Package 'PlaneGeometry'

August 6, 2020

Type Package

Title Plane Geometry

Version 1.2.0

Date 2020-08-06

Author Stéphane Laurent

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description An extensive set of plane geometry routines. Provides R6 classes representing triangles, circles, circular arcs, ellipses, elliptical arcs and lines, and their plot methods. Also provides R6 classes representing transformations: rotations, reflections, homotheties, scalings, general affine transformations, inversions, Möbius transformations.

License GPL-3

Encoding UTF-8

LazyData true

Imports R6, methods, graphics, stringr, gsl, uniformly, sdpt3r

RoxygenNote 7.0.2

Suggests testthat, knitr, rmarkdown, viridisLite, sets, ellipse

VignetteBuilder knitr

URL https://github.com/stla/PlaneGeometry

BugReports https://github.com/stla/PlaneGeometry/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2020-08-06 18:20:02 UTC

R topics documented:

`Affine` *R6 class representing an affine map.*

Description

An affine map is given by a 2x2 matrix (a linear transformation) and a vector (the "intercept").

Active bindings

`A` get or set the matrix `A`

`b` get or set the vector `b`

Methods

Public methods:

- `Affine$new()`
- `Affine$print()`
- `Affine$get3x3matrix()`
- `Affine$inverse()`
- `Affine$compose()`
- `Affine$transform()`
- `Affine$transformLine()`
- `Affine$transformEllipse()`
- `Affine$clone()`

Method `new()`**:** Create a new `Affine` object.

Usage:

```
Affine$new(A, b)
```

Arguments:

`A` the 2x2 matrix of the affine map

`b` the shift vector of the affine map

Returns: A new `Affine` object.

Method `print()`**:** Show instance of an `Affine` object.

Usage:

```
Affine$print(...)
```

Arguments:

`...` ignored

Examples:

```
Affine$new(rbind(c(3.5,2),c(0,4)), c(-1, 1.25))
```

Method `get3x3matrix()`**:** The 3x3 matrix representing the affine map.

Usage:

`Affine$get3x3matrix()`

Method `inverse()`: The inverse affine transformation, if it exists.

Usage:

`Affine$inverse()`

Method `compose()`: Compose the reference affine map with another affine map.

Usage:

`Affine$compose(transfo, left = TRUE)`

Arguments:

`transfo` an `Affine` object

`left` logical, whether to compose at left or at right (i.e. returns `f1 o f0` or `f0 o f1`)

Returns: An `Affine` object.

Method `transform()`: Transform a point or several points by the reference affine map.

Usage:

`Affine$transform(M)`

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `transformLine()`: Transform a line by the reference affine transformation (only for invertible affine maps).

Usage:

`Affine$transformLine(line)`

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `transformEllipse()`: Transform an ellipse by the reference affine transformation (only for an invertible affine map). The result is an ellipse.

Usage:

`Affine$transformEllipse(ell)`

Arguments:

`ell` an `Ellipse` object or a `Circle` object

Returns: An `Ellipse` object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Affine$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Affine$print`
## ------------------------------------------------

Affine$new(rbind(c(3.5,2),c(0,4)), c(-1, 1.25))
```

AffineMappingEllipse2Ellipse
Affine transformation mapping a given ellipse to a given ellipse

---

Description

Return the affine transformation which transforms `ell1` to `ell2`.

Usage

```
AffineMappingEllipse2Ellipse(ell1, ell2)
```

Arguments

`ell1`, `ell2`	`Ellipse` or `Circle` objects

Value

An `Affine` object.

Examples

```
ell1 <- Ellipse$new(c(1,1), 5, 1, 30)
( ell2 <- Ellipse$new(c(4,-1), 3, 2, 50) )
f <- AffineMappingEllipse2Ellipse(ell1, ell2)
f$transformEllipse(ell1) # should be ell2
```

AffineMappingThreePoints
Affine transformation mapping three given points to three given points

---

Description

Return the affine transformation which sends `P1` to `Q1`, `P2` to `Q2` and `P3` to `Q3`.

Usage

```
AffineMappingThreePoints(P1, P2, P3, Q1, Q2, Q3)
```

Arguments

`P1, P2, P3`	three non-collinear points
`Q1, Q2, Q3`	three non-collinear points

Value

An `Affine` object.

`Arc` *R6 class representing a circular arc*

Description

An arc is given by a center, a radius, a starting angle and an ending angle. They are respectively named `center`, `radius`, `alpha1` and `alpha2`.

Active bindings

`center` get or set the center

`radius` get or set the radius

`alpha1` get or set the starting angle

`alpha2` get or set the ending angle

`degrees` get or set the `degrees` field

Methods

Public methods:

- `Arc$new()`
- `Arc$print()`
- `Arc$startingPoint()`
- `Arc$endingPoint()`
- `Arc$isEqual()`
- `Arc$complementaryArc()`
- `Arc$path()`
- `Arc$clone()`

Method `new()`**:** Create a new `Arc` object.

Usage:

```
Arc$new(center, radius, alpha1, alpha2, degrees = TRUE)
```

Arguments:

`center` the center

`radius` the radius

`alpha1` the starting angle

`alpha2` the ending angle

`degrees` logical, whether `alpha1` and `alpha2` are given in degrees

Returns: A new `Arc` object.

Examples:

```
arc <- Arc$new(c(1,1), 1, 45, 90)
arc
arc$center
arc$center <- c(0,0)
arc
```

Method `print()`**:** Show instance of an `Arc` object.

Usage:

```
Arc$print(...)
```

Arguments:

`...` ignored

Examples:

```
Arc$new(c(0,0), 2, pi/4, pi/2, FALSE)
```

Method `startingPoint()`**:** Starting point of the reference arc.

Usage:

```
Arc$startingPoint()
```

Method `endingPoint()`**:** Ending point of the reference arc.

Usage:

```
Arc$endingPoint()
```

Method `isEqual()`**:** Check whether the reference arc equals another arc.

Usage:

```
Arc$isEqual(arc)
```

Arguments:

`arc` an `Arc` object

Method `complementaryArc()`**:** Complementary arc of the reference arc.

Usage:

```
Arc$complementaryArc()
```

Examples:

```
arc <- Arc$new(c(0,0), 1, 30, 60)
plot(NULL, type = "n", asp = 1, xlim = c(-1,1), ylim = c(-1,1),
     xlab = NA, ylab = NA)
draw(arc, lwd = 3, col = "red")
draw(arc$complementaryArc(), lwd = 3, col = "green")
```

Method `path()`**:** The reference arc as a path.

Usage:

```
Arc$path(npoints = 100L)
```

Arguments:

`npoints` number of points of the path

Returns: A matrix with two columns `x` and `y` of length `npoints`. See "Filling the lapping area of two circles" in the vignette for an example.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Arc$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Arc$new`
## ------------------------------------------------

arc <- Arc$new(c(1,1), 1, 45, 90)
arc
arc$center
arc$center <- c(0,0)
arc

## ------------------------------------------------
## Method `Arc$print`
## ------------------------------------------------

Arc$new(c(0,0), 2, pi/4, pi/2, FALSE)

## ------------------------------------------------
## Method `Arc$complementaryArc`
## ------------------------------------------------

arc <- Arc$new(c(0,0), 1, 30, 60)
plot(NULL, type = "n", asp = 1, xlim = c(-1,1), ylim = c(-1,1),
     xlab = NA, ylab = NA)
draw(arc, lwd = 3, col = "red")
draw(arc$complementaryArc(), lwd = 3, col = "green")
```

`Circle` *R6 class representing a circle*

Description

A circle is given by a center and a radius, named `center` and `radius`.

Active bindings

`center` get or set the center

`radius` get or set the radius

Methods

Public methods:

- `Circle$new()`
- `Circle$print()`
- `Circle$pointFromAngle()`
- `Circle$diameter()`
- `Circle$tangent()`
- `Circle$tangentsThroughExternalPoint()`
- `Circle$isEqual()`
- `Circle$isOrthogonal()`
- `Circle$angle()`
- `Circle$includes()`
- `Circle$orthogonalThroughTwoPointsOnCircle()`
- `Circle$orthogonalThroughTwoPointsWithinCircle()`
- `Circle$power()`
- `Circle$radicalCenter()`
- `Circle$radicalAxis()`
- `Circle$rotate()`
- `Circle$translate()`
- `Circle$invert()`
- `Circle$asEllipse()`
- `Circle$randomPoints()`
- `Circle$clone()`

Method `new()`**:** Create a new `Circle` object.

Usage:

```
Circle$new(center, radius)
```

Arguments:

`center` the center

`radius` the radius

Returns: A new `Circle` object.

Examples:

```
circ <- Circle$new(c(1,1), 1)
circ
circ$center
circ$center <- c(0,0)
circ
```

Method `print()`: Show instance of a circle object.

Usage:

```
Circle$print(...)
```

Arguments:

`...` ignored

Examples:

```
Circle$new(c(0,0), 2)
```

Method `pointFromAngle()`: Get a point on the reference circle from its polar angle.

Usage:

```
Circle$pointFromAngle(alpha, degrees = TRUE)
```

Arguments:

`alpha` a number, the angle

`degrees` logical, whether `alpha` is given in degrees

Returns: The point on the circle with polar angle `alpha`.

Method `diameter()`: Diameter of the reference circle for a given polar angle.

Usage:

```
Circle$diameter(alpha)
```

Arguments:

`alpha` an angle in radians, there is one diameter for each value of `alpha` modulo `pi`

Returns: A segment (`Line` object).

Examples:

```
circ <- Circle$new(c(1,1), 5)
diams <- lapply(c(0, pi/3, 2*pi/3), circ$diameter)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-5,7),
     xlab = NA, ylab = NA)
draw(circ, lwd = 2, col = "yellow")
invisible(lapply(diams, draw, col = "blue"))
```

Method `tangent()`: Tangent of the reference circle at a given polar angle.

Usage:

```
Circle$tangent(alpha)
```

Arguments:

`alpha` an angle in radians, there is one tangent for each value of `alpha` modulo `2*pi`

Examples:

```
circ <- Circle$new(c(1,1), 5)
tangents <- lapply(c(0, pi/3, 2*pi/3, pi, 4*pi/3, 5*pi/3), circ$tangent)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-5,7),
     xlab = NA, ylab = NA)
draw(circ, lwd = 2, col = "yellow")
invisible(lapply(tangents, draw, col = "blue"))
```

Method `tangentsThroughExternalPoint()`: Return the two tangents of the reference circle passing through an external point.

Usage:

```
Circle$tangentsThroughExternalPoint(P)
```

Arguments:

`P` a point external to the reference circle

Returns: A list of two `Line` objects, the two tangents; the tangency points are in the `B` field of the lines.

Method `isEqual()`: Check whether the reference circle equals another circle.

Usage:

```
Circle$isEqual(circ)
```

Arguments:

`circ` a `Circle` object

Method `isOrthogonal()`: Check whether the reference circle is orthogonal to a given circle

Usage:

```
Circle$isOrthogonal(circ)
```

Arguments:

`circ` a `Circle` object

Method `angle()`: Angle between the reference circle and a given circle, if they intersect.

Usage:

```
Circle$angle(circ)
```

Arguments:

`circ` a `Circle` object

Method `includes()`: Check whether a point belongs to the reference circle.

Usage:

```
Circle$includes(M)
```

Arguments:

`M` a point

Method `orthogonalThroughTwoPointsOnCircle()`: Orthogonal circle passing through two points on the reference circle.

Usage:

```
Circle$orthogonalThroughTwoPointsOnCircle(alpha1, alpha2, arc = FALSE)
```

Arguments:

`alpha1, alpha2` two angles defining two points on the reference circle

`arc` logical, whether to return only the arc at the interior of the reference circle

Returns: A `Circle` object if `arc=FALSE`, an `Arc` object if `arc=TRUE`, or a `Line` object: the diameter of the reference circle defined by the two points in case when the two angles differ by `pi`.

Examples:

```
# hyperbolic triangle
circ <- Circle$new(c(5,5), 3)
arc1 <- circ$orthogonalThroughTwoPointsOnCircle(0, 2*pi/3, arc = TRUE)
arc2 <- circ$orthogonalThroughTwoPointsOnCircle(2*pi/3, 4*pi/3, arc = TRUE)
arc3 <- circ$orthogonalThroughTwoPointsOnCircle(4*pi/3, 0, arc = TRUE)
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type = "n", asp = 1, xlim = c(2,8), ylim = c(2,8))
draw(circ)
draw(arc1, col = "red", lwd = 2)
draw(arc2, col = "green", lwd = 2)
draw(arc3, col = "blue", lwd = 2)
par(opar)
```

Method `orthogonalThroughTwoPointsWithinCircle()`**:** Orthogonal circle passing through two points within the reference circle.

Usage:

```
Circle$orthogonalThroughTwoPointsWithinCircle(P1, P2, arc = FALSE)
```

Arguments:

`P1, P2` two distinct points in the interior of the reference circle

`arc` logical, whether to return the arc joining the two points instead of the circle

Returns: A `Circle` object or an `Arc` object, or a `Line` object if the two points are on a diameter.

Examples:

```
circ <- Circle$new(c(0,0),3)
P1 <- c(1,1); P2 <- c(1, 2)
ocirc <- circ$orthogonalThroughTwoPointsWithinCircle(P1, P2)
arc <- circ$orthogonalThroughTwoPointsWithinCircle(P1, P2, arc = TRUE)
plot(0, 0, type = "n", asp = 1, xlab = NA, ylab = NA,
     xlim = c(-3, 4), ylim = c(-3, 4))
draw(circ, lwd = 2)
draw(ocirc, lty = "dashed", lwd = 2)
draw(arc, lwd = 3, col = "blue")
```

Method `power()`**:** Power of a point with respect to the reference circle.

Usage:

```
Circle$power(M)
```

Arguments:

`M` point

Returns: A number.

Method `radicalCenter()`**:** Radical center of two circles.

Usage:

```
Circle$radicalCenter(circ2)
```

Arguments:

`circ2` a `Circle` object

Method `radicalAxis()`**:** Radical axis of two circles.

Usage:

```
Circle$radicalAxis(circ2)
```

Arguments:

`circ2` a `Circle` object

Returns: A `Line` object.

Method `rotate()`**:** Rotate the reference circle.

Usage:

```
Circle$rotate(alpha, O, degrees = TRUE)
```

Arguments:

`alpha` angle of rotation

`O` center of rotation

`degrees` logical, whether `alpha` is given in degrees

Returns: A `Circle` object.

Method `translate()`**:** Translate the reference circle.

Usage:

```
Circle$translate(v)
```

Arguments:

`v` the vector of translation

Returns: A `Circle` object.

Method `invert()`**:** Invert the reference circle.

Usage:

```
Circle$invert(inversion)
```

Arguments:

`inversion` an `Inversion` object

Returns: A `Circle` object or a `Line` object.

Method `asEllipse()`**:** Convert the reference circle to an `Ellipse` object.

Usage:

```
Circle$asEllipse()
```

Method `randomPoints()`**:** Random points on or in the reference circle.

Usage:

```
Circle$randomPoints(n, where = "in")
```

Arguments:

`n` an integer, the desired number of points

`where` `"in"` to generate inside the circle, `"on"` to generate on the circle

Returns: The generated points in a two columns matrix with n rows.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Circle$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

radicalCenter for the radical center of three circles.

Examples

```
## ------------------------------------------------
## Method `Circle$new`
## ------------------------------------------------

circ <- Circle$new(c(1,1), 1)
circ
circ$center
circ$center <- c(0,0)
circ

## ------------------------------------------------
## Method `Circle$print`
## ------------------------------------------------

Circle$new(c(0,0), 2)

## ------------------------------------------------
## Method `Circle$diameter`
## ------------------------------------------------

circ <- Circle$new(c(1,1), 5)
diams <- lapply(c(0, pi/3, 2*pi/3), circ$diameter)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-5,7),
     xlab = NA, ylab = NA)
draw(circ, lwd = 2, col = "yellow")
invisible(lapply(diams, draw, col = "blue"))

## ------------------------------------------------
## Method `Circle$tangent`
## ------------------------------------------------

circ <- Circle$new(c(1,1), 5)
tangents <- lapply(c(0, pi/3, 2*pi/3, pi, 4*pi/3, 5*pi/3), circ$tangent)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-5,7),
     xlab = NA, ylab = NA)
draw(circ, lwd = 2, col = "yellow")
```

```
invisible(lapply(tangents, draw, col = "blue"))

## ------------------------------------------------
## Method `Circle$orthogonalThroughTwoPointsOnCircle`
## ------------------------------------------------

# hyperbolic triangle
circ <- Circle$new(c(5,5), 3)
arc1 <- circ$orthogonalThroughTwoPointsOnCircle(0, 2*pi/3, arc = TRUE)
arc2 <- circ$orthogonalThroughTwoPointsOnCircle(2*pi/3, 4*pi/3, arc = TRUE)
arc3 <- circ$orthogonalThroughTwoPointsOnCircle(4*pi/3, 0, arc = TRUE)
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type = "n", asp = 1, xlim = c(2,8), ylim = c(2,8))
draw(circ)
draw(arc1, col = "red", lwd = 2)
draw(arc2, col = "green", lwd = 2)
draw(arc3, col = "blue", lwd = 2)
par(opar)

## ------------------------------------------------
## Method `Circle$orthogonalThroughTwoPointsWithinCircle`
## ------------------------------------------------

circ <- Circle$new(c(0,0),3)
P1 <- c(1,1); P2 <- c(1, 2)
ocirc <- circ$orthogonalThroughTwoPointsWithinCircle(P1, P2)
arc <- circ$orthogonalThroughTwoPointsWithinCircle(P1, P2, arc = TRUE)
plot(0, 0, type = "n", asp = 1, xlab = NA, ylab = NA,
     xlim = c(-3, 4), ylim = c(-3, 4))
draw(circ, lwd = 2)
draw(ocirc, lty = "dashed", lwd = 2)
draw(arc, lwd = 3, col = "blue")
```

CircleAB *Circle given by a diameter*

Description

Return the circle given by a diameter

Usage

```
CircleAB(A, B)
```

Arguments

A, B	the endpoints of the diameter

Value

A Circle object.

CircleOA *Circle given by its center and a point*

Description

Return the circle given by its center and a point it passes through.

Usage

```
CircleOA(O, A)
```

Arguments

O	the center of the circle
A	a point of the circle

Value

A `Circle` object.

draw *Draw a geometric object*

Description

Draw a geometric object on the current plot.

Usage

```
draw(x, ...)

## S3 method for class 'Triangle'
draw(x, ...)

## S3 method for class 'Circle'
draw(x, npoints = 100L, ...)

## S3 method for class 'Arc'
draw(x, npoints = 100L, ...)

## S3 method for class 'Ellipse'
draw(x, npoints = 100L, ...)

## S3 method for class 'EllipticalArc'
draw(x, npoints = 100L, ...)
```

```
## S3 method for class 'Line'
draw(x, ...)
```

Arguments

x	geometric object (Triangle, Circle, Line, Ellipse, Arc, EllipticalArc)
...	arguments passed to lines for a Triangle object, an Arc object or an ElipticalArc object, to polypath for a Circle object or an Ellipse object, general graphical parameters for a Line object, passed to lines, curve, or abline.
npoints	integer, the number of points of the path

Examples

```
# open new plot window
plot(0, 0, type="n", asp = 1, xlim = c(0,2.5), ylim = c(0,2.5),
     xlab = NA, ylab = NA)
grid()
# draw a triangle
t <- Triangle$new(c(0,0), c(1,0), c(0.5,sqrt(3)/2))
draw(t, col = "blue", lwd = 2)
draw(t$rotate(90, t$C), col = "green", lwd = 2)
# draw a circle
circ <- t$incircle()
draw(circ, col = "orange", border = "brown", lwd = 2)
# draw an ellipse
S <- Scaling$new(circ$center, direction = c(2,1), scale = 2)
draw(S$scaleCircle(circ), border = "grey", lwd = 2)
# draw a line
l <- Line$new(c(1,1), c(1.5,1.5), FALSE, TRUE)
draw(l, col = "red", lwd = 2)
perp <- l$perpendicular(c(2,1))
draw(perp, col = "yellow", lwd = 2)
```

---

ellint2 *Extended elliptic integral of the second kind*

Description

Evaluates the extended incomplete elliptic integral of the second kind. The function is vectorized in m but not in phi.

Usage

```
ellint2(phi, m)
```

Arguments

`phi`	amplitude, a number
`m`	values of the parameter, a vector of numbers lower than `1/sin(phi)^2` (`NaN` is returned if this condition is not satisfied)

Details

For `-pi/2` <= `phi` <= `pi/2`, this is the integral of `sqrt(1 -m*sin(t)^2)` for `t` between `0` and `phi`. Then the function is extended to arbitrary `phi` by the formula `ellint2(phi + k*pi,m) = 2*k*ellint2(pi/2,m) + ellint2(phi,m)` for any integer `k`.

Value

A numeric vector of the same length as `m`.

Note

This function is used to calculate the length of an elliptical arc (method `length` of `EllipticalArc`).

Examples

```
phi <- pi/4; m <- 0.6
ellint2(phi, m)
gsl::ellint_E(phi, sqrt(m))
curve(ellint2(phi, x), -5, 1/sin(phi)^2)
```

`Ellipse` *R6 class representing an ellipse*

Description

An ellipse is given by a center, two radii (`rmajor` and `rminor`), and the angle (`alpha`) between the major axis and the horizontal direction.

Active bindings

`center` get or set the center

`rmajor` get or set the major radius of the ellipse

`rminor` get or set the minor radius of the ellipse

`alpha` get or set the angle of the ellipse

`degrees` get or set the `degrees` field

Methods

Public methods:

- `Ellipse$new()`
- `Ellipse$print()`
- `Ellipse$isEqual()`
- `Ellipse$equation()`
- `Ellipse$includes()`
- `Ellipse$contains()`
- `Ellipse$matrix()`
- `Ellipse$path()`
- `Ellipse$diameter()`
- `Ellipse$pointFromAngle()`
- `Ellipse$pointFromEccentricAngle()`
- `Ellipse$semiMajorAxis()`
- `Ellipse$semiMinorAxis()`
- `Ellipse$foci()`
- `Ellipse$tangent()`
- `Ellipse$normal()`
- `Ellipse$theta2t()`
- `Ellipse$regressionLines()`
- `Ellipse$boundingbox()`
- `Ellipse$randomPoints()`
- `Ellipse$clone()`

Method `new()`**:** Create a new `Ellipse` object.

Usage:

```
Ellipse$new(center, rmajor, rminor, alpha, degrees = TRUE)
```

Arguments:

`center` a point, the center of the rotation

`rmajor` positive number, the major radius

`rminor` positive number, the minor radius

`alpha` a number, the angle between the major axis and the horizontal direction

`degrees` logical, whether `alpha` is given in degrees

Returns: A new `Ellipse` object.

Examples:

```
Ellipse$new(c(1,1), 3, 2, 30)
```

Method `print()`**:** Show instance of an `Ellipse` object.

Usage:

```
Ellipse$print(...)
```

Arguments:

`...` ignored

Method `isEqual()`**:** Check whether the reference ellipse equals an ellipse.

Usage:

```
Ellipse$isEqual(ell)
```

Arguments:

`ell` An `Ellipse` object.

Method `equation()`**:** The coefficients of the implicit equation of the ellipse.

Usage:

```
Ellipse$equation()
```

Details: The implicit equation of the ellipse is `Ax² + Bxy + Cy² + Dx + Ey + F = 0`. This method returns A, B, C, D, E and F.

Returns: A named numeric vector.

Method `includes()`**:** Check whether a point lies on the reference ellipse.

Usage:

```
Ellipse$includes(M)
```

Arguments:

`M` a point

Method `contains()`**:** Check whether a point is contained in the reference ellipse.

Usage:

```
Ellipse$contains(M)
```

Arguments:

`M` a point

Method `matrix()`**:** Returns the 2x2 matrix `S` associated to the reference ellipse. The equation of the ellipse is `t(M-O) %*% S %*% (M-O) = 1`.

Usage:

```
Ellipse$matrix()
```

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 1, 30)
S <- ell$matrix()
O <- ell$center
pts <- ell$path(4L) # four points on the ellipse
apply(pts, 1L, function(M) t(M-O) %*% S %*% (M-O))
```

Method `path()`**:** Path that forms the reference ellipse.

Usage:

```
Ellipse$path(npoints = 100L)
```

Arguments:

`npoints` number of points of the path

Returns: A matrix with two columns `x` and `y` of length `npoints`.

Method `diameter()`**:** Diameter and conjugate diameter of the reference ellipse.

Usage:

```
Ellipse$diameter(t, conjugate = FALSE)
```

Arguments:

`t` a number, the diameter only depends on `t` modulo `pi`; the axes correspond to `t=0` and `t=pi/2`

`conjugate` logical, whether to return the conjugate diameter as well

Returns: A `Line` object or a list of two `Line` objects if `conjugate = TRUE`.

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 2, 30)
diameters <- lapply(c(0, pi/3, 2*pi/3), ell$diameter)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell)
invisible(lapply(diameters, draw))
```

Method `pointFromAngle()`**:** Intersection point of the ellipse with the half-line starting at the ellipse center and forming angle `theta` with the major axis.

Usage:

```
Ellipse$pointFromAngle(theta, degrees = TRUE)
```

Arguments:

`theta` a number, the angle, or a numeric vector

`degrees` logical, whether `theta` is given in degrees

Returns: A point of the ellipse if `length(theta)==1` or a two-column matrix of points of the ellipse if `length(theta) > 1` (one point per row).

Method `pointFromEccentricAngle()`**:** Point of the ellipse with given eccentric angle.

Usage:

```
Ellipse$pointFromEccentricAngle(t)
```

Arguments:

`t` a number, the eccentric angle in radians, or a numeric vector

Returns: A point of the ellipse if `length(t)==1` or a two-column matrix of points of the ellipse if `length(t) > 1` (one point per row).

Method `semiMajorAxis()`**:** Semi-major axis of the ellipse.

Usage:

```
Ellipse$semiMajorAxis()
```

Returns: A segment (`Line` object).

Method `semiMinorAxis()`**:** Semi-minor axis of the ellipse.

Usage:

```
Ellipse$semiMinorAxis()
```

Returns: A segment (`Line` object).

Method `foci()`**:** Foci of the reference ellipse.

Usage:

```
Ellipse$foci()
```

Returns: A list with the two foci.

Method `tangent()`**:** Tangents of the reference ellipse at a point given by its eccentric angle.

Usage:

```
Ellipse$tangent(t)
```

Arguments:

`t` eccentric angle, there is one tangent for each value of `t` modulo `2*pi`; for `t = 0,pi/2,pi,-pi/2`, these are the tangents at the vertices of the ellipse

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 2, 30)
tangents <- lapply(c(0, pi/3, 2*pi/3, pi, 4*pi/3, 5*pi/3), ell$tangent)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, col = "yellow")
invisible(lapply(tangents, draw, col = "blue"))
```

Method `normal()`**:** Normal unit vector to the ellipse.

Usage:

```
Ellipse$normal(t)
```

Arguments:

`t` a number, the eccentric angle in radians of the point of the ellipse at which we want the normal unit vector

Returns: The normal unit vector to the ellipse at the point given by eccentric angle `t`.

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 2, 30)
t_ <- seq(0, 2*pi, length.out = 13)[-1]
plot(NULL, asp = 1, xlim = c(-5,7), ylim = c(-3,5),
     xlab = NA, ylab = NA)
draw(ell, col = "magenta")
for(i in 1:length(t_)){
  t <- t_[i]
  P <- ell$pointFromEccentricAngle(t)
  v <- ell$normal(t)
  draw(Line$new(P, P+v, FALSE, FALSE))
}
```

Method `theta2t()`**:** Convert angle to eccentric angle.

Usage:

```
Ellipse$theta2t(theta, degrees = TRUE)
```

Arguments:

`theta` angle between the major axis and the half-line starting at the center of the ellipse and passing through the point of interest on the ellipse

`degrees` logical, whether `theta` is given in degrees

Returns: The eccentric angle of the point of interest on the ellipse, in radians.

Examples:

```
O <- c(1, 1)
ell <- Ellipse$new(O, 5, 2, 30)
theta <- 20
P <- ell$pointFromAngle(theta)
t <- ell$theta2t(theta)
tg <- ell$tangent(t)
OP <- Line$new(O, P, FALSE, FALSE)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,5),
     xlab = NA, ylab = NA)
draw(ell, col = "antiquewhite")
points(P[1], P[2], pch = 19)
draw(tg, col = "red")
draw(OP)
draw(ell$semiMajorAxis())
text(t(O+c(1,0.9)), expression(theta))
```

Method `regressionLines()`**:** Regression lines. The regression line of y on x intersects the ellipse at its rightmost point and its leftmost point. The tangents at these points are vertical. The regression line of x on y intersects the ellipse at its topmost point and its bottommost point. The tangents at these points are horizontal.

Usage:

```
Ellipse$regressionLines()
```

Returns: A list with two `Line` objects: the regression line of y on x and the regression line of x on y.

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 2, 30)
reglines <- ell$regressionLines()
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, lwd = 2)
draw(reglines$YonX, lwd = 2, col = "blue")
draw(reglines$XonY, lwd = 2, col = "green")
```

Method `boundingbox()`**:** Return the smallest rectangle parallel to the axes which contains the reference ellipse.

Usage:

```
Ellipse$boundingbox()
```

Returns: A list with two components: the x-limits in `x` and the y-limits in `y`.

Examples:

```
ell <- Ellipse$new(c(2,2), 5, 3, 40)
box <- ell$boundingbox()
plot(NULL, asp = 1, xlim = box$x, ylim = box$y, xlab = NA, ylab = NA)
draw(ell, col = "seaShell", border = "blue")
abline(v = box$x, lty = 2); abline(h = box$y, lty = 2)
```

Method `randomPoints()`**:** Random points on or in the reference ellipse.

Usage:

```
Ellipse$randomPoints(n, where = "in")
```

Arguments:

`n` an integer, the desired number of points

`where` `"in"` to generate inside the ellipse, `"on"` to generate on the ellipse

Returns: The generated points in a two columns matrix with `n` rows.

Examples:

```
ell <- Ellipse$new(c(1,1), 5, 2, 30)
pts <- ell$randomPoints(100)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, lwd = 2)
points(pts, pch = 19, col = "blue")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Ellipse$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Ellipse$new`
## ------------------------------------------------

Ellipse$new(c(1,1), 3, 2, 30)

## ------------------------------------------------
## Method `Ellipse$matrix`
## ------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 1, 30)
S <- ell$matrix()
O <- ell$center
pts <- ell$path(4L) # four points on the ellipse
apply(pts, 1L, function(M) t(M-O) %*% S %*% (M-O))

## ------------------------------------------------
```

```
## Method `Ellipse$diameter`
## ---------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 2, 30)
diameters <- lapply(c(0, pi/3, 2*pi/3), ell$diameter)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell)
invisible(lapply(diameters, draw))

## ---------------------------------------------------
## Method `Ellipse$tangent`
## ---------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 2, 30)
tangents <- lapply(c(0, pi/3, 2*pi/3, pi, 4*pi/3, 5*pi/3), ell$tangent)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, col = "yellow")
invisible(lapply(tangents, draw, col = "blue"))

## ---------------------------------------------------
## Method `Ellipse$normal`
## ---------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 2, 30)
t_ <- seq(0, 2*pi, length.out = 13)[-1]
plot(NULL, asp = 1, xlim = c(-5,7), ylim = c(-3,5),
     xlab = NA, ylab = NA)
draw(ell, col = "magenta")
for(i in 1:length(t_)){
  t <- t_[i]
  P <- ell$pointFromEccentricAngle(t)
  v <- ell$normal(t)
  draw(Line$new(P, P+v, FALSE, FALSE))
}

## ---------------------------------------------------
## Method `Ellipse$theta2t`
## ---------------------------------------------------

O <- c(1, 1)
ell <- Ellipse$new(O, 5, 2, 30)
theta <- 20
P <- ell$pointFromAngle(theta)
t <- ell$theta2t(theta)
tg <- ell$tangent(t)
OP <- Line$new(O, P, FALSE, FALSE)
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,5),
     xlab = NA, ylab = NA)
draw(ell, col = "antiquewhite")
points(P[1], P[2], pch = 19)
draw(tg, col = "red")
```

```
draw(OP)
draw(ell$semiMajorAxis())
text(t(O+c(1,0.9)), expression(theta))

## ------------------------------------------------
## Method `Ellipse$regressionLines`
## ------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 2, 30)
reglines <- ell$regressionLines()
plot(NULL, asp = 1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, lwd = 2)
draw(reglines$YonX, lwd = 2, col = "blue")
draw(reglines$XonY, lwd = 2, col = "green")

## ------------------------------------------------
## Method `Ellipse$boundingbox`
## ------------------------------------------------

ell <- Ellipse$new(c(2,2), 5, 3, 40)
box <- ell$boundingbox()
plot(NULL, asp = 1, xlim = box$x, ylim = box$y, xlab = NA, ylab = NA)
draw(ell, col = "seaShell", border = "blue")
abline(v = box$x, lty = 2); abline(h = box$y, lty = 2)

## ------------------------------------------------
## Method `Ellipse$randomPoints`
## ------------------------------------------------

ell <- Ellipse$new(c(1,1), 5, 2, 30)
pts <- ell$randomPoints(100)
plot(NULL, type="n", asp=1, xlim = c(-4,6), ylim = c(-2,4),
     xlab = NA, ylab = NA)
draw(ell, lwd = 2)
points(pts, pch = 19, col = "blue")
```

EllipseEquationFromFivePoints
Ellipse equation from five points

Description

The coefficients of the implicit equation of an ellipse from five points on this ellipse.

Usage

```
EllipseEquationFromFivePoints(P1, P2, P3, P4, P5)
```

Arguments

`P1, P2, P3, P4, P5`	the five points

Details

The implicit equation of the ellipse is $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$. This function returns A, B, C, D, E and F.

Value

A named numeric vector.

Examples

```
ell <- Ellipse$new(c(2,3), 5, 4, 30)
set.seed(666)
pts <- ell$randomPoints(5, "on")
cf1 <- EllipseEquationFromFivePoints(pts[1,],pts[2,],pts[3,],pts[4,],pts[5,])
cf2 <- ell$equation() # should be the same up to a multiplicative factor
all.equal(cf1/cf1["F"], cf2/cf2["F"])
```

`EllipseFromCenterAndMatrix` *Ellipse from center and matrix*

Description

Returns the ellipse of equation `t(X-center) %*% S %*% (X-center) = 1`.

Usage

```
EllipseFromCenterAndMatrix(center, S)
```

Arguments

`center`	a point, the center of the ellipse
`S`	a positive symmetric matrix

Value

An `Ellipse` object.

Examples

```
ell <- Ellipse$new(c(2,3), 4, 2, 20)
S <- ell$matrix()
EllipseFromCenterAndMatrix(ell$center, S)
```

`EllipseFromEquation` *Ellipse from its implicit equation*

Description

Return an ellipse from the coefficients of its implicit equation.

Usage

```
EllipseFromEquation(A, B, C, D, E, F)
```

Arguments

`A, B, C, D, E, F`	the coefficients of the equation

Details

The implicit equation of the ellipse is $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$. This function returns the ellipse given A, B, C, D, E and F.

Value

An `Ellipse` object.

Examples

```
ell <- Ellipse$new(c(2,3), 5, 4, 30)
cf <- ell$equation()
ell2 <- EllipseFromEquation(cf[1], cf[2], cf[3], cf[4], cf[5], cf[6])
ell$isEqual(ell2)
```

`EllipseFromFivePoints` *Ellipse from five points*

Description

Return an ellipse from five given points on this ellipse.

Usage

```
EllipseFromFivePoints(P1, P2, P3, P4, P5)
```

Arguments

`P1, P2, P3, P4, P5`	the five points

Value

An `Ellipse` object.

Examples

```
ell <- Ellipse$new(c(2,3), 5, 4, 30)
set.seed(666)
pts <- ell$randomPoints(5, "on")
ell2 <- EllipseFromFivePoints(pts[1,],pts[2,],pts[3,],pts[4,],pts[5,])
ell$isEqual(ell2)
```

`EllipticalArc` *R6 class representing an elliptical arc*

Description

An arc is given by an ellipse (`Ellipse` object), a starting angle and an ending angle. They are respectively named `ell`, `alpha1` and `alpha2`.

Active bindings

`ell` get or set the ellipse

`alpha1` get or set the starting angle

`alpha2` get or set the ending angle

`degrees` get or set the `degrees` field

Methods

Public methods:

- `EllipticalArc$new()`
- `EllipticalArc$print()`
- `EllipticalArc$startingPoint()`
- `EllipticalArc$endingPoint()`
- `EllipticalArc$isEqual()`
- `EllipticalArc$complementaryArc()`
- `EllipticalArc$path()`
- `EllipticalArc$length()`
- `EllipticalArc$clone()`

Method `new()`**:** Create a new `EllipticalArc` object.

Usage:

```
EllipticalArc$new(ell, alpha1, alpha2, degrees = TRUE)
```

Arguments:

`ell` the ellipse

`alpha1` the starting angle

`alpha2` the ending angle

`degrees` logical, whether `alpha1` and `alpha2` are given in degrees

Returns: A new `EllipticalArc` object.

Examples:

```
ell <- Ellipse$new(c(-4,0), 4, 2.5, 140)
EllipticalArc$new(ell, 45, 90)
```

Method `print()`**:** Show instance of an `EllipticalArc` object.

Usage:

```
EllipticalArc$print(...)
```

Arguments:

`...` ignored

Method `startingPoint()`**:** Starting point of the reference elliptical arc.

Usage:

```
EllipticalArc$startingPoint()
```

Method `endingPoint()`**:** Ending point of the reference elliptical arc.

Usage:

```
EllipticalArc$endingPoint()
```

Method `isEqual()`**:** Check whether the reference elliptical arc equals another elliptical arc.

Usage:

```
EllipticalArc$isEqual(arc)
```

Arguments:

`arc` an `EllipticalArc` object

Method `complementaryArc()`**:** Complementary elliptical arc of the reference elliptical arc.

Usage:

```
EllipticalArc$complementaryArc()
```

Examples:

```
ell <- Ellipse$new(c(-4,0), 4, 2.5, 140)
arc <- EllipticalArc$new(ell, 30, 60)
plot(NULL, type = "n", asp = 1, xlim = c(-8,0), ylim = c(-3.2,3.2),
     xlab = NA, ylab = NA)
draw(arc, lwd = 3, col = "red")
draw(arc$complementaryArc(), lwd = 3, col = "green")
```

Method `path()`**:** The reference elliptical arc as a path.

Usage:

```
EllipticalArc$path(npoints = 100L)
```

Arguments:

`npoints` number of points of the path

Returns: A matrix with two columns `x` and `y` of length `npoints`.

Method `length()`**:** The length of the elliptical arc.

Usage:

```
EllipticalArc$length()
```

Returns: A number, the arc length.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EllipticalArc$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `EllipticalArc$new`
## ------------------------------------------------

ell <- Ellipse$new(c(-4,0), 4, 2.5, 140)
EllipticalArc$new(ell, 45, 90)

## ------------------------------------------------
## Method `EllipticalArc$complementaryArc`
## ------------------------------------------------

ell <- Ellipse$new(c(-4,0), 4, 2.5, 140)
arc <- EllipticalArc$new(ell, 30, 60)
plot(NULL, type = "n", asp = 1, xlim = c(-8,0), ylim = c(-3.2,3.2),
     xlab = NA, ylab = NA)
draw(arc, lwd = 3, col = "red")
draw(arc$complementaryArc(), lwd = 3, col = "green")
```

`GaussianEllipse` *Gaussian ellipse*

Description

Return the ellipse equal to the highest *pdf* region of a bivariate Gaussian distribution with a given probability.

Usage

```
GaussianEllipse(mean, Sigma, p)
```

Arguments

`mean`	numeric vector of length 2, the mean of the bivariate Gaussian distribution; this is the center of the ellipse
`Sigma`	covariance matrix of the bivariate Gaussian distribution
`p`	desired probability level, a number between 0 and 1 (strictly)

Value

An `Ellipse` object.

`Homothety` *R6 class representing a homothety*

Description

A homothety is given by a center and a scale factor.

Active bindings

`center` get or set the center

`scale` get or set the scale factor of the homothety

Methods

Public methods:

- `Homothety$new()`
- `Homothety$print()`
- `Homothety$transform()`
- `Homothety$transformCircle()`
- `Homothety$getMatrix()`
- `Homothety$asAffine()`
- `Homothety$clone()`

Method `new()`: Create a new `Homothety` object.

Usage:

```
Homothety$new(center, scale)
```

Arguments:

`center` a point, the center of the homothety

`scale` a number, the scale factor of the homothety

Returns: A new `Homothety` object.

Examples:

```
Homothety$new(c(1,1), 2)
```

Method `print()`**:** Show instance of a `Homothety` object.

Usage:

```
Homothety$print(...)
```

Arguments:

`...` ignored

Method `transform()`**:** Transform a point or several points by the reference homothety.

Usage:

```
Homothety$transform(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `transformCircle()`**:** Transform a circle by the reference homothety.

Usage:

```
Homothety$transformCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object.

Method `getMatrix()`**:** Augmented matrix of the homothety.

Usage:

```
Homothety$getMatrix()
```

Returns: A 3x3 matrix.

Examples:

```
H <- Homothety$new(c(1,1), 2)
P <- c(1,5)
H$transform(P)
H$getMatrix() %*% c(P,1)
```

Method `asAffine()`**:** Convert the reference homothety to an `Affine` object.

Usage:

```
Homothety$asAffine()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Homothety$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Homothety$new`
## ------------------------------------------------

Homothety$new(c(1,1), 2)

## ------------------------------------------------
## Method `Homothety$getMatrix`
## ------------------------------------------------

H <- Homothety$new(c(1,1), 2)
P <- c(1,5)
H$transform(P)
H$getMatrix() %*% c(P,1)
```

---

intersectionCircleCircle — *Intersection of two circles*

Description

Return the intersection of two circles.

Usage

```
intersectionCircleCircle(circ1, circ2, epsilon = sqrt(.Machine$double.eps))
```

Arguments

`circ1, circ2`	two `Circle` objects
`epsilon`	a small positive number used for the numerical accuracy

Value

`NULL` if there is no intersection, a point if the circles touch, a list of two points if the circles meet at two points, a circle if the two circles are identical.

intersectionCircleLine

Intersection of a circle and a line

Description

Return the intersection of a circle and a line.

Usage

```
intersectionCircleLine(circ, line, strict = FALSE)
```

Arguments

circ	a Circle object
line	a Line object
strict	logical, whether to take into account line$extendA and line$extendB if they are not both TRUE

Value

NULL if there is no intersection; a point if the infinite line is tangent to the circle, or NULL if strict=TRUE and the point is not on the line (segment or half-line); a list of two points if the circle and the infinite line meet at two points, when strict=FALSE; if strict=TRUE and the line is a segment or a half-line, this can return NULL or a single point.

Examples

```
circ <- Circle$new(c(1,1), 2)
line <- Line$new(c(2,-2), c(1,2), FALSE, FALSE)
intersectionCircleLine(circ, line)
intersectionCircleLine(circ, line, strict = TRUE)
```

intersectionEllipseLine

Intersection of an ellipse and a line

Description

Return the intersection of an ellipse and a line.

Usage

```
intersectionEllipseLine(ell, line, strict = FALSE)
```

Arguments

`ell`	an `Ellipse` object or a `Circle` object
`line`	a `Line` object
`strict`	logical, whether to take into account `line$extendA` and `line$extendB` if they are not both `TRUE`

Value

`NULL` if there is no intersection; a point if the infinite line is tangent to the ellipse, or `NULL` if `strict=TRUE` and the point is not on the line (segment or half-line); a list of two points if the ellipse and the infinite line meet at two points, when `strict=FALSE`; if `strict=TRUE` and the line is a segment or a half-line, this can return `NULL` or a single point.

Examples

```
ell <- Ellipse$new(c(1,1), 5, 1, 30)
line <- Line$new(c(2,-2), c(0,4))
( Is <- intersectionEllipseLine(ell, line) )
ell$includes(Is$I1); ell$includes(Is$I2)
```

`intersectionLineLine` *Intersection of two lines*

Description

Return the intersection of two lines.

Usage

```
intersectionLineLine(line1, line2, strict = FALSE)
```

Arguments

`line1`, `line2`	two `Line` objects
`strict`	logical, whether to take into account the extensions of the lines (`extendA` and `extendB`)

Value

If `strict = FALSE` this returns either a point, or `NULL` if the lines are parallel, or a bi-infinite line if the two lines coincide. If `strict = TRUE`, this can also return a half-infinite line or a segment.

`Inversion` *R6 class representing an inversion*

Description

An inversion is given by a pole (a point) and a power (a number, possibly negative, but not zero).

Active bindings

`pole` get or set the pole

`power` get or set the power

Methods

Public methods:

- `Inversion$new()`
- `Inversion$print()`
- `Inversion$invert()`
- `Inversion$transform()`
- `Inversion$invertCircle()`
- `Inversion$transformCircle()`
- `Inversion$invertLine()`
- `Inversion$transformLine()`
- `Inversion$compose()`
- `Inversion$clone()`

Method `new()`: Create a new `Inversion` object.

Usage:

```
Inversion$new(pole, power)
```

Arguments:

`pole` the pole

`power` the power

Returns: A new `Inversion` object.

Method `print()`: Show instance of an inversion object.

Usage:

```
Inversion$print(...)
```

Arguments:

`...` ignored

Examples:

```
Inversion$new(c(0,0), 2)
```

Method `invert()`**:** Inversion of a point.

Usage:

```
Inversion$invert(M)
```

Arguments:

`M` a point or `Inf`

Returns: A point or `Inf`, the image of `M`.

Method `transform()`**:** An alias of `invert`.

Usage:

```
Inversion$transform(M)
```

Arguments:

`M` a point or `Inf`

Returns: A point or `Inf`, the image of `M`.

Method `invertCircle()`**:** Inversion of a circle.

Usage:

```
Inversion$invertCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object or a `Line` object.

Examples:

```
# A Pappus chain
# https://www.cut-the-knot.org/Curriculum/Geometry/InversionInArbelos.shtml
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type = "n", asp = 1, xlim = c(0,6), ylim = c(-4,4),
     xlab = NA, ylab = NA, axes = FALSE)
A <- c(0,0); B <- c(6,0)
ABsqr <- c(crossprod(A-B))
iota <- Inversion$new(A, ABsqr)
C <- iota$invert(c(8,0))
Sigma1 <- Circle$new((A+B)/2, sqrt(ABsqr)/2)
Sigma2 <- Circle$new((A+C)/2, sqrt(c(crossprod(A-C)))/2)
draw(Sigma1); draw(Sigma2)
circ0 <- Circle$new(c(7,0), 1)
iotacirc0 <- iota$invertCircle(circ0)
draw(iotacirc0)
for(i in 1:6){
  circ <- circ0$translate(c(0,2*i))
  iotacirc <- iota$invertCircle(circ)
  draw(iotacirc)
  circ <- circ0$translate(c(0,-2*i))
  iotacirc <- iota$invertCircle(circ)
  draw(iotacirc)
}
par(opar)
```

Method `transformCircle()`: An alias of `invertCircle`.

Usage:

```
Inversion$transformCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object or a `Line` object.

Method `invertLine()`: Inversion of a line.

Usage:

```
Inversion$invertLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Circle` object or a `Line` object.

Method `transformLine()`: An alias of `invertLine`.

Usage:

```
Inversion$transformLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Circle` object or a `Line` object.

Method `compose()`: Compose the reference inversion with another inversion. The result is a Möbius transformation.

Usage:

```
Inversion$compose(iota1, left = TRUE)
```

Arguments:

`iota1` an `Inversion` object

`left` logical, whether to compose at left or at right (i.e. returns `iota1 o iota0` or `iota0 o iota1`)

Returns: A `Mobius` object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Inversion$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`inversionSwappingTwoCircles`, `inversionFixingTwoCircles`, `inversionFixingThreeCircles` to create some inversions.

Examples

```
## ------------------------------------------------
## Method `Inversion$print`
## ------------------------------------------------

Inversion$new(c(0,0), 2)

## ------------------------------------------------
## Method `Inversion$invertCircle`
## ------------------------------------------------

# A Pappus chain
# https://www.cut-the-knot.org/Curriculum/Geometry/InversionInArbelos.shtml
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type = "n", asp = 1, xlim = c(0,6), ylim = c(-4,4),
     xlab = NA, ylab = NA, axes = FALSE)
A <- c(0,0); B <- c(6,0)
ABsqr <- c(crossprod(A-B))
iota <- Inversion$new(A, ABsqr)
C <- iota$invert(c(8,0))
Sigma1 <- Circle$new((A+B)/2, sqrt(ABsqr)/2)
Sigma2 <- Circle$new((A+C)/2, sqrt(c(crossprod(A-C)))/2)
draw(Sigma1); draw(Sigma2)
circ0 <- Circle$new(c(7,0), 1)
iotacirc0 <- iota$invertCircle(circ0)
draw(iotacirc0)
for(i in 1:6){
  circ <- circ0$translate(c(0,2*i))
  iotacirc <- iota$invertCircle(circ)
  draw(iotacirc)
  circ <- circ0$translate(c(0,-2*i))
  iotacirc <- iota$invertCircle(circ)
  draw(iotacirc)
}
par(opar)
```

`inversionFixingThreeCircles` *Inversion fixing three circles*

Description

Return the inversion which lets invariant three given circles.

Usage

```
inversionFixingThreeCircles(circ1, circ2, circ3)
```

Arguments

`circ1, circ2, circ3`
: `Circle` objects

Value

An `Inversion` object, which lets each of `circ1`, `circ2` and `circ3` invariant.

`inversionFixingTwoCircles` *Inversion fixing two circles*

Description

Return the inversion which lets invariant two given circles.

Usage

```
inversionFixingTwoCircles(circ1, circ2)
```

Arguments

`circ1, circ2` `Circle` objects

Value

An `Inversion` object, which maps `circ1` to `circ2` and `circ2` to `circ2`.

`inversionFromCircle` *Inversion on a circle*

Description

Return the inversion on a given circle.

Usage

```
inversionFromCircle(circ)
```

Arguments

`circ` a `Circle` object

Value

An `Inversion` object

inversionKeepingCircle

Inversion keeping a circle unchanged

Description

Return an inversion with a given pole which keeps a given circle unchanged.

Usage

```
inversionKeepingCircle(pole, circ)
```

Arguments

pole	inversion pole, a point
circ	a `Circle` object

Value

An `Inversion` object.

Examples

```
circ <- Circle$new(c(4,3), 2)
iota <- inversionKeepingCircle(c(1,2), circ)
iota$transformCircle(circ)
```

inversionSwappingTwoCircles

Inversion swapping two circles

Description

Return the inversion which swaps two given circles.

Usage

```
inversionSwappingTwoCircles(circ1, circ2, positive = TRUE)
```

Arguments

circ1, circ2	`Circle` objects
positive	logical, whether the sign of the desired inversion power must be positive or negative

Value

An `Inversion` object, which maps `circ1` to `circ2` and `circ2` to `circ1`, except in the case when `circ1` and `circ2` are congruent and tangent: in this case a `Reflection` object is returned (a reflection is an inversion on a line).

`Line` *R6 class representing a line*

Description

A line is given by two distinct points, named `A` and `B`, and two logical values `extendA` and `extendB`, indicating whether the line must be extended beyond `A` and `B` respectively. Depending on `extendA` and `extendB`, the line is an infinite line, a half-line, or a segment.

Active bindings

`A` get or set the point A

`B` get or set the point B

`extendA` get or set `extendA`

`extendB` get or set `extendB`

Methods

Public methods:

- `Line$new()`
- `Line$print()`
- `Line$directionAndOffset()`
- `Line$isEqual()`
- `Line$isParallel()`
- `Line$includes()`
- `Line$perpendicular()`
- `Line$parallel()`
- `Line$projection()`
- `Line$distance()`
- `Line$reflection()`
- `Line$rotate()`
- `Line$translate()`
- `Line$invert()`
- `Line$clone()`

Method `new()`**:** Create a new `Line` object.

Usage:

```
Line$new(A, B, extendA = TRUE, extendB = TRUE)
```

Arguments:

`A, B` points

`extendA, extendB` logical values

Returns: A new `Line` object.

Examples:

```
l <- Line$new(c(1,1), c(1.5,1.5), FALSE, TRUE)
l
l$A
l$A <- c(0,0)
l
```

Method `print()`**:** Show instance of a line object.

Usage:

```
Line$print(...)
```

Arguments:

`...` ignored

Examples:

```
Line$new(c(0,0), c(1,0), FALSE, TRUE)
```

Method `directionAndOffset()`**:** Direction (angle between 0 and 2pi) and offset (positive number) of the reference line.

Usage:

```
Line$directionAndOffset()
```

Details: The equation of the line is $\cos(\theta)x + \sin(\theta)y = d$ where θ is the direction and d is the offset.

Method `isEqual()`**:** Check whether the reference line equals a given line, without taking into account `extendA` and `extendB`.

Usage:

```
Line$isEqual(line)
```

Arguments:

`line` a `Line` object

Returns: `TRUE` or `FALSE`.

Method `isParallel()`**:** Check whether the reference line is parallel to a given line.

Usage:

```
Line$isParallel(line)
```

Arguments:

`line` a `Line` object

Returns: `TRUE` or `FALSE`.

Method `includes()`**:** Whether a point belongs to the reference line.

Usage:

```
Line$includes(M, strict = FALSE, checkCollinear = TRUE)
```

Arguments:

`M` the point for which we want to test whether it belongs to the line

`strict` logical, whether to take into account `extendA` and `extendB`

`checkCollinear` logical, whether to check the collinearity of `A`, `B`, `M`; set to `FALSE` only if you are sure that `M` is on the line `(AB)` (if you use `strict=TRUE`)

Returns: `TRUE` or `FALSE`.

Examples:

```
A <- c(0,0); B <- c(1,2); M <- c(3,6)
l <- Line$new(A, B, FALSE, FALSE)
l$includes(M, strict = TRUE)
```

Method `perpendicular()`**:** Perpendicular line passing through a given point.

Usage:

```
Line$perpendicular(M, extendH = FALSE, extendM = TRUE)
```

Arguments:

`M` the point through which the perpendicular passes.

`extendH` logical, whether to extend the perpendicular line beyond the meeting point

`extendM` logical, whether to extend the perpendicular line beyond the point `M`

Returns: A `Line` object; its two points are the meeting point and the point `M`.

Method `parallel()`**:** Parallel to the reference line passing through a given point.

Usage:

```
Line$parallel(M)
```

Arguments:

`M` a point

Returns: A `Line` object.

Method `projection()`**:** Orthogonal projection of a point to the reference line.

Usage:

```
Line$projection(M)
```

Arguments:

`M` a point

Returns: A point.

Method `distance()`**:** Distance from a point to the reference line.

Usage:

```
Line$distance(M)
```

Arguments:

`M` a point

Returns: A positive number.

Method `reflection()`**:** Reflection of a point with respect to the reference line.

Usage:

```
Line$reflection(M)
```

Arguments:

`M` a point

Returns: A point.

Method `rotate()`**:** Rotate the reference line.

Usage:

```
Line$rotate(alpha, O, degrees = TRUE)
```

Arguments:

`alpha` angle of rotation

`O` center of rotation

`degrees` logical, whether `alpha` is given in degrees

Returns: A `Line` object.

Method `translate()`**:** Translate the reference line.

Usage:

```
Line$translate(v)
```

Arguments:

`v` the vector of translation

Returns: A `Line` object.

Method `invert()`**:** Invert the reference line.

Usage:

```
Line$invert(inversion)
```

Arguments:

`inversion` an `Inversion` object

Returns: A `Circle` object or a `Line` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Line$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Line$new`
## ------------------------------------------------

l <- Line$new(c(1,1), c(1.5,1.5), FALSE, TRUE)
l
l$A
l$A <- c(0,0)
l

## ------------------------------------------------
## Method `Line$print`
## ------------------------------------------------

Line$new(c(0,0), c(1,0), FALSE, TRUE)

## ------------------------------------------------
## Method `Line$includes`
## ------------------------------------------------

A <- c(0,0); B <- c(1,2); M <- c(3,6)
l <- Line$new(A, B, FALSE, FALSE)
l$includes(M, strict = TRUE)
```

LownerJohnEllipse *Löwner-John ellipse (ellipse hull)*

Description

Minimum area ellipse containing a set of points.

Usage

```
LownerJohnEllipse(pts)
```

Arguments

pts	the points in a two-columns matrix (one point per row); at least three distinct points

Value

An Ellipse object.

Examples

```
pts <- cbind(rnorm(30, sd=2), rnorm(30))
ell <- LownerJohnEllipse(pts)
box <- ell$boundingbox()
plot(NULL, asp = 1, xlim = box$x, ylim = box$y, xlab = NA, ylab = NA)
draw(ell, col = "seaShell")
points(pts, pch = 19)
all(apply(pts, 1, ell$contains)) # should be TRUE
```

---

midCircles *Mid-circle(s)*

Description

Return the mid-circle(s) of two circles.

Usage

```
midCircles(circ1, circ2)
```

Arguments

circ1, circ2	Circle objects

Details

A mid-circle of two circles is a generalized circle (i.e. a circle or a line) such that the inversion on this circle swaps the two circles. The case of a line appears only when the two circles have equal radii.

Value

A `Circle` object, or a `Line` object, or a list of two such objects.

See Also

inversionSwappingTwoCircles

Examples

```
circ1 <- Circle$new(c(5,4),2)
circ2 <- Circle$new(c(6,4),1)
midcircle <- midCircles(circ1, circ2)
inversionFromCircle(midcircle)
inversionSwappingTwoCircles(circ1, circ2)
```

Mobius *R6 class representing a Möbius transformation.*

Description

A Möbius transformation is given by a matrix of complex numbers with non-null determinant.

Active bindings

a get or set a

b get or set b

c get or set c

d get or set d

Methods

Public methods:

- Mobius$new()
- Mobius$print()
- Mobius$getM()
- Mobius$compose()
- Mobius$inverse()
- Mobius$power()
- Mobius$gpower()
- Mobius$transform()
- Mobius$transformCircle()
- Mobius$transformLine()
- Mobius$clone()

Method new()**:** Create a new Mobius object.

Usage:

```
Mobius$new(M)
```

Arguments:

M the matrix corresponding to the Möbius transformation

Returns: A new Mobius object.

Method print()**:** Show instance of a Mobius object.

Usage:

```
Mobius$print(...)
```

Arguments:

... ignored

Examples:

```
Mobius$new(rbind(c(1+1i,2),c(0,3-2i)))
```

Method `getM()`**:** Get the matrix corresponding to the Möbius transformation.

Usage:

```
Mobius$getM()
```

Method `compose()`**:** Compose the reference Möbius transformation with another Möbius transformation

Usage:

```
Mobius$compose(M1, left = TRUE)
```

Arguments:

`M1` a `Mobius` object

`left` logical, whether to compose at left or at right (i.e. returns `M1 o M0` or `M0 o M1`)

Returns: A `Mobius` object.

Method `inverse()`**:** Inverse of the reference Möbius transformation.

Usage:

```
Mobius$inverse()
```

Returns: A `Mobius` object.

Method `power()`**:** Power of the reference Möbius transformation.

Usage:

```
Mobius$power(k)
```

Arguments:

`k` an integer, possibly negative

Returns: The Möbius transformation `M^k`, where `M` is the reference Möbius transformation.

Method `gpower()`**:** Generalized power of the reference Möbius transformation.

Usage:

```
Mobius$gpower(k)
```

Arguments:

`k` a real number, possibly negative

Returns: A `Mobius` object, the generalized `k`-th power of the reference Möbius transformation.

Method `transform()`**:** Transformation of a point by the reference Möbius transformation.

Usage:

```
Mobius$transform(M)
```

Arguments:

`M` a point or `Inf`

Returns: A point or `Inf`, the image of `M`.

Examples:

```
Mob <- Mobius$new(rbind(c(1+1i,2),c(0,3-2i)))
Mob$transform(c(1,1))
Mob$transform(Inf)
```

Method `transformCircle()`**:** Transformation of a circle by the reference Möbius transformation.

Usage:

```
Mobius$transformCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object or a `Line` object.

Method `transformLine()`**:** Transformation of a line by the reference Möbius transformation.

Usage:

```
Mobius$transformLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Circle` object or a `Line` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Mobius$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`MobiusMappingThreePoints` to create a Möbius transformation, and also the `compose` method of the `Inversion` R6 class.

Examples

```
## ------------------------------------------------
## Method `Mobius$print`
## ------------------------------------------------

Mobius$new(rbind(c(1+1i,2),c(0,3-2i)))

## ------------------------------------------------
## Method `Mobius$transform`
## ------------------------------------------------

Mob <- Mobius$new(rbind(c(1+1i,2),c(0,3-2i)))
Mob$transform(c(1,1))
Mob$transform(Inf)
```

`MobiusMappingThreePoints` *Möbius transformation mapping three given points to three given points*

Description

Return a Möbius transformation which sends `P1` to `Q1`, `P2` to `Q2` and `P3` to `Q3`.

Usage

```
MobiusMappingThreePoints(P1, P2, P3, Q1, Q2, Q3)
```

Arguments

`P1, P2, P3`	three distinct points, `Inf` allowed
`Q1, Q2, Q3`	three distinct points, `Inf` allowed

Value

A `Mobius` object.

`Projection` *R6 class representing a projection*

Description

A projection on a line `D` parallel to another line `Delta` is given by the line of projection (`D`) and the directrix line (`Delta`).

Active bindings

`D` get or set the projection line

`Delta` get or set the directrix line

Methods

Public methods:

- `Projection$new()`
- `Projection$print()`
- `Projection$project()`
- `Projection$transform()`
- `Projection$getMatrix()`
- `Projection$asAffine()`

- Projection$clone()

Method `new()`: Create a new `Projection` object.

Usage:

```
Projection$new(D, Delta)
```

Arguments:

`D, Delta` two `Line` objects such that the two lines meet (not parallel); or `Delta = NULL` for orthogonal projection onto `D`

Returns: A new `Projection` object.

Examples:

```
D <- Line$new(c(1,1), c(5,5))
Delta <- Line$new(c(0,0), c(3,4))
Projection$new(D, Delta)
```

Method `print()`: Show instance of a projection object.

Usage:

```
Projection$print(...)
```

Arguments:

`...` ignored

Method `project()`: Project a point.

Usage:

```
Projection$project(M)
```

Arguments:

`M` a point

Examples:

```
D <- Line$new(c(1,1), c(5,5))
Delta <- Line$new(c(0,0), c(3,4))
P <- Projection$new(D, Delta)
M <- c(1,3)
Mprime <- P$project(M)
D$includes(Mprime) # should be TRUE
Delta$isParallel(Line$new(M, Mprime)) # should be TRUE
```

Method `transform()`: An alias of `project`.

Usage:

```
Projection$transform(M)
```

Arguments:

`M` a point

Method `getMatrix()`: Augmented matrix of the projection.

Usage:

```
Projection$getMatrix()
```

Returns: A 3x3 matrix.

Examples:

```
P <- Projection$new(Line$new(c(2,2), c(4,5)), Line$new(c(0,0), c(1,1)))
M <- c(1,5)
P$project(M)
P$getMatrix() %*% c(M,1)
```

Method `asAffine()`**:** Convert the reference projection to an `Affine` object.

Usage:

```
Projection$asAffine()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Projection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

For an orthogonal projection, you can use the `projection` method of the `Line` R6 class.

Examples

```
## ------------------------------------------------
## Method `Projection$new`
## ------------------------------------------------

D <- Line$new(c(1,1), c(5,5))
Delta <- Line$new(c(0,0), c(3,4))
Projection$new(D, Delta)

## ------------------------------------------------
## Method `Projection$project`
## ------------------------------------------------

D <- Line$new(c(1,1), c(5,5))
Delta <- Line$new(c(0,0), c(3,4))
P <- Projection$new(D, Delta)
M <- c(1,3)
Mprime <- P$project(M)
D$includes(Mprime) # should be TRUE
Delta$isParallel(Line$new(M, Mprime)) # should be TRUE

## ------------------------------------------------
## Method `Projection$getMatrix`
## ------------------------------------------------

P <- Projection$new(Line$new(c(2,2), c(4,5)), Line$new(c(0,0), c(1,1)))
```

```
M <- c(1,5)
P$project(M)
P$getMatrix() %*% c(M,1)
```

`radicalCenter` *Radical center*

Description

Returns the radical center of three circles.

Usage

```
radicalCenter(circ1, circ2, circ3)
```

Arguments

`circ1, circ2, circ3`
: `Circle` objects

Value

A point.

`Reflection` *R6 class representing a reflection*

Description

A reflection is given by a line.

Active bindings

`line` get or set the line of the reflection

Methods

Public methods:

- `Reflection$new()`
- `Reflection$print()`
- `Reflection$reflect()`
- `Reflection$transform()`
- `Reflection$reflectCircle()`
- `Reflection$transformCircle()`
- `Reflection$reflectLine()`

- Reflection$transformLine()
- Reflection$getMatrix()
- Reflection$asAffine()
- Reflection$clone()

Method `new()`**:** Create a new `Reflection` object.

Usage:

```
Reflection$new(line)
```

Arguments:

`line` a `Line` object

Returns: A new `Reflection` object.

Examples:

```
l <- Line$new(c(1,1), c(1.5,1.5), FALSE, TRUE)
Reflection$new(l)
```

Method `print()`**:** Show instance of a reflection object.

Usage:

```
Reflection$print(...)
```

Arguments:

`...` ignored

Method `reflect()`**:** Reflect a point.

Usage:

```
Reflection$reflect(M)
```

Arguments:

`M` a point, `Inf` allowed

Method `transform()`**:** An alias of `reflect`.

Usage:

```
Reflection$transform(M)
```

Arguments:

`M` a point, `Inf` allowed

Method `reflectCircle()`**:** Reflect a circle.

Usage:

```
Reflection$reflectCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object.

Method `transformCircle()`**:** An alias of `reflectCircle`.

Usage:

`Reflection$transformCircle(circ)`

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object.

Method `reflectLine()`**:** Reflect a line.

Usage:

`Reflection$reflectLine(line)`

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `transformLine()`**:** An alias of `reflectLine`.

Usage:

`Reflection$transformLine(line)`

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `getMatrix()`**:** Augmented matrix of the reflection.

Usage:

`Reflection$getMatrix()`

Returns: A 3x3 matrix.

Examples:

```
R <- Reflection$new(Line$new(c(2,2), c(4,5)))
P <- c(1,5)
R$reflect(P)
R$getMatrix() %*% c(P,1)
```

Method `asAffine()`**:** Convert the reference reflection to an `Affine` object.

Usage:

`Reflection$asAffine()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Reflection$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Reflection$new`
## ------------------------------------------------

l <- Line$new(c(1,1), c(1.5,1.5), FALSE, TRUE)
Reflection$new(l)

## ------------------------------------------------
## Method `Reflection$getMatrix`
## ------------------------------------------------

R <- Reflection$new(Line$new(c(2,2), c(4,5)))
P <- c(1,5)
R$reflect(P)
R$getMatrix() %*% c(P,1)
```

Rotation *R6 class representing a rotation*

Description

A rotation is given by an angle (`theta`) and a center.

Active bindings

`theta` get or set the angle of the rotation

`center` get or set the center

`degrees` get or set the `degrees` field

Methods

Public methods:

- `Rotation$new()`
- `Rotation$print()`
- `Rotation$rotate()`
- `Rotation$transform()`
- `Rotation$rotateCircle()`
- `Rotation$transformCircle()`
- `Rotation$rotateEllipse()`
- `Rotation$transformEllipse()`
- `Rotation$rotateLine()`
- `Rotation$transformLine()`
- `Rotation$getMatrix()`

- `Rotation$asAffine()`
- `Rotation$clone()`

Method `new()`**:** Create a new `Rotation` object.

Usage:

```
Rotation$new(theta, center, degrees = TRUE)
```

Arguments:

`theta` a number, the angle of the rotation

`center` a point, the center of the rotation

`degrees` logical, whether `theta` is given in degrees

Returns: A new `Rotation` object.

Examples:

```
Rotation$new(60, c(1,1))
```

Method `print()`**:** Show instance of a `Rotation` object.

Usage:

```
Rotation$print(...)
```

Arguments:

`...` ignored

Method `rotate()`**:** Rotate a point or several points.

Usage:

```
Rotation$rotate(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `transform()`**:** An alias of `rotate`.

Usage:

```
Rotation$transform(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `rotateCircle()`**:** Rotate a circle.

Usage:

```
Rotation$rotateCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object.

Method `transformCircle()`**:** An alias of `rotateCircle`.

Usage:

```
Rotation$transformCircle(circ)
```

Arguments:

`circ` a `Circle` object

Returns: A `Circle` object.

Method `rotateEllipse()`**:** Rotate an ellipse.

Usage:

```
Rotation$rotateEllipse(ell)
```

Arguments:

`ell` an `Ellipse` object

Returns: An `Ellipse` object.

Method `transformEllipse()`**:** An alias of `rotateEllipse`.

Usage:

```
Rotation$transformEllipse(ell)
```

Arguments:

`ell` an `Ellipse` object

Returns: An `Ellipse` object.

Method `rotateLine()`**:** Rotate a line.

Usage:

```
Rotation$rotateLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `transformLine()`**:** An alias of `rotateLine`.

Usage:

```
Rotation$transformLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `getMatrix()`**:** Augmented matrix of the rotation.

Usage:

```
Rotation$getMatrix()
```

Returns: A 3x3 matrix.

Examples:

```
R <- Rotation$new(60, c(1,1))
P <- c(1,5)
R$rotate(P)
R$getMatrix() %*% c(P,1)
```

Method `asAffine()`**:** Convert the reference rotation to an `Affine` object.

Usage:

```
Rotation$asAffine()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Rotation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Rotation$new`
## ------------------------------------------------

Rotation$new(60, c(1,1))

## ------------------------------------------------
## Method `Rotation$getMatrix`
## ------------------------------------------------

R <- Rotation$new(60, c(1,1))
P <- c(1,5)
R$rotate(P)
R$getMatrix() %*% c(P,1)
```

`Scaling` *R6 class representing a (non-uniform) scaling*

Description

A (non-uniform) scaling is given by a center, a direction vector, and a scale factor.

Active bindings

`center` get or set the center

`direction` get or set the direction

`scale` get or set the scale factor

Methods

Public methods:

- Scaling$new()
- Scaling$print()
- Scaling$transform()
- Scaling$getMatrix()
- Scaling$asAffine()
- Scaling$scaleCircle()
- Scaling$clone()

Method `new()`**:** Create a new `Scaling` object.

Usage:

```
Scaling$new(center, direction, scale)
```

Arguments:

`center` a point, the center of the scaling

`direction` a vector, the direction of the scaling

`scale` a number, the scale factor

Returns: A new `Scaling` object.

Examples:

```
Scaling$new(c(1,1), c(1,3), 2)
```

Method `print()`**:** Show instance of a `Scaling` object.

Usage:

```
Scaling$print(...)
```

Arguments:

`...` ignored

Method `transform()`**:** Transform a point or several points by the reference scaling.

Usage:

```
Scaling$transform(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `getMatrix()`**:** Augmented matrix of the scaling.

Usage:

```
Scaling$getMatrix()
```

Returns: A 3x3 matrix.

Examples:

```
S <- Scaling$new(c(1,1), c(2,3), 2)
P <- c(1,5)
S$transform(P)
S$getMatrix() %*% c(P,1)
```

Method `asAffine()`**:** Convert the reference scaling to an `Affine` object.

Usage:

`Scaling$asAffine()`

Method `scaleCircle()`**:** Scale a circle. The result is an ellipse.

Usage:

`Scaling$scaleCircle(circ)`

Arguments:

`circ` a `Circle` object

Returns: An `Ellipse` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Scaling$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

References

R. Goldman, *An Integrated Introduction to Computer Graphics and Geometric Modeling*. CRC Press, 2009.

Examples

```
Q <- c(1,1); w <- c(1,3); s <- 2
S <- Scaling$new(Q, w, s)
# the center is mapped to itself:
S$transform(Q)
# any vector \code{u} parallel to the direction vector is mapped to \code{s*u}:
u <- 3*w
all.equal(s*u, S$transform(u) - S$transform(c(0,0)))
# any vector perpendicular to the direction vector is mapped to itself
wt <- 3*c(-w[2], w[1])
all.equal(wt, S$transform(wt) - S$transform(c(0,0)))


## ------------------------------------------------
## Method `Scaling$new`
## ------------------------------------------------

Scaling$new(c(1,1), c(1,3), 2)

## ------------------------------------------------
## Method `Scaling$getMatrix`
## ------------------------------------------------

S <- Scaling$new(c(1,1), c(2,3), 2)
P <- c(1,5)
S$transform(P)
S$getMatrix() %*% c(P,1)
```

ScalingXY *R6 class representing an axis-scaling*

Description

An axis-scaling is given by a center, and two scale factors `sx` and `sy`, one for the x-axis and one for the y-axis.

Active bindings

`center` get or set the center

`sx` get or set the scale factor of the x-axis

`sy` get or set the scale factor of the y-ayis

Methods

Public methods:

- `ScalingXY$new()`
- `ScalingXY$print()`
- `ScalingXY$transform()`
- `ScalingXY$getMatrix()`
- `ScalingXY$asAffine()`
- `ScalingXY$clone()`

Method `new()`**:** Create a new `ScalingXY` object.

Usage:

```
ScalingXY$new(center, sx, sy)
```

Arguments:

`center` a point, the center of the scaling

`sx` a number, the scale factor of the x-axis

`sy` a number, the scale factor of the y-axis

Returns: A new `ScalingXY` object.

Examples:

```
ScalingXY$new(c(1,1), 4, 2)
```

Method `print()`**:** Show instance of a `ScalingXY` object.

Usage:

```
ScalingXY$print(...)
```

Arguments:

`...` ignored

Method `transform()`**:** Transform a point or several points by the reference axis-scaling.

Usage:

```
ScalingXY$transform(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Returns: A point or a two-column matrix of points.

Method `getMatrix()`**:** Augmented matrix of the axis-scaling.

Usage:

```
ScalingXY$getMatrix()
```

Returns: A 3x3 matrix.

Examples:

```
S <- ScalingXY$new(c(1,1), 4, 2)
P <- c(1,5)
S$transform(P)
S$getMatrix() %*% c(P,1)
```

Method `asAffine()`**:** Convert the reference axis-scaling to an `Affine` object.

Usage:

```
ScalingXY$asAffine()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ScalingXY$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ScalingXY$new`
## ------------------------------------------------

ScalingXY$new(c(1,1), 4, 2)

## ------------------------------------------------
## Method `ScalingXY$getMatrix`
## ------------------------------------------------

S <- ScalingXY$new(c(1,1), 4, 2)
P <- c(1,5)
S$transform(P)
S$getMatrix() %*% c(P,1)
```

Shear *R6 class representing a shear transformation*

Description

A shear is given by a vertex, two perpendicular vectors, and an angle.

Active bindings

`vertex` get or set the vertex

`vector` get or set the first vector

`ratio` get or set the ratio between the length of `vector` and the length of the second vector, perpendicular to the first one

`angle` get or set the angle

`degrees` get or set the `degrees` field

Methods

Public methods:

- `Shear$new()`
- `Shear$print()`
- `Shear$transform()`
- `Shear$getMatrix()`
- `Shear$asAffine()`
- `Shear$clone()`

Method `new()`**:** Create a new `Shear` object.

Usage:

```
Shear$new(vertex, vector, ratio, angle, degrees = TRUE)
```

Arguments:

`vertex` a point

`vector` a vector

`ratio` a positive number, the ratio between the length of `vector` and the length of the second vector, perpendicular to the first one

`angle` an angle strictly between -90 degrees and 90 degrees

`degrees` logical, whether `angle` is given in degrees

Returns: A new `Shear` object.

Examples:

```
Shear$new(c(1,1), c(1,3), 0.5, 30)
```

Method `print()`**:** Show instance of a `Shear` object.

Usage:

`Shear$print(...)`

Arguments:

`...` ignored

Method `transform()`**:** Transform a point or several points by the reference shear.

Usage:

`Shear$transform(M)`

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `getMatrix()`**:** Augmented matrix of the shear.

Usage:

`Shear$getMatrix()`

Returns: A 3x3 matrix.

Examples:

```
S <- Shear$new(c(1,1), c(1,3), 0.5, 30)
S$getMatrix()
```

Method `asAffine()`**:** Convert the reference shear to an `Affine` object.

Usage:

`Shear$asAffine()`

Examples:

```
Shear$new(c(0,0), c(1,0), 1, atan(30), FALSE)$asAffine()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Shear$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

References

R. Goldman, *An Integrated Introduction to Computer Graphics and Geometric Modeling*. CRC Press, 2009.

Examples

```
P <- c(0,0); w <- c(1,0); ratio <- 1; angle <- 45
shear <- Shear$new(P, w, ratio, angle)
wt <- ratio * c(-w[2], w[1])
Q <- P + w; R <- Q + wt; S <- P + wt
A <- shear$transform(P)
B <- shear$transform(Q)
C <- shear$transform(R)
D <- shear$transform(S)
```

```
plot(0, 0, type = "n", asp = 1, xlim = c(0,1), ylim = c(0,2))
lines(rbind(P,Q,R,S,P), lwd = 2) # unit square
lines(rbind(A,B,C,D,A), lwd = 2, col = "blue") # image by the shear


## ------------------------------------------------
## Method `Shear$new`
## ------------------------------------------------

Shear$new(c(1,1), c(1,3), 0.5, 30)

## ------------------------------------------------
## Method `Shear$getMatrix`
## ------------------------------------------------

S <- Shear$new(c(1,1), c(1,3), 0.5, 30)
S$getMatrix()

## ------------------------------------------------
## Method `Shear$asAffine`
## ------------------------------------------------

Shear$new(c(0,0), c(1,0), 1, atan(30), FALSE)$asAffine()
```

SteinerChain *Steiner chain*

Description

Return a Steiner chain of circles.

Usage

```
SteinerChain(c0, n, phi, shift, ellipse = FALSE)
```

Arguments

c0	exterior circle, a Circle object
n	number of circles, not including the inner circle; at least 3
phi	-1 < phi < 1 controls the radii of the circles
shift	any number; it produces a kind of rotation around the inner circle; values between 0 and n cover all possibilities
ellipse	logical; the centers of the circles of the Steiner chain lie on an ellipse, and this ellipse is returned as an attribute if you set this argument to TRUE

Value

A list of n+1 Circle objects. The inner circle is stored at the last position.

Examples

```
c0 <- Circle$new(c(1,1), 3)
chain <- SteinerChain(c0, 5, 0.3, 0.5, ellipse = TRUE)
plot(0, 0, type = "n", asp = 1, xlim = c(-4,4), ylim = c(-4,4))
invisible(lapply(chain, draw, lwd = 2, border = "blue"))
draw(c0, lwd = 2)
draw(attr(chain, "ellipse"), lwd = 2, border = "red")
```

`Translation` *R6 class representing a translation*

Description

A translation is given by a vector `v`.

Active bindings

`v` get or set the vector of translation

Methods

Public methods:

- `Translation$new()`
- `Translation$print()`
- `Translation$project()`
- `Translation$transform()`
- `Translation$translateLine()`
- `Translation$transformLine()`
- `Translation$translateEllipse()`
- `Translation$transformEllipse()`
- `Translation$getMatrix()`
- `Translation$asAffine()`
- `Translation$clone()`

Method `new()`**:** Create a new `Translation` object.

Usage:

```
Translation$new(v)
```

Arguments:

`v` a numeric vector of length two, the vector of translation

Returns: A new `Translation` object.

Method `print()`**:** Show instance of a translation object.

Usage:

```
Translation$print(...)
```

Arguments:

`...` ignored

Method `project()`**:** Transform a point or several points by the reference translation.

Usage:

```
Translation$project(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `transform()`**:** An alias of `translate`.

Usage:

```
Translation$transform(M)
```

Arguments:

`M` a point or a two-column matrix of points, one point per row

Method `translateLine()`**:** Translate a line.

Usage:

```
Translation$translateLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `transformLine()`**:** An alias of `translateLine`.

Usage:

```
Translation$transformLine(line)
```

Arguments:

`line` a `Line` object

Returns: A `Line` object.

Method `translateEllipse()`**:** Translate a circle or an ellipse.

Usage:

```
Translation$translateEllipse(ell)
```

Arguments:

`ell` an `Ellipse` object or a `Circle` object

Returns: An `Ellipse` object or a `Circle` object.

Method `transformEllipse()`**:** An alias of `translateEllipse`.

Usage:

```
Translation$transformEllipse(ell)
```

Arguments:

`ell` an `Ellipse` object or a `Circle` object

Returns: An `Ellipse` object or a `Circle` object.

Method `getMatrix()`**:** Augmented matrix of the translation.

Usage:

`Translation$getMatrix()`

Returns: A 3x3 matrix.

Method `asAffine()`**:** Convert the reference translation to an `Affine` object.

Usage:

`Translation$asAffine()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Translation$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`Triangle` *R6 class representing a triangle*

Description

A triangle has three vertices. They are named A, B, C.

Active bindings

`A` get or set the vertex `A`

`B` get or set the vertex `B`

`C` get or set the vertex `C`

Methods

Public methods:

- `Triangle$new()`
- `Triangle$print()`
- `Triangle$flatness()`
- `Triangle$a()`
- `Triangle$b()`
- `Triangle$c()`
- `Triangle$edges()`
- `Triangle$orientation()`
- `Triangle$contains()`
- `Triangle$isAcute()`
- `Triangle$angleA()`
- `Triangle$angleB()`

- Triangle$angleC()
- Triangle$angles()
- Triangle$X175()
- Triangle$VeldkampIsoperimetricPoint()
- Triangle$centroid()
- Triangle$orthocenter()
- Triangle$area()
- Triangle$incircle()
- Triangle$inradius()
- Triangle$incenter()
- Triangle$excircles()
- Triangle$excentralTriangle()
- Triangle$BevanPoint()
- Triangle$medialTriangle()
- Triangle$orthicTriangle()
- Triangle$incentralTriangle()
- Triangle$NagelTriangle()
- Triangle$NagelPoint()
- Triangle$GergonneTriangle()
- Triangle$GergonnePoint()
- Triangle$tangentialTriangle()
- Triangle$symmedialTriangle()
- Triangle$symmedianPoint()
- Triangle$circumcircle()
- Triangle$circumcenter()
- Triangle$circumradius()
- Triangle$BrocardCircle()
- Triangle$BrocardPoints()
- Triangle$LemoineCircleI()
- Triangle$LemoineCircleII()
- Triangle$LemoineTriangle()
- Triangle$LemoineCircleIII()
- Triangle$ParryCircle()
- Triangle$pedalTriangle()
- Triangle$CevianTriangle()
- Triangle$MalfattiCircles()
- Triangle$AjimaMalfatti1()
- Triangle$AjimaMalfatti2()
- Triangle$equalDetourPoint()
- Triangle$trilinearToPoint()
- Triangle$pointToTrilinear()
- Triangle$rotate()

- Triangle$translate()
- Triangle$SteinerEllipse()
- Triangle$SteinerInellipse()
- Triangle$randomPoints()
- Triangle$clone()

Method `new()`**:** Create a new `Triangle` object.

Usage:

```
Triangle$new(A, B, C)
```

Arguments:

`A, B, C` vertices

Returns: A new `Triangle` object.

Examples:

```
t <- Triangle$new(c(0,0), c(1,0), c(1,1))
t
t$C
t$C <- c(2,2)
t
```

Method `print()`**:** Show instance of a triangle object

Usage:

```
Triangle$print(...)
```

Arguments:

`...` ignored

Examples:

```
Triangle$new(c(0,0), c(1,0), c(1,1))
```

Method `flatness()`**:** Flatness of the triangle.

Usage:

```
Triangle$flatness()
```

Returns: A number between 0 and 1. A triangle is flat when its flatness is 1.

Method `a()`**:** Length of the side `BC`.

Usage:

```
Triangle$a()
```

Method `b()`**:** Length of the side `AC`.

Usage:

```
Triangle$b()
```

Method `c()`**:** Length of the side `AB`.

Usage:

```
Triangle$c()
```

Method `edges()`**:** The lengths of the sides of the triangle.

Usage:

`Triangle$edges()`

Returns: A named numeric vector.

Method `orientation()`**:** Determine the orientation of the triangle.

Usage:

`Triangle$orientation()`

Returns: An integer: 1 for counterclockwise, -1 for clockwise, 0 for collinear.

Method `contains()`**:** Determine whether a point lies inside the reference triangle.

Usage:

`Triangle$contains(M)`

Arguments:

`M` a point

Method `isAcute()`**:** Determines whether the reference triangle is acute.

Usage:

`Triangle$isAcute()`

Returns: 'TRUE' if the triangle is acute (or right), 'FALSE' otherwise.

Method `angleA()`**:** Angle at the vertex A.

Usage:

`Triangle$angleA()`

Returns: The angle at the vertex A in radians.

Method `angleB()`**:** Angle at the vertex B.

Usage:

`Triangle$angleB()`

Returns: The angle at the vertex B in radians.

Method `angleC()`**:** Angle at the vertex C.

Usage:

`Triangle$angleC()`

Returns: The angle at the vertex C in radians.

Method `angles()`**:** The three angles of the triangle.

Usage:

`Triangle$angles()`

Returns: A named vector containing the values of the angles in radians.

Method `X175()`**:** The X(175) triangle center.

Usage:

`Triangle$X175()`

Method `VeldkampIsoperimetricPoint()`: Isoperimetric point in the sense of Veldkamp.

Usage:

`Triangle$VeldkampIsoperimetricPoint()`

Returns: The isoperimetric point in the sense of Veldkamp, if it exists. Otherwise, returns 'NULL'.

Method `centroid()`: Centroid.

Usage:

`Triangle$centroid()`

Method `orthocenter()`: Orthocenter.

Usage:

`Triangle$orthocenter()`

Method `area()`: Area of the triangle.

Usage:

`Triangle$area()`

Method `incircle()`: Incircle of the triangle.

Usage:

`Triangle$incircle()`

Returns: A `Circle` object.

Method `inradius()`: Inradius of the reference triangle.

Usage:

`Triangle$inradius()`

Method `incenter()`: Incenter of the reference triangle.

Usage:

`Triangle$incenter()`

Method `excircles()`: Excircles of the triangle.

Usage:

`Triangle$excircles()`

Returns: A list with the three excircles, `Circle` objects.

Method `excentralTriangle()`: Excentral triangle of the reference triangle.

Usage:

`Triangle$excentralTriangle()`

Returns: A `Triangle` object.

Method `BevanPoint()`: Bevan point. This is the circumcenter of the excentral triangle.

Usage:

```
Triangle$BevanPoint()
```

Method `medialTriangle()`**:** Medial triangle. Its vertices are the mid-points of the sides of the reference triangle.

Usage:

```
Triangle$medialTriangle()
```

Method `orthicTriangle()`**:** Orthic triangle. Its vertices are the feet of the altitudes of the reference triangle.

Usage:

```
Triangle$orthicTriangle()
```

Method `incentralTriangle()`**:** Incentral triangle.

Usage:

```
Triangle$incentralTriangle()
```

Details: It is the triangle whose vertices are the intersections of the reference triangle's angle bisectors with the respective opposite sides.

Returns: A `Triangle` object.

Method `NagelTriangle()`**:** Nagel triangle (or extouch triangle) of the reference triangle.

Usage:

```
Triangle$NagelTriangle(NagelPoint = FALSE)
```

Arguments:

`NagelPoint` logical, whether to return the Nagel point as attribute

Returns: A `Triangle` object.

Examples:

```
t <- Triangle$new(c(0,-2), c(0.5,1), c(3,0.6))
lineAB <- Line$new(t$A, t$B)
lineAC <- Line$new(t$A, t$C)
lineBC <- Line$new(t$B, t$C)
NagelTriangle <- t$NagelTriangle(NagelPoint = TRUE)
NagelPoint <- attr(NagelTriangle, "Nagel point")
excircles <- t$excircles()
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type="n", asp = 1, xlim = c(-1,5), ylim = c(-3,3),
     xlab = NA, ylab = NA, axes = FALSE)
draw(t, lwd = 2)
draw(lineAB); draw(lineAC); draw(lineBC)
draw(excircles$A, border = "orange")
draw(excircles$B, border = "orange")
draw(excircles$C, border = "orange")
draw(NagelTriangle, lwd = 2, col = "red")
draw(Line$new(t$A, NagelTriangle$A, FALSE, FALSE), col = "blue")
draw(Line$new(t$B, NagelTriangle$B, FALSE, FALSE), col = "blue")
draw(Line$new(t$C, NagelTriangle$C, FALSE, FALSE), col = "blue")
points(rbind(NagelPoint), pch = 19)
par(opar)
```

Method `NagelPoint()`**:** Nagel point of the triangle.

Usage:

```
Triangle$NagelPoint()
```

Method `GergonneTriangle()`**:** Gergonne triangle of the reference triangle.

Usage:

```
Triangle$GergonneTriangle(GergonnePoint = FALSE)
```

Arguments:

`GergonnePoint` logical, whether to return the Gergonne point as an attribute

Details: The Gergonne triangle is also known as the *intouch triangle* or the *contact triangle*. This is the triangle made of the three tangency points of the incircle.

Returns: A `Triangle` object.

Method `GergonnePoint()`**:** Gergonne point of the reference triangle.

Usage:

```
Triangle$GergonnePoint()
```

Method `tangentialTriangle()`**:** Tangential triangle of the reference triangle. This is the triangle formed by the lines tangent to the circumcircle of the reference triangle at its vertices. It does not exist for a right triangle.

Usage:

```
Triangle$tangentialTriangle()
```

Returns: A `Triangle` object.

Method `symmedialTriangle()`**:** Symmedial triangle of the reference triangle.

Usage:

```
Triangle$symmedialTriangle()
```

Returns: A `Triangle` object.

Examples:

```
t <- Triangle$new(c(0,-2), c(0.5,1), c(3,0.6))
symt <- t$symmedialTriangle()
symmedianA <- Line$new(t$A, symt$A, FALSE, FALSE)
symmedianB <- Line$new(t$B, symt$B, FALSE, FALSE)
symmedianC <- Line$new(t$C, symt$C, FALSE, FALSE)
K <- t$symmedianPoint()
opar <- par(mar = c(0,0,0,0))
plot(NULL, asp = 1, xlim = c(-1,5), ylim = c(-3,3),
     xlab = NA, ylab = NA, axes = FALSE)
draw(t, lwd = 2)
draw(symmedianA, lwd = 2, col = "blue")
draw(symmedianB, lwd = 2, col = "blue")
draw(symmedianC, lwd = 2, col = "blue")
points(rbind(K), pch = 19, col = "red")
par(opar)
```

Method `symmedianPoint()`**:** Symmedian point of the reference triangle.

Usage:

`Triangle$symmedianPoint()`

Returns: A point.

Method `circumcircle()`**:** Circumcircle of the reference triangle.

Usage:

`Triangle$circumcircle()`

Returns: A `Circle` object.

Method `circumcenter()`**:** Circumcenter of the reference triangle.

Usage:

`Triangle$circumcenter()`

Method `circumradius()`**:** Circumradius of the reference triangle.

Usage:

`Triangle$circumradius()`

Method `BrocardCircle()`**:** The Brocard circle of the reference triangle (also known as the seven-point circle).

Usage:

`Triangle$BrocardCircle()`

Returns: A `Circle` object.

Method `BrocardPoints()`**:** Brocard points of the reference triangle.

Usage:

`Triangle$BrocardPoints()`

Returns: A list of two points, the first Brocard point and the second Brocard point.

Method `LemoineCircleI()`**:** The first Lemoine circle of the reference triangle.

Usage:

`Triangle$LemoineCircleI()`

Returns: A `Circle` object.

Method `LemoineCircleII()`**:** The second Lemoine circle of the reference triangle (also known as the cosine circle)

Usage:

`Triangle$LemoineCircleII()`

Returns: A `Circle` object.

Method `LemoineTriangle()`**:** The Lemoine triangle of the reference triangle.

Usage:

`Triangle$LemoineTriangle()`

Returns: A `Triangle` object.

Method `LemoineCircleIII()`**:** The third Lemoine circle of the reference triangle.

Usage:

```
Triangle$LemoineCircleIII()
```

Returns: A `Circle` object.

Method `ParryCircle()`**:** Parry circle of the reference triangle.

Usage:

```
Triangle$ParryCircle()
```

Returns: A `Circle` object.

Method `pedalTriangle()`**:** Pedal triangle of a point with respect to the reference triangle. The pedal triangle of a point `P` is the triangle whose vertices are the feet of the perpendiculars from `P` to the sides of the reference triangle.

Usage:

```
Triangle$pedalTriangle(P)
```

Arguments:

`P` a point

Returns: A `Triangle` object.

Method `CevianTriangle()`**:** Cevian triangle of a point with respect to the reference triangle.

Usage:

```
Triangle$CevianTriangle(P)
```

Arguments:

`P` a point

Returns: A `Triangle` object.

Method `MalfattiCircles()`**:** Malfatti circles of the triangle.

Usage:

```
Triangle$MalfattiCircles(tangencyPoints = FALSE)
```

Arguments:

`tangencyPoints` logical, whether to retourn the tangency points of the Malfatti circles as an attribute.

Returns: A list with the three Malfatti circles, `Circle` objects.

Examples:

```
t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
Mcircles <- t$MalfattiCircles(TRUE)
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(0,2.5),
     xlab = NA, ylab = NA)
grid()
draw(t, col = "blue", lwd = 2)
invisible(lapply(Mcircles, draw, col = "green", border = "red"))
invisible(lapply(attr(Mcircles, "tangencyPoints"), function(P){
  points(P[1], P[2], pch = 19)
}))
```

Method `AjimaMalfatti1()`**:** First Ajima-Malfatti point of the triangle.

Usage:

```
Triangle$AjimaMalfatti1()
```

Method `AjimaMalfatti2()`**:** Second Ajima-Malfatti point of the triangle.

Usage:

```
Triangle$AjimaMalfatti2()
```

Method `equalDetourPoint()`**:** Equal detour point of the triangle.

Usage:

```
Triangle$equalDetourPoint(detour = FALSE)
```

Arguments:

`detour` logical, whether to return the detour as an attribute.

Details: Also known as the X(176) triangle center.

Method `trilinearToPoint()`**:** Point given by trilinear coordinates.

Usage:

```
Triangle$trilinearToPoint(x, y, z)
```

Arguments:

`x, y, z` trilinear coordinates

Returns: The point with trilinear coordinates `x:y:z` with respect to the reference triangle.

Examples:

```
t <- Triangle$new(c(0,0), c(2,1), c(5,7))
incircle <- t$incircle()
t$trilinearToPoint(1, 1, 1)
incircle$center
```

Method `pointToTrilinear()`**:** Give the trilinear coordinates of a point with respect to the reference triangle.

Usage:

```
Triangle$pointToTrilinear(P)
```

Arguments:

`P` a point

Returns: The trilinear coordinates, a numeric vector of length 3.

Method `rotate()`**:** Rotate the triangle.

Usage:

```
Triangle$rotate(alpha, O, degrees = TRUE)
```

Arguments:

`alpha` angle of rotation

`O` center of rotation

`degrees` logical, whether `alpha` is given in degrees

Returns: A `Triangle` object.

Method `translate()`**:** Translate the triangle.

Usage:

```
Triangle$translate(v)
```

Arguments:

`v` the vector of translation

Returns: A `Triangle` object.

Method `SteinerEllipse()`**:** The Steiner ellipse (or circumellipse) of the reference triangle. This is the ellipse passing through the three vertices of the triangle and centered at the centroid of the triangle.

Usage:

```
Triangle$SteinerEllipse()
```

Returns: An `Ellipse` object.

Examples:

```
t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
ell <- t$SteinerEllipse()
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(-0.7,2.4),
     xlab = NA, ylab = NA)
draw(t, col = "blue", lwd = 2)
draw(ell, border = "red", lwd =2)
```

Method `SteinerInellipse()`**:** The Steiner inellipse (or midpoint ellipse) of the reference triangle. This is the ellipse tangent to the sides of the triangle at their midpoints, and centered at the centroid of the triangle.

Usage:

```
Triangle$SteinerInellipse()
```

Returns: An `Ellipse` object.

Examples:

```
t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
ell <- t$SteinerInellipse()
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(-0.1,2.4),
     xlab = NA, ylab = NA)
draw(t, col = "blue", lwd = 2)
draw(ell, border = "red", lwd =2)
```

Method `randomPoints()`**:** Random points on or in the reference triangle.

Usage:

```
Triangle$randomPoints(n, where = "in")
```

Arguments:

`n` an integer, the desired number of points

`where` `"in"` to generate inside the triangle, `"on"` to generate on the triangle

Returns: The generated points in a two columns matrix with n rows.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Triangle$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

`TriangleThreeLines` to define a triangle by three lines.

Examples

```
# incircle and excircles
A <- c(0,0); B <- c(1,2); C <- c(3.5,1)
t <- Triangle$new(A, B, C)
incircle <- t$incircle()
excircles <- t$excircles()
JA <- excircles$A$center
JB <- excircles$B$center
JC <- excircles$C$center
JAJBJC <- Triangle$new(JA, JB, JC)
A_JA <- Line$new(A, JA, FALSE, FALSE)
B_JB <- Line$new(B, JB, FALSE, FALSE)
C_JC <- Line$new(C, JC, FALSE, FALSE)
opar <- par(mar = c(0,0,0,0))
plot(NULL, asp = 1, xlim = c(0,6), ylim = c(-4,4),
     xlab = NA, ylab = NA, axes = FALSE)
draw(t, lwd = 2)
draw(incircle, border = "orange")
draw(excircles$A); draw(excircles$B); draw(excircles$C)
draw(JAJBJC, col = "blue")
draw(A_JA, col = "green")
draw(B_JB, col = "green")
draw(C_JC, col = "green")
par(opar)


## ------------------------------------------------
## Method `Triangle$new`
## ------------------------------------------------

t <- Triangle$new(c(0,0), c(1,0), c(1,1))
t
t$C
t$C <- c(2,2)
t

## ------------------------------------------------
## Method `Triangle$print`
```

```
## ------------------------------------------------

Triangle$new(c(0,0), c(1,0), c(1,1))

## ------------------------------------------------
## Method `Triangle$NagelTriangle`
## ------------------------------------------------

t <- Triangle$new(c(0,-2), c(0.5,1), c(3,0.6))
lineAB <- Line$new(t$A, t$B)
lineAC <- Line$new(t$A, t$C)
lineBC <- Line$new(t$B, t$C)
NagelTriangle <- t$NagelTriangle(NagelPoint = TRUE)
NagelPoint <- attr(NagelTriangle, "Nagel point")
excircles <- t$excircles()
opar <- par(mar = c(0,0,0,0))
plot(0, 0, type="n", asp = 1, xlim = c(-1,5), ylim = c(-3,3),
     xlab = NA, ylab = NA, axes = FALSE)
draw(t, lwd = 2)
draw(lineAB); draw(lineAC); draw(lineBC)
draw(excircles$A, border = "orange")
draw(excircles$B, border = "orange")
draw(excircles$C, border = "orange")
draw(NagelTriangle, lwd = 2, col = "red")
draw(Line$new(t$A, NagelTriangle$A, FALSE, FALSE), col = "blue")
draw(Line$new(t$B, NagelTriangle$B, FALSE, FALSE), col = "blue")
draw(Line$new(t$C, NagelTriangle$C, FALSE, FALSE), col = "blue")
points(rbind(NagelPoint), pch = 19)
par(opar)

## ------------------------------------------------
## Method `Triangle$symmedialTriangle`
## ------------------------------------------------

t <- Triangle$new(c(0,-2), c(0.5,1), c(3,0.6))
symt <- t$symmedialTriangle()
symmedianA <- Line$new(t$A, symt$A, FALSE, FALSE)
symmedianB <- Line$new(t$B, symt$B, FALSE, FALSE)
symmedianC <- Line$new(t$C, symt$C, FALSE, FALSE)
K <- t$symmedianPoint()
opar <- par(mar = c(0,0,0,0))
plot(NULL, asp = 1, xlim = c(-1,5), ylim = c(-3,3),
     xlab = NA, ylab = NA, axes = FALSE)
draw(t, lwd = 2)
draw(symmedianA, lwd = 2, col = "blue")
draw(symmedianB, lwd = 2, col = "blue")
draw(symmedianC, lwd = 2, col = "blue")
points(rbind(K), pch = 19, col = "red")
par(opar)

## ------------------------------------------------
## Method `Triangle$MalfattiCircles`
## ------------------------------------------------
```

```
t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
Mcircles <- t$MalfattiCircles(TRUE)
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(0,2.5),
     xlab = NA, ylab = NA)
grid()
draw(t, col = "blue", lwd = 2)
invisible(lapply(Mcircles, draw, col = "green", border = "red"))
invisible(lapply(attr(Mcircles, "tangencyPoints"), function(P){
  points(P[1], P[2], pch = 19)
}))

## ------------------------------------------------
## Method `Triangle$trilinearToPoint`
## ------------------------------------------------

t <- Triangle$new(c(0,0), c(2,1), c(5,7))
incircle <- t$incircle()
t$trilinearToPoint(1, 1, 1)
incircle$center

## ------------------------------------------------
## Method `Triangle$SteinerEllipse`
## ------------------------------------------------

t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
ell <- t$SteinerEllipse()
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(-0.7,2.4),
     xlab = NA, ylab = NA)
draw(t, col = "blue", lwd = 2)
draw(ell, border = "red", lwd =2)

## ------------------------------------------------
## Method `Triangle$SteinerInellipse`
## ------------------------------------------------

t <- Triangle$new(c(0,0), c(2,0.5), c(1.5,2))
ell <- t$SteinerInellipse()
plot(NULL, asp = 1, xlim = c(0,2.5), ylim = c(-0.1,2.4),
     xlab = NA, ylab = NA)
draw(t, col = "blue", lwd = 2)
draw(ell, border = "red", lwd =2)
```

`TriangleThreeLines` *Triangle defined by three lines*

Description

Return the triangle formed by three lines.

Usage

```
TriangleThreeLines(line1, line2, line3)
```

Arguments

`line1, line2, line3`
`Line` objects

Value

A `Triangle` object.

---

unitCircle *Unit circle*

Description

Circle centered at the origin with radius 1.

Usage

```
unitCircle
```

Format

An object of class `Circle` (inherits from `R6`) of length 24.

Index