International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:11, No:2, 2017

Jointly Learning Python Programming and Analytic Geometry

Cristina-Maria Păcurar

***Abstract*—**The paper presents an original Python-based application that outlines the advantages of combining some elementary notions of mathematics with the study of a programming language. The application support refers to some of the first lessons of analytic geometry, meaning conics and quadrics and their reduction to a standard form, as well as some related notions. The chosen programming language is Python, not only for its closer to an everyday language syntax – and therefore, enhanced readability – but also for its highly reusable code, which is of utmost importance for a mathematician that is accustomed to exploit already known and used problems to solve new ones. The purpose of this paper is, on one hand, to support the idea that one of the most appropriate means to initiate one into programming is throughout mathematics, and reciprocal, one of the most facile and handy ways to assimilate some basic knowledge in the study of mathematics is to apply them in a personal project. On the other hand, besides being a mean of learning both programming and analytic geometry, the application subject to this paper is itself a useful tool for it can be seen as an independent original Python package for analytic geometry.

***Keywords*—**Analytic geometry, conics, Python programming language, quadrics.

I. Introduction

NOWADAYS, the most suitable words to describe our world constantly change. Therefore, learning should keep up with the new tendencies and adapt to all the changes. The present paper combines the learning of a new programming language with the advanced learning of basic analytic geometry notions.

The programming language chosen is Python. According to the IEEE *Spectrum*'s Top Ten Languages for 2016 [4], Python has just reached the top three, being preferred by users right after the old consecrated C and Java. The quick growth for Python preference is a significant argument in favor of learning this programming language and, as well, learning through it.

The chapter of analytic geometry related to conics and quadrics is of utmost importance in understanding further more complex notion. Also, conics and quadrics have remarkable applications in numerous fields.

One of the reasons that this chapter of analytic geometry has been chosen is the thorough algorithmically approach used to actually reduce a conic (or a quadric) to its canonical form.

Cristina-Maria Păcurar is a student at Transylvania University of Brasov, Faculty of Mathematics and Informatics, Brasov, no. 50 Iuliu Maniu street, Postal Code: 500091, Romania (phone: 004 0268 412 776; e-mail: maria.pacurar@ xu.unitbv.ro).

II. Python Programming

Python is a dynamic programming language in active development since 1989. Checking the official Python site the following words, which briefly, but best describe this programming language, welcome the users:

"Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open"[3].

Following the description provided by the developers themselves, Python appears to be one of the most appropriate programming languages to start with. Its simplicity is supported by one of the most widely spread examples in favor of Python, meaning the "Hello World" program. Every programmer's first program requires just the: print 'Hello World!' line or print ('Hello World!') for the Python 3 version. Compared to the amount of code require to do the same task in other programming languages, simplicity becomes one of the top advantages of using Python.

Another keyword for Python may be strictness. The syntax of Python follows the *off-side* rule. Thus, special attention must be paid to indentations, as this is how code blocks are delimited instead of brackets used by other programming languages. Although following strictly those indentations might seem a disadvantage, it may be compared to a dress code that is restrictive at first, but then saves us both precious time and teaches us to have a certain behavior. The architecture of a code is more important than it might seem at a first glance, as it is a key point in writing readable and reusable code.

Another important advantage of the code structure that Python utilizes, and the way that it facilitates the communication between programmers on top of a piece of code, is reflected in a multitude of open-source projects available on the internet, with several authors and many more contributors.

The way that the Python code is formatted not only contributes to the code readability and determines the new programmer to be more organized and therefore more productive, but it is also closer to the everyday language.

One more top advantage of using Python programming language is the easy built and usage of modules. Organizing the code into smaller, independent source-code files that can be reused as modules anytime, anywhere with a simple import into the main program is of utmost importance. The usage of modules saves precious resources, time, and contributes to a neat code.

III. Analytic Geometry

A. Conics

Conics are a special class of plane curves that possess remarkable properties. The study of conics is compulsory as they have numerous applications in various domains.

Non-degenerated conics have been obtained in the Hellenistic period as the intersection of a circular cone with a plane.

1. General Notions

Definition 1. A set of points in the Euclidean plane E_2 is called a **conic**, if the coordinates of its point satisfy the equation:

$$a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y + a_{33} = 0$$

Let $\mathbf{A} = (a_{ij})$, be the matrix associated to the conic, called **matrix of the conic** (γ).

Let us consider

$$\begin{aligned} \Delta &= \det(A) \text{– cubic invariant} \\ \delta &= a_{11}a_{22} - a_{12}a_{21} \text{ – quadratic invariant} \\ I &= a_{11} + a_{22} \text{ – linear invariant} \end{aligned} \quad (1)$$

Definition 2. The quantities **Δ, δ** and **I** are the **invariants** of a conic, hence Δ = Δ', δ= δ', I=I'. (as demonstrated in reference textbook [2])

Definition 3. The center of symmetry of the set of points belonging to (γ) is called the **center** of the conic (γ).

Let us consider the system

$$\begin{cases} a_{11}x_0 + a_{12}y_0 + a_{13} = 0 \\ a_{11}x_0 + a_{12}y_0 + a_{13} = 0 \end{cases} \quad (2)$$

The system (2) represents the equation of the center $C = (x_0, y_0)$ of the conic(γ). In order to determine whether the center exists, the quadratic invariant must be studied. If it is non-zero, then the system listed above has a unique solution, which represents the center of the conic. If δ is zero, then the system has no solution or infinity many solutions, thus the center of the conic does not exist or is not unique at a finite distance.

Let us associate the characteristic equation

$$\det(\delta - I\lambda) = 0 \quad (3)$$

Solving (3), the eigenvalues λ_1 and λ_2 are found.

2. Reduction of the Equation of a Quadric to the Standard Form

In order to study the reduction of the equation of a conic to the canonical form, a division must be made based on the center of the conic. The existence of the center is based on the observations made in Section II *A (General notions)*. The algorithms are slightly different for the two cases, and thus are treated separately.

Δ (nature)	δ (class)		
Δ ≠ 0 Non-degenerated conics	δ> 0	I Δ < 0	Real ellipse
		I Δ > 0	Empty set
	δ= 0		Parabola
	δ< 0		Hyperbola
Δ = 0 Degenerated conics	δ> 0		Double point
	δ= 0		Pair of parallel or confounded lines or the empty set
	δ< 0		Pair of intersecting lines

Fig. 1 Isometric classification of conics

For the conics that have a center, their canonical equation may be written knowing only the orthogonal invariants Δ, δ and I. Based on (3) and the eigenvalues λ_1 and λ_2, the standard form of the equation of the conic (γ) is written as:

$$\lambda_1 X^2 + \lambda_2 Y^2 + \frac{\Delta}{\delta} = 0 \quad (4)$$

For degenerated conics, (4) becomes:

$$\lambda_1 X^2 + \lambda_2 Y^2 = 0 \quad (5)$$

For a conic without a center, that has the cubic invariant Δ nonzero, the standard form is

$$Y^2 = 2pX \text{ , where } p = \sqrt{\frac{-\Delta}{I^3}} \quad (6)$$

If Δ equals 0, then p becomes 0, which implies the equation: $Y^2 = 0$

B. Quadrics

Quadrics represent an important class of surfaces in space in the Euclidean theory of surfaces. Like conics, quadrics also have various applications in different domains.

In the punctual three dimensional Euclidian space E3 = (E3, V3, ~), in the Cartesian frame, quadrics are analytically characterized by second-degree equations.

1. General Notions

Definition 4. In the punctual Euclidean space E_3, is called **quadratic** surface the locus of the points that satisfy the implicit equation:

$$a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12} + 2a_{12}xy + 2a_{23}yz + 2a_{13}xz + 2a_{14}x + 2a_{24}y + 2a_{34}z + a_{44} = 0$$

Let $\bar{\mathbf{A}} = (a_{ij})$, *b*e the matrix associated to the conic, called **matrix of the quadric**(Σ).

Let

$$\begin{aligned} \Delta &= \det(\bar{A}) \\ \delta &= \det(A), \text{ where} A = (a_{ij}), \text{ for i, j} = \overline{1,3} \end{aligned} \quad (7)$$

$$J = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix}$$
$$I = a_{11} + a_{22} + a_{33}$$

Definition 5. The quantities **δ, I**, **J, and rankA** are the **invariants** of a quadric surface, hence δ= δ', I=I', J=J' and rankA= rankA'. (as demonstrated in reference textbook [2]).

Δ	δ		
$\Delta \neq 0$ non-degenerated quadrics	$\delta \neq 0$ - with a center		Ellipsoids Hyperboloids Empty Set
	$\delta = 0$ - without center		Paraboloids
$\Delta = 0$ degenerated quadrics	$\delta \neq 0$ - with a center		Cones Empty set
	$\delta = 0$ with a straight line of centers	$J \neq 0$	Cylinders Empty set
		$J = 0$	Parabolic cylinders Parallel planes Empty set

Fig. 2 Isometric classification of quadrics

The center of a quadric is determined by δ. If δ is nonzero, then the quadric is with center, otherwise, it is without center.

Solving the characteristic equation associated to a surface: $\det(\delta - I\lambda) = 0$, the eigenvalues λ_1, λ_2, and λ_3 are obtained.

2. Reduction of the Equation of a Quadric to the Canonical Form

Similar to conics, in order to study the reduction of the equation of a quadric to the canonical form, a separation must be done based on the center of the conic. The existence of the center is based on the observations made in Section II B *(General notions)*.

In order to reduce a quadratic surface, the study of invariants is, as for conics, compulsory. Depending on δ, we distinguish two cases. First one, for δ nonzero, the canonical form which corresponds to the invariants Δ and δ, and the eigenvalues λ_1, λ_2, and λ_3 of the given quadric (Σ) is:

$$\lambda_1 X^2 + \lambda_2 Y^2 + \lambda_3 Z^2 + \frac{\Delta}{\delta} = 0 \quad (8)$$

For quadratic surfaces with a center, being degenerated implies Δ= 0, in which case, the free term, $\frac{\Delta}{\delta}$, becomes null.

If δ is zero, and Δ is nonzero, then the canonical equation associated is as follows:

$$\lambda_1 X^2 + \lambda_2 Y^2 \pm 2\sqrt{\frac{-\Delta}{J}} Z = 0 \quad (9)$$

If both Δ and δ are zero, then there are more than one special cases discriminated by a set of additional parameters J is an invariant, that correspond to a different reduced equation.

IV. The Basic Application

The application is built entirely in Python 2.7 and consists of several smaller programs that are gathered by a configuration file. The *geome.config* file is responsible to enable each of the used files: *get_coeficients.py*, *get_matrix.py*, *get_invariants.py*, *get_type.py*, *get_center.py*, *get_eigenvalues.py* and *get_equation.py*.

Each independent feature which approaches a general task that is likely to be needed again in a different context is treated in a separate file. Thus, all topics of analytic geometry that contribute to the main project are addressed in a separate file that may be later imported as a module in any other application.

The first step in building the application is to collect the coefficients of the given equation, done in the *get_coeficents.py* file. The most facile way of doing it is by using the built in function *raw_input* for Python2 or just *input* for Python3. Although the number of coefficients is different for a conic's equation, which has six coefficients and a quadric's equation at which ten coefficients occur, the algorithm for collecting them is identical.

For a better visualization and in order to follow the algorithm exactly, the *get_matrix.py* file is dedicated to the construction of the matrix associated to a conic (**A**) or a quadric ($\bar{\mathbf{A}}$). Although the matrix may be built as a multidimensional array, the *numpy* package (available at [6]), designed for numerical computation may also be imported.

A separate file, *get_invariants.py*, operates with the matrix, and the invariants of the conic, and the quadric, respectively. Based on the formulas (1) and (7), independent functions are built in order to construct Δ, δ, and I for conics, and Δ, δ, I, and J for quadrics. The *numpy* package may be used in order to work with matrixes as it has built-in functions designed especially to solve a determinant, or calculate the trace of a matrix. However, as the number of coefficients is relatively low, the usage of external packages is not compulsory, and the computations may be handwritten based only on the coefficients in the source-code of the file.

The *get_type.py* file is responsible for the isometric classification regarding conics, based on orthogonal invariants provided by the *get_invariants.py* file that is imported as a module. As shown in Fig. 1, the classification may easily be transposed in code by means of an extended conditional statement which can be visualized in Fig. 3, according to the variation of Δ, δ, and respective Iδ. Regarding quadrics, a similar approach may be utilized, as it can be seen in Fig. 2.

As the name properly suggests, the *get_center.py* file is built in order to get the coordinates of the center following equation (1).

The following step is to obtain the eigenvalues. The *get_eigenvalues.py* module utilizes the characteristic equation listed at (3) in order to return the required values. Once again, a personal, but more time-consuming method to solve a quadratic equation with the unknown λ is constructed. However, the more elegant and quicker usage of *numpy* is preferred.

Finally, the main application gathers all the necessary information from the previous built files in order to return the reduced equation that has been searched.

The final part, meaning the reduction of the equation, is composed of two separate parts; for the conic's part,

respectively, the quadratic surfaces' one. Based on the observations made in Sections II *A* and *B*, a complex conditional statement (similar to the one constructed in order to get the type) is used in order to return the correct result. The values for the invariants and for the eigenvalues are gathered through modules from previous built files and used in this main file.

```
29
30 def set_type():
31     if DELTA != 0:
32         if delta > 0:
33             if DELTA*I < 0:
34                 return 'The conic is a nondegenerated real ellipse.'
35             elif DELTA*I > 0:
36                 return 'The conic is the empty set.'
37         elif delta == 0:
38             return  'The conic is a nondegenerated parabola.'
39         elif delta < 0:
40             return  'The conic is a nondegenerated hyperbola.'
41     else:
42         if delta > 0:
43             return 'The conic is a degenerated conic of ellipse type, a double point.'
44         elif delta == 0:
45             return 'The conic is a degenerated conic parabola type, a pair of lines or empty s
46         elif delta < 0:
47             return 'The conic is a degenerated conic of hyperbola type, a pair of intersecting
48
49 what_type = set_type()
50
51 if Config.get('main','debug') == str(1):
52     print what_type
53
```

Fig. 3 Python code for conics classification

Thus, in function of the existence – or nonexistence – of a center, more explicitly, based on whether δ is zero, or nonzero, (4), (5), or (6) are being returned for a conic. Similarly, (8) and (9) are the result for the reduction of a quadratic surface.

V. A Basic UI And an Introduction to TKINTER

The next step in building the application is constructing the basis for a first easy User Interface (UI). The most appropriate and facile way to build a UI in Python is to use the preinstalled package, Tkinter which is "the standard Python interface to the Tk GUI toolkit. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows systems (Tk itself is not part of Python; it is maintained at ActiveState)" [5].

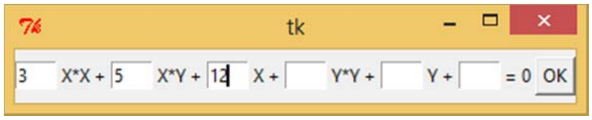

Fig. 4 GUI for collecting the coefficients

The only interaction with the user that the application requires is requesting the user to input the coefficients for the equation. Although the previous used function *raw_input* does not fail in collecting the needed information, building a friendlier UI is preferred.

For the application, the UI is responsible for getting the coefficients of the equation. One of the advantages is that the knowledge required to build this UI is basic. On the other hand, the "Tk Interface" is an important tool in building simple, but strong desktop UIs. For every new programmer, learning how to build a UI is a significant advantage.

```
67
68         self.y_entry = Entry(top, width=4)
69         self.y_entry.pack(side='left')
70         self.yunit_label = Label(top, text='Y +')
71         self.yunit_label.pack(side='left')
72
73         buttons.append(self.y_entry)
74
75         self.constant_entry = Entry(top, width=4)
76         self.constant_entry.pack(side='left')
77         self.constantunit_label = Label(top, text='= 0')
78         self.constantunit_label.pack(side='left')
79
80         buttons.append(self.constant_entry)
81
82         ok = Button(top, text="OK", command=self.ok)
83         ok.pack(pady=5)
84
85     def ok(self):
86         global a, b, c, d, e, f
87         a = self.xx_entry.get()
88         a = float(a)
89         b = self.xy_entry.get()
90         b = float(b)
```

Fig. 5 Python code for building the UI with Tkinter

The code written in order to create the simple UI in order to request the user to introduce the coefficients can be seen in in Fig. 4. As shown in Fig. 5, building this use interface requires some simple Entry widgets attached to each factor of the equation. An Entry widget is available in the Tkinter package as a standard widget used in order to enter or display a line of text. Therefore, the contents entered by the user are passed by as a String and must be converted to a float. Python's easily way of converting variables from some data type to another is a huge plus in this situation.

In order to collect the information entered by the user, a button – here, ok button – must be implemented. The entry widgets and the button are created in separate functions that are part of the unique class Coefficients.

VI. Conclusion and Further Development

The built application is a mean of learning the basic concepts of analytic geometry required to reduce a conic, and a quadric, respectively. Python is a preferred programming language for mathematical purposes as the various examples – from beginner to expert – presented in reference book [1]. Nonetheless, by implementing the algorithm learned in order to reduce to the canonical form some given equations for conics and quadrics, various areas of Python programming are being exploited, from working with mathematical functions and the special package numpy dedicated to numerical computation, to creating a simple UI. On the other hand, the application outlines the advantages of using Python. Besides benefiting of a syntax closer to everyday language, and thus being more accessible, Python's major advantage that emerges from the way that the application is built, is the easy way to create and make use of modules that Python offers.

The separate and independent files that compose the whole application are subsequent one to another. Although the splitting of a program in multiple independent small programs may seem to produce a division and require more valuable time, this strategy is of utmost importance. Working with modules that Python encourages facilitates the avoidance of WET, or Write Everything Twice, problem often encountered in programming.

For further development, the application is going to be enriched with a complete UI. More features regarding other topics of analytic geometry, as well as more advanced parts related to conics and quadrics are to be added.

As Python is becoming more and more powerful, some ways of developing android applications entirely in Python are already popular. Thus, the application is to be developed for android use and become a useful tool in learning analytic geometry in everyone's pocket.

References

[1] Langtangen H.P., *A primer on Scientific Programming with Python.*
[2] STOICA E. et al , *Linear Algebra, Analytic Geometry, Differential Geometry,* Brasov: Editura Universitatii "Transilvania", 2008, pp. 119-160.
[3] About Python, https://www.python.org/about/ Accessed on 02/01/2017.
[4] The 2016 top ten programming languages, http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages Accessed on 20/10/2016.
[5] Tkinter — Python interface to Tcl/Tk, https://docs.python.org/3/library/tkinter.html?highlight=tkinter#module-tkinter Accessed on 04/01/2017.
[6] Numpy, http://www.numpy.org/ Accessed on 04/01/2017.