www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 5 may, 2014 Page No. 6062-6098

A New Automated Scheme of Quadrilateral Mesh Generation for Finite Element Analysis

H.T. Rathod[a*] , Bharath Rathod[b] , K. T. Shivaram[c] , K. Sugantha Devi[d]

[a] Department of Mathematics, Central College Campus, Bangalore University,
Bangalore -560001, Karnataka state, India.
Email: htrathod2010@gmail.com

[b] Xavier Institute of Management and Entrepreneurship, Hosur Road,
Electronic City Phase II, Bangalore, Karnataka 560034.
Email: rathodbharath@gmail.com

[c] Department of Mathematics, Dayananda Sagar College of Engineering,
Bangalore- 560078, Karnataka state, India.
Email: shivaramktshiv@gmail.com

[d] Department of Mathematics, Dr. T. Thimmaiah Institute of Technology, Oorgam Post,
Kolar Gold Field, Kolar District, Karnataka state, Pin- 563120, India.
Email: suganthadevik@yahoo.co.in

Abstract

This paper presents a novel mesh generation scheme of all quadrilateral elements for a convex polygonal domain. This scheme converts the elements in background quadrilateral mesh into quadrilaterals through the operation of splitting. We first decompose the convex polygon into simple sub regions in the shape of quadrilaterals. These simple regions are then triangulated to generate a fine mesh of six node triangular elements. We propose then an automatic triangular to quadrilateral conversion scheme. Each isolated triangle is split into three quadrilaterals according to the usual scheme, adding three vertices in the middle of the edges and a vertex at the barrycentre of the element. To preserve the mesh conformity a similar procedure is also applied to every triangle of the domain to fully discretize the given convex polygonal domain into all quadrilaterals, thus propagating uniform refinement and quadrangulation. This simple method generates a high quality mesh whose elements confirm well to the requested shape by refining the problem domain. Examples are presented to illustrate the simplicity and efficiency of the new mesh generation method for standard and arbitrary shaped domains. We have appended MATLAB programs which incorporate the mesh generation scheme developed in this paper. These programs provide valuable output on the nodal coordinates ,element connectivity and graphic display of the all quadrilateral mesh for application to finite element analysis.

Keywords: finite elements, triangulation ,quadrilateral mesh generation, convex polygonal domain, uniform refinement, quadrangulation

1. Introduction

The finite element method (FEM) developed in the 1950's as a method to calculate the elastic deformations in solids. Sixty years later, the point of view is more abstract which allows FEM to be used as a general purpose method applicable to all kinds of partial differential equations. The advent of modern computer technologies provided a powerful tool in numerical simulations for a range of problems in partial differential equations over arbitrary complex domains. A mesh is required for finite element method as it uses finite elements of a domain for analysis. Finite Element Analysis (FEA) is widely used for many fields including structures and optimization. The FEA in engineering applications comprises three phases: domain discretization, equation solving and error analysis. The domain discretization or mesh generation is the preprocessing phase which plays an important role in the achievement of accurate solutions.

FEM requires dividing the analysis region into many sub regions. These small regions are the elements which are connected with adjacent elements at their nodes. Mesh generation is a procedure of generating the geometric data of the elements and their nodes, and involves computing the coordinates of nodes, defining their connectivity and thus constructing the elements. Hence mesh designates aggregates of elements nodes and lines representing their connectivity. Though the FEM is a powerful and versatile tool, its usefulness is often hampered by the need to generate a mesh. Creating a mesh is the first step in a wide range of applications, including scientific and engineering computing and computer graphics. But generating a mesh can be very time consuming and prone to error if done manually. In recognition of this problem a large number of methods have been devised to automate the mesh generation task. An attempt to create a fully automatic mesh generator that is capable of generating a valid finite element meshes over arbitrary complex domains and needs only the information of the specified geometric boundary of the domain and the element size, started from the pioneering work [1] in the early 1970's. Since then many methodologies have been proposed and different algorithms have been devised in the development of automatic mesh generators [2-4]. In order to perform a reliable finite element simulation a number of researchers [5-7] have made efforts to develop adaptive FEA method which integrates with error estimation and automatic mesh modification. Traditionally adaptive mesh generation process is started from coarse mesh which gives large discretization error levels and takes a lot of iterations to get a desired final mesh. The research literature on the subject is vast and different techniques have been proposed [8]. As several engineering applications to real world problems cannot be defined on a rectangular domain or solved on a structured square mesh. The description and discretization of the design domain geometry, specification of the boundary conditions for the governing state equation, and accurate computation of the design response may require the use of unstructured meshes.

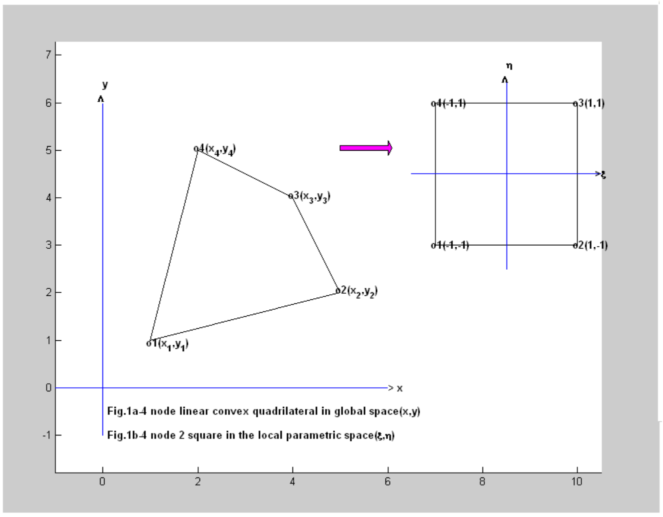
An unstructured simplex mesh requires a choice of mesh points (vertex nodes) and triangulation. Many mesh generators produce a mesh of triangles by first creating all the nodes and then connecting nodes to form of triangles. The question arises as to what is the 'best' triangulation on a given set of points. One particular scheme, namely Delaunay triangulation [8], is considered by many researchers to be most suitable for finite element analysis. If the problem domain is a subset of the Cartesian plane, triangular or quadrilateral meshes are typically employed.

The method used for mesh generation can greatly affect the quality of the resulting mesh. Usually the geometry and physical problem of the domain direct the user which method to apply. The real problems in 2D and 3D involve the complex topology, and distribution of the boundary conditions. Such situation requires automatic mesh generator to reduce the user influence to this process as much as possible. The advancing front is another popular mesh generation method that can be used for adapting FE mesh strategies. Conceptually , the advancing front method is one of the simplest mesh generation processes. This element generating algorithm starts from an initial front formed from the specified boundary of the domain and then generates elements, one by one, as the front advances into the region to be discretized until the whole domain is completely covered by elements [9-10]. In general, good quality meshes of quadrilateral elements cannot be directly obtained from these meshing techniques. An additional step is therefore required to obtain quadrilateral meshes from the triangular meshes. It is generally known that FEA using quadrilateral mesh is more accurate than that of a triangular one [11-20].

In this paper, we present a novel mesh generation scheme of all quadrilateral elements for convex polygonal domains. This scheme converts the elements in background quadrilateral mesh into quadrilaterals through the operation of splitting. We first decompose the convex polygon into simple subregions in the shape of quadrilaterals. These simple subregions are then triangulated to generate a fine mesh of six node triangles. We propose then an automatic triangular to quadrilateral conversion scheme in which each isolated triangle is split into three quadrilaterals according to the usual scheme, adding three vertices in the middle of edges and a vertex at the barrycentre of the triangular element. Further, to preserve the mesh conformity a similar procedure is also applied to every triangle of the domain and this fully discretizes the given convex polygonal domain into all quadrilaterals, thus propogating uniform refinement and quadrangulation. In section-2 of this paper,we present a scheme to discretize the arbitrary quadrilaterals and standard squares into a fine mesh of six node triangular elements.In section- 3,we explain the procedure to split these triangles into quadrilaterals. In section-4,we have presented a method of piecing together of all quadrilateral subregions and eventually creating a all quadrilateral mesh for the given convex polygonal domain. In section-5,we present several examples to illustrate the simplicity and efficiency of the proposed mesh generation method for standard squares and arbitrary quadrilaterals,rectangles and convex polygonal domains.

2. Division of an Arbitrary Quadrilateral

We can map an arbitrary four noded linear convex quadrilateral in the global (Cartesian) coordinate system (x, y) as in Fig 1a, into a 2-square in the local(natural) parametric coordinate system (ξ , η) as in Fig 1b.



y
x
η
ξ
1(x1,y1)
2(x2,y2)
3(x3,y3)
4(x4,y4)
4(-1,1)
3(1,1)
1(-1,-1)
2(1,-1)
Fig.1a-4 node linear convex quadrilateral in global space(x,y)
Fig.1b-4 node 2 square in the local parametric space(ξ,η)

The necessary transformation is given by the equations

$$\begin{pmatrix}x\\y\end{pmatrix} = \sum_{k=1}^{4}\begin{pmatrix}x_k\\y_k\end{pmatrix} M_k(\xi,\eta) \qquad \text{-----------------------} \ (1)$$

Where (x_k, y_k) , (k=1,2,3,4) are the vertices of the original arbitrary linear convex quadrilateral in (x, y) plane and $M_k(\xi, \eta)$ denote the well known bilinear basis functions [13,14] in the local parametric space (u, v) and they are given by

$$M_k(\xi, \eta) = \frac{1}{4}(1+\xi\xi_k)(1+\eta\eta_k)\ , \quad k = 1, 2,3,4 \qquad \text{----------------------- (2a)}$$

Where { $(\xi_k, \eta_k), k = 1,2,3,4$} = {(-1,-1),(1,-1),(1,1),(-1,1)} ----------------------- (2b)

describe the transformations over a linear convex quadrilateral element from the original global space(x,y) into the local parametric space (ξ , η).

We can also map an arbitrary four noded linear convex quadrilateral in the global (Cartesian) coordinate system (x, y) as in Fig 1c, into a 1-square in the local(natural) parametric coordinate system (u, v) as in Fig 1d.

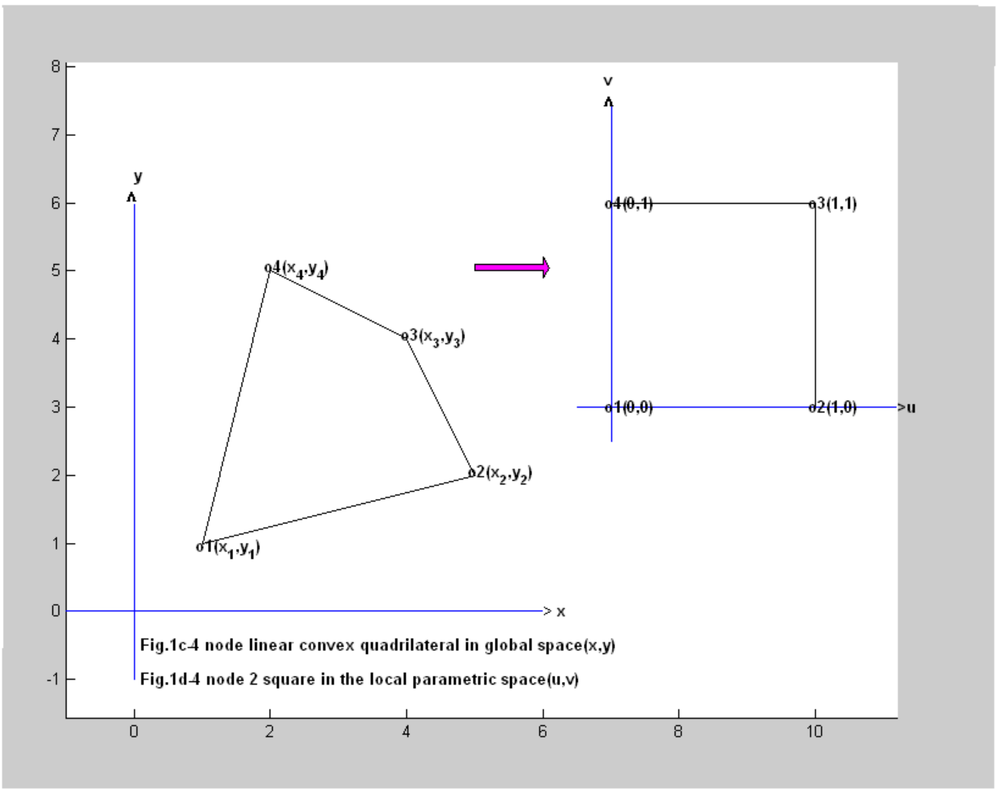


Fig.1c-4 node linear convex quadrilateral in global space(x,y)

Fig.1d-4 node 2 square in the local parametric space(u,v)

The necessary transformation is given by the equations

$$\begin{pmatrix} x \\ y \end{pmatrix} = \sum_{k=1}^{4} \begin{pmatrix} x_k \\ y_k \end{pmatrix} N_k(u, v) \qquad \text{------------------------ (3)}$$

Where (x_k, y_k) , (k=1,2,3,4) are the vertices of the original arbitrary linear convex quadrilateral in (x, y) plane and $N_k(u, v)$ denote the well known bilinear basis functions [13,14] in the local parametric space (u, v) and they are given by

$$N_1(u, v) = (1-u)(1-v),\ \ N_2(u, v) = u(1-v),\ \ N_3(u, v) = uv,\ \ N_4(u, v) = (1-u)v \qquad \text{------------------ (4)}$$

describe the transformations over a linear convex quadrilateral element from the original global space(x,y) into the local parametric space (u, v).

The mappings of eqns.(1-4) describes a unique relation between the coordinate systems.

This is illustrated by using the division of each side into three equal parts in Fig. 2a,b and Fig. 2c,d . It is clear that all the coordinates of this division can be determined by knowing the coordinates $((x_i, y_i),\ i = 1, 2, 3, 4)$ of the vertices for the arbitrary quadrilateral. In general , it is well known that by making 'n' equal divisions on all sides, we can divide an arbitrary quadrilateral into n^2 smaller quadrilaterals.

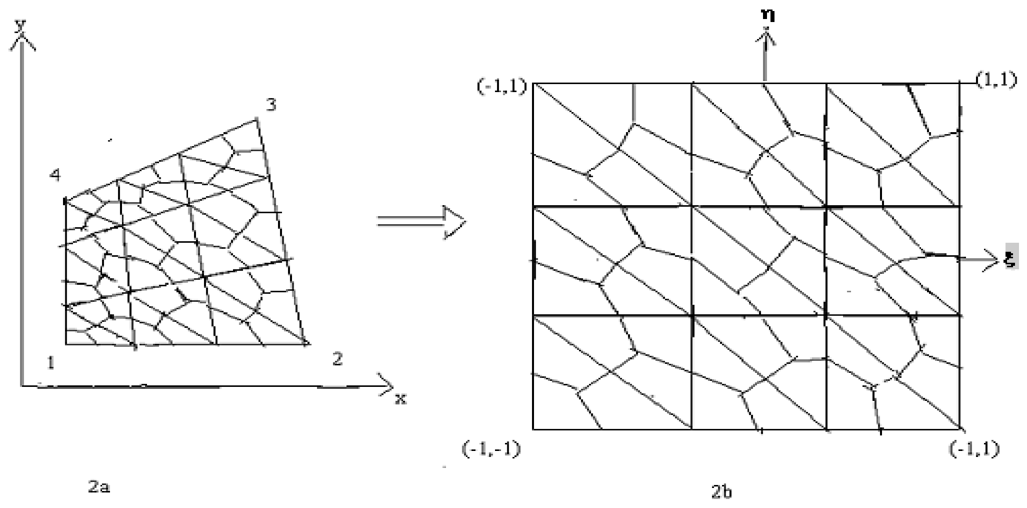


Fig.2a,b: MAPPING OF AN ARBITRARY QUADRILATERAL INTO A 2-SQUARE WITH SPECIAL QUADRILATERALS FROM (x,y) TO (ξ,η) SPACE

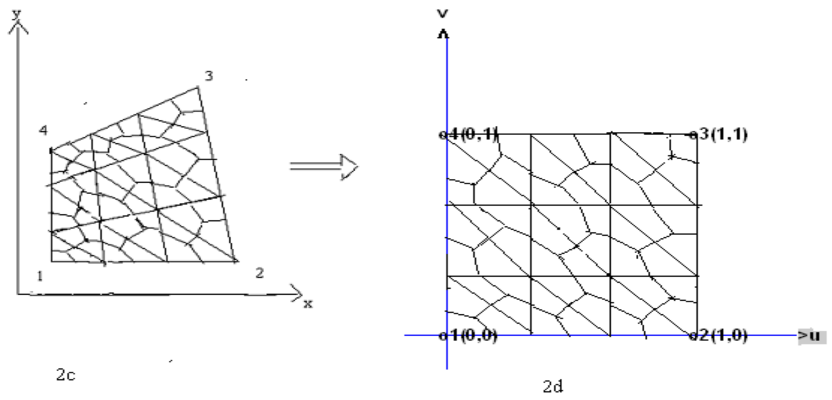


Fig.2c,d: MAPPING OF AN ARBITRARY QUADRILATERAL INTO A 2-SQUARE WITH SPECIAL QUADRILATERALS FROM (x,y) TO (u,v) SPACE

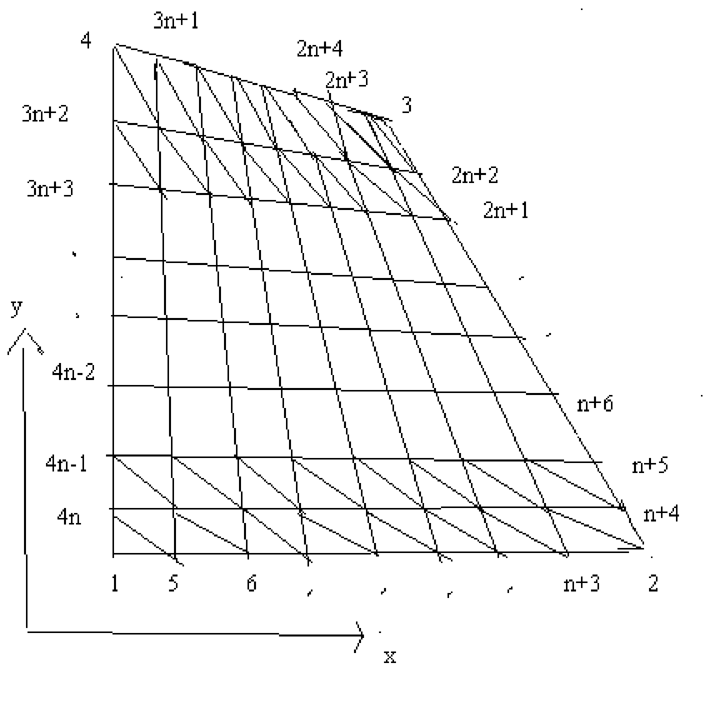


Fig. 3a:Division of an arbitrary quadrilateral into (n*n) small quadrilaterals

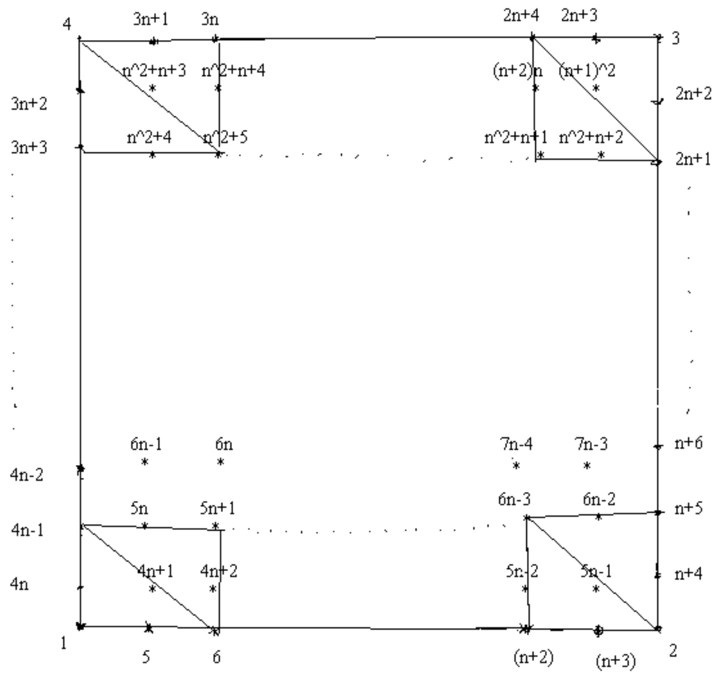


Fig. 3b:Division of a 2-square into six node triangles

We have shown the division of an arbitrary quadrilateral and a 2-square in Fig. 3a , Fig. 3b, We divided each side of the quadrilateral or 2-square (either in Cartesian space(x,y) or natural space(u,v)) into n equal parts and join opposite sides by lines. This creates $(n+1)^2$ nodes. These nodes are numbered from base line l_{12} (letting l_{ij} as the line joining the vertex (x_i, y_i) and $(x_j, y_j)\ or (u_i, v_i) and (u_j, v_j)$),now with respect to the 2-squqre in Fig.3b,we have along the line $v = 0$ and upwards up to the line $v = 1$. The nodes 1, 2, 3,4 are numbered anticlockwise and then nodes 5,6, ------, (n+3) are along line $v = 0$ and the nodes (n+4), (n+5), ------, (2n+1), (2n+2) are numbered along the line l_{23} i.e. $u = 1$ and then the node (2n+3), (2n+4), -------, (3n+1) are numbered along the line $v = 1$, and finally along the nodes (3n+2),(3n+3),….4n are numbered along the line $u = 0$, Then the interior nodes are numbered in increasing order from left to right along the line $v = \frac{i}{n}, i = 0,1,2, \dots\dots, n$ bounded on the right by the line $u = 1$. Thus the entire square is covered by $(n + 1)^2$ nodes. This is shown in the $\underline{rr}$ matrix of size $(n + 1) \times (n + 1)$

$$\underline{rr} = \begin{bmatrix} 1, & 5, & 6, \dots\dots\dots & (n+2), & (n+3), & 2 \\ 4n, & (4n+1), & (4n+2), \dots\dots\dots & (5n-2), & (5n-1), & (n+4) \\ (4n-1), & 5n, & (5n+1), \dots\dots\dots & (6n-3), & (6n-2), & (n+5) \\ (4n-2), & (6n-1), & 6n, \dots\dots\dots & (7n-4), & (7n-3), & (n+6) \\ & & & & & \\ & & & & & \\ (3n+3), & (n^2+4), & (n^2+5), \dots\dots\dots & (n^2+n+1), & (n^2+n+2), & (2n+1) \\ (3n+2), & (n^2+n+3), & (n^2+n+4), \dots\dots\dots & n(n+2), & (n+1)^2, & (2n+2) \\ 4, & (3n+1), & 3n, & (2n+4), & (2n+4), & 3 \end{bmatrix}$$

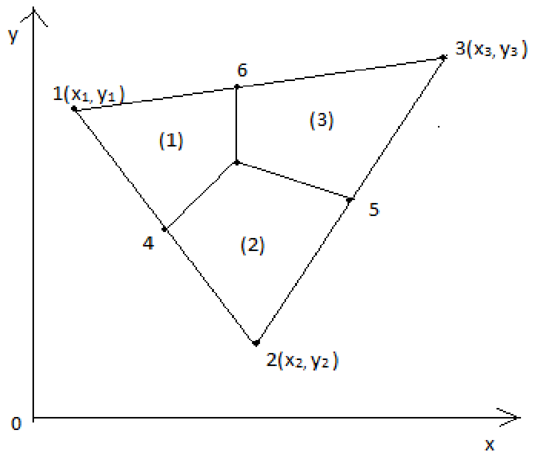
Fig 3c. Matrix rr for the division of a Square

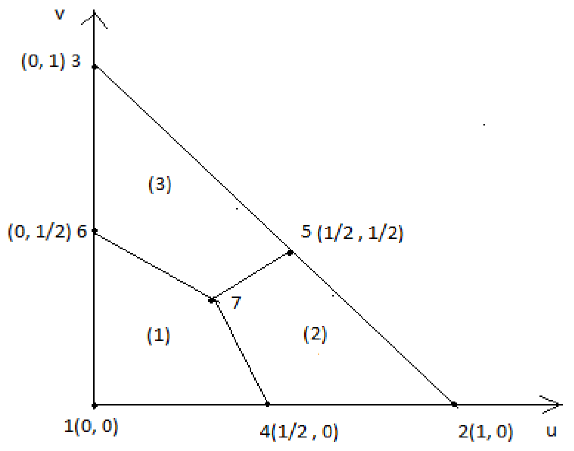
…………………………………………(5)

3. Quadrangulation of an Arbitrary Quadrilateral

We now consider the quadrangulation of an arbitrary quadrilateral. We first divide the arbitrary quadrilateral into two arbitrary triangles. Let us define l_{ij} as the line joining the points (x_i, y_i) and (x_j, y_j) in the Cartesian space (x, y). Then the arbitrary triangle with vertices at $((x_i, y_i), i = 1,2,3)$ is bounded by three lines l_{12}, l_{23}, and l_{31} . By dividing the sides l_{12}, l_{23}, l_{31} into $n = 2m$ divisions (m, an integer) creates m^2 six node triangular divisions. Then by joining the centroid of these six node triangles to the midpoints of their sides, we obtain three quadrilaterals for each of these triangle. We have illustrated this process for the two and four divisions of l_{12}, l_{23}, and l_{31} sides of the arbitrary and standard triangles in Figs. 4 and 5

Two Divisions of Each side of an Arbitrary Triangle



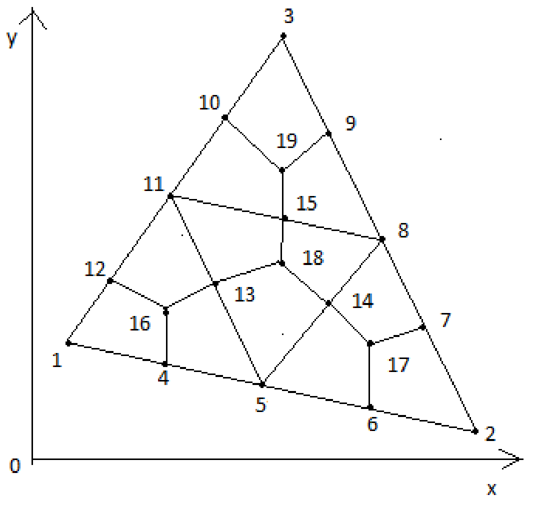


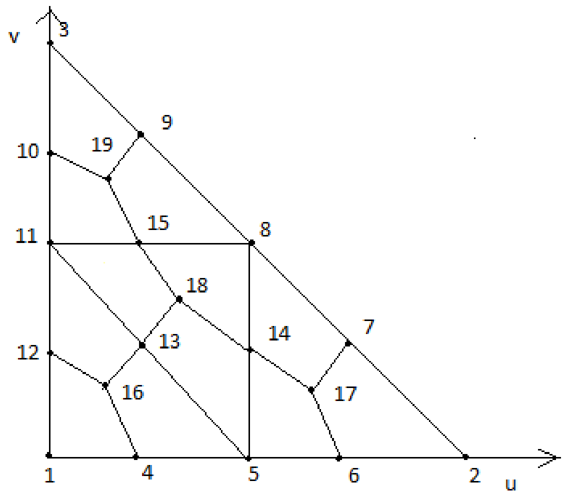
4(a) 4(b)

Fig 4(a). Division of an arbitrary triangle into three quadrilaterals

Fig 4(b). Division of a standard triangle into three quadrilaterals

Four Divisions of Each side of an Arbitrary Triangle





5(a) 5(b)

Fig 5a. Division of an arbitrary triangle into 4 six node triangles

Fig 5b. Division of a standard triangle into 4 right isosceles triangle

In general, we note that to divide an arbitrary quadrilateral into equal size six node triangle, we must divide each side of the quadrilateral into an even number of divisions and locate points in the interior of quadrilateral at equal spacing. We also do similar divisions and locations of interior points for the standard square. Thus n (even) divisions creates n^2 six node triangles in both the spaces. If the entries of the sub matrix $\underline{rr}\,(i\,;\,i+2,\; j\,;\,j+2)$ are nonzero then two six node triangles can be formed. If $\underline{rr}\,(i+1,\; j+2) = \underline{rr}\;(i+2,\; j+1;\, j+2) = 0$ then one six node triangle can be formed. If the sub matrices $\underline{rr}$ $(i\,;\, i+2,\; j\,;\, j+2)$ is a (3×3) zero matrix , we cannot form the six node triangles. We now explain the creation of the six node triangles using the $\underline{rr}$ matrix_of eqn.( ). We can form six node triangles by using node points of three consecutive rows and columns of $\underline{rr}$ matrix. This procedure is depicted in Fig. 6 for three consecutive rows $i\,, i+1,\; i+2$ and three consecutive columns $j\,,\; j+1,\; j+2$ of the $\underline{rr}$ sub matrix

Formation of six node triangle using sub matrix $\underline{rr}$

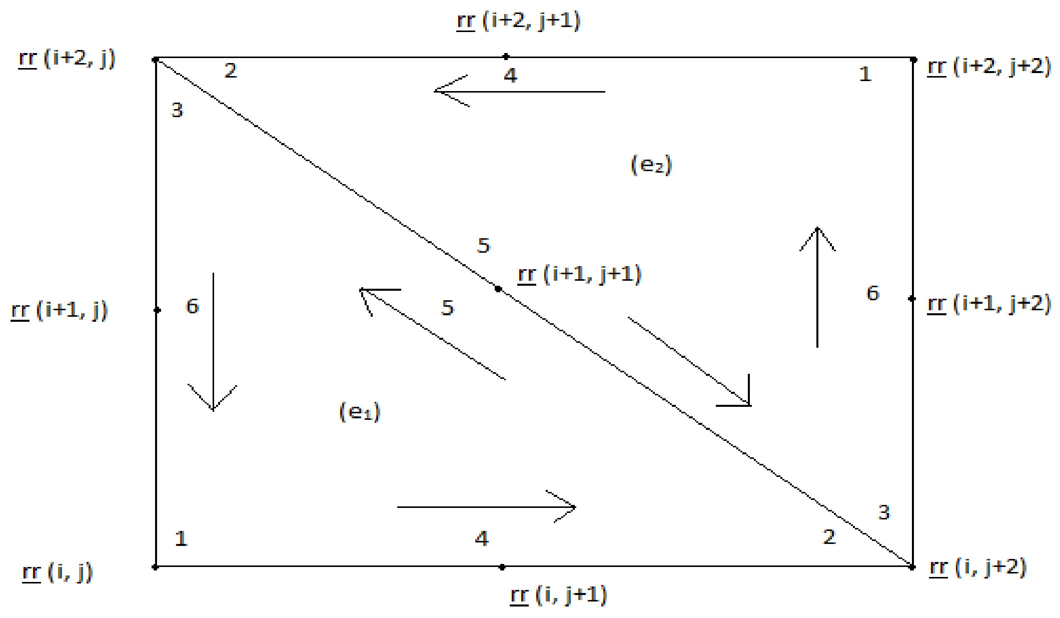


Fig. 6 Six node triangle formation for non zero sub matrix $\underline{rr}$

If the sub matrix ($(\underline{rr}\,(\,k,l), k=i,i+1,i+2), l=j,\; j+1,\; j+2)$ is nonzero, then we can construct two six node triangles. The element nodal connectivity is then given by

(e₁) $<\underline{rr}\;(i,\; j), \underline{rr}\,(i,i+2), \underline{rr}\,(i+2,\; j), \underline{rr}\,(i,j+1),\underline{rr}\,(i+1,j+1), \underline{rr}\,(i+1,j)>$

(e₂) $<\underline{rr}\;(i+2,j+2), \underline{rr}\,(i+2,j), \underline{rr}\,(i,j+2), \underline{rr}\,(i+2,j+1),\underline{rr}\,(i+1,j+1), \underline{rr}$

$(i+1,j+2)>$

………………………………………………………(6)

If the elements of sub matrix ($(\underline{rr}\;(\,k,l), k=i,i+1,i+2),\; l=j,\; j+1,\; j+2)$ are nonzero, then as standard earlier, we can construct two six node triangles. We can create three quadrilaterals in each of these six node triangles. The nodal connectivity for the 3 quadrilaterals created in (e₁) are given as

$Q_{3n_1-2} < c_1\,, \underline{rr}\;(i+1,\; j),\; \underline{rr}\,(i,\; j)\,, \underline{rr}\,(i,j+1)>$

$Q_{3n_1-1} <$ c$_1$, $\underline{rr}\ (i,\ j+1),\ \underline{rr}\,(i,\ j+2)\ ,\underline{rr}\,(i+1, j+1) >$

$Q_{3n_1} <$ c$_1$, $\underline{rr}\ (i+1,\ j+1),\ \underline{rr}\,(i+2,\ j)\ ,\underline{rr}\,(i+1, j) >$ ……………………..(7)

and the nodal connectivity for the 3 quadrilaterals created in (e$_2$) are given as

$Q_{3n_2-2} <$ c$_2$, $\underline{rr}\ (i+1,\ j+2),\ \underline{rr}\,(i+2,\ j+2)\ ,\underline{rr}\,(i+2,\ j+1) >$

$Q_{3n_2-1} <$ c$_2$, $\underline{rr}\ (i+2,\ j+1),\ \underline{rr}\,(i+2,\ j)\ ,\underline{rr}\,(i+1,\ j+1) >$

$Q_{3n_1} <$ c$_2$, $\underline{rr}\ (i+1,\ j+1),\ \underline{rr}\,(i,\ j+2)\ ,\underline{rr}\,(i+1, j+2) >$ **………………………**(8)

3. Quadrangulation of an Arbitrary Polygon

In a recent paper[ ], a new mesh generation method for a convex polygonal domain was presented.This method decomposes the convex polygon into simple subregions in the shape of triangles. These simple regions are then triangulated to generate a fine mesh of triangular elements. We propose then an automatic triangular to quadrilateral conversion scheme. Each isolated triangle is split into three quadrilaterals according to the usual scheme, adding three vertices in the middle of the edges and a vertex at the barrycentre of the element. To preserve the mesh conformity a similar procedure is also applied to every triangle of the domain to fully discretize the given convex polygonal domain into all quadrilaterals, thus propagating uniform refinement. This simple method generates a high quality mesh whose elements confirm well to the requested shape by refining the problem domain.**In this paper we have proposed the decomposition of convex polygonal domain into simple subregions in the shape of arbitrary quadrilaterals**. These simple regions are then quadrangulated as explained in the previous section.This is further explained in the following Figs.7-8. We consider a convex polygon in Fig.7 which is divided into four arbitrary quadrilateral,we generate quadrangular mesh over each of them.The exploded view of the polygonal domain is shown in Fig.8.It is clear that by pieceing together all the four arbitrary quadrilateral which are already quadrangulated by the method of previous section,we can obtain the desired mesh

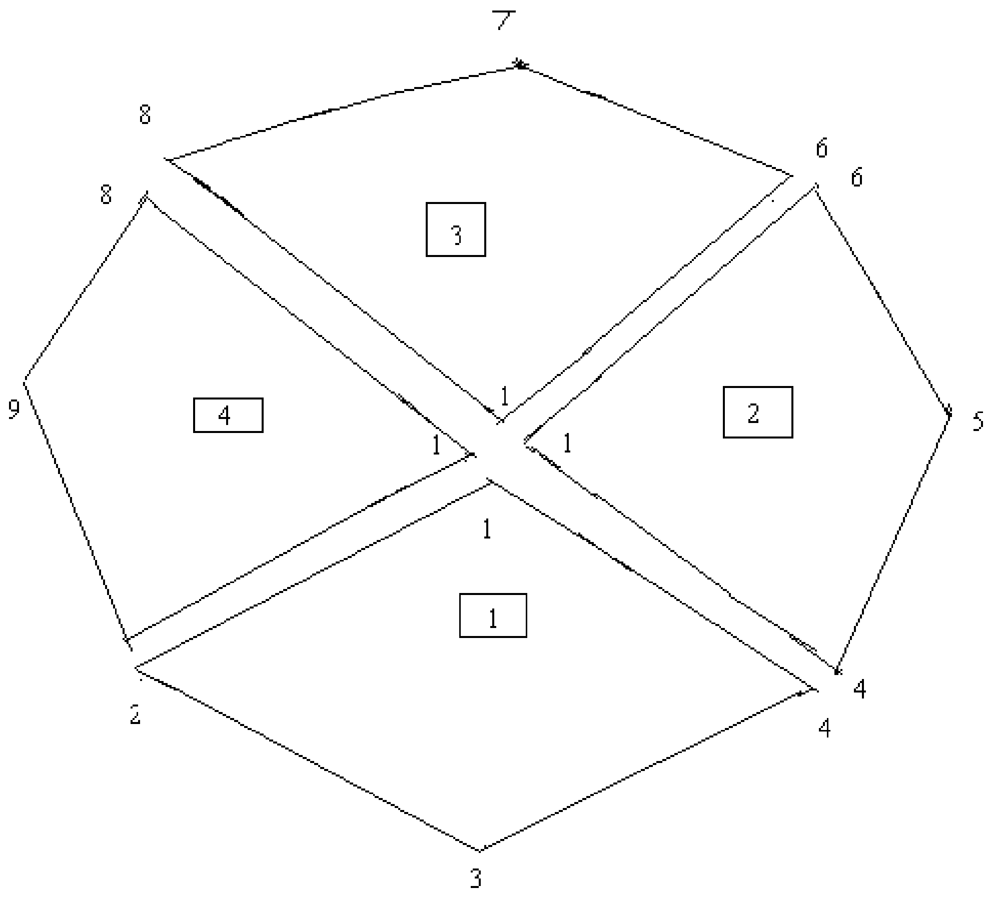


Fig.8 EXPLODED VIEW OF POLYGONAL DOMAIN WITH FOUR QUADRILATERALS

5. Application Examples

Mesh Generation Over an Arbitrary Square Region and a Convex Polygon

In applications to boundary value problems, we may have to discretize an arbitrary square region Our purpose is to have a code which automatically generates convex quadrangulations of the domain by assuming the input as coordinates of the vertices. We use the theory and procedure developed in section 2 and section 3 of this paper for this purpose. The following MATLAB codes are written for this purpose.

(1) quadrilateral_mesh4arbitrarytriangle_q4.m
(2) coordinate_special_quadrilateral_in_unitsquare.m
(3) nodaladdresses_special_convex_quadrilaterals_inQUAD.m
(4) quadrilateral_mesh4convexpolygoneightsidesq4.m
(5) polygonal_domain_QUADcoordinates. m

We have included some meshes generated by using the above codes written in MATLAB. We illustrate the mesh generation for a unit square and an arbitrary quadrilateral.Mesh generation for convex polygon with four arbitrary quadrilaterals(eight sides) is implemented using the codes developed in our earlier paper[26] which uses an assembly of arbitrary triangles.Similar MATLAB codes can be developed from the present theory for a convex polygon which will generate quadrangular mesh by an assembly of arbitrary quadrilaterals.This is an open problem.Some sample commands to generate these quadrilaterals are included in comment lines. In all the codes sample input data is included for easy access.

6 Conclusions

An automatic indirect quadrilateral mesh generator which uses the splitting technique is presented for the two dimensional convex polygonal domains. This mesh generation is made fully automatic and allows the user to define the problem domain with minimum amount of input such as coordinates of boundary. Once this input is created, by selecting an appropriate interior point of the convex polygonal domain, we form the quadrilateral subdomains. These subdomains are then triangulated to generate a fine mesh of six node triangular elements. We have then proposed an automatic triangular to quadrilateral conversion scheme in which each isolated triangle is split into three quadrilaterals according to the usual scheme, adding three vertices in the middle of the edges and a vertex at the barrycentre of the triangular element. This task is made a bit simple since a fine mesh of six node triangles is first generated. Further, to preserve the mesh conformity a similar procedure is also applied to every triangle of the domain and this discretizes the given convex polygonal domain into all quadrilaterals, thus propogating a uniform refinement. This simple method generates high quality mesh whose elements confirm well to the requested shape by refining the problem domain. We have also appended MATLAB programs which provide the nodal coordinates, element nodal connectivity and graphic display of the generated all quadrilateral mesh for the standard square, an arbitrary quadrilateral, and an arbitrary convex polygonal domain. However, similar MATLAB codes can be developed from the present theory for a convex polygon which will generate quadrangular mesh by an assembly of arbitrary quadrilaterals. This is an open problem which requires careful coding .But the mesh generation will be similar to the ones presented in this paper where an arbitrary quadrilateral is made up of two arbitrary triangles. The only difference is the nodal connectivity which has to be done completely for each arbitrary quadrilateral in the sequence of assembly process.This certainly requires less input of the geometry which has a definite advantage over earlier research work[26]. We believe that this work will be useful for various applications in science and engineering.

References

[1] Zienkiewicz. O. C, Philips. D. V, An automatic mesh generation scheme for plane and curved surface by isoparametric coordinates, Int. J.Numer. Meth.Eng, 3, 519-528 (1971)

[2] Gardan. W. J and Hall. C. A, Construction of curvilinear coordinates systems and application to mesh generation, Int. J.Numer. Meth. Eng 3, 461-477 (1973)

[3] Cavendish. J. C, Automatic triangulation of arbitrary planar domains for the finite element method, Int. J. Numer. Meth. Eng 8, 679-696 (1974)

[4] Moscardini. A. O , Lewis. B. A and Gross. M A G T H M – automatic generation of triangular and higher order meshes, Int. J. Numer. Meth. Eng, Vol 19, 1331-1353(1983)

[5] Lewis. R. W, Zheng. Y, and Usmam. A. S, Aspects of adaptive mesh generation based on domain decomposition and Delaunay triangulation, Finite Elements in Analysis and Design 20, 47-70 (1995)

[6] W. R. Buell and B. A. Bush, Mesh generation a survey, J. Eng. Industry. ASME Ser B. 95

332-338(1973)

[7] Rank. E, Schweingruber and Sommer. M, Adaptive mesh generation and transformation of triangular to quadrilateral meshes, Common. Appl. Numer. Methods 9, 11 121-129(1993)

[8] Ho-Le. K, Finite element mesh generation methods, a review and classification, Computer Aided Design Vol.20, 21-38(1988)

[9] Lo. S. H, A new mesh generation scheme for arbitrary planar domains, Int. J. Numer. Meth. Eng. 21, 1403-1426(1985)

[10] Peraire. J. Vahdati. M, Morgan. K and Zienkiewicz. O. C, Adaptive remeshing for compressible flow computations, J. Comp. Phys. 72, 449-466(1987)

[11] George. P.L, Automatic mesh generation, Application to finite elements, New York , Wiley (1991)

[12] George. P. L, Seveno. E, The advancing-front mesh generation method revisited. Int. J. Numer. Meth. Eng 37, 3605-3619(1994)

[13] Pepper. D. W, Heinrich. J. C, The finite element method, Basic concepts and applications, London, Taylor and Francis (1992)

[14] Zienkiewicz. O. C, Taylor. R. L and Zhu. J. Z, The finite element method, its basis and fundamentals, 6th Edn, Elsevier (2007)

[15] Masud. A, Khurram. R. A, A multiscale/stabilized finite element method for the advection-diffusion equation, Comput. Methods. Appl. Mech. Eng 193, 1997-2018(2004)

[16] Johnston. B.P, Sullivan. J. M and Kwasnik. A, Automatic conversion of triangular finite element meshes to quadrilateral elements, Int. J. Numer. Meth. Eng. 31, 67-84(1991)

[17] Lo. S. H, Generating quadrilateral elements on plane and over curved surfaces, Comput. Stuct. 31(3) 421-426(1989)

[18] Zhu. J, Zienkiewicz. O. C, Hinton. E, Wu. J, A new approach to the developed of automatic quadrilateral mesh generation, Int. J. Numer. Meth. Eng 32(4), 849-866(1991)

[19] Lau. T. S, Lo. S. H and Lee. C. S, Generation of quadrilateral mesh over analytical curved surfaces, Finite Elements in Analysis and Design, 27, 251-272(1997)

[20] Park. C, Noh. J. S, Jang. I. S and Kang. J. M, A new automated scheme of quadrilateral mesh generation for randomly distributed line constraints, Computer Aided Design 39, 258-267(2007)

[21]Moin.P, Fundamentals of Engineering Numerical Analysis,second edition,Cambridge University Press(2010)

[22]Fausett.L.V, Applied Numerical Analysis Using MATLAB, second edition, Pearson Education.Inc(2008)

[23]Thompson.E.G, Introduction to the finite element method,John Wiley & Sons Inc.(2005)

[24]ProgramMESHGEN:www.ce.memphis.edu/7111/notes/fem_code/MESHGEN_ tutorial.pdf

[25] Rathod H.T, Venkatesh.B, Shivaram. K.T,Mamatha.T.M, Numerical Integration over polygonal domains using convex quadrangulation and Gauss Legendre Quadrature Rules, International Journal of Engineering and Computer Science, Vol. 2,issue 8,pp2576-2610(2013)

[26]Rathod H.T,Rathod .Bharath,Shivaram.K.T,Sugantha Devi.K, A new approach to automatic generation of all quadrilateral mesh for finite analysis, International Journal of Engineering and Computer Science, Vol. 2,issue 12,pp3488-3530(2013)

PROGRAMS

Program(1) quadrilateralmesh_arbitraryquadrilateral_q4

```
function[]=quadrilateralmesh_arbitraryquadrilateral_q4(nmesh)
%skip=0 or 1
%skip=0,generates  meshes for the nodal and coordinate data given for ten cases
%skip=1,generates  meshes automatically by dividing sides of triangle into equal sizes of 2,4,6,8,etc......
clf
%skip=1%unit square-1/2<=u,v<=1/2
%skip=2;%arbitrary quadrilateral
%skip=3;%arbitrary quadrilateral(different orientation)
skip=input('enter the value:skip=1%unit square-1/2<=u,v<=1/2 ;skip=2 OR 3;%arbitrary quadrilateral')

syms coord ui vi
for mesh=1:nmesh

    figure(mesh)
  ndiv=mesh*2;

[ui,vi,nodes,nodetel,nnode,nel]=coordinate_special_quadrilaterals_in_unitsquare(ndiv);
%coordinates for 1-square
switch skip
case 1 %unit square-1/2<=u,v<=1/2
for i=1:nnode
  gcoord(i,1)=double(2*ui(i,1)-1)/2;
  gcoord(i,2)=double(2*vi(i,1)-1)/2;
end
gcoord

case 2%arbitrary quadrilateral
  xx1=1;xx2=5;xx3=4;xx4=2;
  yy1=1;yy2=2;yy3=4;yy4=5;

   %xx4=1;xx1=5;xx2=4;xx3=2;
   %yy4=1;yy1=2;yy2=4;yy3=5;
for i=1:nnode
  %gcoord(i,1)=double(xx1+ui(i,1)*(-xx1+xx2)+vi(i,1)*(-xx1+xx4)+ui(i,1)*vi(i,1)*(xx1-xx2+xx3-xx4))
  %gcoord(i,2)=double(yy1+ui(i,1)*(-yy1+yy2)+vi(i,1)*(-yy1+yy4)+ui(i,1)*vi(i,1)*(yy1-yy2+yy3-yy4))
  U=double(ui(i,1));V=double(vi(i,1));
  if((U+V)<1)
   gcoord(i,1)=double(xx1+U*(-xx1+xx2)+V*(-xx1+xx4))
   gcoord(i,2)=double(yy1+U*(-yy1+yy2)+V*(-yy1+yy4))
  end
  if((U+V)==1)
   gcoord(i,1)=double(xx2+V*(-xx2+xx4))
   gcoord(i,2)=double(yy2+V*(-yy2+yy4))
  end

   if((U+V)>1)
    gcoord(i,1)=double(xx3+(1-U)*(-xx3+xx4)+(1-V)*(-xx3+xx2))
    gcoord(i,2)=double(yy3+(1-U)*(-yy3+yy4)+(1-V)*(-yy3+yy2))
  end
end
gcoord
case 3%arbitrary quadrilateral
  %xx1=1;xx2=5;xx3=4;xx4=2;
  %yy1=1;yy2=2;yy3=4;yy4=5;

   xx4=1;xx1=5;xx2=4;xx3=2;
   yy4=1;yy1=2;yy2=4;yy3=5;
for i=1:nnode
  %gcoord(i,1)=double(xx1+ui(i,1)*(-xx1+xx2)+vi(i,1)*(-xx1+xx4)+ui(i,1)*vi(i,1)*(xx1-xx2+xx3-xx4))
  %gcoord(i,2)=double(yy1+ui(i,1)*(-yy1+yy2)+vi(i,1)*(-yy1+yy4)+ui(i,1)*vi(i,1)*(yy1-yy2+yy3-yy4))
```

```
   U=double(ui(i,1));V=double(vi(i,1));
   if((U+V)<1)
    gcoord(i,1)=double(xx1+U*(-xx1+xx2)+V*(-xx1+xx4))
    gcoord(i,2)=double(yy1+U*(-yy1+yy2)+V*(-yy1+yy4))
   end
   if((U+V)==1)
    gcoord(i,1)=double(xx2+V*(-xx2+xx4))
    gcoord(i,2)=double(yy2+V*(-yy2+yy4))
   end

    if((U+V)>1)
     gcoord(i,1)=double(xx3+(1-U)*(-xx3+xx4)+(1-V)*(-xx3+xx2))
     gcoord(i,2)=double(yy3+(1-U)*(-yy3+yy4)+(1-V)*(-yy3+yy2))
   end
end
gcoord


end



[nel,nnel]=size(nodes)
[nnode,dimension]=size(gcoord)
%plot the mesh for the generated data
%x and y coordinates
xcoord(1:nnode,1)=gcoord(1:nnode,1);
ycoord(1:nnode,1)=gcoord(1:nnode,2);
%extract coordinates for each element

if skip==1
for i=1:nel
for j=1:nnel
x(1,j)=xcoord(nodes(i,j),1);
y(1,j)=ycoord(nodes(i,j),1);
end;%j loop
xvec(1,1:5)=[x(1,1),x(1,2),x(1,3),x(1,4),x(1,1)];
yvec(1,1:5)=[y(1,1),y(1,2),y(1,3),y(1,4),y(1,1)];
%axis equal
axis tight
xmin=-0.5;xmax=0.5;ymin=-0.5;ymax=0.5;
axis([xmin,xmax,ymin,ymax]);
plot(xvec,yvec);%plot element
hold on;
%place element number
if mesh<6
midx=mean(xvec(1,1:4));
midy=mean(yvec(1,1:4));
text(midx,midy,['(',num2str(i),')']);
end
end;%i loop
xlabel('x axis')
ylabel('y axis')
st1='quadrilateral mesh for unit square ';
st2=' using  ';
st3='bilinear ';
st4='quadriateral';
st5=' elements'
title([st1,st2,st3,st4,st5])
text(-0.6,-0.6,['MESH NO.=',num2str(mesh)])
text(-0.2,-0.6,['number of elements=',num2str(nel)])
text(0.2,-0.6,['number of nodes=',num2str(nnode)])

%put node numbers
if mesh<6
for jj=1:nnode
text(gcoord(jj,1),gcoord(jj,2),['o',num2str(jj)]);
end
```

```
end
hold on
%axis off
else
for i=1:nel
for j=1:nnel
x(1,j)=xcoord(nodes(i,j),1);
y(1,j)=ycoord(nodes(i,j),1);
end;%j loop
xvec(1,1:5)=[x(1,1),x(1,2),x(1,3),x(1,4),x(1,1)];
yvec(1,1:5)=[y(1,1),y(1,2),y(1,3),y(1,4),y(1,1)];
%axis equal
axis tight
xmin=0;xmax=6;ymin=0;ymax=6;
axis([xmin,xmax,ymin,ymax]);
plot(xvec,yvec);%plot element
hold on;
%place element number
if mesh<6
midx=mean(xvec(1,1:4));
midy=mean(yvec(1,1:4));
text(midx,midy,['(',num2str(i),')']);
end
end;%i loop
xlabel('x axis')
ylabel('y axis')
st1='quadrilateral mesh for arbitrary quadrilateral ';
st2=' using  ';
st3='bilinear ';
st4='quadriateral';
st5=' elements'
title([st1,st2,st3,st4,st5])
text(0.1,0.5,['MESH NO.=',num2str(mesh)])
text(2,0.5,['number of elements=',num2str(nel)])
text(4,0.5,['number of nodes=',num2str(nnode)])

%put node numbers
if mesh<6
for jj=1:nnode
text(gcoord(jj,1),gcoord(jj,2),['o',num2str(jj)]);
end
end
hold on
%axis off
end%switch skip

end%for nmesh-the number of meshes
```

Program(2): coordinate_special_quadrilaterals_in_unitsquare

```
function[ui,vi,nodes,nodetel,nnode,nel]=coordinate_special_quadrilaterals_in_unitsquare(n)
%n=number of divisions of the unit square
%n must be even:n=2,4,6,......
syms ui vi wi
ui(1:4,1)=[0;1;1;0];
vi(1:4,1)=[0;0;1;1];
%wi(1:3,1)=[1;0;0];
if (n-1)>0
%BASE OF THE UNIT SQUARE
 k=4;
 for i=1:n-1
    k=k+1;
    ui(k,1)=sym(i/n);
    vi(k,1)=sym(0);
    %wi(kk,1)=sym(1-ui(kk,1)-vi(kk,1));
 end
 %RIGHT EDGE OF THE UNIT SQUARE
 kk=k;
 for ii=1:n-1
```

```
   kk=kk+1;
   ui(kk,1)=sym(1);
   vi(kk,1)=sym(ii/n);
   %wi(kkk,1)=0;
 end;
 %top edge of the unitsquare
 kkk=kk;
  for ii=1:n-1
   kkk=kkk+1;
   ui(kkk,1)=sym(1-ii/n);
   vi(kkk,1)=sym(1);
   %wi(kkk,1)=0;
 end;

 %LEFT EDGE OF THE UNIT SQUARE
 kkkk=kkk;
  for iii=1:n-1
   kkkk=kkkk+1;
   ui(kkkk,1)=0;
   vi(kkkk,1)=sym(1-iii/n);
   %wi(kkkk,1)=sym(iii/n);
  end
end%if (n-1)>0
%INTERIOR NODES EXIST ONLY FOR N>1
if (n-1)>0
  kkkkk=kkkk;
  for iiii=1:(n-1)
   for jjjj=1:(n-1)
        kkkkk=kkkkk+1;
        ui(kkkkk,1)=sym(jjjj/n);
        vi(kkkkk,1)=sym(iiii/n);
       % wi(kkkkk,1)=sym(1-ui(kkkkk,1)-vi(kkkkk,1));
    end
  end
end%if (n-1)>0
%if n==2

 %   num=(1:6)';
 %else
  num=(1:kkkkk)';

  %end
%disp([ui'])
%disp([vi'])
%disp([wi'])
%length(ui)
%length(vi)
%length(wi)

%disp([num ui vi wi])
[eln,nodes,nodetel]=nodaladdresses4special_convex_quadrilaterals_in_QUAD(n)
qq=(n+1)*(n+1);
   nc=(n)^2/2;
for pp=1:nc
 qq=qq+1;
 q1=eln(pp,1);
 q2=eln(pp,2);
 q3=eln(pp,3);
 ui(qq,1)=(ui(q1,1)+ui(q2,1)+ui(q3,1))/3;
 vi(qq,1)=(vi(q1,1)+vi(q2,1)+vi(q3,1))/3;
 %wi(qq,1)=1-ui(qq,1)-vi(qq,1);
end
 %disp([ui vi wi])
%length(ui)
%length(vi)
%length(wi)
nnode=qq;
[nel,nnel]=size(nodes);
```

```
 num=(1:qq)';
disp([num ui vi])
%
 [nel,nnel]=size(nodes)
 PROGRAM(3): nodaladdresses4special_convex_quadrilaterals_in_QUAD.m
function[eln,nodes,nodetel]=nodaladdresses4special_convex_quadrilaterals_in_QUAD(n)
%eln=6-node triangles with centroid
%spqd=4-node special convex quadrilateral
%n must be even,i.e.n=2,4,6,.......
syms mst_tri x
%disp('vertex nodes of the arbitrary linear convex quadrilateral')
elm(1,1)=1;
elm(n+1,1)=2;
elm((n+1)*(n+1),1)=3;
elm((n+1)*n+1,1)=4;
%generate the nodal addresses for base edge
kk=4;
for k=2:n
   kk=kk+1;
   elm(k,1)=kk;
end
%generate the nodal addresses for left edge
nni=1;
for i=0:(n-2)
   nni=nni+(n+1);
   elm(nni,1)=4*n-i;
end
%generate the nodal addresses for right edge
nni=n+1;
for i=0:(n-2)
   nni=nni+(n+1);
   elm(nni,1)=(n+4)+i;
end
%generate the nodal addresses for top edge
nni=(n+1)^2;
for i=0:(n-2)
   nni=nni-1;
   elm(nni,1)=(2*n+3)+i;
end
%generate interior nodes
nni=1;jj=0;
for i=1:(n-1)
   nni=nni+(n+1);
   for j=1:(n-1)
      jj=jj+1;
      nnj=nni+j;
      elm(nnj,1)=4*n+jj;
   end
end
%row nodes bottom(base) to top
jj=0;kk=0;
for j=0:n
   jj=j+1;
for k=1:(n+1)
   kk=kk+1;
   row_nodes(jj,k)=elm(kk,1);
end
end
%for jj=(n+1):-1:1
%    (row_nodes(jj,:));
%end
%[row_nodes]
rr=row_nodes;
rr
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%element nodal address computations
%n must be an even number
if rem(n,2)==0
```

```
ne=0;N=n+1;

for k=1:2:n
%N=N-2;
i=k;
for j=1:2:N-2
   ne=ne+1;
eln(ne,1)=rr(i,j);
eln(ne,2)=rr(i,j+2);
eln(ne,3)=rr(i+2,j);
eln(ne,4)=rr(i,j+1);
eln(ne,5)=rr(i+1,j+1);
eln(ne,6)=rr(i+1,j);
end%i%j
%me=ne
%N-2
%if (N-2)>0
for jj=1:2:N-2
ne=ne+1;
eln(ne,1)=rr(i+2,jj+2);
eln(ne,2)=rr(i+2,jj);
eln(ne,3)=rr(i,jj+2);
eln(ne,4)=rr(i+2,jj+1);;
eln(ne,5)=rr(i+1,jj+1);
eln(ne,6)=rr(i+1,jj+2);
end%jj%i
%end
end%k
end%if rem
%ne
%for kk=1:ne
%[eln(kk,1:6)]
%end
%add node numbers for element centroids

nnd=(n+1)*(n+1);
for kkk=1:ne
   nnd=nnd+1;
   eln(kkk,7)=nnd;
end
%for kk=1:ne
%[eln(kk,1:7)]
%end

%to generate special quadrilaterals
% and the spanning triangle
mm=0;
for iel=1:ne
   for jel=1:3
   mm=mm+1;
      switch jel
       case 1
        nodes(mm,1:4)=[eln(iel,7) eln(iel,6) eln(iel,1) eln(iel,4)];
        nodetel(mm,1:3)=[eln(iel,2) eln(iel,3) eln(iel,1)];
        case 2
        nodes(mm,1:4)=[eln(iel,7) eln(iel,4) eln(iel,2) eln(iel,5)];
        nodetel(mm,1:3)=[eln(iel,3) eln(iel,1) eln(iel,2)];
        case 3
        nodes(mm,1:4)=[eln(iel,7) eln(iel,5) eln(iel,3) eln(iel,6)];
        nodetel(mm,1:3)=[eln(iel,1) eln(iel,2) eln(iel,3)];
      end
    end
end

spqd=nodes;

%for mmm=1:mm
   %spqd(:,1:4)
```

```
 %end
%
%ss1='number of 6-node triangles with centroid=';
%[p1,q1]=size(eln);
%disp([ss1 num2str(p1)])
%
eln
%
%ss2='number of 4-node special convex quadrilaterals =';
%[p2,q2]=size(spqd);
%disp([ss2 num2str(p2)])
%
```

PROGRAM(4)

```
function[]=quadrilateral_mesh4convexpolygoneightsidesq4(n1,n2,n3,nmax,numtri,ndiv,mesh,xlength,ylength)
clf
%(1)=generate 2-D quadrilateral mesh
%for a rectangular shape of domain
%quadrilateral_mesh_q4(xlength,ylength)
%xnode=number of nodes along x-axis
%ynode=number of nodes along y-axis
%xzero=x-coord of bottom left corner
%yzero=y-coord of bottom left corner
%xlength=size of domain alog x-axis
%ylength=size of domain alog y-axis
%quadrilateral_mesh4MOINEX_q4([1;1;1;1;1;1;1],[2;3;4;5;6;7;8],[3;4;5;6;7;8;2],8,1,2,1,1,1)
%quadrilateral_mesh4MOINEX_q4([1;1;1;1;1;1;1],[2;3;4;5;6;7;8],[3;4;5;6;7;8;2],8,4,4,1,1,1)
%quadrilateral_mesh4MOINEX_q4([1;1;1;1;1;1;1;1],[2;3;4;5;6;7;8;9],[3;4;5;6;7;8;9;2],9,1,2,2,1,1)
%quadrilateral_mesh4MOINEX_q4([1;1;1;1;1;1;1;1],[2;3;4;5;6;7;8;9],[3;4;5;6;7;8;9;2],9,4,4,2,1,1)
%quadrilateral_mesh4MOINEX_q4([1;1;1;1;1;1;1;1],[2;3;4;5;6;7;8;9],[3;4;5;6;7;8;9;2],9,1,2,4,1,1)
%quadrilateral_mesh4convexpolygonsixside_q4([1;1;1;1;1;1;1;1],[2;3;4;5;6;7],[3;4;5;6;7;2],7,1,2,5,1,1)
%quadrilateral_mesh4convexpolygonsixside_q4([1;1;1;1;1;1;1;1],[2;3;4;5;6;7],[3;4;5;6;7;2],7,4,4,5,1,1)
%quadrilateral_mesh4convexpolygonsixside_q4([1;1;1;1;1;1;1;1;1],[2;3;4;5;6;7;8],[3;4;5;6;7;8;2],8,1,2,8,1,1)
%quadrilateral_mesh4convexpolygonsixside_q4([1;1;1;1;1;1;1;1;1],[2;3;4;5;6;7;8],[3;4;5;6;7;8;2],8,4,4,8,1,1)
%quadrilateral_mesh4convexpolygoneightsideq4([9;9;9;9;9;9;9;9],[1;2;3;4;5;6;7;8],[2;3;4;5;6;7;8;1],9,1,2,9,1,1)
%[eln,spqd,rrr,nodes,nodetel]=nodaladdresses4special_convex_quadrilateralsQUADtrial([9;9;9;9;9;9;9;9],[1;2;3;4;5;6;7;8],[2;3;4;5;6;7;8;1],
9,1,2)
[coord,gcoord,nodes,nodetel,nnode,nel]=polygonal_domain_QUADcoordinates(n1,n2,n3,nmax,numtri,ndiv,mesh)
[nel,nnel]=size(nodes);

disp([xlength,ylength,nnode,nel,nnel])
%gcoord(i,j),where i->node no. and j->x or y
%_____________________________________________________________________

%plot the mesh for the generated data
%x and y coordinates
xcoord(:,1)=gcoord(:,1);
ycoord(:,1)=gcoord(:,2);
%extract coordinates for each element
clf
for i=1:nel
for j=1:nnel
x(1,j)=xcoord(nodes(i,j),1);
y(1,j)=ycoord(nodes(i,j),1);
end;%j loop
xvec(1,1:5)=[x(1,1),x(1,2),x(1,3),x(1,4),x(1,1)];
yvec(1,1:5)=[y(1,1),y(1,2),y(1,3),y(1,4),y(1,1)];
%axis equal
axis tight
switch mesh
case 1
axis([0 xlength 0 ylength])
case 2
axis([0 xlength 0 ylength])
case 3
 xl=xlength/2;yl=ylength/2;
axis([-xl xl -yl yl])
case 4
```

```
xl=xlength/2;yl=ylength/2;
axis([-xl xl -yl yl])
case 5
axis([0 xlength 0 ylength])
case 8
   axis([0 xlength 0 ylength])
case 9
   axis([0 xlength 0 ylength])
end
 %
plot(xvec,yvec);%plot element
hold on;
%place element number
if (ndiv<=4)
midx=mean(xvec(1,1:4))
midy=mean(yvec(1,1:4))
text(midx,midy,['[',num2str(i),']']);
end% if ndiv
end;%i loop
xlabel('x axis')
ylabel('y axis')
st1='Mesh With ';
st2=num2str(nel);
st3=' Four Noded  ';
st4='Quadrilateral';
st5=' Elements'
st6='& Nodes='
st7=num2str(nnode);
title([st1,st2,st3,st4,st5,st6,st7])
%put node numbers
if (ndiv<=4)
for jj=1:nnode
text(gcoord(jj,1),gcoord(jj,2),['o',num2str(jj)]);
end
end%if ndiv
%axis off
```

PROGRAM(5) polygonal_domain_QUADcoordinates

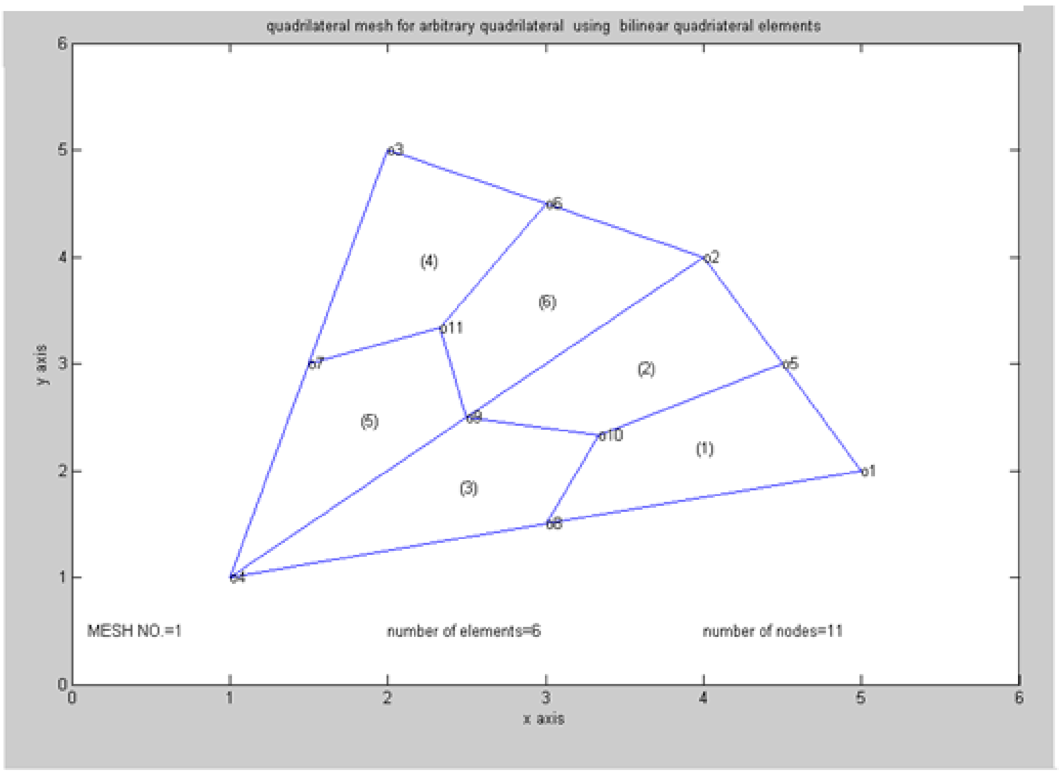
```
function[coord,gcoord,nodes,nodetel,nnode,nel]=polygonal_domain_QUADcoordinates(n1,n2,n3,nmax,numtri,n,mesh)
%n1=node number at(0,0)for  a choosen triangle
%n2=node number at(1,0)for  a choosen triangle
%n3=node number at(0,1)for  a choosen triangle
%eln=6-node triangles with centroid
%spqd=4-node special convex quadrilateral
%n must be even,i.e.n=2,4,6,.......i.e number of divisions
%nmax=one plus the number of segments of the polygon
%nmax=the number of segments of the polygon plus a node interior to the polygon
%numtri=number of T6 triangles in each segment i.e a triangle formed by
%joining the end poits of the segment to the interior point(e.g:the centroid) of the polygon
%[eln,spqd]=nodaladdresses_special_convex_quadrilaterals_trial(n1=1,n2=2,n3=3,nmax=3,n=2,4,6,...)
%[eln,spqd]=nodaladdresses_special_convex_quadrilaterals_trial([1;1;1;1],[2;3;4;5],[3;4;5;2],5,1,2)
%[eln,spqd]=nodaladdresses_special_convex_quadrilaterals_trial([1;1;1;1],[2;3;4;5],[3;4;5;2],5,4,4)
%[eln,spqd]=nodaladdresses_special_convex_quadrilaterals_trial([1;1;1;1],[2;3;4;5],[3;4;5;2],5,9,6)
%[eln,spqd]=nodaladdresses_special_convex_quadrilaterals_trial([1;1;1;1],[2;3;4;5],[3;4;5;2],5,16,8)
%PARVIZ MOIN EXAMPLE
syms U V W xi yi

switch mesh
case 1%for MOIN POLYGON
x=sym([1/2;1/2;1;  1;1/2;0;  0;0])%for MOIN EXAMPLE
y=sym([1/2;  0;0;1/2;  1;1;1/2;0])%for MOIN EXAMPLE


case 2%for a unit square: 0<=x,y<=1
x=sym([1/2;1/2;1;  1;  1;1/2;0;  0;0])%FOR UNIT SQUARE
y=sym([1/2;  0;0;1/2;  1;  1;1;1/2;0])%FOR UNIT SQUARE

case 3%for A POLYGON like MOIN OVER(-1/2)<=x,y<=(1/2)
```

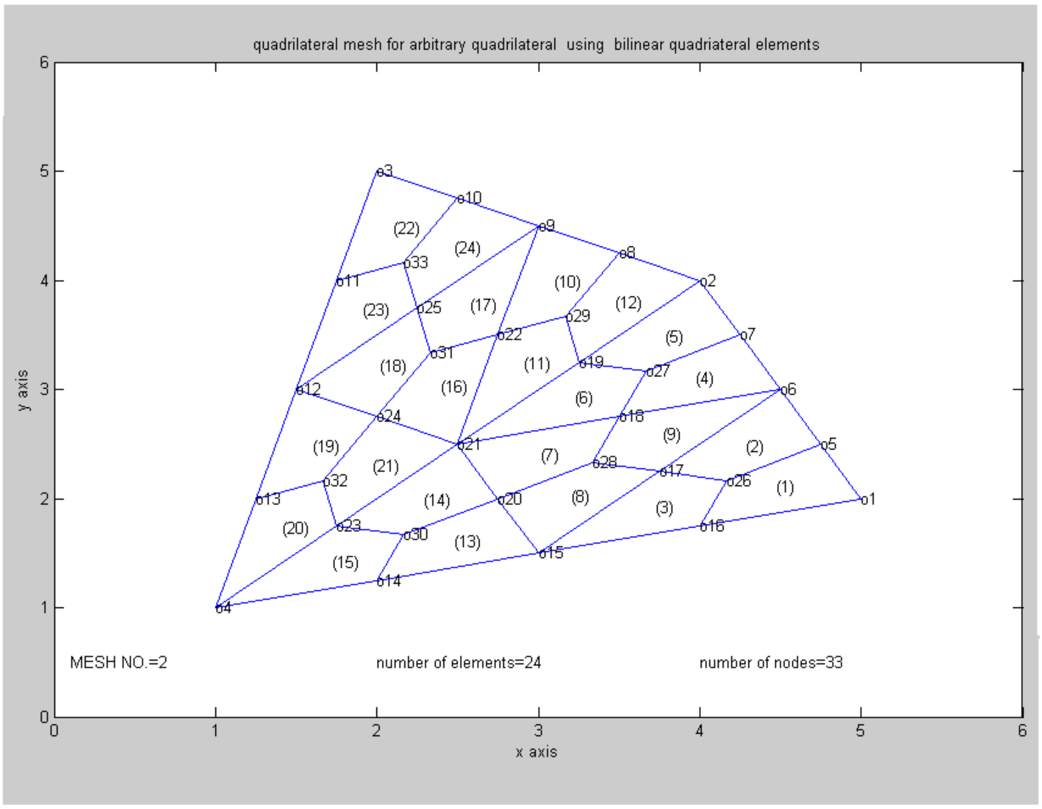
```
  % 1   2   3  4    5  6   7  8
 x=sym([0;   0; 1/2;1/2;  0;-1/2;-1/2;-1/2])
 y=sym([0;-1/2;-1/2;  0;1/2; 1/2;   0;-1/2])

case 4%for a unit square: -0.5<=x,y<=0.5
 % 1   2   3  4  5  6   7  8
 x=sym([0;   0; 1/2;1/2;1/2;  0;-1/2;-1/2;-1/2])
 y=sym([0;-1/2;-1/2;  0;1/2;1/2; 1/2;   0;-1/2])
case 5%for a convexpolygonsixside
 %   1   2  3  4   5  6 7
x=sym([0.5;0.1;0.7;1;.75;.5;0])
y=sym([0.5;0;0.2;.5;.85;1;.25])
case 6%standard triangle
x=sym([0;1;0])
y=sym([0;0;1])
case 7%equilateral triangle

x=sym([0;1;1/2])
y=sym([0;0;sqrt(3)/2])
case 8%for a convexpolygonsixside
 %   1   2  3  4   5  6  7  8
x=sym([0.5;0.1;0.7;1;.75;.5;.25;0])
y=sym([0.5;0;0.2;.5;.85;1;.625;.25])
case 9
%   1  2   3  4   5    6  7    8   9
x=([.2;0.5;.8;1.0;.75;0.5;0.25;0.0;0.5])
y=([.2;.05;.2;0.5;.85;1.0;0.90;0.6;0.5])
case 10
%   1  2   3  4   5    6  7    8   9
x=([0.5;.2;0.5;.8;1.0;.75;0.5;0.25;0.0])
y=([0.5;.2;.05;.2;0.5;.85;1.0;0.90;0.6])


end
if (nmax>3)
[eln,spqd,rrr,nodes,nodetel]=nodaladdresses4special_convex_quadrilateralsQUADtrial(n1,n2,n3,nmax,numtri,n)
%[eln,spqd,rrr,nodes,nodetel]=nodaladdresses_special_convex_quadrilateralsQUADtrial(n1,n2,n3,nmax,numtri,n);
end
if (nmax==3)
  [eln,spqd,rrr,nodes,nodetel]=nodaladdresses_special_convex_quadrilaterals(n);
end
[U,V,W]=generate_area_coordinate_over_the_standard_triangle(n);
ss1='number of 6-node triangles with centroid=';
[p1,q1]=size(eln);
disp([ss1 num2str(p1)])
%
eln
%
ss2='number of special convex quadrilaterals elements&nodes per element =';
[nel,nnel]=size(spqd);
disp([ss2 num2str(nel) ','  num2str(nnel)])
%
spqd
%
nnode=max(max(spqd));
ss3='number of nodes of the triangular domain& number of special quadrilaterals=';
disp([ss3 num2str(nnode) ',' num2str(nel)])



nitri=nmax-1;
if (nmax==3)nitri=1
end
for itri=1:nitri
disp('vertex nodes of the itri triangle')
[n1(itri,1) n2(itri,1) n3(itri,1)]
x1=x(n1(itri,1),1)
x2=x(n2(itri,1),1)
```

```
x3=x(n3(itri,1),1)
%
y1=y(n1(itri,1),1)
y2=y(n2(itri,1),1)
y3=y(n3(itri,1),1)
rrr(:,:,itri)
U'
V'
W'
kk=0;
for ii=1:n+1
   for jj=1:(n+1)-(ii-1)
      kk=kk+1;
      mm=rrr(ii,jj,itri);
      uu=U(kk,1);vv=V(kk,1);ww=W(kk,1);
      xi(mm,1)=x1*ww+x2*uu+x3*vv;
      yi(mm,1)=y1*ww+y2*uu+y3*vv;
   end
end
[xi yi]
%add coordinates of centroid
 ne=(n/2)^2;
% stdnode=kk;
 for iii=1+(itri-1)*ne:ne*itri
    %kk=kk+1;
    node1=eln(iii,1)
    node2=eln(iii,2)
    node3=eln(iii,3)
    mm=eln(iii,7)
    xi(mm,1)=(xi(node1,1)+xi(node2,1)+xi(node3,1))/3;
    yi(mm,1)=(yi(node1,1)+yi(node2,1)+yi(node3,1))/3;
 end

end
N=(1:nnode)'
[N xi yi]
%
coord(:,1)=(xi(:,1));
coord(:,2)=(yi(:,1));
gcoord(:,1)=double(xi(:,1));
gcoord(:,2)=double(yi(:,1));
%disp(gcoord)
```

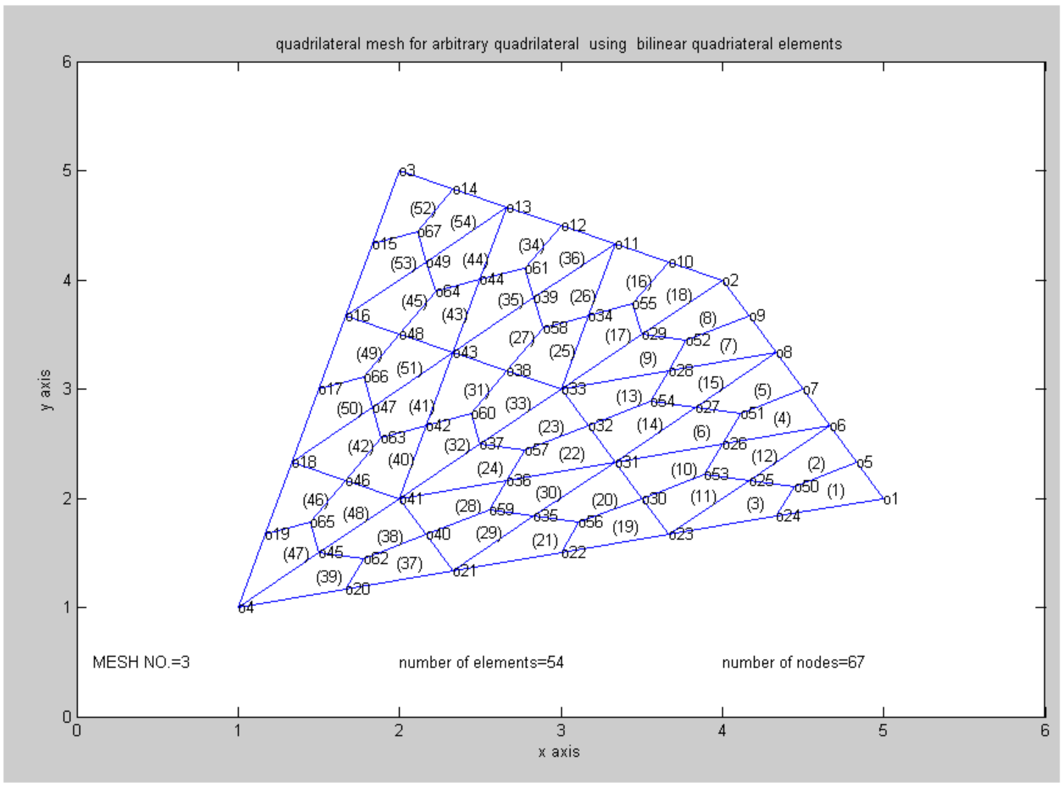
FIGURES

(1) MESHGENERATION OF AN ARBITRARY QUADRILATERAL

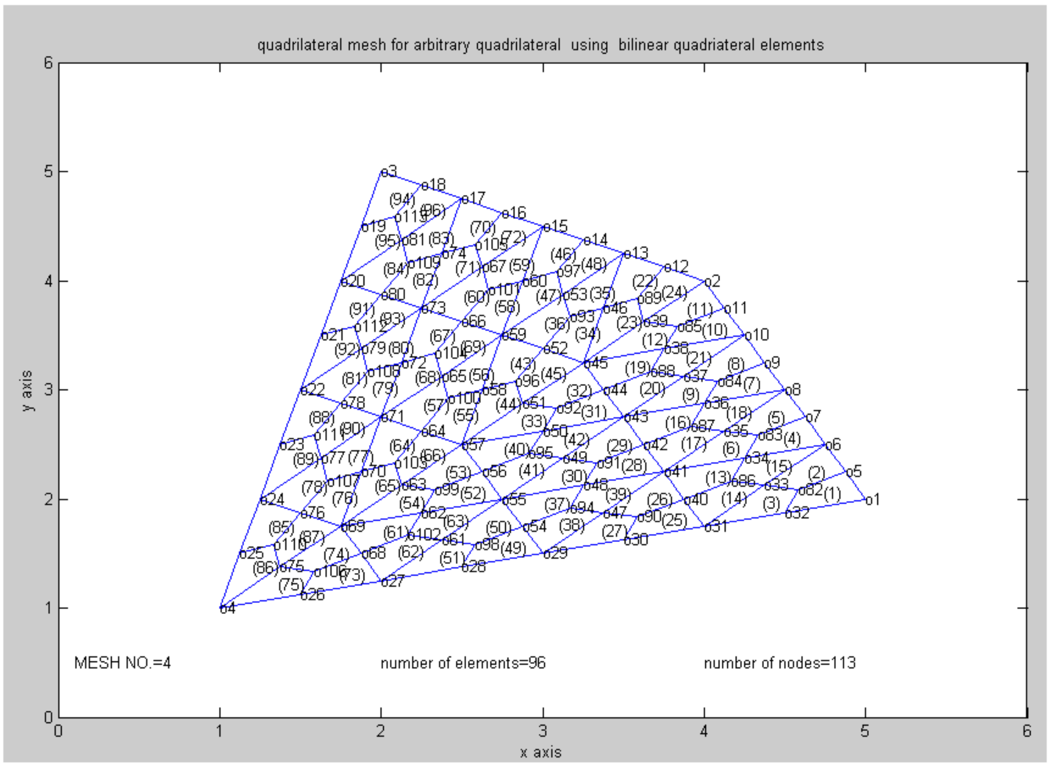


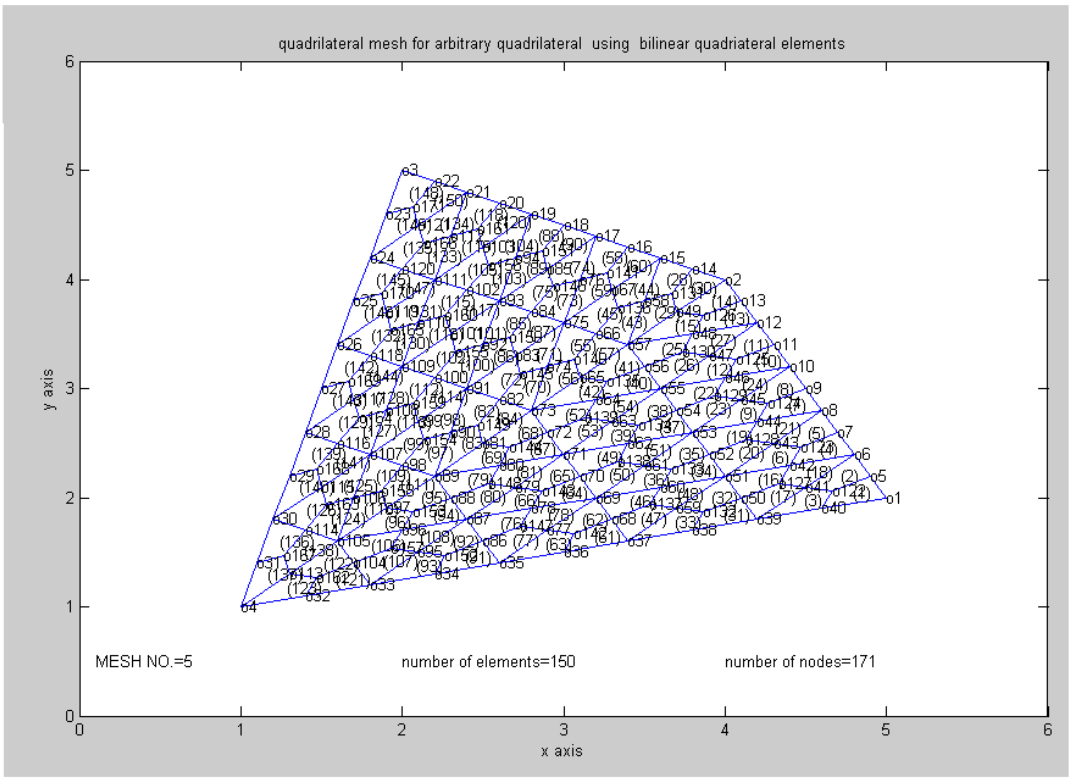


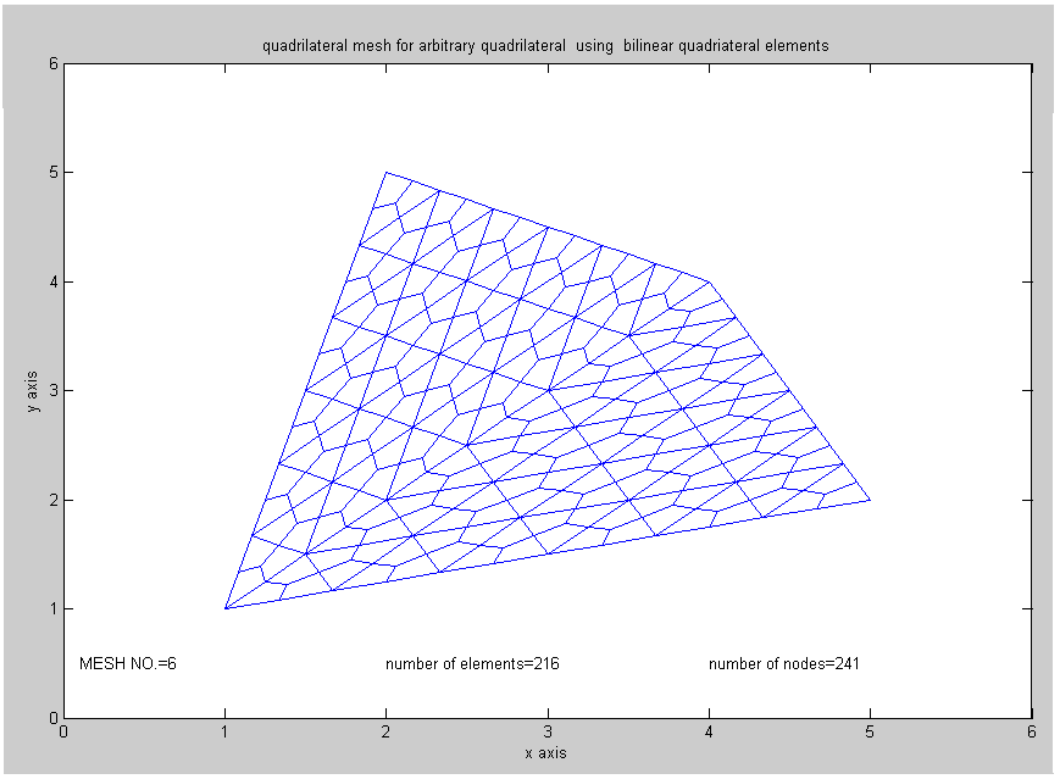
quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=2
number of elements=24
number of nodes=33

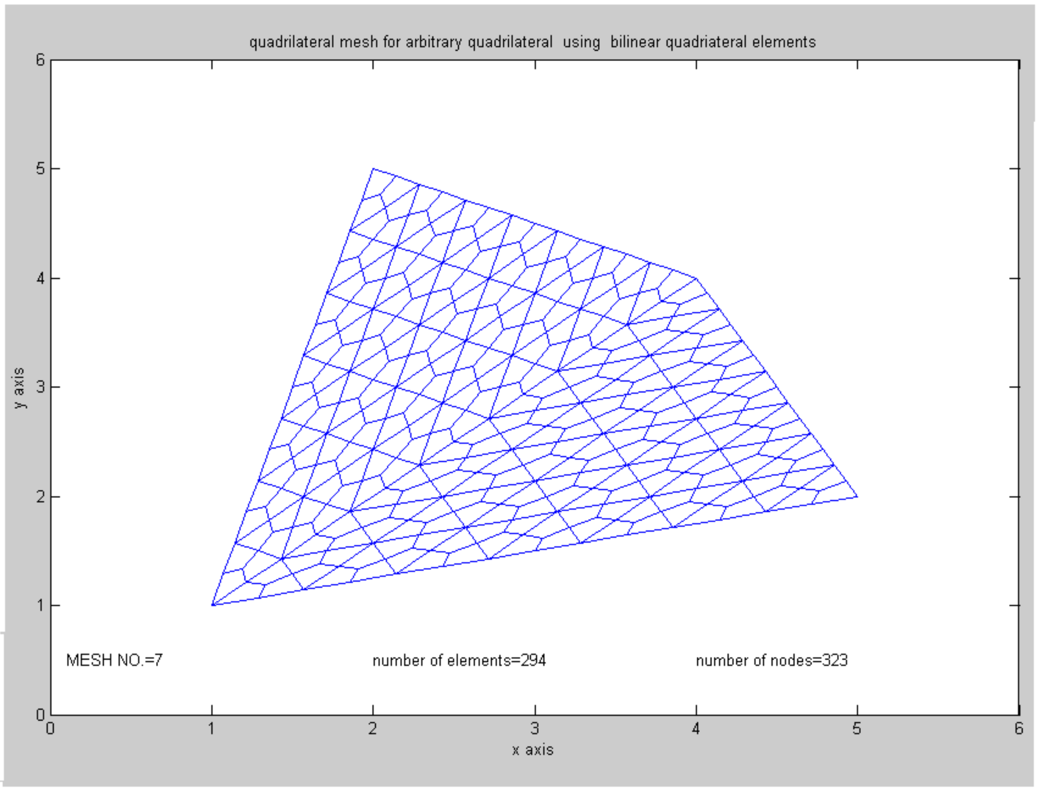


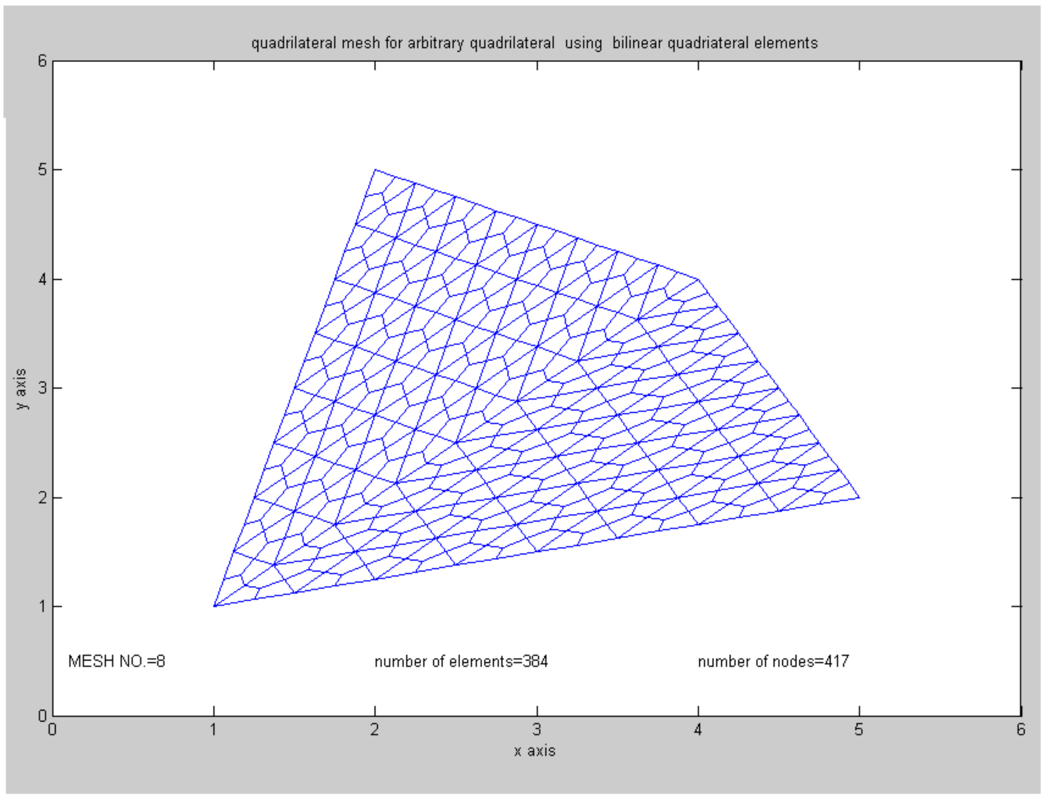
quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=3
number of elements=54
number of nodes=67

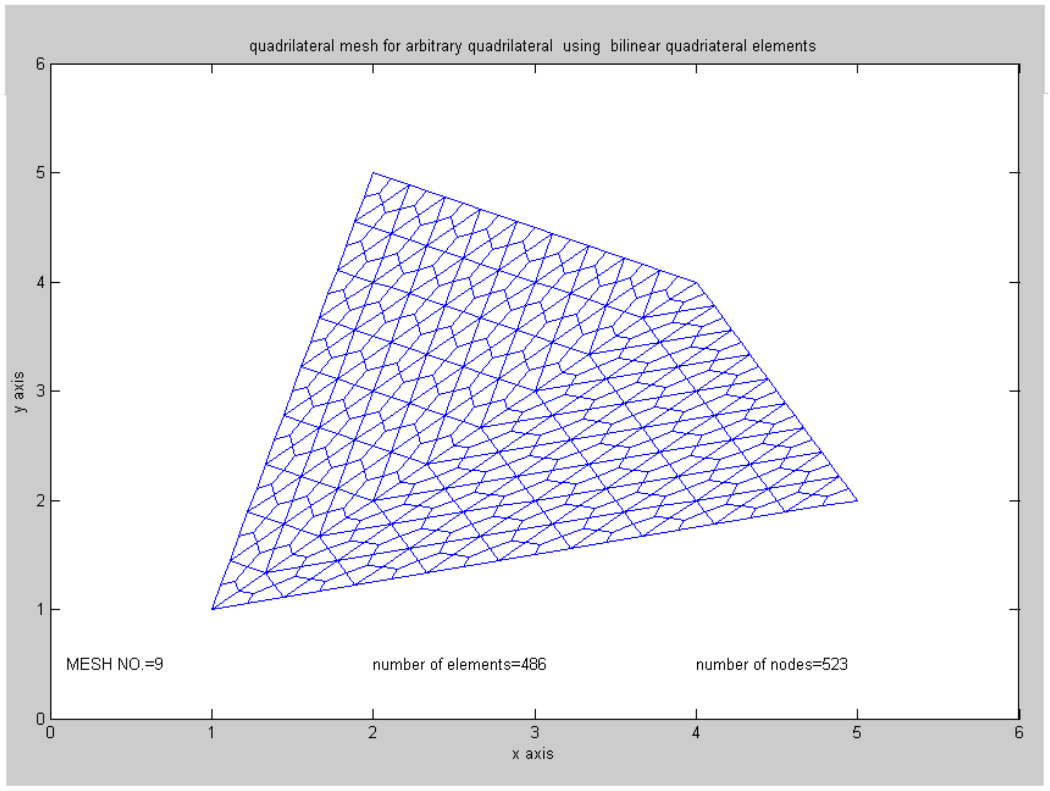

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
MESH NO.=4
number of elements=96
number of nodes=113
x axis
y axis

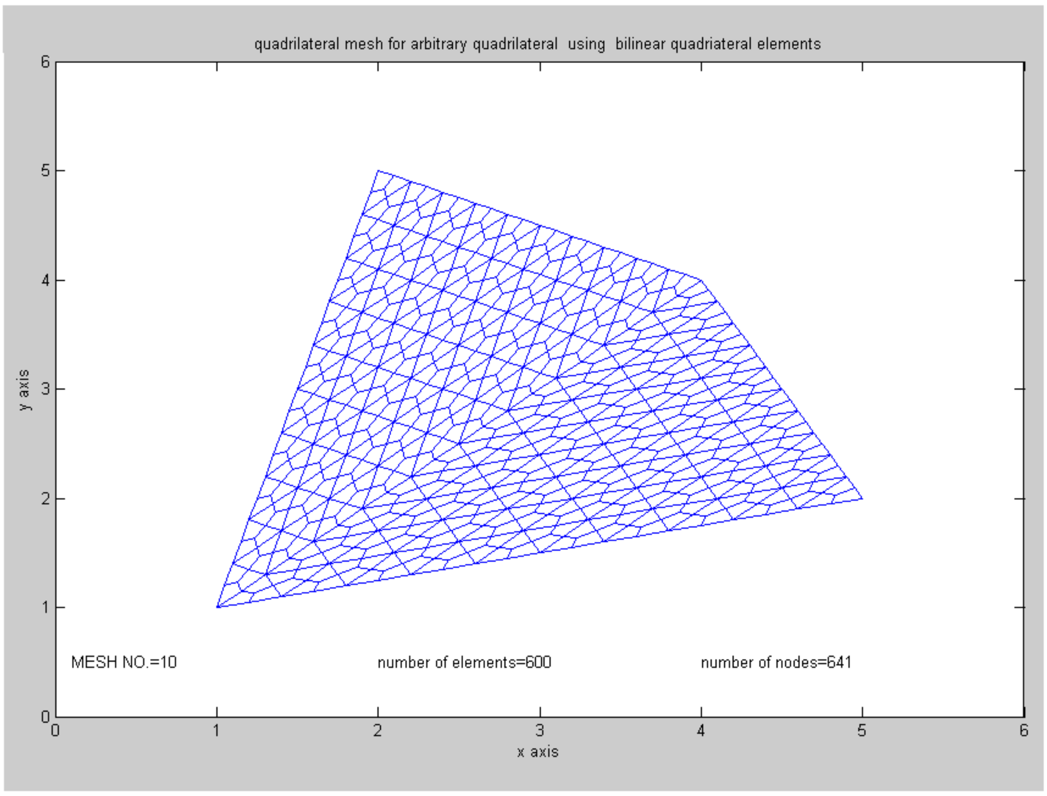

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
MESH NO.=5
number of elements=150
number of nodes=171
x axis
y axis


quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=6
number of elements=216
number of nodes=241

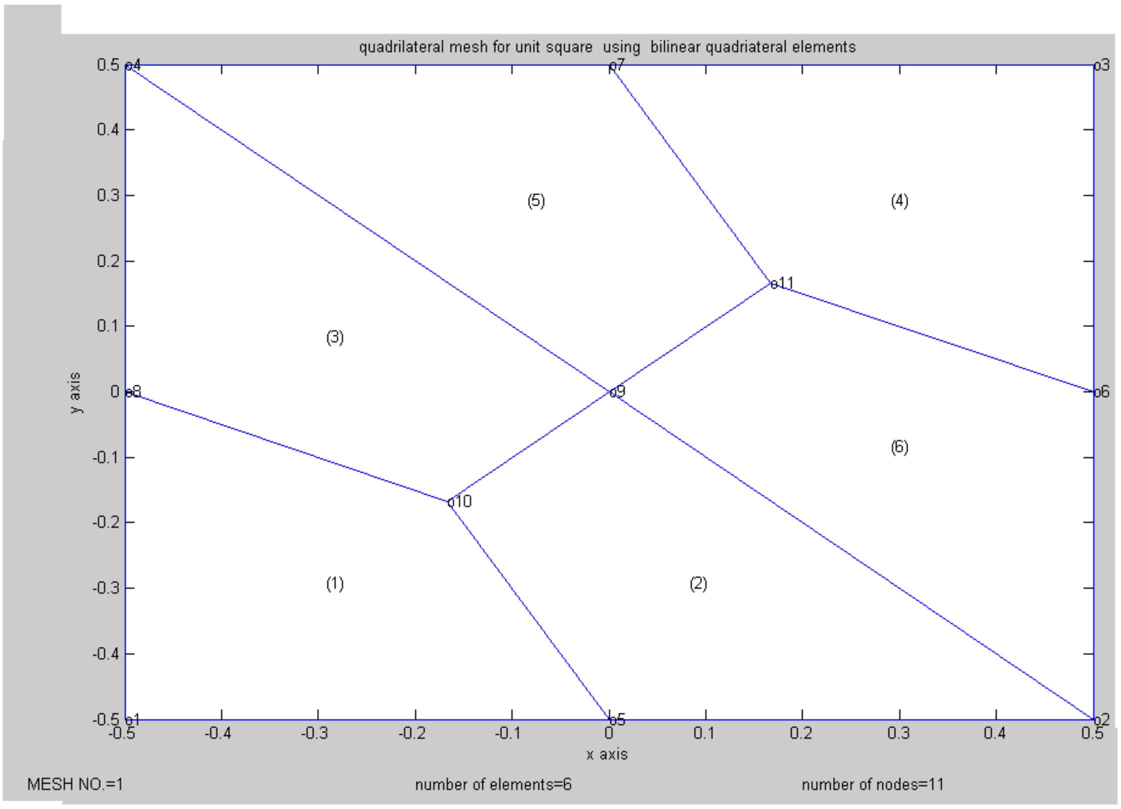

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=7
number of elements=294
number of nodes=323

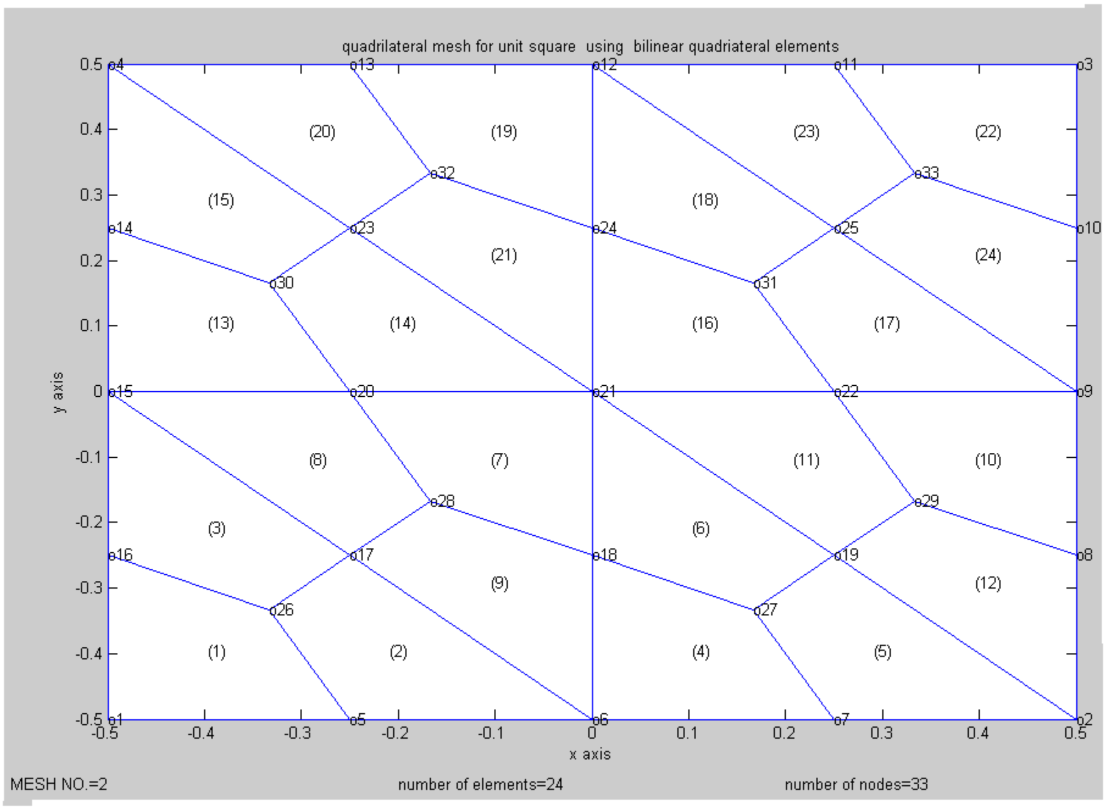

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=8
number of elements=384
number of nodes=417

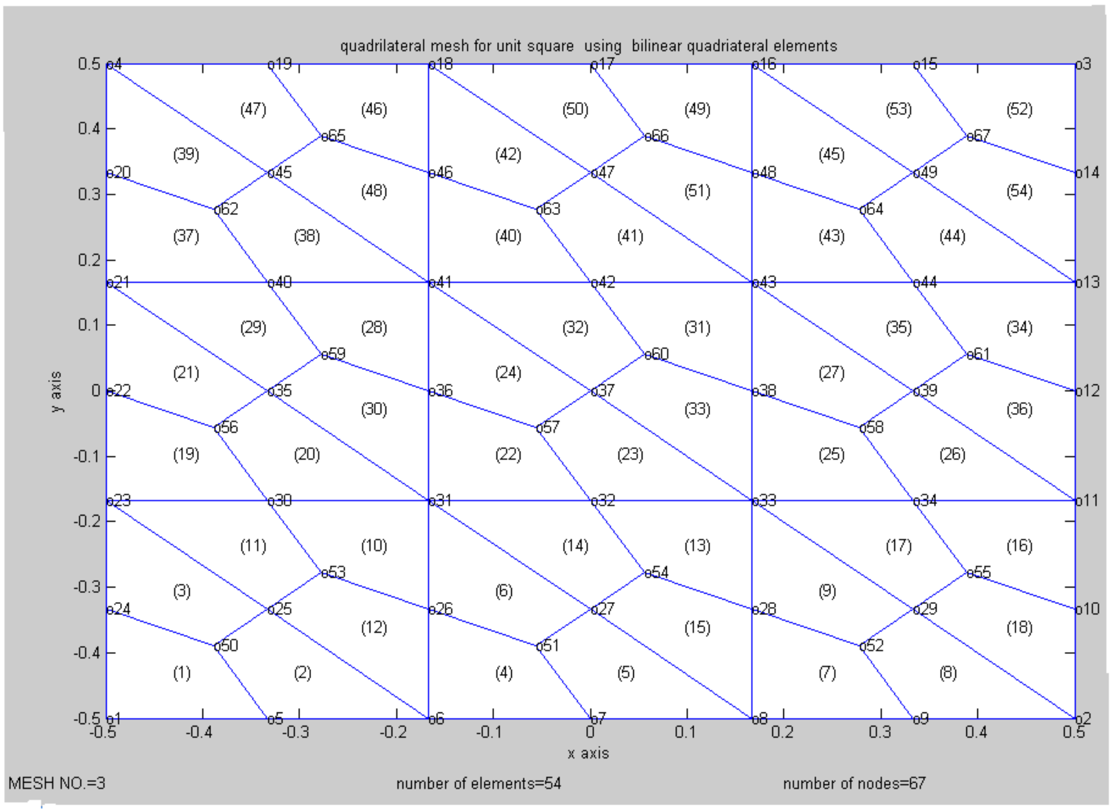

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
y axis
x axis
MESH NO.=9
number of elements=486
number of nodes=523

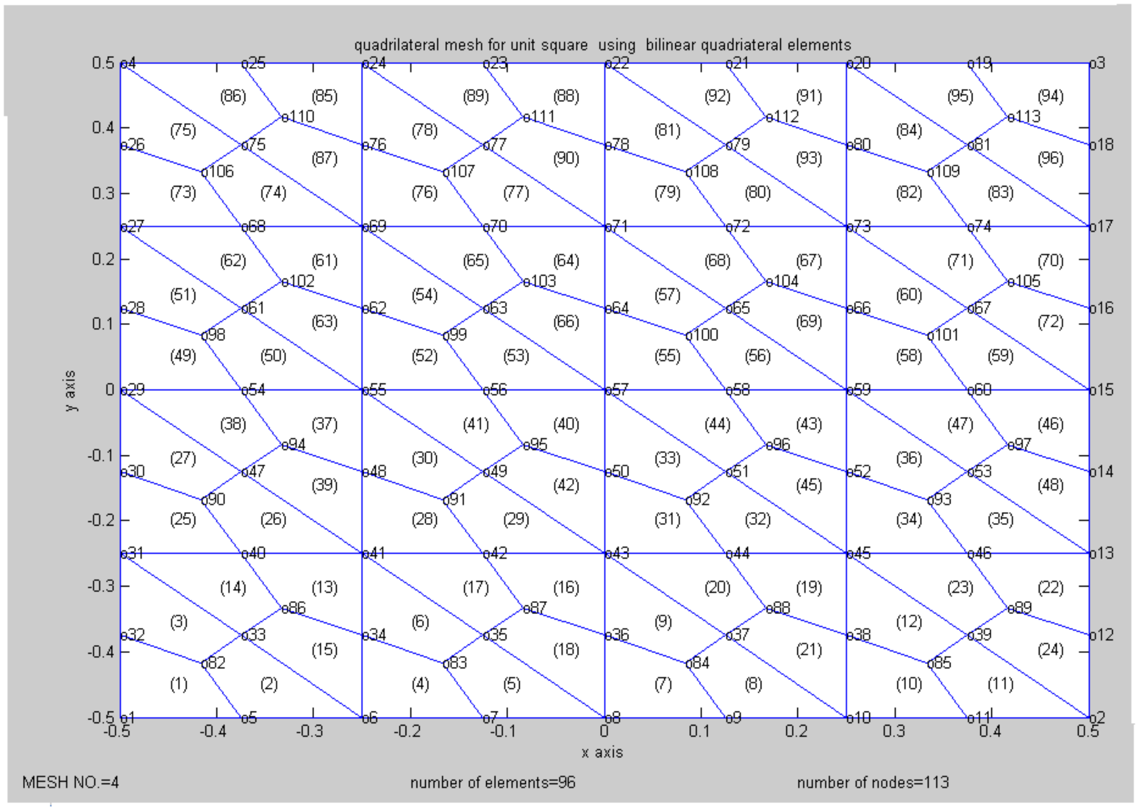

quadrilateral mesh for arbitrary quadrilateral using bilinear quadriateral elements
MESH NO.=10
number of elements=600
number of nodes=641
x axis
y axis

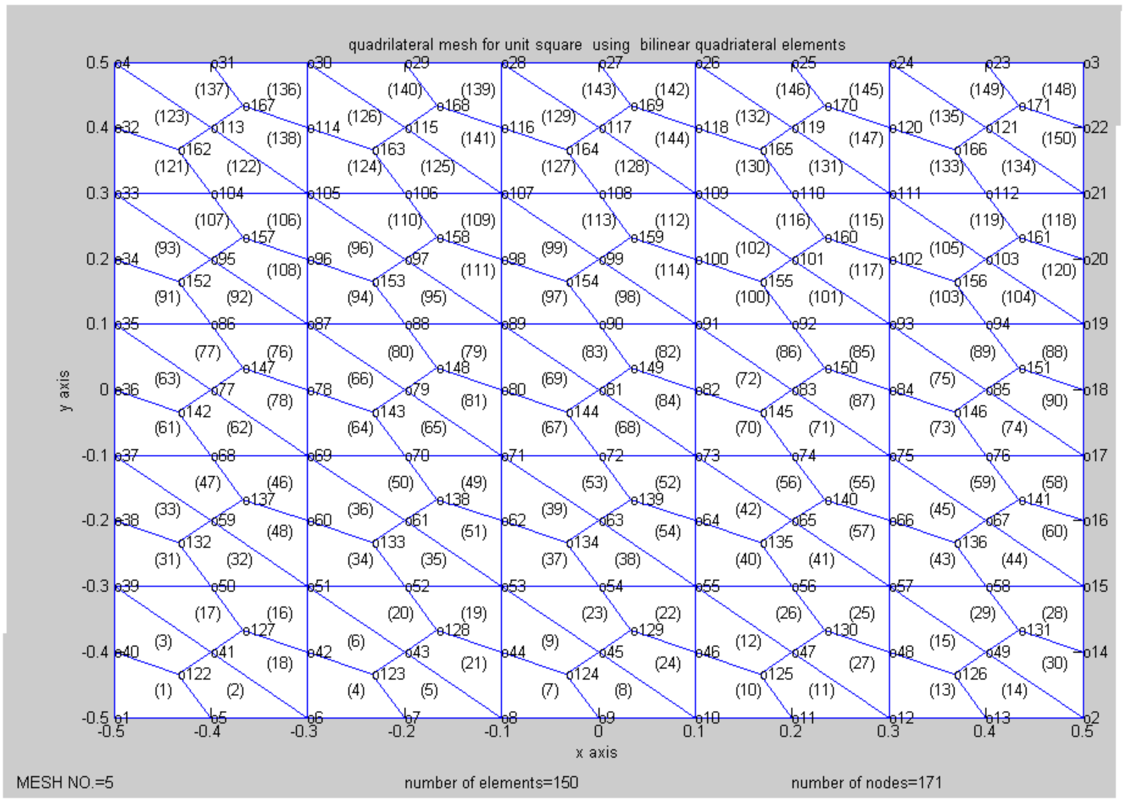
(2) MESH GENERATION OF A UNIT SQUARE $-0.5<=x,y<=+0.5$

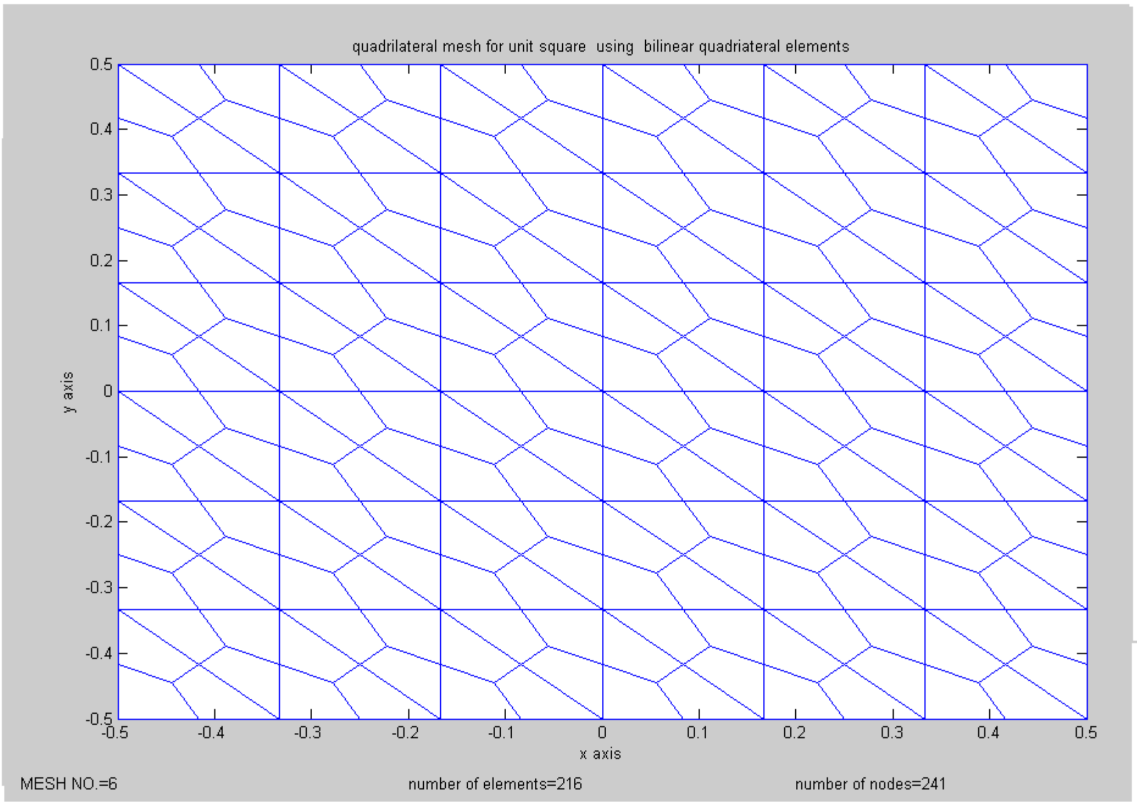


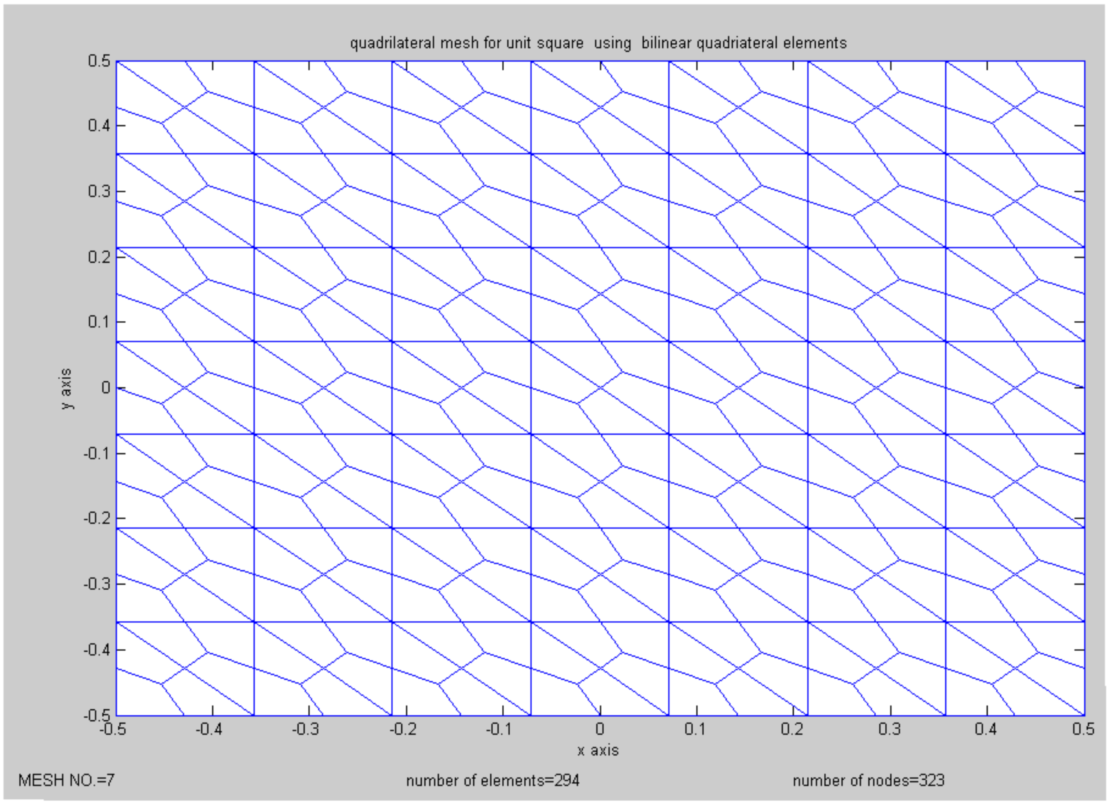

quadrilateral mesh for unit square using bilinear quadriateral elements
x axis
y axis
MESH NO.=2
number of elements=24
number of nodes=33

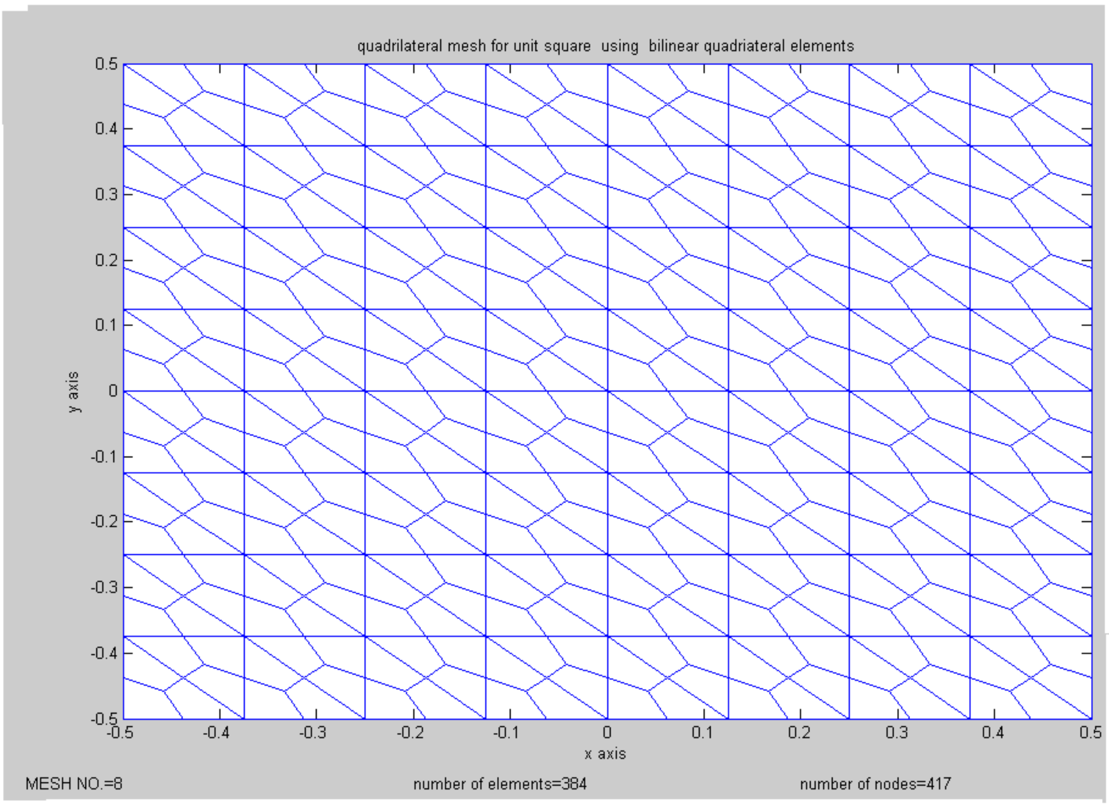

quadrilateral mesh for unit square using bilinear quadriateral elements
x axis
y axis
MESH NO.=3
number of elements=54
number of nodes=67

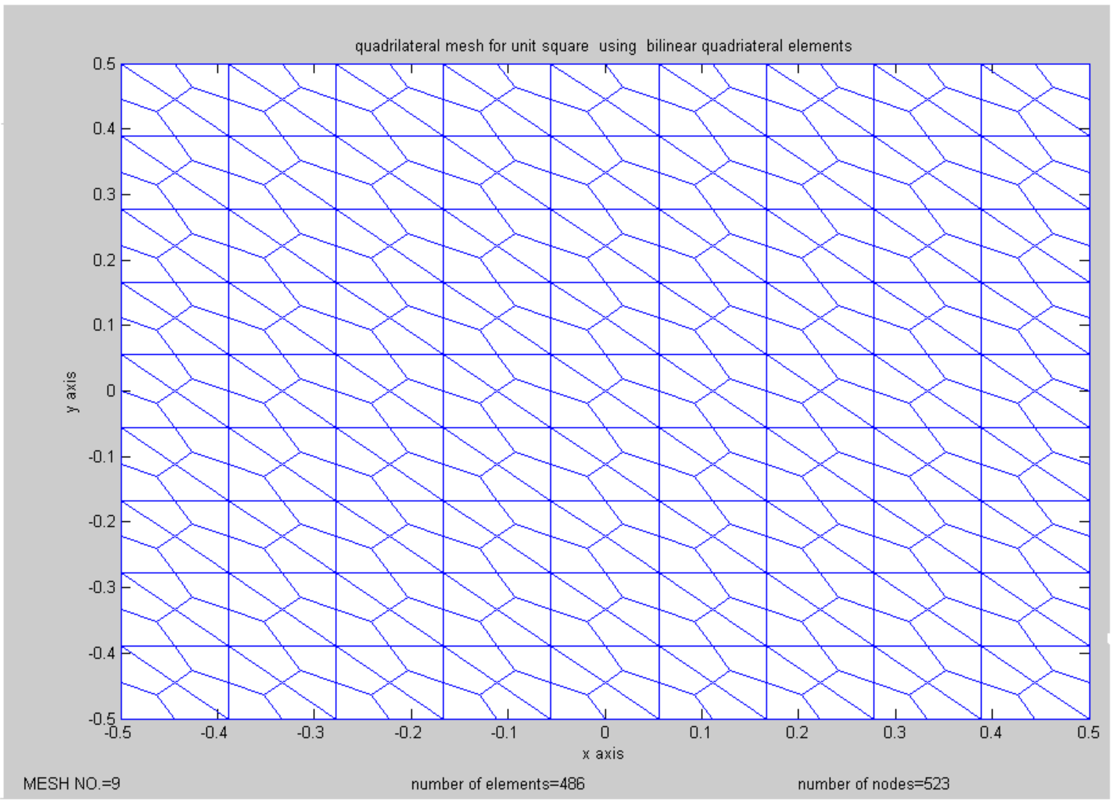

quadrilateral mesh for unit square using bilinear quadriateral elements
x axis
y axis
MESH NO.=4
number of elements=96
number of nodes=113

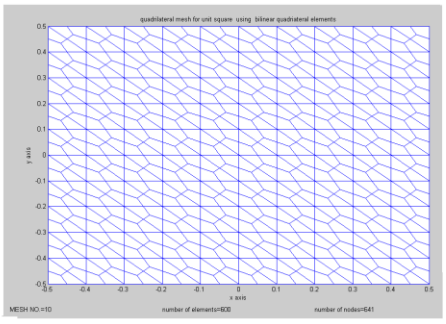

quadrilateral mesh for unit square using bilinear quadriateral elements
x axis
y axis
MESH NO.=5
number of elements=150
number of nodes=171


quadrilateral mesh for unit square using bilinear quadriateral elements
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
y axis
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
x axis
MESH NO.=6
number of elements=216
number of nodes=241


quadrilateral mesh for unit square using bilinear quadriateral elements
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
y axis
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
x axis
MESH NO.=7
number of elements=294
number of nodes=323

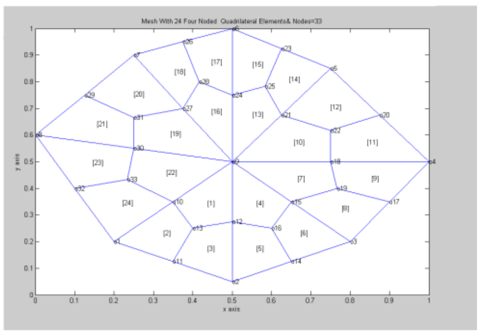

quadrilateral mesh for unit square using bilinear quadriateral elements
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
y axis
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
x axis
MESH NO.=8
number of elements=384
number of nodes=417


quadrilateral mesh for unit square using bilinear quadriateral elements
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
y axis
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
x axis
MESH NO.=9
number of elements=486
number of nodes=523

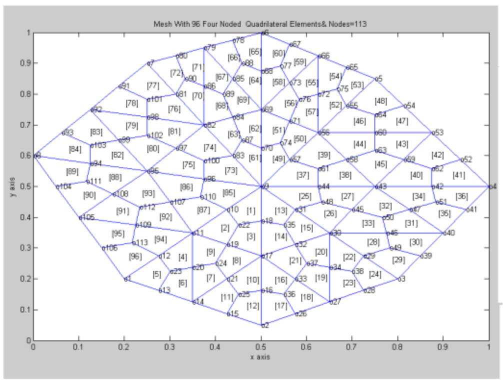

quadrilateral mesh for unit square using bilinear quadrilateral elements
y axis
x axis
MESH NO =10
number of elements=600
number of nodes=641

(3) MESH GENERATION FOR AN EIGHT SIDE CONVEX POLYGON

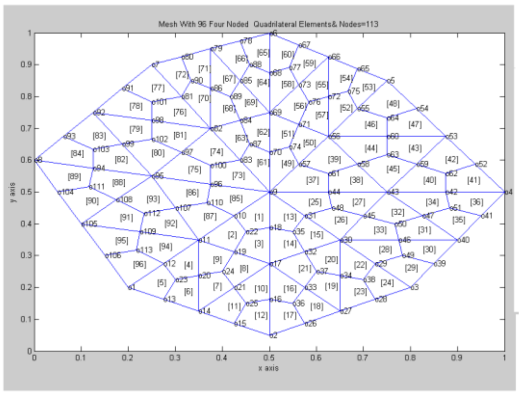
MESH NO.01



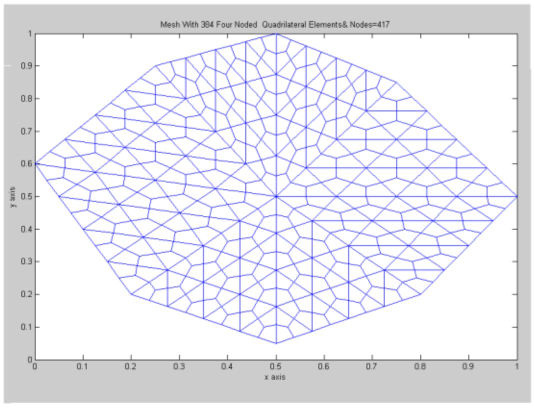
MESH NO.02



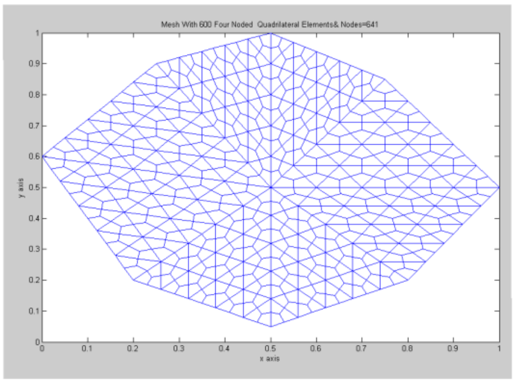
MESH NO.03



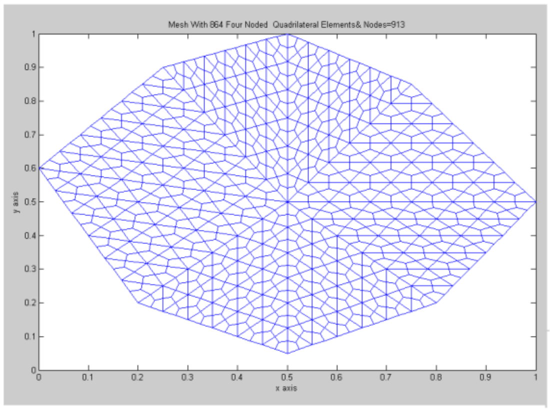
MESH NO.4



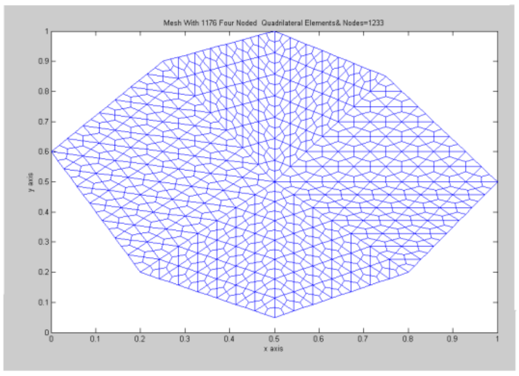
MESH NO.5



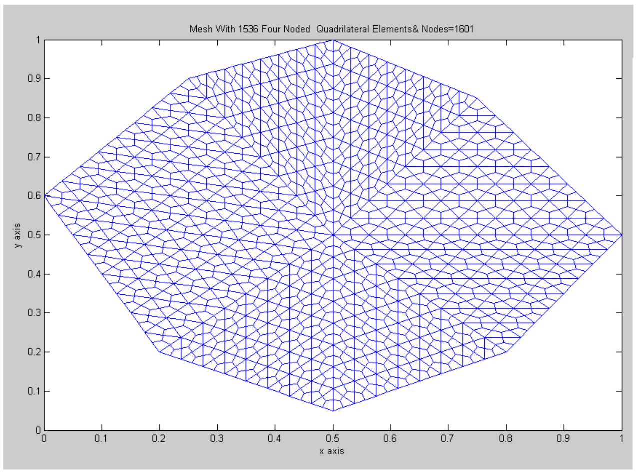
MESH NO.6



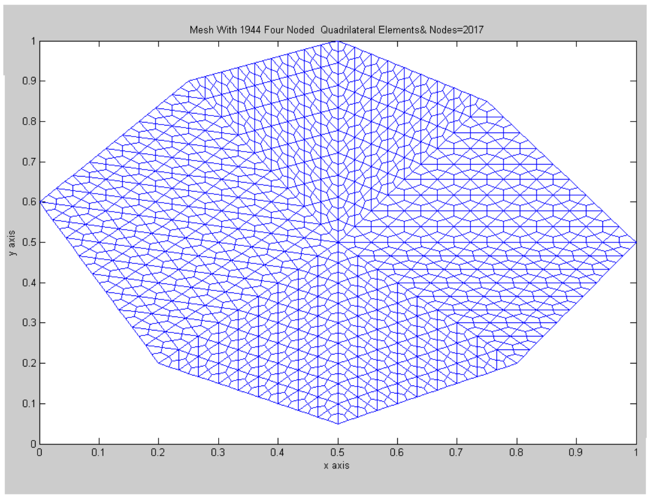
MESH NO.7


Mesh With 1176 Four Noded Quadrilateral Elements& Nodes=1233
y axis
x axis
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

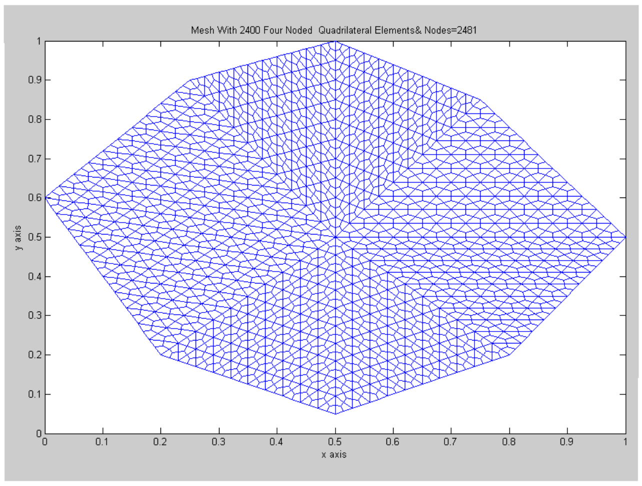
MESH NO.8


Mesh With 1536 Four Noded Quadrilateral Elements& Nodes=1601
y axis
x axis
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

MESH NO. 9


Mesh With 1944 Four Noded Quadrilateral Elements& Nodes=2017
y axis
x axis
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

MESH NO.10


Mesh With 2400 Four Noded Quadrilateral Elements& Nodes=2481
y axis
x axis
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1