Package 'gMOIP'

August 5, 2019

Type Package

Title '2D and 3D plots of linear mathematical programming models'

Version 1.3.0

URL https://github.com/relund/gMOIP/

BugReports https://github.com/relund/gMOIP/issues

Description Make 2D and 3D plots of the polytope of a LP, ILP or MILP problem, including integer points and iso profit curve. Can also make a plot of the bi-objective criterion space.

License GPL (>= 3.3.2)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.5.0)

Imports ggrepel, geometry, ggplot2, rgl, MASS, plyr

Suggests tikzDevice, grid, gridExtra, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Lars Relund [aut, cre]

Maintainer Lars Relund <lars@relund.dk>

Repository CRAN

Date/Publication 2019-08-05 14:10:06 UTC

R topics documented:

`cornerPoints` *Calculate the corner points for the polytope Ax<=0.*

Description

Calculate the corner points for the polytope Ax<=0.

Usage

```
cornerPoints(A, b, type = rep("c", ncol(A)), nonneg = rep(TRUE,
  ncol(A)))
```

Arguments

`A`	Constraint matrix.
`b`	Right hand side.
`type`	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
`nonneg`	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.

Value

A data frame with a corner point in each row.

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
A <- matrix( c(3,-2, 1, 2, 4,-2,-3, 2, 1), nc = 3, byrow = TRUE)
b <- c(10, 12, 3)
cornerPoints(A, b, type = c("c", "c", "c"))
cornerPoints(A, b, type = c("i", "i", "i"))
cornerPoints(A, b, type = c("i", "c", "c"))
```

`cornerPointsCont` *Calculate the corner points for the polytope Ax<=b assuming all variables are continuous.*

Description

Calculate the corner points for the polytope Ax<=b assuming all variables are continuous.

Usage

```
cornerPointsCont(A, b, nonneg = rep(TRUE, ncol(A)))
```

Arguments

`A`	Constraint matrix.
`b`	Right hand side.
`nonneg`	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.

Value

A data frame with a corner point in each row.

Author(s)

Lars Relund <lars@relund.dk>

`criterionPoints` *Calculate the criterion points of a set of points and ranges to find the set of non-dominated points (Pareto points) and classify them into extreme supported, non-extreme supported, non-supported.*

Description

Calculate the criterion points of a set of points and ranges to find the set of non-dominated points (Pareto points) and classify them into extreme supported, non-extreme supported, non-supported.

Usage

```
criterionPoints(points, obj, crit, labels = "coord")
```

Arguments

points	A data frame with a column for each variable in the solution space (can also be a rangePoints).
obj	A p x n matrix(one row for each criterion).
crit	Either max or min.
labels	If NULL or "n" don't add any labels (empty string). If 'coord' labels are the solution space coordinates. Otherwise number all points from one based on the solution space points.

Value

A data frame with columns x1, ..., xn, z1, ..., zp, lbl (label), nD (non-dominated), ext (extreme), nonExt (non-extreme supported).

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
A <- matrix( c(3, -2, 1, 2, 4, -2, -3, 2, 1), nc = 3, byrow = TRUE)
b <- c(10,12,3)
points <- integerPoints(A, b)
obj <- matrix( c(1,-3,1,-1,1,-1), byrow = TRUE, ncol = 3 )
criterionPoints(points, obj, crit = "max", labels = "numb")
```

---

df2String *Convert each row to a string.*

Description

Convert each row to a string.

Usage

```
df2String(df, round = 2)
```

Arguments

df	Data frame.
round	How many digits to round

Value

A vector of strings.

gMOIP *2D plots of linear/integer programming models.*

Description

Make 2D and 3D plots of the polytope of a linear programming (LP), integer linear programming (ILP) model, or mixed integer linear programming (MILP) model with 2 or 3 variables, including integer points, ranges and iso profit curve.

Details

Can also make a plot of the bi-objective criterion space and the non-dominated (Pareto) set for bi-objective LP/ILP/MILP programming models.

Author(s)

Lars Relund <lars@relund.dk> [aut, cre].

See Also

plotPolytope and plotCriterion2D.

---

hullSegment *Find segments (lines) of a face.*

Description

Find segments (lines) of a face.

Usage

```
hullSegment(vertices, hull = geometry::convhulln(vertices),
  tol = mean(mean(abs(vertices))) * sqrt(.Machine$double.eps))
```

Arguments

vertices	A mxp array of vertices of the convex hull, as used by convhulln.
hull	Tessellation (or triangulation) generated by convhulln If hull is left empty or not supplied, then it will be generated.
tol	Tolerance on the tests for inclusion in the convex hull. You can think of tol as the distance a point may possibly lie outside the hull, and still be perceived as on the surface of the hull. Because of numerical slop nothing can ever be done exactly here. I might guess a semi-intelligent value of tol to be tol = 1.e-13*mean(abs(vertices(:))) In higher dimensions, the numerical issues of floating point arithmetic will probably suggest a larger value of tol.

Value

A matrix with segments.

Author(s)

Lars Relund <lars@relund.dk>

`inHull` *Efficient test for points inside a convex hull in n dimensions. Inspired by the Matlab code by John D'Errico http://www.mathworks.com/matlabcentral/fileexchange/10226-inhull and https://tolstoy.newcastle.edu.au/R/e8/help/09/12/8784.html*

Description

Efficient test for points inside a convex hull in n dimensions. Inspired by the Matlab code by John D'Errico http://www.mathworks.com/matlabcentral/fileexchange/10226-inhull and https://tolstoy.newcastle.edu.au/R/e8/help

Usage

```
inHull(testPts, vertices, hull = geometry::convhulln(vertices),
  tol = mean(mean(abs(vertices))) * sqrt(.Machine$double.eps))
```

Arguments

`testPts`	A nxp array to test, n data points, in p dimensions If you have many points to test, it is most efficient to call this function once with the entire set.
`vertices`	A mxp array of vertices of the convex hull, as used by convhulln.
`hull`	Tessellation (or triangulation) generated by convhulln If hull is left empty or not supplied, then it will be generated.
`tol`	Tolerance on the tests for inclusion in the convex hull. You can think of tol as the distance a point may possibly lie outside the hull, and still be perceived as on the surface of the hull. Because of numerical slop nothing can ever be done exactly here. I might guess a semi-intelligent value of tol to be tol = 1.e-13*mean(abs(vertices(:))) In higher dimensions, the numerical issues of floating point arithmetic will probably suggest a larger value of tol.

Value

An integer vector of length n

Author(s)

Lars Relund <lars@relund.dk>

integerPoints *Integer points in the feasible region (Ax<=b).*

Description

Integer points in the feasible region (Ax<=b).

Usage

```
integerPoints(A, b, nonneg = rep(TRUE, ncol(A)))
```

Arguments

A	Constraint matrix.
b	Right hand side.
nonneg	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.

Value

A data frame with all integer points inside the feasible region.

Note

Do a simple enumeration of all integer points between min and max values found using the continuous polytope.

Author(s)

Lars Relund <lars@relund.dk>.

Examples

```
A <- matrix( c(3,-2, 1, 2, 4,-2,-3, 2, 1), nc = 3, byrow = TRUE)
b <- c(10, 12, 3)
integerPoints(A, b)

A <- matrix(c(9, 10, 2, 4, -3, 2), ncol = 2, byrow = TRUE)
b <- c(90, 27, 3)
integerPoints(A, b)
```

loadView *Help function to load the view angle for the RGL 3D plot from a file or matrix*

Description

Help function to load the view angle for the RGL 3D plot from a file or matrix

Usage

```
loadView(fname = "view.RData", v = NULL, clear = TRUE,
  close = FALSE, zoom = 1, ...)
```

Arguments

fname	The file name of the view.
v	The view matrix.
clear	Call clear3d.
close	Call rgl.close.
zoom	Zoom level.
...	Additional parameters passed to view3d.

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
view <- matrix( c(-0.412063330411911, -0.228006735444069, 0.882166087627411, 0,
0.910147845745087, -0.0574885793030262, 0.410274744033813, 0, -0.042830865830183,
0.97196090221405, 0.231208890676498, 0, 0, 0, 0, 1), nc = 4)

loadView(v = view)
A <- matrix( c(3, 2, 5, 2, 1, 1, 1, 1, 3, 5, 2, 4), nc = 3, byrow = TRUE)
b <- c(55, 26, 30, 57)
obj <- c(20, 10, 15)
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")

# Try to modify the angle in the RGL window
saveView(print = TRUE)  # get the viewangle to insert into R code
```

mergeLists *Merge two lists to one*

Description

Merge two lists to one

Usage

```
mergeLists(a, b)
```

Arguments

a	First list.
b	Second list.

plotCriterion2D *Create a plot of the criterion space of a bi-objective problem*

Description

Create a plot of the criterion space of a bi-objective problem

Usage

```
plotCriterion2D(A, b, obj, type = rep("c", ncol(A)), nonneg = rep(TRUE,
  ncol(A)), crit = "max", addTriangles = FALSE, addHull = TRUE,
  plotFeasible = TRUE, latex = FALSE, labels = NULL)
```

Arguments

A	The constraint matrix.
b	Right hand side.
obj	A p x n matrix(one row for each criterion).
type	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
nonneg	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.
crit	Either max or min (only used if add the iso profit line).
addTriangles	Add search triangles defined by the non-dominated extreme points.
addHull	Add the convex hull and the rays.
plotFeasible	If True then plot the criterion points/slices.
latex	If true make latex math labels for TikZ.
labels	If NULL don't add any labels. If 'n' no labels but show the points. If 'coord' add coordinates to the points. Otherwise number all points from one.

Value

The ggplot2 object.

Note

Currently only points are checked for dominance. That is, for MILP models some nondominated points may in fact be dominated by a segment.

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
### Set up 2D plot
# Function for plotting the solution and criterion space in one plot (two variables)
plotBiObj2D <- function(A, b, obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = FALSE,
   labels = "numb",
   addTriangles = TRUE,
   addHull = TRUE)
{
   p1 <- plotPolytope(A, b, type = type, crit = crit, faces = faces, plotFaces = plotFaces,
                  plotFeasible = plotFeasible, plotOptimum = plotOptimum, labels = labels)
   p2 <- plotCriterion2D(A, b, obj, type = type, crit = crit, addTriangles = addTriangles,
                         addHull = addHull, plotFeasible = plotFeasible, labels = labels)
   gridExtra::grid.arrange(p1, p2, nrow = 1)
}


### Bi-objective problem with two variables
A <- matrix(c(-3,2,2,4,9,10), ncol = 2, byrow = TRUE)
b <- c(3,27,90)

## LP model
obj <- matrix(
   c(7, -10, # first criterion
     -10, -10), # second criterion
   nrow = 2)
plotBiObj2D(A, b, obj, addTriangles = FALSE)

## ILP models with different criteria (maximize)
obj <- matrix(c(7, -10, -10, -10), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)))
obj <- matrix(c(3, -1, -2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)))
obj <- matrix(c(-7, -1, -5, 5), nrow = 2)
```

```
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)))
obj <- matrix(c(-1, -1, 2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)))

## ILP models with different criteria (minimize)
obj <- matrix(c(7, -10, -10, -10), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)), crit = "min")
obj <- matrix(c(3, -1, -2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)), crit = "min")
obj <- matrix(c(-7, -1, -5, 5), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)), crit = "min")
obj <- matrix(c(-1, -1, 2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = rep("i", ncol(A)), crit = "min")


# More examples

## MILP model (x1 integer) with different criteria (maximize)
obj <- matrix(c(7, -10, -10, -10), nrow = 2)
plotBiObj2D(A, b, obj, type = c("i", "c"))
obj <- matrix(c(3, -1, -2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = c("i", "c"))
obj <- matrix(c(-7, -1, -5, 5), nrow = 2)
plotBiObj2D(A, b, obj, type = c("i", "c"))
obj <- matrix(c(-1, -1, 2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = c("i", "c"))

## MILP model (x2 integer) with different criteria (minimize)
obj <- matrix(c(7, -10, -10, -10), nrow = 2)
plotBiObj2D(A, b, obj, type = c("c", "i"), crit = "min")
obj <- matrix(c(3, -1, -2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = c("c", "i"), crit = "min")
obj <- matrix(c(-7, -1, -5, 5), nrow = 2)
plotBiObj2D(A, b, obj, type = c("c", "i"), crit = "min")
obj <- matrix(c(-1, -1, 2, 2), nrow = 2)
plotBiObj2D(A, b, obj, type = c("c", "i"), crit = "min")


### Set up 3D plot
# Function for plotting the solution and criterion space in one plot (three variables)
plotBiObj3D <- function(A, b, obj,
                        type = rep("c", ncol(A)),
                        crit = "max",
                        faces = rep("c", ncol(A)),
                        plotFaces = TRUE,
                        plotFeasible = TRUE,
                        plotOptimum = FALSE,
                        labels = "numb",
                        addTriangles = TRUE,
                        addHull = TRUE)
{
   plotPolytope(A, b, type = type, crit = crit, faces = faces, plotFaces = plotFaces,
                plotFeasible = plotFeasible, plotOptimum = plotOptimum, labels = labels)
```

```
   plotCriterion2D(A, b, obj, type = type, crit = crit, addTriangles = addTriangles,
                   addHull = addHull, plotFeasible = plotFeasible, labels = labels)
}

### Bi-objective problem with three variables
loadView <- function(fname = "view.RData", v = NULL) {
   if (!is.null(v)) {
      rgl::view3d(userMatrix = v)
   } else {
      if (file.exists(fname)) {
         load(fname)
         rgl::view3d(userMatrix = view)
      } else {
         warning(paste0("Can'TRUE load view in file ", fname, "!"))
      }
   }
}

## Ex
view <- matrix( c(-0.452365815639496, -0.446501553058624, 0.77201122045517, 0, 0.886364221572876,
            -0.320795893669128, 0.333835482597351, 0, 0.0986008867621422, 0.835299551486969,
                  0.540881276130676, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
Ab <- matrix( c(
   1, 1, 2, 5,
   2, -1, 0, 3,
   -1, 2, 1, 3,
   0, -3, 5, 2
), nc = 4, byrow = TRUE)
A <- Ab[,1:3]
b <- Ab[,4]
obj <- matrix(c(1, -6, 3, -4, 1, 6), nrow = 2)

# LP model
plotBiObj3D(A, b, obj, crit = "min", addTriangles = FALSE)

# ILP model
plotBiObj3D(A, b, obj, type = c("i","i","i"), crit = "min")

# MILP model
plotBiObj3D(A, b, obj, type = c("c","i","i"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","c","i"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","i","c"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","c","c"), crit = "min")
plotBiObj3D(A, b, obj, type = c("c","i","c"), crit = "min")
plotBiObj3D(A, b, obj, type = c("c","c","i"), crit = "min")


## Ex
view <- matrix( c(0.976349174976349, -0.202332556247711, 0.0761845782399178, 0, 0.0903248339891434,
            0.701892614364624, 0.706531345844269, 0, -0.196427255868912, -0.682940244674683,
                  0.703568696975708, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
```

```
A <- matrix( c(
   -1, 1, 0,
   1, 4, 0,
   2, 1, 0,
   3, -4, 0,
   0, 0, 4
), nc = 3, byrow = TRUE)
b <- c(5, 45, 27, 24, 10)
obj <- matrix(c(1, -6, 3, -4, 1, 6), nrow = 2)

# LP model
plotBiObj3D(A, b, obj, crit = "min", addTriangles = FALSE, labels = "coord")

# ILP model
plotBiObj3D(A, b, obj, type = c("i","i","i"))

# MILP model
plotBiObj3D(A, b, obj, type = c("c","i","i"))
plotBiObj3D(A, b, obj, type = c("i","c","i"), plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("i","i","c"))
plotBiObj3D(A, b, obj, type = c("i","c","c"), plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("c","i","c"), plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("c","c","i"))


## Ex
view <- matrix( c(-0.812462985515594, -0.029454167932272, 0.582268416881561, 0, 0.579295456409454,
            -0.153386667370796, 0.800555109977722, 0, 0.0657325685024261, 0.987727105617523,
                  0.14168381690979, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
A <- matrix( c(
   1, 1, 1,
   3, 0, 1
), nc = 3, byrow = TRUE)
b <- c(10, 24)
obj <- matrix(c(1, -6, 3, -4, 1, 6), nrow = 2)

# LP model
plotBiObj3D(A, b, obj, crit = "min", addTriangles = FALSE, labels = "coord")

# ILP model
plotBiObj3D(A, b, obj, type = c("i","i","i"), crit = "min", labels = "n")

# MILP model
plotBiObj3D(A, b, obj, type = c("c","i","i"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","c","i"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","i","c"), crit = "min")
plotBiObj3D(A, b, obj, type = c("i","c","c"), crit = "min")
plotBiObj3D(A, b, obj, type = c("c","i","c"), crit = "min", plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("c","c","i"), crit = "min", plotFaces = FALSE)


## Ex
```

```
view <- matrix( c(-0.412063330411911, -0.228006735444069, 0.882166087627411, 0, 0.910147845745087,
            -0.0574885793030262, 0.410274744033813, 0, -0.042830865830183, 0.97196090221405,
                  0.231208890676498, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
A <- matrix( c(
3, 2, 5,
2, 1, 1,
1, 1, 3,
5, 2, 4
), nc = 3, byrow = TRUE)
b <- c(55, 26, 30, 57)
obj <- matrix(c(1, -6, 3, -4, 1, -1), nrow = 2)

# LP model
plotBiObj3D(A, b, obj, crit = "min", addTriangles = FALSE, labels = "coord")

# ILP model
plotBiObj3D(A, b, obj, type = c("i","i","i"), crit = "min", labels = "n")

# MILP model
plotBiObj3D(A, b, obj, type = c("c","i","i"), crit = "min", labels = "n")
plotBiObj3D(A, b, obj, type = c("i","c","i"), crit = "min", labels = "n", plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("i","i","c"), crit = "min", labels = "n")
plotBiObj3D(A, b, obj, type = c("i","c","c"), crit = "min", labels = "n")
plotBiObj3D(A, b, obj, type = c("c","i","c"), crit = "min", labels = "n", plotFaces = FALSE)
plotBiObj3D(A, b, obj, type = c("c","c","i"), crit = "min", labels = "n")
```

`plotPolytope` *Plot the polytope (bounded convex set) of a linear mathematical program*

Description

Plot the polytope (bounded convex set) of a linear mathematical program

Usage

```
plotPolytope(A, b, obj = NULL, type = rep("c", ncol(A)),
  nonneg = rep(TRUE, ncol(A)), crit = "max", faces = type,
  plotFaces = TRUE, plotFeasible = TRUE, plotOptimum = FALSE,
  latex = FALSE, labels = NULL, ...)
```

Arguments

`A`	The constraint matrix.
`b`	Right hand side.
`obj`	A vector with objective coefficients.

type	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
nonneg	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.
crit	Either max or min (only used if add the iso profit line)
faces	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous. Useful if e.g. want to show the linear relaxation of an IP.
plotFaces	If True then plot the faces.
plotFeasible	If True then plot the feasible points/segments (relevant for IPLP/MILP).
plotOptimum	Show the optimum corner solution point (if alternative solutions only one is shown) and add the iso profit line.
latex	If True make latex math labels for TikZ.
labels	If NULL don't add any labels. If 'n' no labels but show the points. If 'coord' add coordinates to the points. Otherwise number all points from one.
...	If 2D arguments passed to the aes_string function in geom_point or geom_line.

Value

If 2D a ggplot2 object. If 3D a rgl window with 3D plot.

Note

The feasible region defined by the constraints must be bounded otherwise you may see strange results.

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
### 2D examples
# Define the model max/min coeff*x st. Ax<=b, x>=0
A <- matrix(c(-3,2,2,4,9,10), ncol = 2, byrow = TRUE)
b <- c(3,27,90)
obj <- c(7.75, 10)

## LP model
# The polytope with the corner points
plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
```

```
   plotFeasible = TRUE,
   plotOptimum = FALSE,
   labels = NULL
)
# With optimum and labels:
plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)
# Minimize:
plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "min",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "n"
)
# Note return a ggplot so can e.g. add other labels on e.g. the axes:
p <- plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)
p + ggplot2::xlab("x") + ggplot2::ylab("y")

# More examples

## LP-model with no non-negativity constraints
A <- matrix(c(-3, 2, 2, 4, 9, 10, 1, -2), ncol = 2, byrow = TRUE)
b <- c(3, 27, 90, 2)
obj <- c(7.75, 10)
plotPolytope(
   A,
```

```
   b,
   obj,
   type = rep("c", ncol(A)),
   nonneg = rep(FALSE, ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = FALSE,
   labels = NULL
)



## The package don't plot feasible regions that are unbounded e.g if we drop the 2 and 3 constraint
A <- matrix(c(-3,2), ncol = 2, byrow = TRUE)
b <- c(3)
obj <- c(7.75, 10)
# Wrong plot
plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = FALSE,
   labels = NULL
)
# One solution is to add a bounding box and check if the bounding box is binding
A <- rbind(A, c(1,0), c(0,1))
b <- c(b, 10, 10)
plotPolytope(
   A,
   b,
   obj,
   type = rep("c", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = FALSE,
   labels = NULL
)


## ILP model
A <- matrix(c(-3,2,2,4,9,10), ncol = 2, byrow = TRUE)
b <- c(3,27,90)
obj <- c(7.75, 10)
# ILP model with LP faces:
```

```
plotPolytope(
   A,
   b,
   obj,
   type = rep("i", ncol(A)),
   crit = "max",
   faces = rep("c", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)
#ILP model with IP faces:
plotPolytope(
   A,
   b,
   obj,
   type = rep("i", ncol(A)),
   crit = "max",
   faces = rep("i", ncol(A)),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)


## MILP model
A <- matrix(c(-3,2,2,4,9,10), ncol = 2, byrow = TRUE)
b <- c(3,27,90)
obj <- c(7.75, 10)
# Second coordinate integer
plotPolytope(
   A,
   b,
   obj,
   type = c("c", "i"),
   crit = "max",
   faces = c("c", "i"),
   plotFaces = FALSE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)
# First coordinate integer and with LP faces:
plotPolytope(
   A,
   b,
   obj,
   type = c("i", "c"),
   crit = "max",
   faces = c("c", "c"),
   plotFaces = TRUE,
```

```
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)
# First coordinate integer and with LP faces:
plotPolytope(
   A,
   b,
   obj,
   type = c("i", "c"),
   crit = "max",
   faces = c("i", "c"),
   plotFaces = TRUE,
   plotFeasible = TRUE,
   plotOptimum = TRUE,
   labels = "coord"
)



### 3D examples
# Ex 1
view <- matrix( c(-0.412063330411911, -0.228006735444069, 0.882166087627411, 0, 0.910147845745087,
            -0.0574885793030262, 0.410274744033813, 0, -0.042830865830183, 0.97196090221405,
                  0.231208890676498, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
A <- matrix( c(
   3, 2, 5,
   2, 1, 1,
   1, 1, 3,
   5, 2, 4
), nc = 3, byrow = TRUE)
b <- c(55, 26, 30, 57)
obj <- c(20, 10, 15)
# LP model
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")
# ILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","i"), plotOptimum = TRUE, obj = obj)
# MILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","c","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("c","i","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotFaces = FALSE)
plotPolytope(A, b, type = c("i","c","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","i","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","c","i"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)

# Ex 2
view <- matrix( c(-0.812462985515594, -0.029454167932272, 0.582268416881561, 0, 0.579295456409454,
            -0.153386667370796, 0.800555109977722, 0, 0.0657325685024261, 0.987727105617523,
                  0.14168381690979, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
```

```
A <- matrix( c(
   1, 1, 1,
   3, 0, 1
), nc = 3, byrow = TRUE)
b <- c(10, 24)
obj <- c(20, 10, 15)
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")
# ILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","i"), plotOptimum = TRUE, obj = obj)
# MILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","c","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("c","i","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotFaces = FALSE)
plotPolytope(A, b, type = c("i","c","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","i","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","c","i"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)

# Ex 3
view <- matrix( c(0.976349174976349, -0.202332556247711, 0.0761845782399178, 0, 0.0903248339891434,
            0.701892614364624, 0.706531345844269, 0, -0.196427255868912, -0.682940244674683,
                  0.703568696975708, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
A <- matrix( c(
   -1, 1, 0,
   1, 4, 0,
   2, 1, 0,
   3, -4, 0,
   0, 0, 4
), nc = 3, byrow = TRUE)
b <- c(5, 45, 27, 24, 10)
obj <- c(5, 45, 15)
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")
# ILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","i"), plotOptimum = TRUE, obj = obj)
# MILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","c","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("c","i","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotFaces = FALSE)
plotPolytope(A, b, type = c("i","c","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","i","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","c","i"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)

# Ex 4
view <- matrix( c(-0.452365815639496, -0.446501553058624, 0.77201122045517, 0, 0.886364221572876,
            -0.320795893669128, 0.333835482597351, 0, 0.0986008867621422, 0.835299551486969,
                  0.540881276130676, 0, 0, 0, 0, 1), nc = 4)
loadView(v = view)
Ab <- matrix( c(
   1, 1, 2, 5,
   2, -1, 0, 3,
   -1, 2, 1, 3,
```

```
   0, -3, 5, 2
   #   0, 1, 0, 4,
   #   1, 0, 0, 4
), nc = 4, byrow = TRUE)
A <- Ab[,1:3]
b <- Ab[,4]
obj = c(1,1,3)
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")
# ILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","i"), plotOptimum = TRUE, obj = obj)
# MILP model
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","c","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("c","i","i"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotOptimum = TRUE, obj = obj)
plotPolytope(A, b, faces = c("c","c","c"), type = c("i","i","c"), plotFaces = FALSE)
plotPolytope(A, b, type = c("i","c","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, type = c("c","i","c"), plotOptimum = TRUE, obj = obj, plotFaces = FALSE)
plotPolytope(A, b, faces = c("c","c","c"), type = c("c","c","i"), plotOptimum = TRUE, obj = obj)
```

`plotPolytope2D` *Plot the polytope (bounded convex set) of a linear mathematical program*

Description

Plot the polytope (bounded convex set) of a linear mathematical program

Usage

```
plotPolytope2D(A, b, obj = NULL, type = rep("c", ncol(A)),
  nonneg = rep(TRUE, ncol(A)), crit = "max", faces = rep("c",
  ncol(A)), plotFaces = TRUE, plotFeasible = TRUE,
  plotOptimum = FALSE, latex = FALSE, labels = NULL, ...)
```

Arguments

`A`	The constraint matrix.
`b`	Right hand side.
`obj`	A vector with objective coefficients.
`type`	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
`nonneg`	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.
`crit`	Either max or min (only used if add the iso profit line)
`faces`	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous. Useful if e.g. want to show the linear relaxation of an IP.

plotFaces	If True then plot the faces.
plotFeasible	If True then plot the feasible points/segments (relevant for ILP/MILP).
plotOptimum	Show the optimum corner solution point (if alternative solutions only one is shown) and add the iso profit line.
latex	If True make latex math labels for TikZ.
labels	If NULL don't add any labels. If 'n' no labels but show the points. If 'coord' add coordinates to the points. Otherwise number all points from one.
...	If 2D arguments passed to the aes_string function in geom_point or geom_line.

Value

A ggplot2 object.

Author(s)

Lars Relund <lars@relund.dk>

`plotPolytope3D` *Plot the polytope (bounded convex set) of a linear mathematical program*

Description

Plot the polytope (bounded convex set) of a linear mathematical program

Usage

```
plotPolytope3D(A, b, obj = NULL, type = rep("c", ncol(A)),
  nonneg = rep(TRUE, ncol(A)), crit = "max", faces = rep("c",
  ncol(A)), plotFaces = TRUE, plotFeasible = TRUE,
  plotOptimum = FALSE, latex = FALSE, labels = NULL, ...)
```

Arguments

A	The constraint matrix.
b	Right hand side.
obj	A vector with objective coefficients.
type	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
nonneg	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.
crit	Either max or min (only used if add the iso profit line)
faces	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous. Useful if e.g. want to show the linear relaxation of an IP.

plotFaces	If True then plot the faces.
plotFeasible	If True then plot the feasible points/segments (relevant for IPLP/MILP).
plotOptimum	Show the optimum corner solution point (if alternative solutions only one is shown) and add the iso profit line.
latex	If True make latex math labels for TikZ.
labels	If NULL don't add any labels. If 'n' no labels but show the points. If 'coord' add coordinates to the points. Otherwise number all points from one.
...	Arguments passed to axes3d, plot3d, title3d. Parsed using lists argsAxes3d, argsPlot3d and argsTitle3d.

Value

A rgl window with 3D plot.

Note

The feasible region defined by the constraints must be bounded otherwise you may see strange results.

Author(s)

Lars Relund <lars@relund.dk>

---

saveView *Help function to save the view angle for the RGL 3D plot*

Description

Help function to save the view angle for the RGL 3D plot

Usage

```
saveView(fname = "view.RData", overwrite = FALSE, print = FALSE)
```

Arguments

fname	The file name of the view.
overwrite	Overwrite existing file.
print	Print the view so can be copied to R code (no file is saved).

Value

The ggplot2 object.

Note

Only save if the file name don't exists.

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
view <- matrix( c(-0.412063330411911, -0.228006735444069, 0.882166087627411, 0,
0.910147845745087, -0.0574885793030262, 0.410274744033813, 0, -0.042830865830183,
0.97196090221405, 0.231208890676498, 0, 0, 0, 0, 1), nc = 4)

loadView(v = view)
A <- matrix( c(3, 2, 5, 2, 1, 1, 1, 1, 3, 5, 2, 4), nc = 3, byrow = TRUE)
b <- c(55, 26, 30, 57)
obj <- c(20, 10, 15)
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord")

# Try to modify the angle in the RGL window
saveView(print = TRUE)  # get the viewangle to insert into R code
```

---

slices *Find all corner points in the slices define for each fixed integer combination.*

Description

Find all corner points in the slices define for each fixed integer combination.

Usage

```
slices(A, b, type = rep("c", ncol(A)), nonneg = rep(TRUE, ncol(A)),
  collapse = FALSE)
```

Arguments

A	The constraint matrix.
b	Right hand side.
type	A character vector of same length as number of variables. If entry k is 'i' variable k must be integer and if 'c' continuous.
nonneg	A boolean vector of same length as number of variables. If entry k is TRUE then variable k must be non-negative.
collapse	Collapse list to a data frame with unique points.

Value

A list with the corner points (one entry for each slice).

Examples

```
A <- matrix( c(3, -2, 1,2, 4, -2,-3, 2, 1), nc = 3, byrow = TRUE)
b <- c(10,12,3)
slices(A, b, type=c("i","c","i"))

A <- matrix(c(9,10,2,4,-3,2), ncol = 2, byrow = TRUE)
b <- c(90,27,3)
slices(A, b, type=c("c","i"), collapse = TRUE)
```

Index