

# Package ‘LearnGeom’

May 7, 2026

**Title** Learning Plane Geometry

**Version** 1.5

**Author** Alvaro Briz-Redon, Angel Serrano-Aroca

**Maintainer** Alvaro Briz-Redon <albrizre@gmail.com>

**Description** Contains some functions to learn and teach basic plane Geometry at undergraduate level with the aim of being helpful to young students with little programming skills.

**Depends** R (>= 3.0.2)

**License** GPL-2

**LazyData** true

**RoxygenNote** 7.1.1

**Suggests** knitr, rmarkdown, testthat

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-07-14 16:00:03 UTC

## Contents

AddPointPoly . . . . .	2
Angle . . . . .	3
CenterPolygon . . . . .	4
Circumcenter . . . . .	5
CoordinatePlane . . . . .	6
CreateArcAngles . . . . .	6
CreateArcPointsDist . . . . .	7
CreateLineAngle . . . . .	9
CreateLinePoints . . . . .	9
CreatePolygon . . . . .	10
CreateRegularPolygon . . . . .	11
CreateSegmentAngle . . . . .	11
CreateSegmentPoints . . . . .	12
DistanceLines . . . . .	13
DistancePointLine . . . . .	14

DistancePoints . . . . .	14
Draw . . . . .	15
Duopoly . . . . .	16
FractalSegment . . . . .	17
Homothety . . . . .	18
Incenter . . . . .	19
IntersectLineCircle . . . . .	20
IntersectLines . . . . .	21
Koch . . . . .	22
LinesAngles . . . . .	23
MidPoint . . . . .	23
PolygonAngles . . . . .	24
ProjectPoint . . . . .	25
ReflectedPoint . . . . .	26
ReflectedPolygon . . . . .	27
RemovePointPoly . . . . .	28
Rotate . . . . .	28
SelectPoints . . . . .	29
ShearedPolygon . . . . .	30
Sierpinski . . . . .	31
SimilarPolygon . . . . .	32
Soddy . . . . .	32
Star . . . . .	34
Tessellation . . . . .	35
Translate . . . . .	36

<b>Index</b>	<b>38</b>
--------------	-----------

---

AddPointPoly	<i>Adds a point to a previously defined polygon</i>
--------------	---

---

## Description

AddPointPoly creates a matrix to represent the polygon that connects several points

## Usage

AddPointPoly(Poly, point, position)

## Arguments

Poly	Polygon object, previously created with function CreatePolygon or CreateRegularPolygon
point	Vector containing the xy-coordinates of the point to be added to the polygon
position	Integer indicating the position of the point in the original polygon, after which the new point is being added (considering that every polygon is an ordered list of points). It is convenient to visualize the polygon with label = T in order to avoid mistakes

**Value**

Returns a matrix which contains the points of the polygon. Each row represents one of the points

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
n <- 5
C <- c(0,0)
l <- 2
Penta <- CreateRegularPolygon(n, C, l)
Penta <- AddPointPoly(Penta, CenterPolygon(Penta), 1)
Draw(Penta, "blue", label = TRUE)
```

---

Angle

*Computes the angle between three points*

---

**Description**

Angle computes the angle between three points

**Usage**

```
Angle(A, B, C, label = FALSE)
```

**Arguments**

A	Vector containing the xy-coordinates of point A
B	Vector containing the xy-coordinates of point B. This point acts as the vertex of angle ABC
C	Vector containing the xy-coordinates of point C
label	Boolean. When label = TRUE, the plot displays the angle in the point that acts as the vertex. If missing, it works as with label = FALSE, so the angle is not displayed

**Value**

Angle between the three points in degrees

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
A <- c(-1,0)
B <- c(0,0)
C <- c(0,1)
Draw(CreatePolygon(A, B, C), "transparent")
angle <- Angle(A, B, C, label = TRUE)
angle <- Angle(A, C, B, label = TRUE)
angle <- Angle(B, A, C, label = TRUE)
```

---

CenterPolygon	<i>Computes the center of a given polygon. The center is obtained by averaging the x and y coordinates of the polygon</i>
---------------	---

---

**Description**

CenterPolygon computes the center of a polygon

**Usage**

```
CenterPolygon(Poly)
```

**Arguments**

Poly	Polygon object, previously created with either of the functions CreatePolygon or CreateRegularPolygon
------	---

**Value**

Vector which contains the xy-coordinates of the center of the polygon

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
C <- CenterPolygon(Poly)
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
Draw(Poly, "blue")
Draw(C, "red")
```

---

Circumcenter	<i>Computes the circumcenter of a given triangle, that is, the intersection of its three medians</i>
--------------	--

---

### Description

Circumcenter computes the center of a triangle

### Usage

```
Circumcenter(Tri, lines = F)
```

### Arguments

Tri	Triangle object, previously created with function CreatePolygon
lines	Boolean. When lines = TRUE, the plot displays the lines that represent the medians of each of the sides of the triangle. If missing, it works as with lines = FALSE, so the lines are not displayed

### Value

Vector which contains the xy-coordinates of the circumcenter of the triangle

### References

<http://mathworld.wolfram.com/Circumcenter.html>

### Examples

```
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Tri <- CreatePolygon(P1, P2, P3)
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
Draw(Tri, "transparent")
I <- Circumcenter(Tri, lines = TRUE)
Draw(I, "red")
```

---

CoordinatePlane	<i>Plots an empty coordinate (cartesian) plane with customizable limits for the X and Y axis</i>
-----------------	--

---

**Description**

CoordinatePlane plots an empty coordinate (cartesian) plane with customizable limits for the X and Y axis.

**Usage**

```
CoordinatePlane(x_min, x_max, y_min, y_max)
```

**Arguments**

x_min	Lowest value for the X axis
x_max	Highest value for the X axis
y_min	Lowest value for the Y axis
y_max	Highest value for the Y axis

**Value**

None. It produces a plot of a coordinate plane with axes and grid

**Examples**

```
x_min <- -5  
x_max <- 5  
y_min <- -5  
y_max <- 5  
CoordinatePlane(x_min, x_max, y_min, y_max)
```

---

CreateArcAngles	<i>Creates an arc of a circumference</i>
-----------------	--

---

**Description**

CreateArcAngles creates an arc of a circumference

**Usage**

```
CreateArcAngles(C, r, angle1, angle2, direction = "anticlock")
```

**Arguments**

C	Vector containing the xy-coordinates of the center of the circumference
r	Radius for the circumference (or arc)
angle1	- Angle in degrees (0-360) at which the arc starts
angle2	- Angle in degrees (0-360) at which the arc finishes
direction	- String indicating the direction which is considered to create the arc, from the smaller to the higher angle. It has two possible values: "clock" (clockwise direction) and "anticlock" (anti-clockwise direction)

**Value**

Returns a vector which contains the center, radius, angles (0-360) and direction (1 - "clock", 2 - "anticlock") that define the created arc

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
C <- c(0,0)
r <- 3
angle1 <- 90
angle2 <- 180
direction <- "anticlock"
Arc1 <- CreateArcAngles(C, r, angle1, angle2, direction)
Draw(Arc1, "black")
direction <- "clock"
Arc2 <- CreateArcAngles(C, r, angle1, angle2, direction)
Draw(Arc2, "red")
```

---

CreateArcPointsDist    *Creates an arc of a circumference to connect two points*

---

**Description**

CreateArcPointsDist creates an arc of a circumference to connect two points

**Usage**

```
CreateArcPointsDist(P1, P2, r, choice, direction)
```

**Arguments**

P1	Vector containing the xy-coordinates of point 1
P2	Vector containing the xy-coordinates of point 2
r	Radius for the circumference which is used to generate the arc. This parameter is necessary because there are infinite possible arcs that connect two points. In the case the radius is smaller than half the distance between P1 and P2, there is no possible arc, so the function tells the user
choice	- Integer indicating which of the two possible centers is chosen to create the arcs. A value of 1 means the center of the circle that contains the arc is chosen in the direction of $M + v$ , being $M$ the middle point between P1 and P2 and $v$ the orthogonal vector of $P2 - P1$ normalized to the appropriate length for creating the desired arc. A value of 2 means the center of the resulting circle is chosen in the direction of $M - v$ . Remark: There are as well two options for vector $v$ . If $P1 = (a,b)$ and $P2 = (c,d)$ , $v$ is written in the internal function as $(b-d,c-a)$
direction	- String indicating the direction which is considered to create the arc, from the smaller to the higher angle. It has two possible values: "clock" (clockwise direction) and "anticlock" (anti-clockwise direction)

**Value**

Returns a vector which contains the center, radius and angles (0-360) that define the created arc

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(-3,2)
P2 <- c(0,0)
r <- sqrt(18)/2
choice=1
direction="anticlock"
Arc <- CreateArcPointsDist(P1, P2, r, choice, direction)
Draw(Arc, "red")
choice=2
direction="anticlock"
Arc <- CreateArcPointsDist(P1, P2, r, choice, direction)
Draw(Arc, "blue")
choice=1
direction="clock"
Arc <- CreateArcPointsDist(P1, P2, r, choice, direction)
Draw(Arc, "pink")
choice=2
direction="clock"
Arc <- CreateArcPointsDist(P1, P2, r, choice, direction)
Draw(Arc, "green")
```

---

CreateLineAngle	<i>Creates a vector to represent a line that passes through a point and forms certain angle with X axis</i>
-----------------	---

---

**Description**

CreateLineAngle creates a vector to represent a line that passes through a point and forms certain angle with X axis

**Usage**

```
CreateLineAngle(P, angle)
```

**Arguments**

P	Vector containing the xy-coordinates of a point
angle	Angle in degrees (0-360) for the line

**Value**

Returns a vector which contains the slope and intercept of the defined line. If the angle is defined as 90, the slope is set to Inf and the intercept is replaced by the x-value for the line (which is a vertical line in this situation)

**Examples**

```
P <- c(0,0)
angle <- 45
Line <- CreateLineAngle(P, angle)
```

---

CreateLinePoints	<i>Creates a vector that represents the line that connects two points</i>
------------------	---

---

**Description**

CreateLinePoints creates a vector that represents the line that connects two points

**Usage**

```
CreateLinePoints(P1, P2)
```

**Arguments**

P1	Vector containing the xy-coordinates of point 1
P2	Vector containing the xy-coordinates of point 2

**Value**

Returns a vector which contains the slope and intercept of the defined line. If the points have the same x-coordinate, the slope is set to Inf and the intercept is replaced by the x-value for the line (which is a vertical line in this situation)

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
Line <- CreateLinePoints(P1, P2)
```

---

CreatePolygon	<i>Creates a matrix to represent the polygon that connects several points</i>
---------------	---

---

**Description**

CreatePolygon creates a matrix to represent the polygon that connects several points

**Usage**

```
CreatePolygon(...)
```

**Arguments**

... An undetermined number of points introduced by the user in the form of vectors

**Value**

Returns a matrix which contains the points of the polygon. Each row represents one of the points

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
```

---

CreateRegularPolygon *Creates a matrix to represent a regular polygon*

---

**Description**

CreateRegularPolygon creates a matrix to represent the polygon that connects several points

**Usage**

```
CreateRegularPolygon(n, C, l)
```

**Arguments**

n	Number of sides for the polygon
C	Vector containing the xy-coordinates for the center of the regular polygon
l	Length of the sides for the polygon

**Value**

Returns a matrix which contains the points of a regular polygon given its number of points and the length of its sides. Each row represents one of the points

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
n <- 5
C <- c(0,0)
l <- 1
Penta <- CreateRegularPolygon(n, C, l)
Draw(Penta, "blue", label = TRUE)
```

---

CreateSegmentAngle *Creates a matrix that represents the segment that starts from a point with certain length and angle*

---

**Description**

DrawSegment plots the segment that connects two points in a previously generated coordinate plane

**Usage**

```
CreateSegmentAngle(P, angle, l)
```

**Arguments**

P	Vector containing the xy-coordinates of the point
angle	Angle in degrees (0-360) for the segment
l	Positive number that indicates the length for the segment

**Value**

Returns a matrix which contains the points that determine the extremes of the segment

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P <- c(0,0)
angle <- 30
l <- 1
Segment <- CreateSegmentAngle(P, angle, l)
Draw(Segment, "black")
```

---

CreateSegmentPoints *Creates a matrix that represents the segment that connects two points*

---

**Description**

DrawSegment plots the segment that connects two points in a previously generated coordinate plane

**Usage**

```
CreateSegmentPoints(P1, P2)
```

**Arguments**

P1	Vector containing the xy-coordinates of point 1
P2	Vector containing the xy-coordinates of point 2

**Value**

Returns a matrix which contains the points that determine the extremes of the segment

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
Segment <- CreateSegmentPoints(P1, P2)
Draw(Segment, "black")
```

---

DistanceLines

*Computes the distance between two lines*

---

**Description**

DistanceLines computes the distance between two lines

**Usage**

```
DistanceLines(Line1, Line2)
```

**Arguments**

Line1	Line object previously created with CreateLinePoints or CreateLineAngle
Line2	Line object previously created with CreateLinePoints or CreateLineAngle

**Value**

Returns the distance between two points

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
Line1 <- CreateLinePoints(P1, P2)
P3 <- c(1,-1)
P4 <- c(2,0)
Line2 <- CreateLinePoints(P3, P4)
d <- DistanceLines(Line1, Line2)
```

DistancePointLine      *Computes the distance between a point and a line*

---

**Description**

DistancePointLine computes the distance between a point and a line

**Usage**

```
DistancePointLine(P, Line)
```

**Arguments**

P                      Vector containing the xy-coordinates of a point  
Line                    Vector object previously created with CreateLinePoints or CreateLineAngle

**Value**

Returns the distance between a point and a line. This distance corresponds to the distance between the point and its orthogonal projection into the line

**Examples**

```
P <- c(2,1)
P1 <- c(0,0)
P2 <- c(1,1)
Line <- CreateLinePoints(P1, P2)
d <- DistancePointLine(P, Line)
```

---

DistancePoints      *Computes the distance between two points*

---

**Description**

DistancePoints computes the distance between two points

**Usage**

```
DistancePoints(P1, P2)
```

**Arguments**

P1                      Vector containing the xy-coordinates of point 1  
P2                      Vector containing the xy-coordinates of point 2

**Value**

Returns the euclidean distance between two points

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
d <- DistancePoints(P1, P2)
```

---

 Draw

---

*Plots a geometric object*


---

**Description**

Draw plots geometric objects

**Usage**

```
Draw(object, colors = c("black", "black"), label = FALSE)
```

**Arguments**

object	geometric object of any of these five types: point, segment, arc, line or polygon. A point is simply a vector of length 2, which contains the xy-coordinates for the point. For the other four types, there can be created with any of the following functions: <ul style="list-style-type: none"> <li>- CreateArcAngles</li> <li>- CreateArcPointsDist</li> <li>- CreateLineAngle</li> <li>- CreateLinePoints</li> <li>- CreatePolygon</li> <li>- CreateRegularPolygon</li> <li>- CreateSegmentAngle</li> <li>- CreateSegmentPoints</li> </ul>
colors	Vector containing information about the color for the object to be plotted. In the case of polygons, the vector should have length 2 to define the background color and the border color (in this order). Moreover, it can be used "transparent" in the case no background color is needed for the polygon. For the other four types of objects, color should be a vector of length 1 (or a simple string) to indicate the color for the object. If this parameter is not specified the default color is black (for polygons, it is black for the background and the border)
label	Boolean, only used for polygons. When label = TRUE and the object is a polygon, the plot displays the numbers that correspond to the order of the points of the polygon. If missing, it works as with label = FALSE, so the numbers are not displayed

**Value**

None. It produces the plot of a geometric object (point, segment, arc, line or polygon) in the current coordinate plane

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, c("blue"))
```

---

Duopoly

*Plots a fractal curve from the trochoids family. Any curve from this family can be defined by some parametrical equations, but they can also be produced (approximated) through a simple iterative process based on segment drawing for certain angles and lengths*

---

**Description**

Duopoly plots a closed curve from the trochoids family

**Usage**

```
Duopoly(P, l1, angle1, l2, angle2, time = 0, color = "transparent")
```

**Arguments**

P	Vector containing the xy-coordinates of the starting point for the curve
l1	Number that indicates the length side of the segment drawn the first in each of the steps of the process
angle1	Angle (0-360) that indicates the direction of the segment which is drawn the first in each of the steps of the process
l2	Number that indicates the length side of the segment drawn the second in each of the steps of the process
angle2	Angle (0-360) that indicates the direction of the segment which is drawn the second in each of the steps of the process
time	Number of seconds to wait for the program before drawing each of the segments that make the trochoid curve. If no time is specified, default value is 0 (no waiting time). If the chosen time is very small ( <code>time &lt; 0.05</code> ) it is possible that the program shows the plot directly. In this case, it should be increased the time parameter.

**color** Color to indicate the points that are obtained during the process to approximate the trochoid. If missing, the points are not indicated and only the segments are drawn in the plot

### Value

None. It produces the plot of a curve from the trochoids family

### References

- Abelson, H., & DiSessa, A. A. (1986). Turtle geometry: The computer as a medium for exploring mathematics. MIT press
- Armon, U. (1996). Representing trochoid curves by DUOPOLY procedure. International Journal of Mathematical Education in Science and Technology, 27(2), 177-187

### Examples

```
x_min <- -100
x_max <- 100
y_min <- -50
y_max <- 150
CoordinatePlane(x_min, x_max, y_min, y_max)
P <- c(0,0)
l1 <- 2
angle1 <- 3
l2 <- 2
angle2 <- 10
Duopoly(P, l1, angle1, l2, angle2)
```

---

FractalSegment

*Plots a fractal curve starting from a segment*

---

### Description

FractalSegment plots the first iterations of a fractal curve, starting from a segment in the plane

### Usage

```
FractalSegment(P1, P2, angle, cut1, cut2, f, it)
```

### Arguments

- P1** Vector containing the xy-coordinates of point 1. This point is the left extreme of the segment that corresponds to the first iteration ( $it = 1$ )
- P2** Vector containing the xy-coordinates of point 2. This point is the right extreme of the segment that corresponds to the first iteration ( $it = 1$ )
- angle** Angle (0-360) that determines the angle with which the new segments are drawn at the cut points

cut1	Number bigger than 0 and smaller than 1 that indicates the proportional part of the segment at which the first cut occurs. This parameter determines the position of the first cut point
cut2	Number bigger than 0 and smaller than 1 that indicates the proportional part of the segment at which the second cut occurs. This parameter determines the position of the second cut point
f	Positive number that produces an enlargement or a reduction for the new drawn segment in each iteration
it	Number of iterations to be performed for the construction of the fractal curve. It is not recommended to choose a number higher than 7 in order to avoid an excess of computation

**Value**

None. It produces the plot of the first  $n$  iterations of a fractal curve in the current coordinate plane. The choice of parameters  $\text{cut1} = 1/3$ ,  $\text{cut2} = 2/3$ ,  $\text{angle} = 60$  and  $f = 1$  produces the Koch curve

**References**

<http://mathworld.wolfram.com/Fractal.html>

**Examples**

```
x_min <- -6
x_max <- 6
y_min <- -4
y_max <- 8
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(-5,0)
P2 <- c(5,0)
angle <- 90
cut1 <- 1/3
cut2 <- 2/3
f <- 1
it <- 4
FractalSegment(P1, P2, angle, cut1, cut2, f, it)
```

---

Homothety

*Creates an homothety from a given polygon*

---

**Description**

Homothety creates an homothety from a given polygon

**Usage**

Homothety(Poly, C, k, lines = F)

**Arguments**

Poly	Polygon object, previously created with function CreatePolygon
C	Vector containing the xy-coordinates of the center of the homothety
k	Number which represents the expansion or contraction factor for the homothety
lines	Boolean. When lines = TRUE, the plot displays the lines that connect the center of the homothety with the points of the polygons (the original and the transformed one). If missing, it works as with lines = FALSE, so the lines are not displayed

**Value**

Returns the coordinates of a polygon that has been transformed according to the homothety with center at C and factor k

**References**

<https://www.encyclopediaofmath.org/index.php/Homothety>

**Examples**

```
x_min <- -2
x_max <- 6
y_min <- -3
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, "blue")
C <- c(-1,-2)
k1 <- 0.5
Poly_homothety1 <- Homothety(Poly, C, k1, lines = TRUE)
Draw(Poly_homothety1, "orange")
k2 <- 2
Poly_homothety2 <- Homothety(Poly, C, k2, lines = TRUE)
Draw(Poly_homothety2, "orange")
```

---

Incenter

*Computes the incenter of a given triangle*

---

**Description**

Incenter computes the center of a triangle

**Usage**

```
Incenter(Tri, lines = F)
```

**Arguments**

Tri	Triangle object, previously created with function CreatePolygon
lines	Boolean. When lines = TRUE, the plot displays the lines that bisect each of the angles of the triangle. If missing, it works as with lines = FALSE, so the lines are not displayed

**Value**

Vector which contains the xy-coordinates of the incenter of the triangle

**References**

<http://mathworld.wolfram.com/Incenter.html>

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Tri <- CreatePolygon(P1, P2, P3)
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
Draw(Tri, "transparent")
I <- Incenter(Tri, lines = TRUE)
Draw(I, "red")
```

---

IntersectLineCircle *Finds the intersection between a line and a circumference*

---

**Description**

IntersectLineCircle finds the intesection between a line and a circumference

**Usage**

```
IntersectLineCircle(Line, C, r)
```

**Arguments**

Line	Line object previously created with CreateLinePoints or CreateLineAngle
C	Vector containing the xy-coordinates of the center of the circumference
r	Radius for the circumference

**Value**

Returns a vector containing the xy-coordinates of the intersection points. In case of no intersection, the function tells the user

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
Line <- CreateLinePoints(P1, P2)
C <- c(0,0)
r <- 2
intersection <- IntersectLineCircle(Line, C, r)
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
Draw(Line, "black")
Draw(CreateArcAngles(C, r, 0, 360), "black")
Draw(intersection[1,], "red")
Draw(intersection[2,], "red")
```

---

IntersectLines	<i>Finds the intersection of two lines</i>
----------------	--

---

**Description**

IntersectLines finds the intesection of two lines

**Usage**

```
IntersectLines(Line1, Line2)
```

**Arguments**

Line1	Line object previously created with CreateLinePoints or CreateLineAngle
Line2	Line object previously created with CreateLinePoints or CreateLineAngle

**Value**

Returns a vector containing the xy-coordinates of the intersection point. In case of no intersection, the function tells the user

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
Line1 <- CreateLinePoints(P1, P2)
P3 <- c(1,-1)
P4 <- c(2,0)
Line2 <- CreateLinePoints(P3, P4)
intersection <- IntersectLines(Line1, Line2)
```

---

Koch

*Plots the Koch curve*

---

**Description**

Koch plots the first iterations of Koch curve, a well-known fractal

**Usage**

```
Koch(P1, P2, it)
```

**Arguments**

P1	Vector containing the xy-coordinates of point 1. This point is the left extreme of the segment that corresponds to the first iteration ( $it = 1$ )
P2	Vector containing the xy-coordinates of point 2. This point is the right extreme of the segment that corresponds to the first iteration ( $it = 1$ )
it	Number of iterations to be performed for the construction of Koch curve. It is not recommended to choose a number higher than 7 in order to avoid an excess of computation

**Value**

None. It produces the plot of the first  $n$  iterations of Koch curve in the current coordinate plane

**References**

<http://mathworld.wolfram.com/KochSnowflake.html>

**Examples**

```
x_min <- -6
x_max <- 6
y_min <- -4
y_max <- 8
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(-5,0)
P2 <- c(5,0)
it <- 4
Koch(P1, P2, it)
```

---

LinesAngles	<i>Computes the angle that form two lines</i>
-------------	---

---

**Description**

LinesAngles computes the angle that form two lines

**Usage**

```
LinesAngles(Line1, Line2)
```

**Arguments**

Line1	Line object previously created with CreateLinePoints or CreateLineAngle
Line2	Line object previously created with CreateLinePoints or CreateLineAngle

**Value**

Returns the angle that form the two lines

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
Line1 <- CreateLinePoints(P1, P2)
P3 <- c(1,-1)
P4 <- c(2,3)
Line2 <- CreateLinePoints(P3, P4)
angle <- LinesAngles(Line1, Line2)
```

---

MidPoint	<i>Computes the middle point of the segment that connects two points</i>
----------	--

---

**Description**

MidPoint computes the middle point of the segment that connects two points

**Usage**

```
MidPoint(P1, P2)
```

**Arguments**

P1	Vector containing the xy-coordinates of point 1
P2	Vector containing the xy-coordinates of point 2

**Value**

Returns a vector containing the xy-coordinates of the middle point of the segment that connects P1 and P2

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
mid <- MidPoint(P1, P2)
```

---

PolygonAngles

*Computes each of the existing angles in a given polygon*

---

**Description**

PolygonAngles computes each of the existing angles in a given polygon

**Usage**

```
PolygonAngles(Poly)
```

**Arguments**

Poly                    Polygon object, previously created with function CreatePolygon

**Value**

Returns a vector containing the angles for each of the points of a polygon. The resulting vector follows the order of the points in the defined polygon

**Examples**

```
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
angles <- PolygonAngles(Poly)
```

---

ProjectPoint	<i>Computes the orthogonal projection of a point onto a line</i>
--------------	--

---

**Description**

ProjectPoint computes the orthogonal projection of a point onto a line

**Usage**

```
ProjectPoint(P, Line)
```

**Arguments**

P	Vector containing the xy-coordinates of a point
Line	Line object previously created with CreateLinePoints or CreateLineAngle, to be used as the axis of symmetry

**Value**

Returns a vector which contains the xy-coordinates of the projection point

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
xx <- c(0,1,2)
yy <- c(0,1,0)
P1 <- c(0,0)
P2 <- c(1,1)
Line <- CreateLinePoints(P1, P2)
Draw(Line, "black")
P <- c(-2,2)
Draw(P, "black")
projection <- ProjectPoint(P, Line)
Draw(projection, "red")
```

---

ReflectedPoint	<i>Computes the reflected point about a line of a given point</i>
----------------	---

---

### Description

ReflectedPoint computes the reflected point about a line of a given point

### Usage

```
ReflectedPoint(P, Line)
```

### Arguments

P	Vector containing the xy-coordinates of a point
Line	Line object previously created with CreateLinePoints or CreateLineAngle, to be used as the axis of symmetry

### Value

Returns a vector which contains the xy-coordinates of the reflected point

### Examples

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
xx <- c(0,1,2)
yy <- c(0,1,0)
P1 <- c(0,0)
P2 <- c(1,1)
Line <- CreateLinePoints(P1, P2)
Draw(Line, "black")
P <- c(-2,2)
Draw(P, "black")
reflected <- ReflectedPoint(P, Line)
Draw(reflected, "red")
```

---

ReflectedPolygon	<i>Creates the reflection about a line of a given polygon</i>
------------------	---

---

### Description

ReflectedPolygon creates the reflection about a line of a given polygon

### Usage

```
ReflectedPolygon(Poly, Line)
```

### Arguments

Poly	Polygon object, previously created with function CreatePolygon or CreateRegularPolygon
Line	Line object previously created with CreateLinePoints or CreateLineAngle, to be used as the axis of symmetry

### Value

Returns the reflection of a polygon about a line

### Examples

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, "blue")
P1 <- c(-3,2)
P2 <- c(1,-4)
Line <- CreateLinePoints(P1, P2)
Draw(Line, "black")
Poly_reflected <- ReflectedPolygon(Poly, Line)
Draw(Poly_reflected, "orange")
```

---

RemovePointPoly	<i>Removes a point from a previously defined polygon</i>
-----------------	--

---

### Description

RemovePointPoly creates a matrix to represent the polygon that connects several points

### Usage

```
RemovePointPoly(Poly, position)
```

### Arguments

Poly	Polygon object, previously created with function CreatePolygon or CreateRegularPolygon
position	Integer indicating the position of the point in the original polygon that is being removed. It is convenient to visualize the polygon with label = T in order to avoid mistakes

### Value

Returns a matrix which contains the points of the polygon. Each row represents one of the points

### Examples

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
n <- 5
C <- c(0,0)
l <- 2
Penta <- CreateRegularPolygon(n, C, l)
Penta <- RemovePointPoly(Penta, 4)
Draw(Penta, "blue", label = TRUE)
```

---

Rotate	<i>Rotates a geometric object</i>
--------	-----------------------------------

---

### Description

Rotate rotates a geometric object of any of the following types: line, polygon or segment

### Usage

```
Rotate(object, fixed, angle)
```

**Arguments**

object	geometric object of type line, polygon or segment, previously created with any of the functions in the package
fixed	Vector containing the xy-coordinates of the only point of the plane which remains fixed during rotation
angle	Angle of rotation in degrees (0-360), considering the clockwise direction

**Value**

Returns a geometric object which is the rotation of the original one, following the clockwise direction

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, "blue")
fixed <- c(-1,-1)
angle <- 30
Poly_rotated <- Rotate(Poly, fixed, angle)
Draw(Poly_rotated, "orange")
fixed <- c(2,0)
Poly_rotated <- Rotate(Poly, fixed, angle)
Draw(Poly_rotated, "transparent")
```

---

SelectPoints

*Selection of points from the coordinate plane*

---

**Description**

SelectPoints allows the selection of points from the coordinate plane

**Usage**

```
SelectPoints(n)
```

**Arguments**

n	Number of points to select from the current coordinate plane
---	--

**Value**

Returns a vector or matrix which contains the xy-coordinates of the selected points. Each row represents one of the points. If  $n = 1$  the output is a numeric vector, if  $n = 2$  then it is a Segment, and for  $n > 2$  the object is a polygon.

**Examples**

```
n <- 3
points <- SelectPoints(n)
```

---

ShearedPolygon	<i>Creates a sheared polygon from a given one</i>
----------------	---

---

**Description**

ShearedPolygon creates a sheared polygon from a given one

**Usage**

```
ShearedPolygon(Poly, k, direction)
```

**Arguments**

Poly	Polygon object, previously created with function CreatePolygon or CreateRegularPolygon
k	Number that represents the shear factor which is applied to the original polygon
direction	String with value "horizontal" or "vertical" which indicates the direction in which shearing is applied. Horizontal means the shearing is parallel to the X axis, while vertical means parallel to the Y axis

**Value**

Returns a sheared polygon, in any of the two axis, to the original one

**Examples**

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
Square <- CreateRegularPolygon(4, c(-2, 0), 1)
Draw(Square, "blue")
k <- 1
Square_shearX <- Translate(ShearedPolygon(Square, k, "horizontal"), c(3,0))
Draw(Square_shearX, "orange")
Square_shearY <- Translate(ShearedPolygon(Square, k, "vertical"), c(3,0))
Draw(Square_shearY, "orange")
```

---

Sierpinski

*Plots the Sierpinski triangle*

---

### Description

Sierpinski plots the first iterations of Sierpinski triangle, a well-known fractal

### Usage

```
Sierpinski(Tri, it)
```

### Arguments

Tri	Regular triangle, previously created with function CreateRegularPolygon
it	Number of iterations to be performed for the construction of Sierpinski triangle. It is not recommended to choose a number higher than 10 in order to avoid an excess of computation

### Value

None. It produces the plot of the first n iterations of Sierpinski triangle in the current coordinate plane

### References

<http://mathworld.wolfram.com/SierpinskiSieve.html>

### Examples

```
x_min <- -6
x_max <- 6
y_min <- -6
y_max <- 6
CoordinatePlane(x_min, x_max, y_min, y_max)
n <- 3
C <- c(0,0)
l <- 5
Tri <- CreateRegularPolygon(n, C, l)
it <- 6
Sierpinski(Tri, it)
```

---

SimilarPolygon                      *Creates a similar polygon to a given one*

---

### Description

SimilarPolygon creates a sheared polygon from a given one

### Usage

```
SimilarPolygon(Poly, k)
```

### Arguments

Poly	Polygon object, previously created with function CreatePolygon or CreateRegularPolygon
k	Positive number that represents the expansion ( $k > 1$ ) or contraction ( $k < 1$ ) factor which is applied to the original polygon

### Value

Returns a similar polygon, expanded or contracted, to the original polygon

### Examples

```
x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, "blue")
k <- 2
Poly_similar <- SimilarPolygon(Poly, k)
Draw(Translate(Poly_similar, c(-1,2)), "orange")
```

---

Soddy                                      *Finds the inner and outer Soddy circles of three given mutually tangent circles*

---

### Description

Soddy finds inner and outer Soddy circles of three given mutually tangent circles

**Usage**

Soddy(A, r1, B, r2, C, r3)

**Arguments**

A	Vector containing the xy-coordinates of the center of circumference 1
r1	Radius for circumference 1
B	Vector containing the xy-coordinates of the center of circumference 2
r2	Radius for circumference 2
C	Vector containing the xy-coordinates of the center of circumference 3
r3	Radius for circumference 3

**Value**

A list which contains the Soddy center and the radiuses of Soddy inner and outer circle of three mutually tangent circles

**References**

<http://mathworld.wolfram.com/SoddyCircles.html>

**Examples**

```
x_min <- -3
x_max <- 3
y_min <- -2.5
y_max <- 3.5
CoordinatePlane(x_min, x_max, y_min, y_max)
A <- c(-1,0)
B <- c(1,0)
C <- c(0,sqrt(3))
r1 <- 1
r2 <- 1
r3 <- 1
Draw(CreateArcAngles(A, r1, 0, 360), "black")
Draw(CreateArcAngles(B, r2, 0, 360), "black")
Draw(CreateArcAngles(C, r3, 0, 360), "black")
result <- Soddy(A, r1, B, r2, C, r3)
soddy_point <- result[[1]]
inner_radius <- result[[2]]
outer_radius <- result[[3]]
Draw(soddy_point, "red")
Draw(CreateArcAngles(soddy_point, inner_radius, 0, 360), "red")
Draw(CreateArcAngles(soddy_point, outer_radius, 0, 360), "red")
```

---

Star	<i>Creates a closed curve with the shape of a star. Each of the stars produced by this function is built through a simple iterative process based on segment drawing for certain angles and lengths. It can also produce regular polygons for some combinations of the parameters</i>
------	---

---

### Description

Star creates a star with multiple building possibilities

### Usage

```
Star(P, angle, l, time = 0, color = "transparent")
```

### Arguments

P	Vector containing the xy-coordinates of the starting point for the star
angle	Angle (0-360) that is related to the direction of the two segments which are drawn in each of the steps of the process. This parameter really represents the angle (in clockwise and anti-clockwise direction) for the two first drawn segments, but it is modified according to rotations of 144 degrees in all the following steps, including the last one, which closes the curve.
l	Number that indicates the length side of the segments that are drawn. This parameter will determine the size of the star
time	Number of seconds to wait for the program before drawing each of the segments that make star. If no time is specified, default value is 0 (no waiting time). If the chosen time is very small (time < 0.05) it is possible that the program shows the plot directly. In this case, it should be increased the time parameter.
color	Color to indicate the points that are obtained during the process to draw the star. If missing, the points are not indicated and only the segments are drawn in the plot

### Value

None. It produces the plot of a closed curve with the shape of a star, if the parameters are chosen properly

### References

Abelson, H., & DiSessa, A. A. (1986). Turtle geometry: The computer as a medium for exploring mathematics. MIT press

**Examples**

```

x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P <- c(0,0)
angle <- 0
l <- 1
Star(P, angle, l)

```

---

Tessellation

*Creates a tessellation from a starting set of geometric objects*


---

**Description**

Tessellation creates a geometric pattern by the repetitive translation of an initial geometric object

**Usage**

```
Tessellation(objects_list, colors, direction, separation, it)
```

**Arguments**

<code>objects_list</code>	A list composed by several geometric objects (mainly polygons created with <code>CreatePolygon</code> or <code>CreateRegularPolygon</code> )
<code>colors</code>	Vector containing the colors for each of the objects of the initial geometric object
<code>direction</code>	Vector containing the xy-coordinates of the direction in which tessellation is being generated
<code>separation</code>	Number indicating the distance that separates any of the geometric objects in the repetitive pattern. This distance must be understood in the sense of a translation of the initial object. Indeed, this distance is only preserved in the direction of the chosen vector <code>direction</code> when generating the pattern. Moreover, the choice of <code>separation = 0</code> implies no pattern is generated
<code>it</code>	Number of iterations to be performed for the construction of the tessellation

**Value**

None. It produces the plot of a repetitive pattern, usually known as a tessellation

**References**

<http://mathworld.wolfram.com/Tessellation.html>

**Examples**

```

x_min <- -6
x_max <- 6
y_min <- -2
y_max <- 10
CoordinatePlane(x_min, x_max, y_min, y_max)
Hexa <- CreateRegularPolygon(6, c(-3,0), 1)
Draw(Hexa, "purple")
Tri <- CreatePolygon(c(-3,-1), c(Hexa[4,1],-2), c(Hexa[1,1],-2))
Draw(Tri, "pink")
objects_list <- list(Tri, Hexa)
cols <- c("pink", "purple")
direction <- c(1,0)
separation <- 1.732051
it <- 3
Tessellation(objects_list, cols, direction, separation, it)
direction <- c(0,1)
separation <- 3
it <- 4
Tessellation(objects_list, cols, direction, separation, it)

```

---

Translate

*Translates a geometric object*


---

**Description**

Translate translates a geometric object of any of the following types: line, polygon or segment

**Usage**

```
Translate(object, v)
```

**Arguments**

object	geometric object, previously created with function CreatePolygon
v	Vector containing the xy-coordinates of the translation vector

**Value**

Returns a polygon whose coordinates are translated according to vector v

**Examples**

```

x_min <- -5
x_max <- 5
y_min <- -5
y_max <- 5
CoordinatePlane(x_min, x_max, y_min, y_max)
P1 <- c(0,0)

```

```
P2 <- c(1,1)
P3 <- c(2,0)
Poly <- CreatePolygon(P1, P2, P3)
Draw(Poly, "blue")
v <- c(1,2)
Poly_translated <- Translate(Poly, v)
Draw(Poly_translated, "orange")
```

# Index

AddPointPoly, 2  
Angle, 3  
  
CenterPolygon, 4  
Circumcenter, 5  
CoordinatePlane, 6  
CreateArcAngles, 6  
CreateArcPointsDist, 7  
CreatelineAngle, 9  
CreatelinePoints, 9  
CreatePolygon, 10  
CreateRegularPolygon, 11  
CreateSegmentAngle, 11  
CreateSegmentPoints, 12  
  
DistanceLines, 13  
DistancePointLine, 14  
DistancePoints, 14  
Draw, 15  
Duopoly, 16  
  
FractalSegment, 17  
  
Homothety, 18  
  
Incenter, 19  
IntersectLineCircle, 20  
IntersectLines, 21  
  
Koch, 22  
  
LinesAngles, 23  
  
MidPoint, 23  
  
PolygonAngles, 24  
ProjectPoint, 25  
  
ReflectedPoint, 26  
ReflectedPolygon, 27  
RemovePointPoly, 28  
  
Rotate, 28  
  
SelectPoints, 29  
ShearedPolygon, 30  
Sierpinski, 31  
SimilarPolygon, 32  
Soddy, 32  
Star, 34  
  
Tessellation, 35  
Translate, 36