XLIII CILAMCE
21–25, November, 2022
FOZ DO IGUAÇU – PR – BRAZIL

Mesh generation and manipulation for finite difference method usage

Pedro Zaffalon da Silva[1], Neyva Maria Lopes Romeiro[2], Rafael Furlanetto Casamaximo[1], Iury Pereira de Souza,[1], Paulo Laerte Natti[2]

[1]*Dept. of Computer Science, State University of Londrina*
Rodovia Celso Garcia Cid - PR 445 Km 380, 86.057-970, Paraná/Londrina, Brasil
pedro.zaffalon@uel.br, rafael.furlanetto@uel.br, iury.pereira.souza@uel.br
[2]*Dept. of Math, State University of Londrina*
Rodovia Celso Garcia Cid - PR 445 Km 380, 86.057-970, Paraná/Londrina, Brasil
nromeiro@uel.br, plnatti@uel.br

Abstract. This work proposes a rectangular mesh generator software which discretize complex geometries, allowing computational fluids dynamics (CFD) simulations. To this porpuse, the software uses different image processing and data analysis techniques to extract a finite set of points that describe the image contour. The resulting coordinates represents the maximum refinement, describing each pixel from the image contour. Hence, procedures with meshs would require high cost of memory. For this reason, the software approximate the coordinates to mesh nodes, compliant with the mesh size selected by the user, allowing a efficient contour description with low memory cost. Then, the approximate contour mesh is obtained, which can be used as parameters to numerical simulations of partial differential equations using the finite difference method. Also, the software allows selection of regions from the geometry to contain more nodes than the rest of the mesh, creating sparse meshs, resulting in better refinement using less memory. Finally, the obtained meshs are compared with the original coordinates by their area, ensuring the mesh generation efficiency.

Keywords: Image processing, mesh nodes, finite difference.

1 Introduction

The computational mesh consists of the discretized representation of a physical domain, described through a given contour, limited by edges or faces, containing vertices called nodes. The mesh is mainly used in modelling and simulations by manipulating differential equations, an essential tool for the analysis and mathematical description of several phenomena. As the vast majority of differential equations do not have an analytical solution, numerical methods are used for their resolution, making it necessary to know information about the geometry of the medium investigated through the computational mesh.

However, in modelling natural phenomena, the domain where the boundary conditions of the problem are defined is hardly found under the nodes of the computational mesh Cuminato and Meneguette [1]. Thus, Cartesian meshes in a two-dimensional plane find it challenging to prescribe boundary conditions in non-regular domains. This makes it difficult to solve the problem considering, for example, the finite difference method Othechar [2]. On the other hand, discretizations using Cartesian meshes are attractive due to their efficiency and low memory usage Fernández-Fidalgo et al. [3]. To avoid this problem, several authors employ methods that use algebraic polynomial interpolations to construct the difference equations at the points of the given contour, allowing the incorporation of the irregular contour into the method, that is, all calculations on irregular domains are reduced to regular domains, obtaining thus a more precise numerical solution to the problem Othechar [2], Jomaa and Macaskill [4], Fukuchi [5], Codina and Baiges [6].

In this context, it is proposed the development of a software called Context, using the Python programming language Rossum and Guido [7], to generate meshes representing the contour of geometry present in an image. The extraction of the coordinates of the pixels of the contour of a certain object in the image, using techniques of digital image processing and the algorithm Moore Neighborhood Weisstein and W [8], was presented in Casamaximo et al. [9]. In this work, the process of transforming the set of pixels coordinates into a mesh is described, by approximating the points to mesh nodes. In this way, the software allows obtaining a discretized representation of a physical domain present in an image, allowing the application of computational fluid dynamics (DFC) simulations.

Also, the possibility of generating sparse meshes is presented. With this option, you can select specific mesh regions to have a greater number of nodes than the rest. Thus, images with regions with greater discretization complexity can be better represented without increasing the number of nodes in the entire mesh. Consequently, better results are obtained in applying the finite difference method and better memory utilization.

2 Development

For the development of the software, the Python programming language [7] was used together with the OpenCV library [10], which allows manipulations for the image processing, data and treatment functions. The Click library [7] was used to manage the command line tools, and the Tkinter framework [7] for the development of the graphical interface (GUI). The developed software has three different executables: context, contextMesh, and contextSparseMesh, responsible, respectively, for contour extraction, mesh generation and generation of sparse or adaptive meshes.

2.1 Contour extraction

As described in Casamaximo et al. [9], this part of the software has the function of extracting the contour of a geometry present in an image through image processing techniques. This result is achieved through two functionalities offered by the executable, being necessary first to extract the mask from the image and then obtain the contour. These processes are illustrated in Figure 1 using the image of a tomography Figure 1a), provided by the patient who developed breast cancer, where the tumour is highlighted at the top of the image. The image mask, Figure 1b) is a conversion of the pixels that form the original image into two colours, black and white, through conditions defined by the user, which can vary for each image, according to the HSV parameters. In this context, the software offers a graphical interface that allows the user to insert an image and choose the necessary parameters to adjust better and extract the mask. To obtain the contour of the breast, Figure 1c), the Moore Neighborhood Weisstein and W [8] algorithm is applied to the image mask. This algorithm acts on a matrix using the colour of the points to differentiate between the contour and fill of images whose colours are binary, black and white. In this step, the software offers options to manipulate the contour scale, allowing changing the width and height values of the set of pixels, in addition to the displacement of the X axis and the Y axis from the origin of the points.

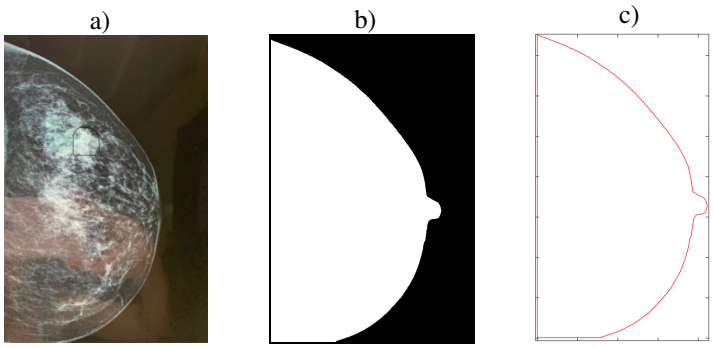

Figure 1. Example of contour extraction: a) original image, b) mask, c) contour plot, in red, in Octave.

2.2 Mesh generation

For the mesh generation, the pixel set, obtained in Figure 1c), can be considered mesh nodes, however, the mesh size would be equal to the image resolution, making its use unfeasible due to the high computational cost. In this context, this executable aims to return an approximate contour, that is, a set of coordinates representing the image contour in nodes of a mesh with user-defined numbers of partitions. Thus, a mesh is defined starting from the rectangular domain $R = [x_0, x_f] \times [y_0, y_f]$, with partitions N_i and N_j, in the directions y and z, respectively, where $dx = (x_f - x_0)/N_j$ and $dy = (y_f - y_0)/N_i$ are the edge sizes (distance between neighboring nodes) on each axis. Figure 2 shows the results of two meshes generated considering $N_i = N_j = 50$ and $N_i = N_j = 100$. A mesh zoom is illustrated in Figure 2, where the contour obtained by the executable, in blue, and the contour extracted from the image, in red, are shown in Figure 2c), noting that the software managed to generate an outline close to the image in Figure 1c).

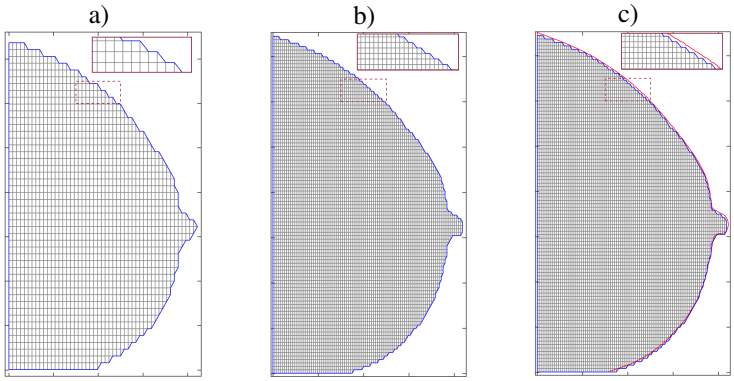

Figure 2. Examples of meshes generated with highlighted zoom: a) mesh with $N_i = N_j = 50$, b) and c) mesh with $N_i = N_j = 100$

In this way, using the sets of pixel coordinates resulting from the extraction of coordinates, Figure 1c), it is possible to form an approximate representation of the geometry contour under the mesh nodes, as shown in Figure 2a)-c). The mesh generation process is described in Algorithm 1.

Algorithm 1: Mesh generation

Input: A vector v of pixel coordinates (x, y) of the contour, function *getNode* that returns the mesh node in which the pixel is contained.
Output: A linked list *list* containing the coordinates of mesh nodes that form the contour.

```
1  begin
2      Create variable previousPt, assign to it the return of getNode with respect to v_0, and add it to list;
3      Create flagx and flagy variables and assign falso to them;
4      Create variables dirx and diry and assign difference between the values of x and y of the first and last point of v to them;
5      for i ← 1 to vector size v − 1 do
6          Create variable currentPt and assign to it the return of getNode with respect to v_i;
7          if currentPt_x != previousPt_x or currentPt_y != previousPt_y then
8              if flagx e currentPt_y != previousPt_y and ((currentPt_x > previousPt_x) != diry) then
9                  Replace last node of list with currentPt;
10             else if flagy and currentPt_x != previousPt_x and ((currentPt_y > previousPt_y) == dirx) then
11                 Replace last node of list with currentPt;
12             else if flagy and currentPt_y != previousPt_y and ((currentPt_x > previousPt_x) != dirx) then
13                 Replace last node of list with currentPt;
14             else if flagx and currentPt_x != previousPt_x and ((currentPt_y > previousPt_y) != diry) then
15                 Replace last node of list with currentPt;
16             else
17                 Add currentPt to list;
18             Create ptAuxiliar variable and assign the penultimate node of list to it;
19             flagx = currentPt_x == ptAtuxiliar_x;
20             flagy = currentPt_y == ptAtuxiliar_y;
21             dirx = currentPt_x > ptAtuxiliar_x;
22             diry = currentPt_y > ptAtuxiliar_y;
23             previousPt = currentPt;
24         end
25     end
26 end
```

The function *getNode*, used in the Algorithm 1, is responsible for approximating a pixel *point* to a node. In this function the coordinates of the approximate node is calculated, being x = **floor**((*point*$_x$ - *xmin*)/*dx*) * *dx* + x_0 and y = **floor**((*point*$_y$ - *ymin*)/*dy*) * *dy* + y_0.

The contour coordinates are traversed and associated with a given mesh node to generate the mesh. After each pixel is approximated, the node is added to the approximate contour if it is not repeated or redundant. Redundant nodes are not adjacent to a node outside the image. Therefore, in addition to not adding extra information to the contour, it decreases the area and number of internal nodes of the geometry. To remove them without affecting other nodes, it is necessary to inform whether the set of pixels is ordered clockwise or counterclockwise. The contour extracted from Figure 1 can be obtained in both directions. If the pixels are passed counterclockwise, the vector will be inverted. Consequently, the Algorithm input 1 and the resulting approximate contour will be clockwise.

2.3 Sparse meshes

Similarly, as described in the 2.2 subsection, this part of the software aims to generate an approximate representation in mesh nodes of the contour extracted from an image. However, this executable returns the approximate contour formed in relation to a sparse mesh, which consists of a mesh containing regions with different dx and dy, that is, regions with $dx/2^p$ and $dy/2^p$, for $p = 0, 1, 2, \ldots$, depending on the refinement desired by the user, thus forming sub-meshes with eigenvalues of dx, dy, N_i and N_j. This executable results in regions with the highest number of nodes compared to the rest of the mesh. Figure 3a)-c) illustrate this procedure, where the region selected for mesh refinement refers to the upper part of the breast, which contains the tumour, as can be seen in the mammogram shown in Figure 1a). Finally, it also presents a zoom of the mesh highlighting the regions with different dx and dy in Figure 3c) the given and approximate contours, in blue and red, respectively.

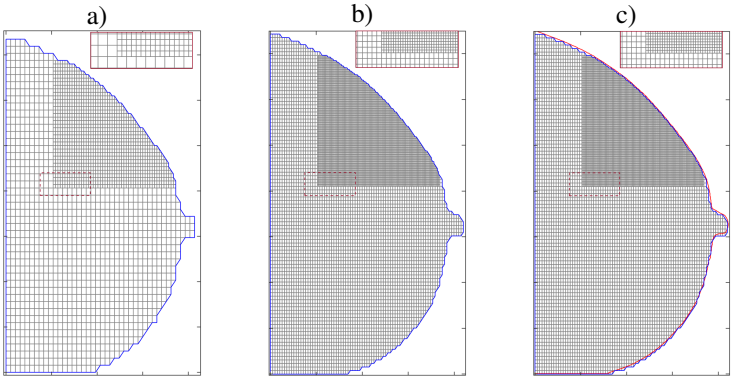

Figure 3. Examples of sparse meshes: a) mesh with $N_i = N_j = 50$, the sub-mesh partitions being $N_i = 37$ and $N_j = 53$, b) and c) mesh with $N_x = N_y = 100$ being the sub-mesh partitions $N_i = 75$ and $N_j = 105$

.

Due to the variation of dx and dy, the mesh definition and the values provided are informed through a separate text file, where each line consists of the definition of a region as if defining a set of sub- meshes. The first line must be in the format $N_x\ N_y\ x_0\ y_0\ x_f\ y_f$ and enclose all regions defined afterwards. The other lines must be in the format $N_{dx}\ N_{dy}\ x_0\ y_0\ x_f\ y_f$, and in this area the values of dx and dy will be the values of dx and dy defined from the first line divided by N_{dx} and N_{dy}. Therefore, the different values of dx and dy must be proportional. The process for generating sparse meshes occurs in the same way as shown in Algorithm 1, with the difference that the approximation is performed as described in Algorithm 2.

Algorithm 2: getNode for sparse mesh

Input: A point pt, a list of mesh areas *areas*.
Output: A node coordinate *node*.

```
1  begin
2      for i ← size of areas - 1 to 0 by −1 do
3          Create variable area and assign areas_i to it;
4          if area_x0 <= pt_x <= area_xf e area_y0 <= pt_y <= area_yf then
5              node_x = floor((pt_x - area_x0)/area_dx) * area_dx + area_x0);
6              node_y = floor((pt_y - area_y0)/area_dy) * area_dy + area_y0);
7              break;
8          end
9      end
10 end
```

Thus, the contextSparseMesh executable becomes appropriate considering the context of the application of numerical simulations, in which the geometry needs to be refined in certain regions. An example would be studies of breast cancer tumor growth with the finite difference technique using Figure 1a), in which greater refinement in the region close to the tumor can generate a more realistic tumor contour, consequently better results, with less memory usage, without affecting the quality of the mesh elements, as can be seen in the zoom shown in Figure 3. However, its use implies greater complexity in implementing the finite difference method due to the abstractions necessary to deal with sub-mesh boundary regions. Also, due to the implementation complexity, the execution is slower. However, this complexity is mitigated by the smaller number of nodes in a fully refined mesh.

Adaptive meshes

The software also offers the option of generating adaptive meshes instead of sparse meshes. With this option, nodes above or below a sub-mesh are selected, but with the value of y within the region's limits, they present dx equal to the present in the sub-mesh. Similarly, left or right nodes, but with x within the range, have the value of dx equal to the sub-mesh. This way, it is possible to form a region in the more refined mesh without changing its structure, as illustrated in Figure 4. During the generation of the meshes present in Figures 4a) and 4b), they were initially defined with $N_i = N_j = 50$ and $N_i = N_j = 100$ respectively. However, due to the process performed to increase the refinement in regions close to the tumour, they present a greater number of N_i and N_j.

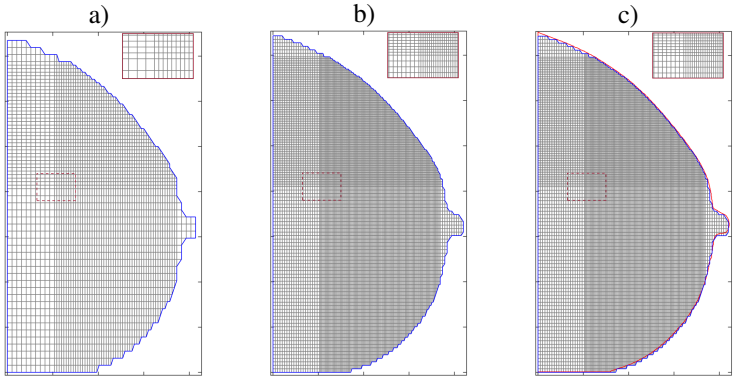

Figure 4. Examples of adaptive meshes: a) mesh with N_i =68 and N_j =76, b) and c) mesh with N_i =137 and N_j =152.

For its creation, two vectors vx and vy are generated from the information present in the definitions of each sub-mesh. These vectors contain the coordinates of all nodes in the mesh for each axis. Then, the same process is done as in Algorithm 1, with the difference that the approximation is made as shown in Algorithm 3.

Algorithm 3: getNode for adaptive mesh

Input: A point pt, two vectors containing the possible coordinates of the nodes in sequence vx and vy.
Output: A node coordinate no.

```
1  begin
2      for i ← 0 to size of vx − 2 do
3          if pt_x >= vx_i e pt_x < vx_{i+1} then
4              no_x = vx_i;
5              break;
6          end
7      end
8      if pt_x == vx_{size of vx−1} then
9          no_x = vx_{size of vx−1};
10     end
11     for i ← 0 to size of vy − 2 do
12         if pt_y >= vy_i e pt_y < vy_{i+1} then
13             no_y = vy_i;
14             break;
15         end
16     end
17     if pt_y == vy_{size of vy−1} then
18         no_y = vy_{size of vy−1};
19     end
20 end
```

Applying finite differences allows an area of the studied domain to present more significant refinements without needing implementation treatments, resulting in lower memory cost and execution time. On the other hand, due to the variation in the shape of the nodes, it is necessary to check the quality of mesh elements.

3 Results

The results of the tomography representation shown in Figure 1a) are presented using the developed software. For each result obtained, the value of the difference between the areas of the geometries is delimited by the given

contour and approximated in percentage, the number of nodes that form the given contour, and the number of nodes that are internal to the region obtained is informed. The internal meshes were obtained using the octave function *inpolygon* from the approximate contour of the image Eaton et al. [11]. The areas of the geometries were obtained by the Gauss Method, used to calculate the area of irregular polygons from the set of coordinates of the vertices of the polygon ordered in a counterclockwise direction. The method obtains the area as shown in Eq. (1) Braden [12]

$$A = \frac{1}{2}\left\{ \begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \ldots + \begin{vmatrix} x_{n-2} & x_{n-1} \\ y_{n-2} & y_{n-1} \end{vmatrix} + \begin{vmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{vmatrix} \right\}. \tag{1}$$

Table 1 shows the contour of extracted pixels and five refinements for each mesh model. Each refinement is defined by the values of N_i and N_j, and the sparse and adaptive meshes present greater refinement in the same regions as in Figures 3 and 4, with half the value of dx and dy. In the results of the sparse meshes, the column of N_i and N_j inform the values of the sub-mesh in parentheses. Similarly, the final number of N_i and N_j is shown for adaptive meshes. Comparisons are performed in relation to the contour of extracted pixels, which has an area = 2.2532×10^5 and 1852 points. Figure 5 shows graphs of the difference between the areas and the number of internal nodes presented in Table 1. The mesh approximation is in red, the sparse mesh is in blue and the adaptive mesh in black.

Table 1. Results obtained by generating meshes with various refinements for each model

$Ni = Nj$	Difference	Nodes on the contour	Internal nodes
mesh approximation			
50	0.8497	154	1616
75	0.5187	235	3756
100	0.5321	314	6774
150	0.3670	473	15490
200	0.3163	634	27749
300	0.2339	961	62931
sparse mesh approximation			
50(37x53)	0.6818	165	1306
75(55x79)	0.2410	248	3048
100(75x105)	0.2600	333	5509
150(111x157)	0.2092	504	12604
200(149x209)	0.1696	680	22597
300(223X313)	0.1130	981	51322
adaptive mesh approximation			
50(68x76)	0.5023	212	2985
75(102x114)	0.2253	317	6852
100(137x152)	0.2336	427	12417
150(205x228)	0.1645	643	28109
200(274x304)	0.1457	864	50165
300(411x456)	0.1003	1212	113650

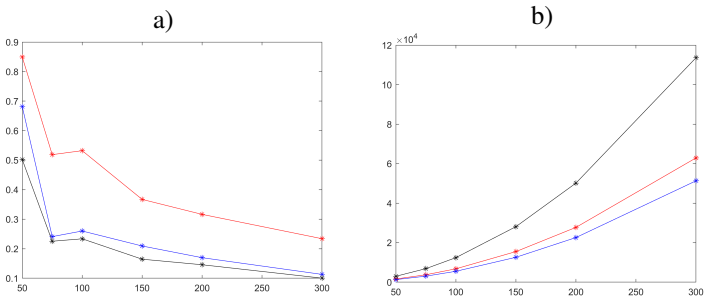

Figure 5. Gráficos ilustrando a evolução dos refinamentos: a) diferença entre as áreas, b) número de nós internos.

Observing Table 1 and Figure 5, it can be concluded that the software generates efficient representations of the mesh image, with an area difference smaller than 1% about the contour of pixels extracted from the image in the most diminutive refinements, reaching 0.23% in the largest. In addition, it can be seen that the sparse and adaptive mesh options allow greater refinement, concentrated in specific regions, since, even not affecting all meshes, they resulted in a greater number of internal nodes and a smaller area difference, reaching a value of 0.1% with these options selected on meshes with $N_i = N_j$ =300.

4 Conclusions

In this work, software was developed to generate meshes representing the contour of objects contained in figures, allowing the application of these data in numerical simulation techniques to solve differential equations. It can be seen from the results obtained that the meshes generated by the developed software satisfactorily describe the unstable region present in the image, Figure 1, with an area difference of less than 1% even in the most diminutive refinements.

Acknowledgements. The work of Silva, P. Z was partially supported by CNPq under the process 152547/2019-3. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] J. A. Cuminato and M. Meneguette. *Discretização de equações diferenciais parciais: técnicas de diferenças finitas*. Sociedade Brasileira de Matemática, 2013.
[2] P. F. S. Othechar. Analysis of finite difference numerical methods for solving the poisson equation in irregular domains (in portuguese). Master's thesis, Programa de Pós-graduação em Matemática Aplicada e Computacional da Universidade Estadual Paulista Júlio de Mesquita Filho,, Presidente Prudente, SP., Brazil, 2013.
[3] J. Fernández-Fidalgo, S. Clain, L. Ramírez, I. Colominas, and X. Nogueira. Very high-order method on immersed curved domains for finite difference schemes with regular cartesian grids. *Computer Methods in Applied Mechanics and Engineering*, vol. 360, pp. 112782, 2020.
[4] Z. Jomaa and C. Macaskill. The embedded finite difference method for the poisson equation in a domain with an irregular boundary and dirichlet boundary conditions. *Journal of Computational Physics*, vol. 202, n. 2, pp. 488 – 506, 2005.
[5] T. Fukuchi. Finite difference method and algebraic polynomial interpolation for numerically solving poisson's equation over arbitrary domains. *AIP Advances*, vol. 4, n. 6, pp. 060701, 2014.
[6] R. Codina and J. Baiges. Approximate imposition of boundary conditions in immersed boundary methods. *International Journal for Numerical Methods in Engineering*, vol. 80, n. 11, pp. 1379–1405, 2009.
[7] V. Rossum and Guido. *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.
[8] Weisstein and E. W. Moore neighborhood. `https://mathworld.wolfram.com/MooreNeighborhood.html`
[9] R. Casamaximo, N. Romeiro, Zaffalon da P. Silva, I. Souza, da J. Silva, P. L. Natti, and E. Cirilo. Algorithm for extracting points from images: irregular contours, 2021.
[10] Bradski, Gary, and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.
[11] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring. *GNU Octave version 5.1.0 manual: a high-level interactive language for numerical computations*, 2019.
[12] B. Braden. The surveyor's area formula. *The College Mathematics Journal*, vol. 17, n. 4, pp. 326–337, 1986.