Research Ideas and Outcomes 4: e25485
doi: 10.3897/rio.4.e25485

R Package

Novel pedagogical tool for simultaneous learning of plane geometry and R programming

Álvaro Briz-Redón‡, Ángel Serrano-Aroca§

‡ Departament d'Estadística i Investigació Operativa, Facultat de Matemàtiques, Universitat de València, Valencia, Spain
§ Facultad de Veterinaria y Ciencias Experimentales, Universidad Católica de Valencia San Vicente Mártir, Valencia, Spain

Corresponding author: Ángel Serrano-Aroca (angel.serrano@ucv.es)

Reviewable v1

Received: 03 Apr 2018 | Published: 05 Apr 2018

Citation: Briz-Redón Á, Serrano-Aroca Á (2018) Novel pedagogical tool for simultaneous learning of plane geometry and R programming. Research Ideas and Outcomes 4: e25485. https://doi.org/10.3897/rio.4.e25485

Abstract

Programming a computer is an activity that can be very beneficial to undergraduate students in terms of improving their mental capabilities, collaborative attitudes and levels of engagement in learning. Despite the initial difficulties that typically arise when learning to program, there are several well-known strategies to overcome them, providing a very high benefit-cost ratio to most of the students. Moreover, the use of a programming language usually raises the interest of students to learn any specific concept, which has caused that many teachers around the world employ a programming language as a learning environment to treat almost every possible topic. Particularly, mathematics can be taught and learnt while using a suitable programming language. The R programming language is endowed with a wide range of capabilities that allow its use to learn different kind of concepts while programming. Therefore, complex subjects such as mathematics could be learnt with the help of this powerful programming language. In addition, since the R language provides numerous graphical functions, it could be very useful to acquire simultaneously basic plane geometry and programming knowledge at the undergraduate level. This paper describes the LearnGeom R package, a novel pedagogical tool, which contains multiple functions to learn geometry in R at different levels of difficulty, from the most basic geometric objects to high-complexity geometric constructions, while developing numerous programming skills.

Keywords

Mathematics Education; Geometry Education; Computational Thinking; Programming Language; R package

Introduction

In the field of informatics, programming is the activity that basically consists of translating from human language to the language understood by a computer McCracken (1957).

In addition to its inherent importance in the computational sciences, learning to program provides powerful strategies for thinking, designing and solving problems Booth (1958), Howe et al. (1981), Soloway (1993). In fact, these authors describe a two-phase process, consisting of finding the problem solution and then rewriting this solution in an alternative and precise language that can be understood by the computer.

The nature of programming provides certain advantages to the user that very few mental activities can offer. For example, it allows the programmer to explore thinking processes, improve logical reasoning and increase capacity to correct mistakes Papert (1980). Moreover, the practice of programming is highly correlated with the development of a good linguistic capacity and, consequently, with a better academic performance Lehrer and DeBernard (1987), Palumbo (1990).

There are multiple studies that have confirmed the benefits of programming on young students. For example, already in the early 1970s, it was concluded in a survey conducted by Feurzeig et al. 2011 that the use of LOGO programming language could enhance the reading ability of some students, as well as increasing the interest in learning and the level of self-confidence of many of them.

At the same time, there is little disagreement about the declaration of programming as a complex activity Govender (2009). More precisely, Du Boulay (1986) identified five kinds of difficulties: lack of orientation, ignorance about the machine being employed (usually a computer), inadaptation to the language, problems of comprehension and absence of method. Although these five frequent difficulties are interconnected, students specially tend to have problems with the programming language itself. Language's syntax, that is, the collection of words and symbols that allow a human to communicate with a machine, can be excessively rigid to a novel user of a programming language. In connection with the syntax, the meaning of each of its elements forces the students to carry out processes of abstraction and reasoning that they are not always able to assume, deriving sometimes in misconceptions and wrong analogies Spohrer et al. (1985). Furthermore, it is really difficult to teach students to systematize their programming practices in order to be more efficient and make less mistakes. In this regard, the common stages recommended when programming could be summarized as follows: problem approach, language development, testing and correction/improvement.

In view of all these difficulties, what strategies should be followed for the correct teaching of programming? While being aware that each student learns in a different way, there are some accepted strategies in order to reduce, at least to a certain extent, the difficulties that arise when programming. In the first place, the choice of the programming language is key. The basic operating rules and the rigid syntax are common to all programming languages, but not all of them are equally accessible, especially from the point of view of a totally inexperienced user. The wide variety of languages available today, the majority of which are free software, render this task much easier.

Once the language has been conveniently chosen, different strategies can be carried out. Some methodological examples of successful programming teaching approaches include: simplifying and dividing problems into simpler sub-problems, defining and explaining a model based on the machine-language environment in which the student is working Statz and Miller (1978), slightly modifying sets of instructions to solve similar problems Linn and Dalbey (1985), or constantly addressing applied problems.

Learning through the use of a programming language can result more interesting and exciting for a student, and explains somehow why it has been used to teach and study mathematics during the last decades, with overall successful results, as described by Wang et al. (2015). In addition, the use of programming as a learning vehicle favours collaboration between students and the development of competencies related to autonomous learning Falloon (2016).

Currently, there are already some languages and programs focused on mathematics learning and teaching at undergraduate level, being Scratch Resnick et al. (2009) and GeoGebra Hohenwarter (2002) two of the most important by far.

Scratch possesses most of the common features that are associated to a programming language. However, its style is predominantly graphical, which differs to the vast majority of programming languages. This graphical orientation clearly reduces the initial difficulty of learning to program, but also avoids the development of some other programming skills usually acquired in the classical educational way.

The ScratchMaths project Benton et al. (2016) was developed in four schools located in London with students aged 9 to 11 years to learn mathematics through Scratch programming on four basic objectives: to explore, explain, predict and share. This project is still in progress and has recently shown that the correct design and sequencing of activities around a mathematical concept, together with the continuous help of specialized teachers, allows students to learn with interest and in a meaningful way Benton et al. (2018).

In addition, researchers such as Foerster (2016) and Akpinar and Aslan (2015) proved that the application of the Scratch language to teaching mathematics can be very successful. Thus, Foerster (2016) applied this Scratch programming methodology in a class of about 11-year-old students while developing mathematical concepts related to polygon congruence and tessellation construction. The results of this study showed that the learning process run very smoothly and pleasantly for the students, and after about two

years, the students achieved an excellent knowledge of geometry in comparison with the other students who did not follow this pedagogical methodology. These results clearly demonstrate the possible long-term positive effects that this learning methodology can produce. In the same line of research, Akpinar and Aslan (2015) applied Scratch language to create a learning environment that allowed the study of probability. Although Scratch does not have specific elements for dealing with probabilistic concepts, the implementation of experimental and simulated situations was really helpful to improve students' proficiency in this branch of mathematics. The tests carried out before and after the sessions showed that the students significantly increased their knowledge of probability. At the same time, the sessions allowed students to exploit their creative facets by designing multiple games associated with random experiments.

On the other hand, GeoGebra is dynamic mathematics software, which brings together geometry, algebra, spreadsheets, graphing, statistics and calculus in one easy-to-use package. Several works confirm the advantages of using GeoGebra at undergraduate level. Zengin et al. (2012) designed an assisted instruction of trigonometry with GeoGebra that resulted more effective than the classical approach, based on constructionist theory. In the same way, Priatna (2017) showed that students being taught with the aid of GeoGebra significantly outperformed students subject to classical education in terms of mathematical representation ability. Other researchers such as Akkaya et al. (2011) proposed the learning of symmetry with GeoGebra, facilitating students to visualize, generalize and share with their classmates, producing an active and collaborative learning environment. Very recently, Masri et al. (2017) tested GeoGebra with some degree of success with a group of Malaysian students, whose mathematical level was later compared to students of the same year that had followed traditional teaching techniques. Despite there were not found significant differences between the two groups of students regarding the mathematical level acquired, students that were taught with GeoGebra clearly appreciated the use of this program during the development of the lessons. Finally, a very interesting finding was achieved by Saha et al. (2010). The use of GeoGebra to teach plane geometry did not appear as significantly better for students already possessing a high visual-spatial ability, when compared to students being taught without the help of the program. However, these differences were clearly revealed when considering students with a low visual-spatial ability, confirming the positive effects of using this software.

Another programming language, which is currently very popular is R Team (2018). This language, even though it has always been conceived as an environment for performing statistical analysis, is also endowed with a wide range of capabilities that allow its use to learn different kind of concepts while programming. Therefore, complex subjects such as mathematics could be learnt with the help of this powerful programming language.

For example, Pruim et al. (2017) have recently developed the Mosaic project, which integrates a set of functions to facilitate the introduction of basic statistics and data science concepts such as visualization, modelling and simulation. Other authors such as Mascaró et al. (2014) and Mascaró et al. (2016) have shown how beneficial can be for a university student to be taught statistics with the aid of the R language, in terms of understanding and engagement to the subject matter.

In addition, since the R language provides numerous graphical functions, it can be very useful to simultaneously acquire basic plane geometry and R programming knowledge at the undergraduate level.

Therefore, in this work we present a novel pedagogical tool to simultaneously learn plane geometry and programming based on R, which we have denominated LearnGeom. This tool is available for secondary school teachers, which may be interested in teaching plane geometry and R programming skills at the same time, and may desire to complement the use of other languages and software with this R package. It can be downloaded once the R programming language is installed in your computer (https://www.r-project.org) and after typing *install.packages(LearnGeom)* in the program console to install the developed LearnGeom R package. The LearnGeom R-package can also be found as a github repository in Briz-Redón and Serrano-Aroca (2018).

Basic functions of the package

The LearnGeom R package provides basic functions to treat plane geometry. Therefore, the user is expected to work on a coordinate plane in order to manipulate different geometric objects and constructions. Thus, the function *CoordinatePlane* allows the user to plot an empty coordinate plane with customizable limits for the X and Y axis. For example, *CoordinatePlane(-5, 5, -5, 5)* can be typed to set a coordinate plane in the range [-5,5] x [-5,5], which is the one used for most of the examples included in the paper. Once a coordinate plane is started, different geometric objects can be created and plotted on it. The basic geometric objects that can be utilised in this version of the package are five: points, segments, arcs, lines and polygons.

All of them can be plotted in the coordinate plane with function *Draw*, and the appropriate use of the methods that R offers can lead to different mathematical concepts and many kind of geometric problems. A point can be created by simply defining a two-dimensional vector in R, e.g., *P = c(0,0)* for the usual origin of coordinates. This definition is also used for the creation of a geometric vector to determine a direction in the plane. The coincidence of both definitions also exists if classical mathematical notations are followed. Therefore, it should not be a source of confusion for the user.

The first method to define a segment in the plane, which is the most basic one, consists of the choice of two points of the plane, P and Q, and applying the shortest path (in euclidean distance) to connect them. There is another common method to define a segment in the plane: from a starting point, choosing an angle and a length for the segment. Both possibilities can be achieved with the *CreateSegmentPoints* and *CreateSegmentAngle* functions of the package. A line, as a segment, can be defined from two points, or from a point and an angle. Moreover, there is a standard combination of parameters to characterize every line in the plane: the slope and the intercept. For this reason, *CreateLinePoints* and *CreateLineAngle* functions return a two-dimensional vector that contains the slope and intercept of the line, regardless of the way it is defined. The use of the pair slope-intercept has a problem with vertical lines, which are parallel to Y axis. In the

case that the user defines a line of this kind, with any of the two available functions, the returning object will be a string two-dimensional vector. It will include the word "*Inf*" for the first position (infinite slope) and the constant X-value for the line in the second (as a character).

Table 1.

The basic functions available in the LearnGeom R package to create geometric objects and the parameters that must be used to precisely define these objects. The third column refers to the geometric object that is created by each of the functions, which is also the class that the functions assign to their outputs.

Function	Parameters	Geometric object
CreateArcAngles	*C, r, angle1, angle2, direction*	*Arc*
CreateArcPointsDist	*P1, P2, r, choice, direction*	
CreateLineAngle	*P, angle*	*Line*
CreateLinePoints	*P1, P2*	
CreatePolygon	*List of points*	*Polygon*
CreateRegularPolygon	*n, C, l*	
CreateSegmentAngle	*P, angle, l*	*Segment*
CreateSegmentPoints	*P1, P2*	

An arc is simply a part of a circumference, or even the circumference itself. The *CreateArcAngle* function allows the user to make an arc from a circumference with four parameters to choose: the center of the circumference, the radius of the circumference and the two angles, from 0 to 360 degrees that determine the part of the circumference to be plotted. Another possibility to create an arc in the plane consists of connecting two points. The *CreateArcPointsDist* function allows the user to connect any two given points in the plane by an arc. Of course, there are many (infinite) arcs that pass through every two points in a plane. The parameter *radius* fixes a radius for the arc to be built. If the selected radius is smaller than half the distance between the points, the problem has no solution and no arc is produced (a message is shown on the screen to inform the user).

Polygons are closed figures made of a finite number of points (there must be 3 points at least). The segments that join the points of a polygon are called the sides of the polygon, and each of the sides must intersect only at the two points that connects (auto-intersections are not allowed in a polygon). It is usual to represent a polygon by an ordered list of points, which indicates the way the points are connected. For example, if a polygon is represented by a list of points [P1, P2,P3], it means that the three segments of this polygon (a triangle) join P1 with P2, P2 with P3 and P3 with P1. The *CreatePolygon* function admits any finite number of points to produce a polygon, which corresponds to the definition of a polygon as a list of points. It is important to introduce the points in a certain order to get the desired output because the same combination of points can lead to different figures. In

order to make a polygon without self-intersections, the points must be passed to the function following a clockwise (or counterclockwise) direction. Moreover, this function includes a procedure to detect collinearity between the points, which makes the function to show the message "Some of the inserted points are collinear. This could lead to a defective polygon" when this occurs. If the user is interested in building a regular polygon given its center, number of sides and side length, the *CreateRegularPolygon* function can do it.

Table 2.

Short description of all the parameters available in the basic functions of the LearnGeom R package.

Function	Parameters description
CreateArcAngles	*C* - Center for the circumference which generates the arc
	r - Radius for the circumference which generates the arc
	angle1 - Angle (0-360) from which the arc starts
	angle2 - Angle (0-360) at which the arc arrives
	direction - Clockwise or anti-clockwise direction to properly define the arc
CreateArcPointsDist	*P1* - Point 1 to be joined by an arc to point 2
	P2 - Point 2 to be joined by an arc to point 1
	r - Radius for the arc being built
	choice - Integer which allows the user to choose every arc from two possibilities
	direction - Clockwise or anti-clockwise direction to properly define the arc
CreateLineAngle	*P* - Point through which the line passes
	angle - Angle (0-360) that the line must form with X-axis
CreateLinePoints	*P1* - Point 1 to be joined by a line to point 2
	P2 - Point 2 to be joined by a line to point 1
CreatePolygon	... - Ordered list of points to build the polygon
CreateRegularPolygon	*n* - Number of sides for the polygon
	C - Center for regular polygon
	l - Side length for the polygon
CreateSegmentAngle	*P* - Starting point for the segment being built
	angle - Angle (0-360) that the segment must form with X-axis
	l - Length for the segment
CreateSegmentPoints	*P1* - Point 1 to be joined by a segment to point 2
	P2 - Point 2 to be joined by a segment to point 1

It is quite simple to make these objects with basic graphical R functions. However, the goal was to create a homogeneous group of functions with a minimal number of parameters. Although the already existing functions in the R language are simple for a programmer, they may contain too many parameters for a novice user. Moreover, it is essential to define the functions with a little number of parameters in order to highlight the different existing methods to define the same geometric object.

Finally, the *Draw* function is programmed to be able to accept all kind of geometric objects as a parameter. This fact seems to be essential to understand geometric ideas because of providing the user a quick and intuitive tool to easily visualize while programming. The only visible difference among the different geometric objects and *Draw* function occurs with polygons. This function adds, only for polygons, the option of using two colours and the possibility of inserting labels at each of the points of the object.

Table 3.

Information contained in each of the possible objects that are produced by the basic functions. As it is shown in the table, the outputs of these functions are basically a vector or a matrix which contain the points that make the geometric object or the parameters that define the object unambiguously.

Geometric object	Data type	Object fields
Arc	*Vector*	*X* – X-coordinate for the center of the arc
		Y – Y-coordinate for the center of the arc
		r – radius of the arc
		angle1 – Angle (0º-360º) from which the arc starts
		angle2 – Angle (0º-360º) at which the arc arrives
		dir – 1 for anti-clockwise direction, 2 for clockwise
Line	*Vector*	*slope* – Slope of the line
		intercept – Intercept (Y-axis cut) of the line
Polygon	*Matrix*	*X* – X-coordinates for the points that are part of the polygon
		Y – Y-coordinates for the points that are part of the polygon
Segment	*Matrix*	*X* – X-coordinates for the two points that form the segment
		Y – Y-coordinates for the two points that form the segment

As a summary, Table 1 describes the eight basic functions included in the package to create geometric objects.

A brief explanation of the parameters which are associated to each of these functions can be found in Table 2.

Table 3 clarifies the structure of the outputs produced by each of the functions and the information which they provide about the geometric object that they represent.

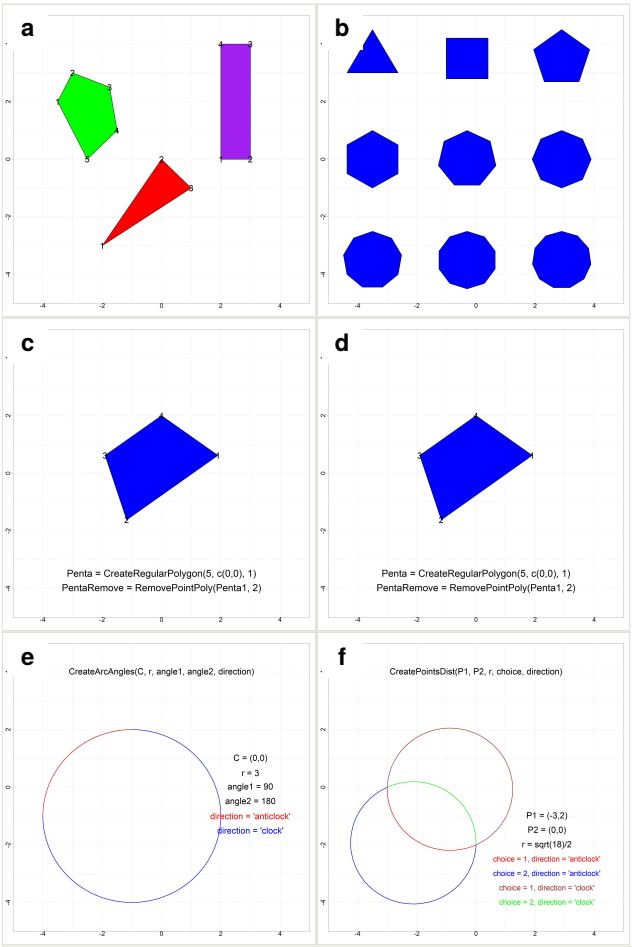

Figure 1.

Several examples of use of some of the basic functions included in the LearnGeom R package with their required lines of code to produce them.

a: Three polygons created with the *CreatePolygon* function. doi
b: Regular polygons with 9 or less sides created with *CreateRegularPolygon*. doi
c: Example of use of the *RemovePointPoly function*. A point is removed from a regular pentagon. doi
d: Example of use of the *AddPointPoly* function. A point is added to a regular pentagon. doi
e: Examples of use of the *CreateArcAngles* function. The red arc follows the anti-clockwise direction; the blue one the clockwise direction. doi
f: Examples of use of the *CreateArcPointsDist* function. The combination of the parameters direction and choice allows the creation of four different arcs doi

Fig. 1 contains examples of use of several functions of the LearnGeom R package, including some of the R instructions needed to use them and their graphical outputs. Fig. 1a shows three polygons of 3, 4 and 5 sides that can be freely defined with the *CreatePolygon* function. Fig. 1b includes the first nine regular polygons as can created by the function *CreateRegularPolygon* with lengthside *l* = 1 and varying center. Fig. 1c, d also contains examples of removing and adding points with the *RemovePointPoly* and *AddPointPoly* functions, along with the lines of code required to obtain these polygons after a regular pentagon (*Penta*) is defined. Fig. 1e, f show examples of use of the *CreateArcAngles* and *CreateArcPointsDist* functions, which attempt to clarify the use of each oftheir parameters.

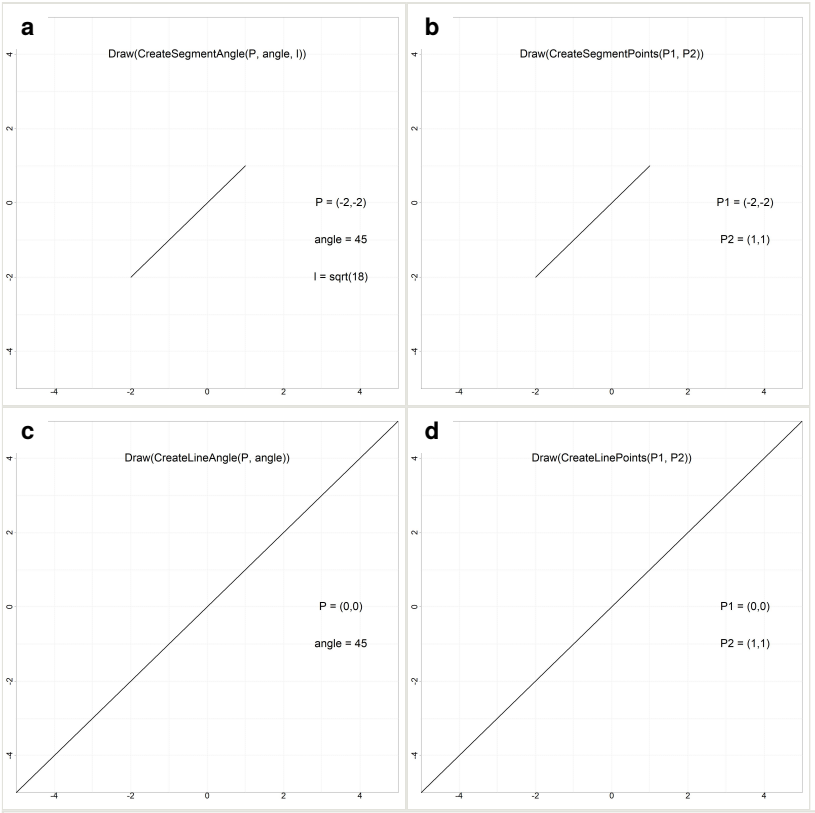

Figure 2.

Examples of use of the specific functions for lines and segments creation. The coincidence in the outputs is due to the choice of equivalent parameters, which are visible in their corresponding figures.

a: Example of use of *CreateSegmentAngle*. doi
b: Example of use of *CreateSegmentPoints*. doi
c: Example of use of *CreateLineAngle*. doi
d: Example of use of *CreateLinePoints*. doi

Fig. 2 contains a couple of examples of use of the specific functions available for creating lines and segments, including the required R code. This figure shows that both lines and segments are the same because the introduced parameters are equivalent. Depending on the situation and the available information, users may find one or the other one as the most suitable tool.

Moreover, if a more experienced user of the language aims to create a line by specifying its slope and intercept, even though there is no function in the package to perform this, it can be achieved by simply defining a two-dimensional vector containing these basic parameters of the line and appending the class *Line* to it. This also applies to the other three geometric objects. However, this is not the use of the package we would expect for a new user of it. The rest of this paper contains multiple examples of use of some functions of the package that have not been mentioned yet and several applications which are possible to perform by combining classical geometric problems, different programming techniques and all the functions included in the package. These applications are thus classified into three categories according to their level of difficulty: basic, intermediate and advanced.

Basic level

Affine transformations

Affine functions are geometric transformations that preserve collinearity and ratios of distances. LearnGeom contains functions to apply six different affine transformations: homothety, reflection, rotation, shear, similarity and translation. Each of these transformations is associated with a 2 x 2 matrix, depending on one or several specific parameters.

All of these functions apply to polygons. However, they also can be used with lines and segments in the case of rotation and translation. For this reason, the names for the functions related to these transformation miss the word *Polygon*.

The use of the affine transformations functions included in the package can be seen in Fig. 3. The lines of code that a user needs to type to produce the outputs included in this figure are also provided. The prior setting of a coordinate plane and the creation and drawing of a triangle (the blue one in Fig. 3) is needed but omitted for greater clarity. Once a triangle has been created, each of the transformations can be applied to it, and both the original triangle and the transformed one can be plotted in the same coordinate plane (in blue and orange, respectively, in Fig. 3).

Reflection

A reflection needs the definition of a line to be used as the axis of reflection, which can be built by the *CreateLinePoints* or *CreateLineAngle* functions (Fig. 3a).

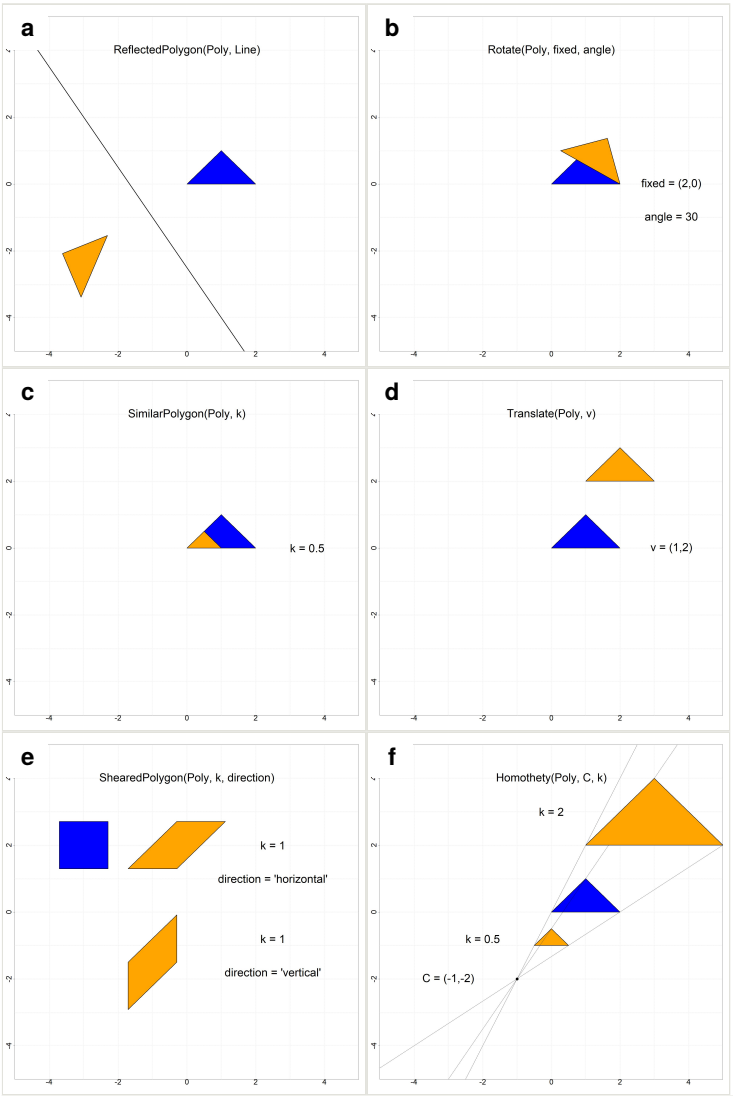

Figure 3.

Examples of use of the functions included in the package that represent affine transformations in the plane. In all the pictures, the blue triangle, placed at the points A(0,0), B(2,0) and C(1,1), is the one passed to each of the functions, being the orange triangle the output resulting for each of the transformations.

a: A reflection. doi
b: A rotation. doi
c: A similarity. doi
d: A translation. doi
e: A shear transformation. doi
f: A homothety. doi

Rotation

The most familiar parameter related to this transformation is the angle of rotation, which is defined as 30º this time. However, this angle does not fully characterise this transformation because it is also necessary the only point remaining fixed after the transformation (for this example, point (2,0) of the triangle) (Fig. 3b).

Similarity

A similarity only needs the selection of one parameter, *k*, in the *SimilarPolygon* package function. This parameter allows the user to create a polygon similar to the original one, which can be a contraction ($k < 1$) or an expansion ($k > 1$), altering the size of the polygon without changing its shape (Fig. 3c).

Translation

Translation simply represents the movement of a polygon in the direction of vector *v*, conserving the angles and lengths of the initial polygon (Fig. 3d).

Shear and Homothety

Shear and homothety are the remaining affine transformations available in the package. Even though they are less common and known than the other four, they offer many possibilities to the user.

ShearedPolygon shares parameter *k* with *SimilarPolygon*, as both transformations enable a change in the size of the polygon. However, the transformation produced by *ShearedPolygon* does not conserve the shape of the object. The *direction* parameter has two possible values, *horizontal* and *vertical*, which allow the deformation of the initial polygon in the two directions of the plane (Fig. 3e).

On the other hand, the function *Homothety* can also produce an enlargement or reduction of a polygon in relation to a point of the plane called the center of the homothety. The function contains an option to display all the lines that connect the points of the original and the transformed polygons passing through the center (Fig. 3f).

Points of the triangle

We can appreciate a combined use of the package fuctions and the R language capabilities by obtaining a notable point of a triangle: its circumcenter, that is, the intersection of its three perpendicular bisector lines.

Building a triangle and its middle points

For example, consider the triangle of points (0,0), (1,1) and (2,0). The *CreatePolygon* function is used for this step. Letters A, B and C are used to represent the three points of the triangle. It is also convenient to calculate and visualize the middle points of the triangle sides. The *MidPoint* function of this package can perform this task (see Fig. 4a).

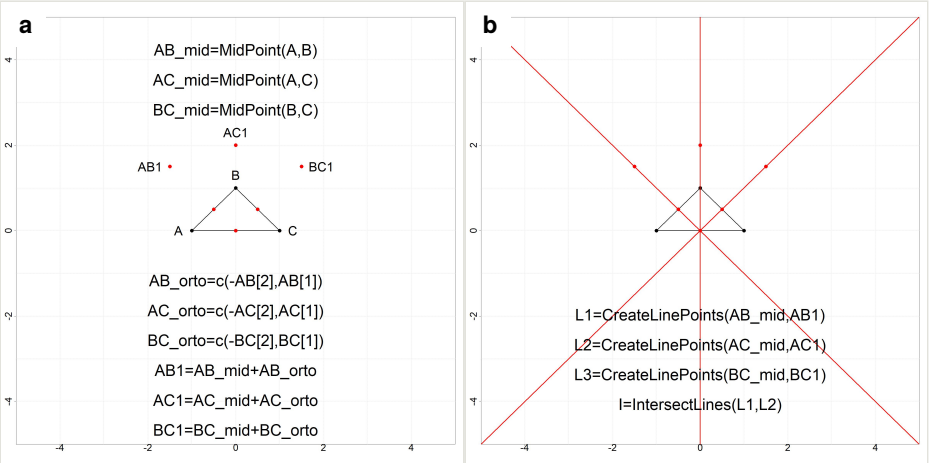

Figure 4.

Partial results during the process of finding the circumcenter of the triangle of points (-1,0), (0,1) and (1,0).

a: Triangle creation and obtention of the middle points of the sides and three auxiliary points in the orthogonal direction of each of the sides. doi

b: Bisector lines and intersection in the circumcenter of the triangle. doi

Orthogonal vectors and auxiliary points

In order to find the bisectors of the sides, it is necessary to find an orthogonal vector to each of them. Prior to that, the vectors that connect the three points have to be computed (defined as the difference between the points). Now, one alternative to find the orthogonal of each of these vectors consists of changing the order of the coordinates and the sign of one of them, as it is shown in the code in Fig. 4a. Following the orthogonal direction, and starting from each of the middle points, three auxiliary new points are found (see Fig. 4a).

Lines creation and intersection

The auxiliary points previously obtained are then connected to the middle points of the sides of the triangle through a line (see Figure Fig. 4b).

As it can be observed in these blocks of code, several functions of the package reduce the difficulty of some of the steps, allowing the user to focus on the most advanced topics

involved in the resolution of the problem, such as orthogonality and operations on points and vectors.

Intermediate level

Tessellations

A tessellation, also known as a tiling, is a pattern that is made by repeating a basic geometric figure, or combination of figures, along a plane Grunbaum and Shephard (1977). Tessellations appear sometimes in nature and are often present in architectural constructions Lu and Steinhardt (2007). For example, beehives, the perfect structures created by bees to produce their honey, can be considered as a tessellation made of regular hexagons. Therefore, with the help of the function *Tessellation*, a user with some experience using this package could be able to generate a pattern that reminds a beehive. Besides, the user could divide the problem into two parts: setting an initial regular hexagon in the plane finding two other hexagons that are contiguous to it and a second part consisting of allowing the extension of the pattern to a wider region of the plane by the appropriate use of the *Tessellation* function.

Setting an initial hexagon

The process starts with the creation of a regular hexagon in the plane (see Fig. 5a).

Contiguous hexagons

In order to obtain two hexagons that are contiguous to the initial one, *Hexa0* can be translated in the direction determined by its center and the middle point of the segment that joins points 1 and 6 (for *Hexa1*) and the one that joins 2 and 3 (for *Hexa2*) (see Fig. 5b).

Creating tessellations

The final step requires the use of the *Tessellation* function with the right separation parameter (see Fig. 5c). The use of the *MidPoint* function and visualization are essential for this task. Moreover, it should be noted that the hexagons are passed to the *Tessellation* function as a list. In general, this fact allows to apply simultaneously the tessellation function to a set of several polygons. In order to achieve the desired pattern, it is very important to set correctly the separation parameter of the function. Otherwise, the resulting pattern could contain some overlaps, which are uncommon in the creation of tessellations. Moreover, one could minimize the code to get the *Hexa1* and *Hexa2* hexagons by using the *ReflectedPolygon* function. It is clearly noticeable that these two hexagons must be symmetric to *Hexa0* about the two lines that connect point 1 with 6, and 2 with 3. The optimized code and its output is included in Fig. 5d. The two options to create *Hexa1* and *Hexa2* seem enough to show that the use of this learning approach could be beneficial to

improve, simultaneously, geometric thinking and programming skills.As another possible application, *Tessellation* could allow the user to attempt a classical mathematical problem: to complete a plane region by tiling polygons Schattschneider (1981). Some functions of the package such as *RotatedPolygon* or *TranslatedPolygon* would be necessary for this matter.

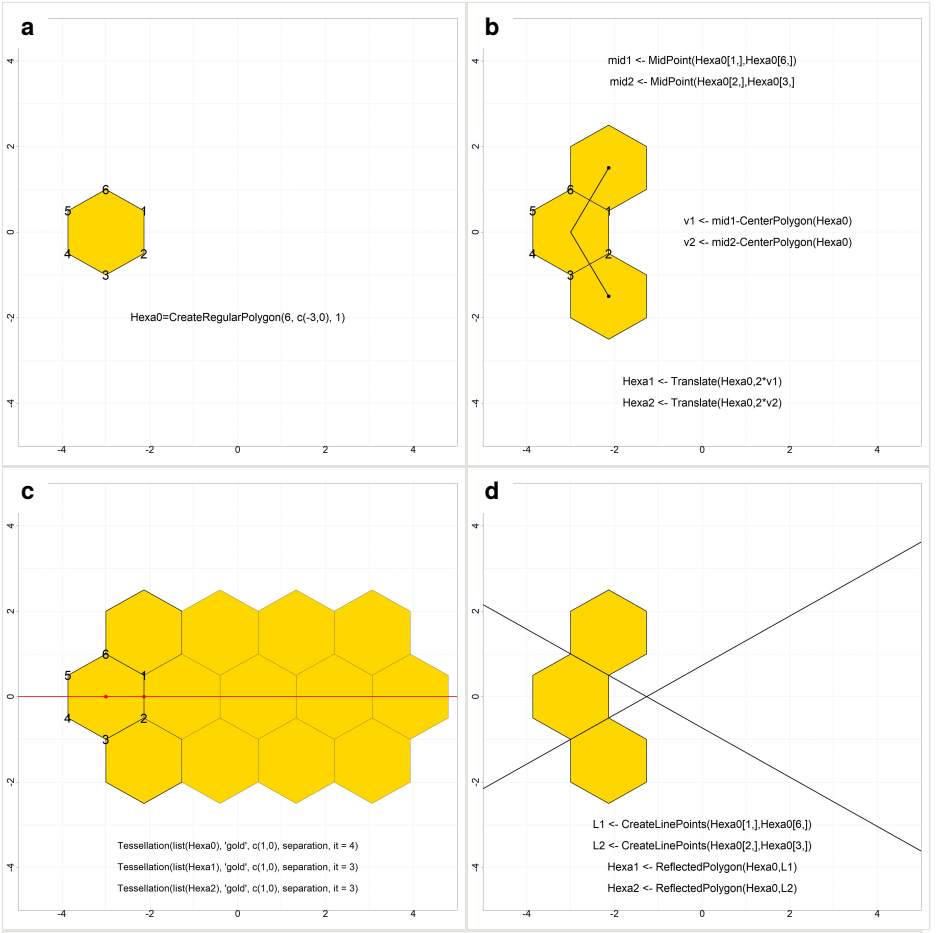

Figure 5.

Different stages of the creation of a beehive structure with the aid of tessellations.

a: Creating a regular hexagon that works as the start of the tessellation. doi

b: Creating two contiguous hexagons to the starting one. These hexagons are derived from the middle points of some of the sides of the initial hexagon. doi

c: Once the contiguous hexagons are obtained, function *Tessellation* allows the creation of the structure. doi

d: Alternative to step in **b**, which minimizes the R code required to obtain the output. This strategy is based on the property of reflection. doi

Working on the real world

The *GetMap* function of the *RgoogleMaps* package gives users the possibility to treat geometric objects over a plane that represents a piece of the world we live in. The zoom parameter ranges from 0 to 21, depending on the location, allowing the user to visualize big pictures of the world but also little details of some buildings.

The function *CoordinateImage* of this package works as *CoordinatePlane*. However, in this case, it is capable of setting the axis and the grid over an image obtained from Google Maps. As an illustration, the approximation of the British coast with a polygon can be performed, which is a problem that is connected to the posterior definition of fractal Mandelbrot (1967). The last lines of this section contain an R code (in *italics*) for achieving this task, in which a polygon of 30 points is created with the aid of the *SelectPoints* function of the package. The first argument of the *GetMap* function is a vector containing the latitude and longitude, in degrees, which are chosen as the center of the picture. Latitude and longitude can range from -90º to 90º and from -180º to 180º respectively as usual. A zoom of 5 is selected for being able to visualize the complete British coast.

```
> library(RgoogleMaps)
> ima <- GetMap(c(53.404059,-3.351493), zoom = 5, maptype = "satellite")
> ima <- ima[[4]]
> CoordinateImage(-10, 10, -10, 10, ima)
> S <- SelectPoints(30)
> Draw(S, c("transparent","white"))
```

Advanced level

Recursive programming

Recursive programming is one of the most efficient strategies to find the solution of some problems. However, it is also a difficult task for novice programmers. For quite advanced students, recursive programming could be treated to build a well-known mathematical structure: a fractal Mandelbrot (1977). Fractals are geometric objects which satisfy the property of self-similarity, which basically means that each of their parts satisfy the same properties and internal relationships as the complete object does.

As an illustration, this definition can be easily imagined with the help of one of the most famous fractals: the Sierpinski triangle Knaster and Kuratowski (1927). The process of building this triangle consists of the following steps:

1. Start from an equilateral triangle.
2. Find the middle points of each side of this triangle.
3. Connect the three points obtained in step 2 to obtain a new triangle. This triangle is also equilateral and rotated 180º with respect to the initial position.

4. Remove the triangle built in step 2 from the initial triangle. Now you have three little triangles inside the initial one. This is the first iteration to build the Sierpinski triangle.

If the steps 1-4 are repeated in each of these three triangles, nine smaller triangles will be obtained, which are similar to the previous ones. This is already the second iteration of the construction and a constant repetition of the steps leads to any iteration of the triangle. The Sierpinski triangle is theoretically defined as the infinite repetition of the steps 1-4. However, from a practical point of view, we consider interesting only the building of the first iterations. This process can be implemented in R with a few lines of code due to the efficiency of recursive programming and some of the functions contained in the LearnGeom R package. For example, the code available in the following lines (in *italics*) is a possible approach to produce the first six iterations of the Sierpinski triangle. It is necessary to create a function *Sierpinski* to call it recursively during the triangle construction:

```
> 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 <- function(Tri, it){
        if (it==1){
                Draw(Tri,"black")
        }
        if (it>1){
                Sierpinski(CreatePolygon(Tri[1,], MidPoint(Tri[1,],Tri[2,]), MidPoint(Tri
[1,],Tri[3,])), it-1)
                Sierpinski(CreatePolygon(Tri[2,] ,MidPoint(Tri[1,],Tri[2,]), MidPoint(Tri
[2,],Tri[3,])), it-1)
                Sierpinski(CreatePolygon(Tri[3,] ,MidPoint(Tri[1,],Tri[3,]), MidPoint(Tri
[2,],Tri[3,])), it-1)
        }
}
> Sierpinski(Tri, it)
```

The R code included just displayed is basically the same one that contains the function *Sierpinski* of the package. This function allows users to visualize the first iterations of the fractal avoiding the difficulty of building it (see Fig. 6a). However, for users ready to learn recursive programming, the best option would be to propose the construction on their own. The package also contains a function related to the Koch curve, another very well-known fractal Koch (1904). The most famous version of this fractal is called Koch's snowflake, which can be obtained by following the next three steps, starting from a segment in the plane (see Fig. 6b):

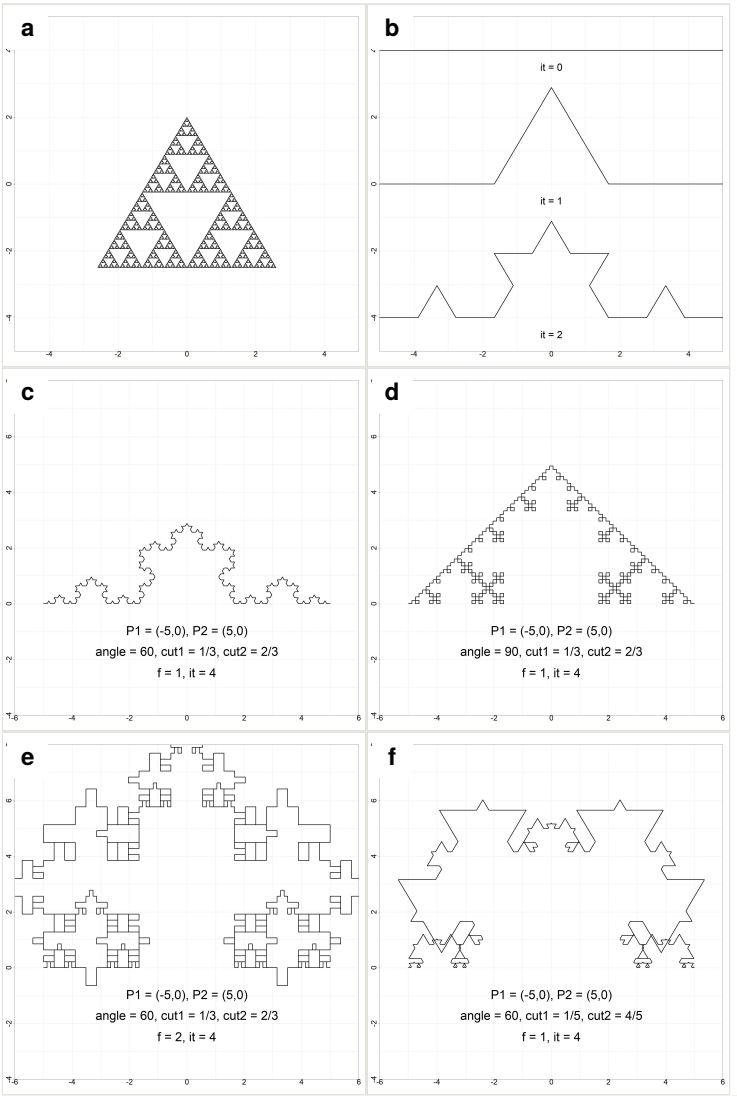

Figure 6.

Examples of different fractals produced with the functions of the LearnGeom package. The examples obtained with *FractalSegment* show how minimal modifications of the parameters can lead to very different curves.

a: First seven iterations of the Sierpinski triangle. doi
b: First three first iterations of the Koch's curve. doi
c: First five first iterations of the Koch's as obtained with *FractalSegment*. doi
d: A modificaction of the first five iterations of the Koch's (**c**) by changing parameter *angle* from 60º to 90º. doi
e: A modificaction of the first five iterations of the Koch's (**c**) by changing parameter *f* from 1 to 2. doi
f: A modificaction of the first five iterations of the Koch's (**c**) by changing parameters *cut1* and *cut2* from 1/3 and 2/3 to 1/5 and 4/5, respectively. doi

1. Divide the segment into three equal segments.
2. Replace the middle segment by the two sides of an equilateral triangle whose side length is the same as for the segment being removed. Now, there are four segments of the same length.
3. Build an equilateral triangle by using the middle segment as its base. Remove this segment to leave four different segments in the picture.

The continuous application of these steps to each of the new segments that are created after each iteration produces every state of the fractal. The *FractalSegment* function, available in this package, allows the user to create an infinity number of curves whose construction is based on these steps. The user can experiment with *cut*, *angle* and *f* parameters to achieve all kind of fractals (starting from iteration *it = 0*). The *cut* parameter allows a non-equal division of the segments at step 1, however, the *angle* parameter allows the user to build more than an equilateral triangle at step 2. Finally, the *f* parameter, which is a positive number, produces a length enlargement or a reduction for the new segments at each iteration. Users are free to try these parameters and produce some fractals like those shown in figures Fig. 6c, d, e, f. As it can be observed in these figures, the fractal complexities and diversities are enormous, including auto-intersections. For this reason, to display fractals for the first iterations is really helpful to better understand the underlying process of construction.

Generation of curves

A trochoid is a closed curve that can be obtained by the conjunction of three geometric figures: two circles, from which one of them is fixed and the other mobile, and a mobile point, which is connected to the mobile circle. There are three parameters to characterize each trochoid: the fixed circle radius, the mobile circle radius and the distance from the mobile point to the center of the mobile circle Armon (1996).

The presence of these parameters can be used to define a trochoid as a set of parametrical equations, involving trigonometry functions. However, as it was first proposed by Abelson and DiSessa (1986) with the aid of the LOGO turtle, these curves can be approximated by iteratively drawing segments of certain lengths and angles. The LOGO turtle, which originally was a real robot developed at Massachusetts Institute of Technology at the end of the 60s, refers to an on-screen cursor implemented in the LOGO language that was capable of responding to easy instructions from the user (basic direction setting of the turtle and rectilinear movements). This turtle is also available in R in the package TurtleGraphics. As it is shown in the publication of Abelson and DiSessa (1986), the procedure to obtain any trochoid needs a pair of angles and a pair of lengths (and an initial point to start the curve). An iterative process is then defined based on these angles and lengths, which makes possible the continuous generation of every kind of trochoid.

This package also includes a function called *Star*, which can be utilised to represent very different star-like objects as another application of the LOGO turtle also taken from Abelson and DiSessa (1986). This function is based on the generation of the object by drawing

segments iteratively as *Duopoly*. The variation of the parameters included in the function render possible to understand the basis of the process and can incite the user to achieve many kinds of star-like shapes. Thus, Fig. 7 represents some examples of use of the functions *Duopoly* (Fig. 7a, b, c) and *Star* (Fig. 7d, e, f). The parameters *color* and *time* available in the function let the user to appreciate all the points that are drawn in the generation of the curve and the order at which they are drawn.

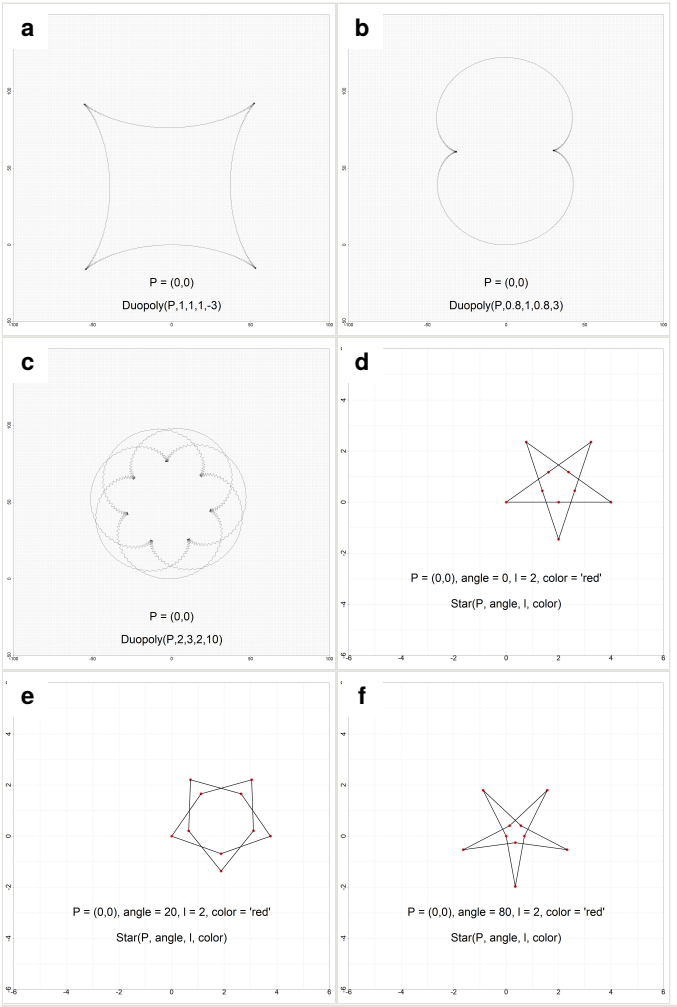

Figure 7.

Examples of use of the functions *Duopoly* and *Star*.

a: Example of use of the function *Duopoly*. Creation of an astroid. doi
b: Example of use of the function *Duopoly*. Creation of a nefroid. doi
c: Example of use of the function *Duopoly*. Creation of an epicycloid doi
d: Example of use of the function *Star*. Parameter *angle* is set to 0º. doi
e: Example of use of the function *Star*. Parameter *angle* is set to 20º. doi
f: Example of use of the function *Star*. Parameter *angle* is set to 80º. doi

Conclusions

Currently, there are some powerful and useful free software products, such as GeoGebra and Scratch, to learn geometry at undergraduate level. However, all of them present some limitations when compared to common programming languages. Although the R programming language is mainly focused on statistical computing, it can also be employed as a pedagogical tool for simultaneously learning mathematical concepts and advanced programming skills. In this work, an R programming package named LearnGeom is presented as a novel pedagogical tool that provides a set of functions to facilitate the exploration of plane geometry while programming in R. Beginning with some easy functions and definitions, the combined use of the functions included in the package with the own capabilities of the language itself offers a novel teaching source for current secondary educators.

Conflict of Interest

The authors declare that they have no conflict of interest.

Acknowledgements

The authors would like to acknowledge the Universidad Católica de Valencia San Vicente Mártir and the Ministry of Economy, Industry and Competitiveness for the financial support of this work through the 2018-231-001UCV and MAT2015-69315-C3-1-R grants respectively.

References

- Abelson H, DiSessa AA (1986) Turtle geometry: The computer as a medium for exploring mathematics. MIT press
- Akkaya A, Tatar E, Kağızmanlı TB (2011) Using Dynamic Software in Teaching of the Symmetry in Analytic Geometry: The Case of GeoGebra. Procedia - Social and Behavioral Sciences 15: 2540-2544. https://doi.org/10.1016/j.sbspro.2011.04.141
- Akpinar Y, Aslan Ü (2015) Supporting Children's Learning of Probability Through Video Game Programming. Journal of Educational Computing Research 53 (2): 228-259. https://doi.org/10.1177/0735633115598492
- Armon U (1996) Representing trochoid curves by DUOPOLY procedure. International Journal of Mathematical Education in Science and Technology 27 (2): 177-187. https://doi.org/10.1080/0020739960270202
- Benton L, Hoyles C, Kalas I, Noss R (2016) Building mathematical knowledge with programming: insights from the ScratchMaths project.
- Benton L, Saunders P, Kalas I, Hoyles C, Noss R (2018) Designing for learning mathematics through programming: A case study of pupils engaging with place value.

International Journal of Child-Computer Interaction https://doi.org/10.1016/j.ijcci.2017.12.004

- Booth KHV (1958) Programming for an automatic digital calculator. Butterworths
- Briz-Redón Á, Serrano-Aroca Á (2018) LearnGeom: An R-package for Learning Plane Geometry. URL: https://doi.org/10.5281/zenodo.1211646
- Du Boulay B (1986) Some difficulties of learning to program. Journal of Educational Computing Research 2 (1): 57-73.
- Falloon G (2016) An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. Journal of Computer Assisted Learning 32 (6): 576-593. https://doi.org/10.1111/jcal.12155
- Feurzeig W, Papert S, Lawler B (2011) Programming-languages as a conceptual framework for teaching mathematics. Interactive Learning Environments 19 (5): 487-501. https://doi.org/10.1080/10494820903520040
- Foerster K (2016) Integrating Programming into the Mathematics Curriculum: Combining Scratch and Geometry in Grades} 6 and 7. Proceedings of the 17th Annual Conference on Information Technology Education.
- Govender I (2009) Learning to program, learning to teach programming: pre-and in service teachers' experiences of an object-oriented language. University of South Africa
- Grunbaum B, Shephard GC (1977) Tilings by regular polygons. Mathematics Magazine 50 (5): 227-247.
- Hohenwarter M (2002) GeoGebra: Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene. Paris Lodron University, Salzburg, Austria
- Howe JA, Ross PM, Johnson KR, Plane F, Inglis R (1981) Teaching mathematics through programming in the classroom. Computer Assisted Learning. https://doi.org/10.1016/b978-0-08-028111-7.50017-9
- Knaster B, Kuratowski C (1927) A connected and connected im kleinen point set which contains no perfect subset. Bulletin of the American Mathematical Society 33 (1): 106-110. https://doi.org/10.1090/s0002-9904-1927-04326-9
- Koch H (1904) On a Continuous Curve Without Tangent Constructable from Elementary Geometry. Classics on fractals.
- Lehrer R, DeBernard A (1987) Language of learning and language of computing: The perceptual-language model. Journal of Educational Psychology 79 (1): 41-48. https://doi.org/10.1037/0022-0663.79.1.41
- Linn M, Dalbey J (1985) Cognitive consequences of Programming Instruction: Instruction, Access, and Ability. Educational Psychologist 20 (4): 191-206. https://doi.org/10.1207/s15326985ep2004_4
- Lu PJ, Steinhardt PJ (2007) Decagonal and Quasi-Crystalline Tilings in Medieval Islamic Architecture. Science 315 (5815): 1106-1110. https://doi.org/10.1126/science.1135491
- Mandelbrot B (1967) How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension. Science 156 (3775): 636-638. https://doi.org/10.1126/science.156.3775.636
- Mandelbrot BB (1977) Fractals. Wiley Online Library
- Mascaró M, Sacristán AI, Rufino M (2014) Teaching and learning statistics and experimental analysis for environmental science students, through programming activities in R. Constructionism and Creativity-Proceedings 3rd Intl. Constructionism Conf.

- Mascaró M, Sacristán AI, Rufino M (2016) For the love of statistics: appreciating and learning to apply experimental analysis and statistics through computer programming activities. Teaching Mathematics and its Applications 35 (2): 74-87. https://doi.org/10.1093/teamat/hrw006
- Masri RM, Ting SH, Zamzamir Z, Ma'amor RLZR (2017) The effects of using GeoGebra teaching strategy in Malaysian secondary schools: A case study from Sibu, Sarawak. Geografia-Malaysian Journal of Society and Space 12 (7): .
- McCracken DD (1957) Digital computer programming. John Wiley & Sons
- Palumbo D (1990) Programming Language/Problem-Solving Research: A Review of Relevant Issues. Review of Educational Research 60 (1): 65. https://doi.org/10.2307/1170225
- Papert S (1980) Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.
- Priatna N (2017) AIP Conference Proceedings. The application of brain-based learning principles aided by GeoGebra to improve mathematical representation ability. https://doi.org/10.1063/1.4995157
- Pruim R, Kaplan DT, Horton NJ (2017) The mosaic Package: Helping Students to Think with Data Using R. R Journal.
- Resnick M, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Millner A, Rosenbaum E, Silver J, Silverman B (2009) Scratch: programming for all. Communications of the ACM 52 (11): 60-67.
- Saha RA, Mohd Ayub AF, Tarmizi RA (2010) The Effects of GeoGebra on Mathematics Achievement: Enlightening Coordinate Geometry Learning. Procedia - Social and Behavioral Sciences 8: 686-693. https://doi.org/10.1016/j.sbspro.2010.12.095
- Schattschneider D (1981) In Praise of Amateurs. The Mathematical Gardner. https://doi.org/10.1007/978-1-4684-6686-7_16
- Soloway E (1993) Should we teach students to program? Communications of the ACM 36 (10): 21-25.
- Spohrer JC, Soloway E, Pope E (1985) Where the bugs are. ACM SIGCHI Bulletin.
- Statz J, Miller L (1978) The Egg Series: Using Simple Computer Models. The Mathematics Teacher 71 (5): 459-467.
- Team RC (2018) R: A Language and Environment for Statistical Computing. URL: https://www.R-project.org/
- Wang H, Huang I, Hwang G (2015) Comparison of the effects of project-based computer programming activities between mathematics-gifted students and average students. Journal of Computers in Education 3 (1): 33-45. https://doi.org/10.1007/s40692-015-0047-9
- Zengin Y, Furkan H, Kutluca T (2012) The effect of dynamic mathematics software geogebra on student achievement in teaching of trigonometry. Procedia - Social and Behavioral Sciences 31: 183-187. https://doi.org/10.1016/j.sbspro.2011.12.038