ALGORITHM FOR ADAPTIVE GRID GENERATION USING GALERKIN FINITE ELEMENT METHOD

by

MONALKUMAR PATEL

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN AEROSPACE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

ACKNOWLEDGEMENTS

I would like to express deepest gratitude for my supervising professor Dr. Brian Dennis for his incredible support, patience and consistent guidance throughout my research work. I would also like thank Dr. Guojun Liao and Dr. Mehmet Ali Akinlar for their invaluable support and guidance to understand the basic concepts of grid deformation method and monitor function utilized in my work. I also want to thank Dr. Kent Lawrence and Dr. Seiichi Nomura for being part of my supervising committee members.

I would like to recognize my CFD lab colleague and friend Harsh Shah for his valuable support during my research work. He has provided me with really valuable suggestions and insightful thoughts with my work. I would like to recognize Darshak Joshi and Kamal Chauhan not only as CFD lab colleagues, but also for their valuable friendship and continuous support throughout my work. I would also like to recognize my other CFD lab colleagues Dr. Rajeev Kumar, Wei Han (Katie), Deval Pandya and Travis for their support and friendship.

I am gratefully obliged to the SOAR-University Tutorial Department at UT Arlington for providing me with a Graduate Assistantship to support my education. My sincere and personal thanks to Robin Melton, Vivian Pham, Amy Fortlage, Beena and all my colleagues at SOAR for their moral support and encouragement through this journey.

Last but not least, I would like to acknowledge my parents Harikrishna Patel and Kailasben Patel for their endless love, moral support, hard work and blessings. This work would not have existed without their belief in me. A special word of appreciation goes to my younger brother Krutagn for his encouragement and love.

November 24, 2009

ABSTRACT

ALGORITHM FOR ADAPTIVE GRID GENERATION USING

GALERKIN FINITE ELEMENT METHOD

Monalkumar Patel, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Brian Dennis

The adaptive grid concept is developed in order to achieve more accurate and stable results for the numerical simulations of Partial Differential Equations (PDEs). There are two main types of strategies used for adaptive grids: local refinement by increasing the number of elements (*h*-refinement) and deforming grids where the number of elements remains fixed (*r*-refinement) [2]. The *h*-refinement method requires inserting additional elements and nodes in a certain region of the mesh. This significantly affects the software data structure and makes programming more difficult. To maintain the same data structure, the method must delete the same number of nodes from another region. Again, this complicates the program. However, the deforming grid method works by simply moving the nodes of an existing mesh. The proposed method formulates this deformation as an unsteady problem where the position of the nodes can be determined from their velocities. For each time step, the node movement is controlled by a user defined error indicator or user desired target node distribution. The number of nodes and elements remains constant for each time step and this greatly simplifies the program structure compared to the *r*-method. During the deformation process, the element shape changes to achieve the desired distribution and solution accuracy at each time step. The current research

work focuses on the development of the algorithm for unstructured meshes and its application to the solution of unsteady elliptic PDEs solved by finite element (unstructured). The successful development of the Adaptive Grid Generation is achieved with the implementation of Galerkin Finite Element Method on Grid Deformation Method for unstructured and the results are validated with the structured grids results. It is also implemented on few model fluid problems.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

In field of Computational Fluid Dynamics (CFD) and Finite Element Analysis (FEA), physical problems can be solved by modeling it to the system of the Partial Differential Equations (PDEs). The solution of these PDEs can be found either analytically or numerically. It is very challenging to find the analytical solution for system of PDEs governing the physical phenomenon, such as, heat transfer problems, shock wave problems and boundary layer problems; therefore numerical solution of the PDEs is encouraged and the available computational technology makes it much faster to obtain.

The numerical solution of the PDE is dependent on grid generation for the given domain. The inappropriate grid generation can result in the unstable and inaccurate solution of the PDE. There are two types of grids available: structured grid and unstructured grid. In structured grid the nodes are distributed in structured fashion and each node is shared by the same number of the elements for the entire domain. The unstructured grid has random node distribution over the domain and each node is not shared by the same number of elements or we can say it say irregular element connectivity. The structured grids are useful for simple geometries and problems with no sudden variations during the solution. The unstructured grids are better choice than the structured grids as it provides more accurate and stable numerical solutions for complex geometries although they require complex data structure than structure grids. The structured or unstructured grids remain fixed (static) and the number of nodes is not changed throughout the computation. These static grids are not able to acquire accurate and stable results for the PDEs having sudden solution variations, such as boundary layer formation, flow separation or shock waves. To achieve more stable and accurate results for

such numerical simulations of the partial differential equations (PDEs), the adaptive grid concept has been used in the last three decades[3].

There are main two types of strategies used to achieve adaptive grids: local refinement of the grid and deformation of the grid (also known as *moving grid*) [2]. The unique goal of the both strategies is to generate the fine grid over the region where the partial differential equations (PDEs) show the large solution variations and, generate coarse grid where solution is comparatively stable. Local refinement of the grid is achieved by inserting additional nodes and/or elements in certain region(s) of the grid where we need fine grid, and removing the same number of the nodes and/or elements from the region(s) where we need coarse grid. The deformation of grid achieves same refinement as local refinement, but the only difference is, it moves nodes closer to the certain region(s) where fine grid is desired, and nodes moves away from each other to the region(s) where solution doesn't impact much. The deformation of grid works without changing number of nodes or element connectivity over the domain.

There are two main local refinement methods developed over the last couple of decades: *h-refinement method* and *p-refinement method [2]*. The *h-refinement* method works on insertion and elimination of the nodes, which significantly affects the data structure and makes it complex for the programming. It also requires higher storage memory and longer computational time. The *p-refinement* method works by changing the order of the basic polynomial function as per requirement during the computational time. These local refinement methods showed popularity when they are used with the Finite Element Method (FEM) to solve the PDEs governed by physical problems.

A Moving Grid Finite Difference Method [3] is one of the techniques available for grid adaptation which works on the *r-refinement* strategy. The moving grid finite difference method is also known as the Grid Deformation Method developed by G. Liao and D. Anderson [4], which works on the idea of the Moser's deformation method [3]. This method was improved by G. Liao, T. Pan and J. Su in [6]. The Grid Deformation Method formulates the deformation as an

unsteady problem where the position of the nodes can be determined from their velocities. For each time step, the node movement is controlled by a user defined error indicator or user desired target node distribution. This user define error indicator is known as “monitor function” in this method. The number of nodes and elements remains constant for each time step and this greatly simplifies the program structure compared to local refinement methods. It also requires less computational time as compared to the local refinement methods.

In the present work, Grid Deformation Method is utilized. The grid deformation method deforms the grid at certain desired region due to movement of the nodes. This nodal movement happens due to the vector field generated by the solution of div-curl system of the PDE. The present work focuses on application of this Grid Deformation Method to the unstructured grids and solution of unsteady elliptic PDEs using Finite Element Method (FEM).

The roadmap of this thesis is, chapter 1 reviewed over the adaptive grids and the techniques available to achieve these grid deformation. The chapter 2 describes about the adaptive grid generation by Grid Deformation Method and the construction of the monitor function for appropriate grid refinement. The chapter 3 talks about the Galerkin Finite Element Method and the numerical implementation of it in the Grid Deformation Method. The chapter 4 describes the validation of the code and some more numerical examples with the use of this grid adaptation technique. Finally, chapter 5 outlines the summary of the work and future recommendations.

CHAPTER 2

ADAPTIVE GRID BY GRID DEFORMATION METHOD

2.1 Introduction

As we have discussed in the previous chapter the Grid Deformation Method developed by G. Liao and D. A. Anderson [4] is one of the moving grid methods available for the grid refinement. This method works on the movement of the nodes to certain desired region by providing them appropriate velocity and the element connectivity remains constant throughout the computation. This method was used for steady Euler flow calculations as well [7].

There are three versions available for this Grid Deformation Method and they are described with details in [8,9]. The "monitor function" is important parameter to achieve proper grid refinement in this method. It can be constructed on the gradient of solution or error indication from the solution. We will talk about the monitor function in later part of this chapter.

2.2 Grid Deformation Method

2.2.1 Grid Deformation Method Version 1

This is the very preliminary version of the grid deformation method, which has steady features in it. This method works in following steps as outlined in [8]. This method applies the adaptation on the old grid.

- The first step in this method is to construct the monitor function f (see section 2.3 of this chapter for the details of monitor function).
- This monitor function is normalized in this method such as it satisfies the following condition over the domain Ω.

$$\int_{\Omega}(f-1)d\Omega = 0, \tag{2.1}$$

Where, f is the normalized monitor function for the domain Ω. For 2D case Ω would be the area and in 3D case it would be volume of the domain.

- Now, to achieve the grid refinement we have to find the transformation function $\vec{\phi}$ such that,

$$J(\vec{\phi}) = f \tag{2.2}$$

Where, $J(\vec{\phi})$ is Jacobian determinant of the transformation function $\vec{\phi}$

- To find this transformation function $\vec{\phi}$, this method uses the div-curl equation system to find the velocity field $\vec{u}(\vec{\mathrm{x}})$ as following.

$$\nabla \cdot \vec{u} = f(\vec{\mathrm{x}}) - 1 \quad \text{in } \Omega \tag{2.3}$$

$$\vec{u} \cdot \hat{n} = 0 \qquad \text{on} \quad \Gamma \tag{2.4}$$

- From the solution of equation (2.3) using the boundary condition as equation (2.4) we can have the velocity field $\vec{u}$. Now, this velocity field can be used to find

$$\vec{h}(t,\vec{\mathrm{x}}) = \frac{\vec{u}}{t + (1-t)f(\vec{\mathrm{x}})} \qquad 0 \le \mathrm{t} \le 1 \tag{2.5}$$

- The equation (2.5) is used to form the deformation ordinary differential equation (ODE) as follows and which can be solved using the initial condition as $\vec{\phi}_k(t,\vec{\mathrm{x}}) = \vec{\phi}(t_{k-1},\vec{\mathrm{x}})$

$$\frac{d\vec{\phi}}{dt} = \vec{h}(t,\vec{\phi}(t,\vec{\mathrm{x}})) \qquad 0 \le \mathrm{t} \le 1 \tag{2.6}$$

It has been proved by J. Liu [9] with mathematical theorem, that the transformation function found from the equation (2.6) meets the criteria defined in the equation (2.2)

2.2.2 Grid Deformation Method Version 2

This is also preliminary version of the grid deformation method, which has steady features in it. In this method the Jacobian determinant is specified to the new grid coordinates before grid refinement occurs. This version of the method has identical first steps as we have discussed in the previous version of this method. The following steps show the remaining procedure to achieve desired refinement of the grid using this version and it is outlined in details in [8].

- This monitor function is normalized in this method such as it satisfies the following condition over the domain Ω.

$$\int_{\Omega}\left(\frac{1}{f}-1\right)d\Omega = 0 \qquad \text{or} \qquad \int_{\Omega}\frac{1}{f} = |\Omega| \tag{2.7}$$

  Where, f is the normalized monitor function.

- In this version also we have to find the transformation function $\vec{\phi}$ such that it will satisfy the equation (2.2).
- Now, this version uses the following div-curl equation system to find the velocity field $\vec{u}(\vec{x})$,

$$\nabla \cdot \vec{u} = 1 - \frac{1}{f(\vec{\mathrm{x}})} \qquad \text{on} \quad \Omega \tag{2.8}$$

$$\vec{u} \cdot \hat{n} = 0 \qquad \text{on} \quad \Gamma \tag{2.9}$$

- Compute the velocity field $\vec{u}(\vec{\mathrm{x}})$ by solving equation (2.8) and (2.9). Now, this velocity field can be used to find

$$\vec{h}(t,\vec{\phi}(t,\vec{\mathrm{x}})) = \frac{\vec{u}(\vec{\mathrm{x}})}{t\dfrac{1}{f(\vec{\phi}(t,\vec{\mathrm{x}}))} + (1-t)} \qquad 0 \le \mathrm{t} \le 1, \tag{2.10}$$

- The equation (2.10) is used to form the deformation ordinary differential equation (ODE) as follows and which can be solved using the initial condition as $\vec{\phi}_k(t,\vec{\mathrm{x}}) = \vec{\phi}(t_{k-1},\vec{\mathrm{x}})$, as described in the version 1 of the grid deformation method.

2.2.3 Grid Deformation Method Version 3

This is most recent version of the grid deformation method, which is real time grid adaptation version. This version of the method also uses the monitor function like its previous two versions. This version of the method uses the same normalization conditions for the monitor function as used in the second version of this method. This method has difference in the div-curl system and solution of it. This method is used for the present work with little modification in the solution technique using Galerkin Finite Element Method. The following steps show that how the div-curl system has been formed and solved for the third version of the grid deformation method.

- As we have previously discussed in first version and second version, the monitor function is formed with the use of the gradient of solution or error of the solution and it is normalized in such a way that it would satisfy the condition described in equation (2.7). Here, the monitor function is $f(t,\vec{\mathrm{x}})$, where $\vec{\mathrm{x}}$ represents the vector for the nodal coordinates. Thus, the monitor function in this version is function of space (nodal displacement) and time.
- To find the transformation function $\vec{\phi}$, we use the following div-curl system of the equation to evaluate the velocity field $\vec{u}(\vec{\mathrm{x}})$,

$$\nabla \cdot \vec{u} = -\frac{\partial}{\partial t}\left(\frac{1}{f}\right) \qquad \text{on } \Omega \tag{2.11}$$

$$\vec{u} \cdot \hat{n} = 0 \quad \text{on } \Gamma \qquad \text{or} \qquad (\nabla \times \vec{u} = 0) \tag{2.12}$$

To solve this div-curl system, it has been assumed that $\vec{u} = \nabla\omega$. So, if we substitute this assumption into the equation (2.11), it becomes

$$\nabla \cdot (\nabla \omega) = \nabla^2 \omega = \Delta \omega = -\frac{\partial}{\partial t}\left(\frac{1}{f}\right) \qquad \text{in} \quad \Omega \tag{2.13}$$

The equation (2.12) becomes, $\nabla \omega \cdot \hat{n} = 0$ on Γ (2.14)

The equation (2.13) can be rewritten as the $\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} = -\frac{\partial}{\partial t}\left(\frac{1}{f}\right)$ in Ω, which is the scalar Poisson equation if we consider the right hand side term as the source term in it. Now, in this version, this Poisson equation is solved using the iterative finite difference solution methods such as Successive Over Relaxation method (SOR) with appropriate relaxation factor in [10].

- The value of ω is obtained by solving the above scalar Poisson equation. Now, this ω is used to calculate the velocity field by substituting the value of ω back in the assumption $\vec{u} = \nabla\omega$. Thus, $u = \frac{\partial \omega}{\partial x}$ & $v = \frac{\partial \omega}{\partial y}$. Where, u is x-component of the velocity field and v is y-component of the velocity field.
- Here, the ODE would be formed as follows,

  $$\frac{d\vec{\phi}(t, \vec{\mathrm{x}})}{dt} = f(t, \vec{\phi}(t, \vec{\mathrm{x}}))\, \vec{u}(\vec{\mathrm{x}}) \tag{2.15}$$

  or it can be rewritten as,

  $\frac{dx}{dt} = fu$ & $\frac{dy}{dt} = fv$ in terms of the nodal coordinates for 2D domain.
- The last step is the same for this version, as like previous two versions to solve the ordinary differential equation with the initial condition same as described in the first version.

The third version of grid deformation method is used in our work and the all three versions of this grid deformation method repeats the steps described above until the final adapted grid is generated.

2.3 The Monitor Function

We have talked about Grid Deformation Method in the section 2.2, the monitor function is very important parameter for appropriate grid adaptation as it controls the movements of nodes throughout computation. So, care must be taken while we choose the monitor function. There are several ways to form the monitor function such as from the gradient approximation of solution during computation, error calculations. D. A. Anderson used equidistribution principle [11] to form this monitor function. G. Liao and De la Pena used the equidistribution principle over the domain to construct this monitor function (See [10] for more details). As it is outlined in [10], if we have some positive error estimator or gradient approximation of the property that could be responsible for the rapid change in the solution or large variation during intermediate computation time step $\delta(x, y, t)$, the monitor function f can be constructed as

$$f(x, y, t) = \frac{C}{\delta(x, y, t)} \tag{2.16}$$

where, C is the normalization factor such that it satisfies the following relation for each time step during the computation (as we consider the third version of grid deformation method),

$$\int_{\Omega}\left(\frac{1}{f(x, y, t)} - 1\right) d\Omega = 0 \text{ or } \int_{\Omega}\frac{1}{f(x, y, t)} = |\Omega| \tag{2.17}$$

where, $|\Omega|$ is the volume of the domain (area in 2D). The monitor function f has some distinct properties like it is small in the region where gradient or error in solution is large and it is large in the region where gradient or error in solution is small. This property of the monitor function leads the nodes pulled towards the region with large error or gradient of solution and pulled

away from the region with small error or gradient of solution. Thus, it starts adapting the grid in the desired region where the solution changes rapidly. There is theorem presented by De la Pena in [10], proves that this grid deformation with finite difference method ensures that the grid will not fold onto itself throughout computation and adaptation of the grid.

As we have talked in previous section, monitor functions are constructed from the use of equidistribution principles. The equidistribution principle works on the assumption that residuals, truncation errors and posteriori error estimates (if available during computation) are equally distributed all over the domain and they are equally weighted [11]. The following are some examples of monitor functions which can be constructed from the solution during computation (i.e. gradient of solution).

If we want to construct the monitor function that can be used for the heat problems,

$$f = \frac{C1}{1 + C2\left|\nabla T\right|} \tag{2.18}$$

Where, $C1$ is the normalization factor and can be calculated from the normalization of the monitor function and $C2$ is the constant for adaptation intensity and the gradient of the temperature is $\left|\nabla T\right|$.

If the flow that can produce the shock wave during the solution the monitor function can be constructed as,

$$f = \frac{C1}{1 + C2\left|\nabla P\right|^2} \quad \text{or} \quad f = \frac{C1}{1 + C2\left|\nabla M\right|} \tag{2.19}$$

Where, P is the pressure and M is the Mach number for the given flow conditions. $C1$ and $C2$ are the normalization factor and grid adaptation intensity constant respectively. We can construct the appropriate monitor function using the parameter responsible for the rapid change in solution during the computation using the equidistribution principles like above.

The following example of the monitor function is constructed using some variable of interest from solution and to make it better the gradient of that variable as well as the second derivatives of the variable are also included. For example, equation (2.20) shows the monitor function constructed for some unknown variable of interest v, such as in [10],

$$f = \frac{\kappa}{1 + \alpha|v|^2 + \beta|\nabla v|^2 + \gamma|\nabla\nabla v|^2} \tag{2.20}$$

Where, α, β and γ are the grid adaptation intensity constants for $|v|$, $|\nabla v|$ and $|\nabla\nabla v|$ respectively.

The monitor function f can also be constructed for the interface grid adaptation problems or known shape geometry problems using the signed distance function *d* which can be find using the level set deformation method as described in [10].

CHAPTER 3

THE FINITE ELEMENT METHOD AND ITS IMPLETMENTATION TO ADAPTIVE GRID GENERATION

The Finite Element Method is popular to solve fluid dynamics problems lately because of its consistency to form mathematical formation and computer programming over the finite difference and finite volume method. It is also capable to handle the complex geometry problems which are sometimes difficult to handle with the finite difference schemes.

3.1 Overview: Finite Element Method

The finite element method is based on classical variational approach (i.e. Ritz method) and weighted-residual method to solve the differential equation [12]. The general idea behind these methods is to solve the differential equation in two steps. First we form the governing equation into equivalent weighted-integral form and later on approximate the solution for the whole domain with having an assumption that solution would be linear combination of the assumed approximate functions (N_i) and undetermined coefficients (c_i). These undetermined coefficients c_i are calculated such that they satisfy the differential equation in weighted-integral sense. The appropriate approximate functions N_i should be chosen to satisfy the given boundary conditions for the problem. This traditional variational weighted-residual approach has drawback of forming the approximation function that satisfy boundary conditions for real-world problems with complex geometries and different boundary conditions for different regions of the whole domain. Thus, the finite element method is originated with idea of dividing the whole domain into finite number of “sub-domains” which are simple geometrical shapes like triangles or quadrilaterals also known as *finite elements*.

It is comparatively easy to generate the systematic approximation functions N_i, which satisfy boundary conditions for the finite elements instead of whole domain by use of either variational method or weighted-residual method. These approximate functions for the finite elements are generally formed using interpolation theory or the geometrical shape of finite elements so they are also known as the "interpolation functions" or "shape functions".

The following steps show general procedure to solve the problem using finite element method as described in [12, 13, 14].

- Discretize the domain into set of finite elements (grid generation).
- Weighted-integral or weak formulation of the differential equation.
- Develop the finite element model for the physical problem using this weak form or weak statement.
- Assemble all the finite element equations to obtain the global system of algebraic equations.
- Apply appropriate boundary conditions.
- Solve this global system of algebraic equations (use direct solution method or iterative solution method).
- Visualization of the results and post processing for the important parameters of the physical problem.

3.1.1 Method of Weighted Residuals

As we have seen in the section 3.1, the Method of Weighted Residuals (also known as MWR) works in two steps. The first step is to find the approximation function that satisfies boundary condition for the finite elements which is equivalent to the physical boundary conditions. When we substitute this approximate solution into differential equation it may result in some residuals. This residual needs to be minimized or vanished during the solution process over the domain. The second step for this method is to solve the equations in such a way that the residual from the first step would almost vanish or becomes zero. The following table shows

the list of the available methods with the choice of the interpolation functions N_i and weighting functions w_i.

Table 3.1 The Methods produced by choices of the Weighting function [15]

Weighting Function, $w_i(x)$	***Method***
$w_i(x) = \delta(x - x_i)$	Collocation
$w_i(x) = \begin{cases} 1 & \text{inside } \Omega^i \\ 0 & \text{inside } \Omega^i \end{cases}$	Finite Volume (Subdomain)
$w_i(x) = \dfrac{\partial R}{\partial \hat{u}_i}$	Least-Square
$w_i(x) = x^i$	Method of Moments
$w_i(x) = N_i(x)$	Galerkin
$w_i(x) = \Psi_i(x)$	Petrov-Galerkin

The MWR methods most probably used to formulate the Galerkin Finite Element Method as the weighting function is chosen as same as the interpolation function or approximation function (i.e. $w_i(x) = N_i(x)$).

3.2 The Galerkin Finite Element Method

We have seen in the MWR methods (table 3.1), the Galerkin finite element method works on the basis that weighting functions for this method is same as the interpolation functions i.e. $w_i(x) = N_i(x)$. The approximate solution is expressed in form of the linear combination of undetermined coefficients and these approximation functions $N_i(x)$. Now, to

illustrate this method in detail let us take an example of the 1D Poisson Equation as outlined in [13] with the boundary conditions as follows,

$$\frac{d^2\phi}{dx^2} = -f(x) \tag{3.1}$$

with the boundary conditions as, $\phi(x_1) = A1$, $\phi(x_2) = A2$. Where, x_1 and x_2 are the boundary points. To find the solution using this Galerkin FEM method we follow the same procedure as we described in the section 3.1. Now, to start with solution first we approximate the analytical solution $\phi(x)$ by $\tilde{\phi}(x)$, which could be represented in the form of interpolation functions as follows.

$$\tilde{\phi} \approx \sum_{i=1}^{n} N_i(x)\phi_i \tag{3.2}$$

where, n is the number of nodes per element. As we move to the next step for the solution procedure, apply the weighting function to the equation (3.2) and form its integral form as follows,

$$\int_{x_1}^{x_2} N_i(x)\left[\frac{d^2\tilde{\phi}}{dx^2} + f(x)\right]dx = 0 \tag{3.3}$$

where, x_1 and x_2 are the starting and end points (boundary points) of the linear element respectively. Now, by using the integration by parts for the equation (3.3) we have,

$$N_i \frac{d\tilde{\phi}}{dx}\bigg|_{x_1}^{x_2} - \int_{x_1}^{x_2} \frac{d\tilde{\phi}}{dx}\frac{dN_i}{dx}dx + \int_{x_1}^{x_2} f(x)N_i(x)dx = 0 \tag{3.4}$$

The equation (3.4) is known as the weak form of the equation (3.3). Now, let us use the equation (3.2) into the equation (3.4) we will have,

$$\frac{d\tilde{\phi}}{dx} = \sum_{i=1}^{n} \frac{dN_i}{dx}\phi_i = \left[\frac{dN_i}{dx}\right]\{\phi\}^{(e)} \tag{3.5}$$

where, $\{\phi\}^{(e)}$ is the column vector for the elemental nodal unknowns. Now, we know that ϕ must have C^1 continuity in the original boundary value problem as must $\widetilde{\phi}$ in the integral form of the Galerkin method showed in equation (3.3). In the weak form requirement the continuity is reduced to C^0 from C^1 for the $\widetilde{\phi}$. Now substitute the equation (3.5) into the weak form of the differential equation (i.e. equation (3.4)) and with rearrangement of some terms,

$$\int_{x_1}^{x_2}\left[\frac{dN_i}{dx}\right]\frac{dN_i}{dx}dx\{\phi\}^{(e)} = N_i\frac{d\widetilde{\phi}}{dx}\Bigg|_{x_1}^{x_2} + \int_{x_1}^{x_2} f(x)N_i(x)dx \quad , \; i=1,2 \tag{3.6}$$

As it is linear element, it has only two nodes. The first term on the right-hand side of the equation (3.6) represents the natural boundary conditions. It is applied to global level after the assembly of the global matrix system for the domain. So, at the element level the system of equation is

$$[Ke]^{(e)}\{\phi\}^{(e)} = \{Fe\}^{(e)} \tag{3.7}$$

In equation (3.7),

$$[Ke]^{(e)} = \int_{x_1}^{x_2}\left[\frac{dN_i}{dx}\right]\frac{dN_i}{dx}dx$$

$$\{Fe\}^{(e)} = \int_{x_1}^{x_2} f(x)N_i(x)dx$$

where, $[Ke]^{(e)}$ and $\{Fe\}^{(e)}$ are elemental stiffness matrix and elemental force vector respectively. Here, the elements are linear for 1D so, there are only 2 nodes for each element. So, the elemental stiffness matrix and elemental force vector would be

$$[Ke]^{(e)} = \int_{x_1}^{x_2}\begin{bmatrix}\frac{dN_1}{dx}\frac{dN_1}{dx} & \frac{dN_1}{dx}\frac{dN_2}{dx}\\ \frac{dN_2}{dx}\frac{dN_1}{dx} & \frac{dN_2}{dx}\frac{dN_2}{dx}\end{bmatrix}dx \quad \text{and} \quad \{Fe\}^{(e)} = \int_{x_1}^{x_2}\begin{Bmatrix} fN_1 \\ fN_2\end{Bmatrix}dx \tag{3.8}$$

These elemental stiffness matrix and elemental force vectors are used to form the global stiffness matrix and global force vector by assembling it for all elements of the domain. The boundary condition term in the equation (3.6) would cancel at all interior nodes of the domain during global assembly of the matrices. This global matrix system represents global system of algebraic equation and afterwards it can be solved by direct inversion of the matrix with application of the appropriate boundary condition. We will see the numerical implementation of this method in the following section for adaptive grid generation.

3.3 Numerical Implementation of Galerkin FEM to Grid Deformation Method

In the present work, the algorithm to generate adaptive grid is formed on the basis of grid deformation method third version as described in section 2.2.3 of chapter 2. This section describes the mathematical implementation of Galerkin Finite Element Method to generate adaptive grids with the use of grid deformation method. The following subsections describe systematic way to achieve new grid coordinates with application of Galerkin Finite Element Method for grid deformation method.

3.3.1 The div-curl System

The original div-curl system is given as follows from the third version of grid deformation method,

$$\nabla \cdot \vec{u} = -\frac{\partial}{\partial t}\left(\frac{1}{f}\right) \quad \text{in } \Omega \tag{3.9}$$

$$\nabla \times \vec{u} = 0 \tag{3.10}$$

This system of equation can be rewritten as follows for 2D case. Let's take u and v as the x-component and y-component of the velocity field $\vec{u}$.

Thus, $\vec{u} = u\hat{i} + v\hat{j}$

The equation (3.9) becomes, $\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=-\frac{\partial}{\partial t}\left(\frac{1}{f}\right)$ (3.11)

The equation (3.10) can be broken down to $\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}=0$ (3.12)

The boundary condition is $\vec{u}\cdot\hat{n}=0$ (3.13)

In this algorithm we solved the div-curl system represented in equation (3.11) and equation (3.12) with reformation as following,

Let us say $\Re=-\frac{\partial}{\partial t}\left(\frac{1}{f}\right)$ (3.14)

Now, substitute the equation (3.14) into the equation (3.11) and differentiate it with respect to x

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial x\partial y}=\frac{\partial \Re}{\partial x} \tag{3.15}$$

Let, differentiate equation (3.12) with respect to y , we have

$$\frac{\partial^2 v}{\partial y\partial x}-\frac{\partial^2 u}{\partial y^2}=0 \tag{3.16}$$

Now subtracting the equation (3.16) from equation (3.15) we have,

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=\frac{\partial \Re}{\partial x} \tag{3.17}$$

which is similar form of the scalar Poisson Equation for u with the right hand side is space dependent. Similarly, the Poisson Equation for v can be obtained by differentiating the equation (3.11) with respect to y and then adding the differentiation of the equation (3.12) with respect to x ,

$$\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}=\frac{\partial \Re}{\partial y} \tag{3.18}$$

The equation (3.17) and equation (3.18) are same as the div-curl system of equations defined in the equation (3.9) and equation (3.10) with boundary conditions as $\vec{u} \cdot \hat{n} = 0$. By solving equation (3.17) and equation (3.18), we can directly have velocity field for the given domain. To solve these two Poisson Equations in 2D, Galerkin FEM method is used. The following section describes the Galerkin FEM method formulation for these two equations.

3.3.2 Galerkin FEM formulation to solve 2D Poisson Equation

This section of chapter talks about the solution of 2D Poisson equation with the use of Galerkin Finite Element Method. The equation (3.17) and equation (3.18) are almost identical and having same Dirichlet boundary conditions. The Neumann boundary conditions are applied to different boundaries for x-component of velocity and y-component of velocity. Thus, we will take one of these two equations for demonstration and repeat same procedure for the other one. Let's consider the equation (3.17) ,

$$\nabla^2 u = \frac{\partial \Re}{\partial x}$$

Now, apply Galerkin method to the above 2D Poisson equation,

$$\int_{\Omega} w\left(\nabla^2 u - \frac{\partial \Re}{\partial x} \right) = 0 \tag{3.19}$$

Now, with the use of integration by parts,

$$\int_{\Gamma} \hat{n} \cdot (w \nabla u) - \int_{\Omega} \nabla w \cdot \nabla u d\Omega = \int_{\Omega} w \frac{\partial \Re}{\partial x} d\Omega \tag{3.20}$$

Now Galerkin FEM method uses same weight function as the approximation function, i.e. we can have $w = \sum_{i=1}^{n} u_i N_i$, $u = \sum_{i=1}^{n} u_i N_i$ and $\Re = \sum_{i=1}^{n} \Re_i N_i$ (n is the number of nodes in element).

The final variational form can be written as after the Neumann boundary condition (the velocity in normal direction is zero at x-direction boundaries for u or $\vec{u} \cdot \hat{n} = 0$) applied to the equation (3.20),

$$-\int_{\Omega}\nabla w\cdot\nabla u d\Omega=\int_{\Omega}w\frac{\partial\Re}{\partial x}d\Omega \tag{3.21}$$

The equation (3.21) can be rewritten as follows in matrix form,

$$-[K]\{u\}^{(e)}=[CX]\{\Re\}^{(e)} \tag{3.22}$$

Where, $[K]=\int_{\Omega}[\nabla N][\nabla N]^{T}d\Omega$ (3.23a)

$$[CX]=\int_{\Omega}[N]\left[\frac{\partial N}{\partial x}\right]^{T}d\Omega \tag{3.23b}$$

The formulation of the equation (3.18) would be exactly same as the formulation described above. The following equations shows the final weak form and matrix form for equation (3.18)

$$-\int_{\Omega}\nabla w\cdot\nabla v d\Omega=\int_{\Omega}w\frac{\partial\Re}{\partial y}d\Omega \tag{3.24}$$

The equation (3.24) can be rewritten as following in matrix form,

$$-[K]\{v\}^{(e)}=[CY]\{\Re\}^{(e)} \tag{3.25}$$

Where, $[K]=\int_{\Omega}[\nabla N][\nabla N]^{T}d\Omega$ (3.26a)

$$[CY]=\int_{\Omega}[N]\left[\frac{\partial N}{\partial y}\right]^{T}d\Omega \tag{3.26b}$$

In equation (3.22) and equation (3.25), the value of $\Re$ can be found explicitly from its assumption made in the equation (3.14). The Appendix B shows the normalization of monitor function for linear triangular element. Here, Euler forward difference scheme is used in time to find value of $\Re$, using the Normalized monitor function f, as following formula,

$$\Re=-\left(\frac{\frac{1}{f^{n+1}}-\frac{1}{f^{n}}}{\Delta t}\right) \tag{3.27}$$

We need to note that the value of $\Re$ is negative in the equation (3.27) and which will cancel out the negative sign on the left hand side of the equation (3.21). The equation (3.23a) and equation (3.26a) are known as the elemental stiffness matrices and they would be used to construct global stiffness matrices by assembling all other elemental stiffness matrices. The equation (3.22) and equation (3.25) are solved in MATLAB with the use of invert matrix method with the application of appropriate Dirichlet boundary condition. See the Appendix A for details of shape functions, elemental stiffness matrices and elemental force vectors used for linear triangular elements.

3.3.3 Boundary Condition Treatment

To solve this Poisson equation we need to apply boundary conditions. There are main two types of boundary conditions needed to make it well posed problem: Neumann (Natural) Boundary condition and Dirichlet Boundary Condition. As we have seen in the previous section, the Natural boundary condition is automatically applied during the Galerkin finite element formulations. So, now we need to apply the Dirichlet boundary conditions in order to solve system of algebraic equations.

In our case the problem is posed such that all the boundary nodes remain stick on the boundary throughout the computation. Thus, the boundary nodes are allowed to have tangential velocity but not the normal velocity. The normal velocity of the boundary nodes are fixed as zero by applying the Neumann boundary conditions. The corner nodes are on both x-direction and y-direction boundaries, so they need to be fixed in both directions (i.e. tangential and normal velocity for them are zero). Thus, when we solve for the global system of algebraic equations we can treat this requirement as our Dirichlet boundary condition to solve the problem. We are solving the matrix form obtained in equation (3.22) and equation (3.25) by enforcing the boundary condition as all the corner nodes are fixed.

3.4 Solution of Deformation ODE

The above section describes that how to implement Galerkin finite element method to solve 2D Poisson equation formed with the use of the div-curl system. So, at the end of above step we already found the velocity components (i.e. u and v) for each node. To find new locations of nodes we need to form the ordinary differential equation (ODE). Here, we can form directly two ODEs as follows,

$$\frac{dx}{dt} = fu \tag{3.28a}$$

$$\frac{dy}{dt} = fv \tag{3.28b}$$

The equations (3.28a) and (3.28b) can be solved using Euler's first order method or Runge-Kutta method (either 2nd order or 4th order as per accuracy requirement). Here, we used Euler's first order method to solve these ODEs. The formulation is as follows,

$$x_i^{n+1} = x_i^n + dt * fu$$

$$y_i^{n+1} = y_i^n + dt * fv$$

Where, x_i^{n+1} and y_i^{n+1} represent the new node coordinates for i^{th} node at $(n+1)^{\text{th}}$ time step and similarly x_i^n and y_i^n are the coordinates of node at $(n)^{\text{th}}$ time step.

The procedure described in this chapter can be represented as the following flow chart and it could be repeated for the sufficient number of time steps to achieve the desired refinement (adaptation) of the grid. This procedure can be terminated when desired grid refinement achieved or the time can be defined in proportion of actual governing equation's time step.

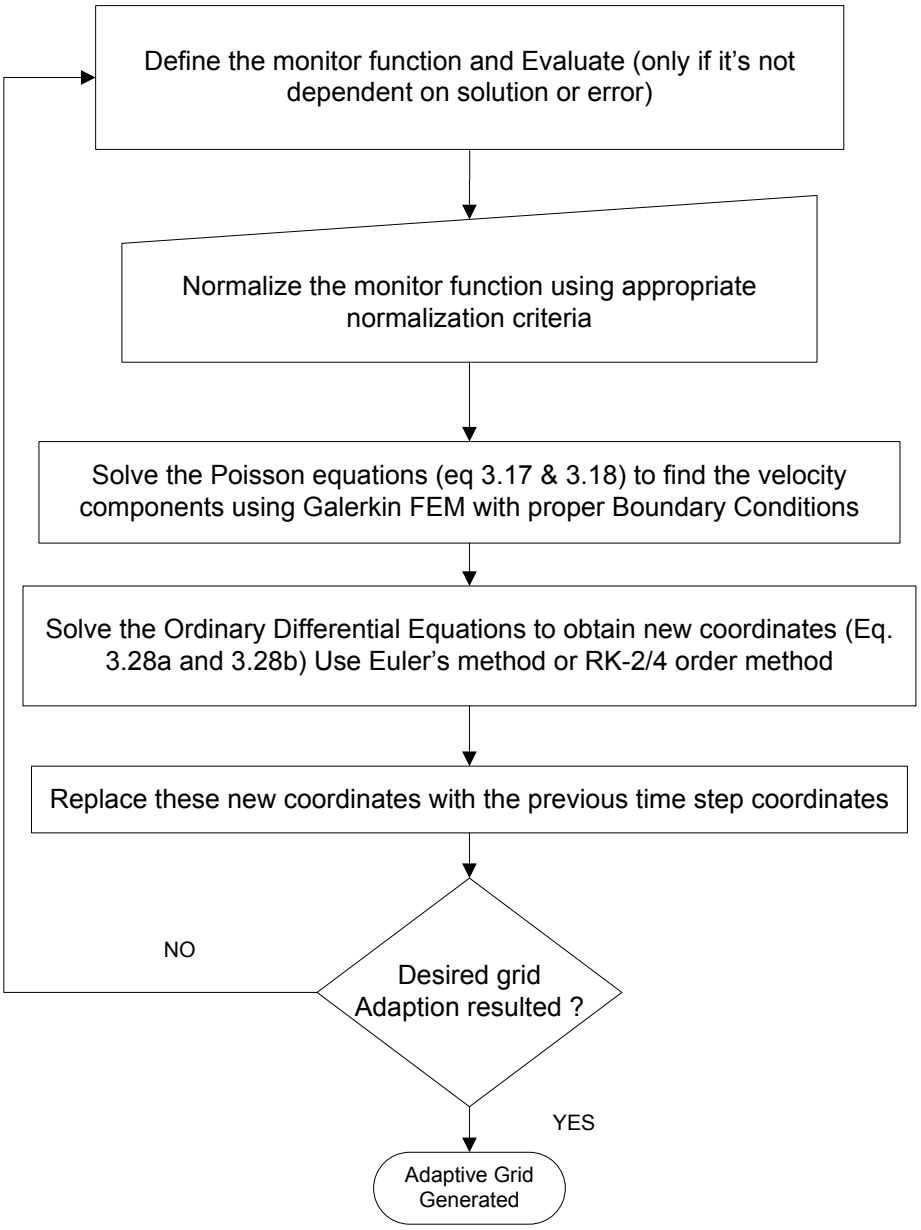


Figure 3.1: Roadmap for Adaptive Grid Generation using GFEM

CHAPTER 4

VALIDATION OF THE CODE AND NUMERICAL RESULTS

The present chapter discusses the construction and validation of the code developed using the Galerkin Finite Element Method (GFEM) to generate the adaptive grid around region of interest inside domain as we have discussed in the chapter 3.

4.1 Adaptive Grid Code

The code is developed with use of MATLAB. It is capable of generating 2D adaptive grids for specified monitor function. A moving grid to this adapted grid is added as extra feature to this code.

4.1.1 Input Parameters

This code needs the initial grid data (i.e. Nodal coordinates and Element connectivity) as input parameter. This work used initial grid data as grid generated from GAMBIT (commercial grid generation package). We can generate our own grid to consider as input grid data but it would be easier if we want to start with structured grid. The unstructured grid generation is bit difficult and takes much time so, we directly use it from the commercial grid generation packages and use the grid data for our work.

The code also requires the monitor function to be defined. Here, we used the model cases to validate the code and for that the monitor function is constructed on the basis of the distance function d which was used from the reference [10]. This code defines monitor function as a separate function file.

4.1.2 Adaptation of Grid

This part of the code solves two scalar Poisson equations as formed in the previous chapter with using boundary condition as also discussed in the previous chapter. The result of

these Poisson equations gives two components of the velocity field (u and v). Now, these components are used to solve two ordinary differential equations (ODEs) to find the new nodal coordinates. As described in the chapter 3, these ODEs are solved by using the forward Euler scheme in time with same size of time-step used to solve Poisson equations. Thus, we have new coordinates at the end of this part of code and they would be replaced as new coordinates for next time step before we move on to output part of the code.

4.1.3 Output from the Program

The output from this program is generated as Tecplot file format with grid adapted in the region of the large solution variation or region where we are interested to refine grid as per requirement. The new node coordinates file could be also generated as the program output if we need it to use for real flow problems. The result Tecplot file is visualized using Tecplot 9, which is available at Computational Fluid Dynamics (CFD) lab at The University of Texas at Arlington.

4.2 Validation of the code

4.2.1 Case I: Grid Refinement around sine Wave Curve

To validate the code for adaptive grid generation, the example is taken with use of uniform grid for 27 x 27 (729 nodes) over domain of [0,1] x [0,1] and the grid is clustered around sine wave curve with the use of following d and monitor function as used in [10].

$$d = \frac{1}{2} + \frac{1}{4}\sin(2\pi x) \qquad 0 \le t \le 1$$

$$f = \begin{cases} 1 & 0 \le y \le d - 0.1 \\ \left[0.3 - 7(y-d)t + (1-t)\right] & d - 0.1 < y \le d \\ \left[0.3 + 7(y-d)t + (1-t)\right] & d < y \le d + 0.1 \\ 1 & d + 0.1 \le y \le 2 \end{cases}$$

The grid is clustered around the sine wave in time 0 to 1 with using grid deformation method and the div-curl system solved using finite difference method (SOR method was used). The initial unstructured grid is created in GAMBIT with having almost same number of nodes (753 nodes) as in structured grid over the same domain [0,1] x [0,1]. The figure 4.1 and figure 4.2 shows initial grid for the unstructured grid and structured grid respectively.

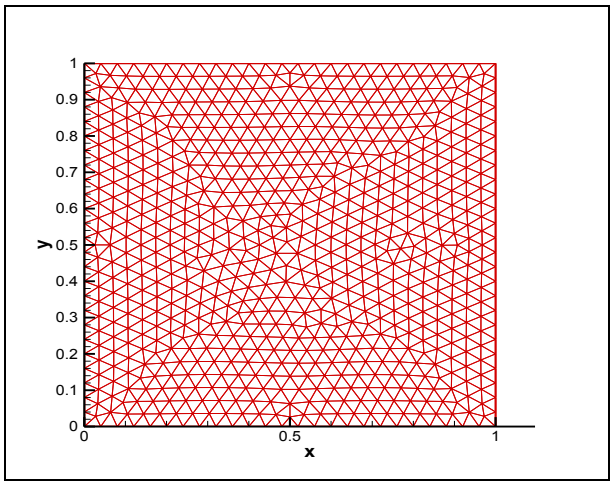


Figure 4.1: Adaptive Grid at time t=0 (Unstructured)

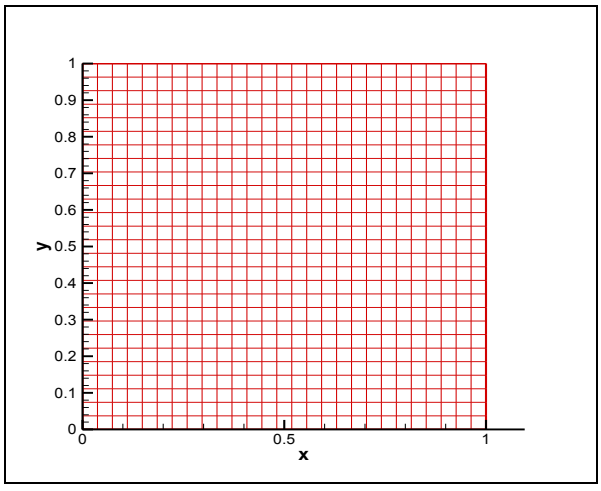


Figure 4.2: Adaptive Grid at time t=0 (Structured)

The figure 4.3 and figure 4.4 shows comparison between unstructured grid deformation using present work and the grid deformation using finite difference method for structured grid as demonstrated by De la Pena in [10] at total time t=0.5. Here, the time step size is kept same for both cases to validated results. From the figures we can verify that the results are pretty much in agreement at present total time t=0.5.

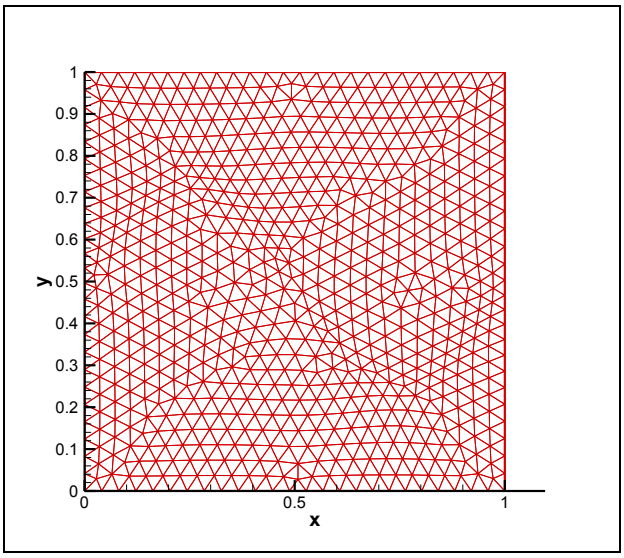


Figure 4.3: Adaptive Grid at time t=0.5 (Unstructured)

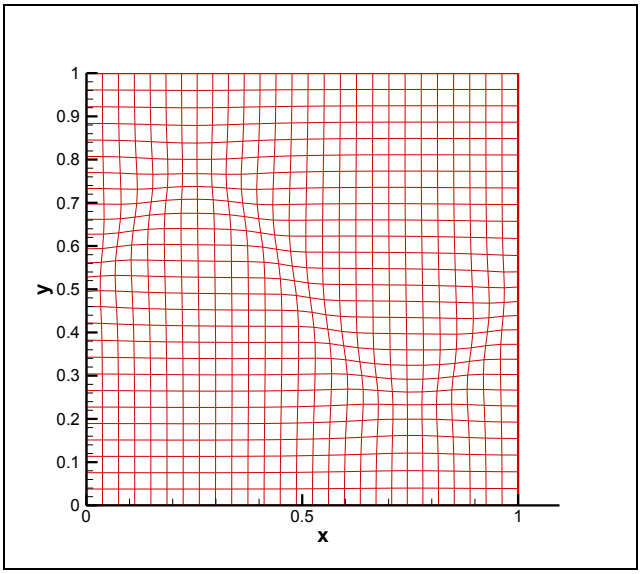


Figure 4.4: Adaptive Grid at time t=0.5 (Structured)

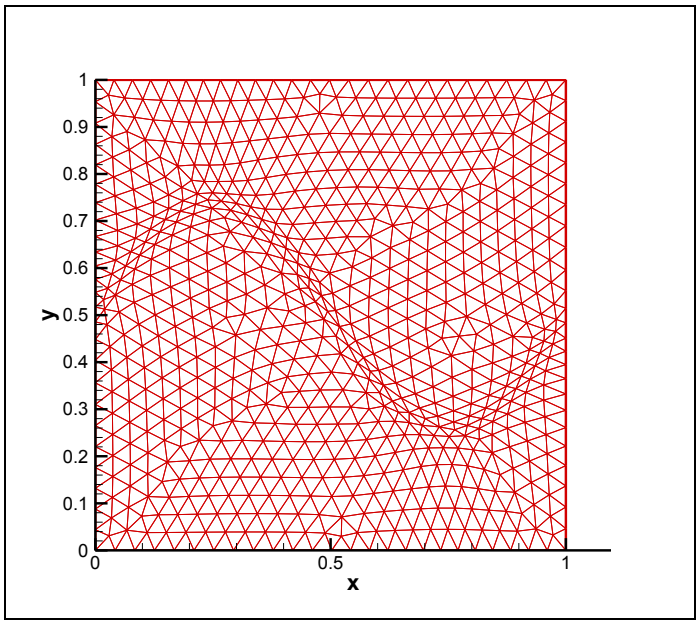


Figure 4.5: Adaptive Grid at time t=1 (Unstructured)

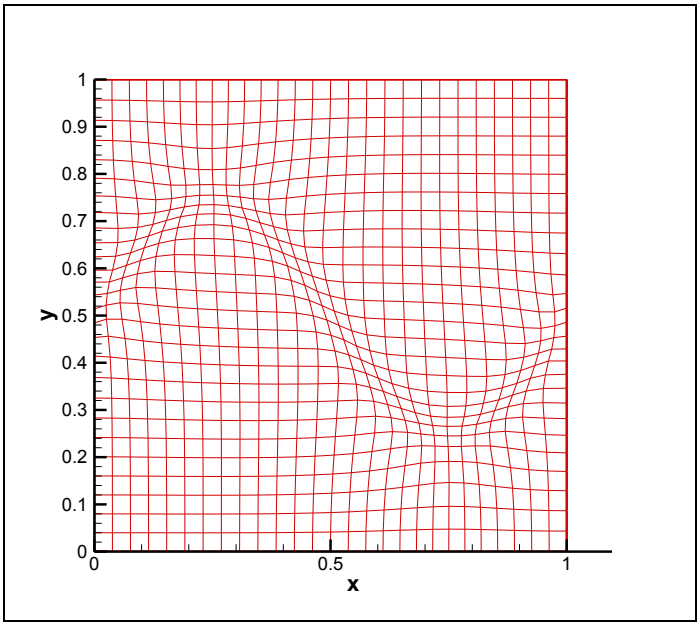


Figure 4.6: Adaptive Grid at time t=1 (Structured)

The figure 4.5 shows final adapted or clustered grid around sine wave curve at final time step t=1 using present work. The figure 4.6 represents final clustered grid using finite difference method for structured grids. The comparison of both figures validates that present work gives almost same results as derived with use of finite difference method to solve div-curl system by De la Pena in [10].

4.3 Numerical Example

4.3.1 Cluster the Grid around Circle

This example shows the demonstration of adaptive grid generation by showing the grid refinement at final time around particular region. Here, for shake of simplicity we clustered the grid around circular shape with a center (0.5,0.5) and radius of 0.15. The initial grid used is exported from GAMBIT and having 753 nodes and 1404 elements over the domain of [0,1] x [0,1]. The final time is defined as 1 and the time step size used is 0.1 for the adaptation. The monitor function used here is constructed using the distance function d defined [10] as follows,

$$d = (x-a)^2 + (y-b)^2 - r^2$$

Where, (a,b) is the center coordinate of circle and r is the radius of circle. (x,y) is the coordinates for node. Thus, it finds distance of each node from the center of circle and then use the following real-time monitor function in order to refine grid in desired region.

$$f = \begin{cases} 1-t+tc_1 & d \le 0 \\ 1-t+t(0.1+9d) & 0 \le d \le 0.1 \\ c_2 & d \ge 0.1 \end{cases}$$

where, c_1 and c_2 are size controlling parameters for grid adaptation intensity and they can be decided manually by using trial-error method as per refinement requirement. c_1 controls intensity of grid adaptation inside the boundary region and c_2 controls intensity of grid

adaptation outside the boundary region. The following results shows with the value of c_1 =0.3 and c_2 =1.2.

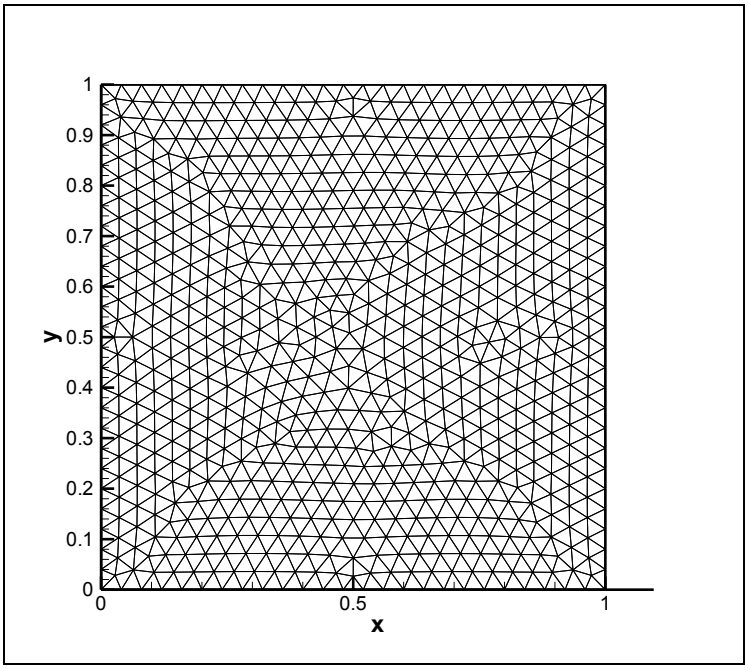


Figure 4.7: Adaptive grid for circle at initial time t=0

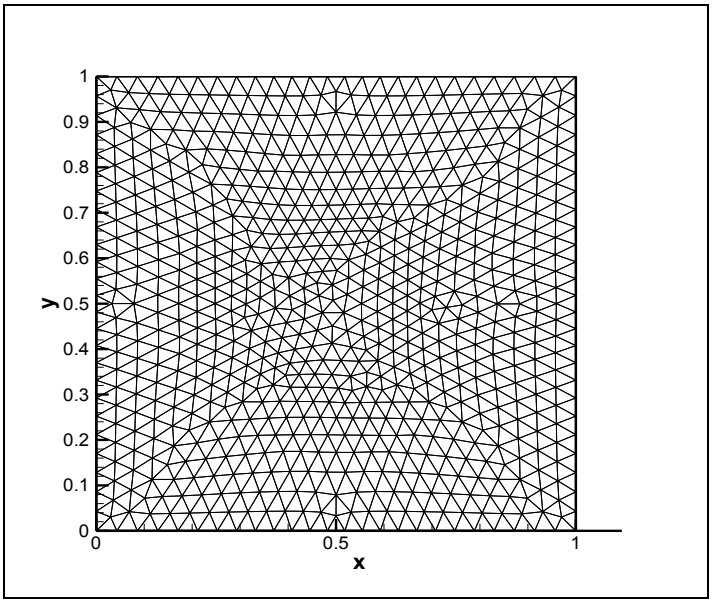


Figure 4.8: Adaptive grid for circle at time t=0.5

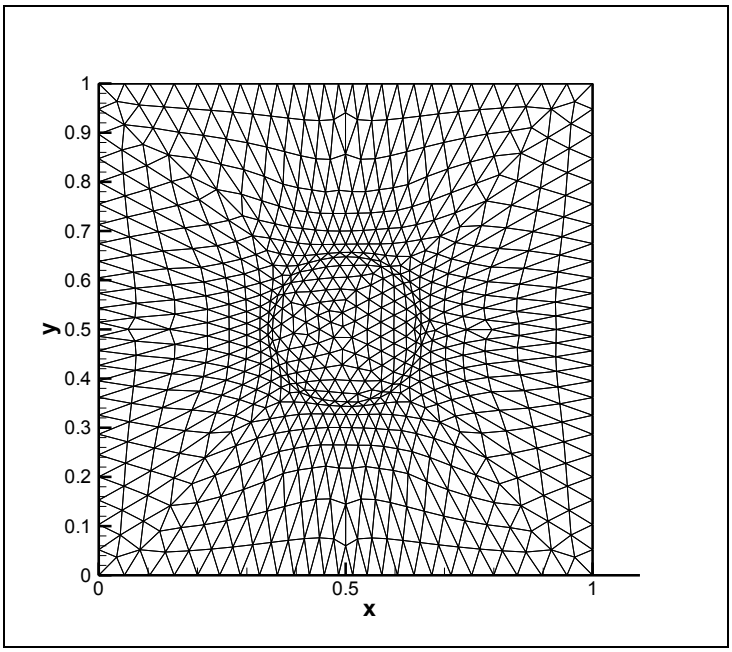


Figure 4.9: Adaptive grid for circle at final time t=1

The figure 4.7 shows initial grid used to adapt the circle with above mentioned center and radius. This grid was exported from GAMBIT and grid data are used as input for present grid adaptation code. The figure 4.8 shows adaptive grid for clustering around the circle at total time =0.5. It is clearly visible that center nodes stared refining towards circular region and its outer boundary. The figure 4.9 shows adapted grid at final time step of the prescribed time for adaptive grid generation. Here, we can clearly see that the grid is adapted in circular region and the circular boundary shows fine refinement as per appropriately chosen c_1 and c_2. We can change this refinement by changing the c_1 and c_2 values. The figure 4.10 is one of the examples of such change in grid adaptation intensity.

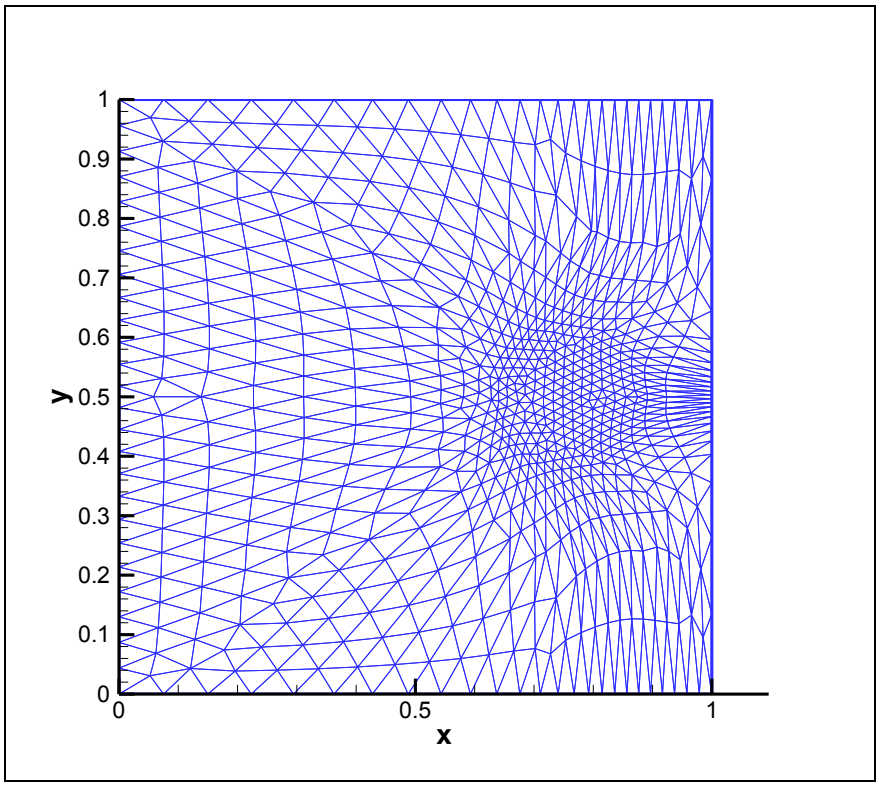


Figure 4.10: Adaptive grid for circle with different intensity of adaptation

The parameters used to generate adaptive grid shown in figure 4.10 are as follows,

Center coordinates = (0.75,0.5)

Radius of the circle = 0.1

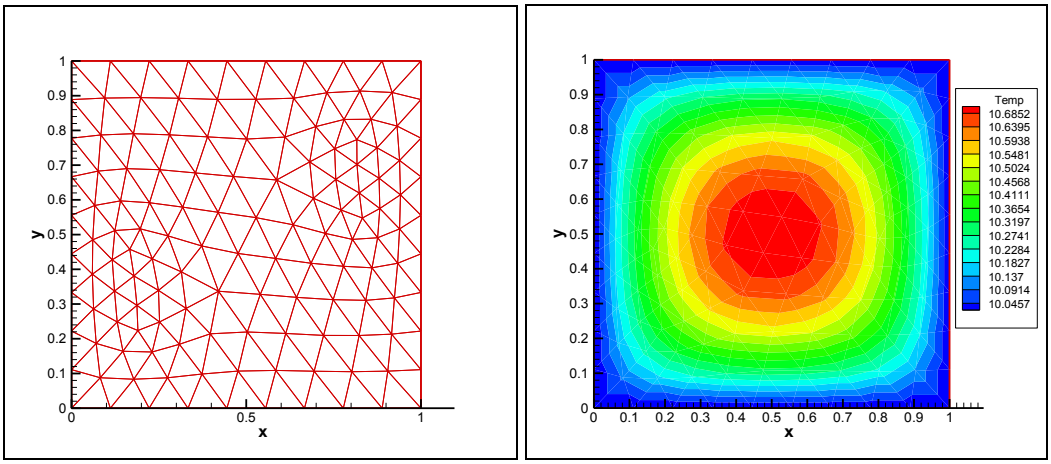
c_1 = 0.1

c_2 = 1.2

4.3.2 Heat Equation Solution using Adaptive Grid

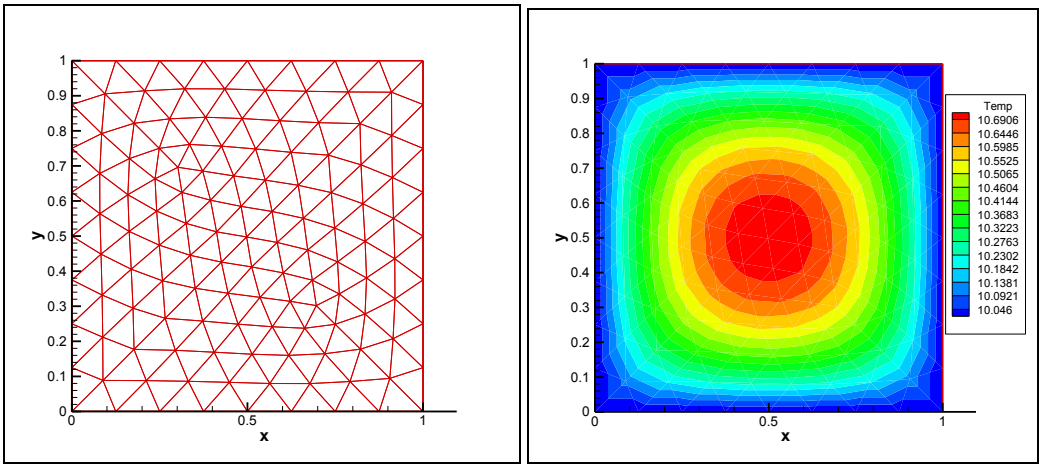
This section demonstrates use of this adaptive grid to solve the steady state heat equation with constant source term Q=10. In this part first, adaptive grid is generate over the given domain [0,1] x [0,1]. The initial grid is generated in GAMBIT with the 136 nodes and 234 elements for triangular shape elements. The grid is clustered around the center region of this linear triangular mesh. The final grid data is used to solve the steady state heat equation and

results are compared with solution of the steady state heat equation using same initial grid without adaptation as well as the same size of grid generated using quadratic triangular elements. The figure 4.11(a) shows initial grid using linear triangular elements and the figure 4.11(b) shows temperature distribution over the domain with use of grid shown in figure 4.11(a).



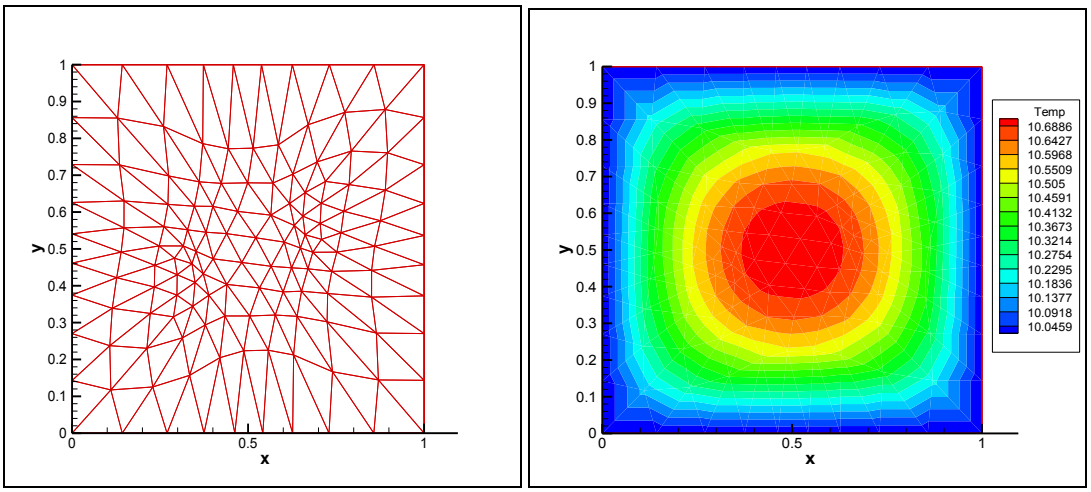
(a) (b)

Figure 4.11: (a) Unstructured grid with linear triangular elements (b) Temperature distribution using linear triangular element grid as shown in (a)



(a) (b)

Figure 4.12: (a) Unstructured grid with quadratic triangular elements (b) Temperature distribution using quadratic triangular element grid as shown in (a)



(a) (b)

Figure 4.13: (a) Unstructured grid with Adaptive grid around the center section (b) Temperature distribution using Adaptive grid around center as shown in (a)

The figure 4.12(a) shows grid using triangular quadratic elements (triangular element with 6 nodes instead of 3). The figure 4.12(b) shows the temperature distribution with use of the grid shown in figure 4.12(a). From the comparison of temperature distributions in figure 4.11(b) and figure 4.12(b), it is clearly visible that the temperature distribution is very good with use of quadratic triangular elements instead of linear triangular elements in the grid. The only disadvantage to use these quadratic triangular elements is, as it has 6 nodes in each element it makes programming complex. It is also complicated to formulate stiffness matrix during solution process. Thus, quadratic triangular element grid requires more computation time than linear triangular element grid for all other parameters remained fixed.

Now, we use adaptive grid for the same problem. We adapted grid around center of the region (we can adapt grid around any region of interest. It was clustered at center just as an example). The results with use of the adapted grids are shown in the figure 4.13(b) and the adapted grid is shown in figure 4.13(a). Now, if we compare results from figure 4.13(b) and figure 4.11(b), it shows that there is much more improvement in the temperature distribution over the adapted region. If we compare these results with the temperature distribution obtained

with use of quadratic triangular element grid, we can conclude that it gives as much as efficient result around adapted grid region although it doesn't give that much good for the non adapted region as in quadratic triangular element grid. This takes just little more time than the time required to solve this problem without adaptation as presented in figure 4.11(a) & (b).

Thus, we can conclude that adaptive grid gives the same order of accuracy in adaptation region as the grid with quadratic triangular elements at less computational time than quadratic triangular element grid. The only possible limitation is accuracy level changes from refined grid region to non-refined grid region.

CHAPTER 5

SUMMARY AND FUTURE RECOMMENDATIONS

5.1 Concluding Remarks

An algorithm is developed to generate adaptive grid with the use of Galerkin Finite Element Method and Grid Deformation Method. The Grid Deformation Method was originally developed by G. Liao and D. A. Anderson [4] for finite difference schemes and later on used by *X. Cai et al.* [16] and Tilak [17] with LSFEM (Least Square FEM) method. A mesh adaptation algorithm based on div-curl system [17] was successfully used with some modification to solve the system of equations and also validated.

The proposed algorithm successfully demonstrated the use of the Grid Deformation Method with the triangular elements and Galerkin Finite Element Method. The numerical computation for the steady state Heat equation is carried out with use of this adaptive mesh using triangular elements and compared it with the solution carried out by using triangular quadratic elements. The ℓ^2 –norm of truncation error in the grid refined region with triangular element almost showed the same result as the static grid with use of triangular quadratic elements. Thus, it proves that this method can produce efficient results where the solutions show large variation with less computational efforts and less computational time.

The moving of this adapted grid with change in the behavior of the solution is also achieved up to certain limitation such as it allows moving this adapted grid by small amount of distance per time step.

5.1 Scope for Future Work

This method uses div-curl system of equations as defined in the chapter 3 and it adapts the grid by using monitor function which could be constructed as discussed in section 2.3. This

div-curl system shows that the field is not truly irrotational as we use $\nabla \times \vec{u} = 0$ and the actual velocity which is used to find new coordinates (for next time step) of nodes in ODE is $f\vec{u}$. So, there could be some rotation found in the nodal coordinates even if we choose slight wrong monitor function or even if it is not properly normalized over the domain. The method can be constructed as completely irrotational if the div-curl equation system would be modified in such a way that it uses $\nabla \times (f\vec{u}) = 0$ instead of $\nabla \times \vec{u} = 0$ and then solve this new modified div-curl system of equation to achieve the refined grid. The formation of this new modified method requires more skillful work to form the weak form and then in assembly of the global stiffness matrix compared to one used in present work. The use of higher order elements can be also future recommendation for this method and it needs to be tested on more real problems, before it could be applicable to general problems.

There is no direct control over skewness of the elements with using this grid adaptation method. The skewness problems only appeared when we used triangular elements. Thus, there should be some work possible to control skewness of elements during grid adaptation process.

APPENDIX A

SHAPE FUNCTIONS, ELEMENTAL STIFFNESS MATRIX AND FORCE VECTOR USED FOR LINEAR TRIANGULAR ELEMENT

The present works used 2D linear triangular element for unstructured grid generation. The following figure shows the linear triangular element.

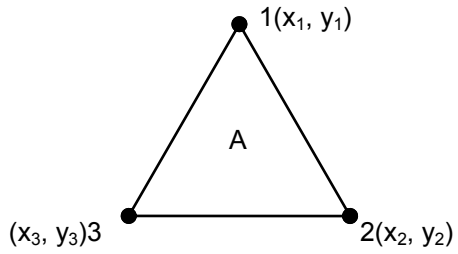


Figure A.1: 2D linear triangular element

The area of the above triangle can be computed as,

$$A = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \tag{A.1}$$

The shape function N is defined as follows,

$$[N] = \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \end{Bmatrix} \tag{A.2}$$

Where, L_1, L_2, L_3 are natural coordinates of any point on the triangular element and they can be written as linear function of x and y as follows,

$$L_i = \frac{A_i}{A} = \frac{1}{2A}(a_i + b_i x + c_i y), \qquad i = 1,2,3 \tag{A.3}$$

where, A is area of the linear triangular element and the other parameters are defined as follows,

$$a_1 = \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} = (x_2 y_3 - x_3 y_2) \qquad b_1 = -\begin{vmatrix} 1 & y_2 \\ 1 & y_3 \end{vmatrix} = (y_2 - y_3) \qquad c_1 = \begin{vmatrix} 1 & x_2 \\ 1 & x_3 \end{vmatrix} = (x_3 - x_2)$$

$$a_2 = \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix} = (x_3 y_1 - x_1 y_3) \qquad b_2 = -\begin{vmatrix} 1 & y_3 \\ 1 & y_1 \end{vmatrix} = (y_3 - y_1) \qquad c_2 = \begin{vmatrix} 1 & x_3 \\ 1 & x_1 \end{vmatrix} = (x_1 - x_3)$$

$$a_3 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} = (x_1 y_2 - x_2 y_1) \qquad b_3 = -\begin{vmatrix} 1 & y_1 \\ 1 & y_2 \end{vmatrix} = (y_1 - y_2) \qquad c_3 = \begin{vmatrix} 1 & x_1 \\ 1 & x_2 \end{vmatrix} = (x_2 - x_1)$$

Now, the elemental stiffness matrix used for present work as mentioned in equations (3.23a) and equation (3.26a) can be obtained by substituting the shape function defined in equation (A.2). The stiffness matrix for linear triangular elements is obtained as following,

$$[K] = \frac{1}{4A}\begin{bmatrix} b_1^2 + c_1^2 & b_1 b_2 + c_1 c_2 & b_1 b_3 + c_1 c_3 \\ b_1 b_2 + c_1 c_2 & b_1^2 + c_1^2 & b_2 b_3 + c_2 c_3 \\ b_1 b_3 + c_1 c_3 & b_2 b_3 + c_2 c_3 & b_1^2 + c_1^2 \end{bmatrix} \tag{A.4}$$

The elemental force vector used to solve the matrix system described in equation (3.22) and defined in the equation (3.23b) can be expressed as follows,

$$[CX] = \frac{1}{6}\begin{bmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \tag{A.5}$$

The elemental force vector used to solve the matrix system described in equation (3.25) and defined in the equation (3.26b) can be expressed as follows,

$$[CY] = \frac{1}{6}\begin{bmatrix} c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{A.6}$$

The following formula is used to find the integration for equations (3.23a), (3.23b), (3.26a) and (3.26b),

$$\int_{x_1}^{x_2} \alpha^a \beta^b \gamma^c dx = \frac{a!b!c!(x_2 - x_1)}{(a+b+c+1)!} \quad \text{or} \quad \int \alpha^a \beta^b \gamma^c dA = \frac{a!b!c!}{(a+b+c+1)!} A \tag{A.7}$$

The matrices represented in equations (A.4), (A.5) and (A.6) are the elemental matrices. The global matrices can be obtained by assembling all elemental matrices together.

APPENDIX B

NORMALIZATION OF MONITOR FUNCTION FOR LINEAR TRIANGULAR ELEMENTS

The normalization of the monitor function is necessary in order to achieve the stable results for grid refinement. The present work uses grid deformation method third version. So, monitor function must satisfy the condition shown in equation (2.7) in order to be normalized over the domain. The way we normalize the monitor function for linear triangular unstructured grid is different than the normalization in structured grid. The structure grid has regular node distribution and regular element connectivity over the entire domain. Thus, areas of all the elements are same at the initial time step for structured grids. The unstructured grids have random node distribution over the domain. We are using linear triangular elements for unstructured grid which has different area for each element. This makes the normalization little more complex and it can be carried out in the following way.

We have the non-normalized function value at each node to begin with. Let say the non-normalized function is denoted by $\tilde{f}$ and normalized values are denoted by f . We have to find the integral of the non-normalized monitor functions over the domain as normalization factor to satisfy the equation (2.7). The following equation can be used to find the integral of non-normalized monitor function over the domain.

$$\int \frac{1}{\tilde{f}} d\Omega = \sum_{i=1}^{m} \frac{1}{\tilde{f}_i} A_i \tag{B.1}$$

where, m is total number of elements for the given domain, $\tilde{f}_i$ is non-normalized monitor function for i^{th} element and A_i is the elemental area for i^{th} element.

The $\tilde{f}_i$ can be found for each element by using following method. We will see the procedure for one element with having the following nodal coordinates as shown in figure B.1.

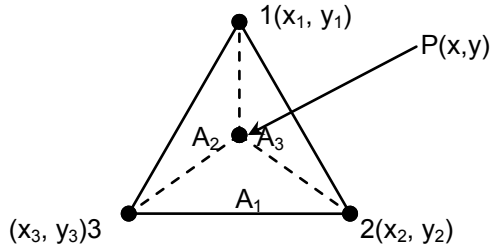


Figure B.1: 2D linear triangular element for normalization factor

We have centroid of triangle P is located as shown in figure with having its coordinates (x,y). We have value of monitor function for the nodes 1, 2 and 3. If we can find the value of monitor function at centroid, we can have the normalization factor by using area integral method for each element over the domain. To find the value of monitor function at centroid we need to find the areas for triangles produced by connecting nodes to the centroid as shown in figure B.1 (i.e. A_1, A_2, and A_3). These areas can be found using following formulas.

$$A_1 = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \qquad A_2 = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_3 & y_3 \\ 1 & x_1 & y_1 \end{vmatrix} \qquad A_3 = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix} \tag{B.2}$$

where,

$$x = \frac{x_1 + x_2 + x_3}{3} \quad \text{and} \quad y = \frac{y_1 + y_2 + y_3}{3} \tag{B.3}$$

Now if we have non-normalized monitor function values at nodes 1,2 and 3 as $\widetilde{f}_1$, $\widetilde{f}_2$ and $\widetilde{f}_3$ respectively. To find the value of non-normalized monitor function for element we can use the following interpolation formula,

$$\frac{1}{\widetilde{f}} = \frac{A_1}{A}\frac{1}{\widetilde{f}_1} + \frac{A_2}{A}\frac{1}{\widetilde{f}_2} + \frac{A_3}{A}\frac{1}{\widetilde{f}_3} \tag{B.4}$$

In equation (B.4), the A is elemental area and that can be found using equation (A.1). Now, we can have the value of non-normalized function for each element by using the equation (B.4) for each element over the domain. Now substitute these all elemental non-normalized function values in equation (B.1) and we will find the integration of non-normalized function over the domain which is normalization factor. Now we multiply this normalization factor to the non-normalized monitor function. The result would be the normalized monitor function f and it can be represented as following,

$$f_j = \widetilde{f}_j * \left(\sum_{i=1}^{m} \frac{1}{\widetilde{f}_i} A_i \right) \tag{B.5}$$

where, j designate the node number.

REFERENCES

[1] T. J. Baker, “Mesh Adaptation Strategies for problems in Fluid Dynamics”, (1997), Finite Elements in Analysis and Design, 25.

[2] http://www.cfd-online.com/Wiki/Mesh_adaptation

[3] G. Liao, Zhong Lei, Gary C. de la Pena and D. Anderson (2002), A Moving Grid Finite Difference Method for Partial Differential Equations.

[4] G. Liao and D. A. Anderson (1992), A new approach to grid generation. Appl. Anal., 44.

[5] J. Moser, Volume elements of a Rieman Manifold (1965), Trans AMS, 120.

[6] G. Liao, T. Pan, and J. Su “Numerical Grid Generator Based on Moser’s Deformation Method”,(1994), Numer. Math. PDE, 10, 21.

[7] F. Liu, S. Ji, and G. Liao, “An Adaptive Grid Method and its Application to Steady Euler Flow Calculations”, (1998), SIAM J. Sci. Comp., 20

[8] M. A. Akinlar, “A New method for Nonrigid Registration of 3D Images”, (2009), The University of Texas at Arlington.

[9] J. Liu, “New Development of the Deformation Method”, (2006), The University of Texas at Arlington.

[10] G.L.A.C.De la Pena, “Adaptive Grid Generation”, (1998), The University of Texas at Arlington.

[11] D.A. Anderson, “Equidistribution Schemes, Poisson Generators, and Adaptive Grids”, (1987), Applied Math. and Comp., 24.

[12] J. N. Reddy and D.K. Gartling, “The Finite Element Method in Heat Transfer and Fluid Dynamics”, (2001), Second Edition, CRC Press.

[13] K. H. Huebner, D. L. Dewhirst, D. E. Smith, T. G. Byrom, “The Finite Element Methods for Engineers”, (2001), Fourth Edition, John Wiley & Sons.

[14] O.C. Zienkiewicz, R. L. Taylor & P. Nithiarasu, “The Finite Element Method for Fluid Dynamics”, (2005), Sixth Edition.

[15] R. Kumar, “A least-squares/ galerkin split finite element method For incompressible and compressible Navier-stokes equations”, (2008), The University of Texas at Arlington.

[16] X. Cai, B. Jiang, and G. Liao, “Adaptive Grid Generation based on the Least-Square Finite-Element Method”, (2004), Computers and Mathematics with Applications, Elsevier.

[17] A. Tilak, “Solution Adaptive Mesh using Moving Mesh Method”, (2003), The University of Texas at Arlington.

BIOGRAPHICAL INFORMATION

Monalkumar Patel received his Bachelors degree in Aeronautical Engineering from Gujarat University, Gujarat, India in 2007. He worked as Project Graduate Trainee at National Aerospace Laboratories (NAL), Bangalore, India during his senior project. He has started working on his Master of Science in Aerospace Engineering at UT Arlington in Fall 2007. His current research of interest includes Adaptive grid generation using Finite Element Methods and its application to CFD problems.