Scientific Research Publishing

Applied Mathematics, 2026, 17(2), 97-102
https://www.scirp.org/journal/am
ISSN Online: 2152-7393
ISSN Print: 2152-7385

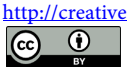
A Code for Generating Edge Points of a Plane Domain with Complex Geometric Shape

Gabyi Sewore[1], Ange Gar S. Nkokolo Massamba[2], Ndogotar Nélio[1], Benjamin Mampassi[3]

[1]Department of Mathematics and Informatic, University of Sarh, Sarh, Chad
[2]Department of Mathematics, Catholic University of West Africa, Bobo-Dioulasso, Burkina Faso
[3]Department of Mathematics and Informatics, University of Dakar, Dakar, Sénégal
Email: gabyisewore@yahoo.fr, massamba_ange@yahoo.fr, neliondogotar@gmail.com, mampassi@yahoo.fr

How to cite this paper: Sewore, G., Massamba, A.G.S.N., Nélio, N. and Mampassi, B. (2026) A Code for Generating Edge Points of a Plane Domain with Complex Geometric Shape. *Applied Mathematics*, **17**, 97-102.
https://doi.org/10.4236/am.2026.172007

Received: August 22, 2025
Accepted: February 3, 2026
Published: February 6, 2026



Abstract

In this paper, we set out to implement a code for processing the edges of a plane domain with a complex geometric shape. The algorithm that enabled us to write this code is based on the classical properties of analytic geometry. The coordinates of the vertices of the polygon-like domain must first be given. The code developed automatically generates indices for interior and edge points. Any point located at a distance defined by user from the edge of a domain is considered an edge point. To this condition, we add a second condition that requires the edge point to be inside the circle with a diameter of two consecutive vertices of the polygon. An illustration is included to clarify the readers of this paper.

Keywords

Complex Domain, MatLab Code, Onpolygon

1. Introduction

The physical problems encountered are generally governed by non-linear partial differential equations. The existence and uniqueness of solutions for these problems have been studied by Haim Brezis [1]. Available analytical methods are limited, that's why we have to resort to numerical methods based on algorithms. The pseudo-spectral method [2] [3] is an excellent methods for solving these problems, but it has the disadvantage to take into account edge conditions for plane domains with complex geometrical shapes. In response to this difficulty, we have developed a numerical code in MatLab called "**onpolygon**" to facilitate the treatment of the edges of a plane domain with a complex geometric shape. Note that similar work has been approached in our previous works [4] [5]. We can also cite

the work of Amann, who also approaches the problem of edges [6] [7]. This code is a contribution to the treatment of the edges of a plane domains with complex geometries shape, such as lakes, rivers, basins etc.

2. Theoretical Approach

We consider any point $[xP, yP]$ located at a distance ε from any segment of the polygon to be a point on the edge of the polygon. Let A_i, A_{i+1} two consecutive vertices of the polygon and P any point of the polygon, P is assimilated to a point of the segment $[A_iA_{i+1}]$ if the distance which separates it from the line (A_iA_{i+1}) is smaller or equal to ε and it belongs to the circle of diameter $[A_iA_{i+1}]$. We explain here how these two conditions can be translated analytically [4].

2.1. Distance from a Point to a Line Passing through Two Points

Let $A_i(x_i, y_i)$, $A_{i+1}(x_{i+1}, y_{i+1})$, $P(x_p, y_p)$ three given points, H the orthogonal project of P on the line of equation:

$$(D): \alpha x + \beta y + \gamma = 0$$

passing through the two points A_i, A_{i+1}. Thus, the distance from P to (D) is PH and is given by:

$$PH = \frac{\left|\alpha x_p + \beta y_p + \gamma\right|}{\left\|\boldsymbol{A}_i\boldsymbol{A}_{i+1}\right\|}$$

And yet we have:

$$\alpha = y_{i+1} - y_i$$

$$\beta = -\left(x_{i+1} - x_i\right)$$

$$\gamma = -x_i\left(y_{i+1} - y_i\right) + y_i\left(x_{i+1} - x_i\right)$$

This gives us $\alpha x_p + \beta y_p + \gamma = \det\left(\boldsymbol{A}_i\boldsymbol{P}, \boldsymbol{A}_i\boldsymbol{A}_{i+1}\right)$, and therefore

$$PH = \frac{\left|\det\left(\boldsymbol{A}_i\boldsymbol{P}, \boldsymbol{A}_i\boldsymbol{A}_{i+1}\right)\right|}{\left\|\boldsymbol{A}_i\boldsymbol{A}_{i+1}\right\|}$$

2.2. Points on the Segments

Consider N points $A_i, i = 1, \cdots, N$ and $P_j, j = 1, \cdots, m$ of any points in the plane represented by Figure 1 below.

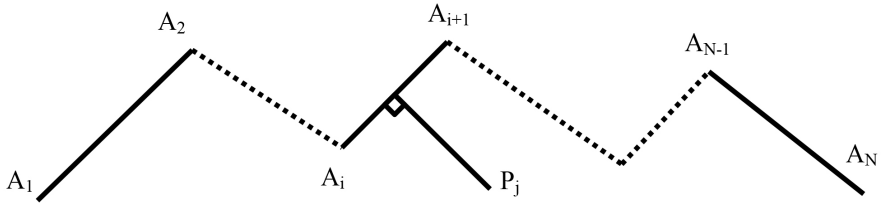


Figure 1. Illustrative diagram of the projection of any point in the plane onto a line segment.

We have $A_i(x_i, y_i)$, $P(x_{p_j}, y_{p_j})$, $\boldsymbol{A_iP_j} = \begin{pmatrix} x_{p_j} - x_i \\ y_{p_j} - y_i \end{pmatrix}$, and

$$\boldsymbol{A_iA_{i+1}} = \begin{pmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{pmatrix}$$

Let's put $X = (x_1, x_2, \cdots, x_N)^\top, Y = (y_1, y_2, \cdots, y_N)^\top$ the coordinates of segment ends and,

$$x_p = (x_{p_1}, x_{p_2}, \cdots, x_{p_m}), y_p = (y_{p_1}, y_{p_2}, \cdots, y_{p_m})$$

the coordinates of the points in the plane. For matrix programming, we consider the following matrices

$$Z_x = \begin{pmatrix} x_{p_1} - x_1 & x_{p_2} - x_1 & \cdots & x_{p_m} - x_1 \\ x_{p_1} - x_2 & x_{p_2} - x_2 & \cdots & x_{p_m} - x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_{p_1} - x_{N-1} & x_{p_2} - x_{N-1} & \cdots & x_{p_m} - x_{N-1} \end{pmatrix}$$

$$Z_y = \begin{pmatrix} y_{p_1} - y_1 & y_{p_2} - y_1 & \cdots & y_{p_m} - y_1 \\ y_{p_1} - y_2 & y_{p_2} - x_2 & \cdots & y_{p_m} - y_2 \\ \vdots & \vdots & \ddots & \vdots \\ y_{p_1} - y_{N-1} & y_{p_2} - y_{N-1} & \cdots & y_{p_m} - y_{N-1} \end{pmatrix}$$

In MatLab, these matrices are obtained using the following command lines:

```
>> Zx=repmat(xp,N-1,1)-repmat(x(1:end-1),1,m);
>> Zy=repmat(yp,N-1,1)-repmat(y(1:end-1),1,m);
```

The repmat command creates a large array in which each block is identical to the array passed as the first argument; the two other arguments respectively represent the number of times the argument-matrix is repeated according to the columns, respectively along the lines [5]. In order to define the conditions under which any point in the plane can be assimilated to a point on the segment, and above all to optimize our calculation codes, we need to set the following conditions

$$K_x = \begin{pmatrix} x_2 - x_1 \\ x_3 - x_2 \\ \vdots \\ x_N - x_{N-1} \end{pmatrix} \text{ et } K_y = \begin{pmatrix} y_2 - y_1 \\ y_3 - y_2 \\ \vdots \\ y_N - y_{N-1} \end{pmatrix}.$$

It follows from these notations that the determinant is calculated using the command line:

>> DET=Zx.*repmat(Ky,1,m)-Zy.*repmat(Kx,1,m) et a = sqrt(sum([Kx0;Ky0].2))

The vector d of distances is then obtained using the following MatLab command:

$$>> \text{d} = \text{DET.} / \text{repmat(a ',1,m)} \quad (1)$$

The condition for any point P_j in the plane to be considered a point or infinitely close to the line D passing through the points A_i, A_{i+1} is given by:

$$>> \mathrm{d} = \mathrm{DET}./\mathrm{repmat}(\mathrm{a}',1,\mathrm{m}) < \varepsilon \tag{2}$$

To this condition, we add the condition for this point to be in the circle of diameter $[A_iA_{i+1}]$. Simply put:

$$X_0 = \left(\frac{x_2+x_1}{2}, \frac{x_3+x_2}{2}, \cdots, \frac{x_N+x_{N-1}}{2}\right)^{\mathrm{T}}$$

$$Y_0 = \left(\frac{y_2+y_1}{2}, \frac{y_3+y_2}{2}, \cdots, \frac{y_N+y_{N-1}}{2}\right)^{\mathrm{T}}$$

$$H_x = \begin{pmatrix} x_{p_1} - \frac{x_2+x_1}{2} & x_{p_2} - \frac{x_2+x_1}{2} & \cdots & x_{p_m} - \frac{x_2+x_1}{2} \\ x_{p_1} - \frac{x_3+x_2}{2} & x_{p_2} - \frac{x_3+x_2}{2} & \cdots & x_{p_m} - \frac{x_3+x_2}{2} \\ \vdots & \vdots & & \vdots \\ x_{p_1} - \frac{x_N+x_{N-1}}{2} & x_{p_2} - \frac{x_N+x_{N-1}}{2} & \cdots & x_{p_m} - \frac{x_N+x_{N-1}}{2} \end{pmatrix}$$

$$H_y = \begin{pmatrix} y_{p_1} - \frac{y_2+y_1}{2} & y_{p_2} - \frac{y_2+y_1}{2} & \cdots & y_{p_m} - \frac{y_2+y_1}{2} \\ y_{p_1} - \frac{y_3+y_2}{2} & y_{p_2} - \frac{y_3+y_2}{2} & \cdots & y_{p_m} - \frac{y_3+y_2}{2} \\ \vdots & \vdots & & \vdots \\ y_{p_1} - \frac{y_N+y_{N-1}}{2} & y_{p_2} - \frac{y_N+y_{N-1}}{2} & \cdots & y_{p_m} - \frac{y_N+y_{N-1}}{2} \end{pmatrix}$$

In MatLab, these matrices can be written using command lines:

```
>>X0=(X(1:end-1)+X(2:end))/2;
>>Y0=(Y(1:end-1)+Y(2:end))/2;
>>Hx=repmat(xP,N-1,1)-repmat(X0,1,m);
>>Hy=repmat(yP,N-1,1)-repmat(Y0,1,m);
```

Posing $b = \left\| \left[Kx^{\mathrm{T}}; Ky^{\mathrm{T}} \right] \right\|^2$, then the condition for a point P_j in the circle of diameter $[A_iA_{i+1}]$ is written:

$$C = Hx \cdot Hx + Hy \cdot Hy - \frac{\mathrm{repmat}\left(b^{\mathrm{T}}, 1, m\right)}{4} \leq 0. \tag{3}$$

This is a component-by-component multiplication and b^{T} is the transpose of b. This translates into MatLab language by:

```
>>  C=Hx.^2+Hy.^2-repmat(b',1,m)/4 <=0
```

2.3. Presentation of Developed Code

For a given set of domain points, this code identifies which are on the edge and which are not. The developed code is as follows:

```
function [IDon,IDout]=onpolygon(xP,yP,X,Y,EPS)
% resizing X and Y
if size(X,1)==1,X=X';Y=Y';end
% resizing xP and yP
if size(xP,2)==1,xP=xP';yP=yP';end
% Calculating the components of vectors A_iP_j
N=length(X);m=length(xP);
Zx=repmat(xP,N-1,1)-repmat(X(1:end-1),1,m);
Zy=repmat(yP,N-1,1)-repmat(Y(1:end-1),1,m);
%Calculation of the matrix representing the vectors A_iA_{i+1}
Kx=X(2:end)-X(1:end-1);Ky=Y(2:end)-Y(1:end-1);
% Determinant
DET=Zx.*repmat(Ky,1,m)-Zy.*repmat(Kx,1,m);
% Norm
a=sqrt(sum([Kx';Ky'].^2));
% Distances to the polygon
DIST=abs(DET)./repmat(a',1,m);
% Condition on point inside circle of diameter the segment
X0=(X(1:end-1)+X(2:end))/2;
Y0=(Y(1:end-1)+Y(2:end))/2;
Hx=repmat(xP,N-1,1)-repmat(X0,1,m);
Hy=repmat(yP,N-1,1)-repmat(Y0,1,m);
b=sum([Kx';Ky'].^2);
C=Hx.^2+Hy.^2-repmat(b',1,m)/4;
% Locating points relative to the edge
% +0 changes the boolean to double float
L=prod((DIST>EPS | C >0)+0);
IDon=find(L==0); IDout=find(L~=0);
```

Executing this code is very simple. Simply give [X, Y] the vertices coordinates of the complex domain assimilated to a polygon and [Xp, Yp] any point of this domain. The code we've developed automatically generates the indices of the edge points, enabling us to extract the points we're looking for.

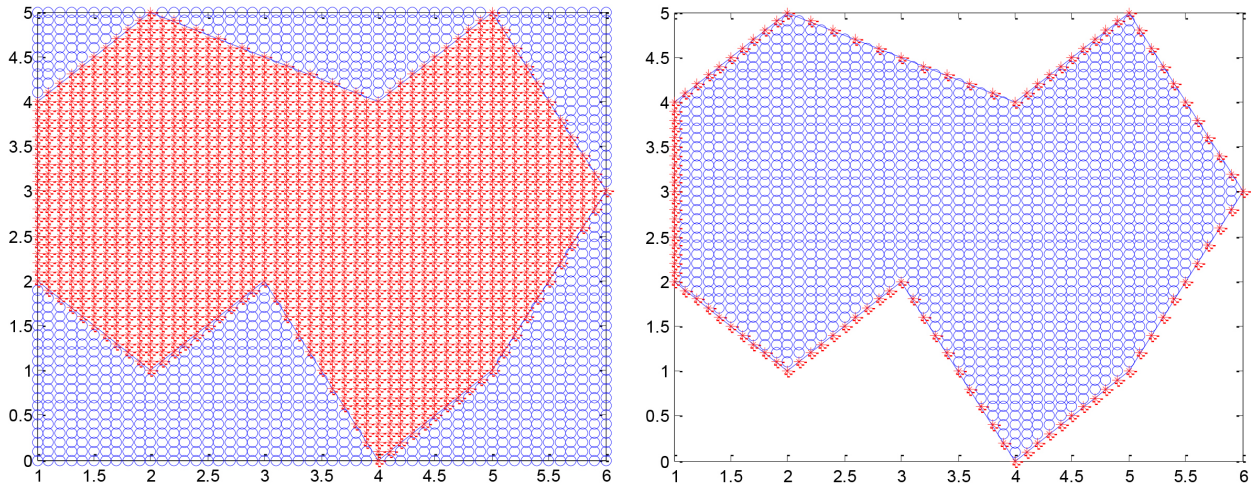


Figure 2. On the left, the exterior points and the points of the non-tensor domain obtained by MatLab's integrated code inpolygon and on the right, the interior and the edge points of the non-tensor domain obtained by onpolygon code.

3. Illustration of Developed Code

An illustration of this code is shown in the figure below. In the following illustration, we consider a polygon with vertices (2, 1); (0, 2); (1, 4); (2, 5); (4, 4); (5, 5); (6, 3); (5, 1); (4, 0) and (3, 2). First, we use the MatLab's integrated code **inpolygon**, which distinguishes between the points of the polygon and the external points (see Figure 2 in left). The points within the domain are shown in red and the external points in blue. Next, we apply our code **onpolygon** to identify the interior and the edge points. These points are shown in red and blue, respectively (see Figure 2 in right).

4. Conclusion and Remarks

This code extracts the indices of edge and interior points. It's a complement of MatLab's integrated code "**inpolygon**", which enables edge conditions of either Dirichlet or Neumann type to be handled. The difference between our developed code and MatLab's integrated code is that our code determines the indices of the interior and the edge points, whereas MatLab's integrated code gives the indices of the polygon points (including inner points and edges) and the indices of the outer points. The combination of these two codes generates complete indices for interior, edge and exterior points.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Brézis, H. (2005) Analyse fonctionnelle: Théorie et application. Dunod.

[2] Mercier, B. (1989) An Introduction to the Numerical Analysis of Spectral Methods (Lecture Notes in Physics). Springer, 318 p.

[3] Bengt, F. (1998) A Practical Guide to Pseudospectral Methods. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press.

[4] Seworé, G. (2014) Développement des codes numériques adaptés aux schémas pseudo-spectraux pour les équations aux dérivées partielles paraboliques non linéaires. Thése de Doctorat, Université Cheikh Anta Diop de Dakar.

[5] Hilaire Nkounkou, A., Traore, S., Gabyi, M., Abani, A. and Mampassi, B. (2011) Spectral Differentiation on Unstructured Meshes Using Jacobi Gauss-Lobatto Points. *Far East Journal of Applied Mathematics*, **59**, 105-1221.

[6] Amann, H. (1986) Parabolic Evolution Equations with Nonlinear Boundary Conditions. *Proceedings of a Symposium in Pure Mathematics of the American mathematical Society*, **45**, 17-27.

[7] Amann, H. (1986) Quasilinear Evolution Equations and Parabolic Systems. *Transactions of the American Mathematical Society*, **293**, 191-227. https://doi.org/10.1090/S0002-9947-1986-0814920-4