

# Package ‘gMOIP’

January 26, 2023

**Type** Package

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

**Version** 1.4.8

**URL** <https://relund.github.io/gMOIP/>, <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). The package also contains an function for checking if a point  
is inside the convex hull.

**License** GPL (>= 3.3.2)

**Language** en-US

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Depends** R (>= 3.5.0)

**Imports** ggrepel, geometry, ggplot2, rgl, MASS, Matrix, grDevices,  
stats, Rfast, plyr, tidymodels, tidyr, tibble, purrr, dplyr,  
rlang, png, sp

**Suggests** tikzDevice, grid, gridExtra, knitr, rmarkdown, roxygen2,  
ggsci, tidyverse, magrittr, scales, pdftools, testthat (>=  
2.1.0), webshot2

**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** 2023-01-26 16:30:02 UTC

**R topics documented:**

addNDSet	3
addRays	4
binaryPoints	5
classifyNDSet	6
convexHull	7
cornerPoints	9
cornerPointsCont	10
criterionPoints	11
df2String	12
dimFace	12
finalize3D	13
genNDSet	14
genSample	16
getTexture	20
gMOIPTheme	21
hullSegment	22
inHull	22
ini3D	24
integerPoints	25
loadView	26
mergeLists	27
plotCones2D	27
plotCones3D	29
plotCriterion2D	30
plotHull2D	35
plotHull3D	37
plotMTeX3D	40
plotNDSet2D	40
plotPlane3D	42
plotPoints3D	43
plotPolygon3D	44
plotPolytope	46
plotPolytope2D	54
plotPolytope3D	55
plotRectangle3D	57
plotTeX3D	58
plotTitleTeX3D	59
pngSize	60
saveView	61
slices	62
texToPng	63

---

addNDSet	<i>Add discrete points to a non-dominated set and classify them into extreme supported, non-extreme supported, non-supported.</i>
----------	---

---

### Description

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

### Usage

```
addNDSet(
  pts,
  nDSet = NULL,
  crit = "max",
  keepDom = FALSE,
  dubND = FALSE,
  classify = TRUE
)
```

### Arguments

pts	A data frame with points to add (a column for each objective).
nDSet	A data frame with current non-dominated set (NULL is none yet). Column names of the p objectives must be z1, ..., zp.
crit	A max or min vector. If length one assume all objectives are optimized in the same direction.
keepDom	Keep dominated points in output.
dubND	Duplicated non-dominated points are classified as non-dominated.
classify	Non-dominated points are classified into supported extreme (se), supported non-extreme (sne) and unsupported (us)

### Value

A data frame with a column for each objective (z columns) and nd (non-dominated). Moreover if classify then columns se, sne, us and cls.

### Author(s)

Lars Relund <lars@relund.dk>

**Examples**

```

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

addNDSet(c(2,2), crit = "min")

nDSet <- data.frame(z1=c(12,14,16,18), z2=c(18,16,12,4), z3 = c(1,7,0,6))
pts <- data.frame(z1=c(12,14,16,18), z2=c(18,16,12,4), z3 = c(2,2,2,6))
crit = c("min", "min", "max")
di <- c(1,1,-1)
li <- c(-1,20)
ini3D(argsPlot3d = list(xlim = li, ylim = li, zlim = li))
plotCones3D(nDSet, direction = di, argsPolygon3d = list(color = "green", alpha = 1),
            drawPoint = FALSE)
plotHull3D(nDSet, addRays = TRUE, direction = di)
plotPoints3D(nDSet, argsPlot3d = list(col = "red"), addText = "coord")
plotPoints3D(pts, addText = "coord")
finalize3D()
addNDSet(pts, nDSet, crit, dubND = FALSE)
addNDSet(pts, nDSet, crit, dubND = TRUE)
addNDSet(pts, nDSet, crit, dubND = TRUE, keepDom = TRUE)
addNDSet(pts, nDSet, crit, dubND = TRUE, keepDom = TRUE, classify = FALSE)

```

---

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 values 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)[,1:3]
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)[,1:5]
addRays(pts)
```

---

binaryPoints

*Binary (0-1) points in the feasible region ( $Ax \leq b$ ).*

---

**Description**

Binary (0-1) points in the feasible region ( $Ax \leq b$ ).

**Usage**

```
binaryPoints(A, b)
```

**Arguments**

A	Constraint matrix.
b	Right hand side.

**Value**

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

**Note**

Do a simple enumeration of all binary points. Will not work if `ncol(A)` large.

**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)
binaryPoints(A, b)
```

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

---

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**

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

**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 <- matrix(c(0,0,1, 0,0,1, 0,1,0, 0.5,0.2,0.5, 1,0,0, 0.5,0.2,0.5, 0.25,0.5,0.25), ncol = 3,
  byrow = TRUE)
classifyNDSet(pts)

pts <- genNDSet(3,50)[,1:3]
ini3D(argsPlot3d = list(xlim = c(0,max(pts$z1)+2),
  ylim = c(0,max(pts$z2)+2),
  zlim = c(0,max(pts$z3)+2)))
plotHull3D(pts, addRays = TRUE, argsPolygon3d = list(alpha = 0.5))
pts <- classifyNDSet(pts[,1:3])
plotPoints3D(pts[pts$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

```

**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,
  m = apply(pts, 2, min) - 5,
  M = apply(pts, 2, max) + 5
)
```

**Arguments**

<code>pts</code>	A matrix with a point in each row.
<code>addRays</code>	Add the ray defined by <code>direction</code> .
<code>useRGLBBox</code>	Use the RGL bounding box when add rays.
<code>direction</code>	Ray direction. If <i>i</i> 'th entry is positive, consider the <i>i</i> 'th column of <code>pts</code> plus a value greater than on equal zero (minimize objective $i$ ). If negative, consider the <i>i</i> 'th column of <code>pts</code> minus a value greater than on equal zero (maximize objective $i$ ).
<code>tol</code>	Tolerance on standard deviation if using PCA.
<code>m</code>	Minimum values of the bounding box.
<code>M</code>	Maximum values of the bounding box.

**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  $Ax \leq b$ .*


---

### Description

Calculate the corner points for the polytope  $Ax \leq b$ .

### 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	<i>Calculate the corner points for the polytope <math>Ax \leq b</math> assuming all variables are continuous.</i>
------------------	---

---

**Description**

Calculate the corner points for the polytope  $Ax \leq b$  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 &lt;lars@relund.dk&gt;

---

criterionPoints	<i>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.</i>
-----------------	---

---

**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(pts, obj, crit, labels = "coord")
```

**Arguments**

pts	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 equals 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 (

**Author(s)**

Lars Relund &lt;lars@relund.dk&gt;

**Examples**

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

---

df2String	<i>Convert each row to a string.</i>
-----------	--------------------------------------

---

**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	<i>Return the dimension of the convex hull of a set of points.</i>
---------	--

---

**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**

The RGL object (using `rgl::highlevel()`).

**Examples**

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

```
ini3D()
pts<-matrix(c(1,1,1,5,5,5), ncol = 3, byrow = TRUE)
plotPoints3D(pts)
finalize3D(argsAxes3d = list(edges = "bbox"))
```

---

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,
  crit = "min",
  dubND = TRUE,
  ...
)
```

**Arguments**

p	Dimension of the points.
n	Number of samples generated (note only a subset of these will be non-dominated).
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.
crit	Criteria used (a vector of min/max).
dubND	Should duplicated non-dominated points be considered as non-dominated.
...	Further arguments passed on to <a href="#">genSample</a> .

**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 )
pts <- genNDSet(3, 50, range = range, random = TRUE, keep = TRUE)
head(pts)
Rfast::colMinsMaxs(as.matrix(pts))
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)))
plotPoints3D(pts[,1:3])
plotPoints3D(pts[pts$nd,1:3], argsPlot3d = list(col = "red", size = 10))
plotCones3D(pts[pts$nd,1:3], argsPolygon3d = list(alpha = 1))
finalize3D()

ini3D()
range <- c(1,100)
cent <- rep(range[1] + (range[2]-range[1])/2, 3)
pts <- genNDSet(3, 100, 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$nd,], 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$nd,], 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: <ul style="list-style-type: none"> <li>• argsSphere: A list of arguments for generating points on a sphere:           <ul style="list-style-type: none"> <li>– radius: The radius of the sphere.</li> <li>– center: The center of the sphere.</li> <li>– plane: The plane used.</li> <li>– below: Either true (generate points below the plane), false (generate points above the plane) or NULL (generated on the whole sphere).</li> <li>– factor: If using a plane. Then the factor to multiply n with, so generate enough points below/above the plane.</li> <li>– closeToPlane: If TRUE only return points close to the plane.</li> </ul> </li> <li>• argsBox: A list of arguments for generating points inside boxes:           <ul style="list-style-type: none"> <li>– 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.</li> </ul> </li> </ul>



- 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  $\text{floor}(p/2) : \text{ceiling}(p/2)$  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))

```

---

getTexture

*Save a point symbol as a temporary file.*


---

## Description

Save a point symbol as a temporary file.

## Usage

```
getTexture(pch = 16, cex = 10, ...)
```

## Arguments

pch	Point number/symbol.
cex	Point size
...	Further arguments passed to plot.

## Value

The file name.

## Examples

```

# Pch shapes
generateRPointShapes<-function(){
  oldPar<-par()
  par(font=2, mar=c(0.5,0,0,0))

```

```

y=rev(c(rep(1,6),rep(2,5), rep(3,5), rep(4,5), rep(5,5)))
x=c(rep(1:5,5),6)
plot(x, y, pch = 0:25, cex=1.5, ylim=c(1,5.5), xlim=c(1,6.5),
      axes=FALSE, xlab="", ylab="", bg="blue")
text(x, y, labels=0:25, pos=3)
par(mar=oldPar$mar, font=oldPar$font )
}
generateRPointShapes()

getTexture()

```

---

gMOIPTheme

*The ggplot theme for the package*


---

## Description

The ggplot theme for the package

## Usage

```
gMOIPTheme(...)
```

## Arguments

... Further arguments parsed to `ggplot2::theme()`.

## Value

The theme object.

## Examples

```

pts <- matrix(c(1,1), ncol = 2, byrow = TRUE)
plotHull2D(pts)
pts1 <- matrix(c(2,2, 3,3), ncol = 2, byrow = TRUE)
pts2 <- matrix(c(1,1, 2,2, 0,1), ncol = 2, byrow = TRUE)
ggplot2::ggplot() +
  plotHull2D(pts2, drawPoints = TRUE, addText = "coord", drawPlot = FALSE) +
  plotHull2D(pts1, drawPoints = TRUE, drawPlot = FALSE) +
  gMOIPTheme() +
  ggplot2::xlab(expression(x[1])) +
  ggplot2::ylab(expression(x[2]))

```

---

hullSegment	<i>Find segments (lines) of a face.</i>
-------------	---

---

**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 $m \times p$ array of vertices of the convex hull, as used by <code>geometry::convhulln()</code> .
hull	Tessellation (or triangulation) generated by <code>geometry::convhulln()</code> 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 <code>tol</code> 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 <code>tol</code> to be <code>tol = 1.e-13*mean(abs(vertices(:)))</code> In higher dimensions, the numerical issues of floating point arithmetic will probably suggest a larger value of <code>tol</code> .

**Value**

A matrix with segments.

**Author(s)**

Lars Relund <lars@relund.dk>

---

inHull	<i>Efficient test for points inside a convex hull in <math>p</math> dimensions.</i>
--------	---

---

**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**

<code>pts</code>	A $n \times p$ 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.
<code>vertices</code>	A $m \times p$ array of vertices of the convex hull. May contain redundant (non-vertex) points.
<code>hull</code>	Tessellation (or triangulation) generated by <code>convhulln</code> (only works if the dimension of the hull is $p$ ). If <code>hull</code> is <code>NULL</code> , then it will be generated.
<code>tol</code>	Tolerance on the tests for inclusion in the convex hull. You can think of <code>tol</code> as the difference a point value may be different from the values of 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 <code>tol</code> . <code>tol</code> is not used if the dimension of the hull is larger than one and not equal $p$ .

**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 and [how to find a point inside a hull](#). 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 = TRUE, clear = FALSE, ...)
```

**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: <ul style="list-style-type: none"> <li>argsPlot3d: A list of arguments for <code>rgl::plot3d</code>.</li> <li>argsAspect3d: A list of arguments for <code>rgl::aspect3d</code>.</li> </ul>

**Value**

NULL (invisible).

**Examples**

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

lim <- c(-1, 7)
ini3D(argsPlot3d = list(xlim = lim, ylim = lim, zlim = lim))
plotPoints3D(pts)
finalize3D()
```

---

integerPoints	<i>Integer points in the feasible region (<math>Ax \leq b</math>).</i>
---------------	--

---

**Description**

Integer points in the feasible region ( $Ax \leq b$ ).

**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 <code>rgl::clear3d()</code> .
close	Call <code>rgl::close3d()</code> .
zoom	Zoom level.
...	Additional parameters passed to <code>rgl::view3d()</code> .

**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, 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 view angle 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.

---

plotCones2D

*Plot a cone defined by a point in 2D.*

---

**Description**

The cones are defined as the point plus/minus rays of R<sup>2</sup>.

**Usage**

```
plotCones2D(
  pts,
  drawPoint = TRUE,
  drawLines = TRUE,
  drawPolygons = TRUE,
  direction = 1,
  rectangle = FALSE,
  drawPlot = TRUE,
  m = apply(pts, 2, min) - 5,
  M = apply(pts, 2, max) + 5,
  ...
)
```

**Arguments**

<code>pts</code>	A matrix with a point in each row.
<code>drawPoint</code>	Draw the points defining the cone.
<code>drawLines</code>	Draw lines of the cone.
<code>drawPolygons</code>	Draw polygons of the cone.
<code>direction</code>	Ray direction. If <i>i</i> 'th entry is positive, consider the <i>i</i> 'th column of <code>pts</code> plus a value greater than on equal zero (minimize objective $i$ ). If negative, consider the <i>i</i> 'th column of <code>pts</code> minus a value greater than on equal zero (maximize objective $i$ ).
<code>rectangle</code>	Draw the cone as a rectangle.
<code>drawPlot</code>	Draw the ggplot. Set to FALSE if you want to combine hulls in a single plot.
<code>m</code>	Minimum values of the bounding box.
<code>M</code>	Maximum values of the bounding box.
<code>...</code>	Further arguments passed to <a href="#">plotHull2D</a>

**Value**

A ggplot object

**Examples**

```
library(ggplot2)
plotCones2D(c(4,4), drawLines = FALSE, drawPoint = TRUE,
            argsGeom_point = list(col = "red", size = 10),
            argsGeom_polygon = list(alpha = 0.5), rectangle = TRUE)
plotCones2D(c(1,1), rectangle = FALSE)
plotCones2D(matrix(c(3,3,2,2), ncol = 2, byrow = TRUE))

## The Danish flag
lst <- list(argsGeom_polygon = list(alpha = 0.85, fill = "red"),
           drawPlot = FALSE, drawPoint = FALSE, drawLines = FALSE)
p1 <- do.call(plotCones2D, args = c(list(c(2,4), direction = 1), lst))
```

```

p2 <- do.call(plotCones2D, args = c(list(c(1,2), direction = -1), lst))
p3 <- do.call(plotCones2D, args = c(list(c(2,2), direction = c(1,-1)), lst))
p4 <- do.call(plotCones2D, args = c(list(c(1,4), direction = c(-1,1)), lst))
ggplot() + p1 + p2 + p3 + p4 + theme_void()

```

---

plotCones3D

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


---

## Description

The cones are defined as the point plus  $R^3_+$ .

## 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 or equal zero (minimize objective $i$ ). If negative, consider the $i$ 'th column of pts minus a value greater than or 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: <ul style="list-style-type: none"> <li>argsPlot3d: A list of arguments for <code>rgl::plot3d</code>.</li> <li>argsSegments3d: A list of arguments for <code>rgl::segments3d</code>.</li> <li>argsPolygon3d: A list of arguments for <code>rgl::polygon3d</code>.</li> </ul>

## Value

Object ids (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))
ids <- plotCones3D(c(2,2,4), direction = c(-1,-1,1))
finalize3D()
# pop3d(id = ids) # remove last cone
```

---

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 equal coord add coordinates to the points. Otherwise number all points from one.

**Value**

The ggplot 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't 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 2D.*


---

### Description

Plot the convex hull of a set of points in 2D.

**Usage**

```
plotHull2D(
  pts,
  drawPoints = FALSE,
  drawLines = TRUE,
  drawPolygons = TRUE,
  addText = FALSE,
  addRays = FALSE,
  direction = 1,
  drawPlot = TRUE,
  drawBBoxHull = FALSE,
  m = apply(pts, 2, min) - 5,
  M = apply(pts, 2, max) + 5,
  ...
)
```

**Arguments**

<code>pts</code>	A matrix with a point in each row.
<code>drawPoints</code>	Draw the points.
<code>drawLines</code>	Draw lines of the facets.
<code>drawPolygons</code>	Fill the hull.
<code>addText</code>	Add text to the points. Currently <code>coord</code> (coordinates), <code>rownames</code> (rownames) and both supported or a vector with text.
<code>addRays</code>	Add the ray defined by <code>direction</code> .
<code>direction</code>	Ray direction. If <i>i</i> 'th entry is positive, consider the <i>i</i> 'th column of <code>pts</code> plus a value greater than on equal zero (minimize objective $i$ ). If negative, consider the <i>i</i> 'th column of <code>pts</code> minus a value greater than on equal zero (maximize objective $i$ ).
<code>drawPlot</code>	Draw the <code>ggplot</code> . Set to <code>FALSE</code> if you want to combine hulls in a single plot.
<code>drawBBoxHull</code>	If <code>addRays</code> then draw the hull areas hitting the bounding box also.
<code>m</code>	Minimum values of the bounding box.
<code>M</code>	Maximum values of the bounding box.
<code>...</code>	Further arguments passed on the the <code>ggplot</code> plotting functions. This must be done as lists. Currently the following arguments are supported: <ul style="list-style-type: none"> <li><code>argsGeom_point</code>: A list of arguments for <code>ggplot2::geom_point</code>.</li> <li><code>argsGeom_path</code>: A list of arguments for <code>ggplot2::geom_path</code>.</li> <li><code>argsGeom_polygon</code>: A list of arguments for <code>ggplot2::geom_polygon</code>.</li> <li><code>argsGeom_label</code>: A list of arguments for <code>ggplot2::geom_label</code>.</li> </ul>

**Value**

The `ggplot` object if `drawPlot = TRUE`; otherwise, a list of `ggplot` components.

**Examples**

```

library(ggplot2)
pts<-matrix(c(1,1), ncol = 2, byrow = TRUE)
plotHull2D(pts)
pts1<-matrix(c(2,2, 3,3), ncol = 2, byrow = TRUE)
plotHull2D(pts1, drawPoints = TRUE)
plotHull2D(pts1, drawPoints = TRUE, addRays = TRUE, addText = "coord")
plotHull2D(pts1, drawPoints = TRUE, addRays = TRUE, addText = "coord", drawBBoxHull = TRUE)
plotHull2D(pts1, drawPoints = TRUE, addRays = TRUE, direction = -1, addText = "coord")
pts2<-matrix(c(1,1, 2,2, 0,1), ncol = 2, byrow = TRUE)
plotHull2D(pts2, drawPoints = TRUE, addText = "coord")
plotHull2D(pts2, drawPoints = TRUE, addRays = TRUE, addText = "coord")
plotHull2D(pts2, drawPoints = TRUE, addRays = TRUE, direction = -1, addText = "coord")
## Combine hulls
ggplot() +
  plotHull2D(pts2, drawPoints = TRUE, addText = "coord", drawPlot = FALSE) +
  plotHull2D(pts1, drawPoints = TRUE, drawPlot = FALSE) +
  gMOIPTheme() +
  xlab(expression(x[1])) +
  ylab(expression(x[2]))

# Plotting an LP
A <- matrix(c(-3,2,2,4,9,10), ncol = 2, byrow = TRUE)
b <- c(3,27,90)
obj <- c(7.75, 10)
pts3 <- cornerPoints(A, b)
plotHull2D(pts3, drawPoints = TRUE, addText = "coord", argsGeom_polygon = list(fill = "red"))

```

---

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: <ul style="list-style-type: none"> <li>• argsPlot3d: A list of arguments for <code>rgl::plot3d</code>.</li> <li>• argsSegments3d: A list of arguments for <code>rgl::segments3d</code>.</li> <li>• argsPolygon3d: A list of arguments for <code>rgl::polygon3d</code>.</li> <li>• argsShade3d: A list of arguments for <code>rgl::shade3d</code>.</li> <li>• argsText3d: A list of arguments for <code>rgl::text3d</code>.</li> </ul>

**Value**

A list with hull, pts classified and object ids (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)
lst <- plotHull3D(pts, argsPolygon3d = list(alpha=0.9), argsSegments3d = list(color="red"))
finalize3D()
# pop3d(id = lst$ids) # remove last hull

## 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, dubND = FALSE)
pts <- as.data.frame(pts[,1:3])

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, 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[,1])+10),
  ylim = c(0,max(pts[,2])+10),
  zlim = c(0,max(pts[,3])+10)))
plotHull3D(pts, drawPoints = TRUE, drawLines = FALSE,
  argsPolygon3d = list(alpha = 1), 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, argsPolygon3d = list(color = "red"), addRays = TRUE)
plotCones3D(pts, argsPolygon3d = list(alpha = 1), rectangle = TRUE)
finalize3D()

```

---

plotMTeX3D

*Plot TeX in the margin*


---

### Description

Plot TeX in the margin

### Usage

```
plotMTeX3D(tex, edge, line = 0, at = NULL, pos = NA, ...)
```

### Arguments

tex	TeX string
edge	The position at which to draw the axis or text.
line	The “line” of the plot margin to draw the label on.
at	The value of a coordinate at which to draw the axis.
pos	The position at which to draw the axis or text.
...	Further arguments passed to <code>plotTeX3D()</code> .

### Value

The object IDs of objects added to the scene.

---

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). A vector is currently not supported.
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 equal coord add coordinates to the points. Otherwise number all points from one.

**Value**

The ggplot 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 <- addNDSet(dat, crit = "min", keepDom = TRUE)  
plotNDSet2D(points, crit = "min", addTriangles = TRUE)  
plotNDSet2D(points, crit = "min", addTriangles = FALSE)  
plotNDSet2D(points, crit = "min", addTriangles = TRUE, addHull = FALSE)  
points <- addNDSet(dat, crit = "max", keepDom = TRUE)  
plotNDSet2D(points, crit = "max", addTriangles = TRUE)  
plotNDSet2D(points, crit = "max", addHull = FALSE)
```

---

plotPlane3D	<i>Plot a plane in 3D.</i>
-------------	----------------------------

---

### Description

Plot a plane in 3D.

### Usage

```
plotPlane3D(
  normal,
  point = NULL,
  offset = 0,
  useShade = TRUE,
  useLines = FALSE,
  usePoints = FALSE,
  ...
)
```

### Arguments

normal	Normal to the plane.
point	A point on the plane.
offset	The offset of the plane (only used if point = NULL).
useShade	Plot shade of the plane.
useLines	Plot lines inside the plane.
usePoints	Plot point shapes inside the plane.
...	Further arguments passed on the the RGL plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: <ul style="list-style-type: none"> <li>argsPlanes3d: A list of arguments for <code>rgl::planes3d()</code> used when useShade = TRUE.</li> <li>argsLines: A list of arguments for <code>rgl::persp3d()</code> when useLines = TRUE. Moreover, the list may contain lines: number of lines.</li> </ul>

### 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(color="red"))
```

```

plotPoints3D(c(2,2,2))
plotPlane3D(c(2,1,1), offset = -6, argsPlanes3d = list(color="blue"))
plotPlane3D(c(2,1,1), argsPlanes3d = list(color="green"))
finalize3D()

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), useLines = TRUE, useShade = TRUE)
ids <- plotPlane3D(c(1,2,1), point = c(2,2,2), argsLines = list(col="blue", lines = 100),
                  useLines = TRUE)
finalize3D()
# pop3d(id = ids) # remove last plane

```

---

plotPoints3D	<i>Plot points in 3D.</i>
--------------	---------------------------

---

### Description

Plot points in 3D.

### Usage

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

### 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: <ul style="list-style-type: none"> <li>argsPlot3d: A list of arguments for <code>rgl::plot3d</code>.</li> <li>argsPch3d: A list of arguments for <code>rgl::pch3d</code>.</li> <li>argsText3d: A list of arguments for <code>rgl::text3d</code>.</li> </ul>

### Value

Object ids (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")
ids <- plotPoints3D(c(3,3,3, 4,4,4), addText = "rownames")
finalize3D()
rgl::rglwidget()
# pop3d(ids) # remove the last again

```

---

plotPolygon3D	<i>Plot a polygon.</i>
---------------	------------------------

---

### Description

Plot a polygon.

### Usage

```

plotPolygon3D(
  pts,
  useShade = TRUE,
  useLines = FALSE,
  usePoints = FALSE,
  useFrame = TRUE,
  ...
)

```

### Arguments

pts	Vertices.
useShade	Plot shade of the polygon.
useLines	Plot lines inside the polygon.
usePoints	Plot point shapes inside the polygon.
useFrame	Plot a frame around the polygon.
...	Further arguments passed on the RGL plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: <ul style="list-style-type: none"> <li>argsShade: A list of arguments for <code>rgl::polygon3d</code> (<math>n &gt; 4</math> vertices), <code>rgl::triangles3d()</code> (<math>n = 3</math> vertices) and <code>rgl::quads3d()</code> (<math>n = 4</math> vertices) if <code>useShade = TRUE</code>.</li> <li>argsFrame: A list of arguments for <code>rgl::lines3d</code> if <code>useFrame = TRUE</code>.</li> <li>argsPoints: A list of arguments for <code>rgl::shade3d</code> if <code>usePoints = TRUE</code>. It is important to give a texture using <code>texture</code>. A texture can be set using <code>getTexture()</code>.</li> <li>argsLines: A list of arguments for <code>rgl::persp3d()</code> when <code>useLines = TRUE</code>. Moreover, the list may contain <code>lines: number of lines</code>.</li> </ul>

**Value**

Object ids (invisible).

**Examples**

```
pts <- data.frame(x = c(1,0,0,0.4), y = c(0,1,0,0.3), z = c(0,0,1,0.3))
pts <- data.frame(x = c(1,0,0), y = c(0,1,0), z = c(0,0,1))

ini3D()
plotPolygon3D(pts)
finalize3D()

ini3D()
plotPolygon3D(pts, argsShade = list(color = "red", alpha = 1))
finalize3D()

ini3D()
plotPolygon3D(pts, useFrame = TRUE, argsShade = list(color = "red", alpha = 0.5),
              argsFrame = list(color = "green"))
finalize3D()

ini3D()
plotPolygon3D(pts, useFrame = TRUE, useLines = TRUE, useShade = TRUE,
              argsShade = list(color = "red", alpha = 0.2),
              argsLines = list(color = "blue"))
finalize3D()

ini3D()
ids <- plotPolygon3D(pts, usePoints = TRUE, useFrame = TRUE,
                    argsPoints = list(texture = getTexture(pch = 16, cex = 20)))
finalize3D()
# pop3d(id = ids) # remove object again

# In general you have to finetune size and numbers when you use textures
# Different pch
for (i in 0:3) {
  fname <- getTexture(pch = 15+i, cex = 30)
  ini3D(TRUE)
  plotPolygon3D(pts, usePoints = TRUE, argsPoints = list(texture = fname))
  finalize3D()
}

# Size of pch
for (i in 1:4) {
  fname <- getTexture(pch = 15+i, cex = 10 * i)
  ini3D(TRUE)
  plotPolygon3D(pts, usePoints = TRUE, argsPoints = list(texture = fname))
  finalize3D()
}

# Number of pch
```

```
fname <- getTexture(pch = 16, cex = 20)
for (i in 1:4) {
  ini3D(TRUE)
  plotPolygon3D(pts, usePoints = TRUE,
    argsPoints = list(texture = fname, texcoords = rbind(pts$x, pts$y, pts$z)*5*i))
  finalize3D()
}
```

---

plotPolytope	<i>Plot the polytope (bounded convex set) of a linear mathematical program</i>
--------------	--

---

### Description

This is a wrapper function calling [plotPolytope2D\(\)](#) (2D graphics) and [plotPolytope3D\(\)](#) (3D graphics).

### 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 equal coord add coordinates to the points. Otherwise number all points from one.
...	<p>If 2D, further arguments passed on the the ggplot plotting functions. This must be done as lists. Currently the following arguments are supported:</p> <ul style="list-style-type: none"> <li>• argsFaces: A list of arguments for <code>plotHull2D</code>.</li> <li>• argsFeasible: A list of arguments for ggplot2 functions: <ul style="list-style-type: none"> <li>– geom_point: A list of arguments for <code>ggplot2::geom_point</code>.</li> <li>– geom_line: A list of arguments for <code>ggplot2::geom_line</code>.</li> </ul> </li> <li>• argsLabels: A list of arguments for ggplot2 functions: <ul style="list-style-type: none"> <li>– geom_text: A list of arguments for <code>ggplot2::geom_text</code>.</li> </ul> </li> <li>• argsOptimum: <ul style="list-style-type: none"> <li>– geom_point: A list of arguments for <code>ggplot2::geom_point</code>.</li> <li>– geom_abline: A list of arguments for <code>ggplot2::geom_abline</code>.</li> <li>– geom_label: A list of arguments for <code>ggplot2::geom_label</code>.</li> </ul> </li> <li>• argsTheme: A list of arguments for <code>ggplot2::theme</code>.</li> </ul> <p>If 3D further arguments passed on the the RGL plotting functions. This must be done as lists. Currently the following arguments are supported:</p> <ul style="list-style-type: none"> <li>• argsAxes3d: A list of arguments for <code>rgl::axes3d</code>.</li> <li>• argsPlot3d: A list of arguments for <code>rgl::plot3d</code> to open the RGL window.</li> <li>• argsTitle3d: A list of arguments for <code>rgl::title3d</code>.</li> <li>• argsFaces: A list of arguments for <code>plotHull3D</code>.</li> <li>• argsFeasible: A list of arguments for RGL functions: <ul style="list-style-type: none"> <li>– points3d: A list of arguments for <code>rgl::points3d</code>.</li> <li>– segments3d: A list of arguments for <code>rgl::segments3d</code>.</li> <li>– triangles3d: A list of arguments for <code>rgl::triangles3d</code>.</li> </ul> </li> <li>• argsLabels: A list of arguments for RGL functions: <ul style="list-style-type: none"> <li>– points3d: A list of arguments for <code>rgl::points3d</code>.</li> <li>– text3d: A list of arguments for <code>rgl::text3d</code>.</li> </ul> </li> <li>• argsOptimum: A list of arguments for RGL functions: <ul style="list-style-type: none"> <li>– points3d: A list of arguments for <code>rgl::points3d</code>.</li> </ul> </li> </ul>

**Value**

If 2D a ggplot object. If 3D a RGL window with the 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(-3,2,2,2,4,9,10), 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,
  argsFaces = list(argsGeom_polygon = list(fill = "red"))
)
# 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",
  argsOptimum = list(lty="solid")
)
# 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",
  argsLabels = list(size = 4, color = "blue"),
  argsFeasible = list(color = "red", size = 3)
)
#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",
  argsFeasible = list(color = "red")
)
# 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")
plotPolytope(A, b, plotOptimum = TRUE, obj = obj, labels = "coord",
             argsFaces = list(drawLines = FALSE, argsPolygon3d = list(alpha = 0.95)),
             argsLabels = list(points3d = list(color = "blue")))
# 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	<i>Plot the polytope (bounded convex set) of a linear mathematical program</i>
----------------	--

---

### 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 equal coord add coordinates to the points. Otherwise number all points from one.
...	Further arguments passed on the the ggplot plotting functions. This must be done as lists. Currently the following arguments are supported: <ul style="list-style-type: none"> <li>• argsFaces: A list of arguments for <a href="#">plotHull2D</a>.</li> <li>• argsFeasible: A list of arguments for ggplot12 functions: <ul style="list-style-type: none"> <li>– geom_point: A list of arguments for <a href="#">ggplot2::geom_point</a>.</li> <li>– geom_line: A list of arguments for <a href="#">ggplot2::geom_line</a>.</li> </ul> </li> <li>• argsLabels: A list of arguments for ggplot12 functions: <ul style="list-style-type: none"> <li>– geom_text: A list of arguments for <a href="#">ggplot2::geom_text</a>.</li> </ul> </li> <li>• argsOptimum: <ul style="list-style-type: none"> <li>– geom_point: A list of arguments for <a href="#">ggplot2::geom_point</a>.</li> <li>– geom_abline: A list of arguments for <a href="#">ggplot2::geom_abline</a>.</li> <li>– geom_label: A list of arguments for <a href="#">ggplot2::geom_label</a>.</li> </ul> </li> <li>• argsTheme: A list of arguments for <a href="#">ggplot2::theme</a>.</li> </ul>

**Value**

A ggplot object.

**Note**

In general use [plotPolytope\(\)](#) instead of this function. The feasible region defined by the constraints must be bounded otherwise you may see strange results.

**Author(s)**

Lars Relund <lars@relund.dk>

**See Also**

[plotPolytope\(\)](#) for examples.

---

plotPolytope3D	<i>Plot the polytope (bounded convex set) of a linear mathematical program</i>
----------------	--

---

**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 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 equal coord add coordinates to the points. Otherwise number all points from one.
...	Further arguments passed on the the RGL plotting functions. This must be done as lists. Currently the following arguments are supported: <ul style="list-style-type: none"> <li>• <code>argsAxes3d</code>: A list of arguments for <code>rgl::axes3d</code>.</li> <li>• <code>argsPlot3d</code>: A list of arguments for <code>rgl::plot3d</code> to open the RGL window.</li> <li>• <code>argsTitle3d</code>: A list of arguments for <code>rgl::title3d</code>.</li> <li>• <code>argsFaces</code>: A list of arguments for <code>plotHull3D</code>.</li> </ul>



- argsFeasible: A list of arguments for RGL functions:
  - points3d: A list of arguments for `rgl::points3d`.
  - segments3d: A list of arguments for `rgl::segments3d`.
  - triangles3d: A list of arguments for `rgl::triangles3d`.
- argsLabels: A list of arguments for RGL functions:
  - points3d: A list of arguments for `rgl::points3d`.
  - text3d: A list of arguments for `rgl::text3d`.
- argsOptimum: A list of arguments for RGL functions:
  - points3d: A list of arguments for `rgl::points3d`.

**Value**

A RGL window with 3D plot.

**Note**

In general use `plotPolytope()` instead of this function. The feasible region defined by the constraints must be bounded otherwise you may see strange results.

**Author(s)**

Lars Relund <lars@relund.dk>

**See Also**

`plotPolytope()` for examples.

---

plotRectangle3D

*Plot a rectangle defined by two corner points.*

---

**Description**

The rectangle is defined by  $x|_a \leq x \leq b$  where  $a$  is the minimum values and  $b$  is the maximum values.

**Usage**

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

**Arguments**

- |                  |   |
|------------------|---|
| <code>a</code>   | A vector of length 3.   |
| <code>b</code>   | A vector of length 3.   |
| <code>...</code> | Further arguments passed on the the RGL plotting functions. This must be done as lists (see examples). Currently the following arguments are supported: <ul style="list-style-type: none"> <li>• argsPlot3d: A list of arguments for <code>rgl::plot3d</code>.</li> </ul> |

- `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

Object ids (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))
ids <- plotRectangle3D(c(2,2,2), c(3,3,2.5), argsPolygon3d = list(alpha = 1) )
finalize3D()
# pop3d(id = ids) remove last object
```

---

plotTeX3D

*Plot TeX at a position.*

---

### Description

Plot TeX at a position.

### Usage

```
plotTeX3D(
  x,
  y,
  z,
  tex,
  cex = graphics::par("cex"),
  fixedSize = FALSE,
  size = 480,
  ...
)
```

### Arguments

<code>x</code>	Coordinate.
<code>y</code>	Coordinate.
<code>z</code>	Coordinate.
<code>tex</code>	TeX string.
<code>cex</code>	Expansion factor (you properly have to fine tune it).

fixedSize	Fix the size of the object (no scaling when zoom).
size	Size of the generated png.
...	Arguments passed on to <code>rgl::sprites3d()</code> and <code>texToPng()</code> .

**Value**

The shape ID of the displayed object is returned.

**Examples**

```
## Not run:
tex0 <- "$\mathbb{R}_{\geq 0}$"
tex1 <- "\\LaTeX"
tex2 <- "This is a title"
ini3D(argsPlot3d = list(xlim = c(0, 2), ylim = c(0, 2), zlim = c(0, 2)))
plotTeX3D(0.75,0.75,0.75, tex0)
plotTeX3D(0.5,0.5,0.5, tex0, cex = 2)
plotTeX3D(1,1,1, tex2)
finalize3D()
ini3D(new = TRUE, argsPlot3d = list(xlim = c(0, 200), ylim = c(0, 200), zlim = c(0, 200)))
plotTeX3D(75,75,75, tex0)
plotTeX3D(50,50,50, tex1)
plotTeX3D(100,100,100, tex2)
finalize3D()

## End(Not run)
```

---

plotTitleTeX3D                      *Draw boxes, axes and other text outside the data using TeX strings.*

---

**Description**

Draw boxes, axes and other text outside the data using TeX strings.

**Usage**

```
plotTitleTeX3D(
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  zlab = NULL,
  line = NA,
  ...
)
```

**Arguments**

main	The main title for the plot.
sub	The subtitle for the plot.
xlab	The axis labels for the plot .
ylab	The axis labels for the plot .
zlab	The axis labels for the plot .
line	The “line” of the plot margin to draw the label on.
...	Additional parameters which are passed to <code>plotMTex3D</code> .

**Details**

The rectangular prism holding the 3D plot has 12 edges. They are identified using 3 character strings. The first character ('x', 'y', or 'z') selects the direction of the axis. The next two characters are each '-' or '+', selecting the lower or upper end of one of the other coordinates. If only one or two characters are used, the third is assumed to be '-'. For example edge = 'x+' draws an x-axis at the high level of y and the low level of z.

By default, `axes3d` uses the `bbox3d` function to draw the axes. The labels will move so that they do not obscure the data. Alternatively, a vector of arguments as described above may be used, in which case fixed axes are drawn using `axis3d`.

If `pos` is a numeric vector of length 3, `edge` determines the direction of the axis and the tick marks, and the values of the other two coordinates in `pos` determine the position. See the examples.

**Value**

The object IDs of objects added to the scene.

**Examples**

```
## Not run:
ini3D(argsPlot3d = list(xlim = c(0, 2), ylim = c(0, 2), zlim = c(0, 2)))
plotTitleTex3D(main = "\\LaTeX", sub = "subtitle $\\alpha$",
               xlab = "$x^1_2$", ylab = "$\\beta$", zlab = "$x\\cdot y$")
finalize3D()

## End(Not run)
```

---

pngSize *To size of the png file.*

---

**Description**

To size of the png file.

**Usage**

```
pngSize(png)
```

**Arguments**

png                    Png file name.

**Value**

A list with width and height.

---

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**

NULL (invisible).

**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 view angle to insert into R code
```

---

slices	<i>Find all corner points in the slices define for each fixed integer combination.</i>
--------	--

---

### 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)
```

---

texToPng	<i>Convert LaTeX to a png file</i>
----------	------------------------------------

---

**Description**

Convert LaTeX to a png file

**Usage**

```
texToPng(
  tex,
  width = NULL,
  height = NULL,
  dpi = 72,
  viewPng = FALSE,
  fontsize = 12,
  calcM = FALSE,
  crop = FALSE
)
```

**Arguments**

tex	TeX string. Remember to escape backslash with \.
width	Width of the png.
height	Height of the png (width are ignored).
dpi	Dpi of the png. Not used if width or height are specified.
viewPng	View the result in the plots window.
fontsize	Front size used in the LaTeX document.
calcM	Estimate 1 em in pixels in the resulting png.
crop	Call command line program pdfcrop (must be installed).

**Value**

The filename of the png or a list if calcM = TRUE.

**Examples**

```
## Not run:
tex <- "$\mathbb{R}_{\geq}$"
texToPng(tex, viewPng = TRUE)
texToPng(tex, fontsize = 20, viewPng = TRUE)
texToPng(tex, height = 50, fontsize = 10, viewPng = TRUE)
texToPng(tex, height = 50, fontsize = 50, viewPng = TRUE)
tex <- "MMM"
texToPng(tex, dpi=72, calcM = TRUE)
texToPng(tex, width = 100, calcM = TRUE)
```

```
f <- texToPng(tex, dpi=300)
pngSize(f)

## End(Not run)
```



# Index

.checkPts(), 5

addNDSet, 3  
addRays, 4

bbox3d, 60  
binaryPoints, 5

classifyNDSet, 6  
convexHull, 7  
cornerPoints, 9  
cornerPointsCont, 10  
criterionPoints, 11

df2String, 12  
dimFace, 12

finalize3D, 13

genNDSet, 14  
genSample, 15, 16  
geometry::convhulln(), 22  
getTexture, 20  
getTexture(), 44  
ggplot2::geom\_abline, 47, 55  
ggplot2::geom\_label, 36, 47, 55  
ggplot2::geom\_line, 47, 55  
ggplot2::geom\_path, 36  
ggplot2::geom\_point, 36, 47, 55  
ggplot2::geom\_polygon, 36  
ggplot2::geom\_text, 47, 55  
ggplot2::theme, 47, 55  
ggplot2::theme(), 21  
gMOIPTheme, 21

hullSegment, 22

inHull, 22  
ini3D, 24  
integerPoints, 25  
loadView, 26

mergeLists, 27

plotCones2D, 27  
plotCones3D, 29  
plotCriterion2D, 30  
plotHull2D, 28, 35, 47, 55  
plotHull3D, 37, 47, 56  
plotMTex3D, 40, 60  
plotNDSet2D, 40  
plotPlane3D, 42  
plotPoints3D, 43  
plotPolygon3D, 44  
plotPolytope, 46  
plotPolytope(), 55, 57  
plotPolytope2D, 54  
plotPolytope2D(), 46  
plotPolytope3D, 55  
plotPolytope3D(), 46  
plotRectangle3D, 57  
plotTex3D, 58  
plotTex3D(), 40  
plotTitleTex3D, 59  
pngSize, 60

rgl::aspect3d, 25  
rgl::axes3d, 13, 47, 56  
rgl::clear3d(), 26  
rgl::close3d(), 26  
rgl::highlevel(), 14  
rgl::lines3d, 44  
rgl::pch3d, 43  
rgl::persp3d(), 42, 44  
rgl::planes3d(), 42  
rgl::plot3d, 25, 29, 38, 43, 47, 56, 57  
rgl::points3d, 47, 57  
rgl::polygon3d, 29, 38, 44, 58  
rgl::quads3d(), 44  
rgl::segments3d, 29, 38, 47, 57, 58  
rgl::shade3d, 38, 44, 58  
rgl::sprites3d(), 59

`rgl::text3d`, [38](#), [43](#), [47](#), [57](#)  
`rgl::title3d`, [13](#), [47](#), [56](#)  
`rgl::triangles3d`, [47](#), [57](#)  
`rgl::triangles3d()`, [44](#)  
`rgl::view3d()`, [26](#)

`saveView`, [61](#)  
`slices`, [62](#)

`texToPng`, [63](#)  
`texToPng()`, [59](#)