Available online at https://ganitjournal.bdmathsociety.org/

GANIT: Journal of Bangladesh Mathematical Society

GANIT*J. Bangladesh Math. Soc.* 43.1 (2023) 17–35
DOI: https://doi.org/10.3329/ganit.v43i1.67856

Auto Mesh generation algorithm for the convex domain with the triangular elements

Syeda Sabikun Nahar[a], Md. Sadekur Rahman [a], Md. Shajedul Karim [a,*]

[a]*Department of Mathematics, Shahjalal University of Science and Technology,Sylhet, Bangladesh*

ABSTRACT

Mesh creation is one of the primary tasks in implementing the Finite Element Method (FEM) to solve two- and three-dimensional boundary value problems. However, the whole procedure becomes tedious and problematic when higher-order finite elements are employed to construct mesh and prepare corresponding element data.
In this study, we strive to develop a versatile algorithm for discretizing the two-dimensional domain using linear, quadratic, and cubic triangular finite elements. The algorithm is developed based on n vertices (actual or more in number) that constitute the boundary of the domain, along with a computed (n+1)-th point such as the centroid. The algorithm, using this input, generates an initial mesh consisting of n triangular elements. Then, in every subsequent step, the algorithm increases the number of triangles in the mesh fourfold compared to the previous step.
The algorithm we present here can employ (a) straight-sided (linear, quadratic, and cubic) triangular elements to generate the mesh for domains with polygonal boundaries and (b) both straight-sided and curved (quadratic and cubic) triangular elements with two straight sides and one curved side to generate meshes for domains with curved boundaries.
Thus, the algorithm generates the desired meshes if the vertices are specified once at the initial step. Additionally, the inclusion of the mathematical expression of the curved boundary enables the algorithm to generate the fine mesh for the curved domain utilizing higher-order curved and straight-sided triangular elements. We also present a computer code in MATLAB incorporating the algorithm to create the mesh, prepare the element's data, and determine the element's connectivity.

Received: April 26, 2023 **Accepted:** May 25, 2023 **Published Online:** July 15, 2023

Keywords: Auto Mesh; convex; polygon; curve; boundaries; code

1. Introduction

The finite element method (FEM) is a powerful numerical technique widely implemented as an analytical tool in many branches of science and engineering. The versatility and popularity of the method are well established. An excellent overview of the method's utility for one-dimensional problems, often encountered in the real world, can be readily seen

*Corresponding Author. *Email Address:*msk-mat@sust.edu

in [1–2]. In applied mathematics, for two-dimensional problems, researchers thoroughly investigated the stability, discretization errors, and convergence rates of the FEM. It has been a very active field of research ever since. The standard way, in FEM, to achieve the desired accuracy of the approximate solution is to refine the triangulation of the computational domain, which introduces more degrees of freedom. A vast number of publications deal with this so-called h-version. An alternative way of increasing the number of unknowns is to increase the polynomial degree of finite elements. This alternative way is commonly known as the p-version, which is less common, and its convergence speed strongly depends on the regularity of the solution to the PDE [3]. The FEA solution procedure comprises three phases: domain discretization, equation solving, and error analysis. The mesh generation of the domain is a prime task for the pre-processing phase and plays a vital role in obtaining accurate solutions.

A fine mesh of the domain in the FEM solution procedure is one of the main ingredients for obtaining the solutions with the desired accuracy. The versatile mesh generator scheme and the mesh adaptive technique or procedure now turns out to be the prime requirements for creating such adequate meshes. Consequently, automatic mesh generation has received much attention to minimize manual intervention, improve mesh quality, and obtain more efficient procedures. Moreover, unstructured mesh generation methodologies are the predominant technique due to their ability to model geometrically complex designs. Since these are the natural environments for adaptivity, they may be the only hope for resolving very small-scale features. Most of the FEM and FVM codes use unstructured triangulations due to their geometrical flexibility and the low cost of the linear triangular elements [3–5].

Recent studies [3, 6] have discussed and provided detailed information on the necessity of automatic mesh generation as well as the advantages, disadvantages, and mesh refinement strategies. In [6], the mesh generation scheme is developed based on the following steps: (a) First of all, triangulate the linear convex polygonal domain; (b) divide each triangle into m^2 smaller triangles by dividing each of its sides into m equal parts; and finally, (c) each triangle into three special quadrilaterals and further each quadrilateral into 4 (four) quadrilaterals. In [3], an automatic triangular mesh generator has been developed utilizing the advancing front technique as the prime or main building block of a computational system for mesh generation that builds triangular and quadrilateral meshes over arbitrary domains. More specifically, this technique generates a quadrilateral mesh from the created triangular mesh using the specific process of splitting each triangle into quadrilaterals.

A general algorithm and computer code have been presented in [4] for creating h- and p-version meshes directly using quadrilaterals for a convex domain with a polygonal or curved boundary. The (developed) computer code also allows the generation of the p-version meshes using both the Lagrange and serendipity type (higher-order) quadrilateral elements. We believe that the computer programs developed in [3, 4, 6] can be implemented to automatically create meshes of the domain, prepare element data, and display corresponding quadrilateral meshes.

The unstructured triangulations, as mentioned above, due to their geometrical flexibility and the low cost of the linear triangular elements, are utilized almost in all the commercial FEM and FVM codes. So, it is clearly apparent that the automatic mesh generator uses linear triangular elements to generate only the h-version meshes. Usually, in practical situations, numerous linear triangular elements are required to mesh the domain, resulting in triangles with smaller areas.

Then, at this stage, a genuine question arises as to whether it is logical to split each of these smaller triangles into three quadrilaterals and then further split each quadrilateral into four quadrilaterals. The only possible answer to this genuine question is to triangulate the domain with fewer triangular elements and then again split each of these triangles into three quadrilaterals. Therefore, this entails attention to generating the refined mesh and causes difficulties for automatic mesh generation. Instead, a better possible alternative procedure is to generate first a p-version mesh for the domain with fewer triangular elements and then split these triangles further with the triangles if necessary.

The suitability, geometrical flexibility, advantages, disadvantages, and usefulness of the triangular elements (both lower and higher order) have been extensively explained in detail in several studies, and for ready references, one can turn to [7–10]. Integration schemes (analytical and numerical) for the triangular domain integrals have been developed in [8, 10-13]. Still, research is going on to mitigate the drawbacks or disadvantages relevant to the p-version triangular mesh generations. On the same line, we have attempted to develop a general algorithm and a complete computer code for automatic (h- and p-versions) mesh generation using triangular elements for the convex domain having polygonal or curved boundaries.

This piece of writing is structured as follows: Section 2 contains (1) the algorithm for triangulating complex domains with polygonal and non-polygonal boundaries; (2) code-generated meshes and element data outputs that are suitable for human verifications; and (3) meshes with more (linear, quadratic, and cubic) triangular elements. Necessary and important conclusions are discussed concisely in Section 3. Finally, references are listed, and the developed computer code is provided as an appendix.

2. Triangulation of the convex domain with polygonal and non-polygonal boundaries

In this section, we wish to present an algorithm and code for generating a mesh of a convex domain with (a) a

polygonal boundary using the actual number of vertices on the boundary and (b) a non-polygonal boundary using more nodes on the curve boundary either by selecting more nodes on the curve or computing the required nodal coordinates using the curve's equation. The code needs three inputs: n (number of vertices), coordinates of vertices, and the order of triangular elements.
With these given inputs, the program creates an initial mesh consisting of n triangles. Subsequently, if desired, each triangle will be divided into four triangles of equal area. So, this quadruples the number of triangular elements in each subsequent step. The output will be an array of the relations between local and global nodes and another of nodal coordinates. These outputs are indeed very essential element data needed to compute the components of element matrices and assemble element matrices to form the global system of equations.

2.1 Algorithm

For the known values of **N** (the number of vertices), **Nodes** (nodal coordinates of the vertices), **NE** (number of elements), and **OE** (the order of element under consideration) desired mesh will be generated by the following algorithm. The algorithm will generate two dynamic arrays **Tri** (for element-wise relation between the local and global nodes) of size and **Nodes** (coordinates of global nodes of the mesh) of size for outputs of essential data.
Step 1: Calculate (**N+1**)th node by arithmetic mean of coordinates of **N** vertices.
Step 2: Use the procedure mesh1 (Tri, Nodes, NE, n, curve, cs)
If **OE**=2 use procedure mesh2, if **OE**=3 use procedure mesh3
Step 3: Print **Tri** and **Nodes** array
Step 4: Draw the created mesh of the domain by using data of **Tri** and **Nodes** arrays.
Procedure mesh1(Tri, Nodes, NE, n, curve, cs):
Step 1: Compute **noTri** (row of dynamic **Tri** array) and **LastNode (**row number of dynamic array **Nodes)**
Step 2: Repeat step 3 to 5 for **I**=1 to **noTri**
Step 3: Calculate **I**th mid nodes for **Tri** and compare with coordinates of **Nodes** array.
Step 4: Calculate nodal coordinates without repetition that is for non-repeating case increase the index of array **Nodes** by one.
Step 5: based on calculated nodes spilt **I**th **Tri** element into four elements.
Step 6: If row of **Tri** is less than **NE** go to step 1
Step 7: Print **Tri** and **Nodes** arrays
Step 8: Draw the figure of the discretized domain

Procedures **mesh2** and **mesh3** are same to the procedure **mesh1** and hence their steps are not written. Based on the above algorithm “mesh1.m”, "mesh2.m" and "mesh3.m", three separate independent function programs have been developed for calling them by the main function "twoDimMesh.m". The computer code in MATLAB (appended in the appendix) is self-explanatory, and therefore we avoided duplications of mathematical expressions.

2.2 Test Examples for automatic mesh generation of polygonal domain

For easy (manual) verification and clear understanding, we consider a simple polygonal domain with vertices. We wish to discretize the domain by using linear, quadratic, and cubic triangular elements.

2.2.1 Mesh using linear triangular elements

Case 1: Discretizing the domain with vertices $(0,0),(2,0),(1,2)$. NE=3 and OE=1

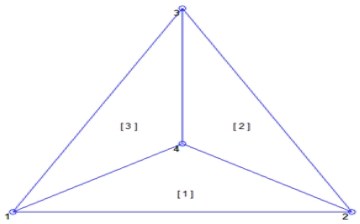


Figure 2.1: Mesh of the domain using 3-linear triangular elements

Output of the program

Figure 2.1 displays the mesh of a triangular domain, consisting of three linear triangle elements. The mesh was created using the code developed in accordance with the aforementioned algorithm. The computed element data for this mesh is

presented below:

(a) Element wise Global Nodes for element [e]:

[1] : 1, 2, 4	[2] : 2, 3, 4	[3] : 3, 1, 4

(b) Nodal Coordinates for global nodes:

1 : (0.000, 0.000)	2 : (2.000, 0.000)	3 : (1.000, 2.000)	4 : (1.000, 0.667)

Case 2: Discretizing the domain with vertices $(0,0), (2,0), (1,2)$. NE=12 and OE=1

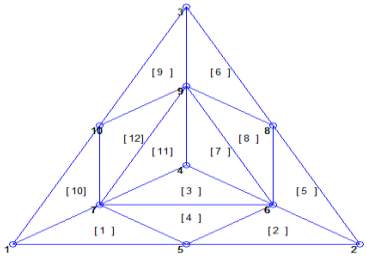


Figure 2.2: Mesh of the domain using 12-linear triangular elements

Output of the program:

Figure 2.2 showcases the mesh of a triangular domain comprising 12 linear triangular elements. The mesh was generated using the developed code. We found the other computed element data to be accurate; it is not explicitly listed in tabular form.

2.2.2 Mesh using quadratic triangular elements

Case 1: Discretizing the domain with vertices $(0,0), (2,0), (1,2)$. NE=3 and OE=2

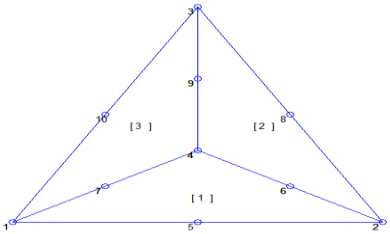


Figure 2.3: Mesh of the domain using 3 quadratic triangular elements

Output of the code

Figure 2.3 displaces the mesh of a triangular domain consisting of three quadratic triangular elements. This mesh was generated by the code developed in accordance with the aforesaid algorithm. The computed element data for this mesh is as follows:

(a) Element wise Global Nodes for element [e]:

[1]: 1, 5, 2, 6, 4, 7	[2]: 2, 8, ,3 9, 4, 6	[3]: 3, 10, 1, 7, 4, 9

(a) Nodal Coordinates for global nodes:

1 : (0.000, 0.000)	2 : (2.000, 0.000)	3 : (1.000, 2.000)	4 : (1.000, 0.667)	5 : (1.000, 0.000)
6 : (1.500, 0.333)	7 : (0.500, 0.333)	8 : (1.500, 1.000)	9 : (1.000, 1.333)	10 : (0.500, 1.000)

Case 2: Discretizing the domain with vertices $(0,0), (2,0), (1,2)$. NE=12 and OE=2

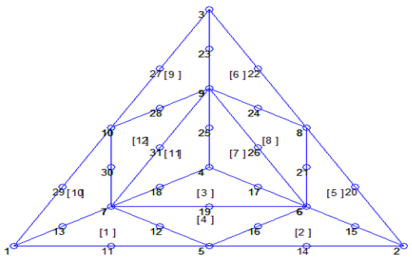


Figure 2.4: Mesh of the domain using 12 quadratic triangular elements

Output of the code:

The mesh of a triangular domain consisting of 12 quadratic triangular elements is shown in Figure 2.4. The mesh is generated using code developed in accordance with the aforementioned algorithm. The computed element data were verified and found correct, but they are not tabulated.

2.2.3 Mesh using cubic triangular elements

Case 1: Discretizing the domain with vertices $(0, 0), (2, 0), (1, 2)$. NE=1 and OE=1

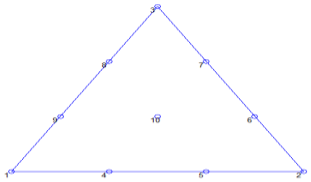


Figure 2.5: Mesh of the domain using one cubic triangular element

Output of the code:

Figure 2.5 presents the mesh of a triangular domain with one cubic triangular element. The mesh is constructed by the developed code, and the computed element data are as follows:

(a) Element wise Global Nodes for element [e]:
[1] : 1, 4, 5, 2, 6, 7, 3, 8, 9, 10

(b) Nodal Coordinates for global nodes:

1 : (0.000, 0.000)	2 : (2.000, 0.000)	3 : (1.000, 2.000)	4 : (0.667, 0.000)	5 : (1.333, 0.000)
6 : (1.667, 0.667)	7 : (1.333, 1.333)	8 : (0.667, 1.333)	9 : (0.333, 0.667)	10 : (1.000, 0.667)

Case 2: Discretizing the domain with vertices (0, 0), (2, 0), (1, 2). NE=3 and OE=3

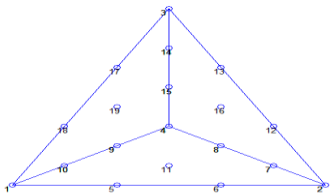


Figure 2.6: Mesh of the domain using 3 cubic triangular elements

Output of the code:

Figure 2.6 displaces the mesh of a triangular domain with one cubic triangular element. This mesh is created by the developed code, and the computed element data are as follows:

(b) Element wise Global Nodes for element [e]:
[1] : 1, 5, 6, 2, 7, 8, 4, 9, 10, 11
[2] : 2, 12, 13, 3, 14, 15, 4, 8, 7, 16

[3] : 3, 17, 18, 1, 10, 9, 4, 15, 14, 19

(c) Nodal Coordinates for global nodes:

1 : (0.000, 0.000)	2 : (2.000, 0.000)	3 : (1.000, 2.000)	4 : (1.000, 0.667)	5 : (0.667, 0.000)
6 : (1.333, 0.000)	7 : (1.667, 0.222)	8 : (1.333, 0.444)	9 : (0.667, 0.444)	10 : (0.333, 0.222)
11 : (1.000, 0.222)	12 : (1.667, 0.667)	13 : (1.333, 1.333)	13 : (1.333, 1.333)	15 : (1.000, 1.111)
16 : (1.333, 0.889)	17 : (0.667, 1.333)	18 : (0.333, 0.667)	19 : (0.667, 0.889)	

Case 3: Discretizing the domain with vertices (0, 0), (2, 0), (1, 2). NE=12 and OE=3

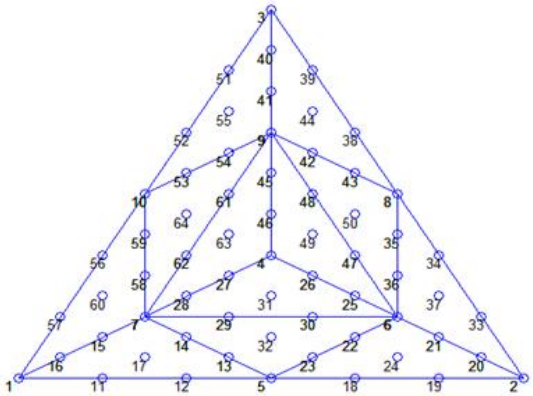


Figure 2.7: Mesh of the domain using 12 cubic triangular elements

Output of the code:

Figure 2.7 exhibits the mesh of a triangular domain with 12 cubic triangular elements. This mesh is generated by the developed code. We verified all the computed element data and found it correct. This is an illustration of a p-version mesh employing cubic triangular elements. There are 12 triangular elements of the same size, e.g., triangles $T_{157,}$ $T_{526,}$ T_{567}, T_{647}, $T_{286,}$ $T_{689,}$ T_{839}, T_{694}, and other triangles within the triangle T_{143} are all of equal size in area, and the area of each triangle is equal to one-twelfth of the area of T_{123}.

2.3 Test Examples for automatic mesh generation of non-polygonal domain

We consider here the domain, for clear understanding and easy manual verification, with one curved side described by $x^2 + y^2 = 1,\ x \geq 0,\ y \geq 0$ to generate a fine mesh using the code (appended in the appendix). To describe the domain, we give the input of eight vertices, where most of the vertices lying on the curved side, which are:

$$(0,0), (0.5,0), (1,0), (0.938, 0.348), (0.688, 0.726), (0.375, 0.927), (0,1), (0, 0.5)$$

We wish to discretize the domain by using linear, quadratic, and cubic triangular elements.

2.3.1 Mesh with linear triangular elements

Case 1: Discretizing the domain with vertices $(0,0), (0.5,0), (1,0), (0.938, 0.348), (0.688, 0.726), (0.375, 0.927), (0,1), (0, 0.5)$, NE=8 and OE=1.

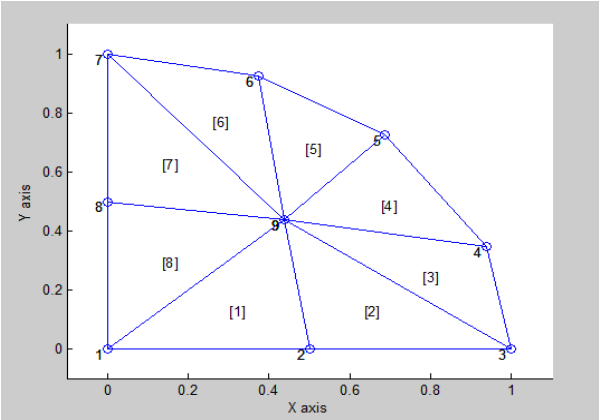


Figure 2.8: Mesh of the domain using 8 linear triangular elements

Output of the code

In Figure 2.8, we present a mesh of a triangular domain with eight linear triangular elements. Notice that the domain has a curved boundary. This mesh is generated by the developed code, and the computed element data are as follows:

(a) Element wise Global Nodes for element [e]:

[1] : 1, 2, 9	[2] : 2, 3, 9	[3] : 3, 4, 9	[4] : 4, 5, 9
[5] : 5, 6, 9	[6] : 6, 7, 9	[7] : 7, 8, 9	[8] : 8, 1, 9

(b) Nodal Coordinates of global nodes:

1 : (0.000, 0.000)	2 : (0.500, 0.000)	3 : (1.000, 0.000)
4 : (0.938, 0.348)	5 : (0.688, 0.726)	6 : (0.375, 0.927)
7 : (0.000, 1.000)	8 : (0.000, 0.500)	9 : (0.438, 0.438)

Case 2: Discretizing the domain with vertices $(0,0),(0.5,0),\ (1,0),(\ 0.938,0.348),(\ 0.688,\ 0.726),\ (\ 0.375,0.927),$ $(0\ ,1),\ (0,0.5)$ NE=32 and OE=1

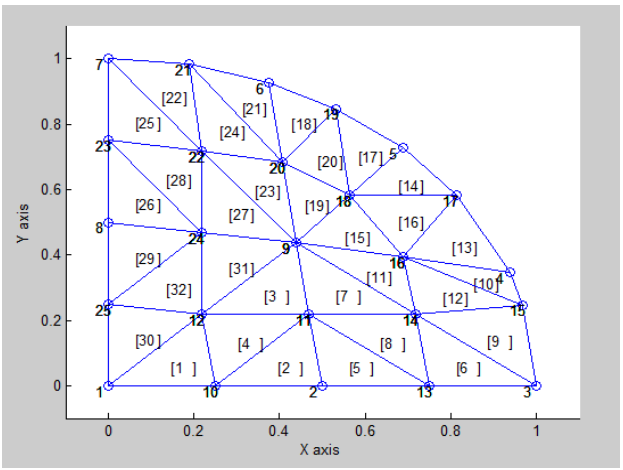


Figure 2.9: Mesh of the domain using 32 linear triangular elements

Output of the code

In Figure 2.9, we present the mesh of a curved triangular domain with 32 linear triangular elements. This mesh is created by the developed code appended in the appendix. The computed element data is as follows:

(a) Element wise Global Nodes for element [e]:

[1] : 1, 10, 12	[2] : 10, 2, 11	[3] : 11, 9, 12	[4] : 10, 11, 12
[5] : 2, 13, 11	[6] : 13, 3, 14	[7] : 14, 9, 11	[8] : 13, 14, 11
[9] : 3, 15, 14	[10] : 15, 4, 16	[11] : 16, 9, 14	[12] : 15, 16, 14
[13] : 4, 17, 16	[14] : 17, 5, 18	[15] : 18, 9, 16	[16] : 17, 18, 16
[17] : 5, 19, 18	[18] : 19, 6, 20	[19] : 20, 9, 18	[20] : 19, 20, 18
[21] : 6, 21, 20	[22] : 21, 7, 22	[23] : 22 , 9, 20	[24] : 21, 22, 20
[25] : 7, 23, 22	[26] : 23, 8, 24	[27] : 24, 9, 22	[28] : 23, 24, 22
[29] : 8, 25, 24	[30] : 25, 1, 12	[31] : 12, 9, 24	[32] : 25, 12, 24

(a) Nodal Coordinates of global nodes:

1 : (0.000, 0.000)	2 : (0.500, 0.000)	3 : (1.000, 0.000)	4 : (0.938, 0.348)	5 : (0.688, 0.726)
6 : (0.375, 0.927)	7 : (0.000, 1.000)	8 : (0.000, 0.500)	9 : (0.438, 0.438)	10 : (0.250, 0.000)
11 : (0.469, 0.219)	12 : (0.219, 0.219)	13: (0.750, 0.000)	14 : (0.719, 0.219)	15 : (0.969, 0.247)
16 : (0.688, 0.393)	17 : (0.813, 0.582)	18 : (0.563, 0.582)	19 : (0.531, 0.847)	20 : (0.406, 0.682)
21 : (0.188, 0.982)	22 : (0.219, 0.719)	23 : (0.000, 0.750)	24 : (0.219, 0.469)	25 : (0.000, 0.250)

2.3.2 Mesh with quadratic elements

Case 1: Discretizing the domain with vertices $(0,0), (0.5,0), (1,0), (0.938, 0.348), (0.688, 0.726), (0.375, 0.927), (0,1), (0,0.5)$, NE=8 and OE=2.

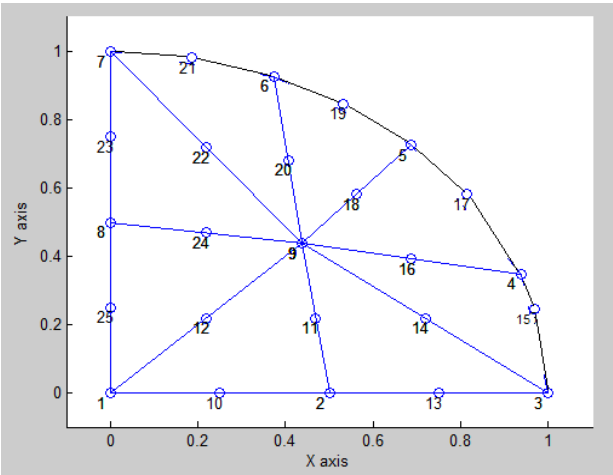


Figure 2.10: Problem domain meshed with 8 quadratic triangular elements

The mesh in Figure 2.10 illustrates the importance of a higher-order (quadratic) triangular element's implementation to create a fine mesh of a curved domain, reducing geometrical errors. This mesh (in Figure 2.10), consisting only of 8 quadratic triangular elements, is better than the mesh (in Figure 2.9), created using 32 linear triangular elements.

2.3.3 Mesh with cubic elements:

Discretizing the domain with vertices $(0,0), (0.5,0), (1,0), (0.938, 0.348), (0.688, 0.726), (0.375, 0.927), (0,1), (0, 0.5)$, NE=8 and OE=3.

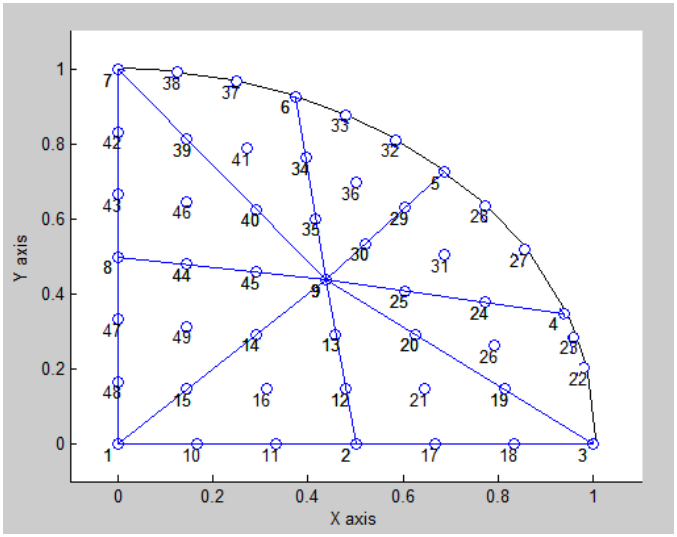


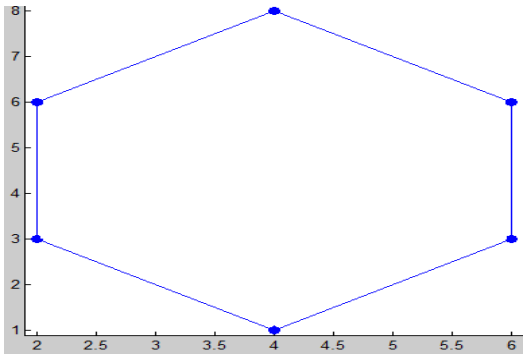
Figure 2.11: Mesh of the domain using 8 cubic triangular elements

The mesh in Figure 2.11 illustrates the importance of a higher-order (cubic) triangular element's employment to create a fine mesh of a curved domain, reducing geometrical errors. This mesh (in Figure 2.11) consisting only of 8 cubic triangular elements is better than the meshes (in Figures 2.9-2.10) created using 32 linear and 8 quadratic triangular elements, respectively.

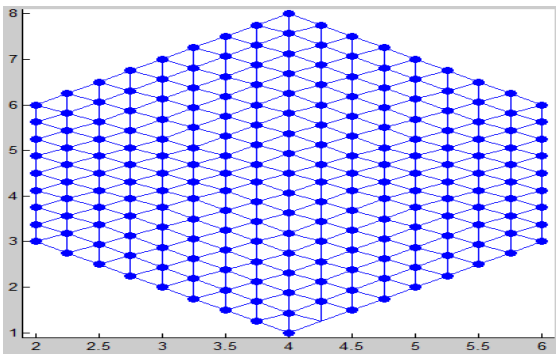
2.4 Test Examples for automatic mesh generation of polygonal and non-polygonal domains with more elements

In the above section, we have shown meshes with a minimum number of elements. In a practical situation, numerous triangular elements are used in h- and p-version meshes. So, in this section, we wish to present only the (code generated) meshes for some convex domains with more linear, quadratic, and cubic triangular elements. We have generated fine meshes but did not include them in this article due to their visibility in printed form. The code generates meshes as desired or required.

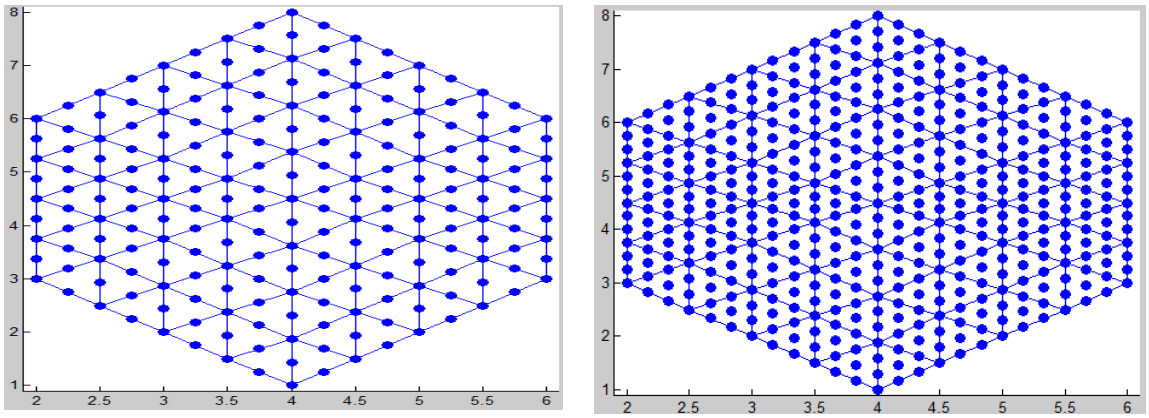
Example-1: Mesh of hexagonal domain



(a)



(b)

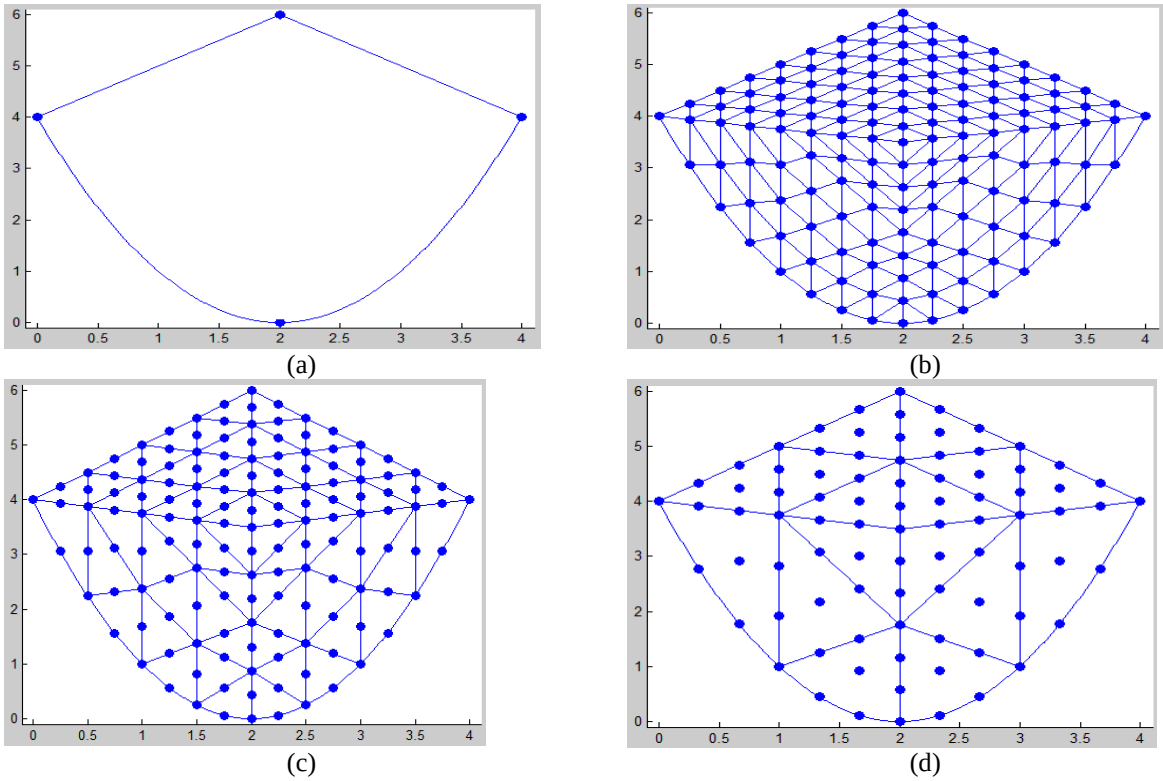


(c) (d)

Figure 2.12: Mesh of a covex polygon; (a) domain to be meshed, (b) Mesh using 384 linear elements, (c) Mesh using 96 quadratic elements, (d) Mesh using 96 cubic elements.

The domain (shown in Figure 2.12(a)) is created by entering the coordinates of six vertices. The code then generates meshes (shown in Figures 2.12(b)-2.12(d)) based on the order of the elements (2 for quadratic and 3 for cubic).

Example-2: Mesh of non-polygonal domain with one parabolic curve side



(a) (b)

(c) (d)

Figure 2.13: Mesh of a non-polygonal domain; (a) domain to be meshed, (b) 256 linear elements, (c) 64 quadratic elements, (d) 16 cubic elements

The domain (shown in Figure 2.13(a)) is produced by first incorporating the mathematical expression of the curve and then by inputting the coordinates of four vertices. The algorithm then generates meshes (shown in Figures 2.13(b)-2.13(d)) based on the order of the elements (2 for quadratic and 3 for cubic). Notice that the domain contains two straight sides and one curved side. Then also, the mesh using a few higher-order elements is free from geometrical errors.

Example-3: Mesh of non-polygonal domain

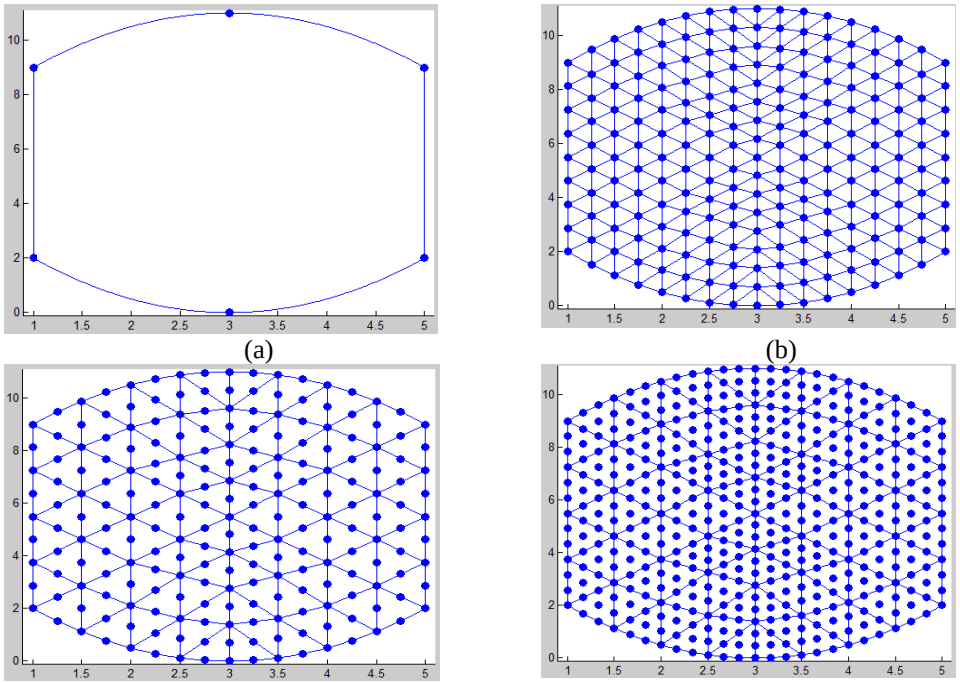


Figure 2.14: Mesh of a domain bounded by two curved(parabolic) side and two staright side; (a) domain to be meshed, (b) Mesh using 384 linear elements, (c) Mesh using 96 quadratic elements, (d) Mesh using 96 cubic elements

The domain (shown in Figure 2.14(a)) is generated by incorporating two mathematical equations for two curved sides into the code. Then, after inputting the coordinates of six vertices; the code then generates meshes (shown in Figures 2.14(b)–2.14(d)) based on the order of the elements (2 for quadratic and 3 for cubic).

Example-4: Mesh of non-polygonal (curved) domain

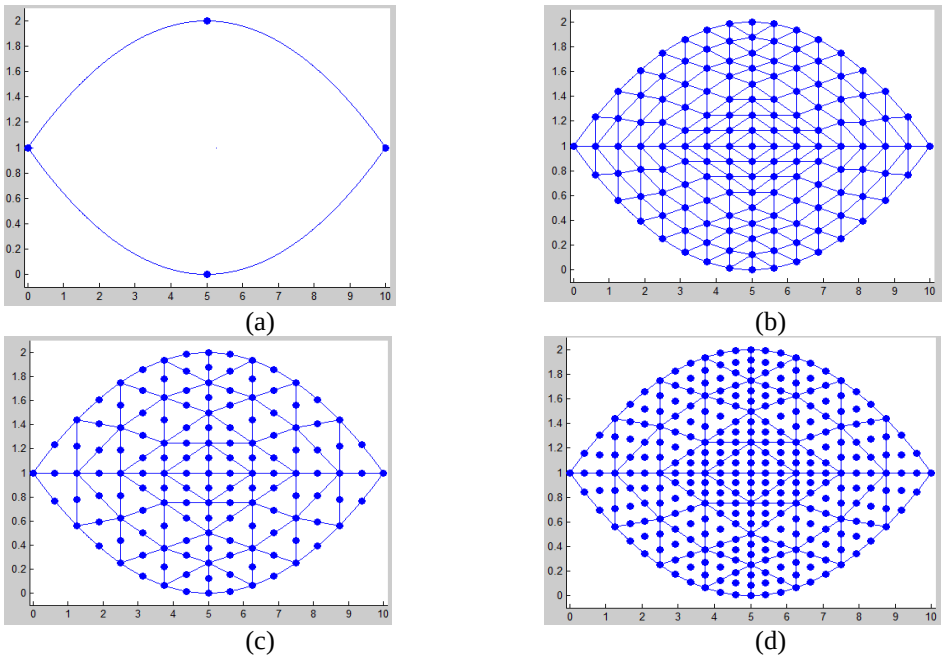


Figure 2.15: Mesh of a domain bounded by two curved(parabolic) sides; (a) domain to be meshed, (b) Mesh using 256 linear elements, (c) Mesh using 64 quadratic elements, and (d) Mesh using 64 cubic elements.

The domain (in Figure 2.15(a)) is created by including the mathematical expressions of two curves into the code. Then, after the input of the coordinates of four vertices; the meshes (in Figures 2.15(b)–2.15(d)) are created by the code according to the given choice of order (2 for quadratic and 3 for cubic) of the elements.

3. Conclusions

In this study, we developed a general algorithm to discretize the two-dimensional convex domain of the BVP using a finite number of linear, quadratic, and cubic triangular elements. This mesh generation algorithm is developed based on the n vertices (actual or more in number) that describe the boundary of the domain and the computed (n+1)-th point (like the centroid). The algorithm generates the mesh using exactly n triangular elements in the first step. Then in every subsequent step, it uses four times as many triangles as the previous step to generate the new mesh. Further, the algorithms employ (a) the straight-sided triangular elements (linear, quadratic, and cubic) to generate the mesh for the domain with a polygonal boundary and (b) the straight-sided linear and curved (quadratic and cubic) triangular elements (with two straight sides and one curved side) to generate the mesh for the domain containing a curved boundary.
Thus, if vertices are specified once at the initial step, the algorithm generates the desired meshes. Further, the inclusion of the mathematical expression of the curved boundary enables the algorithm to generate the fine mesh for the curved domain utilizing the higher-order curved triangular elements. Finally, we developed the computer code in MATLAB to create the mesh, prepare the elements' data, and determine the elements' connectivity.
We tested the program for the mesh generation of different types of convex domains with polygonal and non-polygonal boundaries. We found correct element data and connectivity between local and global nodes. We expect that the code can also be used to create the mesh of a concave domain by adding extra nodes on the boundary or decomposing the domain into a few convex subdomains.
Therefore, we assert that the developed algorithm and code will mitigate the severe difficulties pertinent to automatic mesh generation and provide an easy way for generating the p-version fine meshes with triangular (linear, quadratic, and cubic) elements.
We also believe that the present technique will be useful in developing algorithms for constructing meshes of two-dimensional concave as well as three-dimensional convex domains. We are intending to undertake the development of algorithms for the two-dimensional general (convex, concave, or combination of both) domains.

Acknowledgements
We would like to express our heartfelt gratitude to all those who have contributed to this research in one way or another. Your support and encouragement have been invaluable, and we are grateful for your contributions. We are also thankful to authorities of SUST research center and Mathematics Department, Shahjalal University of Science and Technology (SUST) for providing us with access to their resources and facilities.

Funding

We appreciate the financial support provided by BANBEIS, Education Ministry of Bangladesh, which enabled us to conduct this research. We are also thankful to authority of central library, SUST for providing us with access to their resources and facilities.

References

[1] Hazrat Ali, Md. Kamrujjaman, Numerical solutions of nonlinear parabolic equations with robin condition: Galerkin approach, TWMS Journal of Applied and Engineering Mathematics, Volume 12 (3), 2022, 851- 863.
[2] Hazrat Ali, Md. Kamrujjaman, and Md. Shafiqul Islam, Numerical computation of Fitzhugh-Nagumo equation: A novel Galerkin finite element method approach, International Journal of Mathematical Research, Volume 1, 2020, 20-27.
[3] H.T. Rathod, Bharat Rathod, A new approach to Automesh Generation of all β graded triangular and quadrilateral Finite Elements over Analytical Surfaces by using the parabolic Arcs passing through four points on the boundary curve, International Journal of Engineering and Computer Science, Volume 7, Issue 9, 2018, 24214-24310
[4] Rina Paul, M.S. Karim, Md. Sadekur Rahman, Side based Automatic Mesh generation scheme for a general convex domain with quadrilaterals, IOSR Journal of Mathematics (IOSR-JM), Volume 15, Issue 6, 2019, 65-75
[5] Szabo. B.A. and Metha. A.K., p-Convergent Finite Element Approximations in Fructure Mechanics, International Journal for Numerical Methods in Engineering, 12, 551-560, 1978
[6] H.T. Rathod, K.Sugantha Devi, A New Approach To Automatic Generation Of All Quadrilateral Meshes Over A Linear Convex Polygon With H-Refinements For Finite Element Analysis, Volume 5, Issue 7, 2016, 17172-17238

[7] H.T. Rathod and M.S. Karim, Synthetic division based integration of rational function of bivariate polynomial numerators with linear denominators over a unit triangle. Computers Methods in Applied Mechanics and Engineering, 181:191-235, 2000.10, 28

[8] M.S. Karim, Integration of some Bivariate Polynomial with Rational Denominators- An Application to finite element method. A Ph.D. Thesis, Department of Mathematics, Bangalore University, Bangalore, India. 2001.

[9] H.T. Rathod and M.S. Karim, An explicit integration scheme based on recursion for the curved triangular finite elements. Computer and Structures, 80(1):73-93, 2001, 243,244,245

[10] O.C. Zienkiewicz and Y.K. Cheung, Finite elements in the solution of field problems. The Engineers, 220:507-510, 1965, 19

[11] Farzana Hussain, Efficient Integration Schemes and Algorithms for Finite Element Domain Integrals- An Application to Computational Engineering Science, A Ph.D. Thesis, Department of Mathematics, Shahjalal University of Science and Technology, Sylhet, Bangladesh, January 2014.

[12] O.C. Zienkiewicz and R.L Taylor, The Finite element method. Maidenhead, Mc Graw Hill, U.K. 1989. 20,42, 55, 67, 101

[13] G.R Cowper, Gaussian quadrature formulas for triangles. International journal of numerical methods and engineering, 7: 405-408, 1973. 6,9, 42, 56, 67, 100, 129

APPENDIX A: Computer code in MATLAB

```
function twoDimMesh
format short
n=input('Enter no of Vertices of the Domain : ');
x=[0 0];
disp('Enter coordinates of the vertices in format [a b]');
for i=1:n
   fprintf('Node %i: ', i);
   node(i, :) = input('');
   x=x+node(i,:);
end
%1st triangulation by using centroid
x=x./n;
node(i+1,:)=x;
lastNode=i+1;
 for j=1:n
   m=j+1;
   if j==n
      m=1;
   end
   tri(j,:)=[j m lastNode];
 end

%curve domain information
cs=input('Is there any curve side for yes enter 1 or 0 for no:')
if cs==1
   %func of curve side (here,1st quater of unit circle)
   curve=@(x) sqrt(1-x.^2)
else
   curve=@(x)999;
end
% info for elements
```

```
fprintf('Enter Number of Elements: 1 0r %i or %i or %i or %i  and so',n,4*n,4*4*n,4*4*4*n);
ne=input('elements?:');
oe=input('Enter order of element(1 or 2 or 3):');

if oe==1
      [nodeCord triNode]=mesh1(tri,node,ne,n,curve,cs);
      tri=triNode
      node=nodeCord
      [nodeCord triNode]=mesh1(tri,node,ne,n,curve,cs);
      print(triNode,nodeCord);
      drawDomain(tri,triNode,nodeCord,cs,curve);
elseif oe==2
      [nodeCord triNode]=mesh1(tri,node,ne,n,curve,cs);
      tri=triNode;
      node=nodeCord;
      [nodecord triNode]=mesh2(tri,node,curve,cs);
      print(triNode,nodecord);
      drawDomain(tri,triNode,nodecord,cs,curve);
else if oe==3
      [nodeCord triNode]=mesh1(tri,node,ne,n,curve,cs);
      tri=triNode;
      node=nodeCord;
      [nodecord triNode]=mesh3(tri,node,curve,cs);
      print(triNode,nodecord);
      drawDomain(tri,triNode,nodecord,cs,curve);

   end
end

end


function [node tri]= mesh1(tri,node,ne,n,gg,cs)
if ne==1
   return;
else
   x=[0 0];
   for i=1:n
      x=x+node(i,:);
   end

   x=x./n;
   node(i+1,:)=x;
   lastNode=i+1;
   for j=1:n
       m=j+1;
       if j==n
          m=1;
       end
       tri(j,:)=[j m lastNode];
   end
end


if ne==n
   return;
else
```

```
    for l=1:10

      [noTri g ]=size(tri);
      [lastNode g]=size(node);

       if noTri==ne
          break;
       end

       elementsFlag=1;
       tempNode=zeros(1,3);

       for i=1:noTri
          for j=1:3
             m=j+1;
             if j==3
                m=1;
             end
       if cs==1 && (abs(gg(node(tri(i,j),1))-node(tri(i,j),2))<=0.01 && abs(gg(node(tri(i,m),1))-
       node(tri(i,m),2))<=0.01)
       x(1)=(node(tri(i,j),1)+node(tri(i,m),1))/2;
       x(2)=gg(x(1));
       else
       x(1)=(node(tri(i,j),1)+node(tri(i,m),1))/2;
       x(2)=(node(tri(i,j),2)+node(tri(i,m),2))/2;
       end

       temp=find(ismember(node,x,'rows'));

       if temp<=lastNode
          tempNode(j)=temp;
       else
          node(lastNode+1,:)=x;
          lastNode=lastNode+1;
          tempNode(j)=lastNode;
       end
          end

           elements(elementsFlag,:)=[tri(i,1) tempNode(1) tempNode(3)];
           elements(elementsFlag+1,:)=[tempNode(1) tri(i,2) tempNode(2)];
           elements(elementsFlag+2,:)=[tempNode(2) tri(i,3) tempNode(3)];
           elements(elementsFlag+3,:)=[tempNode(1) tempNode(2)  tempNode(3)];
           elementsFlag=elementsFlag+4;
       end
          tri=elements;
    end
end
end

function [node tri]= mesh2(tri,node,gg,cs)

[ne tem]=size(tri);
[l temp2]=size(node);

for e=1:ne
   if cs==1 && (abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.01 && abs(gg(node(tri(e,2),1))-node(tri(e,2),2))<=0.01)
       x(1)=(node(tri(e,1),1)+node(tri(e,2),1))/2;
       x(2)=gg(x(1));
```

```
        temp=x;
     else
     temp=(node(tri(e,1),:)+node(tri(e,2),:))/2;
     end
   t=find(ismember(node,temp,'rows'));
    if t<=l
      t1=t;
    else
       t1=l+1;
       l=l+1;
       node(l,:)=temp;
    end

    if cs==1 && (abs(gg(node(tri(e,2),1))-node(tri(e,2),2))<=0.01 && abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01)
        x(1)=(node(tri(e,2),1)+node(tri(e,3),1))/2;
        x(2)=gg(x(1));
        temp=x;
     else
     temp=(node(tri(e,2),:)+node(tri(e,3),:))/2;
     end
     t=find(ismember(node,temp,'rows'));
    if t<=l
      t2=t;
    else
       t2=l+1;
       l=l+1;
       node(l,:)=temp;
    end
     if cs==1 && (abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.01 && abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01)
        x(1)=(node(tri(e,3),1)+node(tri(e,3),1))/2;
        x(2)=gg(x(1));
        temp=x;
     else
     temp=(node(tri(e,1),:)+node(tri(e,3),:))/2;
     end
     t=find(ismember(node,temp,'rows'));
    if t<=l
      t3=t;
    else
       t3=l+1;
       l=l+1;
       node(l,:)=temp;
    end

   newtri(e,:)=[tri(e,1) t1 tri(e,2) t2 tri(e,3) t3];

end
tri=newtri;
return;
end

function [node tri]= mesh3(tri,node,gg,cs)
[ne tem]=size(tri);
[l temp2]=size(node);

for e=1:ne
   if cs==1 && (abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.1 && abs(gg(node(tri(e,2),1))-node(tri(e,2),2))<=0.1)
        x(1)=(2*node(tri(e,1),1)+node(tri(e,2),1))/3;
```

```
          x(2)=gg(x(1));
          temp=x;
       else
       temp=(2*node(tri(e,1),:)+node(tri(e,2),:))/3;
      end

    t=find(ismember(node,temp,'rows'));
      if t<=l
         t1=t;
      else
          t1=l+1;
          l=l+1;
          node(l,:)=temp;
      end
        if cs==1 && (abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.01 && abs(gg(node(tri(e,2),1))-
node(tri(e,2),2))<=0.01)
          x(1)=(node(tri(e,1),1)+2*node(tri(e,2),1))/3;
          x(2)=gg(x(1));
          temp=x;

       else
       temp=(node(tri(e,1),:)+2*node(tri(e,2),:))/3;
      end
        t=find(ismember(node,temp,'rows'));
      if t<=l
         t2=t;
      else
          t2=l+1;
          l=l+1;
          node(l,:)=temp;
      end
      if cs==1 && (abs(gg(node(tri(e,2),1))-node(tri(e,2),2))<=0.01 && abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01)
          x(1)=(2*node(tri(e,2),1)+node(tri(e,3),1))/3;
          x(2)=gg(x(1));
          temp=x;

       else
       temp=(2*node(tri(e,2),:)+node(tri(e,3),:))/3;
     end
      t=find(ismember(node,temp,'rows'));
      if t<=l
         t3=t;
      else
          t3=l+1;
          l=l+1;
          node(l,:)=temp;
      end
      if cs==1 && (abs(gg(node(tri(e,2),1))-node(tri(e,2),2))<=0.01 && abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01)
          x(1)=(node(tri(e,2),1)+2*node(tri(e,3),1))/3;
          x(2)=gg(x(1));
          temp=x;

       else
       temp=(node(tri(e,2),:)+2*node(tri(e,3),:))/3;
     end
      t=find(ismember(node,temp,'rows'));
      if t<=l
         t4=t;
```

```
    else
       t4=l+1;
       l=l+1;
       node(l,:)=temp;
    end
    if cs==1 && (abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01 && abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.01)
        x(1)=(node(tri(e,1),1)+2*node(tri(e,3),1))/3;
        x(2)=gg(x(1));
        temp=x;

     else
     temp=(2*node(tri(e,3),:)+node(tri(e,1),:))/3;
   end
    t=find(ismember(node,temp,'rows'));
    if t<=l
      t5=t;
    else
       t5=l+1;
       l=l+1;
       node(l,:)=temp;
    end
    if cs==1 && (abs(gg(node(tri(e,3),1))-node(tri(e,3),2))<=0.01 && abs(gg(node(tri(e,1),1))-node(tri(e,1),2))<=0.01)
        x(1)=(2*node(tri(e,1),1)+node(tri(e,3),1))/3;
        x(2)=gg(x(1));
        temp=x;

     else
     temp=(2*node(tri(e,1),:)+node(tri(e,3),:))/3;
   end
     t=find(ismember(node,temp,'rows'));
    if t<=l
      t6=t;
    else
       t6=l+1;
       l=l+1;
       node(l,:)=temp;
    end
     node(l+1,:)=(node(tri(e,1),:)+node(tri(e,2),:)+node(tri(e,3),:))/3;
     l=l+1;
     t7=l;
     newtri(e,:)=[tri(e,1) t1  t2 tri(e,2) t3 t4  tri(e,3) t5  t6  t7];

end
tri=newtri;
return;
end


function print(tri,node)

[m n]=size(tri);
[k l]=size(node);
disp('Global and local Nodes relationship matrix:');
for i=1:m
   fprintf('\n[%3i] :',i);
   for j=1:n
      fprintf('%3i  ',tri(i,j));
   end
```

```
end
fprintf('\n');
disp('Cordinates:');
for i=1:k
 fprintf('%3i : %0.3f  %0.3f\n',i,node(i,1),node(i,2));
end
end

 function drawDomain(t,ht,c,cs,gg)
[k l]=size(t);
       xmin = min(c(:,1)) ;
       ymin = min(c(:,2));
       xmax = max(c(:,1)) ;
       ymax = max(c(:,2));
       figure
       xlim([xmin-0.1, xmax+0.1]); ylim([ymin-0.1, ymax+0.1]);
       hold on ;
  if cs==1
   p=[0:0.01:1]
   y=gg(p)
   plot(p,y)
   hold on
end
for i=1:k
          cx = sum(c(t(i,:), 1))/l;
          cy = sum(c(t(i,:), 2))/l;
          cx=(c(t(i,1), 1)+3*cx)/4;
          cy=(c(t(i,1), 2)+3*cy)/4;
          text(cx,cy,['[',int2str(i),']']);
          hold on
       for j=1:3
       m=j+1;
       if j==3
          m=1;
       end
       if abs(gg(c(t(i,j),1))-c(t(i,j),2))<=0.01 && abs(gg(c(t(i,m),1))-c(t(i,m),2))<=0.01
          continue
       end
       x=[c(t(i,j),1) c(t(i,m),1)];
       y=[c(t(i,j),2) c(t(i,m),2)];
       plot(x,y);
       hold on;
   end
end
[k l]=size(ht);
for i=1:k
  for j=1:l
          plot(c(ht(i,j),1),c(ht(i,j),2), 'Marker', 'O', 'MarkerFaceColor', 'blue');
          hold on
          text(c(ht(i,j),1)-0.02, c(ht(i,j),2)-0.02, int2str(ht(i,j)));
          hold on
   end
end
end
```