Package: mooplot (via r-universe)

March 24, 2025

Type Package

Title Graphical Visualizations for Multi-Objective Optimization

Version 0.1.1

Description Visualization of multi-dimensional data arising in multi-objective optimization, including plots of the empirical attainment function (EAF), M. López-Ibáñez, L. Paquete, and T. Stützle (2010) <doi:10.1007/978-3-642-02538-9_9>, and symmetric Vorob'ev expectation and deviation, M. Binois, D. Ginsbourger, O. Roustant (2015) <doi:10.1016/j.ejor.2014.07.032>, among others.

Depends R (>= 4.0)

Imports Rdpack, collapse (>= 2.0.8), grDevices, graphics, matrixStats, moocore

Suggests extrafont, viridisLite, spelling, testthat (>= 3.0.0), withr

License LGPL (>= 2)

BugReports https://github.com/multi-objective/mooplot/issues/

URL https://multi-objective.github.io/mooplot/r/, https://github.com/multi-objective/mooplot/

LazyLoad true

LazyData true

Encoding UTF-8

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

RdMacros Rdpack

Config/testthat/edition 3

Language en-GB

Repository https://multi-objective.r-universe.dev

RemoteUrl https://github.com/multi-objective/mooplot

RemoteRef HEAD

RemoteSha d5fe2449d3c7f15c59f179d27dc0f465ff85b22b

RemoteSubdir r

Contents

`choose_eafdiffplot` *Interactively choose according to empirical attainment function differences*

Description

Creates the same plot as `eafdiffplot()` but waits for the user to click in one of the sides. Then it returns the rectangles the give the differences in favour of the chosen side. These rectangles may be used for interactive decision-making as shown in Diaz and López-Ibáñez (2021). The function `moocore::choose_eafdiff()` may be used in a non-interactive context.

Usage

```
choose_eafdiffplot(
  data_left,
  data_right,
  intervals = 5,
  maximise = c(FALSE, FALSE),
  title_left = deparse(substitute(data_left)),
  title_right = deparse(substitute(data_right)),
  ...
)
```

Arguments

`data_left, data_right`
: Data frames corresponding to the input data of left and right sides, respectively. Each data frame has at least three columns, the third one being the set of each point. See also `read_datasets()`.

`intervals` `(integer(1)|character())`
: The absolute range of the differences $[0, 1]$ is partitioned into the number of intervals provided. If an integer is provided, then labels for each interval are computed automatically. If a character vector is provided, its length is taken as the number of intervals.

`maximise` (`logical()|logical(1)`)
: Whether the objectives must be maximised instead of minimised. Either a single logical value that applies to all objectives or a vector of logical values, with one value per objective.

`title_left`, `title_right`
: Title for left and right panels, respectively.

`...`
: Other graphical parameters are passed down to `eafdiffplot()`.

Value

`matrix()` where the first 4 columns give the coordinates of two corners of each rectangle and the last column. In both cases, the last column gives the positive differences in favor of the chosen side.

References

Juan Esteban Diaz, Manuel López-Ibáñez (2021). "Incorporating Decision-Maker's Preferences into the Automatic Configuration of Bi-Objective Optimisation Algorithms." *European Journal of Operational Research*, **289**(3), 1209–1222. doi:10.1016/j.ejor.2020.07.059.

See Also

`moocore::read_datasets()`, `eafdiffplot()`, `moocore::whv_rect()`

Examples

```
library(moocore)
extdata_dir <- system.file(package="moocore", "extdata")
A1 <- read_datasets(file.path(extdata_dir, "wrots_l100w10_dat"))
A2 <- read_datasets(file.path(extdata_dir, "wrots_l10w100_dat"))
if (interactive()) {
  rectangles <- choose_eafdiffplot(A1, A2, intervals = 5)
} else { # Choose A1
  rectangles <- eafdiff(A1, A2, intervals = 5, rectangles = TRUE)
  rectangles <- choose_eafdiff(rectangles, left = TRUE)
}
reference <- c(max(A1[, 1], A2[, 1]), max(A1[, 2], A2[, 2]))
x <- split.data.frame(A1[,1:2], A1[,3])
hv_A1 <- sapply(split.data.frame(A1[, 1:2], A1[, 3]),
                hypervolume, reference=reference)
hv_A2 <- sapply(split.data.frame(A2[, 1:2], A2[, 3]),
                hypervolume, reference=reference)
boxplot(list(A1=hv_A1, A2=hv_A2), main = "Hypervolume")

whv_A1 <- sapply(split.data.frame(A1[, 1:2], A1[, 3]),
                whv_rect, rectangles=rectangles, reference=reference)
whv_A2 <- sapply(split.data.frame(A2[, 1:2], A2[, 3]),
                whv_rect, rectangles=rectangles, reference=reference)
boxplot(list(A1=whv_A1, A2=whv_A2), main = "Weighted hypervolume")
```

`eafdiffplot` *Plot empirical attainment function differences*

Description

Plot the differences between the empirical attainment functions (EAFs) of two data sets as a two-panel plot, where the left side shows the values of the left EAF minus the right EAF and the right side shows the differences in the other direction.

Usage

```
eafdiffplot(
  data_left,
  data_right,
  col = c("#FFFFFF", "#808080", "#000000"),
  intervals = 5L,
  percentiles = 50,
  full.eaf = FALSE,
  type = "area",
  legend.pos = if (full.eaf) "bottomleft" else "topright",
  maximise = c(FALSE, FALSE),
  title_left,
  title_right,
  xlim = NULL,
  ylim = NULL,
  cex = par("cex"),
  cex.lab = par("cex.lab"),
  cex.axis = par("cex.axis"),
  grand.lines = TRUE,
  sci.notation = FALSE,
  left.panel.last = NULL,
  right.panel.last = NULL,
  ...
)
```

Arguments

`data_left`, `data_right`
: Data frames corresponding to the input data of left and right sides, respectively. Each data frame has at least three columns, the third one being the set of each point. See also `read_datasets()`.

`col`
: A character vector of three colors for the magnitude of the differences of 0, 0.5, and 1. Intermediate colors are computed automatically given the value of `intervals`. Alternatively, a function such as `viridisLite::viridis()` that generates a colormap given an integer argument.

`intervals` (`integer(1)|character()`)
: The absolute range of the differences $[0, 1]$ is partitioned into the number of intervals provided. If an integer is provided, then labels for each interval are computed automatically. If a character vector is provided, its length is taken as the number of intervals.

`percentiles`
: The percentiles of the EAF of each side that will be plotted as attainment surfaces. `NA` does not plot any. See `eafplot()`.

`full.eaf`
: Whether to plot the EAF of each side instead of the differences between the EAFs.

`type` (`"points"|"area"`)
: Whether the EAF differences are plotted as points (`"points"`) or whether to color the areas that have at least a certain value (`"area"`).

`legend.pos` (`character(1)`)
: The position of the legend. See `legend()`. A value of `"none"` hides the legend.

`maximise` (`logical()|logical(1)`)
: Whether the objectives must be maximised instead of minimised. Either a single logical value that applies to all objectives or a vector of logical values, with one value per objective.

`title_left`, `title_right`
: Title for left and right panels, respectively.

`xlim`, `ylim`, `cex`, `cex.lab`, `cex.axis`
: Graphical parameters, see `plot.default()`.

`grand.lines`
: Whether to plot the grand-best and grand-worst attainment surfaces.

`sci.notation`
: Generate prettier labels

`left.panel.last`, `right.panel.last`
: An expression to be evaluated after plotting has taken place on each panel (left or right). This can be useful for adding points or text to either panel. Note that this works by lazy evaluation: passing this argument from other `plot` methods may well not work since it may be evaluated too early.

`...`
: Other graphical parameters are passed down to `plot.default()`.

Details

This function calculates the differences between the EAFs of two data sets, and plots on the left the differences in favour of the left data set, and on the right the differences in favour of the right data set. By default, it also plots the grand best and worst attainment surfaces, that is, the 0%- and 100%-attainment surfaces over all data. These two surfaces delimit the area where differences may exist. In addition, it also plots the 50%-attainment surface of each data set.

With `type = "point"`, only the points where there is a change in the value of the EAF difference are plotted. This means that for areas where the EAF differences stays constant, the region will appear in white even if the value of the differences in that region is large. This explains "white holes" surrounded by black points.

With `type = "area"`, the area where the EAF differences has a certain value is plotted. The idea for the algorithm to compute the areas was provided by Carlos M. Fonseca. The implementation uses R polygons, which some PDF viewers may have trouble rendering correctly (See `https://cran.`

r-project.org/doc/FAQ/R-FAQ.html#Why-are-there-unwanted-borders). Plots (should) look correct when printed.

Large differences that appear when using `type = "point"` may seem to disappear when using `type = "area"`. The explanation is the points size is independent of the axes range, therefore, the plotted points may seem to cover a much larger area than the actual number of points. On the other hand, the areas size is plotted with respect to the objective space, without any extra borders. If the range of an area becomes smaller than one-pixel, it won't be visible. As a consequence, zooming in or out certain regions of the plots does not change the apparent size of the points, whereas it affects considerably the apparent size of the areas.

Value

Returns a representation of the EAF differences (invisibly).

References

Viviane Grunert da Fonseca, Carlos M. Fonseca, Andreia O. Hall (2001). "Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function." In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, David Corne (eds.), *Evolutionary Multi-criterion Optimization, EMO 2001*, volume 1993 of *Lecture Notes in Computer Science*, 213–225. Springer, Berlin~/ Heidelberg. doi:10.1007/3540447199_15.

Manuel López-Ibáñez, Luís Paquete, Thomas Stützle (2010). "Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization." In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, Mike Preuss (eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, 209–222. Springer, Berlin~/ Heidelberg. doi:10.1007/9783642025389_9.

See Also

`read_datasets()` `eafplot()` `pdf_crop()`

Examples

```
## NOTE: The plots in the website look squashed because of how pkgdown
## generates them. They should look fine when you generate them yourself.
extdata_dir <- system.file(package="moocore", "extdata")
A1 <- read_datasets(file.path(extdata_dir, "ALG_1_dat.xz"))
A2 <- read_datasets(file.path(extdata_dir, "ALG_2_dat.xz"))
# These take time
eafdiffplot(A1, A2, full.eaf = TRUE)
if (requireNamespace("viridisLite", quietly=TRUE)) {
  viridis_r <- function(n) viridisLite::viridis(n, direction=-1)
  eafdiffplot(A1, A2, type = "area", col = viridis_r)
} else {
  eafdiffplot(A1, A2, type = "area")
}
A1 <- read_datasets(file.path(extdata_dir, "wrots_l100w10_dat"))
A2 <- read_datasets(file.path(extdata_dir, "wrots_l10w100_dat"))
eafdiffplot(A1, A2, type = "point", sci.notation = TRUE, cex.axis=0.6)

# A more complex example
```

```
DIFF <- eafdiffplot(A1, A2, col = c("white", "blue", "red"), intervals = 5,
                    type = "point",
                    title_left=expression("W-RoTS," ~ lambda==100 * "," ~ omega==10),
                    title_right=expression("W-RoTS," ~ lambda==10 * "," ~ omega==100),
                    right.panel.last={
                      abline(a = 0, b = 1, col = "red", lty = "dashed")})

## Save the values to a file.
# DIFF$right[,3] <- -DIFF$right[,3]
# write.table(rbind(DIFF$left,DIFF$right),
#             file = "wrots_l100w10_dat-wrots_l10w100_dat-diff.txt",
#             quote = FALSE, row.names = FALSE, col.names = FALSE)
```

---

eafplot *Plot the Empirical Attainment Function for two objectives*

Description

Computes and plots the Empirical Attainment Function (EAF), either as attainment surfaces for certain percentiles or as points.

Usage

```
eafplot(x, ...)

## Default S3 method:
eafplot(
  x,
  sets,
  groups = NULL,
  percentiles = c(0, 50, 100),
  attsurfs = NULL,
  maximise = c(FALSE, FALSE),
  type = "point",
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  log = "",
  col = NULL,
  lty = c("dashed", "solid", "solid", "solid", "dashed"),
  lwd = 1.75,
  pch = NA,
  cex.pch = par("cex"),
  las = par("las"),
 legend.pos = paste0(ifelse(maximise[1L], "bottom", "top"), ifelse(rep_len(maximise,
    2L)[2L], "left", "right")),
```

```
  legend.txt = NULL,
  extra.points = NULL,
  extra.legend = NULL,
  extra.pch = 4:25,
  extra.lwd = 0.5,
  extra.lty = NA,
  extra.col = "black",
  xaxis.side = "below",
  yaxis.side = "left",
  axes = TRUE,
  sci.notation = FALSE,
  ...
)

## S3 method for class 'list'
eafplot(x, ...)
```

Arguments

x	Either a matrix of data values, or a data frame, or a list of data frames of exactly three columns.
...	Other graphical parameters to plot.default().
sets	Vector indicating which set each point belongs to. Will be coerced to a factor.
groups	This may be used to plot data for different algorithms on the same plot. Will be coerced to a factor.
percentiles	(numeric()) Vector indicating which percentile should be plot. The default is to plot only the median attainment curve.
attsurfs	TODO
maximise	(logical()|logical(1)) Whether the objectives must be maximised instead of minimised. Either a single logical value that applies to all objectives or a vector of logical values, with one value per objective.
type	("point"|"area") Type of plot.
xlab, ylab, xlim, ylim, log, col, lty, lwd, pch, cex.pch, las	Graphical parameters, see plot.default().
legend.pos	(character(1)|list()|data.frame()) Position of the legend. This may be xy coordinates or a keyword ("bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"). See Details in legend(). A value of "none" hides the legend.
legend.txt	(expression()|character()) Character or expression vector to appear in the legend. If NULL, appropriate labels will be generated.
extra.points	A list of matrices or data.frames with two-columns. Each element of the list defines a set of points, or lines if one of the columns is NA.

`extra.legend` A character vector providing labels for the groups of points.

`extra.pch`, `extra.lwd`, `extra.lty`, `extra.col`
Control the graphical aspect of the points. See `points()` and `lines()`.

`xaxis.side` (`"below"|"above"`)
On which side the x-axis is drawn. See `axis()`.

`yaxis.side` (`"left"|"right"`)
On which side the y-axis is drawn. See `axis()`.

`axes` (`logical(1)`)
A logical value indicating whether both axes should be drawn on the plot.

`sci.notation` (`logical(1)`)
Generate prettier labels

Details

This function can be used to plot random sets of points like those obtained by different runs of biobjective stochastic optimisation algorithms (López-Ibáñez et al. 2010). An EAF curve represents the boundary separating points that are known to be attainable (that is, dominated in Pareto sense) in at least a fraction (quantile) of the runs from those that are not (Grunert da Fonseca et al. 2001). The median EAF represents the curve where the fraction of attainable points is 50%. In single objective optimisation the function can be used to plot the profile of solution quality over time of a collection of runs of a stochastic optimizer (López-Ibáñez et al. 2025).

Value

The attainment surfaces computed (invisibly).

Methods (by class)

- `eafplot(default)`: Main function
- `eafplot(list)`: List interface for lists of data.frames or matrices

References

Viviane Grunert da Fonseca, Carlos M. Fonseca, Andreia O. Hall (2001). "Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function." In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, David Corne (eds.), *Evolutionary Multi-criterion Optimization, EMO 2001*, volume 1993 of *Lecture Notes in Computer Science*, 213–225. Springer, Berlin~/ Heidelberg. doi:10.1007/3540447199_15.

Manuel López-Ibáñez, Luís Paquete, Thomas Stützle (2010). "Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization." In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, Mike Preuss (eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, 209–222. Springer, Berlin~/ Heidelberg. doi:10.1007/9783642025389_9.

Manuel López-Ibáñez, Diederick Vermetten, Johann Dreo, Carola Doerr (2025). "Using the Empirical Attainment Function for Analyzing Single-objective Black-box Optimization Algorithms." *IEEE Transactions on Evolutionary Computation*. doi:10.1109/TEVC.2024.3462758.

See Also

moocore::read_datasets() eafdiffplot() pdf_crop()

Examples

```
extdata_path <- system.file(package = "moocore", "extdata")
A1 <- read_datasets(file.path(extdata_path, "ALG_1_dat.xz"))
A2 <- read_datasets(file.path(extdata_path, "ALG_2_dat.xz"))
eafplot(A1, percentiles = 50, sci.notation = TRUE, cex.axis=0.6)
# The attainment surfaces are returned invisibly.
attsurfs <- eafplot(list(A1 = A1, A2 = A2), percentiles = 50)
str(attsurfs)

## Save as a PDF file.
# dev.copy2pdf(file = "eaf.pdf", onefile = TRUE, width = 5, height = 4)


## Using extra.points

data(HybridGA, package="moocore")
data(SPEA2relativeVanzyl, package="moocore")
eafplot(SPEA2relativeVanzyl, percentiles = c(25, 50, 75),
        xlab = expression(C[E]), ylab = "Total switches", xlim = c(320, 400),
        extra.points = HybridGA$vanzyl, extra.legend = "Hybrid GA")

data(SPEA2relativeRichmond, package="moocore")
eafplot (SPEA2relativeRichmond, percentiles = c(25, 50, 75),
         xlab = expression(C[E]), ylab = "Total switches",
         xlim = c(90, 140), ylim = c(0, 25),
         extra.points = HybridGA$richmond, extra.lty = "dashed",
         extra.legend = "Hybrid GA")

eafplot (SPEA2relativeRichmond, percentiles = c(25, 50, 75),
         xlab = expression(C[E]), ylab = "Total switches",
         xlim = c(90, 140), ylim = c(0, 25), type = "area",
         extra.points = HybridGA$richmond, extra.lty = "dashed",
         extra.legend = "Hybrid GA", legend.pos = "bottomright")

data(SPEA2minstoptimeRichmond, package="moocore")
SPEA2minstoptimeRichmond[,2] <- SPEA2minstoptimeRichmond[,2] / 60
eafplot (SPEA2minstoptimeRichmond, xlab = expression(C[E]),
         ylab = "Minimum idle time (minutes)", maximise = c(FALSE, TRUE),
         las = 1, log = "y", main = "SPEA2 (Richmond)",
         legend.pos = "bottomright")

data(tpls50x20_1_MWT, package="moocore")
eafplot(tpls50x20_1_MWT[, c(2,3)], sets = tpls50x20_1_MWT[,4L],
        groups = tpls50x20_1_MWT[["algorithm"]])
```

pdf_crop *Remove whitespace margins from a PDF file (and maybe embed fonts)*

Description

Remove whitespace margins using https://ctan.org/pkg/pdfcrop and optionally embed fonts using grDevices::embedFonts(). You may install pdfcrop using TinyTeX (https://cran.r-project.org/package=tinytex) with tinytex::tlmgr_install('pdfcrop').

Usage

```
pdf_crop(
  filename,
  mustWork = FALSE,
  pdfcrop = Sys.which("pdfcrop"),
  embed_fonts = FALSE
)
```

Arguments

filename	(character(1)) Filename of a PDF file to crop. The file will be overwritten.
mustWork	(logical1) If TRUE, then give an error if the file cannot be cropped.
pdfcrop	(character(1)) Path to the pdfcrop utility.
embed_fonts	(logical(1)) If TRUE, use grDevices::embedFonts() to embed fonts.

Details

You may also wish to consider extrafont::embed_fonts() (https://cran.r-project.org/package=extrafont).

```
library(extrafont)
# If you need to specify the path to Ghostscript (probably not needed in Linux)
Sys.setenv(R_GSCMD = "C:/Program Files/gs/gs9.56.1/bin/gswin64c.exe")
embed_fonts("original.pdf", outfile = "new.pdf")
```

As an alternative, saving the PDF with grDevices::cairo_pdf() should already embed the fonts.

Value

No return value, called for side effects

See Also

grDevices::embedFonts() extrafont::embed_fonts() grDevices::cairo_pdf()

Examples

```
extdata_path <- system.file(package = "moocore", "extdata")
A1 <- read_datasets(file.path(extdata_path, "wrots_l100w10_dat"))
A2 <- read_datasets(file.path(extdata_path, "wrots_l10w100_dat"))
filename <- tempfile("eafplot", fileext=".pdf")
pdf(file = filename, onefile = TRUE, width = 5, height = 4)
eafplot(list(A1 = A1, A2 = A2), percentiles = 50, sci.notation = TRUE)
dev.off()
try(pdf_crop(filename)) # This may fail if pdfcrop is not installed.
```

---

symdevplot *Plot the symmetric deviation function.*

Description

The symmetric deviation function is the probability for a given target in the objective space to belong to the symmetric difference between the Vorob'ev expectation and a realization of the (random) attained set.

Usage

```
symdevplot(
  x,
  sets,
  VE,
  threshold,
  nlevels = 11,
  ve.col = "blue",
  xlim = NULL,
  ylim = NULL,
  legend.pos = "topright",
  main = "Symmetric deviation function",
  col.fun = function(n) gray(seq(0, 0.9, length.out = n)^2)
)
```

Arguments

x	matrix()|data.frame() Matrix or data frame of numerical values, where each row gives the coordinates of a point. If sets is missing, the last column of x gives the sets.
sets	integer() A vector that indicates the set of each point in x. If missing, the last column of x is used instead.

VE, threshold	Vorob'ev expectation and threshold, e.g., as returned by moocore::vorobT().
nlevels	(integer(1)) Number of levels in which is divided the range of the symmetric deviation.
ve.col	Plotting parameters for the Vorob'ev expectation.
xlim, ylim, main	Graphical parameters, see plot.default().
legend.pos	The position of the legend, see legend(). A value of "none" hides the legend.
col.fun	Function that creates a vector of n colors, see heat.colors().

Value

No return value, called for side effects

Author(s)

Mickael Binois

References

M Binois, D Ginsbourger, O Roustant (2015). "Quantifying uncertainty on Pareto fronts with Gaussian process conditional simulations." *European Journal of Operational Research*, **243**(2), 386–394. doi:10.1016/j.ejor.2014.07.032.

C. Chevalier (2013), Fast uncertainty reduction strategies relying on Gaussian process models, University of Bern, PhD thesis.

Ilya Molchanov (2005). *Theory of Random Sets*. Springer.

See Also

moocore::vorobT() moocore::vorobDev() eafplot()

Examples

```
data(CPFs, package = "moocore")
res <- moocore::vorobT(CPFs, reference = c(2, 200))
print(res$threshold)

## Display Vorob'ev expectation and attainment function
# First style
eafplot(CPFs[,1:2], sets = CPFs[,3], percentiles = c(0, 25, 50, 75, 100, res$threshold),
        main = substitute(paste("Empirical attainment function, ",beta,"* = ", a, "%"),
                          list(a = formatC(res$threshold, digits = 2, format = "f"))))

# Second style
eafplot(CPFs[,1:2], sets = CPFs[,3], percentiles = c(0, 20, 40, 60, 80, 100),
        col = gray(seq(0.8, 0.1, length.out = 6)^0.5), type = "area",
        legend.pos = "bottomleft", extra.points = res$VE, extra.col = "cyan",
        extra.legend = "VE", extra.lty = "solid", extra.pch = NA, extra.lwd = 2,
        main = substitute(paste("Empirical attainment function, ",beta,"* = ", a, "%"),
                          list(a = formatC(res$threshold, digits = 2, format = "f"))))
# Vorob'ev deviation
```

```
VD <- moocore::vorobDev(CPFs, reference = c(2, 200), VE = res$VE)
# Display the symmetric deviation function.
symdevplot(CPFs, VE = res$VE, threshold = res$threshold, nlevels = 11)
# Levels are adjusted automatically if too large.
symdevplot(CPFs, VE = res$VE, threshold = res$threshold, nlevels = 200, legend.pos = "none")

# Use a different palette.
symdevplot(CPFs, VE = res$VE, threshold = res$threshold, nlevels = 11, col.fun = heat.colors)
```

Index