

## Article

# Analytical Solution for the Problem of Point Location in Arbitrary Planar Domains

Vitor Santos 

Department of Mechanical Engineering, Institute for Electronics Engineering and Informatics of Aveiro,  
University of Aveiro, 3810-193 Aveiro, Portugal; vitor@ua.pt

**Abstract:** This paper presents a general analytical solution for the problem of locating points in planar regions with an arbitrary geometry at the boundary. The proposed methodology overcomes the traditional solutions used for polygonal regions. The method originated from the explicit evaluation of the contour integral using the Residue and Cauchy theorems, which then evolved toward a technique very similar to the winding number and, finally, simplified into a variant of ray-crossing approach slightly more informed and more universal than the classic approach, which had been used for decades. The very close relation of both techniques also emerges during the derivation of the solution. The resulting algorithm becomes simpler and potentially faster than the current state of the art for point locations in arbitrary polygons because it uses fewer operations. For polygonal regions, it is also applicable without further processing for special cases of degeneracy, and it is possible to use in fully integer arithmetic; it can also be vectorized for parallel computation. The major novelty, however, is the extension of the technique to virtually any shape or segment delimiting a planar domain, be it linear, a circular arc, or a higher order curve.

**Keywords:** cauchy theorem; residue theorem; Jordan curve theorem; generalized polygons; complex calculus; parametric curves; Bézier segments; winding number; negative real axis intersection



**Citation:** Santos, V. Analytical Solution for the Problem of Point Location in Arbitrary Planar Domains. *Algorithms* **2024**, *17*, 444. <https://doi.org/10.3390/a17100444>

Academic Editors: Shuai Li, Dunhui Xiao and Frank Werner

Received: 27 July 2024

Revised: 19 September 2024

Accepted: 3 October 2024

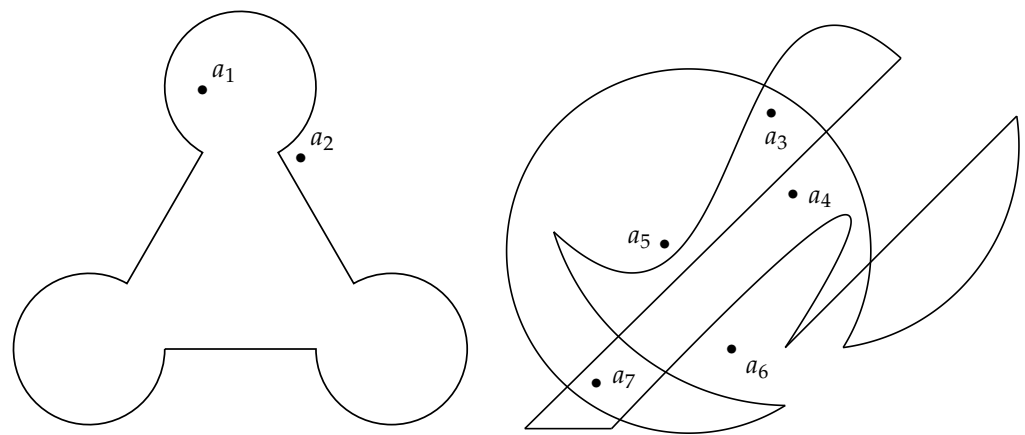
Published: 5 October 2024



**Copyright:** © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Testing whether a point is enclosed by a polygon is a problem that many programmers in numerous fields, such as in computer graphics, geographic information systems, machine vision, or robotics, among others, have certainly come across more than once, as many authors have been asserting for a long time [1]. Indeed, the issue is very important and is the object of several concerns in Computational Geometry. It has already been studied and solved for practically all cases of polygonal regions, despite the fact that some earlier algorithms occasionally needed iterative computations, were ambiguous in special cases of on-boundary point locations, or show numerical instability in situations of extreme proximity to borders and vertices. Nonetheless, these limitations are often not very relevant in terms of practical implementation, and the algorithms are actually used on many applications in the software industry. What these algorithms are not aimed at, though, is the analytical evaluation of whether a point lies inside a more complex region bounded by mixed linear and curved segments. Actually, there is no published solution known to the author that indicates how it is possible, with the very same base algorithm and in a straightforward and scalable procedure, to determine analytically, i.e., without linearizing the contour or performing geometric approximations, whether a point is inside or outside regions such as those illustrated in Figure 1.



**Figure 1.** Examples of shapes bounded by linear, circular, or Bézier segments, including twisting and self-intersection, and for which lacks a procedure to determine with the same algorithmic process the relative location of the points illustrated. The case on the right is much more complex than the one on the left, and that can be challenging, even for humans.

## 2. Related Work

The first known reference to an algorithm for the point location problem dates back to 1962 by M. Shimrat [2]. That early algorithm had some limitations pointed out soon after by R. Hacker [3], and later by W. Randolph Franklin, who also states to have delivered, in 1970, a code written in FORTRAN for this same problem [4]. The books by Preparata and Shamos, published in 1985 [5], and by Sedgewick, published in 1990 [6], also covered the point in the polygon problem and have been useful references for many other authors and programmers who have developed specific implementations with adaptations for their own set-ups. Still, nowadays, it is easy to find papers in conferences and journals—in very diverse contexts—where this problem is addressed, even if the novelties have stabilized a long time ago around the two main principles, as described further below.

Eventually, what has become one of the most well-known references in the literature was the publication *Point in Polygon Strategies*, in 1994, by Eric Haines [7], and also some years later, the work of Hormann and Agathos, in 2001 [8], which is the algorithm being used in Matlab for point location in arbitrary polygons.

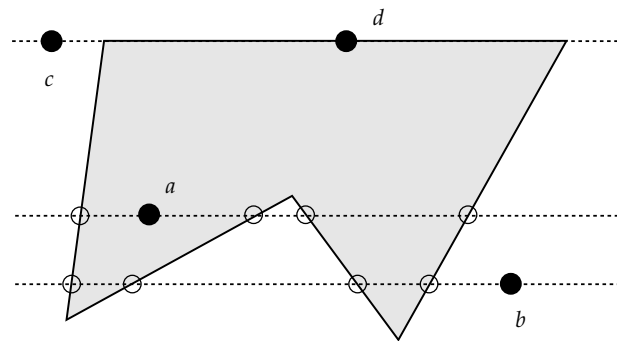
A clarification is recommended here to better focus the scope of the paper. Actually, there are two kinds of challenges in this problem: the “point in region” problem, and the “point location in subdivisions of the plane”. Although related, there are some formal differences in these problems.

In the first problem, there is only one region, and we are testing whether a point lies inside or outside of it. In the second problem, there is a group of regions (typically a subdivision of the plane), and we need to identify which specific region the point belongs to. The first problem is to be solved using containment tests specific to the region (e.g., point-in-polygon), and it has constant-time complexity relative to the number of regions but may depend on the complexity of the region’s boundary. The second problem typically involves building a data structure for efficient spatial querying across many regions, with logarithmic or sublinear complexity relative to the number of regions in the subdivision. The second problem usually involves triangulated regions (using Delaunay triangulation), and several works stand out in that front, ranging from the fundamentals from Voronoi diagrams [9] and Delaunay triangulation, including later optimizations for better data structure management [10], up to actual point location algorithms in such triangular subdivisions of the plane [11,12].

This paper focuses specifically on the first problem and proposes an alternative solution by including additionally the ability to extend the regions to general curve-shaped boundaries. So, in this paper, the expression “point location” refers to assessing whether some point lies inside or outside a given planar region.

### 2.1. Ray-Crossing Approaches

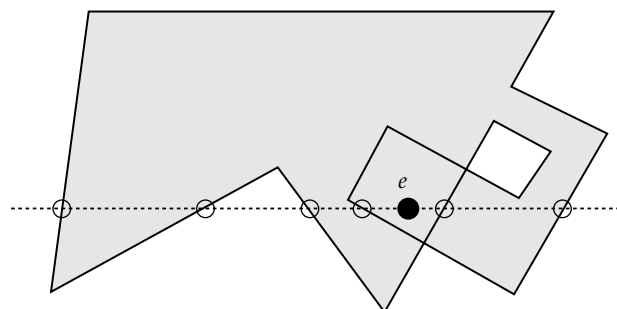
For a long time, one of the most popular algorithms was the one based on the idea of defining an infinite line starting at the point being analyzed: the number of intersections with the polygon boundary gives indications for the solution, as described, for example, by Preparata and Shamos [5]. Briefly stated, a point belongs to a polygon if an infinite straight line starting on it intersects the polygon an odd number of times in one direction (Figure 2). This algorithm is also known as the ray-crossing [13], the crossing number [14] algorithm, and the even-odd rule algorithm [15].



**Figure 2.** The traditional ray-crossing algorithm is based on the parity of intersection points. An odd number of intersections of a horizontal line starting on the point being tested along one of its two sides indicates that the point is inside the polygon, and an even number of intersections (also 0) indicates that the point is outside.

One traditional issue with the ray-crossing algorithm was that it may require further tests to account for the cases where the intersections of the line coincide with one or more vertices or even with an entire edge of the polygon itself. Implementations of this algorithm can be found in many places, and the earlier most well-known implementations occur in the works of Sedgwick [6], Haines [7], and O'Rourke [16].

The ray-crossing algorithm is simple and of limited computational cost,  $\{O(n)\}$ , but, for more functionality, it may require additional steps, hence reducing its elegance. Additionally, applying its technique to generally shaped areas other than polygons, although possible in theory, could pose huge computational and practical difficulties since systems of simultaneous equations might need to be managed and solved. Still, and although it surely has not been intended for that, its applicability for more general self-intersecting (non-simple) domains may fail, as illustrated in Figure 3, where the point  $e$  would be evaluated as being in the “outside”, which contradicts both the mathematical concept based on the winding number [17], and the human perception of “insideness” on a folded (and/or twisted) polygon.

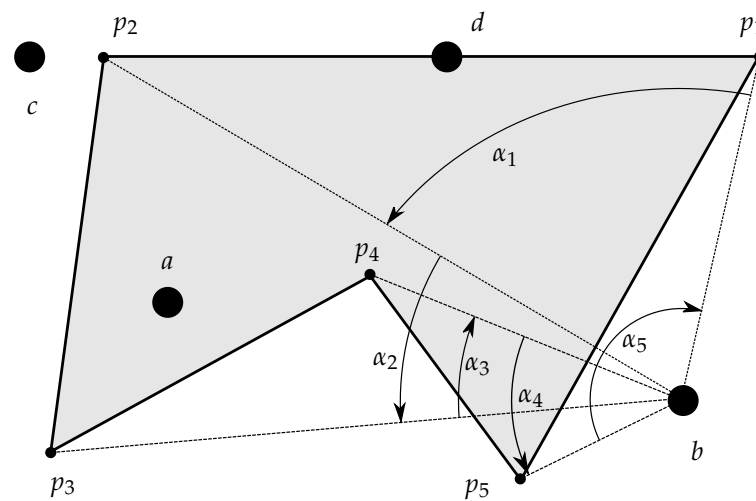


**Figure 3.** The classic ray-crossing algorithm is not meant to deal with non-simple or self-intersecting polygons. Point  $e$  would be detected as “outside”, but it is “inside”.

## 2.2. Winding Number Based Approaches

The problem of non-simple polygons (self-intersecting) has nonetheless been practically solved with algorithms based on the winding number [7], also named nonzero-rule algorithms, which have been refined and even made as efficient as the ray-crossing approach [14]. Its formulation, based on the sum of all the oriented angles with a center in the test point and delimited by each pair of successive vertices of the polygon, allows to obtain the winding number around a point (the number of times a closed curve winds around a given point in the positive sense), and then conclude whether the point is inside or outside the region. If  $\alpha_1, \alpha_2, \dots, \alpha_M$  are the  $M$  oriented angles defined by the testing point and the  $M$  segments defined with the  $M$  vertices of a polygon, the winding number is calculated by  $W_N = \frac{1}{2\pi} \sum_{i=1}^M \alpha_i$ . If  $W_N$  is equal to zero, then the number of winds of the polygon around the point is zero, meaning that the point is “outside” the polygon. Any other value for  $W_N$  will indicate that the point is “inside” the polygon.

Figure 4 illustrates the geometric principle of the procedure for a simple polygon.



**Figure 4.** Algorithm based on the winding number. If  $\sum_i \alpha_i = 0$ , then the testing point ( $b$  in this case) is outside the polygon.

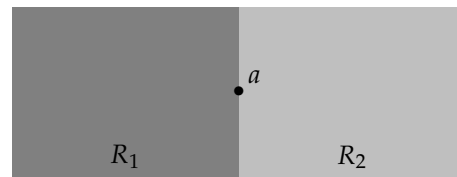
However, the traditional winding number-based algorithms may be ambiguous if the point lies on the boundary and nearly all implementations show some degree of false positives and missed detections, as demonstrated by S. Shirra in [18]. The technique described generically in Figure 4 is often seen in the literature as the calculation of the winding number contribution of each linear segment ( $\overline{p_1 p_2}$ ) relative to a point  $a$ , where all of the contributions are added up. Mathematically, the procedure is simple, as presented, for example in [19], and can be easily understood as the calculation of the winding number contribution of that segment relative to the given test point  $a$ :

$$\begin{cases} \vec{u} = p_1 - a \\ \vec{v} = p_2 - a \\ \alpha = \text{atan2}(\vec{u} \times \vec{v}, \vec{u} \cdot \vec{v}) \end{cases} \quad (1)$$

Demonstration of (1) is straightforward because  $\vec{u} \times \vec{v} = |\vec{u}||\vec{v}| \sin(\alpha)$  and  $\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos(\alpha)$ . In this context  $\vec{u} \times \vec{v}$  is actually supposed to mean the third component of the actual cross product  $(u_x, u_y, 0) \times (v_x, v_y, 0)$ . Possible alternative notations would be to replace  $\vec{u} \times \vec{v}$  in (1) by  $\det \begin{bmatrix} \vec{u}^T & \vec{v}^T & \mathbb{1}^T \end{bmatrix}$ , where  $\mathbb{1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ , or  $(\vec{u} \times \vec{v})_3$  or, of course,  $(u_x v_y - u_y v_x)$ .

When a point falls over the boundary of a region, more than one interpretation may arise. An example of that occurs in Figure 5, where two adjacent regions  $R_1$  and  $R_2$  (delimited by different shades and without the border lines being drawn to visually enhance the effect) seem to share a point  $a$  because it falls precisely on the boundaries of

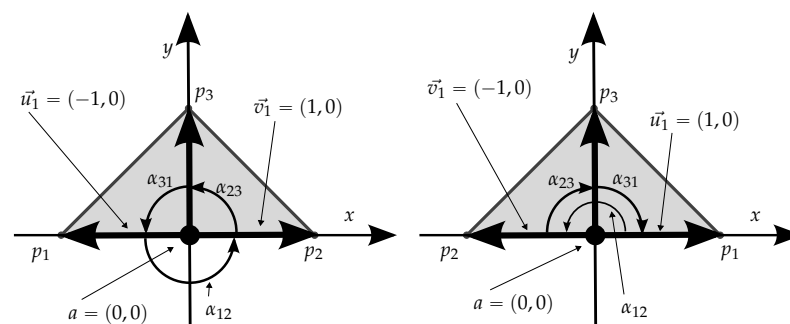
the regions. This raises the question of whether point  $a$  belongs to both regions, to only one of them, or neither of them!



**Figure 5.** Example of two adjacent regions  $R_1$  and  $R_2$  where point  $a$  lies precisely on the border, raising the question to which region it belongs.

The possibility of belonging to neither of the regions must be discarded since  $a$  lies in a area covered by both regions altogether, seeming absurd to affirm that  $a$  does not belong to neither of the regions and at the same time belonging to their union. So, there remains the possibility of belonging to only  $R_1$  or  $R_2$ , or both at the same time. This is where the variation of interpretation can take place. If regions are analyzed independently, and using the winding number approach, any conclusion (belonging or not to  $R_1$  or  $R_2$ ) can be taken depending on the circulation sense of the polygonal lines, as described next. If, on the other hand, the two regions are taken as a degenerate self-intersecting polygon where the common edge is actually a double edge, it will be demonstrated further that  $a$  belongs naturally to the compound region  $(R_1 \cup R_2)$ , independently of the circulation sense.

As mentioned earlier, when the testing point is over the boundary, the winding number approach has a variable behavior, depending on the circulation sense, which can be illustrated with this simple example based on a triangle with vertices  $p_1 = (-1, 0)$ ,  $p_2 = (0, -1)$ ,  $p_3 = (0, 1)$  and a test point  $a = (0, 0)$ . For the triangle longer segment, we have  $\vec{u} = p_1 - (0, 0)$  and  $\vec{v} = p_2 - (0, 0)$ . Also,  $\vec{u} \times \vec{v} = 1 \times 1 \times \sin(\pi) = 0$  and  $\vec{u} \cdot \vec{v} = 1 \times 1 \times \cos(\pi) = -1$ , and consequently  $\alpha = \text{atan2}(0, -1) = \pi$ . If we swap  $p_1$  with  $p_2$ , which is equivalent to reversing the circulation sense, the result is the same for this segment (because it still holds  $\alpha_{12} = +\pi$ ), which indicates that the approach does not distinguish the sense of circulation (winding number contribution) along a segment when the point is over it. But, for the remainder two segments, the winding number contribution is detected accordingly in different values and, therefore, the overall winding number changes! In conclusion, there are cases where the point will be considered inside and others outside the polygon, depending on the circulation sense. Figure 6 shows the elements that provide a simple example demonstrating this situation, which is occasionally mentioned by some authors, but also usually dismissed as irrelevant or avoided with specific argumentation, such as in [4].



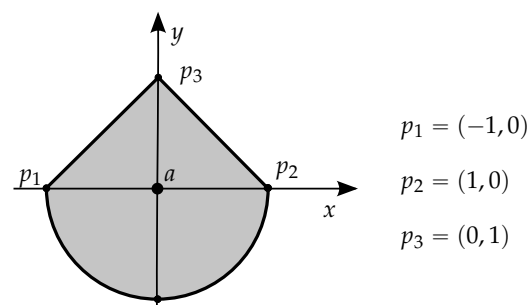
**Figure 6.** Example of the behavior of the winding number technique for a point on the border,  $a = (0, 0)$ , of a triangle when the circulation sense is reversed (the order  $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_1$  is always assumed). For the inclusion of  $a$ , the sum of the three angles is expected to be  $\sum \alpha_{ij} = \pm 2\pi$ , which does not occur on the right (clockwise circulation sense) because  $\alpha_{12} = +\pi$ , causing a net  $\sum \alpha_{ij} = 0$ . For the sake of clarity, auxiliary vectors  $\vec{u}$  and  $\vec{v}$  (from Equation (1)) are shown only for the first oriented segment ( $p_1 \rightarrow p_2$ ).

Although apparently inconvenient and potentially unexplainable (the sense of circulation was not expected to determine the conclusion of the point inclusion test), this behavior is actually predicted by the Jordan curve theorem.

Indeed, an important concept related to the problem of point location is the well-known Jordan curve theorem (JCT) that, in a simple redaction, as is done in [20], states that “Every Jordan curve (a non-self-intersecting continuous loop in the plane) separates the plane into exactly two components”. This means that any point is either in one region or in the other, and implicitly, there is no distinct third region, like, for example, “the border”. So, formally, any results derived from the JCT, namely those based on the winding number, may find ambiguity in assessing which of the two regions is the point lying on the boundary, and this is an unavoidable fact that must be managed when using techniques based on circulation or winding number. Actually, for simple curves, when the observer is traveling along the curve counter-clockwise, the points on this curve (boundary), by convention, will belong to the region on the left hand side; so, the identification of which is the “inside” region depends on the traveling (circulation) sense of the observer, and this is where the ambiguity may take place.

### 2.3. Generalization to Generic Shapes

The generalization of the problem for domains beyond polygons (simple or non-simple) can be carried out by using the winding number approach. However, the literature has not shown a definitive viable implementation to do it, which includes the case of S. Gatilov in [19], who gives a methodology to calculate the winding angle associated with a circular arc. But Gatilov’s approach counts on the winding number technique described in (1), with its ambiguity for points over segments, which in the case of arcs would be points over the chord of the arc. That is not viable since the point will not even be on the boundary, and the calculation may fail depending on the sense of the circulation! Let us consider the example in Figure 7 with  $a = (0, 0)$ . The contribution for the winding number only by the arc  $p_1 \frown p_2$  depends on the circulation sense, as happens with any segment, but in case of this arc, the value is not symmetric: it is  $\pi$  or  $0$ ! Since the contributions of the other two segments are symmetric depending on the circulation sense ( $\pm\pi$ ), this all adds up to ambiguity in detecting whether  $a$  lies inside or outside the region, which is absurd because point  $a$  is clearly inside the region.



**Figure 7.** Case where point  $a$  is without doubt included in the delimited region, including a circular arc, independently of the circulation sense, but which could fail by the application of the traditional winding number technique proposed by [19].

Actually, the problem of point location in regions that may have curve segments was first mentioned, although not actually solved in practice, by Edelsbrunner and Maurer in 1981 [21], and the general problem is seen by Kirkpatrick in 1983 as a challenge that may require a completely new approach:

[...] While the algorithm [proposed by Kirkpatrick] can be adapted to certain other situations (for example, when all internal regions are star shaped), the general problem of optimal search in subdivisions formed from arbitrary curve segments may require a totally new approach. ([22])



The winding number approach has been a well-accepted solution for the problem of point location, and the calculation of the winding number of a polygonal curve  $\Gamma$  around some point  $a$  is easy to calculate by determining the number of intersections of  $\Gamma$  with the real (horizontal) axis, as first pointed out in [23] and later implemented into a specific algorithm by [1]. However, in the literature, nothing specific is added concerning the winding number of other curves for the purpose of point inclusion, although algorithms exist to calculate the winding number contribution of Bézier curves [24].

#### 2.4. Background and Scope of This Paper

This paper demonstrates and gives implementations of a unified approach that covers the traditional solutions based on ray-crossing or winding numbers, solves the cases where those algorithms may find limitations or ambiguities, and extends and demonstrates a practical, viable solution for generic contours that can be expressed in a parametric form, which is its major novelty.

In the early 1990's, information was not so easy to obtain or track, and faced with the need, the author proposed, during his Ph.D. Thesis [25], an approach for the problem of point location in polygons based on the Cauchy and Residue theorems, which is closely related to the winding number approach, although unfamiliar at the time.

The algorithm operates by explicitly evaluating contour integrals in the complex plane of a specially chosen function. The solution, found by accidental simplification (it was not fully demonstrated then), proved to work well with a very simple and elegant mathematical formulation. Only later, when formal full mathematical demonstrations were needed, was the current solution developed, as presented in this paper. Hence, it can be said that the solution came to be after the usage of the Cauchy and Residue theorems, but it later became as it is now in its present state. The next sections describe in detail the foundations of the technique, including algorithmic implementations, followed by illustrative results and conclusions.

Code for this research and demonstrations of the algorithms described in the remainder of this paper are publicly available in a GitHub repository (<https://github.com/vitoruapt/PointInclusion> (accessed on 2 October 2024)).

### 3. Base Approach and Related Theorems

The relative location of a point in a polygon, or actually any other form of closed contour in the plane, can be explored by the results provided by the theorem of Residue and the theorem of Cauchy, which are introduced next.

**Theorem 1 (Residue Theorem).** *If  $\Gamma$  is a closed curve of the plane, and  $f(z)$  is an analytic function in the domain  $R(\Gamma)$  enclosed by  $\Gamma$  (included), except in a finite number of points inside  $R$ , then:*

$$\oint_{\Gamma} f(z) dz = 2\pi j \sum_k r_k \quad (2)$$

being  $r_k$  the residues of  $f(z)$  at the singular points inside  $R$  (being  $j = \sqrt{-1}$ ).

The residue  $r$  of a complex function  $f(z)$  at a point  $a$ , which is a pole of order  $n$  of  $f$ , is calculated by  $r = \frac{1}{(n-1)!} \lim_{z \rightarrow a} \frac{d^{n-1}}{dz^{n-1}} [(z-a)^n f(z)]$ . A complex function is analytic on a region  $R$  if it is complex differentiable at every point in  $R$ . A common alternative designation is “holomorphic function”, usually the preferable variant by some mathematicians.

Another important fact is given by the Cauchy theorem (or Cauchy–Goursat theorem, since Edouard Goursat (1858–1936) proved this theorem without imposing the condition that  $f'(z)$  should be continuous inside  $R(\Gamma)$  and on the boundary, which was an additional condition left by Cauchy [26]) that states the following:

**Theorem 2 (Cauchy–Goursat Theorem).** *If  $R(\Gamma)$  is a simply or multiply connected region whose boundary  $\Gamma$  is sectionally smooth, and if  $f(z)$  is analytic in  $R(\Gamma)$ , then the following result applies:*

$$\oint_{\Gamma} f(z) \, dz = 0. \quad (3)$$

The previous statement is actually the formalization of the common expression, which states that a line integral of any analytic function is independent of the path. So, the advantage here is to use non-analytic functions on the regions of interest. After selecting a function that is non-defined in only one point inside  $R(\Gamma)$ , and whose residue is non-null on that point, by using the Residue and Cauchy theorems, the following reasoning can be engaged:

$$f(z) \text{ is analytic inside } \Gamma \Rightarrow \oint_{\Gamma} f(z) \, dz = 0 \quad (4)$$

or, conversely:

$$\oint_{\Gamma} f(z) \, dz \neq 0 \Rightarrow f(z) \text{ is not analytic inside } \Gamma. \quad (5)$$

Asserting that  $f(z)$  is not analytic inside  $\Gamma$  means that  $f(z)$  has at least one discontinuity inside  $\Gamma$ . Consequently, and being aware that  $a$  is the unique pole (discontinuity) of  $f(z)$  in  $R(\Gamma)$ , i.e., the unique point where it is not defined, a function such as  $f(z) = \frac{1}{z-a}$  allows us to state this final and most valuable conclusion: if the contour integral of  $f(z)$  along a given contour  $\Gamma$  is not null, then its pole lies in the region delimited by that contour, or, in mathematical terms:

$$\left\{ \begin{array}{l} a \text{ is the pole of } f(z) \\ \oint_{\Gamma} f(z) \, dz \neq 0 \end{array} \right. \Leftrightarrow a \in R(\Gamma). \quad (6)$$

It is now necessary to draw some considerations to select an appropriate function and the procedures to apply and take advantage of the previous result. Let a boundary  $\Gamma$  be decomposed in  $M$  parts  $\Omega_k$  with  $1 \leq k \leq M$ . After reminding ourselves that a contour integral is no more than a line integral where the integration path is a closed curve, the following is clear:

$$\oint_{\Gamma} f(z) \, dz = \sum_{i=1}^M \int_{\Omega_i} f(z) \, dz. \quad (7)$$

Although other possibilities exist, we can adopt for  $f(z)$  the one suggested earlier and presented in (8):

$$f(z) = \frac{1}{z-a}. \quad (8)$$

The residue of  $f(z)$  at pole  $a$  has the value 1. The option for  $f(z)$  given in (8) is due to the fact that the integrand function should be simple for concerns of computational cost, and that it must possess a non-null residue, which would otherwise lead to inconclusive results. For example, all functions of the type  $f(z) = \frac{1}{(z-a)^n}$ , with  $n > 1$ , or the type  $f(z) = \frac{z}{(z-a)^n}$ , with  $n > 2$ , have a residue of 0 on point  $a$ , making them useless for this purpose. On the other hand, there are many other possibilities, all with residue 1 at point  $a$ , such as the following:  $f(z) = \frac{z}{(z-a)^2}$ ,  $f(z) = \frac{1}{1-e^{(z-a)}}$  or  $f(z) = \frac{1}{\sin(z-a)}$ . However, in all these (usable) alternatives, there would always be the need to calculate a complex logarithm for the anti-derivative. Therefore, the simplest and most obvious case found was precisely the function in expression (8), but, indeed, the principle is independent of the function, which is, however, useful only if it has poles and non-null residues.



#### 4. Calculation of the Contour Integral

If  $f(z)$  is analytic in any path between  $z_1$  and  $z_2$ , then the fundamental theorem of calculus along curves applies:

$$\int_{\Gamma_{z_1 z_2}} f(z) dz = F(z_2) - F(z_1). \quad (9)$$

Expression (9) states a very well-known and useful technique, where  $F(z)$  is the anti-derivative of  $f(z)$ ; however, it can be directly applied only if  $F(z)$  is continuous along the integration path and  $f(z)$  is defined over the integration path; this later condition may not be satisfied when the point being tested is over the contour. Nonetheless, and based on the definition of the Cauchy Principal Value (CPV), the contour integral can still be calculated even if the function being integrated has a pole on the contour. That can be performed by using expression (10) assuming that the pole is enclosed by a circle of radius  $\delta$ , and the fraction of the path outside that circle is named  $\Gamma(\delta)$ , where the function  $f(z)$  is always integrable independently of how small  $\delta$  becomes [27]:

$$\int_{\Gamma} f(z) dz = \lim_{\delta \rightarrow 0} \int_{\Gamma(\delta)} f(z) dz. \quad (10)$$

Having solved the possible problem of the non-definition of  $f(z)$  in a finite number of points, it only remains the issue of the continuity of  $F(z)$ , which is going to be managed with a methodology derived further.

##### 4.1. The Complex Logarithm

The anti-derivative of the chosen function (8) is the logarithm of a complex number, which, formally, is a multi-valued result. By definition, the logarithm of a complex number  $z$  is the number  $w$  that satisfies the expression  $z = e^w$ . After this definition, it is then clear that the logarithm of  $z$ ,  $\text{Ln}(z)$ , is multi-valued; if  $z = re^{j\theta}$ , using  $j = \sqrt{-1}$ , and being true that  $re^{j\theta} = re^{j(\theta+2k\pi)} = e^{\ln r + j(\theta+2k\pi)}$ , the following arises:

$$\text{Ln}(z) = \ln r + j(\theta + 2k\pi), k = 0, \pm 1, \pm 2, \dots \quad (11)$$

It must be noted that the logarithm of a complex number is here graphed with a capital letter to stress its multi-valued nature. With multi-valued expressions, it is not possible to establish comparisons or other relational operations without precautions. The meaning of  $\text{Ln}(z)$  depends on the branch where the logarithm is being evaluated, that is, the value of  $k$  in (11). However, the logarithm can be restricted to a single branch, called the principal value, where the imaginary part is unique, hence with only one possible argument, called the principal argument (often designated by the mathematical  $\arg()$  function). The principal value is obtained by making  $k = 0$  in (11) resulting in:

$$\ln(z) = \ln r + j \arg(z) = \ln r + j\theta. \quad (12)$$

When restricted to the principal values, logarithms exhibit some particularities like, for example,  $\ln(e^{-j\frac{3}{4}\pi})$  equals  $\frac{3}{4}\pi j$  rather than  $-\frac{5}{4}\pi j$ , and, as well,  $\ln(e^{-j\pi}) = +\pi j$  and not  $-\pi j$ ! As expected, the restriction of the function output to an interval will certainly cause discontinuity of that function and, indeed,  $\ln(z)$  is not continuous for all points of the non-positive real axis because it is clear that:

$$\begin{cases} \lim_{\theta \rightarrow 0^-} \ln e^{j(\pi+\theta)} = \lim_{\theta \rightarrow 0^+} \ln e^{j(\pi-\theta)} = +\pi j \\ \lim_{\theta \rightarrow 0^+} \ln e^{j(\pi+\theta)} = -\pi j \end{cases}. \quad (13)$$

These issues of the principal value and continuity also affect the logarithms of products (or quotients) when expanded to sums (or subtractions) of logarithms; subtraction or addition of complex logarithms may not always give a result with the imaginary part restricted to the principal value and that is why the following definitions are stated for the logarithm of a quotient, or equivalently, subtraction of logarithms:

$$\ln \frac{z_1}{z_2} = \begin{cases} \ln z_1 - \ln z_2 & \Leftarrow -\pi < \arg z_1 - \arg z_2 \leq +\pi \\ \ln z_1 - \ln z_2 - 2\pi j & \Leftarrow \arg z_1 - \arg z_2 > +\pi \\ \ln z_1 - \ln z_2 + 2\pi j & \Leftarrow \arg z_1 - \arg z_2 \leq -\pi \end{cases} \quad (14)$$

and, conversely,

$$\ln z_1 - \ln z_2 = \begin{cases} \ln \frac{z_1}{z_2} & \Leftarrow -\pi < \arg z_1 - \arg z_2 \leq +\pi \\ \ln \frac{z_1}{z_2} + 2\pi j & \Leftarrow \arg z_1 - \arg z_2 > +\pi \\ \ln \frac{z_1}{z_2} - 2\pi j & \Leftarrow \arg z_1 - \arg z_2 \leq -\pi \end{cases} \quad (15)$$

#### 4.2. Line Integral for Linear Segments

Resuming back to the genesis of the algorithm, the essence of the entire procedure is then to evaluate explicitly  $\oint_{\Gamma} \frac{dz}{z-a}$  and verify whether its value is null or not. It will not be null if  $a$  lies within or over the contour  $\Gamma$ . By applying (9) to a segment of a path between points  $z_1$  and  $z_2$ , the following result would potentially occur:

$$\int_{[z_1 z_2]} \frac{dz}{z-a} = \ln(z_2 - a) - \ln(z_1 - a). \quad (16)$$

##### 4.2.1. Concerns When Calculating the Line Integral

Despite the elegant solution given by (16), there remains the uncertainty of its applicability due to the possible non-continuity of  $\ln(z)$  along the path  $[z_1 z_2]$ . That non-continuity will occur precisely when the path from  $z_1$  to  $z_2$  crosses the negative real axis (NRA), that is, in a neighborhood where  $z$  on the path passes from the 2<sup>nd</sup> quadrant (Q2) to the 3<sup>rd</sup> quadrant (Q3) or vice-versa. The following notation is adopted to define the situations of a path crossing the NRA:

$$\begin{aligned} \Downarrow \circ & \text{ path crosses the NRA from Q2 to Q3.} \\ \Uparrow \circ & \text{ path crosses the NRA from Q3 to Q2.} \\ \Updownarrow \circ & \text{ path crosses the NRA in any sense.} \\ \perp \circ & \text{ path touches the NRA from Q2.} \\ \top \circ & \text{ path touches the NRA from Q3.} \\ \times \circ & \text{ path does not cross the NRA.} \end{aligned} \quad (17)$$

So, the problem is then to calculate the line integral in such conditions of discontinuity, which is explained next. The discontinuity of  $\ln(z)$  occurs when  $z$  crosses the negative real axis, but this effect may occur with other functions apart from the logarithm. So, let us assume that some generic function  $f$  can be parametrized on  $\theta$  (to ease the practical calculation, knowing that  $z = re^{j\theta}$ ), and its definite integral is to be evaluated from  $(\pi - \alpha)$  to  $(\pi + \beta)$ , where  $\alpha$  and  $\beta$  are positive angles smaller than  $\pi$  radians, i.e.,  $\alpha, \beta \in ]0, \pi[$ . Being  $\delta$  an infinitesimal positive value, then the following can be written:

$$\int_{\pi-\alpha}^{\pi+\beta} f(\theta) d\theta = \int_{\pi-\alpha}^{\pi-\delta} f(\theta) d\theta + \int_{\pi-\delta}^{\pi+\delta} f(\theta) d\theta + \int_{\pi+\delta}^{\pi+\beta} f(\theta) d\theta. \quad (18)$$

From the three terms on the right side of (18), the first and the last are always defined for any value of  $\delta$  (for the functions involved in this analysis). So, taking the limit when  $\delta \rightarrow 0$  will also force the middle term to be zero because the following holds true for any function  $f(\theta)$ :

$$\lim_{\delta \rightarrow 0} \int_{\pi-\delta}^{\pi+\delta} f(\theta) d\theta = \int_{\pi}^{\pi} f(\theta) d\theta = 0. \quad (19)$$

Therefore, and being  $F(\theta)$  the anti-derivative of  $f(\theta)$ , expression (18) can be developed into (20):

$$\begin{aligned} \int_{\pi-\alpha}^{\pi+\beta} f(\theta) d\theta &= \lim_{\delta \rightarrow 0^+} \left[ F(\theta) \Big|_{\pi-\alpha}^{\pi-\delta} + F(\theta) \Big|_{\pi+\delta}^{\pi+\beta} \right] \\ &= \lim_{\delta \rightarrow 0^+} [F(\pi-\delta) - F(\pi-\alpha) + F(\pi+\beta) - F(\pi+\delta)] \\ &= F(\pi+\beta) - F(\pi-\alpha) + \lim_{\delta \rightarrow 0^+} [F(\pi-\delta) - F(\pi+\delta)]. \end{aligned} \quad (20)$$

Expression (20) shows that, when the integration path crosses the negative horizontal axis, the line integral of a function  $f$ , has an additional term of  $\lim_{\delta \rightarrow 0^+} [F(\pi-\delta) - F(\pi+\delta)]$ , whose concrete value depends obviously on the continuity of  $F(\theta)$ . If  $F(\theta)$  happens to be continuous on the value  $\theta = \pi$ , then that additional term is null, and we fall back to the fundamental theorem (9). Resuming to the case in discussion, using  $F(\theta) = \ln(re^{j\theta})$ , then, by taking into account (13), the following holds:

$$\lim_{\delta \rightarrow 0^+} [\ln(re^{j(\pi-\delta)}) - \ln(re^{j(\pi+\delta)})] = [\ln(r) + \pi j] - [\ln(r) - \pi j] = +2\pi j, \quad (21)$$

which, by recovering the original variable ( $z = re^{j\theta}$ ), yields:

$$\int_{[z_1 z_2]} \frac{dz}{z-a} = \ln(z_2 - a) - \ln(z_1 - a) + 2\pi j, \quad \text{for } \Downarrow \circ. \quad (22)$$

A similar reasoning is easy to replicate for the case where the sense of the path is reversed, e.g., swap the integral limits in (20), that is, when the crossing occurs from 3rd quadrant (Q3) to 2nd quadrant (Q2), and the result for that case is:

$$\int_{[z_1 z_2]} \frac{dz}{z-a} = \ln(z_2 - a) - \ln(z_1 - a) - 2\pi j, \quad \text{for } \Uparrow \circ. \quad (23)$$

In summary, it can be concluded that the line integral for a path between two points  $z_1$  and  $z_2$  of  $\frac{1}{z-a}$  is given by:

$$\int_{[z_1 z_2]} \frac{dz}{z-a} = \begin{cases} \ln(z_2 - a) - \ln(z_1 - a) + 2\pi j & \Leftarrow \Downarrow \circ \\ \ln(z_2 - a) - \ln(z_1 - a) - 2\pi j & \Leftarrow \Uparrow \circ \\ \ln(z_2 - a) - \ln(z_1 - a) & \Leftarrow \times \circ \end{cases} \quad (24)$$

Considering points  $z_1$  and  $z_2$  to be such that

$$\begin{cases} (z_1 - a) \in Q2 & \Rightarrow +\frac{\pi}{2} < \arg(z_1 - a) < \pi \\ (z_2 - a) \in Q3 & \Rightarrow -\pi < \arg(z_2 - a) < -\frac{\pi}{2} \end{cases} \quad (25)$$

makes it simple to assert that, for this case, we have:

$$\lim (\max\{\arg(z_2 - a)\} - \min\{\arg(z_1 - a)\}) = -\pi, \quad (26)$$

and consequently, the following always holds true:

$$\arg(z_2 - a) - \arg(z_1 - a) < -\pi. \quad (27)$$

Similarly, if we have the reverse situation,

$$\begin{cases} (z_1 - a) \in Q3 \\ (z_2 - a) \in Q2 \end{cases}, \quad (28)$$

then, the following can also be demonstrated:

$$\arg(z_2 - a) - \arg(z_1 - a) > +\pi. \quad (29)$$

In summary, when NRA intersection occurs from Q2 to Q3 ( $\rightarrow$ ), we have the first case in (24) which, by using the equality of case 3 from (15), certified by (27), allows to state the following:

$$\begin{aligned} \int_{[z_1 z_2]} \frac{dz}{z - a} &= \ln(z_2 - a) - \ln(z_1 - a) + 2\pi j \\ &= \left( \ln \frac{z_2 - a}{z_1 - a} - 2\pi j \right) + 2\pi j = \ln \frac{z_2 - a}{z_1 - a} \end{aligned} \quad (30)$$

Similarly, when NRA intersection occurs from Q3 to Q2 ( $\leftarrow$ ), we have the second case in (24), which allows us to state the following by using the equality of case 2 from (15), as confirmed by (29):

$$\begin{aligned} \int_{[z_1 z_2]} \frac{dz}{z - a} &= \ln(z_2 - a) - \ln(z_1 - a) - 2\pi j \\ &= \left( \ln \frac{z_2 - a}{z_1 - a} + 2\pi j \right) - 2\pi j = \ln \frac{z_2 - a}{z_1 - a} \end{aligned} \quad (31)$$

In conclusion, and by comparing expressions (24) and (15), the following final result is obtained for any  $a \neq z_1, z_2$ :

$$\int_{[z_1 z_2]} \frac{dz}{z - a} = \ln \frac{z_2 - a}{z_1 - a}. \quad (32)$$

A non-null value for expression (32) as the indicator of point inclusion in polygons is precisely the methodology firstly used by the author in [25] (pp. 149–152), in the early 1990's, as mentioned before, and whose validity has just been formally demonstrated.

#### 4.2.2. Simplifying the Calculation of the Contour Integral

The result expressed by (32) allows to assert the full methodology because it calculates, directly and in a straightforward mode, the circulation of a path made up of linear segments. In the particular case of  $\Gamma$  defined as  $M$  linear segments, and with  $f(z) = \frac{1}{z-a}$ , expression (7) with the results of expression (32) yields the following generic result:

$$\oint_{\Gamma} f(z) dz = \ln \frac{z_2 - a}{z_1 - a} + \ln \frac{z_3 - a}{z_2 - a} + \cdots + \ln \frac{z_M - a}{z_{M-1} - a} + \ln \frac{z_1 - a}{z_M - a}$$

$$= \sum_{n=1}^M \ln \frac{z_{(n \bmod M)+1} - a}{z_n - a}. \quad (33)$$

Furthermore, to simplify the calculation and avoid using at all the complex logarithms, the previous calculations can be even more simplified since the possible variations of the contour calculations only occur in the imaginary part, as it is intuitive from previous statements, and easily demonstrable by expression (34). Indeed, being  $z_k = r_k e^{j\theta_k}$  and  $z_m = r_m e^{j\theta_m}$  and therefore  $\Re\left\{\ln \frac{z_k}{z_m}\right\} = \Re\left\{\ln \frac{r_k}{r_m} + j \arg \frac{z_k}{z_m}\right\} = \ln \frac{r_k}{r_m}$ , the calculation of the real part of (33) for the polygon with  $M$  vertices is always null, as shown next:

$$\Re\left\{\oint_{\Gamma} f(z) dz\right\} = \Re\left\{\ln \frac{z_2}{z_1} + \cdots + \ln \frac{z_M}{z_{M-1}} + \ln \frac{z_1}{z_M}\right\}$$

$$= \ln \frac{r_2}{r_1} + \cdots + \ln \frac{r_M}{r_{M-1}} + \ln \frac{r_1}{r_M}$$

$$= \ln\left(\frac{r_2}{r_1} \frac{r_3}{r_2} \cdots \frac{r_M}{r_{M-1}} \frac{r_1}{r_M}\right) = \ln(1) = 0. \quad (34)$$

Finally, it may then be stated that the circulation can be obtained by simply calculating  $\arg()$  operations using (35):

$$\oint_{\Gamma} f(z) dz = \sum_{n=1}^M \arg \frac{z_{(n \bmod M)+1} - a}{z_n - a}. \quad (35)$$

Reminding the definition in (14), and being  $z_1 = r_1 e^{j\theta_1}$  and  $z_2 = r_2 e^{j\theta_2}$ , the definition of the  $\arg()$  function for a quotient of two complex numbers is, of course, given by (36):

$$\arg \frac{z_1}{z_2} = \begin{cases} \theta_1 - \theta_2 & \Leftarrow -\pi < \theta_1 - \theta_2 \leq +\pi \\ \theta_1 - \theta_2 - 2\pi & \Leftarrow \theta_1 - \theta_2 > +\pi. \\ \theta_1 - \theta_2 + 2\pi & \Leftarrow \theta_1 - \theta_2 \leq -\pi \end{cases} \quad (36)$$

Resorting to the actual geometric problem, and denoting the testing point by  $a = (x_a, y_a)$ , and a generic point to delimit segments by  $P_n = (x_n, y_n)$ , it is immediate to state that

$$\theta_n = \arg(P_n - a) = \text{atan2}(y_n - a_y, x_n - a_x) \quad (37)$$

and, hence, Algorithm 1 on the following page can be promptly established. Moreover, since the circulation of  $f(z)$  always has a null real part (34), only the imaginary part is relevant and, therefore, the variant for the calculation given by (35) is preferred (simpler) than the calculation using expression (33).

In conclusion, in a closed contour, the neat value of (35) will be null if, along the path, there are no transitions between quadrants Q2 and Q3, but if they occur, additional terms of  $\pm 2\pi j$  must be added at each time, as expressed by (36). So, the following proposition can be asserted:

**Theorem 3 (NRA intersection theorem).** *In a region enclosing a singularity of a function  $f(z)$ , the value of the contour integral (hence, the point inclusion) depends exclusively on the number and sense of path transitions over the negative real axis (NRA).*

This section has demonstrated how the contour integral is indeed the translation of the winding number approach introduced earlier as one of the well-known techniques for point location in polygons. Moreover, as stated in the NRA intersection theorem, it is also clear that the issues of the winding number calculation (angle contributions) occur only when the path crosses the NRA, which resembles very much the ray crossing method.

---

**Algorithm 1:** Polygon inclusion test using real arctan()
 

---

**Require:**  $a = (a_x, a_y)$ ,  $P_n = (x_n, y_n)$  for  $n = 1, 2, \dots, M$

```

1: if  $a \in \{P_1, P_2, \dots, P_n\}$  then
2:   return true // point  $a$  is one of the vertices
3: end if
4: for  $n = 1$  to  $M$  do
5:    $\theta_n \leftarrow \text{atan2}(y_n - a_y, x_n - a_x)$ 
6: end for
7:  $S \leftarrow 0$ 
8: for  $n = 1$  to  $M$  do
9:    $\theta \leftarrow \theta_{(n \bmod M)+1} - \theta_n$ 
10:  if  $\theta > \pi$  then
11:     $\theta \leftarrow \theta - 2\pi$ 
12:  else if  $\theta \leq -\pi$  then
13:     $\theta \leftarrow \theta + 2\pi$ 
14:  end if
15:   $S \leftarrow S + \theta$ 
16: end for
17: if  $S \neq 0$  then
18:  return true // point  $a$  is included
19: else
20:  return false // point  $a$  is not included
21: end if

```

---

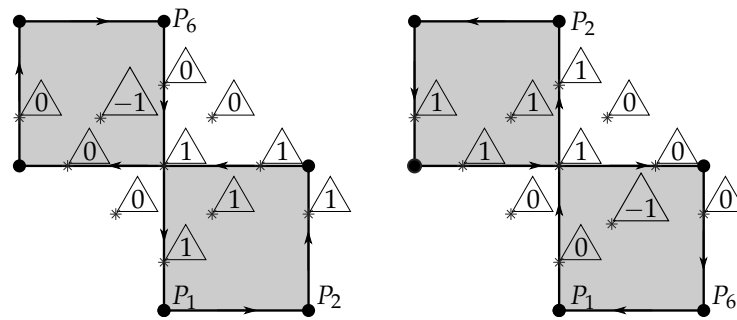
#### 4.2.3. Points on the Boundary of Polygons with Undefined Orientation

Algorithm 1 is valid for all combinations of point  $a$  and polygon  $\{P_1, P_2, \dots, P_M\}$ , and requires no post-processing; however, there can be ambiguity in a special situation described next, which is nevertheless easily solvable with a more general approach.

The special situation just mentioned concerns the points over the boundary when the orientation of the polygon (coarsely, the order in which the vertices are browsed) is not the direct orientation, which may give ambiguous results, as expected by the Jordan curve theorem, and mentioned at the beginning of this paper. Indeed, the contour integral  $\oint_{\Gamma} \frac{dz}{z-a}$ , when  $a$  is over the path  $\Gamma$ , can result either in 0 or  $2\pi j$  or, equivalently, a winding number of 0 or 1. Additionally, if the polygon is not simple (i.e., is self-intersecting), the polygon orientation is not adjustable (by reversing the order of vertices, for example) because the orientation changes locally after each self-intersection of the boundary line! For those cases, the contour integral is nonetheless always defined, and Figure 8 shows several examples of points near a self-intersecting polygon.

In Figure 8, several observations immediately arise: the first is that clear “outside” and “inside” points are always well determined (winding number of zero or non-zero, respectively). A second observation is that points on the boundary may yield a winding number of 0 or 1, depending on the circulation sense.





**Figure 8.** Several examples of winding numbers for testing points (\* in the image) in different situations on a non-simple (self-intersecting) polygon. On the left, the direct sense of circulation was used when starting on  $P_1$  and moving toward  $P_2$ , and on the right the reverse sense was used (also starting on  $P_1$  but toward a new  $P_2$ ). Some points over the boundary result in an ambiguous winding number that depends on the circulation sense.

There is also a particular case of a point being twice on the boundary (in the intersections), which is always considered “inside”, no matter the sense of circulation. The mentioned ambiguity is however not unsolvable since it is detectable: simply, in case of doubt about the sense of the polygon orientation, to cover even for points over the boundary, a double calculation can be performed: one with the given vertex point order, and the other with its reverse sense; in case one of the calculations yields zero and the other non-zero, then it is the case of a point over the boundary, and further decision can be taken (normally to be considered included in the polygon). This avoids the need of special analysis to detect points on the boundary, which could be absolutely not recommend on general shaped contours, as managed ahead in this paper.

The challenge of points over the boundary is recurrent in all algorithms in the literature. Even one of the most common approaches in the state of the art (the already cited work of Hormann and Agathos [8] used in Matlab) has a special treatment for the points on the boundary that the authors call *boundary version* and is included in Algorithm 7 in their paper, and deals efficiently with points on boundaries of polygons.

In conclusion, if there can be points anywhere on the boundary that need to be precisely assessed, and there is no knowledge whether a sequence of points  $\{P_1, P_2, \dots, P_M\}$  of a simple polygon is not given in the positive circulation sense, or whether the polygon is not simple, then Algorithm 1 must be called twice, being the second time done with the polygon vertices in the reverse order. If the two calls of the algorithm yield different results, then we are in the presence of a point over the boundary, and it is (usually) to be considered inside the polygon (Algorithm 2 on the following page).

Algorithm 2 on the next page requires double the computation resources since it calls the actual calculation algorithm twice. This is required only for the absolutely general case of totally random polygons and testing points, although total randomness will hardly generate situations of points exactly over the border. Anyway, alternatives will be presented ahead.

Despite being effective and elegant, in practice, Algorithm 1 is computationally demanding because one trigonometric function (arctan) is used per each side of the polygon, making it less efficient computationally than other alternatives. That, too, is a reason for a change in the paradigm of the calculation, which will align with the existing state of the art algorithms for polygons but also go beyond their performance, not only by outperforming the computational cost but also going where they do not go actually, that is, extension for shapes other than polygons with linear segments. That paradigm is the parametric definition of the integration path.

**Algorithm 2:** Unambiguous test for generalized polygons**Require:**  $a = (a_x, a_y)$ ,  $P_n = (x_n, y_n)$  for  $n = 1, 2, \dots, M$ 

- 1: Create  $R_m = (x_m, y_m)$  for  $m = M, M-1, \dots, 1$
- 2:  $V_D \leftarrow \text{call Algorithm1}(P_n, a)$
- 3:  $V_R \leftarrow \text{call Algorithm1}(R_m, a)$
- 4: **if**  $V_D \neq V_R$  **then**
- 5:   **return true** // point  $a$  is over the boundary
- 6: **else**
- 7:   **return**  $V_D$  // True/false if  $a$  is inside/outside
- 8: **end if**

**4.3. Parametric Definition of the Integration Path**

An observation that can be made about the technique described in the previous sections is that calculating the line integral for a function with a non-continuous anti-derivative requires care because of the sense of the integration path crossing the NRA, which results in different contributions, as seen. Anyway, for linear segments, the method just described circumvents that issue; however, for non-straight paths to be discussed further, that may not be so clear. Hence, let us adopt an alternative approach to obtain (32) by using parametric integration paths because that explicitly defines a path from a start to an end, along with the variation of the parameter that describes the curve. Let us use a parameter  $t$  that varies from 0 to 1 to cover the full path segment, and let us start with the case of linear segments. Being  $z_1$  and  $z_2$  the extremities of a linear path, any point of the path from  $z_1$  to  $z_2$  is given by:

$$z = z_1 + t(z_2 - z_1), 0 \leq t \leq 1. \quad (38)$$

After adjusting the integration variable,  $dz = (z_2 - z_1)dt$ , the line integral can now be calculated as follows:

$$\int_{[z_1 z_2]} \frac{dz}{z - a} = \int_0^1 \frac{(z_2 - z_1) dt}{z_1 + t(z_2 - z_1) - a} \quad (39)$$

that results in:

$$\left[ \ln(z_1 + t(z_2 - z_1) - a) \right]_0^1 = \ln(z_2 - a) - \ln(z_1 - a), \quad (40)$$

which is the same as (16), but shares the same risks of discontinuity because the anti-derivative is still a logarithm, despite the fact that the integration variable is now  $t$ . Therefore, the solution is exactly the same as before, given in (22) and (23). In the case of NRA crossing, and depending on the sense ( $\dashv\circ$  or  $\dashv\circ$ ), there will be a contribution of  $\pm j2\pi$  to the contour integral, or  $\pm 1$  in the NRA crossings counter.

Although it could have been stated and adopted earlier, working for a segment  $\overline{z_1 z_2}$  in respect to a point  $a$  is actually equivalent to working with segment  $\overline{(z_1 - a)(z_2 - a)}$  in respect to  $a = 0$ . So, from now onwards, when  $z$  points are used, it is assumed that they result from the original points subtracted from  $a$ , i.e.,  $(z - a)$ , and that the reference point in analysis is the system origin  $(0, 0)$ .

**5. NRA Crossing in Parametric Paths**

We can now settle the procedures to calculate the contour integral along a closed path or, equivalently, the inclusion of a point by a region based solely on the intersection of that path with the NRA.

### 5.1. NRA Crossings and Their Sense for Linear Segments

What we seek to detect is whether there is an intersection and what is its sense. And this is straightforward to calculate after (38), which can be expanded into:

$$\begin{cases} \Re\{z\} = x = x_1 + t(x_2 - x_1) & \Leftarrow 0 \leq t \leq 1 \\ \Im\{z\} = y = y_1 + t(y_2 - y_1) & \Leftarrow 0 \leq t \leq 1 \end{cases} \quad (41)$$

A NRA intersection occurs at some point  $z_c$  when  $\Im\{z_c\} = 0$  and  $\Re\{z_c\} \leq 0$  in  $z_c = z_1 + t_c(z_2 - z_1)$ ,  $0 \leq t_c \leq 1$ , i.e., intersection occurs for the value  $t_c$  that verifies:

$$\begin{cases} y_1 + t_c(y_2 - y_1) = 0 \\ x_1 + t_c(x_2 - x_1) \leq 0 \end{cases} \Leftrightarrow \begin{cases} t_c = \frac{y_1}{y_1 - y_2} \\ t_c(x_1 - x_2) \geq x_1 \end{cases} \quad (42)$$

Notice that the case  $y_1 = y_2$  represents a horizontal segment, and “intersection” only occurs for  $y_1 = y_2 = 0$ . However, that is not actually an intersection of the NRA, as explained later, and this special situation is to be discarded early in the algorithms (also to avoid the division by 0). Apart from that case, if a valid  $t_c$  is found, then it is necessary to determine the sense of the NRA crossing, and that is easily verifiable by the derivative of the imaginary component  $\Im\{z\}$  in relation to  $t$ . If the derivative is positive for that value of  $t_c$ , it means that the  $y = \Im\{z\}$  component is increasing with  $t$ , therefore the path segment is passing from Q3 to Q2, or  $\nearrow$ , that is, there is a contribution of  $-1$  to the intersection counting ( $I_C$ ), or  $+1$  for the reverse case (Q2 to Q3, or  $\searrow$ ). This can be formalized as:

$$I_C = -\operatorname{sgn}\left(\frac{dy}{dt}\bigg|_{t=t_c}\right). \quad (43)$$

For linear segments, the previous expression is constant for any point of the segment where the NRA crossing might occur:

$$I_C = -\operatorname{sgn}(y_2 - y_1) = \operatorname{sgn}(y_1 - y_2). \quad (44)$$

If, by chance, the testing point happens to be an intersection point of the polygon boundary with the NRA, either along the path segment ( $t_c \neq 0$  and  $t_c \neq 1$ ) or strictly on a vertex ( $t_c$  is 0 or 1), that can be promptly detected because its real coordinate would be zero:  $x_1 + t_c(x_2 - x_1) = 0$ . This indicates that the point is over the boundary, and a decision can be made immediately without further calculations. Formally: if  $x_1 = t_c(x_1 - x_2) = t_c\Delta x$ , then the point is on the boundary, and can be dispatched earlier in the algorithm.

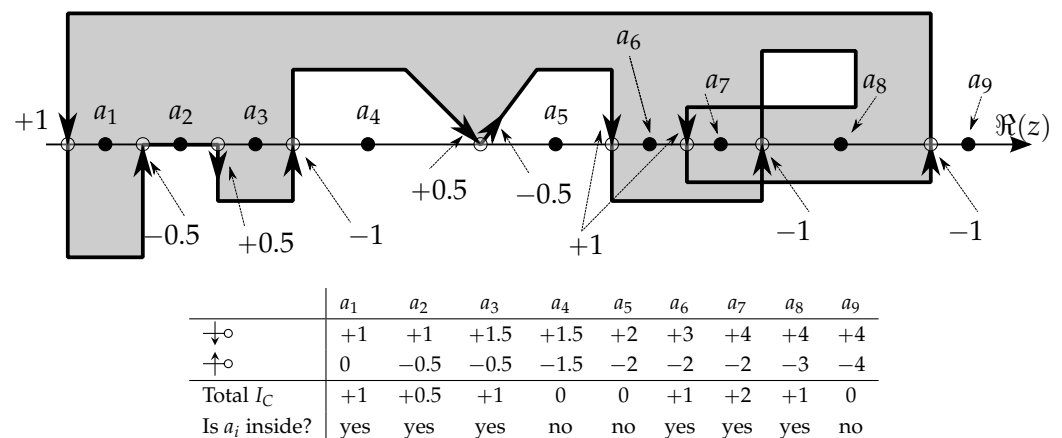
There are also the particular cases of  $t_c = 0$  or  $t_c = 1$ ; these represent the cases where the segment starts or ends exactly on the NRA ( $\nearrow$  or  $\searrow$ ). In those cases, this means that another segment starts or ends there too! To leave the contributions separate, but allowing them to add up or cancel, for those cases, the intersection contribution calculated above should be halved, that is, a  $\pm 0.5$  contribution for segments that originate or end on the NRA is to be applied. In other words, each crossing of the NRA can be accounted with  $\pm 1$ , but reaching or leaving it can be accounted as a half-crossing, or  $\pm 0.5$ . Formally, this can be stated as in (45):

$$t_c = 0 \vee t_c = 1 \Rightarrow I_C = -0.5 \operatorname{sgn}\left(\frac{dy}{dt}\bigg|_{t=t_c}\right). \quad (45)$$

Figure 9 illustrates the situation of  $I_C$  values for several testing points ( $a_1, a_2, \dots, a_9$ ) in a polygonal self-intersecting region; the associated table details the partial and total values of NRA intersection counting. At each case (point  $a_i$ ), the imaginary axis would be a vertical line that crosses the enclosed domain exactly at each of the  $a_i$  points.

In summary, if a path does not intersect the NRA ( $\times$ ), the calculation of the line integral is simple (actually, not necessary in practice) because there are no discontinuities in the process and, in the end, all terms of the closed path will cancel altogether; only the intersections of the path with the NRA will affect contour evaluation or the winding

number. Also, notice that horizontal segments ( $y_1 = y_2$ ) never intersect NRA; they may even lie over it but never actually cross it, so their contribution to the  $I_C$  is null, as can be seen in the segment that contains the  $a_2$  point in Figure 9.



**Figure 9.** Contributions of NRA intersections/touchings on the left side of nine example test points. The sense of NRA reaching/crossing is relevant for the final sum. Since the imaginary axis would lie over each  $a_i$  point, the numbers in the table indicate the  $I_C$  partial and total values, i.e., the counting of intersections (crossings) of the NRA. For example, the total down  $I_C$  for  $a_6$  is given by summing all downward NRA intersections at its left:  $+3 = +1 + 0.5 + 0.5 + 1$  and, similarly, for the total up  $I_C$ , the count is  $-2 = -0.5 - 1 - 0.5$ . The net total is  $I_C = +1$ , hence  $a_6$  is inside. Curiously, the result for  $a_7$  is  $I_C = +2$ , that is, we could consider that  $a_7$  is inside “twice”, which is graspable from the figure!

The major novelties in this approach for polygonal regions are that the existence of NRA crossing can be determined by detecting the value of a single parameter  $t$  and the sense of crossing can be determined by a simple comparison operation.

### 5.2. The Case of Points on the Border, Again!

In Figure 9 specific situations arise if the testing point is exactly a NRA crossing point or, which is to say, the testing point is over the border. Once again, this situation forbids the possibility of seamlessly applying a unique algorithm for points on the border. For example, still in Figure 9, if the testing point occurs on the first crossing point (the one on the left of  $a_1$ ), since it has no other crossing on its left, this implies being outside the region; so, its own intersection contribution ( $I_C$ ) should be counted to consider it inside. But, if this reasoning is applied to the 4<sup>th</sup> intersection (between  $a_3$  and  $a_4$ ), then the overall contribution at that point ( $+1 - 0.5 + 0.5 - 1$ ) would result in 0 and that point would be considered outside the region, contrarily to the intention of considering it as inside. The conclusion is that there is no unique solution to distinguish with the same operation whether a point on the border is always considered inside or outside the region: this is unavoidable, as supported by the Jordan curve theorem presented earlier, where the sense of crossing (circulation) affects the winding contribution, and therefore the conclusion about inclusion. The solution, as already mentioned, is to check those situations before further calculation.

### 5.3. Universal Algorithm for Arbitrary Polygons

All of the previous procedures are integrated in Algorithm 3 on the following page, which shows all the steps to test polygonal inclusion of any point  $a$  on any polygon with  $M$  vertices (for any  $M > 0$ , but only  $M > 2$  is really meaningful), normal or self-intersecting.

The computational cost of Algorithm 3 is  $\mathcal{O}(n)$ , and the arithmetic operations involved are only sums (or subtractions) and at most  $M$  floating point divisions per polygon, although optimizable, as described further.

**Algorithm 3:** Universal inclusion test in arbitrary polygons

---

**Require:**  $a = (a_x, a_y)$ ,  $P_n = (x_n, y_n)$  for  $n = 1, 2, \dots, M$

- 1:  $P_n \leftarrow P_n - a$  for  $n = 1, 2, \dots, M$
- 2:  $I_C \leftarrow 0$  //NRA Intersection counter
- 3: **for**  $n = 1$  **to**  $M$  **do**
- 4:  $Y_1 \leftarrow y_n$ ;  $Y_2 \leftarrow y_{(n \bmod M)+1}$
- 5:  $X_1 \leftarrow x_n$ ;  $X_2 \leftarrow x_{(n \bmod M)+1}$
- 6: **if**  $\text{sgn}(Y_1) == \text{sgn}(Y_2)$  **then**
- 7: **continue** //no NRA crossing: horizontal segment or segment entirely in one quadrant.  
Jump to next segment.
- 8: **end if**
- 9:  $\Delta Y \leftarrow Y_1 - Y_2$
- 10:  $\Delta X \leftarrow X_1 - X_2$
- 11:  $t_c \leftarrow \frac{Y_1}{\Delta Y}$
- 12: **if**  $t_c \Delta X == X_1$  **then**
- 13: **return true** //point  $a$  is on the boundary (included)
- 14: **end if**
- 15: **if**  $t_c \geq 0$  **and**  $t_c \leq 1$  **and**  $t_c \Delta X \geq X_1$  **then**
- 16:  $I_P \leftarrow \text{sgn}(\Delta Y)$
- 17: **if**  $t_c == 0$  **or**  $t_c == 1$  **then**
- 18:  $I_P \leftarrow I_P \times 0.5$  //touch NRA only
- 19: **end if**
- 20:  $I_C \leftarrow I_C + I_P$
- 21: **end if**
- 22: **end for**
- 23: **if**  $I_C \neq 0$  **then**
- 24: **return true** //point  $a$  is included
- 25: **else**
- 26: **return false** //point  $a$  is not included
- 27: **end if**

---

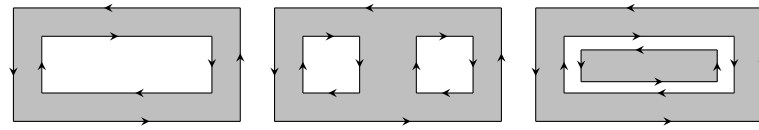
In Algorithm 3, the points on the border, including vertices, are considered inside the polygon, but, if wanted otherwise, this can be easily modified to exclude them. It can be done either with an optional initial checking for  $a$  being one of the vertices or during the normal flow of the algorithm after test from line 12 that results from the NRA crossing conditions in expression (42). The decision on line 13 of Algorithm 3 could be set to **false** if the point on the boundary is defined as being outside.

Notice also that the apparent risk of division by zero in line 11 of the algorithm never occurs because of early analysis in line 6 of the algorithm; indeed, if  $Y_1 == Y_2$ , we also have  $\text{sgn}(Y_1) == \text{sgn}(Y_2)$  and the algorithm skips to the next segment since there is no contribution of the current linear segment to the value of  $I_C$ . Illustration of the functionality and results of Algorithm 3 can be found in the GitHub repository indicated earlier.

#### 5.4. Multi-Ring Polygons and Multi-Polygons

There is a category of planar domains commonly used in Geographical Information Systems (GIS), namely to represent contours of countries and other geographical entities, that accounts both for multiple separate polygons (multi-polygons) or for polygons that include “holes” defined by one (or more) separate closed lines fully included in the outer polygon; each of these separate contours is named a ring. A simple polygon has one ring, and a polygon with two “holes” has three rings [28]. There are also strict cases of multi-ring

multi-polygons where rings nest inside each other. All these situations can be handled with the same concepts described earlier using the sense or circulation of a contour. Figure 10 illustrates three situations of multi-ring polygons.



**Figure 10.** Multi-ring polygons as used in GIS contexts. The orientations are indicative, but appropriate relative senses must be ensured to correctly represent, for example, a polygon with a “hole”.

The figure also illustrates the need for defining correctly the sequence of the polygon vertices in order to apply the same concepts of winding sense and, therefore, obtain the number of NRA intersections. Extension of the algorithm to this type of region is achieved by simply applying the procedure on each of the 2, 3, or more contours (rings) and sum up the separate results of NRA intersections.

### 5.5. Optimizing Algorithm 3

Algorithm 3 on the preceding page can be computationally alleviated by avoiding unnecessary divisions in case the NRA intersection does not occur. Indeed, as required by expression (42), for NRA intersection to occur we must have the following:  $0 \leq \frac{y_1}{y_1 - y_2} \leq 1$ . In other words, if either  $\frac{y_1}{y_1 - y_2} > 1$  or  $\frac{y_1}{y_1 - y_2} < 0$ , then the NRA intersection does not occur, and no more calculations need to be performed. So, a few comparisons (3 at most) can be made before asserting the need to perform the division to find the actual  $t_c$  of NRA intersection. Since  $y_1 - y_2 = \Delta y$ , consider the following:

$$\frac{y_1}{\Delta y} < 0 \Rightarrow (\Delta y > 0 \wedge y_1 < 0) \vee (\Delta y < 0 \wedge y_1 > 0) \quad (46)$$

$$\frac{y_1}{\Delta y} > 1 \Rightarrow (\Delta y > 0 \wedge y_2 > 0) \vee (\Delta y < 0 \wedge y_2 < 0), \quad (47)$$

which can be combined. We are allowed to state that there is no NRA intersection if the following occurs:

$$\begin{cases} \Delta y > 0 \wedge (y_1 < 0 \vee y_2 > 0) \\ \Delta y < 0 \wedge (y_2 < 0 \vee y_1 > 0) \end{cases} \Rightarrow \times_o. \quad (48)$$

Hence, just before line 11 of Algorithm 3 a few lines could be added to optimize the procedure, demonstrating also that at most 3 comparisons are needed to conclude whether there is NRA intersection or not before calculating  $t_c = Y_1/\Delta Y$ , as proposed next:

```

⋮
if  $\Delta Y > 0$  then
  if  $Y_1 < 0$  or  $Y_2 > 0$  then
    continue //no NRA intersection
  end if
else
  if  $Y_2 < 0$  or  $Y_1 > 0$  then
    continue //no NRA intersection
  end if
end if
11:  $t_c \leftarrow \frac{Y_1}{\Delta Y}$ 
⋮

```



Previous expressions can be further optimized because when they are reached, it is already known that  $\text{sgn}(Y_1) \neq \text{sgn}(Y_2)$ . Therefore,  $Y_1 < 0 \vee Y_2 > 0$  is equivalent to only one of them, e.g.,  $Y_1 < 0$ . A similar reasoning applies to the “else”. This further reduces the number of comparisons from 3 to 2.

In case there is intersection (i.e., the previous tests concluded that there may be a NRA intersection because  $0 \leq t_c \leq 1$ ), it is then necessary to calculate its precise value to test the second requirement of expression (42), that is, check the condition  $t_c(x_1 - x_2) \geq x_1$ , which at first glance seems unavoidable to calculate through the division. But, actually, that division operation can be avoided and replaced by two multiplications and a comparison, as is shown next: being  $x_1 - x_2 = \Delta x$  and  $t_c \Delta x \geq x_1 \Leftrightarrow \frac{y_1}{\Delta y} \Delta x \geq x_1$ , if  $\Delta y > 0$  then, for NRA crossing, we need to have:  $y_1 \Delta x \geq x_1 \Delta y$ , otherwise, we need to have the following:  $y_1 \Delta x \leq x_1 \Delta y$ .

The conclusion is summarized by expression (49):

$$\begin{cases} \Delta y > 0 \wedge y_1 \Delta x \geq x_1 \Delta y \\ \Delta y \leq 0 \wedge y_1 \Delta x < x_1 \Delta y \end{cases} \Rightarrow \nexists^\circ. \quad (49)$$

This last simplification not only avoids the mathematical division but also allows the algorithm to fully operate in integer arithmetic representation, which would improve its usage in an approach based purely on the integer representation of points/vertices.

Yet, in line 17, Algorithm 3 still requires to check whether  $t_c$  is equal to 0 or to 1 to conclude if the “crossing” is actually a “touching”, which would imply an adjustment on the intersection counter. But that checking turns out redundant because those cases correspond to  $y_1 = 0$  or  $y_2 = 0$ , that is, there is no need to calculate  $t_c$  for those cases either.

In summary, the algorithm in its full extension and covering all these optimizations, is detailed as Algorithm 4 on the next page. It can be noted, in line 32, that a multiplication by 0.5 is present, which, although being a statistically rare situation, could be modified for further optimization; that would be to count each full crossing as an integer of value  $\pm 2$ , by, for example, changing line 30 into  $I_P \leftarrow \text{sgn}(\Delta Y) + \text{sgn}(\Delta Y)$ , and a half crossing being half of that value ( $\pm 1$ ) with the final conclusions still holding, and integer computations could be present throughout the entire algorithm.

In conclusion, besides the simplicity of the formulation, the algorithm has also the virtue of a straightforward operation with degeneracy situations, such as point over segments or over vertices, or even as null length segments, which could happen in situations of rounding the point representation to integers.

### 5.6. Vectorization of Algorithm 4

Although not necessarily a breakthrough in reducing computational costs, Algorithm 4 on the following page can be easily vectorized for parallel computing, at least for the test of the inclusion of one point. Follows in Figure 11 a fully operational excerpt in Matlab code (listing) that accepts a matrix  $P$  with the polygon vertices (with the last vertex replicated from the first) and a point  $A$ . The procedure is vector based and operates with an entire polygon “at once”.

Despite the possibility of some finer tuning, the vectorized approach, nevertheless, forces all the operations to be performed through the entire chain of tests. Some tests could be dismissed early in a sequential approach, but the vector approach forces all the operations for all cases. Hence, it may not be suited for better performance in all computational setups.

**Algorithm 4:** An optimized version of Algorithm 3**Require:**  $a = (a_x, a_y)$ ,  $P_n = (x_n, y_n)$  for  $n = 1, 2, \dots, M$ 

```

1:  $P_n \leftarrow P_n - a$  for  $n = 1, 2, \dots, M$ 
2:  $I_C \leftarrow 0$  // initialize NRA Intersection counter
3: for  $n = 1$  to  $M$  do
4:    $Y_1 \leftarrow y_n$ ;  $Y_2 \leftarrow y_{(n \bmod M)+1}$ 
5:    $X_1 \leftarrow x_n$ ;  $X_2 \leftarrow x_{(n \bmod M)+1}$ 
6:   if  $\text{sgn}(Y_1) == \text{sgn}(Y_2)$  then
7:     continue // no NRA crossing: horizontal line, etc.
8:   end if
9:    $\Delta Y \leftarrow Y_1 - Y_2$ ,  $\Delta X \leftarrow X_1 - X_2$ 
10:  if  $\Delta Y > 0$  then
11:    if  $Y_1 < 0$  then
12:      // or  $Y_2 > 0$  redundant
13:      continue // no NRA intersection
14:    end if
15:  else
16:    if  $Y_2 < 0$  then
17:      // or  $Y_1 > 0$  redundant
18:      continue // no NRA intersection
19:    end if
20:  end if
21:  if  $\Delta Y \geq 0$  then
22:    if  $Y_1 \times \Delta X < X_1 \times \Delta Y$  then
23:      continue // no NRA intersection
24:    end if
25:  else
26:    if  $Y_1 \times \Delta X \geq X_1 \times \Delta Y$  then
27:      continue // no NRA intersection
28:    end if
29:  end if
30:   $I_p \leftarrow \text{sgn}(\Delta Y)$  // There is NRA intersection
31:  if  $Y_1 == 0$  or  $Y_2 == 0$  then
32:     $I_p \leftarrow I_p \times 0.5$  // touch NRA only
33:  end if
34:   $I_C \leftarrow I_C + I_p$  // Update NRA crossing counter
35: end for
36: if  $I_C \neq 0$  then
37:  return true // point  $a$  is included
38: else
39:  return false // point  $a$  is not included
40: end if

```

```

Q=P-repmat(A,1,size(P,2));%translate to origin
Y1=Q(2,1:end-1);
Y2=Q(2,2:end);
DY=Y1-Y2;
X1=Q(1,1:end-1);
X2=Q(1,2:end);
DX=X1-X2;
Y1DX=Y1.*DX;
X1DY=X1.*DY;
noCrossSgn=(sign(Y1)==sign(Y2));
noCrossYA=(DY >0 & Y1<0);
noCrossYB=(DY<=0 & Y2<0);
noCrossXA=(sign(DY)> 0 & Y1DX < X1DY);
noCrossXB=(sign(DY)< 0 & Y1DX > X1DY);
Ip=~(noCrossSgn | noCrossYA | noCrossYB | noCrossXA | noCrossXB );
Ip=Ip.*sign(DY).*(1-0.5*(Y1==0 | Y2==0));
IC=sum(Ip);

```

**Figure 11.** Example of vectorization of Algorithm 4 in Matlab code.

### 5.7. Comparison to the State-of-the-Art Algorithm

For several years, Matlab has been using a version based on the algorithm from Hormann and Agathos [8], valid for point location in arbitrary polygons. According to the original paper, that algorithm is based on three steps, which, on the optimized variant, includes the following operations:

- Evaluation of the determinant: performs 2 multiplications and 1 subtraction per segment;
- Quadrant classification: uses 6 comparisons per vertex;
- Determination of winding number: performs 2 subtractions/sums and up to 4 + 1 comparisons per vertex.

Algorithm 4 has more steps (though simpler) than Algorithm 3 and requires:

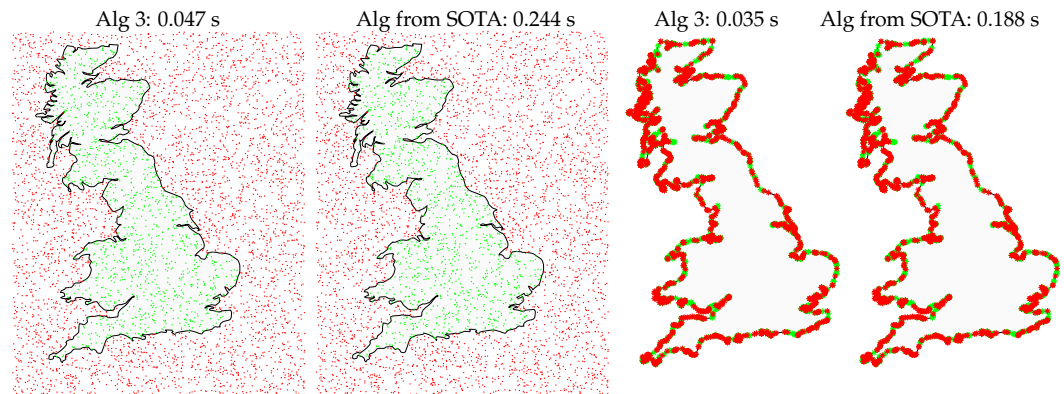
- Initial test: 1 sign comparison per vertex;
- Coordinate differences: 2 per vertex;
- Y coordinate conditions: up to 3 comparisons per vertex;
- Test NRA: 2 multiplications and 2 comparisons per vertex;
- Test special cases end points: 2 comparisons with zero;
- Crossing counter: 1 addition per NRA crossing.

The comparison is given side-by-side in Table 1, where it can be seen that Algorithm 4 has fewer comparisons ( $1 + 2 + 2 + 2 = 7$  vs.  $10 + 1 = 11$ ) and potentially much less if tests dismiss early; there is a similar number of additions/sums:  $2 + 1$  vs.  $3$ ; the multiplications are in equal number but the advantage is that in Algorithm 4 they are needed only when x-axis intersection occurs. In the previous analysis,  $2M$  subtractions were omitted, reporting to the relocation of the polygon around point  $(0,0)$ , but which is a common operation to all.

**Table 1.** Comparison of estimated number of operations per vertex/segment in the algorithms.

	Algorithm 3	Algorithm 4	Algorithm from [8]
Comparisons	$1 + 3 + 2$	$1 + 2 + 2 + 2$	$10 + 1$
Sums/subtractions	$2 + 1$	$2 + 1$	$3$
Multiplications	$1$	$2$	$2$
Divisions	$1$	$0$	$0$

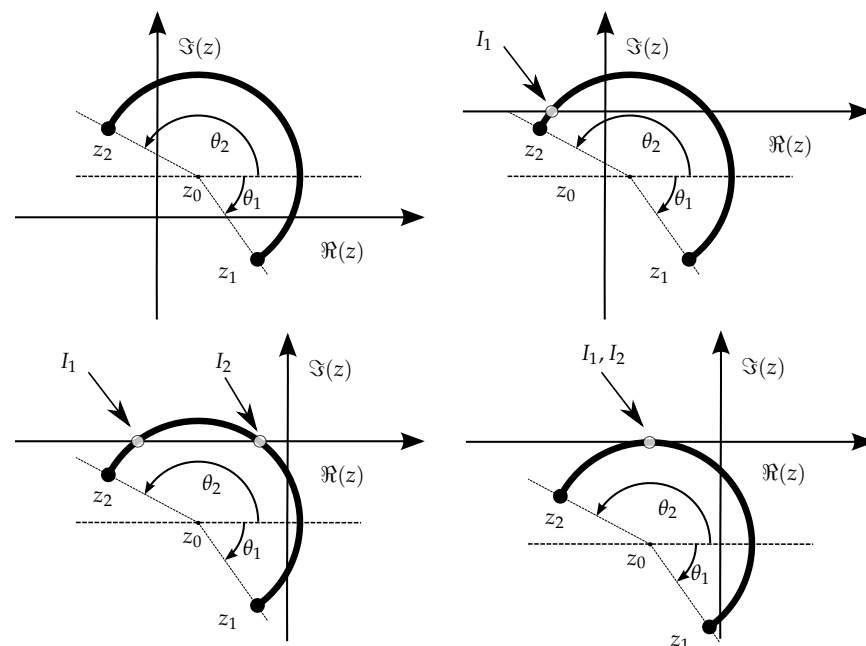
An illustrative example of simple benchmarking is shown in Figure 12 with a polygon with more than 2200 sides and thousands of testing points in two situations: on the left, 5000 completely random points, and on the right more than 2200 points near the border. Results are shown for Algorithm 3 and the state-of-the-art algorithm in Matlab [8]. The result shown is the average of 100 runs and had the same outcome in both algorithms. In both cases, Algorithm 3 proposed in this paper outperforms by a factor of about 5.



**Figure 12.** Example of points tested in a 2216-side polygon: 5000 random on the left and more than 2200 random points very close to the borders on the right. As indicated at the top of each image, Algorithm 3 ran about 5 times faster than the state-of-the-art (SOTA) Matlab’s native algorithm [8]. Tested in a HP ZBook Fury 15 with Intel Core i7-10750H CPU 2.60 GHz with 32 GBytes of RAM in Ubuntu Linux 22.04.3.

## 6. NRA Crossing and Sense for Circular Arcs

For circular arcs, a few changes occur relative to linear segments. What is the most apparent is the fact that circular arcs can cross the NRA more than once or may even be tangent (Figure 13). Moreover, the parametric definition of arcs is a little more elaborate than for straight lines, as described next.



**Figure 13.** Examples of arc segments with center in  $z_0$  starting on point  $z_1$  and ending at point  $z_2$ . Depending on  $z_0$ , the very same arc can intersect the negative real axis (NRA), zero times, once ( $I_1$ ), twice ( $I_1$  and  $I_2$ ), or be tangent ( $I_1 \equiv I_2$ ), which, for the purposes of path integral evaluation, is equivalent to intersect the NRA twice, in opposite senses.

### 6.1. Parametric Expression for Circular Arcs

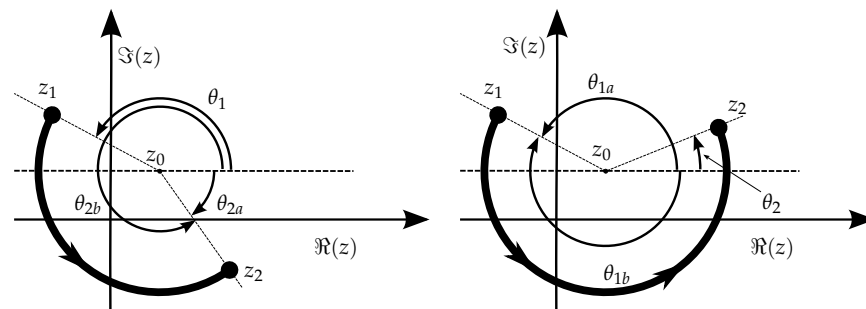
Any point of a circular arc with center at  $z_0$ , and evolving in the positive sense (counter-clockwise—CCW), between point  $z_1 = z_0 + re^{j\theta_1}$  and point  $z_2 = z_0 + re^{j\theta_2}$  is expressed by  $z = z_0 + re^{j\theta}$ , where  $\theta$  covers the interval defined by  $\theta_1$  and  $\theta_2$ , and  $r = |z_0 - z_1| = |z_0 - z_2|$ .

To simplify the parametrization and unify the procedures used for the linear segment, the following parametric representation is used:

$$z = z_0 + re^{j[\theta_1 + t(\theta_2 - \theta_1)]}, 0 \leq t \leq 1. \quad (50)$$

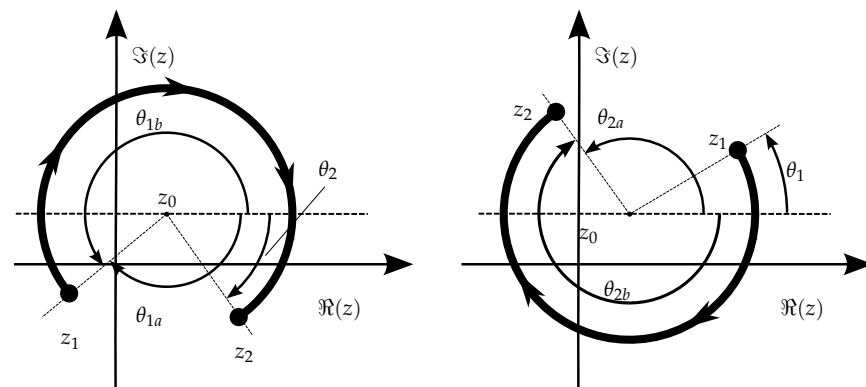
However, to ensure an unambiguous interpretation of the path described for the positive sense, it must be  $\theta_1 \leq \theta_2$  where it is expected that the angles are obtained by the following operations:  $\theta_1 = \arg(z_1 - z_0)$  and  $\theta_2 = \arg(z_2 - z_0)$ . But if that is not the case, then one of the angles must be converted by modulus  $2\pi$ , that is, force  $\theta_1$  and  $\theta_2$  to fall in the interval  $]-2\pi, +2\pi]$ , by applying either  $\theta_2 \leftarrow \theta_2 + 2\pi$  or  $\theta_1 \leftarrow \theta_1 - 2\pi$ , depending on the necessary case.

All the situations in Figure 13 respect the proper condition,  $\theta_1 \leq \theta_2$ , but different situations are illustrated in Figure 14 where  $\theta$  encounters a discontinuity if restricted to the interval  $]-\pi, +\pi]$ . As the figure explains, in one case the solution for a proper parametrization is to convert  $\theta_2$  into  $\theta_2 + 2\pi$  and the other to convert  $\theta_1$  into  $\theta_1 - 2\pi$ .



**Figure 14.** Example of arc segments where the CCW evolution of  $\theta$  from  $\theta_1$  to  $\theta_2$  would not be monotonic when  $\theta_i \in ]-\pi, +\pi]$ , but that can be made monotonic by taking the equivalent angles: on the left,  $\theta_2 \leftarrow \theta_2 + 2\pi$ , that is, use  $\theta_{2b}$  instead of  $\theta_{2a}$ , and on the right  $\theta_1 \leftarrow \theta_1 - 2\pi$ , that is, use  $\theta_{1b}$  instead of  $\theta_{1a}$ .

On the other hand, if a negative (or clockwise, CW) sense is intended, an equivalent operation must be taken into account, as illustrated in Figure 15.



**Figure 15.** Example of arc segments where the CW evolution of  $\theta$  from  $\theta_1$  to  $\theta_2$  would not be monotonic when  $\theta_i \in ]-\pi, +\pi]$ , but that can be made monotonic by taking the equivalent angles: on the left,  $\theta_1 \leftarrow \theta_1 + 2\pi$ , that is, use  $\theta_{1b}$  instead of  $\theta_{1a}$ , and on the right  $\theta_2 \leftarrow \theta_2 - 2\pi$ , that is, use  $\theta_{2b}$  instead of  $\theta_{2a}$ .

In summary, to ensure the proper and unambiguous parametrization of a circular segment, the following steps have to be carried out when defining a circular boundary segment:

- Indicate  $z_1$ ,  $z_2$ , and the center  $z_0$  (or some other means to obtain  $z_0$ , like a third point  $z_p$ , or a radius and the relative placement of the center);
- Obtain  $\theta_1 = \arg(z_1 - z_0)$  and  $\theta_2 = \arg(z_2 - z_0)$ ;
- Define circulation sense (CCW or CW);
- Check the potential angle adjustments given by (51):

$$\begin{cases} \theta_2 \leq \theta_1 \wedge \text{CCW} \\ \theta_1 \leq \theta_2 \wedge \text{CW} \end{cases} \begin{cases} \theta_2 < 0 \Rightarrow \theta_2 \leftarrow \theta_2 + 2\pi \\ \theta_2 \geq 0 \Rightarrow \theta_1 \leftarrow \theta_1 - 2\pi \\ \theta_1 < 0 \Rightarrow \theta_1 \leftarrow \theta_1 + 2\pi \\ \theta_1 \geq 0 \Rightarrow \theta_2 \leftarrow \theta_2 - 2\pi \end{cases} \quad (51)$$

In expression (51), the verification of the equality between  $\theta_1$  and  $\theta_2$  is also considered to include the situation of complete circles (where  $\theta_1$  and  $\theta_2$  would coincide, but actually one of them should be  $2\pi$  larger than the other to ensure that the possibility of a full circle is covered).

Having the circular path segments duly defined and parametrized, the calculation of the NRA intersections is as straightforward as for the linear segment. Hence, to adapt for the circular arc, there must exist one (or two) values for  $t$  that generate an NRA crossing, and let us call them  $t_c$ :

$$x_c + jy_c = (x_0 + jy_0) + re^{j[\theta_1 + t_c(\theta_2 - \theta_1)]}, 0 \leq t_c \leq 1 \quad (52)$$

or in expanded view:

$$\begin{cases} x_c = x_0 + r \cos[\theta_1 + t_c(\theta_2 - \theta_1)] & 0 \leq t_c \leq 1 \\ y_c = y_0 + r \sin[\theta_1 + t_c(\theta_2 - \theta_1)] & 0 \leq t_c \leq 1 \end{cases} \quad (53)$$

For the same conditions as before,  $x_c \leq 0$  and  $y_c = 0$  as seen in Section 5, we then have:

$$\begin{cases} x_0 \leq -r \cos[\theta_1 + t_c(\theta_2 - \theta_1)] & 0 \leq t_c \leq 1 \\ y_0 = -r \sin[\theta_1 + t_c(\theta_2 - \theta_1)] & 0 \leq t_c \leq 1 \end{cases} \quad (54)$$

However, since the circle can cross the NRA twice, two solutions would be expected for  $t_c$  in the second equation of (54); indeed, the total number of solutions in  $\theta$  of an equation of the type  $x = \sin(\theta)$  is given by the following:

$$\theta = \begin{cases} \arcsin x \pm 2k\pi & k = 0, 1, 2, \dots \\ \pi - \arcsin x \pm 2k\pi & k = 0, 1, 2, \dots \end{cases} \quad (55)$$

that, when restricted to the  $[-2\pi, +2\pi]$  interval, simplifies into, at most, the following 5 solutions:

$$\theta = \begin{cases} \arcsin x \\ \arcsin x \pm 2\pi \\ -\arcsin x \pm \pi \end{cases} \quad (56)$$

Resuming to the variables from (54), we have

$$\begin{cases} t_{c1} = \frac{\theta_1 + \arcsin \frac{y_0}{r}}{\theta_1 - \theta_2} \\ t_{c2,3} = \frac{\theta_1 + \arcsin \frac{y_0}{r} \pm 2\pi}{\theta_1 - \theta_2} \\ t_{c4,5} = \frac{\theta_1 - \arcsin \frac{y_0}{r} \pm \pi}{\theta_1 - \theta_2} \end{cases} \quad (57)$$

where all those potential solutions, besides needing to verify the condition  $0 \leq t_c \leq 1$ , must also verify the first condition of (54), that is,  $x_0 \leq -r \cos[\theta_1 + t_c(\theta_2 - \theta_1)]$ .

Naturally, at most two of the solutions from (57) are valid (two intersections of the  $x$  axis), but there can be only one, or even no, solutions at all. As just mentioned, both potential solutions (let us name them  $t_{cA}$  and  $t_{cB}$ ) must be tested whether they verify expression (54). Nonetheless, there are obvious cases that can be tested beforehand to



accelerate the numeric algorithms: one is  $|y_0| > r$ , resulting in a clear impossibility, and the other case is when  $|y_0| = r$ , implying that the arc is tangent to the  $x$  axis.

For the case of circular arcs, the situation of tangency may be interpreted as either a non-intersection or a double touching of the NRA (entering and leaving the NRA at the same point). The net additional contribution for the calculation of the line integral is, therefore, zero. Hence, if  $|y_0| \geq r$ , then the circular segment does not affect the intersection counter  $I_C$ . The exception occurs in case the tangency point ( $|y_0| = r$ ) happens for  $t_c$  being 0 or 1, which is a “half intersection”, as described next.

## 6.2. Sense of NRA Crossing for Circular Arcs

As before, in case there is an NRA intersection, its sense must be determined, and expression (43) can be used and adapted for the case of arcs:

$$I_C = -\operatorname{sgn}\{(\theta_2 - \theta_1)r \cos[\theta_1 + t_c(\theta_2 - \theta_1)]\}. \quad (58)$$

Expression (58) can be further simplified avoiding the need to calculate the trigonometric function. Knowing that  $r > 0$  and being  $\theta(t_c) = (\theta_1 + t_c(\theta_2 - \theta_1)) \bmod 2\pi$ , we have:

$$I_C = \begin{cases} \operatorname{sgn}(\theta_1 - \theta_2) & \Leftarrow -\frac{\pi}{2} < \theta(t_c) < \frac{\pi}{2} \\ \operatorname{sgn}(\theta_2 - \theta_1) & \Leftarrow \text{Otherwise} \end{cases}. \quad (59)$$

Also, as occurred for the linear segments, the particular cases of  $t_c = 0$  or  $t_c = 1$  correspond to “half intersections” (one extremity of the arc is over the NRA) and, therefore, the  $I_C$  calculated with (59) must be adjusted the same way, as given by (45), or formally stated:  $t_c = 0 \vee t_c = 1 \Rightarrow I_C \leftarrow 0.5 \times I_C$ .

## 7. Application to Bézier Curves

Most planar domains can be defined by closed splines or other arbitrary curve segments, delimited by two extreme (anchor) points and some control points in a parametric way, which is most frequently of a polynomial nature. Each segment can however be treated the same way as was demonstrated earlier for linear and circular segments. For example, Bézier curves of order  $n$  in the complex plane can be defined by:

$$B_n(t) = \sum_{k=0}^n t^k \left( \frac{n!}{(n-k)!} \sum_{i=0}^k \frac{(-1)^{i+k} z_i}{i!(k-i)!} \right) \quad (60)$$

where  $z_i$  are the control points, starting at  $z_0$  and ending at  $z_n$  (the anchor points), and with  $0 \leq t \leq 1$ .

Segments are described by polynomials, making it easy both to differentiate and find roots, except for higher orders (like 5, 6, or more), and in that case, it can represent a challenge for analytical approaches, although numerical approaches are always possible. Bézier curves can also be expressed in the more compact notation:

$$B_n(t) = \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i z_i, \quad 0 \leq t \leq 1 \quad (61)$$

from which follows an example of a Bézier segment of the 3<sup>rd</sup> order, which is one of the most commonly used, thus, representative of many challenges in computational geometry:

$$B_3(t) = (1-t)^3 z_0 + 3(1-t)^2 t z_1 + 3(1-t) t^2 z_2 + t^3 z_3 \quad (62)$$

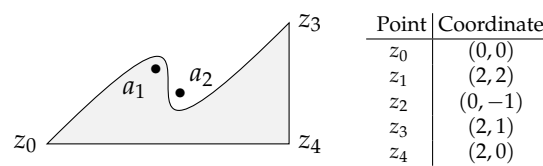
or, in full expansion:

$$B_3(t) = (3z_1 - z_0 - 3z_2 + z_3)t^3 + 3(z_0 - 2z_1 + z_2)t^2 + 3(z_1 - z_0)t + z_0. \quad (63)$$

As described earlier, detecting the NRA crossing of such a segment requires the solution of  $B_3(t) = 0$ , which would yield up to 3 results for  $t$  in the  $[0, +1]$  interval. If less than 3 NRA intersections occur, this means that some solutions for  $t$  are complex, and are to be discarded. To determine the sense of growing at the intersection points with the NRA, the derivative of  $B_3(t)$  in order to  $t$  is required, and that too is a direct operation easily allowed by the polynomial representation of the curve segment:

$$\frac{dB_3(t)}{dt} = 3(3z_1 - z_0 - 3z_2 + z_3)t^2 + 6(z_0 - 2z_1 + z_2)t + 3(z_1 - z_0). \quad (64)$$

Figure 16 shows a region defined by a cubic Bézier curve and two linear segments. In this case, the Bézier anchor points are  $z_0 = (0, 0)$  and  $z_3 = (2, 1)$  and the two control points are  $z_2 = (0, -1)$  and  $z_3 = (2, 1)$ . The two linear segments are  $\overline{z_3z_4}$  and  $\overline{z_4z_0}$ . Points  $a_1 = (0.9, 0.6)$  and  $a_2 = (1.1, 0.4)$  will be tested but only  $a_1$  is included in the region.



**Figure 16.** Region delimited by a cubic Bézier and two linear segments. Points  $z_1$  and  $z_2$  (not plotted in the figure) are the Bézier control points.

To test points  $a_1$  and  $a_2$ , the procedure seems to require the adjustment of the “vertices” values reported to each testing point and calculate the new resulting polynomials. Actually, that turns out unnecessary in the parametric form because the new polynomial only changes by an offset, and the generic expression (63) simply becomes  $B_3(t) \leftarrow B_3(t) - a_i$ , where  $a_i$  is the point being tested.

An additional relevant aspect for computational purposes is that the expression for the Bézier polynomial derivative (64) remains the same whatever the testing point  $a_i$ , which saves computation resources when testing the inclusion of many different points for the same region. The generic equations to test the NRA crossing and contributions ( $I_C$ ) for a given  $a_i = (a_{x_i}, a_{y_i})$  are:

$$\begin{cases} x_{B_3} = 8t^3 - 12t^2 + 6t - a_{x_i} \\ y_{B_3} = 10t^3 - 15t^2 + 6t - a_{y_i} \\ \frac{dy_{B_3}}{dt} = 30t^2 - 30t + 6 \end{cases}.$$

To solve the problem for this particular situation, the procedure is implemented as follows:

1. Find the roots of  $y_{B_3} = 10t^3 - 15t^2 + 6t - a_{y_i}$ ;
2. Test whether the found roots satisfy the condition of NRA intersection, that is,  $x_{B_3} = 8t^3 - 12t^2 + 6t - a_{x_i} \leq 0$ ;
3. For each root that satisfies the previous expression there is a NRA intersection whose sense needs to be assessed by the sign of  $\frac{dy_{B_3}}{dt} = 30t^2 - 30t + 6$ .

Table 2 shows some numerical results that illustrate the functionality of the algorithm for a cubic Bézier segment for two test points  $a_1$  and  $a_2$ .

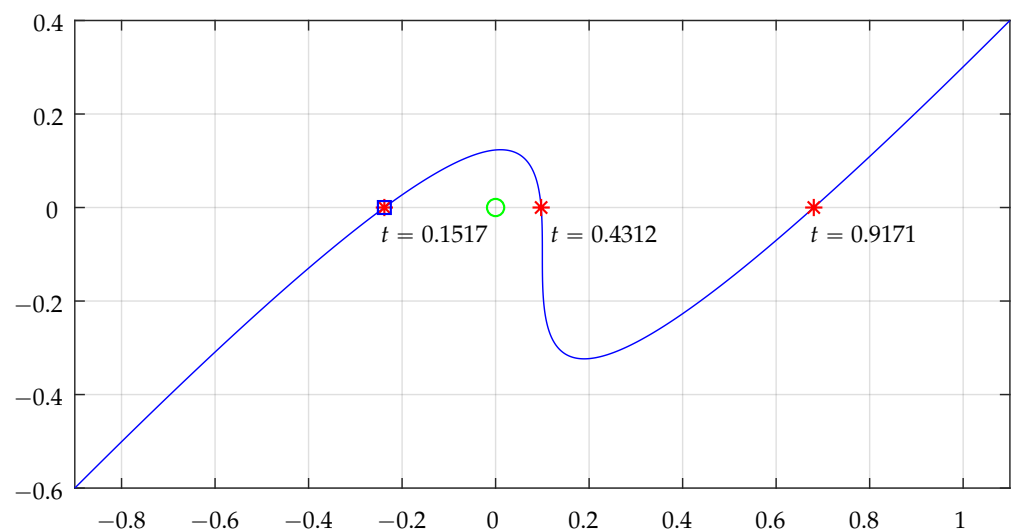
Figures 17 and 18 illustrate the plots concerning NRA intersections by the cubic Bézier segment for the cases of point  $a_1$  and  $a_2$ , respectively.

The intersection contributions  $I_C$  (detailed in Table 2) clearly translate the situations of inclusion of the tested points. For  $a_1$ , total  $I_C$  is  $-1$  and for  $a_2$  total  $I_C$  is  $+1 - 1 = 0$ . Actually, the  $I_C$  contributions from the two linear segments were not analyzed, but it is simple to confirm that they do not contribute at all because they do not cross the NRA: the

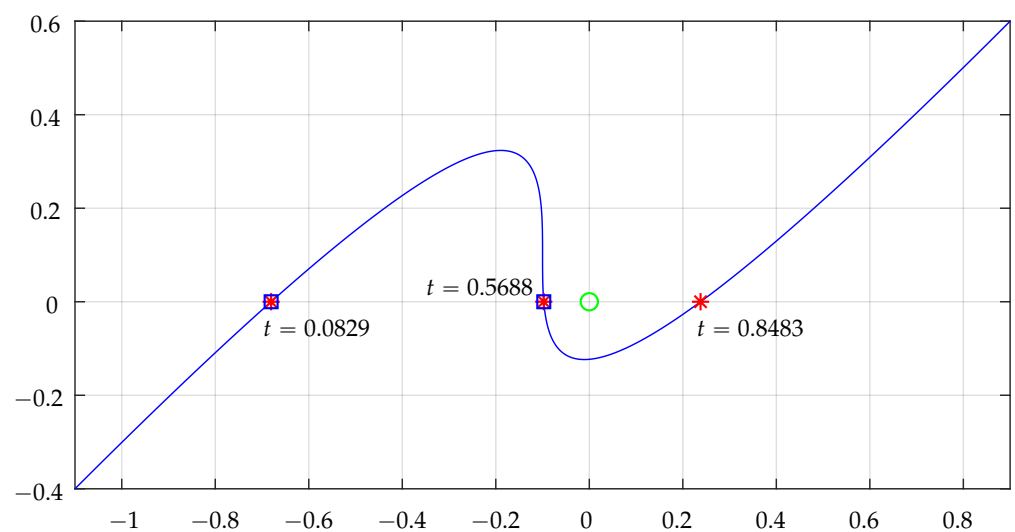
horizontal segment will never contribute, and the vertical segment does not cross the NRA for the points being tested.

**Table 2.** Detailed analysis of operations concerned with NRA crossings by a cubic Bézier segment for two testing points in the case illustrated in Figures 16–18.

Testing Points	$t_i$	$x_{B_3}(t_i)$	$\frac{dy_{B_3}}{dt}(t_i)$	$I_C$	$\sum I_C$
$a_1$	$t_1 = 0.1517$	$-0.2379$	yes	2.139	$-1$
	$t_2 = 0.4312$	$0.0974$	no	$-1.358$	$0$
	$t_3 = 0.9171$	$0.6805$	no	3.719	$0$
$a_2$	$t_1 = 0.0829$	$-0.6805$	yes	3.719	$-1$
	$t_2 = 0.5688$	$-0.0974$	yes	$-1.358$	$+1$
	$t_3 = 0.8483$	$0.2379$	no	2.139	$0$



**Figure 17.** Bézier real axis intersections for testing point  $a_1$ . The “\*” are the points where the curve intersects the horizontal axis, the green circle represents the point being tested when shifted to the system origin, and the square indicates a NRA crossing that effectively contributes to the intersection counting. Check Table 2 for more details.

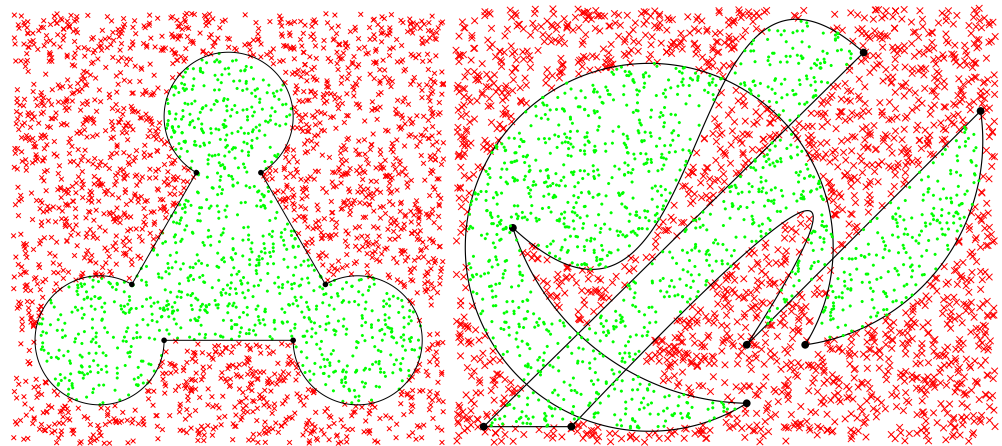


**Figure 18.** Bézier real axis intersections for testing point  $a_2$ . The “\*” are the points where the curve intersects the horizontal axis, the green circle represents the point being tested when shifted to the system origin, and the squares indicate a NRA crossing that effectively contributes to the intersection counting. Check Table 2 for more details.

With higher order curves, such as these 3<sup>rd</sup> order Bézier, it may happen that the intersection point is tangent to the NRA and expression (43) would result in 0 (inconclusive). This is an easy situation to detect (one further comparison) and concerns a case where all three roots  $t_c$  are equal. There are two solutions for that very special case to allow the detection of the intersection sense: one would be to calculate expression (43) in a “small” neighborhood of  $t_c$ , for example,  $t_c - \delta$  instead of  $t_c$  (where the value of  $\delta$  can be on the range of the precision involved—e.g.,  $10^{-6}$ ); the other solution, formally more elegant, is simply to check the quadrants of  $z_1$  and  $z_2$ , i.e., use the value of  $\text{sgn}(y_1 - y_2) = \text{sgn}(\Delta Y_b)$ .

#### *Application to the Challenges Given*

By using the previous techniques for linear, circular, and Bézier segments in an integrated and complementary form, it is possible to test situations as those illustrated in Figure 1 for a set of random points, which results as shown in Figure 19.



**Figure 19.** Results of algorithms applied to the regions of Figure 1. Due to limited resolution in the display above, some points may seem to be on the border, but actually, none of them are.

From the point of view of programming, three functions were created and used: `NRAintLin()`, `NRAintArc()` and `NRAintBez()`, whose source code is also available at the GitHub repository indicated earlier. These functions return the NRA counting for each respective segment when invoked accordingly to the type of segment: 3 linear and 3 circular arcs on the region on the left, and 2 linear, 3 circular arcs and 2 Bézier segments on the region on the right. These three functions are essential to establish the general algorithm for regions delimited by linear, circular, or Bézier segments, which is the subject of the next section.

### **8. Algorithm for Arbitrary Planar Domains**

After the statements, demonstrations, and examples described in the previous sections, it is now straightforward to establish the basis of a general algorithm for arbitrary planar domains that can be described by parametric equations, which have associated a sense of circulation along the boundary. The procedure can be stated as follows:

1. Set up the closed region by segments defined parametrically in such a way that each segment is defined by two extremes  $z_1$  and  $z_2$ , a parameter  $t$ , being  $0 \leq t \leq 1$ , and optional points or parameters depending on the type of the segments (linear, circular arcs, Bézier curves, etc.)
  - Linear segments do not require any further parameters or points;
  - circular arcs require a third point (center), or else a radius along with the definition of its orientation;
  - other curves (of second or higher order) require additional elements to fully define them (3<sup>rd</sup> order Bézier curves require two more points).
2. Normalize the domain delimiting points (vertices) relative positions by subtracting the test point  $a$  to all vertices.

3. Check the intersections of this newly relocated region with the negative half part of the horizontal axis ( $x$ ). Intersections occur from  $Q2$  to  $Q3$  or vice-versa. Full intersections are accounted for an integer value ( $+1$  or  $-1$ , depending on the sense,  $Q2$  to  $Q3$  ( $\downarrow$ ) or  $Q3$  to  $Q2$  ( $\uparrow$ ), respectively), or a partial value ( $+0.5$  or  $-0.5$ ) if the segment starts or ends on the very horizontal axis ( $\uparrow$ ,  $\downarrow$ ), and the positive or negative contribution is determined once again if the segment goes or comes to/from quadrants  $Q2$  or  $Q3$ .
4. The verification of the sense of the evolution of the path must be asserted in these intersection/touching points: it is straightforward for linear segments, but for other types of segments, the descendant gradient of  $y$  with respect to  $t$  must be used at that point to determine the sense of cross/approach to the horizontal axis.

Algorithm 5 formalizes the required steps.

---

**Algorithm 5:** Universal inclusion test in arbitrary regions
 

---

**Require:**  $a = (a_x, a_y)$ ,  $P_n = (x_n, y_n)$  for  $n = 1, 2, \dots, M$

**Require:**  $L_n \in \{0, 1, 2, 3, \dots\}$  // type of segment/number of control points

**Require:**  $C_{n_k} = \{(x_k, y_k)\}_n$  for  $k = 0, 1, \dots, L_n$  // Control points // It is admitted that

$$P_{M+1} = P_1.$$

```

1: if  $a \in \{P_1, P_2, \dots, P_M\}$  then
2:   return true // point  $a$  is one of the vertices
3: end if
4:  $P_n \leftarrow P_n - a$  for  $n = 1, 2, \dots, M$ 
5:  $C_n \leftarrow C_n - a$  for  $n = 1, 2, \dots, M$ 
6:  $I_C \leftarrow 0$ 
7: for  $n = 1$  to  $M$  do
8:    $[X(t), Y(t)] = f(P_n, P_{n+1}, \{C_n\}, L_n)$ 
9:    $T \leftarrow$  solutions of  $Y(t) = 0$ 
10:  for all  $t_c$  in  $T$  do
11:    if  $X(t_c) == 0$  then
12:      return true // point  $a$  is on border (included).
13:    end if
14:    if  $t_c \geq 0$  and  $t_c \leq 1$  and  $X(t_c) \leq 0$  then
15:       $I_P \leftarrow -\text{sgn}\left(\frac{dY(t)}{dt}\bigg|_{t=t_c}\right)$  // intersects NRA
16:      if  $I_P == 0$  and  $L_n$  is a cubic Bézier then
17:         $I_P \leftarrow \text{sgn } \Delta Y_b$  // Cf. text for definition of  $\Delta Y_b$ 
18:      end if
19:    end if
20:    if  $t_c == 0$  or  $t_c == 1$  then
21:       $I_P \leftarrow I_P \times 0.5$  // touch NRA only
22:    end if
23:     $I_C \leftarrow I_C + I_P$ 
24:  end for
25: end for
26: if  $I_C \neq 0$  then
27:   return true // point  $a$  is included
28: else
29:   return false // point  $a$  is not included
30: end if

```

---

A most relevant point when comparing this general algorithm to the algorithms presented in the preceding sections is the input of data: besides the testing point and the list of vertices that require no changes at all, there are now more parameters to be given to distinguish and define the types of segments.

The algorithm then requires the following data as inputs:

- The testing point  $a$  (same as before)
- The vertices (extreme points) of the region segments (same as before)
- A set of numbers indicating the type of segment or number of control points (e.g., 0 for linear, 1 for circular, 2 for 3<sup>rd</sup> order Bézier segments, etc.)
- A set of control points for each segment

The steps of the algorithm that vary with the type of segment occur mainly in lines 8 and 9, where the parametric equations of the segment must be obtained and solved, and also in line 15 to calculate a derivative and obtain the sense of NRA crossing.

As mentioned before, the GitHub repository made available also includes the code and illustration of the application of this general algorithm with several examples besides the ones in Figure 19. No benchmarking is performed with this general algorithm because, in the literature, there are no other algorithms for this same purpose to compare with.

## 9. Discussion and Conclusions

This paper presents a methodology to solve, in analytical form, the problem of point inclusion in planar domains for regions of virtually unlimited complexity, namely with boundary curves of order greater than linear segments. Specifically, for the cases for regions bounded arbitrarily by polygonal, circular arcs and Bézier lines have been derived and implemented, and the methodology for regions with other shapes has been described and is ready to be applied to the general algorithm.

The approaches described are handled in analytical form, so they are independent of any numerical approximations, hence independent of scale. Nonetheless, when performing actual calculations in floating point operations, the limitation of the numeric representation of the computational system naturally occurs. However, due to the nature of the operations (sums and multiplications in the Algorithms 3 and 4 for polygonal regions only), the propagation of uncertainty is minimal, and operations can even be performed in integer representation, which is suitable for some embedded systems. On the other hand, this is not the case for the general approach of non-linear segments (covered in Algorithm 5) because boundary segments are non-linear, and non-linear calculations are required, including trigonometric or root operations. Since the accumulated chain of non-linear operations is short, uncertainty propagation is contained, and results have shown that very good accuracy (in 12 or more decimal places) is kept in the final outcome. This can be checked, for example, by “zooming” into the plots extensively and verifying the correct result of point inclusion. The mentioned GitHub repository allows these tests to be performed by other users.

The paper also demonstrates why a simple winding number based technique for detecting point inclusion will always be ambiguous if the testing point is on the border. Nonetheless, the proposed algorithms detect those situations early, and the technique relies on the need of a single pass operation, i.e., it requires no iterative procedure or further verifications.

Besides this advantage of a single operation, the possibility of extending the algorithm to regions of more complex shapes opens wider directions in the field of simulation and computational geometry. The usage for curves of high orders needs to deal with negative real axis intersection, which is equivalent to finding the roots of polynomials or other transcendental curves. This implies unavoidable additional computational costs when compared to straight polygonal shapes. Nevertheless, the pure analytical formulation of the method may remain an advantage against the alternatives of linearization or approximation of curves to polygonal lines.



Along these lines, a final remark is worth mentioning, which concerns a potential alternative to this approach for general shapes. We could think of the possibility of triangulating a region such as the ones shown in Figure 19 and apply point location techniques cited in Section 2. Although the technique is usually applied for a problem not addressed in this paper, a region could be triangulated and the test of inclusion of a point in the overall region could be performed by identifying which subregion, if any, includes that point. Besides the variable computational costs of this extreme solution, two main limitations immediately arise. The first is concerning self-intersecting regions and regions with holes (like the ones in Figure 10) that require additional care and pre-processing to be triangulated. Secondly, and even more relevant, if the region boundaries are curved, they must be linearized (sampled) to perform the triangulation. This has two drawbacks: the computational demands of the preprocessing and an additional computational cost that depends on the scale. The dependency on scale is particularly relevant near the boundaries: if very high accuracy is required, the samples over the curved boundaries must be more dense; hence, a larger number of triangles is generated in the mesh. This dependency on scale (with consequences on the accuracy in areas closer to the borders) does not occur in the proposed approach because it is analytical.

The next list of topics explains the advantages of the proposed approach in this paper, the reasons for better performance than existing solutions, and the reason for that to happen.

- It involves fewer operations: besides the common sums and subtractions, at most only one floating point division is required per segment (or two multiplications instead) and also fewer comparisons, for the case of polygonal shaped regions. This makes it faster than state of the art solutions (about 5 times as fast was demonstrated for polygons with many sides and a large number of data points for the algorithm implemented in Matlab [8]).
- It has no restrictions on which points define the polygons (or the generically shaped region) nor on the testing points that can be over the border, or being themselves the very vertices, and no particular post processing or operations are required.
- It accepts the geometry of boundary segments to be other than straight lines and is applicable virtually to any type of curve that can be expressed parametrically.
- The parametric formulation of the boundary makes it straightforward to define the curve segments and their extremes (anchor points).
- The parametric definition is intrinsically associated with the concept of sense or circulation, which allows the detection of the sense of axis intersection, and not only the fact that there are intersections; and the sense of intersection is crucial to obtaining the global winding number.
- The parametric description allows a separation of coordinates, making algebraic operations independent, and allows us to overcome limitations in degenerate situations that would occur in two coordinate point representations but do not occur in separate coordinate representations.

Although the formal and analytic procedures for point location in generic planar domains have been fully established and demonstrated in this paper, the future may hold improvements in the computational component of the algorithms for specific curves. This can be true, namely for some curves of very high complexity, as well as for the detection of specific situations of point location relative to those more complex boundaries, in order to accelerate the calculation operations.

**Funding:** An early part of this work was developed under the Ph.D. Grant BD-1657/91\_IA from JNICT, Programa CIENCIA, Portugal.

**Data Availability Statement:** Algorithms and data to support the results of this paper are available at a GitHub repository: <https://github.com/vitoruapt/PointInclusion>.

**Acknowledgments:** The author acknowledges the EC Joint Research Center of Ispra, Italy, for hosting him as a Ph.D. student in the early 1990s' where the basic idea that lead to this research started.

**Conflicts of Interest:** The author declares no conflicts of interest.

## References

1. Alciatori, D.; Miranda, R. *A Winding Number and Point-in-Polygon Algorithm*; Technical Report; Department of Mechanical Engineering, Colorado State University: Fort Collins, CO, USA, 1995.
2. Shimrat, M. Algorithm 112: Position of Point Relative to Polygon. *Commun. ACM* **1962**, *5*, 434. [CrossRef]
3. Hacker, R. Certification of Algorithm 112: Position of Point Relative to Polygon. *Commun. ACM* **1962**, *5*, 606. [CrossRef]
4. Franklin, W.R. PNPOLY-Point Inclusion in Polygon Test. 1994–2006. Available online: [https://wrfranklin.org/Research/Short\\_Notes/pnpoly.html](https://wrfranklin.org/Research/Short_Notes/pnpoly.html) (accessed on 4 October 2024).
5. Preparata, F.P.; Shamos, M.I. *Computational Geometry: An Introduction*; Springer: New York, NY, USA, 1985.
6. Sedgewick, R. *Algorithms in C*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1990.
7. Haines, E. Point in Polygon Strategies. In *Graphics Gems IV*; Heckbert, P.S., Ed.; Academic Press Professional, Inc.: San Diego, CA, USA, 1994; pp. 24–46.
8. Hormann, K.; Agathos, A. The point in polygon problem for arbitrary polygons. *Comput. Geom. Theory Appl.* **2001**, *20*, 131–144. [CrossRef]
9. Aurenhammer, F. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Comput. Surv.* **1991**, *23*, 345–405. [CrossRef]
10. Devillers, O. The Delaunay Hierarchy. *Int. J. Found. Comput. Sci.* **2002**, *13*, 163–180. [CrossRef]
11. Devroye, L.; Mücke, E.P.; Zhu, B. A Note on Point Location in Delaunay Triangulations of Random Points. *Algorithmica* **1998**, *22*, 477–482. [CrossRef]
12. Mücke, E.P.; Saias, I.; Zhu, B. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Comput. Geom.* **1999**, *12*, 63–83. [CrossRef]
13. O'Rourke, J. How Do I Find If a Point Lies within a Polygon. 2003. Available online: <http://www.faqs.org/faqs/graphics/algorithms-faq/> (accessed on 4 October 2024).
14. Sunday, D. Inclusion of a Point in a Polygon. 2012. Available online: [https://web.archive.org/web/20130126163405/http://geomalgorithms.com/a03-\\_inclusion.html](https://web.archive.org/web/20130126163405/http://geomalgorithms.com/a03-_inclusion.html) (accessed on 4 October 2024).
15. Foley, J.D.; van Dam, A.; Feiner, S.K.; Hughes, J.F. *Computer Graphics: Principles and Practice*, 2nd ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1990.
16. O'Rourke, J. *Computational Geometry in C*; Cambridge University Press: Cambridge, CA, USA, 1998.
17. Needham, T. *Visual Complex Analysis*; Oxford University Press: Oxford, UK, 1998.
18. Schirra, S. How Reliable Are Practical Point-In-Polygon Strategies? In Proceedings of the 16th Annual European Symposium on Algorithms (ESA '08), Karlsruhe Germany, 15–17 September 2008; pp. 744–755. [CrossRef]
19. Gatilov, S. Efficient Angle Summation Algorithm for Point Inclusion Test and Its Robustness. *J. Reliab. Comput.* **2013**, *19*, 1–25.
20. Ross, F.; Ross, W.T. The Jordan curve theorem is non-trivial. *J. Math. Arts* **2011**, *5*, 213–219. [CrossRef]
21. Edelsbrunner, H.; Maurer, H.A. A space-optimal solution of general region location. *Theor. Comput. Sci.* **1981**, *16*, 329–336. [CrossRef]
22. Kirkpatrick, D. Optimal search in planar subdivisions. *SIAM J. Comput.* **1983**, *12*, 28–35. [CrossRef]
23. Guibas, L.; Ramshaw, L.; Stolfi, J. A kinetic framework for computational geometry. In Proceedings of the 24th Annual Symposium on Foundations of Computer Science (1983), Washington, DC, USA, 7–9 November 1983; pp. 100–111. [CrossRef]
24. Jackowski, B. Computing the area and winding number for a Bézier curve. In *TUGboat*; Tex Users Group: Portland, OR, USA, 2012; Volume 33. Available online: <https://tug.org/TUGboat/tb33-1/tb103jackowski.pdf> (accessed on 4 October 2024).
25. Santos, V. Robot Autonomous Navigation: Sensorial Data Interpretation and Local Navigation. Ph.D. Thesis, Universidade de Aveiro, Aveiro, Portugal, 1995. Available online: <http://hdl.handle.net/10773/17931> (accessed on 4 October 2024).
26. Wylie, C.; Barrett, L. *Advanced Engineering Mathematics*; McGraw-Hill: New York, NY, USA, 1982.
27. Kanwal, R. *Linear Integral Equations*; Birkhäuser Boston: Boston, MA, USA, 1996.
28. OGC. *OpenGIS Simple Features Specification for SQL, Revision 1.1*; Technical Report; Open Geospatial Consortium, Inc.: Arlington, VA, USA, 1999.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.