Package ‘OpenMORDM’

May 13, 2015

Type Package

Title A library for multiobjective robust decision making (MORDM) and visualizing high-dimensional datasets.

Version 0.1

Date 2014-08-26

Author David Hadka

Maintainer David Hadka `<dmh309@psu.edu>`

Description OpenMORDM provides support for visualizing high-dimensional datasets loaded from matrices or data.frames and a variety of file formats including CSV, XLS, XLSX, and MOEA runtime outputs (from the Borg MOEA or MOEA Framework). This visualization is contained within a web-based viewer capable of generating various 2D and 3D plots as well as performing a number of analyses. Additionally, the R functions provide the means for evaluating models under uncertainty and deep uncertainty and computing different robustness metrics.

License MIT + file LICENSE

Copyright Copyright 2014 The Pennsylvania State University. This work was supported by the National Science Foundation through the Network for Sustainable Climate Risk Management (SCRiM) under NSF cooperative agreement GEO-1240507.

Depends R (>= 3.0)

Imports
shiny,shinyRGL,rgl,scales,grid,prim,MASS,animation,sensitivity,boot,pracma,emoa,stringr,functional,dichromat,rpart

Suggests gdata

R topics documented:

`OpenMORDM-package` *A library for multiobjective robust decision making (MORDM) and visualizing high-dimensional datasets.*

Description

OpenMORDM provides support for visualizing high-dimensional datasets loaded from matrices or data.frames and a variety of file formats including CSV, XLS, XLSX, and MOEA runtime outputs (from the Borg MOEA or MOEA Framework). This visualization is contained within a web-based viewer capable of generating various 2D and 3D plots as well as performing a number of analyses. Additionally, the R functions provide the means for evaluating models under uncertainty and deep uncertainty and computing different robustness metrics.

Details

Package:	OpenMORDM
Type:	Package
Version:	0.1
Date:	2014-08-26
License:	MIT + file LICENSE
Copyright:	Copyright 2014 The Pennsylvania State University. This work was supported by the National Science Found
Depends:	R (>= 3.0)
Imports:	shiny, shinyRGL, rgl, scales, grid, prim, MASS, animation, sensitivity, boot, pracma, emoa, stringr, function
Suggests:	gdata
Built:	R 3.1.1; ; 2015-05-07 20:21:56 UTC; windows

Index:

```
OpenMORDM-package       A library for multiobjective robust decision
                        making (MORDM) and visualizing high-dimensional
                        datasets.
adjust.command          Prepends a ./ to commands on non-Windows
                        systems.
analyze.cart            Determines the vulnerabilities due to deep
                        uncertainties using Classification and
                        Regression Trees (CART).
analyze.prim            Patient rule induction method.
borg.optimize           Optimize the problem using the Borg MOEA.
borg.optimize.external
                        Optimize the problem using the Borg standalone
```

	executable (borg.exe).
borg.optimize.function	Optimize a problem using the Borg shared library (borg.dll or libborg.so).
check.robustness	Computes the robustness metric.
compute.robustness	Computes robustness under deep compute.robustness.
compute.robustness.guassian	Computes robustness under well-characterized compute.robustness (i.e., Gaussian noise).
compute.sensitivity	Standardized interface for sensitivity analysis methods.
define.problem	Define a new problem formulation.
deltafast	Fast version of Plischkes delta-moment sensitivity analysis method.
deltamim	Plischkes delta-moment sensitivity analysis method.
evaluate	Evaluates the decision variables for a given problem.
explore	A web-based tool (powered by Shiny) for exploring high-dimensional data sets.
lhsample	Generate Latin Hypercube sampled random inputs.
mordm.animate	Animates the time series in a GIF.
mordm.as.data.frame	Converts a data set into a data frame.
mordm.cbind	Adds extra colums to the end of a data set.
mordm.colorize	Returns a vector of colors to be used when plotting a data set.
mordm.correlation	Displays the correlations among pairwise factors.
mordm.differences	Identifies key similarities/differences between two sets.
mordm.evaluate.uncertainties	Computes robustness under deep compute.robustness.
mordm.extract.attributes	Extracts the common attributes from the time series data.
mordm.get.set	Returns the individual data set entry to be analyzed.
mordm.get.subset	Returns a subset of a data set.
mordm.identify	Identify and highlight points using the middle mouse button.
mordm.mark.box	Creates a marking rule from PRIM boxes.
mordm.mark.difference	Computes the difference of two markings.
mordm.mark.intersection	Computes the intersection of two markings.
mordm.mark.not	Computes the inverse of a marking.
mordm.mark.points	Creates a marking rule identifying specific points.
mordm.mark.rule	Creates a marking rule based on a function.
mordm.mark.selection	Creates a marking rule identifying the selected points.

mordm.mark.subtract	Computes the subtraction of two markings.
mordm.mark.union	Computes the union of two markings.
mordm.normalize	Normalizes the objectives.
mordm.plot	Sets the current data set and displays a 3D scatter plot.
mordm.plot.box	Display plot of PRIM boxes.
mordm.plot.markings	Display the markings in a box plot (candle stick plot).
mordm.plot.operators	Plot operator probabilities.
mordm.plot.parallel	Display a parallel plot of the current data set.
mordm.print.box	Print descriptive representation of PRIM boxes.
mordm.rbind	Adds extra rows to the end of a data set.
mordm.read	Loads the time series data output from an optimization algorithm.
mordm.read.csv	Loads a data set stored in a CSV file.
mordm.read.matrix	Loads a data set stored in a matrix or data.frame.
mordm.read.xls	Loads a data set stored in a XLS or XLSX file.
mordm.recommend	Make recommendations for analyzing the data.
mordm.sample.uncertainties	Computes robustness under deep compute.robustness.
mordm.select	Returns the subset of rows that are marked.
mordm.select.indices	Returns the row indices in the data set that are marked.
mordm.variable.sensitivities	Computes the sensitivities of the decision variables.
mordm.weight	Computes a vector of weighted preferences
nsample	Generate normally distributed random inputs.
robustness.constraints	Robustness metric based on constraint violations.
robustness.default	Default robustness metric.
robustness.distance	Robustness metric based on distance.
robustness.gap	Experimental robustness metric based on info gap.
robustness.variance	Robustness metric based on variance.
runVisDemo	Runs the exploration tool on a 5-objective problem.
sensitivity.levels	Determines number of replicates for sensitivity analysis.
usample	Generate uniformly distributed random inputs.

Author(s)

David Hadka

Maintainer: David Hadka <dmh309@psu.edu>

`adjust.command` *Prepends a ./ to commands on non-Windows systems.*

Description

Prepends a ./ to commands on non-Windows systems.

Usage

```
adjust.command(command)
```

Arguments

`command`	the R function or executable representing the problem

`analyze.cart` *Determines the vulernabilities due to deep uncertainties using Classification and Regression Trees (CART).*

Description

Determines the vulernabilities due to deep uncertainties using Classification and Regression Trees (CART).

Usage

```
analyze.cart(factors, response)
```

Arguments

`factors`	the sampled deeply uncertain parameterizations
`response`	vector of responses whose length equals the number of factors

`analyze.prim` *Patient rule induction method.*

Description

Performs the patient rule induction method (PRIM) to identify boxes in input space that correlate with data exceeding a given threshold.

Determines the vulernabilities due to deep uncertainties using the Patient Rule Induction Method (PRIM).

Usage

```
analyze.prim(factors, response, bounds = NULL, which.box = 1,
  show.plot = TRUE, ...)

analyze.prim(factors, response, bounds = NULL, which.box = 1,
  show.plot = TRUE, ...)
```

Arguments

`factors` the sampled deeply uncertain parameterizations

`response` vector of responses whose length equals the number of factors

`bounds` bounds of the sampled uncertainties

`which.box` index of the PRIM box to plot

`show.plot` if TRUE, generates a plot representing the PRIM box

`...` optional arguments passed to `prim.box`

`data` the data set

`objective` speecifies the objective index, column name, function, or marking to use

`minimize` if `TRUE`, flip the threshold direction so that smaller values are preferred

`percentages` display percentages in the printout

`expand` if `TRUE`, sets the paste option to 1 to enable expanding the boxes to fill as much space as possible

`...` optional parameters passed to prim.box

`borg.optimize` *Optimize the problem using the Borg MOEA.*

Description

Optimizes the problem. By default, this method uses the Borg MOEA, which must first be compiled into an executable or shared library on your system. If the problem references an R function, then you must have available the shared library (borg.dll or libborg.so). If the problem is an external program, then you must have available the Borg executable (borg.exe). See `borg.optimize.function` and `borg.optimize.external` for details of each method.

Usage

```
borg.optimize(problem, NFE, ...)
```

Arguments

`problem` the problem definition

`NFE` the maximum number of function evaluations

`...` optional parameters passed to the underlying methods

Details

The Borg MOEA is free and open for non-commercial users. Source code can be obtained from `http://borgmoea.org`.

borg.optimize.external

Optimize the problem using the Borg standalone executable (borg.exe).

Description

This method is used to optimize a problem defined by an external executable. The Borg MOEA communicates with the external executable using the open API standardized by the MOEA Framework (http://moeaframework.org). See section 5.2 in the user manual for details of using the API. Since borg.exe targets POSIX systems, this method is typically not available on Windows unless you are running inside Cygwin. See the Borg MOEA documentation for instructions on compiling borg.exe.

Usage

```
borg.optimize.external(problem, NFE, executable = "./borg.exe",
  output = tempfile(), output.frequency = 100, return.output = TRUE,
  verbose = TRUE)
```

Arguments

`problem`	the problem definition
`NFE`	the maximum number of function evaluations
`executable`	the path the the optimization executable
`output`	the location where the runtime output is stored
`output.frequency`	the frequency at which data is output
`return.output`	if `TRUE`, this method loads and returns the contents of the output file
`verbose`	displays additional information for debugging

Details

The Borg MOEA is free and open for non-commercial users. Source code can be obtained from http://borgmoea.org.

borg.optimize.function

Optimize a problem using the Borg shared library (borg.dll or libborg.so).

Description

This method is used to optimize a problem defined by an R function. This method uses R's foreign function interface (FFI) to pass the R function to the Borg MOEA shared library for optimization. See the Borg MOEA documentation for instructions on compiling borg.dll or libborg.so.

Usage

```
borg.optimize.function(problem, NFE, ...)
```

Arguments

`problem`	the problem definition
`NFE`	the maximum number of function evaluations
`...`	additional arguments for setting algorithm parameters

Details

The function should either return a vector containing the objectives and any constraints (e.g., `c(o1, o2, o3, c1, c2)`), or a list containing the objectives and constraints as separate elements (e.g., `list(c(o1, o2, o3), c(c1, c2))`). All objectives are minimized. Any non-zero constraint value is considered a constraint violation.

The Borg MOEA is free and open for non-commercial users. Source code can be obtained from `http://borgmoea.org`.

`check.robustness` *Computes the robustness metric.*

Description

Robustness is represented as a scalar value, where values nearer to positive infinity are considered more robust. Due to differences in how each robustness metric computes its value, you should look at relative differences in values rather than absolute differences.

Usage

```
check.robustness(output, problem, method = "default", verbose = FALSE, ...)
```

Arguments

`output`	the evaluated points
`problem`	the problem definition
`method`	the robustness metric to use (default, variance, constraints, infogap, or distance)
`verbose`	display additional information
`...`	additional arguments passed to the robustness metric

compute.robustness *Computes robustness under deep compute.robustness.*

Description

Adds Gaussian noise to the decision variables and resamples the model output. The samples are distributed across one or more different models for the problem. The result from this method should be passed to `mordm.evaluate.uncertainties` to compute the robustness metrics.

Usage

```
compute.robustness(data, nsamples, models, sd = 0, verbose = TRUE,
  satisficing.fcn = NULL, factors = NULL, custom.fcn = NULL)
```

Arguments

`data`	the data set
`nsamples`	the number of samples to generate for each point
`models`	the problem formulations created using `define.problem`
`sd`	scalar or vector specifying the standard deviation for each decision variable
`verbose`	display additional information
`satisficing.fcn`	the satisficing function for computing the two satisficing robustness metrics
`factors`	matrix of the original compute.robustness factors for use by Satisficing Type II
`custom.fcn`	custom robustness function

Details

If multiple models are provided, it is assumed that all models have the same inputs and outputs; they would only differ in the internal calculcations within the model.

compute.robustness.guassian *Computes robustness under well-characterized compute.robustness (i.e., Gaussian noise).*

Description

Adds Gaussian noise to the decision variables and resamples the model output. Then computes one or more robustness metrics.

Usage

```
compute.robustness.guassian(data, sd, nsamples, problem, method = "default",
  verbose = TRUE)
```

Arguments

`data` the data set

`sd` scalar or vector specifying the standard deviation for each decision variable

`nsamples` the number of samples to generate for each point

`problem` the problem formulation

`method` the robustness metric or a list of metrics to use (see `check.robustness` for available options)

`verbose` display additional information

Details

This method is equivalent to `mordm.compute.robustness` using a single model.

`compute.sensitivity` *Standardized interface for sensitivity analysis methods.*

Description

Attempts to standardize the use of various sensitivity analysis methods. Supports all of the methods provided by the sensitivity library except for those using metamodels.

Usage

```
compute.sensitivity(problem, objective, nsamples, method = "fast99",
  verbose = FALSE, plot = FALSE, raw = FALSE, collapse = TRUE, ...)
```

Arguments

`problem` the problem definition

`objective` the function, objective index, or objective name whose sensitivity is being computed

`nsamples` the desired number of samples

`method` string representation of the sensitivity analysis method (fast99, sobol, sobol2002, sobol2007, sobolEff, soboljansen, sobolmara, sobolroalhs, morris, prc, src, or plischke)

`verbose` if `TRUE`, print additional information

`plot` if `TRUE`, generate any output plots

`raw` if `TRUE`, return the raw model output; otherwise return the standardized output

`collapse` if `TRUE`, collapses the list representation of the variables, objectives, and constraints into a matrix representation

`...` additional options passed to the sensitivity analysis method

Details

In addition to using the same inputs for each method, the outputs are also standardized. For methods computing the first-order indices, the output contains the sensitivity indices (`Si`) and a ranking (`rank`). Methods computing total-order indices, the output contains the total sensitivity indices (`Si.total`) and the ranking (`rank.total`). Where available, the output may also contain confidence intervals (`Ci` and `Ci.total`).

`define.problem` *Define a new problem formulation.*

Description

Constructs a new problem formulation. The command can either be an R function or a command line executable. If using a command line executable, the program must follow the MOEA Framework external problem protocol, typically by using the methods in `moeaframework.h`.

Usage

```
define.problem(command, nvars, nobjs, nconstrs = 0, bounds = NULL,
  names = NULL, epsilons = NULL, maximize = NULL)
```

Arguments

`command`	the R function or executable representing the problem
`nvars`	the number of decision variables
`nobjs`	the number of objectives
`nconstrs`	the number of constraints
`bounds`	the lower and upper bounds for each decision variable
`names`	override the column names
`epsilons`	the epsilon values if using Borg to optimize the problem
`maximize`	vector indicating the columns to be maximized

Details

If using an R function, the function should return a list containing two vectors. The first vector stores the objective values and the second stores the constraint values.

`deltafast` *Fast version of Plischke's delta-moment sensitivity analysis method.*

Description

Fast version of Plischke's delta-moment sensitivity analysis method.

Usage

```
deltafast(x, y, M = 24)
```

Arguments

`x`	the factors
`y`	the response
`M`	the number of partitions

deltamim *Plischke's delta-moment sensitivity analysis method.*

Description

Plischke's delta-moment sensitivity analysis method.

Usage

```
deltamim(x, y, partition.size = min(ceiling(nrow(x)^(2/(7 + tanh((1500 -
  nrow(x))/500)))), 48), quadrature.points = 110, ks.level = 0.95,
  zero.crossing = "on", kd.estimator = "cheap", kd.width = "auto",
  complement = FALSE, plot.enabled = FALSE, plot.cols = min(ncol(x), 4),
  output.trafo = "off", kd.shape = "epanechnikov", ...)
```

Arguments

`x`	the factors
`y`	the response
`partition.size`	the number of partitions
`quadrature.points`	the number of points in the extimated PDFs
`ks.level`	critical value for the KS statistic
`zero.crossing`	detect zero crossings when `"on"`
`kd.estimator`	the kernel density estimator to use (cheap, stats, diffusion, hist)
`kd.width`	the bandwidth for kernel density estimation, or `"auto"`
`complement`	compute sensitivities for the complement
`plot.enabled`	if TRUE, generate a plot showing the PDFs
`plot.cols`	the number of factors to display in the plot
`output.trafo`	transformation applied to the responses (off, cdf, normal, interpol, cdf-tight, cdf-loose)
`kd.shape`	kernel shape (normal, triangle, epanechnikov, box, uniform, biweight, biquadratic) or a function defining the kernel shape
`...`	additional options

evaluate *Evaluates the decision variables for a given problem.*

Description

Evaluates the problem using the given decision variables, returning an object storing the variables, objectives, and constraints.

Usage

```
evaluate(set, problem)
```

Arguments

`set`	the decision variables (inputs) to the problem
`problem`	the problem definition

`explore` *A web-based tool (powered by Shiny) for exploring high-dimensional data sets.*

Description

Starts a Shiny server and launches a webpage to display the given dataset. The Shiny server will remain running even if the web browser is closed. You must interrupt the current R function call (press Ctrl-C or select Session -> Interrupt R in RStudio) to stop the server.

Usage

```
explore(filename, nvars = NULL, nobjs = NULL, nconstrs = 0,
  names = NULL, bounds = NULL, maximize = NULL, order = NULL,
  visible.variables = FALSE, plot3d.width = "600px",
  plot3d.height = "500px", welcome.panel = NULL, selection.panel = NULL,
  ignore = NULL, metadata = NULL, runShinyApp = TRUE)
```

Arguments

`filename`	the name of the file, a matrix, or a data frame
`nvars`	the number of decision variables
`nobjs`	the number of objectives
`nconstrs`	the number of constraints
`names`	see `mordm.read` for details
`bounds`	see `mordm.read` for details
`maximize`	see `mordm.read` for details
`order`	ordering of the objectives in the dropdown menus
`visible.variables`	determines if variables are visible by default
`plot3d.width`	the width of the 3D window
`plot3d.height`	the hight of the 3D window
`welcome.panel`	omordm.plotptional panel for displaying a intro message
`selection.panel`	optional panel for displaying info about the selected point
`ignore`	columns to remove from the dataset (CSV/Excel only)
`metadata`	columns to retain in a metadata attribute (CSV/Excel only)
`runShinyApp`	if TRUE, start the Shiny server with runApp(...); if false, start the server with shinyApp(...)

Details

This method currently supports loading CSV files (.csv), Excel files (.xls or .xlsx), and MOEA runtime files (any other extension). For CSV and Excel files, you may optionally specify nvars and/or nobjs. If unspecified, the method will assume every column is an objective. For MOEA runtime files, nvars and nobjs are mandatory.

For CSV and Excel files, you can optionally ignore certain columns or indicate a column is metadata. Ignored columns are completely removed from the analysis. Metadata is not shown in the visualizations, but will be saved in an attribute (`attr(data[[i]], "metadata")`). For example, if each point has an associated GIF animation, you could treat the column storing the file path as metadata, allowing you to retrieve and display the GIF.

If you are providing a custom welcome or selection panel and would like to display custom resources, use `addResourcePath` to register the directory containing the resources.

`lhsample` *Generate Latin Hypercube sampled random inputs.*

Description

Generate Latin Hypercube sampled random inputs.

Usage

```
lhsample(nsamples, problem)
```

Arguments

`nsamples`	the number of samples to generate
`problem`	the problem definition

`mordm.animate` *Animates the time series in a GIF.*

Description

Animates the 3D scatter plot and saves the results to a GIF file. Each index in `indices` specifies the entry in the time series data that is displayed at each frame. Thus, to show all entries in succession, set `indices=1:length(data)`, but to show a single entry across multiple frames (e.g., if rotating) use `indices=rep(length(data), 50)`. The `transform` function is similar to the transformation function used by `play3d`. However, whereas the transformation in `play3d` is based on the number of elapsed seconds, this method computes the transform based on the frame number. Thus, if you use `spin3d(rpm=1)`, then this method will rotate the plot once every 60 frames (i.e., treating each frame as one second).

Usage

```
mordm.animate(data, output = "animation.gif", indices = 1:length(data),
  transform = NULL, clean = TRUE, close = TRUE, loop = FALSE,
  scale = 0.1, ...)
```

Arguments

`data`	the time series data
`output`	the location where the animated GIF is saved
`indices`	a vector indicating the indices in data that are displayed in each frame
`transform`	function that returns a transformation applied to each frame (see `spin3d`), or the user matrix for a constant projection
`clean`	if `TRUE`, delete the temporary images once the GIF is created
`close`	if `TRUE`, close the RGL window when finished
`loop`	if `TRUE`, loop infinitely; otherwise play the animation once
`scale`	amount to enlarge the plotting limits
`...`	additional arguments passed to `mordm.plot`

`mordm.as.data.frame` *Converts a data set into a data frame.*

Description

When loading data into OpenMORDM, any non-numeric columns are converted into factors and represented internally as integers. This method reverses that process to get a data frame storing the original values.

Usage

```
mordm.as.data.frame(entry)
```

Arguments

`entry`	the data set to convert

`mordm.cbind` *Adds extra colums to the end of a data set.*

Description

The added columns are treated like objectives.

Usage

```
mordm.cbind(set, columns)
```

Arguments

`set`	the data set
`columns`	the extra columns

`mordm.colorize` *Returns a vector of colors to be used when plotting a data set.*

Description

The color data that is plotted depends on the options given to this function in the following order:

- colors- displays the user-defined color values
- mark- displays the user-defined marking rules, each rule with a separate color
- objectives- use color values stored in the column defined by `objective[5]`

Usage

```
mordm.colorize(set, objectives, mark = NULL, palette = NULL, n = 100,
  offset = 0, colors = NULL, clim = NULL, unmarked = "#888888FF",
  alpha = 1, crev = TRUE)
```

Arguments

`set`	the data set
`objectives`	the objectives being plotted (objectives[5] is color)
`mark`	the optional marking rule
`palette`	the color palette to use, either a function that generates the color palette or the color palette itself
`n`	the number of distinct colors to display (only used if palette is a function)
`offset`	DEPRECATED
`colors`	user-defined color values
`clim`	range (lower and upper bounds) of color values
`unmarked`	the color value used for unmarked points
`alpha`	the transparency applied to all colors
`crev`	if `TRUE`, reverse the color palette

`mordm.correlation` *Displays the correlations among pairwise factors.*

Description

Computes the pairwise correlations between the decision variables and objectives and prints the formatted results.

Usage

```
mordm.correlation(data, ht = 0.75, lt = 0.25, all = FALSE,
  objectives = FALSE)
```

Arguments

`data`	the data set to use
`ht`	the threshold for highly correlated pairs
`lt`	the threshold for uncorrelated pairs
`all`	show all correlations
`objectives`	only compute correlations between objectives

`mordm.differences` *Identifies key similarities/differences between two sets.*

Description

Computes key differences between two data sets. This method prints formatted text as well as creates a histogram plot showing key differences.

Usage

```
mordm.differences(set1, set2, scale = TRUE, decreasing = TRUE,
  splits = 20, n = NULL, all = FALSE)
```

Arguments

`set1`	the first set
`set2`	the second set
`scale`	if `TRUE`, scale the plot based on the range of the data
`decreasing`	if `TRUE`, order differences in decreasing order
`splits`	the number of bins used by the histogram method
`n`	the number of variables to plot
`all`	plot all variables

`mordm.evaluate.uncertainties` *Computes robustness under deep compute.robustness.*

Description

Computes robustness under deep compute.robustness.

Usage

```
mordm.evaluate.uncertainties(samples, satisficing.fcn = NULL,
  factors = NULL, custom.fcn = NULL)
```

Arguments

`samples`	the samples generated by `mordm.sample.uncertainties`
`satisficing.fcn`	the satisficing function for computing the two satisficing robustness metrics
`factors`	matrix of the original compute.robustness factors for use by Satisficing Type II
`custom.fcn`	custom robustness function

`mordm.extract.attributes` *Extracts the common attributes from the time series data.*

Description

Reads the common attributes associated with each entry in the time series data and returns them in a matrix.

Usage

```
mordm.extract.attributes(data)
```

Arguments

`data`	the time series data

`mordm.get.set` *Returns the individual data set entry to be analyzed.*

Description

Determines which data set will be analyzed. Almost all methods use this function to convert their `data` arguments into a single data set. The `data` argument can be either a time series or an individual data set.

Usage

```
mordm.get.set(data, index = -1)
```

Arguments

`data`	a time series or individual data set
`index`	if `data` is a time series, specifies which entry to return (default is the last entry)

`mordm.get.subset` *Returns a subset of a data set.*

Description

Returns a subset of a data set.

Usage

```
mordm.get.subset(set, columns = 1:ncol(set), rows = 1:nrow(set))
```

Arguments

`set`	the data set
`columns`	the columns to retain
`rows`	the rows to retain

`mordm.identify` *Identify and highlight points using the middle mouse button.*

Description

Enables a mouse callback for clicking points on the 3D scatter plot and identifying those points. If any secondary plots are displayed (e.g., parallel coordinates plot or marking plot), the selected point will become highlighted in those plots.

Usage

```
mordm.identify(enabled = TRUE, label = FALSE)
```

Arguments

`enabled`	if `TRUE`, enables this functionality
`label`	if `TRUE`, a label will be added to the 3D scatter plot identifying the selected point

`mordm.mark.box` *Creates a marking rule from PRIM boxes.*

Description

PRIM identifies one or more boxes. This method converts from the PRIM box representation to a marking.

Usage

```
mordm.mark.box(box, mean, mass)
```

Arguments

`box`	the box generated by `analyze.prim`
`mean`	the mean of the box
`mass`	the mass of the box

`mordm.mark.difference` *Computes the difference of two markings.*

Description

Markings behave like sets. A point is contained within the difference if it is contained in exactly one individual marking. This is similar to the exclusive-or operator.

Usage

```
mordm.mark.difference(...)
```

Arguments

`...`	the markings

`mordm.mark.intersection` *Computes the intersection of two markings.*

Description

Markings behave like sets. A point is contained within the intersection if it is contained in all individual markings.

Usage

```
mordm.mark.intersection(...)
```

Arguments

`...`	the markings

mordm.mark.not *Computes the inverse of a marking.*

Description

Markings behave like sets. A point is contained within the inverse if it is not contained within the original marking.

Usage

```
mordm.mark.not(rule)
```

Arguments

rule	the original marking

mordm.mark.points *Creates a marking rule identifying specific points.*

Description

Markings allow the user to highlight specific subsets of the data set. These marked sets can subsequently be plotted or used in supported calculations.

Usage

```
mordm.mark.points(points)
```

Arguments

points	one or more rows from the data set considered within the marking

mordm.mark.rule *Creates a marking rule based on a function.*

Description

Markings allow the user to highlight specific subsets of the data set. These marked sets can subsequently be plotted or used in supported calculations.

Usage

```
mordm.mark.rule(condition)
```

Arguments

condition	a function of the form `f:x -> boolean`, where `x` is a single row from the data set, returning `TRUE` if the row is part of the marking

`mordm.mark.selection` *Creates a marking rule identifying the selected points.*

Description

Allows the user to select a rectangular region in the 3D scatter plot and returns a marking containing all points within the selected region.

Usage

```
mordm.mark.selection()
```

`mordm.mark.subtract` *Computes the subtraction of two markings.*

Description

Markings behave like sets. A point is contained within the subtraction if it is contained in `rule1` but not `rule2`.

Usage

```
mordm.mark.subtract(rule1, rule2)
```

Arguments

`rule1`	the first marking
`rule2`	the second marking

`mordm.mark.union` *Computes the union of two markings.*

Description

Markings behave like sets. A point is contained in the union if it is contained within any individual marking.

Usage

```
mordm.mark.union(...)
```

Arguments

`...`	the markings

mordm.normalize *Normalizes the objectives.*

Description

By default, the objectives are all minimized. Maximized objectives are negated. This function negates the maximized objectives, returning them to their original, positive form.

Usage

```
mordm.normalize(data, maximize)
```

Arguments

`data`	the data set
`maximize`	vector indicating the columns to be maximized

mordm.plot *Sets the current data set and displays a 3D scatter plot.*

Description

Creates a 3D scatter plot of the data. As a side effect, this method sets a global variable identifying the current data set and plotting attributes to be used by other plotting methods in this package. This design allows you to easily create secondary plots that are consistent with the primary 3D scatter plot.

Usage

```
mordm.plot(data, mark = NULL, index = -1, objectives = NULL,
  stay = TRUE, identify = TRUE, colors = NULL, clim = NULL,
  ideal = FALSE, selection = NULL, selection.enlarge = FALSE,
  xlim = NULL, ylim = NULL, zlim = NULL, slim = NULL, window = NULL,
  alpha = 1, tick.size = 1, label.size = 1.2, label.line = 1,
  radius.scale = 1, bg = "white", fg = "black", exploring = FALSE, ...)
```

Arguments

`data`	the data set to be displayed (if data is a time series, then the last entry in the time series is displayed)
`mark`	a list of the markings to be displayed
`index`	if data is a time series, controls which entries to display (see `mordm.get.set` for details)
`objectives`	vector specifying the objectives to be plotted on the x, y, z, size, and color axes
`stay`	forces the 3D scatter plot to stay on top of other windows
`identify`	if `TRUE`, clicking a point with the middle mouse button will identify and highlight that point

colors override the color values

clim range (lower and upper bounds) of color values

ideal draw a visual indicator of where the ideal point is on the plot

selection draw a visual indicator on the given row indices

selection.enlarge
if TRUE, enlarges the selected point; otherwise renders a transparent cube around the selected point

xlim range (lower and upper bounds) for the x axis

ylim range (lower and upper bounds) for the y axis

zlim range (lower and upper bounds) for the z axis

slim range (lower and upper bounds) of size values

window the window size (w, h)

alpha vector of transparency values applied to each point

tick.size the size of the tick labels

label.size the size of axis labels

label.line the offset of the labels

radius.scale scaling factor applied to the size of the points

bg background color

fg foreground color

exploring set to TRUE if being called from explore(...) method to properly open null devices

... additional options passed to plot3d

mordm.plot.box *Display plot of PRIM boxes.*

Description

Generates a plot showing the bounds of the PRIM boxes. Currently only works well with one or two PRIM boxes.

Usage

```
mordm.plot.box(data, mark, main = "PRIM Box", scale.width = TRUE,
  bar.width = 3, col = NULL, names = NULL, legend = TRUE,
  defaults = NULL)
```

Arguments

data the original data set

mark a list of the PRIM boxes to display

main the plot title

scale.width if TRUE, reduce the width of the bars as more PRIM boxes are displayed

bar.width the width of the bars

col vector of bar colors

names names of each PRIM box to display in the legend

legend if TRUE, renders a legend on the plot

defaults draw horizontal lines to show default values

`mordm.plot.markings` *Display the markings in a box plot (candle stick plot).*

Description

Creates a box plot showing the range (lower and upper bounds) encompassed by each marking.

Usage

```
mordm.plot.markings(highlight = NULL)
```

Arguments

`highlight`	highlight vector of row indices to be highlighted in the plot

`mordm.plot.operators` *Plot operator probabilities.*

Description

Creates a plot of the Borg MOEA operator probabilities across an entire time series.

Usage

```
mordm.plot.operators(data, time = FALSE, improvements = FALSE,
  log = FALSE, improvement.nfe = 1000, current = NULL)
```

Arguments

`data`	the time series data from a Borg MOEA run
`time`	if `TRUE`, the x-axis displays elapsed time; otherwise the x-axis represents the number of function evaluations (NFE)
`improvements`	if `TRUE`, displays a trace of the number of Pareto improvements
`log`	if `TRUE`, plot the x-axis with a log scale
`improvement.nfe`	the window size in NFE when computing the number of improvements
`current`	draw a line at the current time

`mordm.plot.parallel` *Display a parallel plot of the current data set.*

Description

Creates a parallel axis or parallel coordinates plot of the current data set. All display attributes are taken from the current plotting options.

Usage

```
mordm.plot.parallel(highlight = NULL, alpha = 0.4, label.size = 1,
  line.width = 1, selection.scale = 2)
```

Arguments

`highlight`	vector of row indices to be highlighted in the plot
`alpha`	the transparency value; or `NA`
`label.size`	the font size of labels
`line.width`	the width of lines
`selection.scale`	the

`mordm.print.box` *Print descriptive representation of PRIM boxes.*

Description

Displays the PRIM boxes in a human-readable way.

Usage

```
mordm.print.box(data, mark, threshold = 0.01, digits = 3, indent = "")
```

Arguments

`data`	the original data set
`mark`	the PRIM boxes
`threshold`	fuzzy factor when determining if two numbers are equal
`digits`	number of digits to round numbers
`indent`	character string prepended to each line

mordm.rbind *Adds extra rows to the end of a data set.*

Description

Adds extra rows to the end of a data set.

Usage

```
mordm.rbind(set, rows)
```

Arguments

set	the data set
rows	the extra rows

mordm.read *Loads the time series data output from an optimization algorithm.*

Description

Reads the time series data (a list of matrices) from an optimization algorithm. The format is defined by the Borg MOEA and MOEA Framework.

Usage

```
mordm.read(file, nvars, nobjs, nconstrs = 0, bounds = NULL, names = NULL,
  maximize = NULL, digits = NULL)
```

Arguments

file	the file name
nvars	the number of decision variables
nobjs	the number of objectives
nconstrs	the number of constraints
bounds	the lower and upper bounds of each decision variable
names	override the column names
maximize	vector indicating the columns to be maximized
digits	number of digits to retain

`mordm.read.csv` *Loads a data set stored in a CSV file.*

Description

This method assumes the first N columns are decision variables, and all other columns are objectives. N is determined by `ncol(bounds)`. Unless overridden, this method sets `check.names=FALSE` and `header=TRUE`.

Usage

```
mordm.read.csv(file, nvars = NULL, nobjs = NULL, bounds = NULL,
  maximize = NULL, names = NULL, ignore = NULL, metadata = NULL, ...)
```

Arguments

`file`	the file name
`nvars`	the number of decision variables
`nobjs`	the number of objectives
`bounds`	the lower and upper bounds of each decision variable
`maximize`	vector indicating the columns to be maximized
`names`	override the column names
`ignore`	columns to remove from the dataset
`metadata`	columns to retain in a metadata attribute
`...`	optional arguments passed to read.csv

`mordm.read.matrix` *Loads a data set stored in a matrix or data.frame.*

Description

This method assumes the first N columns are decision variables, and all other columns are objectives. N is determined by `ncol(bounds)`.

Usage

```
mordm.read.matrix(mat, nvars = NULL, nobjs = NULL, bounds = NULL,
  maximize = NULL, names = NULL, ignore = NULL, metadata = NULL)
```

Arguments

`mat`	the matrix or data.frame
`nvars`	the number of decision variables
`nobjs`	the number of objectives
`bounds`	the lower and upper bounds of each decision variable
`maximize`	vector indicating the columns to be maximized
`names`	override the column names
`ignore`	columns to remove from the dataset
`metadata`	columns to retain in a metadata attribute

mordm.read.xls *Loads a data set stored in a XLS or XLSX file.*

Description

This method is similar in use to mordm.read.csv. Requires gdata and its dependencies, including a Perl interpreter on the host system.

Usage

```
mordm.read.xls(file, nvars = NULL, nobjs = NULL, bounds = NULL,
  maximize = NULL, names = NULL, ignore = NULL, metadata = NULL, ...)
```

Arguments

file	the file name
nvars	the number of decision variables
nobjs	the number of objectives
bounds	the lower and upper bounds of each decision variable
maximize	vector indicating the columns to be maximized
names	override the column names
ignore	columns to remove from the dataset
metadata	columns to retain in a metadata attribute
...	optional arguments passed to read.xls

mordm.recommend *Make recommendations for analyzing the data.*

Description

Performs basic checks to ensure the data is formatted correctly. If any issues are identified, then it will attempt to provide details on correcting or dealing with the problem.

Usage

```
mordm.recommend(data)
```

Arguments

data	the data set to be analyzed

`mordm.sample.uncertainties`
Computes robustness under deep compute.robustness.

Description

Adds Gaussian noise to the decision variables and resamples the model output. The samples are distributed across one or more different models for the problem. The result from this method should be passed to `mordm.evaluate.uncertainties` to compute the robustness metrics.

Usage

```
mordm.sample.uncertainties(data, nsamples, models, sd = 0, verbose = TRUE)
```

Arguments

`data`	the data set
`nsamples`	the number of samples to generate for each point
`models`	the problem formulations created using `define.problem`
`sd`	scalar or vector specifying the standard deviation for each decision variable
`verbose`	display additional information

Details

If multiple models are provided, it is assumed that all models have the same inputs and outputs; they would only differ in the internal calculcations within the model.

`mordm.select` *Returns the subset of rows that are marked.*

Description

Applies one or more markings to the data set and returns the subset that are contained within the markings.

Usage

```
mordm.select(data, marking, index = -1, not = FALSE, or = FALSE)
```

Arguments

`data`	the data set to be displayed (if data is a time series, then the last entry in the time series is displayed)
`marking`	list of markings
`index`	if data is a time series, controls which entries to display (see `mordm.get.set` for details)
`not`	DEPRECATED
`or`	DEPRECATED

mordm.select.indices *Returns the row indices in the data set that are marked.*

Description

Applies one or more markings to the data set to determine which rows are contained within the markings.

Usage

```
mordm.select.indices(set, marking, not = FALSE, or = FALSE)
```

Arguments

`set`	the data set
`marking`	list of markings
`not`	DEPRECATED
`or`	DEPRECATED

mordm.variable.sensitivities *Computes the sensitivities of the decision variables.*

Description

Using Plischke's delta-moment sensitivity analysis method, this function computes the sensitivities using the given data. As such, this method does not need to evaluate any new data points, it works with the provided data.

Usage

```
mordm.variable.sensitivities(data, objective, index = -1, all = FALSE, ...)
```

Arguments

`data`	the data set
`objective`	the objective index, column name, function, or marking
`index`	if data is a time series, controls which entries to display (see `mordm.get.set` for details)
`all`	if `TRUE`, include all points from all entries in the time series; otherwise, only the last entry is included
`...`	additional options for Plischke's method

Details

If `objective` is a marking, then this computes the sensitivities that cause a point to be included in the marking. This functionality is still experimental.

`mordm.weight` *Computes a vector of weighted preferences*

Description

Computes a vector of weighted preferences

Usage

```
mordm.weight(data, weights)
```

Arguments

`data`	the data
`weights`	the vector of weights

`nsample` *Generate normally distributed random inputs.*

Description

Generate normally distributed random inputs.

Usage

```
nsample(mean, sd, nsamples, problem)
```

Arguments

`mean`	scalar or vector specifying the mean value for each decision variable
`sd`	scalar or vector specifying the standard deviation for each decision variable
`nsamples`	the number of samples to generate
`problem`	the problem definition

robustness.constraints *Robustness metric based on constraint violations.*

Description

Measures the percentage of the sampled points that violate constraints.

Usage

```
robustness.constraints(output, problem, weights = NULL, verbose = FALSE,
  original.point = NULL)
```

Arguments

`output`	the evaluated points
`problem`	the problem definition
`weights`	unused
`verbose`	unused
`original.point`	unused

robustness.default *Default robustness metric.*

Description

The default robustness metric that combines variances and constraint violations.

Usage

```
robustness.default(output, problem, weights = NULL, verbose = FALSE,
  original.point = NULL)
```

Arguments

`output`	the evaluated points
`problem`	the problem definition
`weights`	the weights assigned to each objective
`verbose`	display additional information
`original.point`	the original point being analyzed

robustness.distance *Robustness metric based on distance.*

Description

Measures the average distance from the original point to the sampled points. This is slightly different from variance in that variance is not effected by translational distance. I.e., two point clouds have the same variance, but one is offset more.

Usage

```
robustness.distance(output, problem, weights = NULL, verbose = FALSE,
  original.point = NULL)
```

Arguments

`output`	the evaluated points
`problem`	the problem definition
`weights`	unused
`verbose`	unused
`original.point`	the original point being analyzed

robustness.gap *Experimental robustness metric based on info gap.*

Description

Info gap measures the distance from the original point to the nearest constraint boundary. This experimental implementation approximates this distance by computing the distance based on the sampled points.

Usage

```
robustness.gap(output, problem, weights = NULL, verbose = FALSE,
  original.point = NULL)
```

Arguments

`output`	the evaluated points
`problem`	the problem definition
`weights`	unused
`verbose`	unused
`original.point`	the original point being analyzed

robustness.variance *Robustness metric based on variance.*

Description

Measures the variance of the sampled points.

Usage

```
robustness.variance(output, problem, weights = NULL, verbose = FALSE,
  original.point = NULL)
```

Arguments

output	the evaluated points
problem	the problem definition
weights	the weights assigned to each objective
verbose	display additional information
original.point	the original point being analyzed

runVisDemo *Runs the exploration tool on a 5-objective problem.*

Description

Runs the exploration tool on a 5-objective problem.

Usage

```
runVisDemo()
```

sensitivity.levels *Determines number of replicates for sensitivity analysis.*

Description

Calculates the number of replicates / levels required by the sensitivity analysis method to produce approximately the given number of samples

Usage

```
sensitivity.levels(problem, nsamples, method)
```

Arguments

problem	the problem definition
nsamples	the desired number of samples
method	the sensitivity analysis method

usample *Generate uniformly distributed random inputs.*

Description

Generate uniformly distributed random inputs.

Usage

```
usample(nsamples, problem)
```

Arguments

nsamples	the number of samples to generate
problem	the problem definition

Index