Package 'gMOIP'

February 20, 2020

Type Package

Title Tools for 2D and 3D Plots of Single and Multi-Objective Linear/Integer Programming Models

Version 1.4.3

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

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

Description Make 2D and 3D plots of linear programming (LP), integer linear programming (ILP), or mixed integer linear programming (MILP) models with up to three objectives. Plots of both the solution and criterion space are possible. For instance the non-dominated (Pareto) set for bi-objective LP/ILP/MILP programming models (see vignettes for an overview).

License GPL (>= 3.3.2)

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Depends R (>= 3.5.0)

Imports ggrepel, geometry, ggplot2, rgl, MASS, Matrix, grDevices, stats, Rfast, plyr, purrr, dplyr, rlang

Suggests tikzDevice, grid, gridExtra, knitr, rmarkdown, roxygen2, testthat (>= 2.1.0)

VignetteBuilder knitr

NeedsCompilation no

Author Lars Relund Nielsen [aut, cre] (<https://orcid.org/0000-0002-4802-3071>)

Maintainer Lars Relund Nielsen <lars@relund.dk>

Repository CRAN

Date/Publication 2020-02-20 15:10:02 UTC

R topics documented:

`.checkPts` *Check if point input is okay*

Description

Check if point input is okay

Usage

```
.checkPts(pts, p = NULL, warn = FALSE)
```

Arguments

pts	Point input.
p	Desired dimension of points.
warn	Output warnings.

Value

Point input converted to a matrix.

---

addNDSet2D *Add 2D discrete points to a non-dominated set and classify them into extreme supported, non-extreme supported, non-supported.*

Description

Add 2D discrete points to a non-dominated set and classify them into extreme supported, non-extreme supported, non-supported.

Usage

```
addNDSet2D(points, nDSet = NULL, crit = "max", keepDom = FALSE)
```

Arguments

points	A data frame. It is assumed that z1 and z2 are in the two first columns.
nDSet	A data frame with current non-dominated set (NULL is none yet).
crit	Either max or min.
keepDom	Keep dominated points.

Value

A data frame with columns z1 and z2, nD (non-dominated), ext (extreme), nonExt (non-extreme supported).

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
nDSet <- data.frame(z1=c(12,14,16,18), z2=c(18,16,12,4))
points <- data.frame(z1 = c(18,18,14,15,15), z2=c(2,6,14,14,16))
addNDSet2D(points, nDSet, crit = "max")
addNDSet2D(points, nDSet, crit = "max", keepDom = TRUE)
addNDSet2D(points, nDSet, crit = "min")
```

addRays *Add all points on the bounding box hit by the rays.*

Description

Add all points on the bounding box hit by the rays.

Usage

```
addRays(
  pts,
  m = apply(pts, 2, min) - 5,
  M = apply(pts, 2, max) + 5,
  direction = 1
)
```

Arguments

pts	A data frame with all points
m	Minimum values of the bounding box.
M	Maximum vlaues of the bounding box.
direction	Ray direction. If i'th entry is positive, consider the i'th column of the pts plus a value greater than on equal zero. If negative, consider the i'th column of the pts minus a value greater than on equal zero.

Value

The points merged with the points on the bounding box. The column pt equals 1 if points from pts and zero otherwise.

Note

Assume that pts has been checked using .checkPts.

Examples

```
pts <- genNDSet(3,10)
addRays(pts)
addRays(pts, dir = c(1,-1,1))
addRays(pts, dir = c(-1,-1,1), m = c(0,0,0), M = c(100,100,100))
pts <- genSample(5,20)
addRays(pts)
```

`classifyNDSet` *Classify a set of nondominated points*

Description

The classification is supported (true/false), extreme (true/false), supported non-extreme (true/false)

Usage

```
classifyNDSet(pts, direction = 1)
```

Arguments

`pts`	A set of non-dominated points. It is assumed that `ncol(pts)` equals the number of objectives (p).
`direction`	Ray direction. If i'th entry is positive, consider the i'th column of the `pts` plus a value greater than on equal zero (minimize objective i). If negative, consider the i'th column of the `pts` minus a value greater than on equal zero (maximize objective i).

Value

The ND set with classification columns.

Note

It is assumed that `pts` are nondominated.

Examples

```
pts <- matrix(c(0,0,1, 0,1,0, 1,0,0, 0.5,0.2,0.5, 0.25,0.5,0.25), ncol = 3, byrow = TRUE)
ini3D(argsPlot3d = list(xlim = c(min(pts[,1])-2,max(pts[,1])+2),
  ylim = c(min(pts[,2])-2,max(pts[,2])+2),
  zlim = c(min(pts[,3])-2,max(pts[,3])+2)))
plotHull3D(pts, addRays = TRUE, argsPolygon3d = list(alpha = 0.5), useRGLBBox = TRUE)
pts <- classifyNDSet(pts[,1:3])
plotPoints3D(pts[pts$se,1:3], argsPlot3d = list(col = "red"))
plotPoints3D(pts[!pts$sne,1:3], argsPlot3d = list(col = "black"))
plotPoints3D(pts[!pts$us,1:3], argsPlot3d = list(col = "blue"))
plotCones3D(pts[,1:3], rectangle = TRUE, argsPolygon3d = list(alpha = 1))
finalize3D()
pts

pts <- matrix(c(0,0,1, 0,1,0, 1,0,0, 0.2,0.1,0.1, 0.1,0.45,0.45), ncol = 3, byrow = TRUE)
di <- -1 # maximize
ini3D(argsPlot3d = list(xlim = c(min(pts[,1])-1,max(pts[,1])+1),
  ylim = c(min(pts[,2])-1,max(pts[,2])+1),
  zlim = c(min(pts[,3])-1,max(pts[,3])+1)))
plotHull3D(pts, addRays = TRUE, argsPolygon3d = list(alpha = 0.5), direction = di,
```

```
           addText = "coord")
pts <- classifyNDSet(pts[,1:3], direction = di)
plotPoints3D(pts[pts$se,1:3], argsPlot3d = list(col = "red"))
plotPoints3D(pts[!pts$sne,1:3], argsPlot3d = list(col = "black"))
plotPoints3D(pts[!pts$us,1:3], argsPlot3d = list(col = "blue"))
plotCones3D(pts[,1:3], rectangle = TRUE, argsPolygon3d = list(alpha = 1), direction = di)
finalize3D()
pts

pts <- genNDSet(3,50)
ini3D(argsPlot3d = list(xlim = c(0,max(pts$x)+2),
  ylim = c(0,max(pts$y)+2),
  zlim = c(0,max(pts$z)+2)))
plotHull3D(pts, addRays = TRUE, argsPolygon3d = list(alpha = 0.5))
pts <- classifyNDSet(pts[,1:3])
plotPoints3D(pts[pts$se,1:3], argsPlot3d = list(col = "red"))
plotPoints3D(pts[!pts$sne,1:3], argsPlot3d = list(col = "black"))
plotPoints3D(pts[!pts$us,1:3], argsPlot3d = list(col = "blue"))
finalize3D()
pts
```

`convexHull` *Find the convex hull of a set of points.*

Description

Find the convex hull of a set of points.

Usage

```
convexHull(
  pts,
  addRays = FALSE,
  useRGLBBox = FALSE,
  direction = 1,
  tol = mean(mean(abs(pts))) * sqrt(.Machine$double.eps) * 2
)
```

Arguments

`pts`	A matrix with a point in each row.
`addRays`	Add the ray defined by `direction`.
`useRGLBBox`	Use the RGL bounding box when add rays.
`direction`	Ray direction. If i'th entry is positive, consider the i'th column of `pts` plus a value greater than on equal zero (minimize objective i). If negative, consider the i'th column of `pts` minus a value greater than on equal zero (maximize objective i).
`tol`	Tolerance on std. dev. if using PCA.

Value

A list with `hull` equal a matrix with row indices of the vertices defining each facet in the hull and `pts` equal the input points (and dummy points) and columns: `pt`, true if a point in the original input; false if a dummy point (a point on a ray). `vtx`, TRUE if a vertex in the hull.

Examples

```
## 1D
pts<-matrix(c(1,2,3), ncol = 1, byrow = TRUE)
dimFace(pts) # a line
convexHull(pts)
convexHull(pts, addRays = TRUE)

## 2D
pts<-matrix(c(1,1, 2,2), ncol = 2, byrow = TRUE)
dimFace(pts) # a line
convexHull(pts)
plotHull2D(pts, drawPoints = TRUE)
convexHull(pts, addRays = TRUE)
plotHull2D(pts, addRays = TRUE, drawPoints = TRUE)
pts<-matrix(c(1,1, 2,2, 0,1), ncol = 2, byrow = TRUE)
dimFace(pts) # a polygon
convexHull(pts)
plotHull2D(pts, drawPoints = TRUE)
convexHull(pts, addRays = TRUE, direction = c(-1,1))
plotHull2D(pts, addRays = TRUE, direction = c(-1,1), addText = "coord")

## 3D
pts<-matrix(c(1,1,1), ncol = 3, byrow = TRUE)
dimFace(pts) # a point
convexHull(pts)
pts<-matrix(c(0,0,0,1,1,1,2,2,2,3,3,3), ncol = 3, byrow = TRUE)
dimFace(pts) # a line
convexHull(pts)
pts<-matrix(c(0,0,0,0,1,1,0,2,2,0,0,2), ncol = 3, byrow = TRUE)
dimFace(pts) # a polygon
convexHull(pts)
convexHull(pts, addRays = TRUE)
pts<-matrix(c(1,0,0,1,1,1,1,2,2,3,1,1), ncol = 3, byrow = TRUE)
dimFace(pts) # a polygon
convexHull(pts) # a polyhedron
pts<-matrix(c(1,1,1,2,2,1,2,1,1,1,1,2), ncol = 3, byrow = TRUE)
dimFace(pts) # a polytope (polyhedron)
convexHull(pts)

ini3D(argsPlot3d = list(xlim = c(0,3), ylim = c(0,3), zlim = c(0,3)))
pts<-matrix(c(1,1,1,2,2,1,2,1,1,1,1,2), ncol = 3, byrow = TRUE)
plotPoints3D(pts)
plotHull3D(pts, argsPolygon3d = list(color = "red"))
convexHull(pts)
plotHull3D(pts, addRays = TRUE)
convexHull(pts, addRays = TRUE)
```

```
finalize3D()
```

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

Description

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

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.

dimFace *Return the dimension of the convex hull of a set of points.*

Description

Return the dimension of the convex hull of a set of points.

Usage

```
dimFace(pts, dim = NULL)
```

Arguments

pts	A matrix/data frame/vector that can be converted to a matrix with a row for each point.
dim	The dimension of the points, i.e. assume that column 1-dim specify the points. If NULL assume that the dimension are the number of columns.

Value

The dimension of the object.

Examples

```
## In 1D
pts <- matrix(c(3), ncol = 1, byrow = TRUE)
dimFace(pts)
pts <- matrix(c(1,3,4), ncol = 1, byrow = TRUE)
dimFace(pts)

## In 2D
pts <- matrix(c(3,3,6,3,3,6), ncol = 2, byrow = TRUE)
dimFace(pts)
pts <- matrix(c(1,1,2,2,3,3), ncol = 2, byrow = TRUE)
dimFace(pts)
pts <- matrix(c(0,0), ncol = 2, byrow = TRUE)
dimFace(pts)

## In 3D
pts <- c(3,3,3,6,3,3,3,6,3,6,6,3)
dimFace(pts, dim = 3)
pts <- matrix( c(1,1,1), ncol = 3, byrow = TRUE)
dimFace(pts)
pts <- matrix( c(1,1,1,2,2,2), ncol = 3, byrow = TRUE)
dimFace(pts)
pts <- matrix(c(2,2,2,3,2,2), ncol=3, byrow= TRUE)
dimFace(pts)
pts <- matrix(c(0,0,0,0,1,1,0,2,2,0,5,2,0,6,1), ncol = 3, byrow = TRUE)
dimFace(pts)
```

```
pts <- matrix(c(0,0,0,0,1,1,0,2,2,0,0,2,1,1,1), ncol = 3, byrow = TRUE)
dimFace(pts)

## In 4D
pts <- matrix(c(2,2,2,3,2,2,3,4,1,2,3,4), ncol=4, byrow= TRUE)
dimFace(pts,)
```

`finalize3D` *Finalize the rgl window.*

Description

Finalize the rgl window.

Usage

```
finalize3D(...)
```

Arguments

`...`	Further arguments passed on the the rgl plotting functions. This must be done as lists. Currently the following arguments are supported:

- `argsAxes3d`: A list of arguments for `rgl::axes3d`.
- `argsTitle3d`: A list of arguments for `rgl::title3d`.

Value

NULL (invisible).

Examples

```
ini3D()
pts<-matrix(c(1,1,1,5,5,5), ncol = 3, byrow = TRUE)
plotPoints3D(pts)
finalize3D()
```

genNDSet *Generate a sample of nondominated points.*

Description

Generate a sample of nondominated points.

Usage

```
genNDSet(
  p,
  n,
  range = c(1, 100),
  random = FALSE,
  sphere = TRUE,
  box = FALSE,
  keep = FALSE,
  ...
)
```

Arguments

p	Dimension of the points.
n	Number of samples generated.
range	The range of the points in each dimension (a vector or matrix with p rows).
random	Random sampling.
sphere	Generate points on a sphere.
box	Generate points in boxes.
keep	Keep dominated points also.
...	Further arguments passed on to genSample.

Value

A data frame with p+1 columns (last one indicate if dominated or not).

Examples

```
range <- matrix(c(1,100, 50,100, 10,50), ncol = 2, byrow = TRUE )
ini3D()
pts <- genNDSet(3, 800, range = range, random = TRUE, keep = TRUE)
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plotPoints3D(pts)
plotPoints3D(pts[!pts$dom,], argsPlot3d = list(col = "red", size = 10))
finalize3D()
```

```
ini3D()
range <- c(1,100)
cent <- rep(range[1] + (range[2]-range[1])/2, 3)
pts <- genNDSet(3, 800, range = range, sphere = TRUE, keep = TRUE,
       argsSphere = list(center = cent))
rgl::spheres3d(cent, radius=49.5, color = "grey100", alpha=0.1)
plotPoints3D(pts)
plotPoints3D(pts[!pts$dom,], argsPlot3d = list(col = "red", size = 10))
rgl::planes3d(cent[1],cent[2],cent[3],-sum(cent^2), alpha = 0.5, col = "red")
finalize3D()

ini3D()
cent <- c(100,100,100)
r <- 75
planeC <- c(cent+r/3)
planeC <- c(planeC, -sum(planeC^2))
pts <- genNDSet(3, 100, keep = TRUE,
  argsSphere = list(center = cent, radius = r, below = FALSE, plane = planeC, factor = 6))
rgl::spheres3d(cent, radius=r, color = "grey100", alpha=0.1)
plotPoints3D(pts)
plotPoints3D(pts[!pts$dom,], argsPlot3d = list(col = "red", size = 10))
rgl::planes3d(planeC[1],planeC[2],planeC[3],planeC[4], alpha = 0.5, col = "red")
finalize3D()
```

`genSample` *Generate a sample of points in dimension p.*

Description

Generate a sample of points in dimension p.

Usage

```
genSample(
  p,
  n,
  range = c(1, 100),
  random = FALSE,
  sphere = TRUE,
  box = FALSE,
  ...
)
```

Arguments

`p`	Dimension of the points.
`n`	Number of samples generated.
`range`	The range of the points in each dimension (a vector or matrix with `p` rows).

`random` Random sampling.

`sphere` Generate points on a sphere.

`box` Generate points in boxes.

`...` Further arguments passed on to the method for generating points. This must be done as lists (see examples). Currently the following arguments are supported:

- `argsSphere`: A list of arguments for generating points on a sphere:
  - `radius`: The radius of the sphere.
  - `center`: The center of the sphere.
  - `plane`: The plane used.
  - `below`: Either true (generate points below the plane), false (generate points above the plane) or `NULL` (generated on the whole sphere).
  - `factor`: If using a plane. Then the factor multiply `n` with so generate enough points below/above the plane.
- `argsBox`: A list of arguments for generating points inside boxes:
  - `intervals`: Number of intervals to split the length of the range into. That is, each range is divided into `intervals` (sub)intervals and only the lowest/highest subrange is used.
  - `cor`: How to correlate indices. If `'idxAlt'` then alternate the intervals (high/low) for each dimension. For instance if `p = 3` and the first dimension is in the high interval range then the second will be in the low interval range and third in the high interval range again. If `idxRand` then choose the low/high interval range for each dimension based on `prHigh`. If `idxSplit` then select `floor(p/2):ceiling(p/2)` dimensions for the high interval range and the other for the low interval range.
  - `prHigh`: Probability for choosing the high interval range in each dimension.

Details

Note having ranges with different length when using the sphere method, doesn't make sense. The best option is properly to use a center and radius here. Moreover, as for higher `p` you may have to use a larger radius than half of the desired interval range.

Value

A data frame with `p` columns

Examples

```
### Using random
## p = 2
range <- matrix(c(1,100, 50,100), ncol = 2, byrow = TRUE )
pts <- genSample(2, 1000, range = range, random = TRUE)
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plot(pts)
```

```
## p = 3
range <- matrix(c(1,100, 50,100, 10,50), ncol = 2, byrow = TRUE )
ini3D()
pts <- genSample(3, 1000, range = range, random = TRUE)
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plotPoints3D(pts)
finalize3D()

## other p
p <- 10
range <- c(1,100)
pts <- genSample(p, 1000, range = range, random = TRUE)
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))


### Using sphere
## p = 2
range <- c(1,100)
cent <- rep(range[1] + (range[2]-range[1])/2, 2)
pts <- genSample(2, 1000, range = range)
dim(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plot(pts, asp=1)
abline(sum(cent^2)/cent[1], -cent[2]/cent[1])

cent <- c(100,100)
r <- 75
planeC <- c(cent+r/3)
planeC <- c(planeC, -sum(planeC^2))
pts <- genSample(2, 100,
  argsSphere = list(center = cent, radius = r, below = FALSE, plane = planeC, factor = 6))
dim(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plot(pts, asp=1)
abline(-planeC[3]/planeC[1], -planeC[2]/planeC[1])

pts <- genSample(2, 100, argsSphere = list(center = cent, radius = r, below = NULL))
dim(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plot(pts, asp=1)

## p = 3
ini3D()
range <- c(1,100)
cent <- rep(range[1] + (range[2]-range[1])/2, 3)
pts <- genSample(3, 1000, range = range)
dim(pts)
Rfast::colMinsMaxs(as.matrix(pts))
rgl::spheres3d(cent, radius=49.5, color = "grey100", alpha=0.1)
plotPoints3D(pts)
rgl::planes3d(cent[1],cent[2],cent[3],-sum(cent^2), alpha = 0.5, col = "red")
```

```
finalize3D()

ini3D()
cent <- c(100,100,100)
r <- 75
planeC <- c(cent+r/3)
planeC <- c(planeC, -sum(planeC^2))
pts <- genSample(3, 100,
  argsSphere = list(center = cent, radius = r, below = FALSE, plane = planeC, factor = 6))
rgl::spheres3d(cent, radius=r, color = "grey100", alpha=0.1)
plotPoints3D(pts)
rgl::planes3d(planeC[1],planeC[2],planeC[3],planeC[4], alpha = 0.5, col = "red")
finalize3D()

ini3D()
pts <- genSample(3, 10000, argsSphere = list(center = cent, radius = r, below = NULL))
Rfast::colMinsMaxs(as.matrix(pts))
rgl::spheres3d(cent, radius=r, color = "grey100", alpha=0.1)
plotPoints3D(pts)
finalize3D()

## Other p
p <- 10
cent <- rep(0,p)
r <- 100
pts <- genSample(p, 100000, argsSphere = list(center = cent, radius = r, below = NULL))
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
apply(pts,1, function(x){sqrt(sum((x-cent)^2))}) # test should be approx. equal to radius


### Using box
## p = 2
range <- matrix(c(1,100, 50,100), ncol = 2, byrow = TRUE )
pts <- genSample(2, 1000, range = range, box = TRUE, argsBox = list(cor = "idxAlt"))
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plot(pts)

pts <- genSample(2, 1000, range = range, box = TRUE, argsBox = list(cor = "idxAlt",
                 intervals = 6))
plot(pts)

pts <- genSample(2, 1000, range = range, box = TRUE, argsBox = list(cor = "idxRand"))
plot(pts)
pts <- genSample(2, 1000, range = range, box = TRUE,
                 argsBox = list(cor = "idxRand", prHigh = c(0.1,0.6)))
points(pts, pch = 3, col = "red")
pts <- genSample(2, 1000, range = range, box = TRUE,
                 argsBox = list(cor = "idxRand", prHigh = c(0,0)))
points(pts, pch = 4, col = "blue")

pts <- genSample(2, 1000, range = range, box = TRUE, argsBox = list(cor = "idxSplit"))
```

```
plot(pts)

## p = 3
range <- matrix(c(1,100, 1,200, 1,50), ncol = 2, byrow = TRUE )
ini3D(argsPlot3d = list(box = TRUE, axes = TRUE))
pts <- genSample(3, 1000, range = range, box = TRUE, , argsBox = list(cor = "idxAlt"))
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
plotPoints3D(pts)
finalize3D()

ini3D(argsPlot3d = list(box = TRUE, axes = TRUE))
pts <- genSample(3, 1000, range = range, box = TRUE, ,
                 argsBox = list(cor = "idxAlt", intervals = 6))
plotPoints3D(pts)
finalize3D()

ini3D(argsPlot3d = list(box = TRUE, axes = TRUE))
pts <- genSample(3, 1000, range = range, box = TRUE, , argsBox = list(cor = "idxRand"))
plotPoints3D(pts)
pts <- genSample(3, 1000, range = range, box = TRUE, ,
                 argsBox = list(cor = "idxRand", prHigh = c(0.1,0.6,0.1)))
plotPoints3D(pts, argsPlot3d = list(col="red"))
finalize3D()

ini3D(argsPlot3d = list(box = TRUE, axes = TRUE))
pts <- genSample(3, 1000, range = range, box = TRUE, , argsBox = list(cor = "idxSplit"))
plotPoints3D(pts)
finalize3D()

## other p
p <- 10
range <- c(1,100)
pts <- genSample(p, 1000, range = range, box = TRUE, argsBox = list(cor = "idxSplit"))
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
```

`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 p dimensions.*

Description

Efficient test for points inside a convex hull in p dimensions.

Usage

```
inHull(
  pts,
  vertices,
  hull = NULL,
  tol = mean(mean(abs(as.matrix(vertices)))) * sqrt(.Machine$double.eps)
)
```

Arguments

pts	A nxp array to test, n data points, in dimension p. 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.
hull	Tessellation (or triangulation) generated by convhulln (only works if the dimension of the hull is p). If hull is NULL, 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. 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 with values 1 (inside hull), -1 (outside hull) or 0 (on hull to precision indicated by tol).

Note

Some of the code are inspired by the Matlab code by John D'Errico http://www.mathworks.com/matlabcentral/fileexchange/1
inhull and https://tolstoy.newcastle.edu.au/R/e8/help/09/12/8784.html. If the dimension of the hull is below p then PCA may be used to check (a warning will be given).

Author(s)

Lars Relund <lars@relund.dk>

Examples

```
## In 1D
vertices <- matrix(4, ncol = 1)
pt <- matrix(c(2,4), ncol = 1, byrow = TRUE)
inHull(pt, vertices)
vertices <- matrix(c(1,4), ncol = 1)
pt <- matrix(c(1,3,4,5), ncol = 1, byrow = TRUE)
inHull(pt, vertices)

## In 2D
vertices <- matrix(c(2,4), ncol = 2)
pt <- matrix(c(2,4, 1,1), ncol = 2, byrow = TRUE)
inHull(pt, vertices)
vertices <- matrix(c(0,0, 3,3), ncol = 2, byrow = TRUE)
pt <- matrix(c(0,0, 1,1, 2,2, 3,3, 4,4), ncol = 2, byrow = TRUE)
inHull(pt, vertices)
vertices <- matrix(c(0,0, 0,3, 3,0), ncol = 2, byrow = TRUE)
pt <- matrix(c(0,0, 1,1, 4,4), ncol = 2, byrow = TRUE)
inHull(pt, vertices)

## in 3D
vertices <- matrix(c(2,2,2), ncol = 3, byrow = TRUE)
pt <- matrix(c(1,1,1, 3,3,3, 2,2,2, 3,3,2), ncol = 3, byrow = TRUE)
inHull(pt, vertices)

vertices <- matrix(c(2,2,2, 4,4,4), ncol = 3, byrow = TRUE)
ini3D()
plotHull3D(vertices)
pt <- matrix(c(1,1,1, 2,2,2, 3,3,3, 4,4,4, 3,3,2), ncol = 3, byrow = TRUE)
plotPoints3D(pt, addText = TRUE)
finalize3D()
inHull(pt, vertices)

vertices <- matrix(c(1,0,0, 1,1,0, 1,0,1), ncol = 3, byrow = TRUE)
ini3D()
plotHull3D(vertices)
pt <- matrix(c(1,0.1,0.2, 3,3,2), ncol = 3, byrow = TRUE)
```

```
plotPoints3D(pt, addText = TRUE)
finalize3D()
inHull(pt, vertices)

vertices <- matrix(c(2,2,2, 2,4,4, 2,2,4, 4,4,2, 4,2,2, 2,4,2, 4,2,4, 4,4,4), ncol = 3,
           byrow = TRUE)
ini3D()
plotHull3D(vertices)
pt <- matrix(c(1,1,1, 3,3,3, 2,2,2, 3,3,2), ncol = 3, byrow = TRUE)
plotPoints3D(pt, addText = TRUE)
finalize3D()
inHull(pt, vertices)

## In 5D
vertices <- matrix(c(4,0,0,0,0, 0,4,0,0,0, 0,0,4,0,0, 0,0,0,4,0, 0,0,0,0,4, 0,0,0,0,0),
           ncol = 5, byrow = TRUE)
pt <- matrix(c(0.1,0.1,0.1,0.1,0.1, 3,3,3,3,3, 2,0,0,0,0), ncol = 5, byrow = TRUE)
inHull(pt, vertices)
```

`ini3D` *Initialize the rgl window.*

Description

Initialize the rgl window.

Usage

```
ini3D(new = FALSE, clear = TRUE, ...)
```

Arguments

`new`	A new window is opened (otherwise the current is cleared).
`clear`	Clear the current rgl window.
`...`	Further arguments passed on the the rgl plotting functions. This must be done as lists. Currently the following arguments are supported: • `argsPlot3d`: A list of arguments for `rgl::plot3d`. • `argsAspect3d`: A list of arguments for `rgl::aspect3d`.

Value

NULL (invisible).

Examples

```
ini3D()
pts<-matrix(c(1,1,1,5,5,5), ncol = 3, byrow = TRUE)
plotPoints3D(pts)
finalize3D()
```

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.

plotCones3D *Plot a cone defined by a point in 3D.*

Description

The cones are defined as the point plus R3+.

Usage

```
plotCones3D(
  pts,
  drawPoint = TRUE,
  drawLines = TRUE,
  drawPolygons = TRUE,
  direction = 1,
  rectangle = FALSE,
  useRGLBBox = TRUE,
  ...
)
```

Arguments

pts	A matrix with a point in each row.
drawPoint	Draw the points defining the cone.
drawLines	Draw lines of the cone.
drawPolygons	Draw polygons of the cone.

direction	Ray direction. If i'th entry is positive, consider the i'th column of pts plus a value greater than on equal zero (minimize objective i). If negative, consider the i'th column of pts minus a value greater than on equal zero (maximize objective i).
rectangle	Draw the cone as a rectangle.
useRGLBBox	Use the RGL bounding box as ray limits for the cone.
...	Further arguments passed on the the rgl plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: • argsPlot3d: A list of arguments for rgl::plot3d. • argsSegments3d: A list of arguments for rgl::segments3d. • argsPolygon3d: A list of arguments for rgl::polygon3d.

Value

NULL (invisible)

Examples

```
ini3D(argsPlot3d = list(xlim = c(0,6), ylim = c(0,6), zlim = c(0,6)))
plotCones3D(c(4,4,4), drawLines = FALSE, drawPoint = TRUE,
           argsPlot3d = list(col = "red", size = 10),
           argsPolygon3d = list(alpha = 1), rectangle = TRUE)
plotCones3D(c(1,1,1), rectangle = FALSE)
plotCones3D(matrix(c(3,3,3,2,2,2), ncol = 3, byrow = TRUE))
finalize3D()

ini3D(argsPlot3d = list(xlim = c(0,6), ylim = c(0,6), zlim = c(0,6)))
plotCones3D(c(4,4,4), direction = 1)
plotCones3D(c(2,2,2), direction = -1)
plotCones3D(c(4,2,2), direction = c(1,-1,-1))
plotCones3D(c(2,2,4), direction = c(-1,-1,1))
finalize3D()
```

---

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")
```

`plotHull2D` *Plot the convex hull of a set of points in 3D.*

Description

Plot the convex hull of a set of points in 3D.

Usage

```
plotHull2D(
  pts,
  drawPoints = FALSE,
  drawLines = TRUE,
  drawPolygons = TRUE,
  addText = FALSE,
  addRays = FALSE,
  direction = 1,
  latex = FALSE,
  ...
)
```

Arguments

`pts`	A matrix with a point in each row.
`drawPoints`	Draw the points.
`drawLines`	Draw lines of the facets.
`drawPolygons`	Fill the hull.
`addText`	Add text to the points. Currently `coord` (coordinates), `rownames` (rownames) and `both` supported or a vector with text.
`addRays`	Add the ray defined by `direction`.
`direction`	Ray direction. If i'th entry is positive, consider the i'th column of `pts` plus a value greater than on equal zero (minimize objective i). If negative, consider the i'th column of `pts` minus a value greater than on equal zero (maximize objective i).
`latex`	If `True` make latex math labels for TikZ.
`...`	Further arguments passed on the the ggplot plotting functions. This must be done as lists. Currently the following arguments are supported: • `argsGeom_point`: A list of arguments for `ggplot2::geom_point`.

- argsGeom_path: A list of arguments for ggplot2::geom_path.
- argsGeom_polygon: A list of arguments for ggplot2::geom_polygon.
- argsGeom_label: A list of arguments for ggplot2::geom_label.

Value

The ggplot.

Examples

```
pts<-matrix(c(1,1), ncol = 2, byrow = TRUE)
plotHull2D(pts)
pts<-matrix(c(1,1, 2,2), ncol = 2, byrow = TRUE)
plotHull2D(pts, drawPoints = TRUE)
plotHull2D(pts, drawPoints = TRUE, addRays = TRUE, addText = "coord")
plotHull2D(pts, drawPoints = TRUE, addRays = TRUE, direction = -1, addText = "coord")
pts<-matrix(c(1,1, 2,2, 0,1), ncol = 2, byrow = TRUE)
plotHull2D(pts, drawPoints = TRUE, addText = "coord")
plotHull2D(pts, drawPoints = TRUE, addRays = TRUE, addText = "coord")
plotHull2D(pts, drawPoints = TRUE, addRays = TRUE, direction = -1, addText = "coord")
```

---

plotHull3D *Plot the convex hull of a set of points in 3D.*

Description

Plot the convex hull of a set of points in 3D.

Usage

```
plotHull3D(
  pts,
  drawPoints = FALSE,
  drawLines = TRUE,
  drawPolygons = TRUE,
  addText = FALSE,
  addRays = FALSE,
  useRGLBBox = TRUE,
  direction = 1,
  drawBBoxHull = TRUE,
  ...
)
```

Arguments

pts	A matrix with a point in each row.
drawPoints	Draw the points.
drawLines	Draw lines of the facets.
drawPolygons	Fill the facets.
addText	Add text to the points. Currently coord (coordinates), rownames (rownames) and both supported or a vector with text.
addRays	Add the ray defined by direction.
useRGLBBox	Use the RGL bounding box when add rays.
direction	Ray direction. If i'th entry is positive, consider the i'th column of pts plus a value greater than on equal zero (minimize objective i). If negative, consider the i'th column of pts minus a value greater than on equal zero (maximize objective i).
drawBBoxHull	If addRays then draw the hull areas hitting the bounding box also.
...	Further arguments passed on the the rgl plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: • argsPlot3d: A list of arguments for rgl::plot3d. • argsSegments3d: A list of arguments for rgl::segments3d. • argsPolygon3d: A list of arguments for rgl::polygon3d. • argsShade3d: A list of arguments for rgl::shade3d. • argsText3d: A list of arguments for rgl::text3d.

Value

The convex hull (invisible).

Examples

```
ini3D()
pts<-matrix(c(0,0,0), ncol = 3, byrow = TRUE)
plotHull3D(pts) # a point
pts<-matrix(c(1,1,1,2,2,2,3,3,3), ncol = 3, byrow = TRUE)
plotHull3D(pts, drawPoints = TRUE) # a line
pts<-matrix(c(1,0,0,1,1,1,1,2,2,3,1,1,3,3,3), ncol = 3, byrow = TRUE)
plotHull3D(pts, drawLines = FALSE, argsPolygon3d = list(alpha=0.6)) # a polygon
pts<-matrix(c(5,5,5,10,10,5,10,5,5,5,5,10), ncol = 3, byrow = TRUE)
plotHull3D(pts, argsPolygon3d = list(alpha=0.9), argsSegments3d = list(color="red"))
finalize3D()

## Using addRays
pts <- data.frame(x = c(1,3), y = c(1,3), z = c(1,3))
ini3D(argsPlot3d = list(xlim = c(0,max(pts$x)+10),
  ylim = c(0,max(pts$y)+10),
  zlim = c(0,max(pts$z)+10)))
plotHull3D(pts, drawPoints = TRUE, addRays = TRUE, , drawBBoxHull = FALSE)
plotHull3D(c(4,4,4), drawPoints = TRUE, addRays = TRUE)
```

```
finalize3D()

pts <- data.frame(x = c(4,2.5,1), y = c(1,2.5,4), z = c(1,2.5,4))
ini3D(argsPlot3d = list(xlim = c(0,max(pts$x)+10),
  ylim = c(0,max(pts$y)+10),
  zlim = c(0,max(pts$z)+10)))
plotHull3D(pts, drawPoints = TRUE, addRays = TRUE)
finalize3D()

pts <- matrix(c(
  0, 4, 8,
  0, 8, 4,
  8, 4, 0,
  4, 8, 0,
  4, 0, 8,
  8, 0, 4,
  4, 4, 4,
  6, 6, 6
  ), ncol = 3, byrow = TRUE)
ini3D(FALSE, argsPlot3d = list(xlim = c(min(pts[,1])-2,max(pts[,1])+10),
  ylim = c(min(pts[,2])-2,max(pts[,2])+10),
  zlim = c(min(pts[,3])-2,max(pts[,3])+10)))
plotHull3D(pts, drawPoints = TRUE, addText = "coord")
plotHull3D(pts, addRays = TRUE)
finalize3D()


pts <- genNDSet(3, 100)
pts <- as.data.frame(pts)

ini3D(argsPlot3d = list(
  xlim = c(0,max(pts$x)+10),
  ylim = c(0,max(pts$y)+10),
  zlim = c(0,max(pts$z)+10)))
plotHull3D(pts, drawPoints = TRUE, addRays = TRUE)
finalize3D()

ini3D(argsPlot3d = list(xlim = c(0,max(pts[,1])+10),
  ylim = c(0,max(pts[,2])+10),
  zlim = c(0,max(pts[,3])+10)))
plotHull3D(pts, drawPoints = TRUE, drawPolygons = TRUE, addText = "coord", addRays = TRUE)
finalize3D()

ini3D(argsPlot3d = list(xlim = c(0,max(pts$x)+10),
  ylim = c(0,max(pts$y)+10),
  zlim = c(0,max(pts$z)+10)))
plotHull3D(pts, drawPoints = TRUE, drawLines = FALSE,
  argsPolygon3d = list(alpha = 1), addRays = TRUE)
finalize3D()

ini3D(argsPlot3d = list(xlim = c(0,max(pts$x)+2),
  ylim = c(0,max(pts$y)+2),
  zlim = c(0,max(pts$z)+2)))
```

```
plotHull3D(pts, drawPoints = TRUE, argsPolygon3d = list(color = "red"), addRays = TRUE)
plotCones3D(pts, argsPolygon3d = list(alpha = 1), rectangle = TRUE)
finalize3D()
```

`plotNDSet2D` *Create a plot of a discrete non-dominated set.*

Description

Create a plot of a discrete non-dominated set.

Usage

```
plotNDSet2D(
  points,
  crit,
  addTriangles = FALSE,
  addHull = TRUE,
  latex = FALSE,
  labels = NULL
)
```

Arguments

`points`	Data frame with non-dominated points.
`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.
`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

```
dat <- data.frame(z1=c(12,14,16,18,18,18,14,15,15), z2=c(18,16,12,4,2,6,14,14,16))
points <- addNDSet2D(dat, crit = "min", keepDom = TRUE)
plotNDSet2D(points, crit = "min", addTriangles = TRUE)
points <- addNDSet2D(dat, crit = "max", keepDom = TRUE)
plotNDSet2D(points, crit = "max", addTriangles = TRUE)
```

`plotPlane3D` *Plot a plane in 3D.*

Description

Plot a plane in 3D.

Usage

```
plotPlane3D(normal, point = NULL, offset = 0, ...)
```

Arguments

`normal`	Normal to the plane.
`point`	A point on the plane.
`offset`	The offset of the plane (only used if `point` = `NULL`).
`...`	Further arguments passed on the the rgl plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: • `argsPlanes3d`: A list of arguments for rgl::planes3d.

Value

NULL (invisible)

Examples

```
ini3D(argsPlot3d = list(xlim = c(-1,10), ylim = c(-1,10), zlim = c(-1,10)) )
plotPlane3D(c(1,1,1), point = c(1,1,1))
plotPoints3D(c(1,1,1))
plotPlane3D(c(1,2,1), point = c(2,2,2), argsPlanes3d = list(col="red"))
plotPoints3D(c(2,2,2))
plotPlane3D(c(2,1,1), offset = -6, argsPlanes3d = list(col="blue"))
plotPlane3D(c(2,1,1), argsPlanes3d = list(col="green"))
finalize3D()
```

plotPoints3D *Plot points in 3D.*

Description

Plot points in 3D.

Usage

```
plotPoints3D(pts, addText = F, ...)
```

Arguments

pts	A vector or matrix with the points.
addText	Add text to the points. Currently coord (coordinates), rownames (rownames) and both supported or a vector with the text.
...	Further arguments passed on the the rgl plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: • argsPlot3d: A list of arguments for rgl::plot3d. • argsPch3d: A list of arguments for rgl::pch3d. • argsText3d: A list of arguments for rgl::text3d.

Value

NULL (invisible)

Examples

```
ini3D()
pts<-matrix(c(1,1,1,5,5,5), ncol = 3, byrow = TRUE)
plotPoints3D(pts)
plotPoints3D(c(2,3,3), argsPlot3d = list(col = "red", size = 10))
plotPoints3D(c(3,2,3), argsPlot3d = list(col = "blue", size = 10, type="p"))
plotPoints3D(c(1.5,1.5,1.5), argsPlot3d = list(col = "blue", size = 10, type="p"))
plotPoints3D(c(2,2,2, 1,1,1), addText = "coord")
plotPoints3D(c(3,3,3, 4,4,4), addText = "rownames")
finalize3D()
```

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 (i.e. no extreme rays) 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>

---

plotRectangle3D *Plot a rectangle defined by two corner points.*

Description

The rectangle is defined by xla <= x <= b where a is the minimum values and b is the maximum values.

Usage

```
plotRectangle3D(a, b, ...)
```

Arguments

a	A vector of length 3.
b	A vector of length 3.
...	Further arguments passed on the the rgl plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: • argsPlot3d: A list of arguments for rgl::plot3d. • argsSegments3d: A list of arguments for rgl::segments3d. • argsPolygon3d: A list of arguments for rgl::polygon3d. • argsShade3d: A list of arguments for rgl::shade3d.

Value

The corner points of the rectangle (invisible).

Examples

```
ini3D()
plotRectangle3D(c(0,0,0), c(1,1,1))
plotRectangle3D(c(1,1,1), c(4,4,3), drawPoints = TRUE, drawLines = FALSE,
           argsPlot3d = list(size=2, type="s", alpha=0.3))
plotRectangle3D(c(2,2,2), c(3,3,2.5), argsPolygon3d = list(alpha = 1) )
finalize3D()
```

---

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