------------------	--

---

**Description**

Computes classification outcomes for members of non-target classes.

**Usage**

```
processStrangers(res, indStrangers)
```

**Arguments**

res	a list with classification outcomes created by method <code>classify</code> (part of it will be filled by this method).
indStrangers	a vector with logical values pointing on data items corresponding to members of non-target classes.

**Value**

the res list where roles vector is filled for class members, values for TN and FP, as well as statistics for computing Type II error and related things (beta, s, f0, hz, Mz, Sz, k, m).

---

randtest	<i>Randomization test for PLS regression</i>
----------	--

---

**Description**

randtest is used to carry out randomization/permutation test for a PLS regression model

**Usage**

```
randtest(
  x,
  y,
  ncomp = 15,
  center = TRUE,
  scale = FALSE,
  nperm = 1000,
  sig.level = 0.05,
  silent = TRUE,
  exclcols = NULL,
  exclrows = NULL
)
```

**Arguments**

x	matrix with predictors.
y	vector or one-column matrix with response.
ncomp	maximum number of components to test.
center	logical, center or not predictors and response values.
scale	logical, scale (standardize) or not predictors and response values.
nperm	number of permutations.
sig.level	significance level.
silent	logical, show or not test progress.
exclcols	columns of x to be excluded from calculations (numbers, names or vector with logical values)
exclrows	rows to be excluded from calculations (numbers, names or vector with logical values)

**Details**

The class implements a method for selection of optimal number of components in PLS1 regression based on the randomization test [1]. The basic idea is that for each component from 1 to ncomp a statistic T, which is a covariance between t-score (X score, derived from a PLS model) and the reference Y values, is calculated. By repeating this for randomly permuted Y-values a distribution of the statistic is obtained. A parameter alpha is computed to show how often the statistic T, calculated for permuted Y-values, is the same or higher than the same statistic, calculated for original data without permutations.

If a component is important, then the covariance for unpermuted data should be larger than the covariance for permuted data and therefore the value for alpha will be quite small (there is still a small chance to get similar covariance). This makes alpha very similar to p-value in a statistical test.

The randtest procedure calculates alpha for each component, the values can be observed using summary or plot functions. There are also several functions, allowing e.g. to show distribution of statistics and the critical value for each component.

**Value**

Returns an object of randtest class with following fields:

nperm	number of permutations used for the test.
stat	statistic values calculated for each component.
alpha	alpha values calculated for each component.
statperm	matrix with statistic values for each permutation.
corrperm	matrix with correlation between predicted and reference y-values for each permutation.
ncomp.selected	suggested number of components.

## References

S. Wiklund et al. Journal of Chemometrics 21 (2007) 427-439.

## See Also

Methods for randtest objects:

<code>print.randtest</code>	prints information about a randtest object.
<code>summary.randtest</code>	shows summary statistics for the test.
<code>plot.randtest</code>	shows bar plot for alpha values.
<code>plotHist.randtest</code>	shows distribution of statistic plot.
<code>plotCorr.randtest</code>	shows determination coefficient plot.

## Examples

```
### Examples of using the test

## Get the spectral data from Simdata set and apply SNV transformation

data(simdata)

y = simdata$conc.c[, 3]
x = simdata$spectra.c
x = prep.snv(x)

## Run the test and show summary
## (normally use higher nperm values > 1000)
r = randtest(x, y, ncomp = 4, nperm = 200, silent = FALSE)
summary(r)

## Show plots

par( mfrow = c(3, 2))
plot(r)
plotHist(r, ncomp = 3)
plotHist(r, ncomp = 4)
plotCorr(r, 3)
plotCorr(r, 4)
par( mfrow = c(1, 1))
```

---

readJSON

*Reads models from JSON file made in web-application (mda.tools).*

---

## Description

Reads models from JSON file made in web-application (mda.tools).

**Usage**

```
readJSON(fileName)
```

**Arguments**

fileName            file name (or full path) to JSON file.

**Value**

object with model (prep, pls, pca, ddsimca, pls1da, pls2da).

---

regcoeffs	<i>Regression coefficients</i>
-----------	--------------------------------

---

**Description**

class for storing and visualisation of regression coefficients for regression models

**Usage**

```
regcoeffs(coeffs, ci.coeffs = NULL, use.mean = TRUE)
```

**Arguments**

coeffs            array (npred x ncomp x nresp) with regression coefficients

ci.coeffs        array (npred x ncomp x nresp x cv) with regression coefficients for computing confidence intervals (e.g. from cross-validation) using Jack-Knifing method

use.mean        logical, tells how to compute standard error for regression coefficients. If TRUE mean values for ci.coeffs is computed first. If FALSE, values (coefficients computed for global model) are used as mean.

**Value**

a list (object of regcoeffs class) with fields, including:

values	an array (nvar x ncomp x ny) with regression coefficients
se	an array (nvar x ncomp x ny) with standard errors for the coefficients
t.values	an array (nvar x ncomp x ny) with t-values for the coefficients
p.values	an array (nvar x ncomp x ny) with p-values for coefficients

last three fields are available if parameter ci.coeffs was provided.

Check also [confint.regcoeffs](#), [summary.regcoeffs](#) and [plot.regcoeffs](#).

---

regcoeffs.getStats      *Distribution statistics for regression coefficients*

---

### Description

calculates standard error, t-values and p-values for regression coefficients based on Jack-Knifing method.

### Usage

```
regcoeffs.getStats(coeffs, ci.coeffs = NULL, use.mean = TRUE)
```

### Arguments

coeffs	array (npred x ncomp x nresp) with regression coefficients
ci.coeffs	array (npred x ncomp x nresp x cv) with regression coefficients for computing confidence intervals (e.g. from cross-validation) using Jack-Knifing method
use.mean	logical, tells how to compute standard error for regression coefficients. If TRUE mean values for ci.coeffs is computed first. If FALSE, values (coefficients computed for global model) are used as mean.

### Value

a list with statistics three arrays: standard error, t-values and p-values computed for each regression coefficient.

---

regres      *Regression results*

---

### Description

Class for storing and visualisation of regression predictions

### Usage

```
regres(y.pred, y.ref = NULL, ncomp.selected = 1)
```

### Arguments

y.pred	vector or matrix with y predicted values
y.ref	vector with reference (measured) y values
ncomp.selected	if y.pred calculated for different components, which to use as default

**Value**

a list (object of regres class) with fields, including:

y.pred	a matrix with predicted values
y.ref	a vector with reference (measured) values
ncomp.selected	selected column/number of components for predictions
rmse	root mean squared error for predicted vs measured values
slope	slope for predicted vs measured values
r2	coefficient of determination for predicted vs measured values
bias	bias for predicted vs measured values
rpd	RPD values

---

regres.bias	<i>Prediction bias</i>
-------------	------------------------

---

**Description**

Calculates matrix with bias (average prediction error) for every response and components

**Usage**

```
regres.bias(err)
```

**Arguments**

err	vector with difference between reference and predicted y-values
-----	---

---

regres.err	<i>Error of prediction</i>
------------	----------------------------

---

**Description**

Calculates array of differences between predicted and reference values.

**Usage**

```
regres.err(y.pred, y.ref)
```

**Arguments**

y.pred	matrix with predicted values
y.ref	vector with reference values

---

regres.r2	<i>Determination coefficient</i>
-----------	----------------------------------

---

**Description**

Calculates matrix with coefficient of determination for every response and components

**Usage**

```
regres.r2(err, ytot)
```

**Arguments**

err	vector with difference between reference and predicted y-values
ytot	total variance for y-values

---

regres.rmse	<i>RMSE</i>
-------------	-------------

---

**Description**

Calculates matrix with root mean squared error of prediction for every response and components.

**Usage**

```
regres.rmse(err)
```

**Arguments**

err	vector with difference between reference and predicted y-values
-----	---

---

regres.slope	<i>Slope</i>
--------------	--------------

---

**Description**

Calculates matrix with slope of predicted and measured values for every response and components.

**Usage**

```
regres.slope(y.pred, y.ref)
```

**Arguments**

y.pred	matrix with predicted values
y.ref	vector with reference values

---

regress.addattrs	<i>Add names and attributes to matrix with statistics</i>
------------------	---

---

**Description**

Add names and attributes to matrix with statistics

**Usage**

```
regress.addattrs(stat, attrs, name)
```

**Arguments**

stat	matrix with statistics
attrs	attributes from error matrix
name	name of statistic

---

repmat	<i>Replicate matrix x</i>
--------	---------------------------

---

**Description**

Replicate matrix x

**Usage**

```
repmat(x, nrows, ncols = nrows)
```

**Arguments**

x	original matrix
nrows	number of times replicate matrix row wise
ncols	number of times replicate matrix columns wise

---

selectCompNum	<i>Select optimal number of components for a model</i>
---------------	--

---

**Description**

Generic function for selecting number of components for multivariate models (e.g. PCA, PLS, ...)

**Usage**

```
selectCompNum(obj, ncomp = NULL, ...)
```

**Arguments**

obj	a model object
ncomp	number of components to select
...	other arguments

---

selectCompNum.pca	<i>Select optimal number of components for PCA model</i>
-------------------	--

---

**Description**

Allows user to select optimal number of components for a PCA model

**Usage**

```
## S3 method for class 'pca'
selectCompNum(obj, ncomp, ...)
```

**Arguments**

obj	PCA model (object of class pca)
ncomp	number of components to select
...	other parameters if any

**Value**

the same model with selected number of components

---

selectCompNum.pls	Select optimal number of components for PLS model
-------------------	---

---

### Description

Allows user to select optimal number of components for PLS model

### Usage

```
## S3 method for class 'pls'  
selectCompNum(obj, ncomp = NULL, selcrit = obj$ncomp.selcrit, ...)
```

### Arguments

obj	PLS model (object of class pls)
ncomp	number of components to select
selcrit	criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
...	other parameters if any

### Details

The method sets `ncomp.selected` parameter for the model and returns it back. The parameter points out to the optimal number of components in the model. You can either specify it manually, as argument `ncomp`, or use one of the algorithms for automatic selection.

Automatic selection by default based on cross-validation statistics. If no cross-validation results are found in the model, the method will use test set validation results. If they are not available as well, the model will use calibration results and give a warning as in this case the selected number of components will lead to overfitted model.

There are two algorithms for automatic selection you can choose between: either first local minimum of RMSE ('selcrit="min"') or Wold's rule ('selcrit="wold"').

The first local minimum criterion finds at which component,  $A$ , error of prediction starts rising and selects  $(A - 1)$  as the optimal number. The Wold's criterion finds which component  $A$  does not make error smaller at least by 5 as the optimal number.

If model is PLS2 model (has several response variables) the method computes optimal number of components for each response and returns the smallest value. For example, if for the first response 2 components give the smallest error and for the second response this number is 3,  $A = 2$  will be selected as a final result.

It is not recommended to use automatic selection for real applications, always investigate your model (via RMSE, Y-variance plot, regression coefficients) to make correct decision.

See examples in help for [pls](#) function.

### Value

the same model with selected number of components

---

selratio	<i>Selectivity ratio calculation</i>
----------	--------------------------------------

---

**Description**

Calculates selectivity ratio for all response variables in the PLS model with given number of components.

**Usage**

```
selratio(obj, ncomp = obj$ncomp.selected)
```

**Arguments**

obj	a PLS model (object of class pls)
ncomp	number of components to use in the model for calculation.

**Value**

array nvar x ny with selectivity ratio values

**References**

[1] Tarja Rajalahti et al. Chemometrics and Laboratory Systems, 95 (2009), pp. 35-48.

---

setDistanceLimits	<i>Set residual distance limits</i>
-------------------	-------------------------------------

---

**Description**

Calculates and sets critical limits for residuals of PCA model

**Usage**

```
setDistanceLimits(obj, ...)
```

**Arguments**

obj	a model object
...	other parameters

---

setDistanceLimits.pca *Compute and set statistical limits for Q and T2 residual distances.*

---

### Description

Computes statistical limits for orthogonal and score distances (based on calibration set) and assign the calculated values as model properties.

### Usage

```
## S3 method for class 'pca'
setDistanceLimits(
  obj,
  lim.type = obj$lim.type,
  alpha = obj$alpha,
  gamma = obj$gamma,
  ...
)
```

### Arguments

obj	object with PCA model
lim.type	type of limits ("jm", "chisq", "ddmoments", "ddrobust")
alpha	significance level for detection of extreme objects
gamma	significance level for detection of outliers (for data driven approach)
...	other arguments

### Details

The limits can be accessed as fields of model objects: \$Qlim and \$T2lim. Each is a matrix with four rows and ncomp columns. First row contains critical limits for extremes, second row - for outliers, third row contains mean value for corresponding distance (or its robust estimate in case of lim.type = "ddrobust") and last row contains the degrees of freedom.

### Value

Model object with the two fields updated.

---

setDistanceLimits.pls *Compute and set statistical limits for residual distances.*

---

### Description

Computes statistical limits for orthogonal and score distances (x-decomposition) and orthogonal distance (y-decomposition) based on calibration set and assigns the calculated values as model properties.

### Usage

```
## S3 method for class 'pls'
setDistanceLimits(
  obj,
  lim.type = obj$lim.type,
  alpha = obj$alpha,
  gamma = obj$gamma,
  ...
)
```

### Arguments

obj	object with PLS model
lim.type	type of limits ("jm", "chisq", "ddmoments", "ddrobust")
alpha	significance level for detection of extreme objects
gamma	significance level for detection of outliers (for data driven approach)
...	other arguments

### Details

The limits can be accessed as fields of model objects: \$Qlim, \$T2lim, and \$Zlim. Each is a matrix with four rows and ncomp columns. In case of limits for x-decomposition, first row contains critical limits for extremes, second row - for outliers, third row contains mean value for corresponding distances (or its robust estimate in case of lim.type = "ddrobust") and last row contains the degrees of freedom.

### Value

Model object with the three fields updated.

---

setParams	<i>Set model parameters other than number of components (generic function)</i>
-----------	--

---

**Description**

Set model parameters other than number of components (generic function)

**Usage**

```
setParams(obj, ...)
```

**Arguments**

obj	model object, e.g. ddsimca
...	other parameters relevant for the method

---

setParams.ddsimca	<i>Set default parameters for the DD-SIMCA model.</i>
-------------------	---

---

**Description**

Set default parameters for the DD-SIMCA model.

**Usage**

```
## S3 method for class 'ddsimca'
setParams(obj, alpha = obj$alpha, gamma = obj$gamma, ...)
```

**Arguments**

obj	a DD-SIMCA model (object of class ddsimca)
alpha	significance level for making the predictions.
gamma	significance level for detection of outliers.
...	other arguments

**Value**

DD-SIMCA model with the redefined parameters.

---

showDistanceLimits	<i>Show residual distance limits</i>
--------------------	--------------------------------------

---

**Description**

Shows critical limits for residuals of PCA model

**Usage**

```
showDistanceLimits(obj, ...)
```

**Arguments**

obj	a model object
...	other parameters

---

showLabels	<i>Show labels on plot</i>
------------	----------------------------

---

**Description**

Show labels on plot

**Usage**

```
showLabels(
  ps,
  show.excluded = FALSE,
  pos = 3,
  cex = 0.65,
  col = "darkgray",
  force.x.values = NULL,
  bwd = 0.8
)
```

**Arguments**

ps	'plotseries' object
show.excluded	logical, are excluded rows also shown on the plot
pos	position of the labels relative to the data points
cex	size of the labels text
col	color of the labels text
force.x.values	vector with forced x-values (or NULL)
bwd	bar width in case of bar plot

---

showPredictions	<i>Predictions</i>
-----------------	--------------------

---

**Description**

Predictions

**Usage**

```
showPredictions(obj, ...)
```

**Arguments**

obj	a model or result object
...	other arguments

**Details**

Generic function for showing predicted values for classification or regression model or results

---

showPredictions.classres	<i>Show predicted class values</i>
--------------------------	------------------------------------

---

**Description**

Shows a table with predicted class values for classification result.

**Usage**

```
## S3 method for class 'classres'
showPredictions(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

obj	object with classification results (e.g. plsdares or simcamres).
ncomp	number of components to show the predictions for (NULL - use selected for a model).
...	other parameters

**Details**

The function prints a matrix where every column is a class and every row is an data object. The matrix has either -1 (does not belong to the class) or +1 (belongs to the class) values.

---

 simca

*SIMCA one-class classification*


---

### Description

simca is used to make SIMCA (Soft Independent Modelling of Class Analogies) model for one-class classification.

### Usage

```
simca(
  x,
  classname,
  ncomp = min(nrow(x) - 1, ncol(x) - 1, 20),
  x.test = NULL,
  c.test = NULL,
  cv = NULL,
  ...
)
```

### Arguments

x	a numerical matrix with data values.
classname	short text (up to 20 symbols) with class name.
ncomp	maximum number of components to calculate.
x.test	a numerical matrix with test data.
c.test	a vector with classes of test data objects (can be text with names of classes or logical).
cv	cross-validation settings (see details).
...	any other parameters suitable for <a href="#">pca</a> method.

### Details

SIMCA is in fact PCA model with additional functionality, so simca class inherits most of the functionality of [pca](#) class. It uses critical limits calculated for Q and T2 residuals calculated for PCA model for making classification decision.

Cross-validation settings, cv, can be a number or a list. If cv is a number, it will be used as a number of segments for random cross-validation (if cv = 1, full cross-validation will be performed). If it is a list, the following syntax can be used: cv = list('rand', nseg, nrep) for random repeated cross-validation with nseg segments and nrep repetitions or cv = list('ven', nseg) for systematic splits to nseg segments ('venetian blinds').

**Value**

Returns an object of `simca` class with following fields:

<code>classname</code>	a short text with class name.
<code>calres</code>	an object of class <code>simcares</code> with classification results for a calibration data.
<code>testres</code>	an object of class <code>simcares</code> with classification results for a test data, if it was provided.
<code>cvres</code>	an object of class <code>simcares</code> with classification results for cross-validation, if this option was chosen.

Fields, inherited from `pca` class:

<code>ncomp</code>	number of components included in the model.
<code>ncomp.selected</code>	selected (optimal) number of components.
<code>loadings</code>	matrix with loading values (nvar x ncomp).
<code>eigenvals</code>	vector with eigenvalues for all existent components.
<code>expvar</code>	vector with explained variance for each component (in percent).
<code>cumexpvar</code>	vector with cumulative explained variance for each component (in percent).
<code>T2lim</code>	statistical limit for T2 distance.
<code>Qlim</code>	statistical limit for Q residuals.
<code>info</code>	information about the model, provided by user when building the model.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**References**

S. Wold, M. Sjostrom. "SIMCA: A method for analyzing chemical data in terms of similarity and analogy" in B.R. Kowalski (ed.), *Chemometrics Theory and Application*, American Chemical Society Symposium Series 52, Wash., D.C., American Chemical Society, p. 243-282.

**See Also**

Methods for `simca` objects:

<code>print.simca</code>	shows information about the object.
<code>summary.simca</code>	shows summary statistics for the model.
<code>plot.simca</code>	makes an overview of SIMCA model with four plots.
<code>predict.simca</code>	applies SIMCA model to new data.

Methods, inherited from `classmodel` class:

<code>plotPredictions.classmodel</code>	shows plot with predicted values.
<code>plotSensitivity.classmodel</code>	shows sensitivity plot.

`plotSpecificity.classmodel` shows specificity plot.  
`plotMisclassified.classmodel` shows misclassified ratio plot.

Methods, inherited from `pca` class:

`selectCompNum.pca` set number of optimal components in the model  
`plotScores.pca` shows scores plot.  
`plotLoadings.pca` shows loadings plot.  
`plotVariance.pca` shows explained variance plot.  
`plotCumVariance.pca` shows cumulative explained variance plot.  
`plotResiduals.pca` shows Q vs. T2 residuals plot.

## Examples

```
## make a SIMCA model for Iris setosa class with full cross-validation
library(mdatools)

data = iris[, 1:4]
class = iris[, 5]

# take first 20 objects of setosa as calibration set
se = data[1:20, ]

# make SIMCA model and apply to test set
model = simca(se, "setosa", cv = 1)
model = selectCompNum(model, 1)

# show information, summary and plot overview
print(model)
summary(model)
plot(model)

# show predictions
par(mfrow = c(2, 1))
plotPredictions(model, show.labels = TRUE)
plotPredictions(model, res = "cal", ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

# show performance, modelling power and residuals for ncomp = 2
par(mfrow = c(2, 2))
plotSensitivity(model)
plotMisclassified(model)
plotLoadings(model, comp = c(1, 2), show.labels = TRUE)
plotResiduals(model, ncomp = 2)
par(mfrow = c(1, 1))
```

---

simcam	<i>SIMCA multiclass classification</i>
--------	--

---

### Description

simcam is used to combine several one-class SIMCA models for multiclass classification.

### Usage

```
simcam(models, info = "")
```

### Arguments

models	list with SIMCA models (simca objects).
info	optional text with information about the object.

### Details

Besides the possibility for multiclass classification, SIMCAM also provides tools for investigation of relationship among individual models (classes), such as discrimination power of variables, Cooman's plot, model distance, etc.

When creating a simcam object, the calibration data from all individual SIMCA models is extracted and combined for making predictions and calculating performance of the multi-class model. The results are stored in \$calres field of the model object.

### Value

Returns an object of simcam class with following fields:

models	a list with provided SIMCA models.
dispower	an array with discrimination power of variables for each pair of individual models.
moddist	a matrix with distance between each pair of individual models.
classnames	vector with names of individual classes.
nclasses	number of classes in the object.
info	information provided by user when creating the object.
calres	an object of class <a href="#">simcamres</a> with classification results for a calibration data.

### See Also

Methods for simcam objects:

print.simcam	shows information about the object.
summary.simcam	shows summary statistics for the models.
plot.simcam	makes an overview of SIMCAM model with two plots.

<code>predict.simcam</code>	applies SIMCAM model to a new data.
<code>plotModelDistance.simcam</code>	shows plot with distance between individual models.
<code>plotDiscriminationPower.simcam</code>	shows plot with discrimination power.
<code>plotCooman.simcam</code>	shows Cooman's plot for calibration data.

Methods, inherited from `classmodel` class:

<code>plotPredictions.classmodel</code>	shows plot with predicted values.
<code>plotSensitivity.classmodel</code>	shows sensitivity plot.
<code>plotSpecificity.classmodel</code>	shows specificity plot.
<code>plotMisclassified.classmodel</code>	shows misclassified ratio plot.

Since SIMCAM objects and results are calculated only for optimal number of components, there is no sense to show such plots like sensitivity or specificity vs. number of components. However they are available as for any other classification model.

## Examples

```
## make a multiclass SIMCA model for Iris data
library(mdatools)

# split data
caldata = iris[seq(1, nrow(iris), 2), 1:4]
x.se = caldata[1:25, ]
x.ve = caldata[26:50, ]
x.vi = caldata[51:75, ]

x.test = iris[seq(2, nrow(iris), 2), 1:4]
c.test = iris[seq(2, nrow(iris), 2), 5]

# create individual models
m.se = simca(x.se, classname = "setosa")
m.se = selectCompNum(m.se, 1)

m.vi = simca(x.vi, classname = "virginica")
m.vi = selectCompNum(m.vi, 2)

m.ve = simca(x.ve, classname = "versicolor")
m.ve = selectCompNum(m.ve, 1)

# combine models into SIMCAM objects, show statistics and plots
m = simcam(list(m.se, m.vi, m.ve), info = "simcam model for Iris data")
summary(m)

# show predictions and residuals for calibration data
par(mfrow = c(2, 2))
plotPredictions(m)
plotCooman(m, nc = c(1, 2))
plotModelDistance(m, nc = 1)
```

```
plotDiscriminationPower(m, nc = c(1, 2))
par(mfrow = c(1, 1))

# apply the SIMCAM model to test set and show statistics and plots
res = predict(m, x.test, c.test)
summary(res)
plotPredictions(res)
```

---

simcam.getPerformanceStats

*Performance statistics for SIMCAM model*

---

### Description

Calculates discrimination power and distance between individual SIMCA models.

### Usage

```
simcam.getPerformanceStats(models, classnames)
```

### Arguments

models	list with SIMCA models (as provided to simcam class)
classnames	names of the classes for each model

---

simcamres

*Results of SIMCA multiclass classification*

---

### Description

simcamres is used to store results for SIMCA multiclass classification.

### Usage

```
simcamres(cres, pred.res)
```

### Arguments

cres	results of classification (class classesres).
pred.res	list with prediction results from each model (pcars objects)

**Details**

Class `simcamres` inherits all properties and methods of class `classres`, plus stores values necessary to visualise prediction decisions (e.g. Cooman's plot or Residuals plot).

In contrast to `simcares` here only values for optimal (selected) number of components in each individual SIMCA models are presented.

There is no need to create a `simcamres` object manually, it is created automatically when you make a SIMCAM model (see `simcam`) or apply the model to a new data (see `predict.simcam`). The object can be used to show summary and plots for the results.

**Value**

Returns an object (list) of class `simcamres` with the same fields as `classres` plus extra fields for Q and T2 values and limits:

<code>c.pred</code>	predicted class values.
<code>c.ref</code>	reference (true) class values if provided.
<code>T2</code>	matrix with T2 values for each object and class.
<code>Q</code>	matrix with Q values for each object and class.
<code>T2lim</code>	vector with T2 statistical limits for each class.
<code>Qlim</code>	vector with Q statistical limits for each class.

The following fields are available only if reference values were provided.

<code>tp</code>	number of true positives.
<code>fp</code>	number of false positives.
<code>fn</code>	number of false negatives.
<code>specificity</code>	specificity of predictions.
<code>sensitivity</code>	sensitivity of predictions.

**See Also**

Methods for `simcamres` objects:

<code>print.simcamres</code>	shows information about the object.
<code>summary.simcamres</code>	shows statistics for results of classification.
<code>plotCooman.simcamres</code>	makes Cooman's plot.

Methods, inherited from `classres` class:

<code>showPredictions.classres</code>	show table with predicted values.
<code>plotPredictions.classres</code>	makes plot with predicted values.

Check also `simcam`.

**Examples**

```
## see examples for simcam method.
```

---

simcares	<i>Results of SIMCA one-class classification</i>
----------	--

---

**Description**

simcares is used to store results for SIMCA one-class classification.

**Usage**

```
simcares(class.res, pca.res = NULL)
```

**Arguments**

class.res	results of classification (class classres).
pca.res	results of PCA decomposition of data (class pcares).

**Details**

Class simcares inherits all properties and methods of class [pcares](#), and has additional properties and functions for representing of classification results, inherited from class [classres](#).

There is no need to create a simcares object manually, it is created automatically when build a SIMCA model (see [simca](#)) or apply the model to a new data (see [predict.simca](#)). The object can be used to show summary and plots for the results.

**Value**

Returns an object (list) of class simcares with the same fields as [pcares](#) plus extra fields, inherited from [classres](#):

c.pred	predicted class values (+1 or -1).
c.ref	reference (true) class values if provided.

The following fields are available only if reference values were provided.

tp	number of true positives.
fp	number of false positives.
fn	number of false negatives.
specificity	specificity of predictions.
sensitivity	sensitivity of predictions.

**See Also**

Methods for `simcares` objects:

```
print.simcares      shows information about the object.
summary.simcares    shows statistics for results of classification.
```

Methods, inherited from `classes` class:

```
showPredictions.classes  show table with predicted values.
plotPredictions.classes  predicted classes plot.
plotSensitivity.classes   sensitivity plot.
plotSpecificity.classes   specificity plot.
plotPerformance.classes   performance plot.
```

Methods, inherited from `ldecomp` class:

```
plotResiduals.ldecomp    makes Q2 vs. T2 residuals plot.
plotScores.ldecomp        makes scores plot.
plotVariance.ldecomp      makes explained variance plot.
plotCumVariance.ldecomp   makes cumulative explained variance plot.
```

Check also `simca` and `pcars`.

**Examples**

```
## make a SIMCA model for Iris setosa class and show results for calibration set
library(mdatools)

data = iris[, 1:4]
class = iris[, 5]

# take first 30 objects of setosa as calibration set
se = data[1:30, ]

# make SIMCA model and apply to test set
model = simca(se, 'Se')
model = selectCompNum(model, 1)

# show information and summary
print(model$calres)
summary(model$calres)

# show plots
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
plotPredictions(model$calres, show.labels = TRUE)
plotResiduals(model$calres, show.labels = TRUE)
```

```
plotPerformance(model$calres, show.labels = TRUE, legend.position = 'bottomright')
layout(1, 1, 1)

# show predictions table
showPredictions(model$calres)
```

---

simdata

*Spectral data of polyaromatic hydrocarbons mixing*

---

## Description

Simdata contains training and test set with spectra and concentration values of polyaromatic hydrocarbons mixings.

## Usage

```
data(simdata)
```

## Format

The data is a list with following fields:

\$spectra.c	a matrix (100x150) with spectral values for the training set.
\$spectra.t	a matrix (100x150) with spectral values for the test set.
\$conc.c	a matrix (100x3) with concentration of components for the training set.
\$conc.t	a matrix (100x3) with concentration of components for the test set.
\$wavelength	a vector with spectra wavelength in nm.

## Details

This is a simulated data containing UV/Vis spectra of three component (polyaromatic hydrocarbons) mixings - C1, C2 and C3. The spectral range is between 210 and 360 nm. The spectra were simulated as a linear combination of pure component spectra plus 5% of random noise. The concentration range is (in moles): C1 [0, 1], C2 [0, 0.5], C3 [0, 0.1].

There are 100 mixings in a training set and 50 mixings in a test set. The data can be used for multivariate regression examples.

---

splitExcludedData	<i>Split the excluded part of data</i>
-------------------	--

---

**Description**

Split the excluded part of data

**Usage**

```
splitExcludedData(data, type)
```

**Arguments**

data	matrix with hidden data values
type	type of plot

---

splitPlotData	<i>Split dataset to x and y values depending on plot type</i>
---------------	---

---

**Description**

Split dataset to x and y values depending on plot type

**Usage**

```
splitPlotData(data, type)
```

**Arguments**

data	matrix with data values (visible or hidden)
type	type of plot

---

summary.classres	<i>Summary statistics about classification result object</i>
------------------	--

---

**Description**

Generic summary function for classification results. Prints performance values for the results.

**Usage**

```
## S3 method for class 'classres'
summary(
  object,
  ncomp = object$ncomp.selected,
  nc = seq_len(object$nclasses),
  ...
)
```

**Arguments**

object	classification results (object of class plsdares, simcamres, etc.).
ncomp	which number of components to make the plot for (use NULL to show results for all available).
nc	vector with class numbers to show the summary for.
...	other arguments

---

summary.ddsimca	<i>Summary method for DD-SIMCA model object</i>
-----------------	---

---

**Description**

Shows performance statistics for the model.

**Usage**

```
## S3 method for class 'ddsimca'
summary(object, ncomp = object$ncomp.selected, res = object$res, ...)
```

**Arguments**

object	a DD-SIMCA model (object of class ddsimca)
ncomp	number of components to show summary for
res	list of result objects to show summary for
...	other arguments

---

summary.ddsimcares      *Summary method for DD-SIMCA results.*

---

### Description

Shows performance statistics for the DD-SIMCA results.

### Usage

```
## S3 method for class 'ddsimcares'
summary(object, ncomp = object$ncomp.selected, limType = "classic", ...)
```

### Arguments

object	DD-SIMCA results (object of class ddsimcares)
ncomp	number of components to show the summary for.
limType	limit type to use ('classic' or 'robust').
...	other arguments

---

summary.ipls      *Summary for iPLS results*

---

### Description

Shows statistics and algorithm parameters for iPLS results.

### Usage

```
## S3 method for class 'ipls'
summary(object, glob.ncomp = object$gm$ncomp.selected, ...)
```

### Arguments

object	an iPLS (object of class ipls).
glob.ncomp	number of components for global PLS model with all intervals.
...	other arguments.

### Details

The method shows information on the algorithm parameters as well as a table with selected or excluded interval. The table has the following columns: 'step' showing on which iteration an interval was selected or excluded, 'start' and 'end' show variable indices for the interval, 'nComp' is a number of components used in a model, 'RMSE' is RMSECV for the model and 'R2' is coefficient of determination for the same model.

---

summary.ldecomp	<i>Summary statistics for linear decomposition</i>
-----------------	--

---

**Description**

Generic summary function for linear decomposition. Prints statistics about the decomposition.

**Usage**

```
## S3 method for class 'ldecomp'  
summary(object, str = NULL, ...)
```

**Arguments**

object	object of class ldecomp
str	user specified text to show as a description of the object
...	other arguments

---

summary.mcrals	<i>Summary method for mcrals object</i>
----------------	---

---

**Description**

Shows some statistics (explained variance, etc) for the case.

**Usage**

```
## S3 method for class 'mcrals'  
summary(object, ...)
```

**Arguments**

object	mcrals object
...	other arguments

---

summary.mcrpure	<i>Summary method for mcrpure object</i>
-----------------	--

---

**Description**

Shows some statistics (explained variance, etc) for the case.

**Usage**

```
## S3 method for class 'mcrpure'  
summary(object, ...)
```

**Arguments**

object	mcrpure object
...	other arguments

---

summary.pca	<i>Summary method for PCA model object</i>
-------------	--

---

**Description**

Shows some statistics (explained variance, eigenvalues) for the model.

**Usage**

```
## S3 method for class 'pca'  
summary(object, ...)
```

**Arguments**

object	a PCA model (object of class pca)
...	other arguments

---

summary.pcares	<i>Summary method for PCA results object</i>
----------------	--

---

**Description**

Shows some statistics (explained variance, eigenvalues) about the results.

**Usage**

```
## S3 method for class 'pcares'
summary(object, ...)
```

**Arguments**

object	PCA results (object of class pcares)
...	other arguments

---

summary.pls	<i>Summary method for PLS model object</i>
-------------	--

---

**Description**

Shows performance statistics for the model.

**Usage**

```
## S3 method for class 'pls'
summary(
  object,
  ncomp = object$ncomp.selected,
  ny = seq_len(nrow(object$yloadings)),
  ...
)
```

**Arguments**

object	a PLS model (object of class pls)
ncomp	how many components to show.
ny	which y variables to show the summary for (can be a vector)
...	other arguments

---

summary.plsda	<i>Summary method for PLS-DA model object</i>
---------------	---

---

**Description**

Shows some statistics for the model.

**Usage**

```
## S3 method for class 'plsda'
summary(
  object,
  ncomp = object$ncomp.selected,
  nc = seq_len(object$nclasses),
  ...
)
```

**Arguments**

object	a PLS-DA model (object of class plsda)
ncomp	how many components to use (if NULL - user selected optimal value will be used)
nc	which class to show the summary for (if NULL, will be shown for all)
...	other arguments

---

summary.plsdares	<i>Summary method for PLS-DA results object</i>
------------------	---

---

**Description**

Shows performance statistics for the results.

**Usage**

```
## S3 method for class 'plsdares'
summary(object, nc = seq_len(object$nclasses), ...)
```

**Arguments**

object	PLS-DA results (object of class plsdares)
nc	which class to show the summary for (if NULL, will be shown for all)
...	other arguments

---

summary.plsres	<i>summary method for PLS results object</i>
----------------	--

---

**Description**

Shows performance statistics for the results.

**Usage**

```
## S3 method for class 'plsres'
summary(object, ny = seq_len(object$nresp), ncomp = NULL, ...)
```

**Arguments**

object	PLS results (object of class plsres)
ny	for which response variable show the summary for
ncomp	how many components to use (if NULL - user selected optimal value will be used)
...	other arguments

---

summary.prepmodel	<i>Show summary of the preprocessing model.</i>
-------------------	---

---

**Description**

Show summary of the preprocessing model.

**Usage**

```
## S3 method for class 'prepmodel'
summary(object, ...)
```

**Arguments**

object	preprocessing model (created by <a href="#">prep.fit</a> ).
...	potential further arguments (required for Method/Generic reasons).

**Value**

the object argument (invisibly).

---

summary.randtest	<i>Summary method for randtest object</i>
------------------	---

---

**Description**

Shows summary for randomization test results.

**Usage**

```
## S3 method for class 'randtest'
summary(object, ...)
```

**Arguments**

object	randomization test results (object of class randtest)
...	other arguments

---

summary.regcoeffs	<i>Summary method for regcoeffs object</i>
-------------------	--

---

**Description**

Shows estimated coefficients and statistics (if available).

**Usage**

```
## S3 method for class 'regcoeffs'
summary(object, ncomp = 1, ny = 1, alpha = 0.05, ...)
```

**Arguments**

object	object of class regcoeffs
ncomp	how many components to use
ny	which y variable to show the summary for
alpha	significance level for confidence interval (if statistics available)
...	other arguments

**Details**

Statistics are shown if Jack-Knifing was used when model is calibrated.

---

summary.regmodel	<i>Summary method for regression model object</i>
------------------	---

---

**Description**

Shows performance statistics for the model.

**Usage**

```
## S3 method for class 'regmodel'
summary(
  object,
  ncomp = object$ncomp.selected,
  ny = seq_len(object$res$cal$nresp),
  res = object$res,
  ...
)
```

**Arguments**

object	a regression model (object of class regmodel)
ncomp	number of components to show summary for
ny	which y variables to show the summary for (can be a vector)
res	list of results to show summary for
...	other arguments

---

summary.regres	<i>summary method for regression results object</i>
----------------	---

---

**Description**

Shows performance statistics for the regression results.

**Usage**

```
## S3 method for class 'regres'
summary(object, ncomp = object$ncomp.selected, ny = seq_len(object$nresp), ...)
```

**Arguments**

object	regression results (object of class regres)
ncomp	model complexity to show the summary for (if NULL - shows for all available values)
ny	for which response variable show the summary for (one value or a vector)
...	other arguments

---

`summary.simca`*Summary method for SIMCA model object*

---

**Description**

Shows performance statistics for the model.

**Usage**

```
## S3 method for class 'simca'  
summary(object, ncomp = object$ncomp.selected, res = object$res, ...)
```

**Arguments**

<code>object</code>	a SIMCA model (object of class <code>simca</code> )
<code>ncomp</code>	number of components to show summary for
<code>res</code>	list of result objects to show summary for
<code>...</code>	other arguments

---

`summary.simcam`*Summary method for SIMCAM model object*

---

**Description**

Shows performance statistics for the model.

**Usage**

```
## S3 method for class 'simcam'  
summary(object, nc = seq_len(object$nclasses), ...)
```

**Arguments**

<code>object</code>	a SIMCAM model (object of class <code>simcam</code> )
<code>nc</code>	number of class to show summary for (can be vector)
<code>...</code>	other arguments

---

summary.simcamres      *Summary method for SIMCAM results object*

---

### Description

Shows performance statistics for the results.

### Usage

```
## S3 method for class 'simcamres'  
summary(object, nc = seq_len(object$nclasses), ...)
```

### Arguments

object	SIMCAM results (object of class simcamres)
nc	number of class to show summary for (can be vector)
...	other arguments

---

summary.simcares      *Summary method for SIMCA results object*

---

### Description

Shows performance statistics for the results.

### Usage

```
## S3 method for class 'simcares'  
summary(object, ...)
```

### Arguments

object	SIMCA results (object of class simcares)
...	other arguments

---

<code>unmix.mcrpure</code>	<i>Unmix spectral data using pure variables estimated before</i>
----------------------------	--

---

**Description**

Unmix spectral data using pure variables estimated before

**Usage**

```
unmix.mcrpure(obj, x)
```

**Arguments**

<code>obj</code>	mcrpure object
<code>x</code>	matrix with spectra

**Value**

Returns a list with resolved spectra and contributions (matrices).

---

<code>vipcores</code>	<i>VIP scores for PLS model</i>
-----------------------	---------------------------------

---

**Description**

Calculates VIP (Variable Importance in Projection) scores for predictors either individual for each response variable or total for the entire model with given number of components.

**Usage**

```
vipcores(obj, ncomp = obj$ncomp.selected, type = "individual")
```

**Arguments**

<code>obj</code>	a PLS model (object of class pls)
<code>ncomp</code>	number of components to use in the model for calculation.
<code>type</code>	type of VIP scores: "individual" computes separate VIP scores for each response variable (returns $nvar \times nresp$ matrix), "combined" computes a single VIP vector by summing explained Y-variance across all responses before normalizing as described in [1] (returns $nvar \times 1$ matrix). For PLS1 models both types give identical results.

**Details**

The "individual" type computes VIP scores separately for each response variable, weighting by the Y-variance explained per response. The "combined" type follows the original formula from [1], which sums explained Y-variance across all responses to produce a single VIP value per predictor. For PLS1 models (single response), both types are equivalent.

If you want to make a plot use for example: `mdaplot(mda.t(v), type = "1")`, where `v` is a vector with computed VIP scores. Or just try [plotVIPScores.pls](#).

**Value**

matrix with VIP score values. If `type = "individual"`, dimensions are `nvar x nresp` (columns correspond to responses). If `type = "combined"`, dimensions are `nvar x 1`.

**References**

[1] Il-Gyo Chong, Chi-Hyuck Jun. *Chemometrics and Laboratory Systems*, 78 (2005), pp. 103-112.

---

 writeCSV

---

*Method to write outcomes of any result object to CSV file*


---

**Description**

Method to write outcomes of any result object to CSV file

**Usage**

```
writeCSV(res, fileName, name, sep = ",", dataFile = "", ...)
```

**Arguments**

<code>res</code>	result object ( <code>plsres</code> , <code>pcares</code> , etc.).
<code>fileName</code>	name (or full path) to CSV file to be created.
<code>name</code>	short name of the result object (e.g. "cal", "test". etc.).
<code>sep</code>	values separator (either " ," or " ; ").
<code>dataFile</code>	optional, name of the data file used to create the results.
<code>...</code>	other optional parameters

---

writeCSV.ddsimcares    *Save DD-SIMCA results to CSV file*

---

## Description

Saves DD-SIMCA results to CSV file with structure identical to the one generated by web-application at <https://mda.tools/ddsimca>

## Usage

```
## S3 method for class 'ddsimcares'
writeCSV(
  res,
  fileName,
  name = "cal",
  sep = ",",
  dataFile = "",
  ncomp = res$ncomp.selected,
  limType = "classic",
  src = "mdatools for R",
  ...
)
```

## Arguments

res	DD-SIMCA results (object of class ddsimcares)
fileName	name (or full path) to CSV file to be created.
name	short name of the result object (e.g. "cal", "test". etc.).
sep	values separator (either ",", " or ";" ).
dataFile	optional, name of the data file used to create the results.
ncomp	model complexity (number of components) to compute the classification results for.
limType	limit type to use ('classic' or 'robust')
src	optional, source of the results
...	other optional parameters

---

writeCSV.pcares	<i>Save PCA results to CSV file</i>
-----------------	-------------------------------------

---

**Description**

Saves PCA results to CSV file with structure identical to the one generated by web-application at <https://mda.tools/pca>

**Usage**

```
## S3 method for class 'pcares'
writeCSV(
  res,
  fileName,
  name,
  sep = ",",
  dataFile = "",
  classes = NULL,
  model = NULL,
  ...
)
```

**Arguments**

res	PCA results (object of class pcares)
fileName	name (or full path) to CSV file to be created.
name	short name of the result object (e.g. "cal", "test". etc.).
sep	values separator (either ",", " or ";" ).
dataFile	optional, name of the data file used to create the results.
classes	vector with class names for every object in the dataset (if any)
model	optional, if PCA model is provided, loadings and vectors for centering/scaling will be added
...	other optional parameters

---

writeCSV.plsres	<i>Save PLS results to CSV file</i>
-----------------	-------------------------------------

---

**Description**

Saves PLS results to CSV file with structure identical to the one generated by web-application at <https://mda.tools/pls>

**Usage**

```
## S3 method for class 'plsres'
writeCSV(res, fileName, name, sep = ",", dataFile = "", ...)
```

**Arguments**

res	PLS results (object of class plsres)
fileName	name (or full path) to CSV file to be created.
name	short name of the result object (e.g. "cal", "test". etc.).
sep	values separator (either "," or ";").
dataFile	optional, name of the data file used to create the results.
...	other optional parameters

---

writeJSON	<i>Save model as JSON file</i>
-----------	--------------------------------

---

**Description**

Save model as JSON file

**Usage**

```
writeJSON(obj, fileName)
```

**Arguments**

obj	model object, e.g. pca
fileName	name (or full path) to JSON file to be created.

---

writeJSON.ddsimca	<i>Saves DD-SIMCA model as JSON file compatible with web-application (<a href="https://mda.tools/ddsimca">https://mda.tools/ddsimca</a>).</i>
-------------------	---

---

**Description**

You can load created JSON file to web-app and use it for prediction.

**Usage**

```
## S3 method for class 'ddsimca'
writeJSON(obj, fileName, ...)
```

**Arguments**

obj            Object with DD-SIMCA model (from [ddsimca](#)).  
 fileName      Name or full path to JSON file to be created.  
 ...            other arguments (passed to [asjson.ddsimca](#)).

---

writeJSON.pca      *Saves PCA model as JSON file compatible with web-application  
 (<https://mda.tools/pca>).*

---

**Description**

You can load created JSON file to web-app and use it for prediction.

**Usage**

```
## S3 method for class 'pca'
writeJSON(obj, fileName)
```

**Arguments**

obj            Object with PCA model (from [pca](#)).  
 fileName      Name or full path to JSON file to be created.

---

writeJSON.pls      *Saves PLS model as JSON file compatible with web-application  
 (<https://mda.tools/pls>).*

---

**Description**

You can load created JSON file to web-app and use it for prediction.

**Usage**

```
## S3 method for class 'pls'
writeJSON(obj, fileName)
```

**Arguments**

obj            Object with PLS model (from [pca](#)).  
 fileName      Name or full path to JSON file to be created.

---

writeJSON.prepmodel    *Saves preprocessing model to JSON file which can be loaded to web-application (mda.tools/prep).*

---

**Description**

Saves preprocessing model to JSON file which can be loaded to web-application (mda.tools/prep).

**Usage**

```
## S3 method for class 'prepmodel'  
writeJSON(obj, fileName)
```

**Arguments**

obj	list with preprocessing methods (created using prep.fit function).
fileName	file name (or full path) to JSON file to save the model into.

# Index

## \* datasets

- carbs, [22](#)
  - pellets, [129](#)
  - people, [129](#)
  - simdata, [329](#)
- arr2int, [12](#)
- as.data.frame.ddsimcares, [12](#)
- as.matrix.classres, [13](#)
- as.matrix.ddsimcares, [14](#)
- as.matrix.ldecomp, [14](#)
- as.matrix.plsdares, [15](#)
- as.matrix.plsres, [15](#)
- as.matrix.regcoeffs, [16](#)
- as.matrix.regres, [16](#)
- as.matrix.simcamres, [17](#)
- as.matrix.simcares, [17](#)
- asjson, [18](#)
- asjson.ddsimca, [18](#), [347](#)
- asjson.pca, [19](#)
- asjson.pls, [19](#)
- asvector, [20](#)
- asvector.pca, [20](#)
- asvector.pls, [21](#)
- capitalize, [21](#)
- carbs, [22](#)
- categorize, [23](#)
- categorize.pca, [23](#), [241](#)
- categorize.pls, [24](#), [241](#)
- chisq.crit, [25](#)
- chisq.prob, [25](#)
- clamp.dof, [26](#)
- classify, [26](#), [303](#), [304](#)
- classify.plsda, [27](#)
- classify.simca, [28](#)
- classmodel.processRefValues, [28](#)
- classres, [29](#), [257](#), [326–328](#)
- classres.getPerformance, [30](#)
- cleanLabels, [31](#)
- confint.regcoeffs, [31](#), [307](#)
- constraint, [32](#)
- constraintAngle, [32](#)
- constraintClosure, [33](#)
- constraintNonNegativity, [33](#)
- constraintNorm, [34](#)
- constraints.list, [34](#)
- constraintUnimod, [34](#)
- create\_categories, [35](#)
- crossval, [35](#)
- crossval.getParams, [36](#)
- crossval.regmodel, [36](#)
- crossval.simca, [37](#)
- crossval.str, [37](#)
- dd.crit, [38](#)
- ddmoments.param, [38](#)
- ddrobust.param, [39](#)
- ddsimca, [18](#), [39](#), [42](#), [44](#), [118](#), [131](#), [263](#), [347](#)
- ddsimca.fromjson, [42](#)
- ddsimca.readJSON, [43](#)
- ddsimcares, [43](#)
- ellipse, [45](#)
- employ.constraint, [46](#)
- employ.prep, [46](#)
- extractArray, [47](#)
- extractBlock, [47](#)
- extractPrep, [48](#)
- extractStringArray, [48](#)
- extractValue, [49](#)
- fprintf, [49](#)
- genhash, [50](#)
- getCalibrationData, [50](#)
- getCalibrationData.pca, [51](#)
- getCalibrationData.simcam, [51](#)
- getConfidenceEllipse, [52](#)
- getConfusionMatrix, [52](#)

- getConfusionMatrix.classres, 53
- getConvexHull, 53
- getDataLabels, 54
- getImplementedConstraints, 54
- getImplementedPrepMethods, 54
- getLabelsAsIndices, 55
- getLabelsAsValues, 55
- getMainTitle, 55
- getPlotColors, 56
- getProbabilities, 56
- getProbabilities.pca, 57
- getProbabilities.simca, 57
- getPureVariables, 58
- getRegcoeffs, 58
- getRegcoeffs.regmodel, 59, 242
- getRes, 60
- getSelectedComponents, 60
- getSelectivityRatio, 61
- getSelectivityRatio.pls, 61
- getVariance.mcr, 62
- getVIPScores, 62
- getVIPScores.pls, 63
  
- hotelling.crit, 63
- hotelling.prob, 64
  
- imshow, 64
- ipls, 65, 115, 203, 242
- ipls.backward, 67
- ipls.forward, 68
  
- jm.crit, 69
- jm.prob, 69
  
- ldecomp, 44, 70, 127, 128, 328
- ldecomp.getDistances, 71
- ldecomp.getLimitsCoordinates, 71
- ldecomp.getLimParams, 72
- ldecomp.getQLimits, 72
- ldecomp.getT2Limits, 73
- ldecomp.getVariances, 73
- ldecomp.plotDistances, 74, 75
- ldecomp.plotResiduals, 75
  
- mcr, 76
- mcrals, 32, 76, 78, 115
- mcrals.cal, 81
- mcrals.fcnnls, 82
- mcrals.nnls, 83
  
- mcrals.ols, 83
- mcrpure, 78, 84, 85, 115
- mda.cbind, 87
- mda.data2im, 87
- mda.df2mat, 88, 98
- mda.exclcols, 88
- mda.exclrows, 89
- mda.getattr, 89
- mda.getexclind, 90
- mda.im2data, 90
- mda.inclcols, 91
- mda.inclrows, 91
- mda.purge, 92
- mda.purgeCols, 92
- mda.purgeRows, 92
- mda.rbind, 93
- mda.setattr, 93
- mda.setimbg, 94
- mda.show, 94
- mda.subset, 95, 98
- mda.t, 95
- mdaplot, 96, 108, 115, 142, 143, 152, 158, 166, 169, 181, 183, 187, 188, 201, 204, 214, 232, 260
- mdaplot.areColors, 99
- mdaplot.formatValues, 99
- mdaplot.getColors, 100, 209
- mdaplot.getXAxisLim, 101
- mdaplot.getXTickLabels, 101
- mdaplot.getXTicks, 102
- mdaplot.getYAxisLim, 102
- mdaplot.getYTickLabels, 103
- mdaplot.getYTicks, 103
- mdaplot.plotAxes, 104
- mdaplot.prepareColors, 105
- mdaplot.showColorbar, 105
- mdaplot.showLines, 106
- mdaplotg, 99, 106, 114, 115, 170, 180, 183, 186, 187, 202
- mdaplotg.getLegend, 109
- mdaplotg.getXLim, 109
- mdaplotg.getYLim, 110
- mdaplotg.prepareData, 111
- mdaplotg.processParam, 111
- mdaplotg.showLegend, 112
- mdaplotyy, 113
- mdatools, 115
  
- paste1, 116

- pca, [19](#), [20](#), [40](#), [41](#), [70](#), [115](#), [116](#), [122](#), [127](#), [128](#), [133](#), [153](#), [160](#), [174](#), [202](#), [216](#), [240](#), [320–322](#), [347](#)
- pca.cal, [121](#)
- pca.fromjson, [122](#)
- pca.getB, [122](#)
- pca.mvreplace, [120](#), [123](#)
- pca.nipals, [124](#)
- pca.readJSON, [119](#), [125](#)
- pca.run, [125](#)
- pca.svd, [126](#)
- pca.syncResAliases, [126](#)
- pcares, [27](#), [44](#), [70](#), [115](#), [119](#), [120](#), [127](#), [327](#), [328](#)
- pellets, [129](#)
- people, [129](#)
- pinv, [130](#)
- plot.classres, [130](#)
- plot.ddsimca, [131](#)
- plot.ddsimcares, [131](#)
- plot.ipls, [132](#)
- plot.mcr, [132](#)
- plot.pca, [133](#)
- plot.pcares, [133](#)
- plot.pls, [134](#), [241](#)
- plot.plsda, [135](#)
- plot.plsdares, [135](#)
- plot.plsres, [136](#), [258](#), [261](#)
- plot.randtest, [136](#), [306](#)
- plot.regcoeffs, [137](#), [192](#), [307](#)
- plot.regres, [138](#)
- plot.simca, [138](#)
- plot.simcam, [139](#)
- plot.simcamres, [139](#)
- plotAcceptance, [140](#)
- plotAcceptance.ddsimca, [140](#)
- plotAcceptance.ddsimcares, [140](#), [141](#)
- plotAliens, [142](#)
- plotAliens.ddsimcares, [142](#)
- plotBars, [143](#), [209](#)
- plotBiplot, [144](#)
- plotBiplot.pca, [119](#), [144](#)
- plotConfidenceEllipse, [145](#), [172](#)
- plotContributions, [146](#)
- plotContributions.mcr, [79](#), [85](#), [147](#)
- plotConvexHull, [147](#), [172](#)
- plotCooman, [148](#)
- plotCooman.simcam, [148](#), [150](#), [324](#)
- plotCooman.simcamres, [149](#), [326](#)
- plotCorr, [150](#)
- plotCorr.randtest, [151](#), [306](#)
- plotCumVariance, [151](#)
- plotCumVariance.ldecomp, [44](#), [128](#), [152](#), [328](#)
- plotCumVariance.mcr, [79](#), [85](#), [152](#)
- plotCumVariance.pca, [41](#), [119](#), [153](#), [322](#)
- plotDensity, [153](#), [209](#)
- plotDiscriminationPower, [154](#)
- plotDiscriminationPower.simcam, [154](#), [324](#)
- plotDistances, [155](#)
- plotDistances.ddsimca, [156](#)
- plotDistances.ddsimcares, [157](#), [157](#)
- plotDistances.ldecomp, [75](#), [128](#), [158](#), [193](#)
- plotDistances.pca, [119](#), [159](#), [194](#)
- plotDistDoF, [120](#), [160](#)
- plotEigenvalues, [161](#)
- plotEigenvalues.pca, [119](#), [161](#)
- plotErrorbars, [162](#), [209](#)
- plotExtreme, [163](#)
- plotExtreme.ddsimca, [163](#)
- plotExtreme.ddsimcares, [164](#)
- plotExtreme.pca, [164](#)
- plotExtremes, [165](#)
- plotExtremes.ddsimca, [163](#), [165](#)
- plotExtremes.ddsimcares, [164](#), [166](#), [166](#)
- plotExtremes.pca, [120](#), [164](#), [167](#)
- plotFoM, [168](#)
- plotFoM.ddsimcares, [168](#), [208](#)
- plotFoMs, [169](#)
- plotFoMs.ddsimcares, [169](#)
- plotHist, [170](#)
- plotHist.randtest, [171](#), [306](#)
- plotHotellingEllipse, [171](#)
- plotLines, [172](#), [209](#)
- plotLoadings, [173](#)
- plotLoadings.pca, [41](#), [119](#), [174](#), [322](#)
- plotMisclassified, [174](#)
- plotMisclassified.classmodel, [41](#), [175](#), [255](#), [322](#), [324](#)
- plotMisclassified.classres, [30](#), [175](#)
- plotModelDistance, [176](#)
- plotModelDistance.simcam, [176](#), [324](#)
- plotModellingPower, [178](#)
- plotPerformance, [179](#)
- plotPerformance.classmodel, [175](#), [179](#)

- [206, 210](#)
- [plotPerformance.classres, 30, 176, 180, 207, 211, 258, 328](#)
- [plotPointsShape, 181](#)
- [plotPredictions, 182](#)
- [plotPredictions.classmodel, 41, 182, 255, 321, 324](#)
- [plotPredictions.classres, 30, 130, 183, 257, 326, 328](#)
- [plotPredictions.regmodel, 184, 242](#)
- [plotPredictions.regres, 138, 185, 258, 261](#)
- [plotPredictions.simcam, 186](#)
- [plotPredictions.simcamres, 186](#)
- [plotProbabilities, 187](#)
- [plotProbabilities.classres, 187](#)
- [plotPurity, 188](#)
- [plotPurity.mcrpure, 85, 189](#)
- [plotPuritySpectra, 189](#)
- [plotPuritySpectra.mcrpure, 85, 190](#)
- [plotQDoF, 120, 191](#)
- [plotRegcoeffs, 191](#)
- [plotRegcoeffs.regmodel, 192](#)
- [plotRegressionLine, 192](#)
- [plotResiduals, 193, 194](#)
- [plotResiduals.ldecomp, 160, 193, 328](#)
- [plotResiduals.pca, 194, 224, 322](#)
- [plotResiduals.regres, 194, 236](#)
- [plotRMSE, 195](#)
- [plotRMSE.ipls, 195, 203](#)
- [plotRMSE.regmodel, 196, 242](#)
- [plotRMSE.regres, 197, 258, 261](#)
- [plotRMSERatio, 198](#)
- [plotRMSERatio.regmodel, 198, 242](#)
- [plotScatter, 199, 209](#)
- [plotScores, 200](#)
- [plotScores.ldecomp, 44, 128, 172, 200, 202, 328](#)
- [plotScores.pca, 41, 119, 172, 201, 322](#)
- [plotSelection, 202](#)
- [plotSelection.ipls, 132, 196, 202](#)
- [plotSelectivityArea, 203](#)
- [plotSelectivityArea.ddsimcares, 204](#)
- [plotSelectivityRatio, 205](#)
- [plotSelectivityRatio.pls, 205, 242](#)
- [plotSensitivity, 206](#)
- [plotSensitivity.classmodel, 41, 206, 255, 321, 324](#)
- [plotSensitivity.classres, 30, 207, 257, 328](#)
- [plotSensitivity.ddsimca, 207](#)
- [plotSensitivity.ddsimcares, 208, 208](#)
- [plotseries, 209](#)
- [plotSpecificity, 210](#)
- [plotSpecificity.classmodel, 41, 210, 255, 322, 324](#)
- [plotSpecificity.classres, 30, 211, 258, 328](#)
- [plotSpectra, 211](#)
- [plotSpectra.mcr, 79, 85, 212](#)
- [plotT2DoF, 120, 212](#)
- [plotVariance, 213](#)
- [plotVariance.ldecomp, 44, 128, 214, 328](#)
- [plotVariance.mcr, 79, 85, 214](#)
- [plotVariance.pca, 41, 119, 215, 322](#)
- [plotVariance.pls, 216](#)
- [plotVariance.plsres, 217](#)
- [plotVIPScores, 217](#)
- [plotVIPScores.pls, 218, 242, 343](#)
- [plotWeights, 218](#)
- [plotWeights.pls, 219, 242](#)
- [plotXCumVariance, 220](#)
- [plotXCumVariance.pls, 220, 242](#)
- [plotXCumVariance.plsres, 221, 258, 261](#)
- [plotXLoadings, 221](#)
- [plotXLoadings.pls, 222, 242](#)
- [plotXResiduals, 222](#)
- [plotXResiduals.pls, 223, 242](#)
- [plotXResiduals.plsres, 224, 258, 261](#)
- [plotXScores, 225](#)
- [plotXScores.pls, 172, 225, 242](#)
- [plotXScores.plsres, 172, 226, 258, 261](#)
- [plotXVariance, 227](#)
- [plotXVariance.pls, 227, 242](#)
- [plotXVariance.plsres, 228, 258, 261](#)
- [plotXYLoadings, 228](#)
- [plotXYLoadings.pls, 229, 242](#)
- [plotXYResiduals, 229](#)
- [plotXYResiduals.pls, 230, 242](#)
- [plotXYResiduals.plsres, 231](#)
- [plotXYScores, 232](#)
- [plotXYScores.pls, 232, 242](#)
- [plotXYScores.plsres, 233, 258, 261](#)
- [plotYCumVariance, 234](#)
- [plotYCumVariance.pls, 234, 242](#)
- [plotYCumVariance.plsres, 235, 258, 261](#)

plotYResiduals, 235  
plotYResiduals.plsres, 236, 258, 261  
plotYResiduals.regmodel, 236, 242  
plotYVariance, 237  
plotYVariance.pls, 237, 242  
plotYVariance.plsres, 238, 258, 261  
pls, 19, 21, 115, 134, 136, 217, 219, 220, 222, 226, 227, 229, 233, 234, 238, 239, 246, 255, 261, 265, 313  
pls.cal, 245  
pls.fromjson, 246  
pls.getLimitsCoordinates, 246  
pls.getpredictions, 247  
pls.getxdecomp, 248  
pls.getxscores, 249  
pls.getydecomp, 249  
pls.getyscores, 250  
pls.getZLimits, 251  
pls.readJSON, 251  
pls.run, 252  
pls.simpls, 241, 252  
pls.syncResAliases, 253  
plsda, 29, 115, 135, 175, 183, 186, 206, 210, 253, 258, 266  
plsdares, 115, 176, 181, 184, 187, 207, 211, 255, 257  
plsres, 115, 136, 217, 221, 225, 226, 228, 233, 235, 238, 242, 257, 259, 265  
predict.ddsimsca, 41, 44, 262  
predict.mcrals, 79, 263  
predict.mcrpure, 85, 264  
predict.pca, 119, 127, 264  
predict.pls, 241, 260, 265  
predict.plsda, 255, 257, 258, 265  
predict.simca, 266, 321, 327  
predict.simcam, 267, 324, 326  
prep, 40, 117, 240, 267  
prep.alsbasecorr, 115, 268  
prep.alsbasecorr.asjson, 269  
prep.alsbasecorr.fromjson, 270  
prep.apply, 270  
prep.asjson, 271  
prep.autoscale, 115, 271  
prep.center, 272  
prep.center.asjson, 272  
prep.center.fromjson, 273  
prep.center.params, 273  
prep.emsc, 274  
prep.emsc.asjson, 274  
prep.emsc.fromjson, 275  
prep.emsc.params, 275  
prep.fit, 276, 298, 337  
prep.fromjson, 276  
prep.generic, 277  
prep.list, 277  
prep.msc, 115, 277  
prep.norm, 115, 278  
prep.norm.asjson, 279  
prep.norm.fromjson, 280  
prep.norm.params, 280  
prep.ref2km, 281  
prep.savgol, 115, 281  
prep.savgol.asjson, 282  
prep.savgol.fromjson, 282  
prep.savgol.params, 283  
prep.scale, 284  
prep.scale.asjson, 284  
prep.scale.fromjson, 285  
prep.scale.params, 285  
prep.snv, 115, 286  
prep.spikes, 287  
prep.spikes.asjson, 287  
prep.spikes.fromjson, 288  
prep.transform, 288  
prep.varsels, 289  
prep.varsels.asjson, 290  
prep.varsels.fromjson, 290  
preparePlotData, 291  
prepCalData, 291  
print.classres, 292  
print.ddsimsca, 292  
print.ddsimscares, 293  
print.ipls, 293  
print.ldecomp, 294  
print.mcrals, 294  
print.mcrpure, 295  
print.pca, 295  
print.pcares, 296  
print.pls, 296  
print.plsda, 297  
print.plsdares, 297  
print.plsres, 298  
print.prepmodel, 298  
print.randtest, 299  
print.regcoeffs, 299  
print.regmodel, 300

print.regres, 300  
print.simca, 301  
print.simcam, 301  
print.simcamres, 302  
print.simcares, 302  
processLimType, 303  
processMembers, 303  
processStrangers, 304

randtest, 115, 137, 151, 171, 242, 304  
readJSON, 118, 240, 306  
regcoeffs, 240, 241, 255, 307  
regcoeffs.getStats, 308  
regres, 308  
regres.bias, 309  
regres.err, 309  
regres.r2, 310  
regres.rmse, 310  
regres.slope, 310  
regress.addattrs, 311  
repmat, 311

selectCompNum, 312  
selectCompNum.pca, 41, 119, 312, 322  
selectCompNum.pls, 240, 241, 313  
selratio, 61, 205, 241, 314  
setDistanceLimits, 314  
setDistanceLimits.pca, 119, 315  
setDistanceLimits.pls, 241, 316  
setParams, 317  
setParams.ddsimca, 317  
showDistanceLimits, 318  
showLabels, 318  
showPredictions, 319  
showPredictions.classres, 30, 257, 319, 326, 328

simca, 29, 115, 118, 175, 183, 186, 206, 210, 266, 320, 327, 328  
simcam, 51, 115, 138, 139, 175, 183, 186, 206, 210, 267, 323, 326  
simcam.getPerformanceStats, 325  
simcamres, 115, 139, 176, 181, 184, 187, 207, 211, 323, 325  
simcares, 29, 40, 115, 321, 327  
simdata, 329  
splitExcludedData, 330  
splitPlotData, 330  
summary.classres, 331  
summary.ddsimca, 331  
summary.ddsimcares, 332  
summary.ipls, 196, 203, 332  
summary.ldecomp, 333  
summary.mcrals, 333  
summary.mcrpure, 334  
summary.pca, 334  
summary.pcares, 335  
summary.pls, 241, 335  
summary.plsda, 336  
summary.plsdares, 336  
summary.plsres, 258, 261, 337  
summary.prepmodel, 337  
summary.randtest, 306, 338  
summary.regcoeffs, 307, 338  
summary.regmodel, 339  
summary.regres, 339  
summary.simca, 340  
summary.simcam, 340  
summary.simcamres, 341  
summary.simcares, 341

unmix.mcrpure, 85, 342

vipscores, 63, 218, 241, 342

writeCSV, 343  
writeCSV.ddsimcares, 344  
writeCSV.pcares, 345  
writeCSV.plsres, 345  
writeJSON, 118, 240, 346  
writeJSON.ddsimca, 346  
writeJSON.pca, 119, 347  
writeJSON.pls, 347  
writeJSON.prepmodel, 348