Element Free Galerkin Method: Theory, Programming and Applications

By

Gohel Vivek P.

08MCL005

DEPARTMENT OF CIVIL ENGINEERING

AHMEDABAD-382481

May 2010

Element Free Galerkin Method: Theory, Programming and Applications

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Civil Engineering

(Computer Aided Structural Analysis and Design)

By

Gohel Vivek P.

08MCL005

Guide:

Dr. B. S. Munjal

Co. Guide:

Dr. H. V. Trivedi

DEPARTMENT OF CIVIL ENGINEERING

AHMEDABAD-382481

May 2010

Declaration

This is to certify that

(i) The thesis comprises my original work towards the degree of Master of Technology in Civil Engineering (CASAD) at Nirma University and has not been submitted elsewhere for a degree.

(ii) Due acknowledgement has been made in the text to all other material used.

Gohel Vivek P.

Certificate

This is to certify that the Major Project entitled "EFG Method:Theory, Programming and Applications" submitted by Gohel Vivek P. (08MCL005), towards the partial fulfillment of the requirements for the degree of Master of Technology in Civil Engineering (CASAD)of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. H. V. Trivedi
Co-Guide,
Director, Academic Development
and Research cell,
Nirma University,
Ahmedabad.

Dr. B. S. Munjal
Guide, Scientist/Engineer-SG at
Antenna Systems Area,
Space Applications Center(SAC),
Indian Space Research Organization (ISRO),
Ahmedabad.

Dr. P. H. Shah
Head of Department(Civil),
Institute of Technology,
Nirma University,
Ahmedabad.

Dr K. Kotecha
Director,
Institute of Technology,
Nirma University,
Ahmedabad.

Abstract

With the advent of high speed digital computers and innovative numerical techniques, there is volcanic proliferation in the domain of handling complex structural problem using umpteen number of approximate methods viz. FEM, Finite Difference Method (FDM), Finite Volume Method (FVM), Finite Strip Method, etc. In today's world of challenge, a buzz word in the corner is the development of an approach which is not banking on the "Meshing Concept" of the geometry or continuum but working on the contemporary "Mesh Free" approach. This can handle a large number of non-conventional problems including those of fracture mechanics and non-linear domains and thermal analysis field. Researchers are still working hard in the maze of this new emerging research area and are still struggling and working very hard in the area of the Mesh free methods.

An humble attempt has been made in this piece of work to first understand the subject which is almost at its infancy in the world, with only handful of researchers are working on this topic. The detailed literature survey has been carried out to explore and understand various methods developed in mid 1990's and their possible application areas. Thrust has been given on the understanding of wide gamut of approaches, with prime focus on Element Free Galerkin (EFG) method only. In the present work, because of availability of limited literature in this domain; efforts have been made in getting an insight into the modus-operandi involved. EFG method uses, least square interpolants to construct the trial and test functions for variational principle (weak form); the dependant variable and its gradient are continuous in the entire domain. Procedure for construction of shape function using moving least square (MLS) approximation is presented. Various terms related to EFG method such as support domain, weak forms, choice of weight function and EFG formulations are presented.

Software for analysis of 1D and 2D problems by EFG method are developed and tested. First a step by step procedure to solve a 1D bar using EFG with interim

computations is presented. Further a problem of 1D bar problem of temperature domain is analyzed and results are plotted. In both the types of problems, results are in good agreement with available results. A problem of 1D cantilever beam is analyzed, here number of gauss point are taken as twice the number of nodes. Results converge very fast as number of nodes increases. To check the versatility of the method as well as of programs, various other boundary conditions such as simply supported, propped cantilever and fixed beams are also considered for analysis.

Finally two dimensional plane stress problems are studied. Detailed flowchart is presented. The program in C language is developed for structural analysis. The Timoshenko beam problem analysis is carried out by two methods i.e. Lagrange multiplier method and Penalty method. The results are discussed and difficulties of both the methods are enumerated.

In chapter 7, important conclusions, summary and future scope are described.

Acknowledgements

I am deeply indebted to my thesis supervisors Dr. H. V. Trivedi and Dr. B. S. Munjal for their constant guidance and motivation. They have devoted significant amount of their time to plan and discuss the thesis work. Without their experience and insights, it would have been very difficult to do such a quality work.

I also like to extend my gratitude to Dr. P. H. Shah, Dr. P. V. Patel, Prof. N. C. Vyas, Prof. G. N. Patel and all other teachers for their moral and constructive support.

Special thanks to Dr. S. Bordas, Dr. V. P. Nguyen and S. Natrajan for their support and suggestions through Internet which provided deeper insight of the subject.

I also like to thanks my parents, my wife Sejal and my family. Their support to me in hundred little ways means a lot.

Last, but not the least, no words are enough to acknowledge constant support of friends and classmates for fruitful discussions during preparation of this thesis.

- Gohel Vivek P.

08MCL005

Abbreviation \ Nomenclature

FEM ..... Finite Element Method

FDM ..... Finite Difference Method

FVM ..... Finite Volume Method

EFG ..... Element Free Galerkin

MLPG ..... Meshless Local Petrov Galerkin

DEM ..... Diffuse Element Method

SPH ..... Smooth Particle Hydrodynamics

RKPM ..... Reproducing Kernel Particle Method

SEFGM ..... Stochastic Element Free Galerkin Method

RBF ..... Radial Basis Funtion

PIM ..... Point Interpolation Method

d_s ..... Dimension of support domain

ϕ_i ..... Shape function

σ ..... Stress vector

ε ..... Strain vector

L ..... Differential operator

E ..... Young's modulas

μ ..... Poisson's ratio

$\hat{W}$ ..... Weight function

Γ ..... Boundary function

Ω ..... Support domain

K ..... Nodal stiffness matrix

G ..... Nodal matrix for boundary nodes

f_t ..... Force vector

u ..... Displacement

t ..... traction

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Introduction

Designing advanced engineering systems require the use of computer aided design (CAD) tools. In such tools, computational simulation techniques are often used to model and investigate physical phenomena in an engineering system. The simulation requires solving the complex differential or partial differential equations that govern these phenomena. Traditionally, such complex partial differential equations are largely solved using numerical methods, such as the finite element method (FEM) and the finite difference method (FDM). In these methods, the spatial domain where the partial differential governing equations are defined is often discretized into mesh. A mesh is defined as any of the open spaces or interstices between the strands of a net that is formed by connecting nodes in a predefined manner. In FDM, the meshes used are also often called grids; in the finite volume method (FVM), the meshes are called volumes or cells; and in FEM, the meshes are called elements. The terminologies of grids, volumes, cells, and elements carry certain physical meanings as they are defined for different physical problems. However, all these grids, volumes, cells, and elements can be termed meshes according to the above definition of mesh. The key here is that a mesh must be predefined to provide a certain relationship between the

nodes, which is the base of the formulation of these conventional numerical methods. By using a properly predefined mesh and by applying a proper principle, complex differential or partial differential governing equations can be approximated by a set of algebraic equations for the mesh. The system of algebraic equations for the whole problem domain can be formed by assembling sets of algebraic equations for all the meshes. The mesh free method is used to establish a system of algebraic equations for the whole problem domain without the use of a predefined mesh. Mesh free methods use a set of nodes scattered within the problem domain as well as sets of nodes scattered on the boundaries of the domain to represent (not discretized) the problem domain and its boundaries. These sets of scattered nodes do not form a mesh, which means that no information on the relationship between the nodes is required, at least for field variable interpolation.

1.2 Why mesh free methods?

FEM is robust and has been thoroughly developed for static and dynamic, linear or nonlinear stress analysis of solids, structures, as well as fluid flows. Most practical engineering problems related to solids and structures are currently solved using a large number of well-developed FEM packages that are commercially available. However, the following limitations of FEM are becoming increasingly evident:

a. Creation of a mesh for the problem domain is a prerequisite in using FEM packages. Usually the analyst spends the majority of his or her time in creating the mesh, and it becomes a major component of the cost of a simulation project because the cost of CPU (central processing unit) time is drastically decreasing. The concern is more the manpower time, and less the computer time. Therefore, ideally the meshing process would be fully performed by the computer without human intervention.

b. In stress calculations, the stresses obtained using FEM packages are discontin-

uous and less accurate.

c. When handling large deformation, considerable accuracy is lost because of the element distortions.

d. It is very difficult to simulate both crack growth with arbitrary and complex paths and phase transformations due to discontinuities that do not coincide with the original nodal lines.

e. It is very difficult to simulate the breakage of material into a large number of fragments as FEM is essentially based on continuum mechanics, in which the elements formulated cannot be broken. The elements can either be totally "eroded" or stay as a whole piece. This usually leads to a misrepresentation of the breakage path. Serious error can occur because the nature of the problem is nonlinear, and therefore the results are highly path dependent.

f. Re-mesh approaches have been proposed for handling these types of problems in FEM. In the re-mesh approach, the problem domain is re-meshed at steps during the simulation process to prevent the severe distortion of meshes and to allow the nodal lines to remain coincident with the discontinuity boundaries. For this purpose, complex, robust, and adaptive mesh generation processors have to be developed. However, these processors are only workable for 2D problems. There are no reliable processors available for creating hexahedral meshes for 3D problems due to the technical difficulty.

g. Adaptive processors require "mappings" of field variables between meshes in successive stages in solving the problem. This mapping process often leads to additional computation as well as a degradation of accuracy. In addition, for large 3D problems, the computational cost of re-meshing at each step becomes very high, even if an adaptive scheme is available.

h. FDM works very well for a large number of problems, especially for solving fluid dynamics problems. It suffers from a major disadvantage in that it relies on

regularly distributed nodes. Therefore, studies have been conducted for a long time to develop methods using irregular grids. Efforts in this direction are still ongoing.

A close examination of these difficulties associated with FEM reveals the root of the problem: the need to use elements, which are the building block of FEM. A mesh with predefined "connectivity" is required to form the elements. As long as elements must be used, the problems mentioned above will not be easy to solve. Therefore, the idea of eliminating the elements and hence the mesh has evolved naturally. The concept of element free or mesh free methods has been proposed, in which the domain of the problem is represented by a set of arbitrarily distributed nodes. The Mesh Free method has great potential for solving the difficult problems mentioned above. Adaptive schemes can be easily developed, as there is no mesh, and hence no connectivity concept involved. Thus, there is no need to provide a priori any information about the relationship of the nodes. This provides flexibility in adding or deleting points/nodes whenever and wherever needed. For stress analysis of a solid domain, for example, there are often areas of stress concentration, even singularity. One can relatively freely add points in the stress concentration area without worrying about their relationship with the other existing nodes. In crack growth problems, nodes can be easily added around the crack tip to capture the stress concentration with desired accuracy. This nodal refinement can be moved with a propagation crack through a background arrangement of nodes associated with the global geometry. Adaptive meshing for a large variety of problems, 2D or 3D, including linear and nonlinear, static and dynamic stress analysis, can be very effectively treated in Mesh free methods in a relatively simple manner.

1.3 Objective of work

a. FEM is very powerful method for analysis of complex structural problems. The major effort required is in discretization of given problem into elements. This

step involves choice of type of element (from wide range available) and number of elements.

b. Mesh free methods recently developed obviate these difficulties. These methods are still in infancy stage. More study is required to be done so that practitioners can use with confidence.

c. In this study, objective is to examine various mesh free methods, choose one of the method after literature survey for further detailed study.

d. Next objective is to study theory of the chosen method in detail, write computer program for application for analysis of different types of structures.

e. To carry out structural analysis of different problems.

1.4 Scope of work

In order to fulfill the objectives mentioned above, the scope of the present study is as follows:

a. Literature survey to examine various mesh free methods.

b. From the literature survey described in chapter 2, it is observed that Element Free Galerkin method can be studied in detail i.e. fundamentals of theory and studying each step in detail for applying to different types of structural analysis problems.

c. Based on the understanding developed, detailed flowchart is to be written and subsequently computer program is to be developed and to be tested for various problems.

d. Finally to discuss the outcome of results as a part of total study of structural analysis by EFG method.

1.5 Organization of thesis

The content of major project is divided into various chapters as follows:

Chapter 1, Introduction, presents the introduction and overview of the mesh free approach. It states various difficulties faced in various analysis tools available in market and suitability of mesh free approach for it. It gives objective of study and scope of work of major project.

Chapter 2, Literature review, presents the literatures which have been studied and used for the preparation of this thesis. It highlights various problems attainted and its suitability studied by the author. **Chapter 3**, Mesh free approach, presents the introduction part of the mesh free approach. It gives understanding about various engineering mechanics problems present in practice. The chapter presents solution procedure followed for analysis of problem by FEM and Mesh free method. It also gives understanding of various terminologies used for mesh free methods. Chapter gives list of various mesh free methods in table form with what type of system of equation it has used and which approximation method has been used for shape function creation.

Chapter 4, Mesh free shape function creation, presents various methods used for creation of shape function in mesh free methods. It gives procedure for creation of shape function for one dimensional and two dimensional problems by Moving Least Square method.

Chapter 5, Element Free Galerkin (EFG) method, presents the theory and procedure of the EFG method. Chapter also presents the analysis steps to be followed for one dimensional bar problem, one dimensional beam problem and two dimensional beam problems.

Chapter 6, EFG application, presents the application of EFG method for various structural engineering problem. Analysis of one dimensional bar has been given with sample problem. Analysis of one dimensional beam problem has been given with a sample problem. A cantilever beam with parabolic end load is analysed and studied

by parametric studies. Analysis of plate in tension with circular hole in centre has been done. Analysis of deep beam problem is done and results have been compared with those of Krishnamoorthy[14].

Chapter 7, Conclusions summary and future scope, the chapter contains the important conculsions, summary and the future scope of work.

Chapter 2

LITERATURE REVIEW

2.1 General

In the present chapter, review of various technical papers is presented.

T. Belytschko, Y. Y. Lu and L. Gu [28] In this paper the authors have described a new method for analysis of the 1D and 2D solid mechanics. It is an extension of Diffuse Element Method given by Nayroles et al. The method developed is called as Element Free Galerkin (EFG) method. In this method, only a mesh nodes and boundary description is needed for generation of Galerkin equation. The Moving Least Square approximation is used as an interpolant for curve and surface fitting. In this paper methodology for using this method has been given. Various examples are given to validate the method.

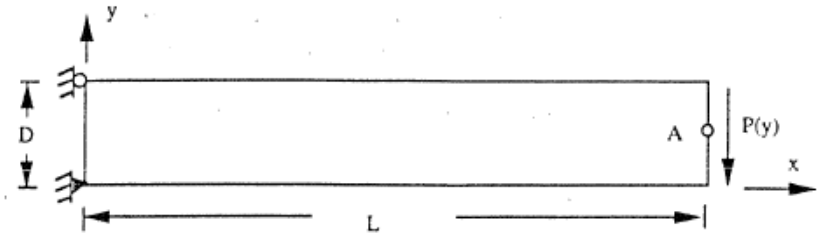


Figure 2.1: Cantilever beam with parabolic shear end load

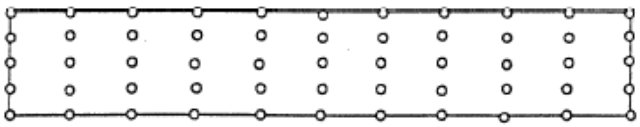


Figure 2.2: Regular data points

They are, Patch test for square plate, cantilever beam with parabolic shear load at the end, a square plate with circular hole in center, plate with edge crack under uniaxial stress. Figs. 2.1 and 2.2 shows the example of cantilever beam with parabolic shear end load and with its equally distributed nodes.

Y. Krongauz, T. Belytchko[33] In this paper a technique for easily treating essential boundary conditions for approximations which are not interpolants, such as Moving Least Square approximations in the Element-Free Galerkin (EFG) method, Smooth Particle Hydrodynamics (SPH) and Reproducing Kernel Particle Method (RKPM) is presented. The rates of convergence are compared to those obtained for essential boundary conditions imposed via a modified variational principle. These comparisons show that the accuracy of the new method is somewhat better. Furthermore, the new method can be directly applied to different systems of partial differential equations without developing a modified variational principle. Various problems such as patch test, cantilever beam and plate with hole are presented.

P. Krysl and T. Belytschko[24] In this paper a mesh free approach to the analysis of arbitrary Kirchhoff shells by the Element-Free Galerkin (EFG) method is presented. The method is mesh free, which means that the discretization is independent of the geometric subdivision into "finite elements". The satisfaction of the C^0 continuity requirements is easily met by EFG since it requires only C^1 weights; therefore, it is not necessary to resort to Mindlin-Reissner theory or to devices such as discrete Kirchhoff theory. The numerical examples which are studied are pinched cylinder, Scordelis-Lo barrel vault, hemispherical shell and extensional bending of

cylinder.

S. Beissel, T. Belytschko [25] A stabilization procedure has been developed for the application of nodal integration to elasto-statics. The benefits are:

a. There is no need to create a quadrature structure – it is generated automatically with the spatial discretization.

b. The data structure is as simple as that of a particle method. All discretized variables are defined over one set of nodes.

c. Evaluation of the integrals of the weak form is much quicker than for complete Gaussian integration.

These advantages make nodal integration the simplest EFG method to implement, especially for problems with complex grids or adaptive refinement. In fact, whereas the original element-free Galerkin method was not truly element-free (a background mesh of cells was retained for the spatial integration of the weak form), nodal integration requires only a cloud of nodes and a specification of the surfaces of the body for a discretization.

J. Dolbow and T. Belytschko[11] Detailed description of the EFG and its numerical implementation has been presented with the goal of familiarizing scientist and engineers with the new computational technique. Flowchart for 1-D and 2-D programming has been given. Problem of various problems such 1-D and 2-D examples have been given.

Y. Krongauz and T. Belytschko[34] A technique for incorporating discontinuous derivatives in Mesh free formulations has been described for one- and two-dimensional problems. The technique is based on adding an approximation function which has a discontinuous first derivative at the discontinuity. The strength of the discontinuity is represented by additional unknowns. In multidimensional problems, the strength of discontinuity is interpolated over the discontinuity line (or surface in three dimensions). The discontinuous approximation has been constructed so that it

has compact support, and consequently the resulting discrete equations are sparse and banded. The results and problems discussed are 1-D in-homogenous rod, inclusion in an infinite plate.

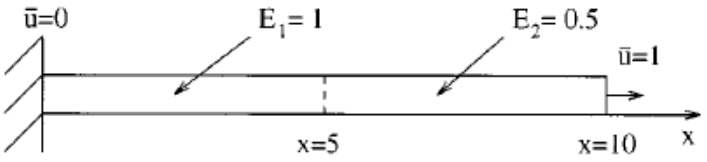


Figure 2.3: 1D Bi-material rod

P. Krysl and T. Belytschko[23] In this paper authors have examined the implications of solving second-order elliptic problems in non-convex, two-dimensional domains by the Element-Free Galerkin method. Their conclusions are,

a. For meshes properly graded to account for the singularity, the consistency term due to the non-conformity of the EFG basis governs the rate of convergence for k > 1. Thus, only the conforming variant of the EFG should be used with higher-order basis.

b. For linear basis (k = 1), the rate of convergence is not affected by the discontinuous shape functions. For a quasi-uniform mesh, the error is governed by the singularity due to the non-convex boundaries, and for properly graded meshes both the approximatively, and the consistency term estimate the same rate of convergence, equal to one.

S. Rahman, B. N. Rao[26] A stochastic element-free Galerkin method (SEFGM) has been developed for reliability analysis of linear-elastic structures with spatially varying random material properties. The SEFGM is used to evaluate the probabilistic response and reliability of linear static structures subject to spatially varying random material properties. In this paper, the problems which are studied are bar element

and square plate in tension.

J. Belinha, L.M.J.S. Dinis [4] In this paper the EFG method has been extended to elastic and elasto plastic analysis of anisotropic symmetric laminates. A Reissner-Mindlin plate theory, which is a first-order shear deformation theory (FSDT) is considered and briefly described in order to define the displacement field and the strain field. A modified version of the Newton-Rahpson method is used for the solution of the nonlinear system of equations and an anisotropic yield surface, Hill yield surface, is considered. The elasto-plastic algorithm of solution is described. The authors have compared the FEM and the EFGM there are some drawbacks for the EFGM. The computational cost of the EFGM is higher, the imposition of the essential boundary conditions is more complex and there is a high sensitivity of the method in what concerns the choice of the influence domain and the choice of the weight function. It has been supported by various problems.

V. P. Nguyen et al.[29] This paper presents a practical overview of meshless methods (for solid mechanics) based on global weak forms through a simple and well-structured MATLAB code, to illustrate our discourse. Several one and two-dimensional examples in elasto-statics are given including weak and strong discontinuities and testing different ways of enforcing essential boundary conditions. Some of the figures described in this paper are as follows. Fig. 2.4 shows the discretization of problem domain which shows various parameters of mesh free methods, fig. 2.5 shows arrangement of integration cell and gauss points, fig. 2.6shows a problem of crack propagation and fig. 2.7 shows the deformed shape of problem defined in fig. 2.6.

G. R. Liu[18] This is the only book available on the subject of Mesh free method. In this book the author has given a very good description of various Mesh free methods and their fundamental equations and practical applications.

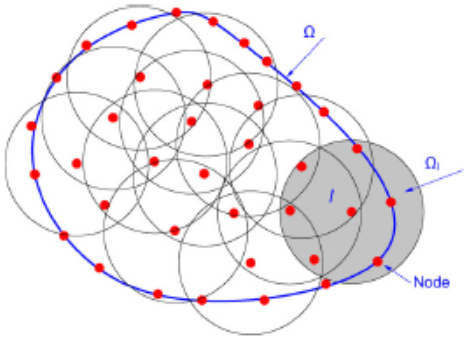


Figure 2.4: Discretization using meshless methods: nodes, domains of influence (circular shape)

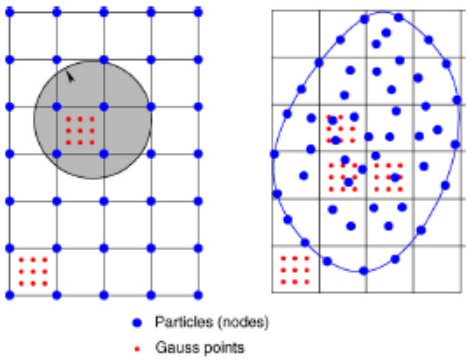


Figure 2.5: Integration in Galerkin-based MMs: background mesh (left) and background structure cells (right)

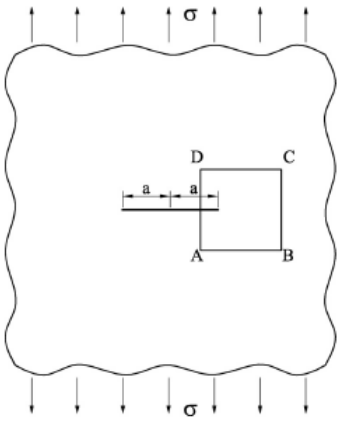


Figure 2.6: Infinite cracked plate under remote tension: geometry and loads.

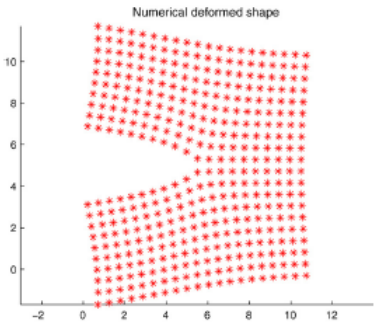


Figure 2.7: Computed deformed configuration (scaled).

2.2 History of mesh free methods

There are a number of Meshfree methods, such as the element free Galerkin (EFG) method (Belytschko et al., 1994b) [28], the meshless local Petrov-Galerkin (MLPG) method (Atluri and Zhu, 1998)[3], the point interpolation method (PIM) (Liu, G. R. and Gu, 1999)[19], the point assembly method (PAM) (Liu, G. R., 1999)[17], the finite point method (Onate et al., 1996)[7], the finite difference method with arbitrary irregular grids (Liszka and Orkisz, 1980; Jensen, 1980)[16], smooth particle hydrodynamics (SPH) (Lucy, 1977[21]; Gingold and Monaghan, 1977[10]), reproducing kernel

particle method (Liu, W. K. et al., 1993)[9], which is an improved version of SPH, and so forth. They all share the same feature that predefined meshes are not used, at least for field variable interpolation.

2.2.1 The Smoothed Particle Hydrodynamics

The advent of the mesh free idea dates back from 1977, with Monaghan and Gingold [10] and Lucy [21] developing a Lagrangian method based on the Kernel Estimates method to model astrophysics problems. This method, named Smoothed Particle Hydrodynamics (SPH), is a particle method based on the idea of replacing the fluid by a set of moving particles and transforming the governing partial differential equations into the kernel estimates integrals [18]. Despite the success of the SPH in modelling astrophysics phenomena, it was only after the 90's that the SPH was applied to model others classes of problem. This consequently exposed some inherent characteristics of the method, such as tensile stability, energy zero, lack of interpolation consistency and difficulty in enforcing essential boundary conditions [27]. The latter two are consequences of using the SPH to model bounded problems, since originally the SPH was formulated to model open problems. Over the past years many modifications and correction functions have been proposed in an attempt to restore the consistency and consequently the accuracy of the method [20]. The SPH method has been successfully applied to a wide range of problems such as free surface, impact, magneto-hydrodynamics, explosion phenomena, heat conduction and many other computational mechanics problems which were presented and discussed in extensive reviews of SPH [15] [20].

2.2.2 The Diffuse Element Method (DEM)

The first mesh-free method based on the Galerkin technique was only introduced over a decade after Monaghan and Gingold first published the SPH method. The Diffuse Element Method (DEM) was introduced by Nayroles and Touzot in 1991. Many

authors state that it was only after the Diffuse Element method that the idea of a mesh-free technique began to attract the interest of the research community. The idea behind the DEM was to replace the FEM interpolation within an element by the Moving Least Square (MLS) local interpolation [22].

2.2.3 Element Free Galerkin (EFG) Method

In 1994 Belytschko and colleagues introduced the Element-Free Galerkin Method (EFG) [9], an extended version of Nayroles's method. The Element-Free Galerkin introduced a series of improvements over the Diffuse Element Method formulation. This method has been discussed in detail in chapter 5. The Element-Free Galerkin method was found to be more accurate than the Diffuse Element method, although the "improvements" implemented in the method increased its computational costs [28]. EFG is one of the most popular mesh-free methods and its application has been extended to different classes of problems such as fracture and crack propagation, wave propagation , acoustics and fluid flow.

2.2.4 Reproducing Kernel Particle Method

In 1995 Liu proposed the Reproducing kernel particle method (RKPM) [9] in an attempt to construct a procedure to correct the lack of consistency in the SPH method. The RKPM has been successfully used in multi scale techniques, vibration analysis, fluid dynamics and many other applications. Later, a combination of MLS and RKPM resulted in the Moving Least Square Reproducing Kernel Particle method [30] [31] .

2.2.5 Finite Point Method

The Finite Point method was proposed by Onate and colleagues in 1996 [8] [7]. It was originally introduced to model fluid flow problems and later applied to model many other mechanics problems such as elasticity and plate bending. The method

is formulated using the Collocation Point technique and any of the following approximation techniques, Least Square approximation(LSQ), Weighted Least Square approximation (WLS) or Moving Least Squares (MLS) can be used to construct the trial functions.

2.2.6 Meshless Local Petrov-Galerkin

The Meshless Local Petrov-Galerkin introduced by Atluri and Zhu in 1998 [3] presents a different approach in constructing a mesh-free method. It is based on the idea of the Local weak form which eliminates the need of the background cell and, consequently, performs the numerical integration in a meshfree sense. The MLPG uses the Petrov-Galerkin method in an attempt to simplify the integrand of the weak form. The method was originally formulated using the MLS technique and later Atluri extended the MLPG formulation to other meshless approximation techniques. The freedom of choice for the test function in the Petrov-Galerkin method gives rise to different MLPG schemes [2]. The MLPG and its different schemes have been applied to a wide range of problems such as Euler-Bernoulli Beam Problems, solid mechanics, vibration analysis for solids, transient heat conduction, amongst many others.

2.2.7 Radial Basis Functions

Radial basis functions (RBF) were first applied to solve partial differential equations in 1991 by Kansa, when a technique based on the direct Collocation method and the Multiquadric RBF was used to model fluid dynamics [12] [13] . The direct Collocation procedure used by Kansa is relatively simple to implement, however it results in an asymmetric system of equations due to the mix of governing equations and boundary conditions. Moreover, the use of Multiquadric RBF results in global approximation, which leads to a system of equations that is characterised by a dense stiffness matrix. The RBF Hermite-Collocation was proposed as an attempt to avoid the asymmetric system of equations. Both globally and compactly supported radial basis functions

have been used to solve PDEs and results have shown that the global RBF yielded better accuracy. However the compactly supported stiffness matrix is sparse, while the global RBF results in a dense matrix that may become very expensive to solve in the case of a large number of collocation points. Recently, another approach based on the global RBFs has been proposed. The method, named Local Multiquadric, suggests the construction of the approximation using sub-domains, causing the Multiquadric RBF to be truncated at the "boundaries" of the sub-domains, resulting in a local approximation and a sparse stiffness matrix [6]. Radial basis functions have also been used in the Boundary Element method formulation, such as in the Dual Reciprocity Method (DRM), Method of Fundamental solution (MFS) and the RBF Boundary Knot method (BKM). These methods have been successfully applied to solve non-linear problems in Computational Mechanics. A variational approach to solve the Boundary Value Partial (BVP) using compactly supported radial basis functions has been investigated by [32] and another approach suggested the use of radial basis functions in the Meshless Local Petrov-Galerkin formulation [2]. In the last decade the radial basis function approximation technique has undergone intensive research. However, a large number of publications on the subject concern its mathematical proof and foundations. An extensive review of the mathematical background of RBFs is presented in [31]. Some applications of the RBFs in the solution of physical problems worthy of mention are transport phenomena, heat conduction, analysis of Kirchoff Plates and Euler-Bernoulli beam problems amongst others.

2.2.8 Point Interpolation Method

The Point Interpolation method (PIM) uses the Polynomial interpolation technique to construct the approximation. It was introduced by Liu in 2001 as an alternative to the Moving Least Square method [5]. The PIM, originally based on the Galerkin method, suffers from the singularity of the interpolation matrix and also from the fact that it does not guarantee the continuity of the approximation function. Several

approaches have been investigated by Liu in an attempt to overcome these problems [18]. Improvements have been obtained using the Local Petrov- Galerkin method and Multiquadric radial basis functions. This procedure resulted in Local Radial Point Interpolation methods (LRPIM). The LRPIM has been applied to solid mechanics, fluid flow problems and others. These applications are referred to and examined in detail in [18].

2.3 Different Mesh free methods

Various mesh free methods developed in recent years are tabulated in table 2.1 [18]. The table includes title of method, the author who presented it, which type of equation solution method used and method used for function approximation.

2.4 Comparison between FEM and Mesh Free Methods

The differences between FEM and Mesh free method are shown in table 2.2

Method	**Ref.**	**System Equation to be Solved**	**Method of Function Approximation**
Diffuse element method	Nayroles et al., 1992	Weak form	MLS approximation, Galerkin method
Element free Galerkin (EFG) method	Belytschko et al.,[28]	Weak form	MLS approximation, Galerkin method
Meshless local Petrov-Galerkin (MLPG) method	Atluri and Zhu, 1998	Local Weak form	MLS approximation, Petrov-Galerkin.
Finite point method	Onate et al., 1996; Liszka and Orkisz, 1980; Jensen, 1980	Strong form	Finite differential representation (Taylor series), MLS
Smooth particle hydrodynamics	Lucy, 1977; Gingold and Monaghan, 1977	Strong form	Integral representation
Reproducing kernel particle method	Liu, W. K. et al., 1993	Strong form or Weak form	Integral representation (RKPM)
hp-clouds	Oden and Abani, 1994; Armando and Oden, 1995	Weak form	Partition of unity, MLS
Partition of unity FEM	Babuka and Melenk,1995	Weak form	Partition of unity, MLS
Point interpolation method	Liu, G. R. and Gu, 1999, 2000b, 2001a,b,c,d	Weak form and Local Weak form	Point interpolation
Boundary node methods	Mukherjee and Mukherjee, 1997a,b	Weak form and Local Weak form	MLS
Boundary point interpolation methods	Liu, G. R. and Gu, 2000d; Gu and Liu, G. R., 2001a,e	Weak form and Local Weak form	Point interpolation

Table 2.1: Different Mesh free methods and their features[18]

Item	**FEM**	**Mesh Free Method**
Element mesh	Yes	No
Mesh creation and automation	Difficult due to the need for element connectivity	Relatively easy and no connectivity is required
Mesh automation and adaptive analysis	Difficult for 3D cases	Can always be done
Shape function creation	Element based	Node based
Shape function property	Satisfy Kronecker delta conditions; valid for all elements of the same type	May or may not satisfy Kronecker delta conditions depending on the method used; different from point to point
Discretized system stiffness matrix	Banded, symmetrical	Banded, may or may not be symmetrical depending on the method used
Imposition of essential boundary condition	Easy and standard	Special methods may be required; depends on the method used
Computation speed	Fast	1.1 to 50 times slower compared to the FEM depending on the method used
Retrieval of results	Special technique required	Standard routine
Accuracy	Accurate compared with FDM	Can be more accurate compared with FEM
Stage of development	Very well developed	Infancy, with many challenging problems
Commercial software package availability	Many	Very few and close to none
Problem suitability	For regular solid mechanics problem	For complex problems such as crack propagation, eddy-current problem, electro-magnetic, wave-propagation, etc.

Table 2.2: Comparison of FEM and Mesh free methods[18]

Chapter 3

MESH FREE APPROACH

3.1 Introduction

In building a modern and advanced engineering system, engineers must undertake a very sophisticated process of modeling, simulation, visualization, analysis, designing, proto typing, testing, fabrication, and construction. The process is illustrated in the flowchart shown in Fig.3.1[18]. The process is very often iterative in nature; that is, some of the procedures are repeated based on the results obtained at the current stage to achieve optimal performance for the system under construction. In process of the concern is mainly to modeling and simulation, as well as some issues related to visualization, which are underlined in Fig. 3.1[18]. The focus will be on physical, mathematical, and computational modeling and computational simulation. These topics play an increasingly important role in building an advanced engineering system in a rapid and cost-effective way. Many methods and computational techniques can be employed to deal with these topics. The literature mainly focuses on the development and use of the Mesh Free methods. This chapter addresses the procedures of modeling and simulation using Mesh free methods and the differences between the Mesh Free method and other existing methods, especially the widely used finite element method. Fig. 3.2[18] presents the flow chart for the FEM and Mesh Free approach. Where

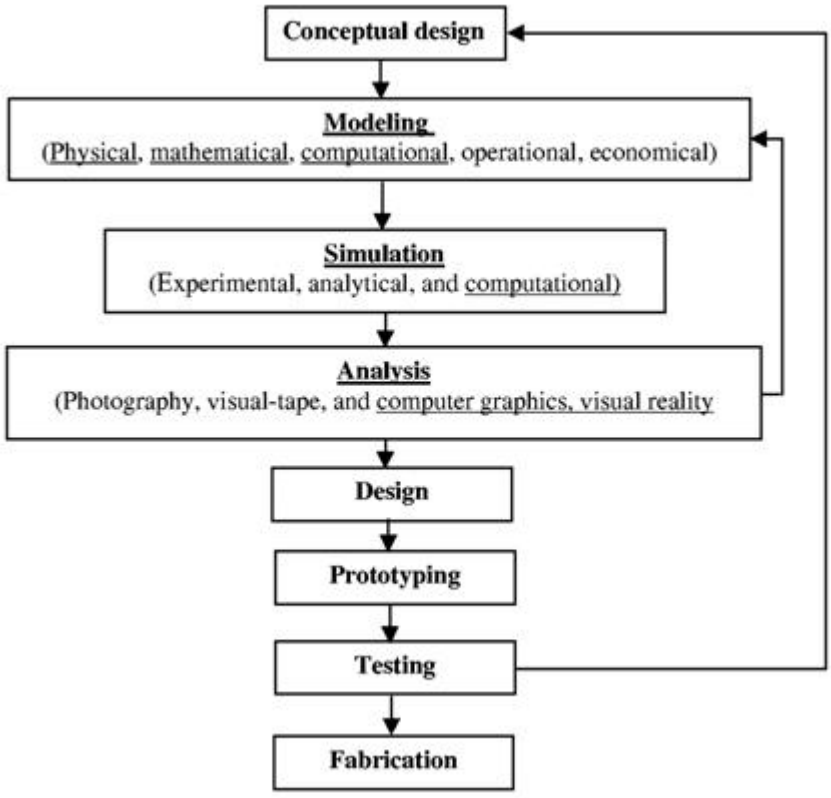


Figure 3.1: Processes that lead to building an engineered system

SPC means single point constraints and MPC means Multi Point Constraints used for the boundary conditions.

3.2 Engineering mechanics problems

The various engineering mechanics problems include structural analysis, heat transfer, fluid flow, mass transport, and electro-magnetic potential.

a. Typical structural problems include:

   - Stress analysis, including frame and truss analysis, and stress concentration problems typically associated with holes, fillets, or other changes in geometry of a body.
   - Buckling
   - Vibration analysis

b. Non-structural problems include:

   - Heat transfer
   - Fluid flow, including seepage through porous media
   - Distribution of electric or magnetic potential

c. A relatively new field of application is in the field of bio-engineering. Some biomechanical engineering problems (which may include stress analysis) typically include analysis of human

   - Spine
   - Skull
   - Hip Joints
   - Jaw / gum tooth implants
   - Heart
   - Eye

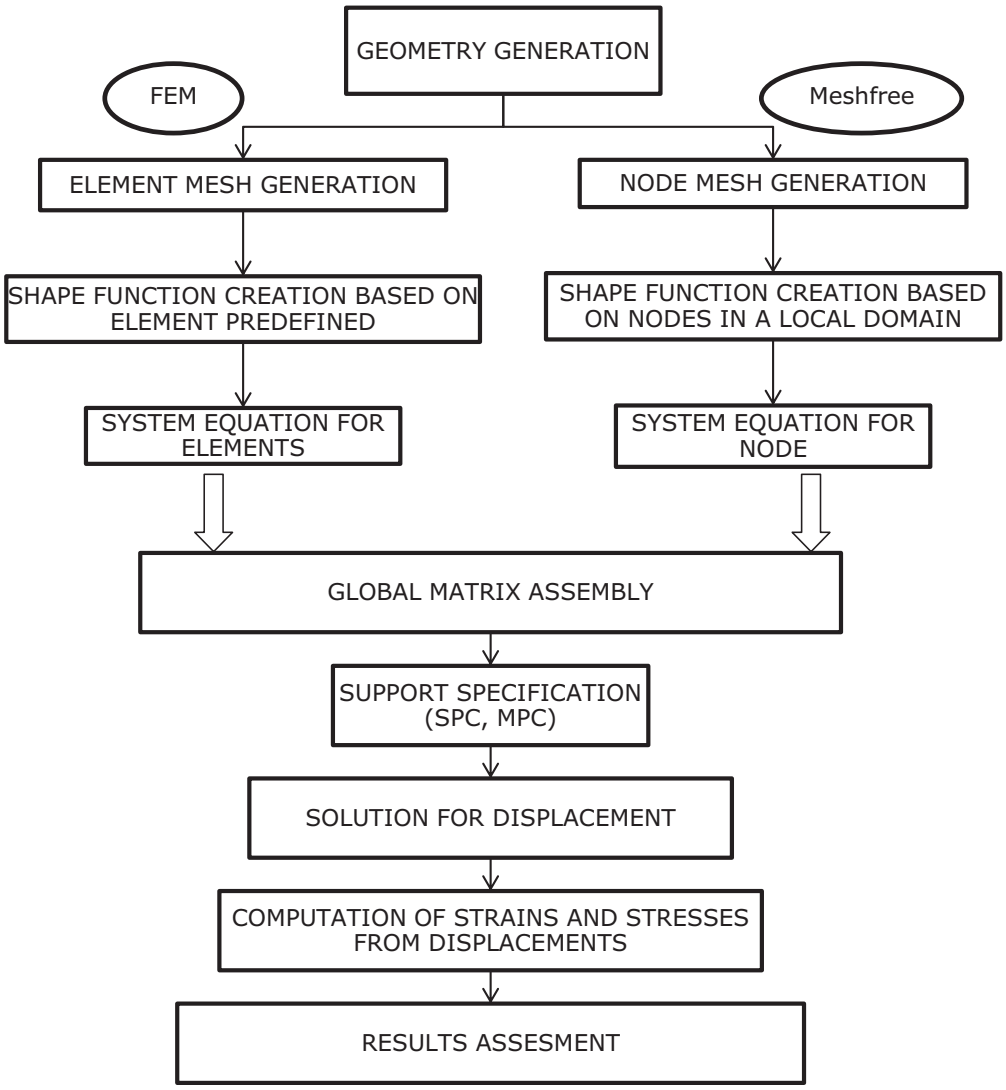


Figure 3.2: Flowchart for Mesh Free Method and FEM procedure

3.3 Theoretical Background of Mesh Free Methods

The procedure in FEM and the Mesh free method for solving engineering problems can in principle be outlined using the chart given in fig. 3.2. These two methods diverge at the stage of mesh creation. The fundamental difference between these two methods is the construction of the shape functions. In FEM, the shape functions are constructed using elements, and the shape functions will be the same for the element. In fact, if the natural coordinate systems are used, the shape functions in the natural coordinates are the same for all the elements of the same type. These shape functions are usually predetermined for different types of elements before the finite element analysis starts. In Mesh free methods, however, the shape functions constructed are usually only for a particular point of interest. The shape function changes as the location of the point of interest changes. The construction of the element free shape function is performed during the analysis, not before the analysis. Once the global discretized system equation is established, the Mesh free method follows a procedure similar to FEM, except for some minor differences in the details of implementation. Therefore, many techniques developed over the past decades in FEM can be utilized in Mesh free methods with or without modifications. The following sections present the basic procedures in Mesh free methods, by discussing the differences between FEM and the Mesh free method at major stages of analysis.

3.3.1 Modeling the Geometry

Real structures, components, or domains are in general very complex and have to be reduced to a manageable geometry. In FEM, curved parts of the geometry and its boundary can be modeled using curves and curved surfaces using high-order elements. However, it should be noted that the geometry is eventually represented by a collection of elements, and the curves and curved surfaces are approximated by piecewise curves

and surfaces of the elements. If linear elements are used, which is often the case in practical situations, these curves and surfaces are straight lines or at surfaces. Fig. 3.3 shows an example of a smooth boundary represented in the finite element model by straight lines of the edges of triangular elements. The accuracy of representation of the curved parts is controlled by the number of the elements and the order of the elements used. A finer mesh of elements can generally lead to more accurate results. However, because of the constraints on time and computational resources including hardware and software, it is always required to limit the number of elements. Therefore, fine details of the geometry need to be modeled only if very accurate results are required for those regions. The results of simulation have to be interpreted with these geometric approximations in mind. The analyst has to determine the distribution of the density of the mesh required to achieve a desired accuracy at important areas and regions of the problem domain.

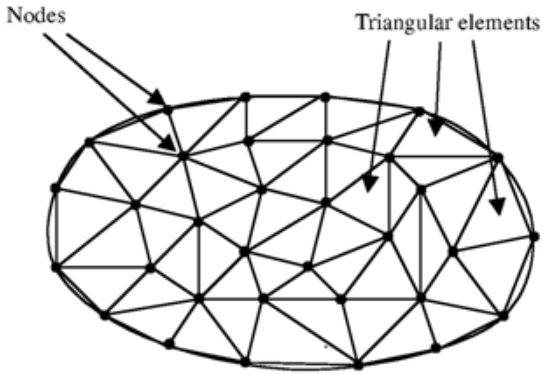


Figure 3.3: Smoothed boundary is represented in FEM by straight lines of the edges of triangular elements

In Mesh free methods, however, the boundary is represented (not discretized) by nodes, as shown in fig.3.4. At any point between two nodes on the boundary, one can interpolate using Mesh free shape functions. Because the Mesh free shape functions are created using nodes in a moving local domain, the curved boundary can be approximated very accurately even if linear polynomial bases are used. It

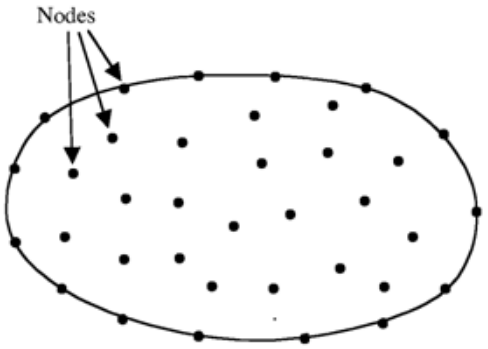


Figure 3.4: Smoothed boundary is represented in the mesh free method by nodes

is common in Mesh free methods to use higher-order polynomials. Note that this geometric interpolation can be performed using the same technique for field variable interpolation in Mesh free methods. Depending on the software used, there are many ways to create a properly simplified geometry in the computer. Points can be created simply by keying in the coordinates of the point. Lines or curves can be created by simply connecting points or nodes. Surfaces can be created by connecting or rotating or translating the existing lines or curves. Solids can be created by connecting or rotating or translating the existing surfaces. Points, lines or curves, surfaces, and solids can be translated or rotated or reflected to form new ones. Graphic interfaces are used for assisting the creation and manipulation of the objects. There are a number of CAD (computer-aided design) software packages used in engineering design that can produce files containing the geometry of the designed engineering system. These files can often be read by modeling software packages. Making use of the CAD files can save significant time in creating the geometry of the models. However, in many cases, the objects read directly from a CAD file may need to be modified and simplified before performing meshing. These tools for creating the geometry of the problem domain can be used for both the FEM and the Mesh free method. Knowledge, experience, and engineering judgment are very important in modeling the geometry of a system. In many cases, finely detailed geometric features play only an aesthetic role,

and will not affect the functionality or the performance of the engineering system very much. These features can be simply deleted, ignored, or simplified. This, however, may not be true for some cases, where a fine geometric change can give rise to a significant difference in the simulation results. Adaptive analysis is ideal for solving this problem objectively and independently of the judgment of the analyst. Mesh free methods provide more flexible ways for adaptive analysis. Another very important issue is the simplification required by mathematic modeling. For example, a plate has three dimensions geometrically, but the plate in the plate theory of mechanics is represented mathematically only in two dimensions. Therefore, the geometry of a "mechanics" plate is a two dimensional (2D) flat surface represented usually by the neutral surface. In FEM, plate elements are used in meshing the plate surfaces. A similar situation occurs in shells. A beam has also three dimensions geometrically. The beam in the beam theory of mechanics is represented mathematically only in one dimension. Therefore, the geometry of a "mechanics" beam is a one-dimensional (1D) straight or curved line. In FEM, beam elements have to be used to model the lines. A similar situation occurs in truss structures. In Mesh free methods, beams, plates, and shells can all be represented using sets of arbitrarily distributed nodes. In the formulation of the Mesh free methods, corresponding theories used in the FEM must be used. The difference, again, lies mainly in the creation of the shape functions.

3.3.2 Node Generation

In FEM, meshing is performed to discretize the geometry created into small meshes called elements or cells, and many types of elements have been developed for different problems. The rationale behind domain discretization can be explained in a very rough and straightforward manner. We can expect that the solution for an engineering problem will be very complex, and will vary in a way that is usually unpredictable using functions defined globally across the whole problem domain. However, if the problem domain can be divided (meshed) into small elements using a set of nodes

that are connected in a predefined manner using nodal lines, the solution within each element can be approximated very easily using simple functions such as polynomials, which are termed shape functions. The solutions for all the individual elements form the solution for the whole problem domain. Mesh generation is a very important part of the pre-process in FEM, and it can be a very time-consuming task for the analyst. The domain has to be meshed properly into elements of specific shapes such as triangles and quadrilaterals. No overlapping and gaps are allowed. Information, such as the element connectivity, must also be created during the meshing for later simulation. It is ideal to have an entirely automated mesh generator; unfortunately, one is not available on the market. Semiautomatic preprocessors are available for most commercially available application software packages. There also exist packages designed mainly for meshing. Such packages can generate files of a mesh, which can be read by other modeling and simulation packages. Triangulation is the most flexible way to create meshes of triangular elements. The process can be almost fully automated for 2D planes and even three-dimensional (3D) spaces. Therefore, it is used in most commercial preprocessors. The additional advantage of using triangles is the flexibility of modeling a complex geometry and its boundaries. The disadvantage is that the accuracy of the simulation results based on triangular elements is often much lower than that obtained using quadrilateral elements for the same density of nodes. Quadrilateral elements, however, are more difficult to generate automatically.

3.3.3 Shape Function Creation

In FEM, shape functions are created based on elements, and therefore, the computation of shape functions has been straightforward. In the early years of the development of FEM, much of the work involved the formulation of all different types of elements. All the shape functions of finite elements satisfy the Kronecker delta function property. In Mess free methods, however, the construction of shape functions has been and still is the central issue. This is because shape functions have to be computed

with the use of predefined knowledge about the relationship of the nodes. This has posed the major challenge for Mess free methods. The currently most widely used method for constructing Mess free shape functions is the method of moving least squares (MLS) approximation. The application of MLS approximation has led to the development of many Mess free methods and techniques. The major problem in MLS approximation is that the shape functions constructed do not possess the Kronecker delta function property.

3.4 Determination of the Dimension of a Support Domain

Support domain determines nodes (marked by o) that are used for approximation or interpolation of field variable at point x. A support domain can have different shapes and can be different from point to point. Most often used shapes are circular or rectangular as shown in figure 3.5.

The accuracy of interpolation depends on the nodes in the support domain of the point of interest (which is often a quadrature point x_Q or the center of integration cells). Therefore, a suitable support domain should be chosen to ensure a proper area of coverage for interpolation.

To define the support domain for a point x_Q, the dimension of the support domain d_s is determined by

$$d_s = \alpha_s * d_c \tag{3.1}$$

where α_s is the dimensionless size of the support domain and d_c is a characteristic length that relates to the nodal spacing near the point at x_Q. If the nodes are uniformly distributed, d_c is simply the distance between two neighboring nodes. In the case where the nodes are non-uniformly distributed, d_c can be defined as an "average" nodal spacing in the support domain of x_Q.

The physical meaning of the dimensionless size of the support domain $\alpha_s$is very clear.

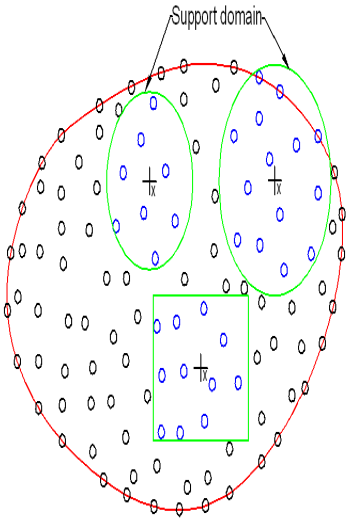


Figure 3.5: Support domain

It is simply the factor of the average nodal spacing. For example, $\alpha_s = 2.1$ means a support domain whose radius is 2.1 times the average nodal spacing. The actual number of nodes, n, can be determined by counting all the nodes in the support domain. The dimensionless size of the support domain α_s should be predetermined by the analyst, usually by carrying out numerical experiments for the same class of problems for which solutions already exist. Generally, an $\alpha_s = 2.0$ to 3.0 leads to good results. Note that, if background cells are provided, support domains can also be defined based on the background cells.

3.5 Property of Mesh free Shape Functions

A compulsory condition that a shape function must satisfy is the partition of unity, that is,

$$\sum_{\mathrm{i=1}}^{\mathrm{n}} \phi_i(x) = 1 \tag{3.2}$$

This is a necessary condition for the shape function to be able to produce any rigid motion of the problem domain.

There are also conditions that a shape function preferably satisfies. The first preferable condition is the linear field reproduction condition, that is,

$$\sum_{\mathrm{i=1}}^{\mathrm{n}} \phi_i(x) x_i = x \tag{3.3}$$

This condition is required for the shape function to pass the standard patch test, which has been used very often in testing finite elements. This condition is not compulsory because shape functions that fail to pass the patch test can still be used as long as a converged solution is produced. Many finite elements cannot pass the patch test but are widely used in FEM packages.

Another preferable condition is the Kronecker delta function property, that is,

$$\phi_i(x = x_i) = \{ \begin{array}{ll} 1 & i = j, \quad j = 1, 2, 3, \ldots, n \\ 0 & i \neq j, \quad i, j = 1, 2, 3, \ldots, n \end{array} \tag{3.4}$$

This condition is preferred because a shape function that possesses this property permits use of a simple procedure to impose essential boundary conditions. In element free methods, however, the shape functions created may or may not satisfy condition 3.4, depending on the method used for creating the shape functions.

3.6 Equations of elasticity

A two-dimensional (2D) linear solid mechanics problem is used to present the procedure of the EFG method in formulating discretized system equations. The stress components σ_{ij} must satisfy the differential equations of equilibrium in a domain Ω, of a body under consideration

$$\sigma_{ij,j} + F_i = 0 \qquad \textit{Equilibrium equation in problem domain } \Omega \tag{3.5}$$

where F_i are the components of body force acting on the object. The field equation are to be solved given certain boundary conditions as

$$u_i = \overline{u}_i \qquad \textit{Boundary condition essential boundary } \Gamma_{\mathrm{u}} \tag{3.6}$$

which gives constraints to the field variable of displacement. The natural boundary conditions are given by

$$t_i = \bar{t}_i \qquad \textit{Boundary condition essential boundary } \Gamma_{\mathrm{t}} \tag{3.7}$$

where the traction on a surface with normal n_j are given as $t_i = \sigma_{ij} n_j$, and $\overline{u}_i$ and $\bar{t}_i$ are the prescribed values of displacements and tractions on boundaries Γ_{u} and Γ_{t}, respectively. $\Gamma_{\mathrm{u}} \cup \Gamma_{\mathrm{t}}$=$\Gamma$ is the boundary of the object domain Ω. The stress components σ_{ij} in eqn.3.5 are symmetric (i.e., $\sigma_{ij} = \sigma_{ji}$). The kinematics equations, relating the strain components ε_{ij} to the displacements u_i are

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \tag{3.8}$$

Where a comma denotes differentiations; thus, $u_{i,j} = \partial u_i / \partial x_j$. The constitutive relations for a linear elastic material are given as

$$\sigma_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij} = E_{ijkl} \varepsilon_{kl} \tag{3.9}$$

Where λ and μ are Lame's constants and E_{ijkl} is the constitutive tensor eqns. 3.5 to 3.9 summarize the equations of elasostatics for a linear, isotropic, homogenous elastic object.

3.7 Concept of weak form

Obtaining the exact solution for a strong form system equation is ideal but, unfortunately, it is usually very difficult for practical engineering problems that are very complex. The finite difference method (FDM), which uses finite differential representation (Taylor series) of a function in a local domain, can be used to solve system equations of strong form to obtain an approximated solution. However, FDM requires a regular mesh of grids, and can usually work only for problems with regular geometry and simple boundary conditions. One of the Mesh free methods for solving strong form system equations to obtain an approximate solution is to use arbitrarily distributed grids based on Taylor series expansions, and least square or moving least squares (MLS) approximations. The formulation is very simple but less stable, and the accuracy of the results often depends on the selection of the nodes for constructing the differential equations. There are also Mesh free methods for solving strong form system equations using an integral representation of field variable functions, such as the smooth particle hydrodynamics (SPH) methods. The major problem with the SPH methods is the treatment of boundaries of the problem domain and the boundary conditions.

Formulation based on weak forms can produce a stable set of algebraic system equations and gives, discretized system equations that produce much more accurate results. Therefore, weak form is often preferred by many to obtain an approximated solution. Here weak form formulations are used to form discretized system equations for Mesh free methods for problems of engineering mechanics.

3.8 Weak forms of the governing equations

The first law of thermodynamics states that for a body in equilibrium and subjected to arbitrary virtual displacement δv_i, the variation in work of the external forces δW

is equal to the variation of internal energy δU. These quantities are written as

$$\delta W - \delta U = \delta W_S + \delta W_B - \delta U = 0 \tag{3.10}$$

$$\delta W_S = \int_{\Gamma_t} \bar{t} \delta v_i d\Gamma \tag{3.11}$$

$$\delta W_B = \int_{\Omega} F_i \delta v_i d\Omega \tag{3.12}$$

$$\delta U = \int_{\Omega} \sigma_{ij} \delta \varepsilon_{ij} d\Omega \tag{3.13}$$

Where $\delta\varepsilon_{ij} = \frac{1}{2}\left(\delta v_{i,j} + \delta v_{j,i}\right)$ is the strain due to the virtual displacement field δv_i. Since $\sigma_{ij} = \sigma_{ji}$, the product $\sigma_{ij}\ \delta\varepsilon_{ij}$ may be expressed eqn.3.8 as $\sigma_{ij}\ \delta v_{ij}$.

In the context of computational mechanics, the displacement field u_i that leads to the stress field σ_{ij} is referred to as the trial function, and the virtual displacement field δv_i is referred to as a test function.

By solving eqns.(3.10 to 3.13) the following weak form is obtained

$$W1: \quad \int_{\Omega} \sigma_{ij} \delta v_{i,j} d\Omega - \int_{\Omega} F_i \delta v_i d\Omega - \int_{\Gamma_t} t_i \delta v_i d\Gamma = 0 \tag{3.14}$$

It is noted that eqn. is the same relation as the virtual work relation in eqns. (3.10 to 3.13). Eqn.3.14 is adequate when the test function u_i satisfies the essential boundary conditions of eqn.(3.7 and 3.8) i.e., $u_i = \overline{u}_i$ on Γ_u. Alternate forms are resorted to when these conditions are not met. A Lagrange multiplier form Belytschko et al.,[tbel] may be written by augmenting eqn.3.14 to give the weak form

$$W2: \int_{\Omega} \sigma_{ij} \delta v_{i,j} d\Omega - \int_{\Omega} \delta F_i \delta v_i d\Omega - \int_{\Gamma_t} \bar{t}_i \delta v_i d\Gamma - \int_{\Gamma_u} \left(u_i - \overline{u}_i\right) \delta\lambda_i d\Gamma - \int_{\Gamma_u} \lambda_i \delta v_i d\Gamma = 0 \tag{3.15}$$

Where λ_i and $\delta\lambda_i$ are the Lagrange multiplier and its variation, respectively.

The essential boundary conditions can also be accounted for by means of a penalty formulation to give

$$W3: \quad \int_{\Omega} \sigma_{ij}\delta v_{i,j} d\Omega - \int_{\Omega} \delta F_i \delta v_i d\Omega - \int_{\Gamma_t} \bar{t}_i \delta v_i d\Gamma - \alpha \int_{\Gamma_u} (u_i - \overline{u}_i)\delta v_i d\Gamma = 0 \quad (3.16)$$

Where $\alpha \gg 1$ is a penalty parameter to enforce $u_i = \overline{u}_i$ on Γ_{u}. The choice of the weak form employed in a formulation depends on the strategy adopted therein to satisfy the essential boundary conditions.

Chapter 4

MESH FREE SHAPE FUNCTION CREATION

4.1 Introduction

Creation of Mesh free shape function is the central and most important issue in Mesh free methods. The challenge is how to create shape functions using only nodes scattered arbitrarily in a domain without any predefined mesh to provide connectivity of the nodes.

Development of more effective methods for constructing shape functions is thus one of the hottest areas of research in the area of Mesh free methods. A good method of shape function construction should satisfy the following basic requirements:

a. The nodal distribution can be arbitrary within reason, at least more flexible than that in the finite element method (FEM) (arbitrary nodal distribution).

b. The algorithm must be stable (stability).

c. The shape function constructed should satisfy a certain order of consistency (consistency).

d. The domain for field variable approximation/interpolation (termed the support

domain or influence domain or smoothing domain) should be small compared with the entire problem domain (compact support).

e. The algorithm should be computationally efficient. It should be of the same order of complexity as that of FEM (efficiency).

f. Ideally, the shape function should possess the Kronecker delta function property (delta function property).

g. Ideally, the field approximation using the shape function should be compatible throughout the problem domain (compatibility).

Satisfaction of the above requirements ensures both easy implementation of the Mesh free method and accuracy of the numerical solutions. The first requirement is obvious. The stability (the ii requirement) of an algorithm should always be checked, because there could be uncertainties caused by the arbitrariness in the distribution of nodes. The consistency condition (requirement iii) is essential for the convergence of the numerical results, when the nodal spacing is reduced. Satisfaction of the compact condition (requirement iv) leads to a banded system matrix that can be handled with good computational efficiency. The domain for field variable approximation/interpolation should be kept as small as possible to ensure a narrow bandwidth in the discretized system matrices. Requirement (v) prevents unacceptably expensive shape function constructions, because a too costly procedure will eventually become impractical, no matter how good it is. The (vi) requirement eases imposition of essential boundary conditions, to place a limit on the extra effort needed for handling the essential boundary conditions. This requirement is not rigid because one can use special measures to impose essential boundary conditions, of course, at additional expense. The last requirement removes the need for enforcing compatibility in using the (global) Galerkin weak forms for establishing the discrete equation systems. The compatibility requirement is unnecessary if the local weight residual weak form is employed. It requires only the reproduction or consistency of the shape function to

achieve convergence of the solution.

A number of ways to construct shape functions have been proposed. The classification given by G. R. Liu [18] of these methods into three major categories:

a. Finite integral representation methods, which include:

   (1) Smoothed particle hydrodynamics (SPH) method

   (2) Reproducing kernel particle method (RKPM)

   (3) General kernel reproduction method (GKR)

b. Finite series representation methods, which include:

   (1) Moving least squares (MLS) methods:

      i. MLS approximation

      ii. Modified MLS approximation

   (2) Point interpolation methods (PIM):

      i. Polynomial PIM

      ii. Radial PIMs

   (3) Partition of unit (PU) methods:

      i. Partition of unity finite element (PUFE)

      ii. hp-clouds

   (4) Finite element methods:

      i. Element-based interpolations

c. Finite differential representation methods, which include:

   (1) Finite difference method (regular grids)

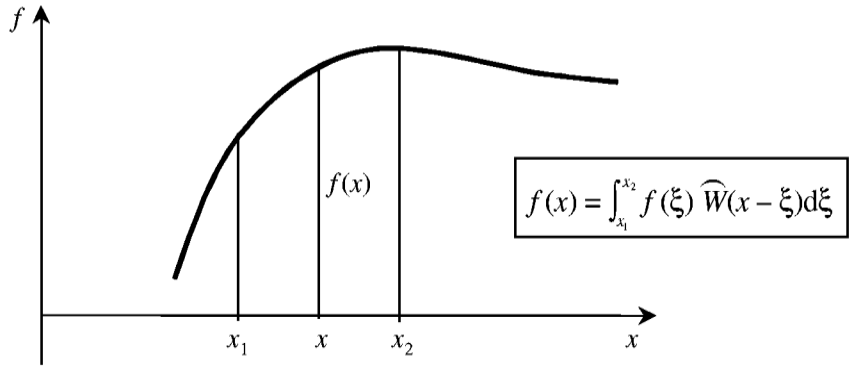
   (2) Finite point method (irregular grids)

Finite integral representation methods are relatively young, but have found a special place in Mesh free methods with the development of smoothed particle hydrodynamics (SPH). The function is represented using its information in a local domain (smoothing domain or influence domain) via an integral form, as illustrated in Fig. 4.1.
Consistency is achieved by properly choosing the weight function. Finite series representation methods have a long history of development. They are well developed in FEM, and are very active now in the area of Mesh free methods. Consistency is ensured by the use of the basis functions. The inclusion of special terms in the basis can also improve the accuracy of the results for certain classes of problems. Finite difference representation methods have also been used for a long time. Convergence of the representation is ensured via the theory of the Taylor series. Finite difference representation methods are usually used for establishing system equations based on strong formulation, where one may, but usually do not, construct shape functions. The following sections detail the first two types of methods, which are widely used for creating shape functions for Mesh free methods.
Consistency is the capability of the field function approximation method to reproduce the fields of lowest orders of complete polynomials at any point in the problem domain.

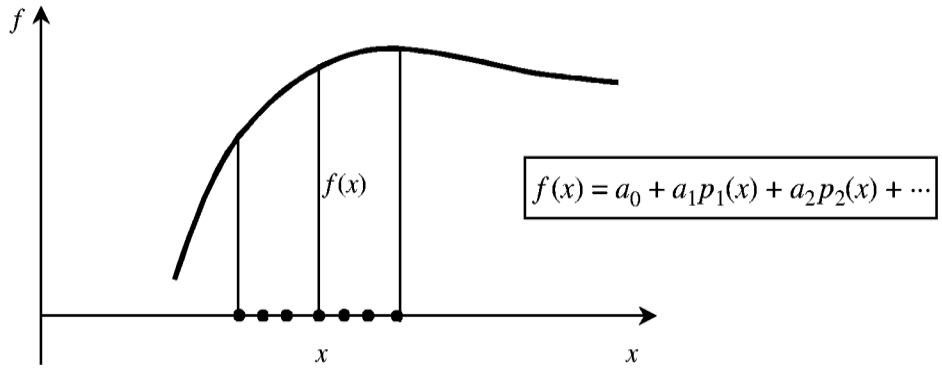
If the method can reproduce polynomials of up to the kth order, the method is said to have kth-order consistency. Compatibility refers to the continuity of the approximation on the boundaries between sub-domains, based on which shape the functions are constructed. Both consistency and compatibility affect the accuracy and convergence of the numerical results. As will be seen later, the MLS shape functions are both consistent and compatible.

4.2 Choice of Weight Function

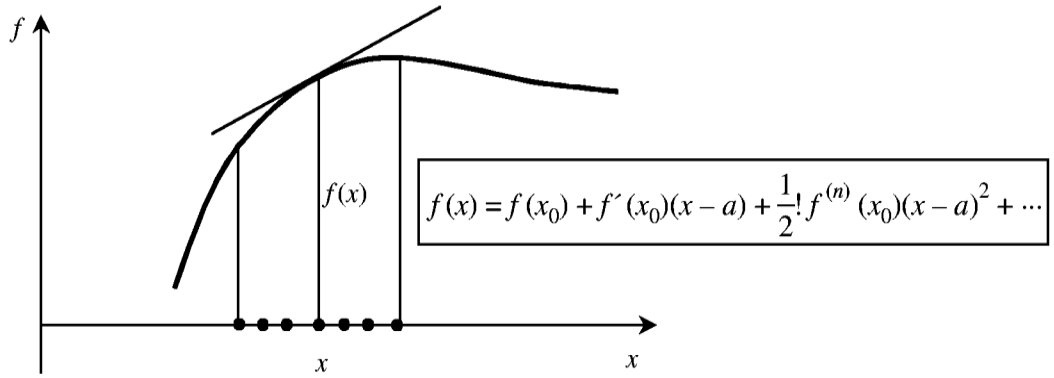
The weight function $\hat{w}_I(x) \equiv \hat{W}(x - x_I)$ play an important role in the performance of the method. They should be constructed so that they are positive and that a



(a)



(b)



(c)

Figure 4.1: Methods of function representation at x using the information in its vicinity. (a) Finite integral representation. $\hat{W}$: weight or smoothing function. (b) Finite series representation. pi(x) are basis functions. (c) Finite differential representation, where derivatives of function are used.

unique solution $a(x)$ is guaranteed; they should be relatively large for the x_I close to x, and relatively small for the more distant x_I; in other words, they should decrease in magnitude as the distance from x to x_I increases. Therefore, weight functions which depend only on the distance between two points as follows:

$$\hat{W}(x - x_I) = \hat{W}_I(d) \tag{4.1}$$

Where $d = \|x - x_I\|$ is the distance between the two points x and x_I. more specifically the weight functions of the following form is considered:

$$\hat{W}_I \equiv \hat{W}_I(d^{2k}) \tag{4.2}$$

Where $\hat{W}_I\ (d^{2k})$ with respect to x or y in two dimensions exist at the data point x_I. To begin, we consider the derivatives of weight function $\hat{W}_I(d^{2k})$ with respect to x:

$$\frac{\partial \hat{W}_I}{\partial x} = 2kd^{2k-1}\frac{\partial \hat{W}_I}{\partial d}\frac{\partial d}{\partial x} = 2k(x - x_I)d^{2k-2}\frac{\partial \hat{W}_I}{\partial d}\ in2D \tag{4.3}$$

Note that the limit of $(x - x_I)/d$ as x approaches x_I does not exist. Therefore, the above derivatives exists only if $k > 1/2$. For the second derivative of the weight function $\hat{W}_I\ (d^{2k})$ with respect to x, Belytchko et. al.[28].

The different types of weight function used for the EFG analysis are as follows:

The cubic spline weight function (W1):

$$\hat{W}\ (x - x_I) \equiv \hat{W}(\overline{d}) = \begin{array}{ll} \frac{2}{3} - 4\overline{d}^2 + 4\overline{d}^3 & for\ \overline{d} \leq \frac{1}{2} \\ \frac{4}{3} - 4\overline{d} + 4\overline{d}^2 - \frac{4}{3}\overline{d}^3 & for \frac{1}{2} \leq \overline{d} \leq 1 \\ 0 \qquad for\ \overline{d}\ > 1 & \end{array} \tag{4.4}$$

The quartic spline weight function (W2):

$$\hat{W}\ (x - x_I) \equiv \hat{W}(\overline{d}) = \begin{array}{ll} 1 - 6\ \overline{d}^2 + 8\overline{d}^3 - 3\overline{d}^4 & for\ \overline{d} \leq \frac{1}{2} \\ 0 & for\ \overline{d}\ > 1 \end{array} \tag{4.5}$$

The exponential weight function (W3):

$$\hat{W}\ (x - x_I) \equiv \hat{W}(\overline{d}) = \begin{array}{ll} e^{-(\overline{d}/\alpha)^2} & for\ \overline{d} \leq \frac{1}{2} \\ 0 & for\ \overline{d}\ > 1 \end{array} \tag{4.6}$$

In equation 4.4 to 4.6,

$$\overline{d} = \frac{|x - x_I|}{d_w} = \frac{d}{d_w} \tag{4.7}$$

where d_w is directly related to the size of the support domain. It defines the dimension of the domain where $\hat{W} \neq 0$. In general, d_w can be different from point to point. By following a general procedure for constructing weight (smoothing) functions (Liu, G. R. et al.,2002a), a new quartic weight (smoothing) function is constructed (W4).

$$\hat{W}\ (x - x_I) \equiv \hat{W}\left(\overline{d}\right) = \begin{array}{ll} \frac{2}{3} - \frac{9}{32}\ \overline{d}^2 + \frac{19}{192}\overline{d}^3 - \frac{5}{512}\overline{d}^4 & for\ \overline{d} \leq 1 \\ 0 & for\ \overline{d}\ > 1 \end{array} \tag{4.8}$$

Fig. 4.2 plots all four weight functions given by eqns. 4.4 to 4.6 and 4.7. From the plot it is seen that the weight function W4 behave same as weight function W1. Fig. 4.2.

4.3 Moving Least Squares Approximation

Moving least squares (MLS), originated by mathematicians for data fitting and surface construction, is often termed local regression and loss (Lancaster and Salkauskas, 1981; Cleveland, 1993). It can be categorized as a method of finite series representation of functions. An excellent description of the MLS method can be found in a paper by Lancaster and Salkauskas (1981). The MLS method is now a widely used alternative for constructing Mesh free shape functions for approximation. Nayroles et

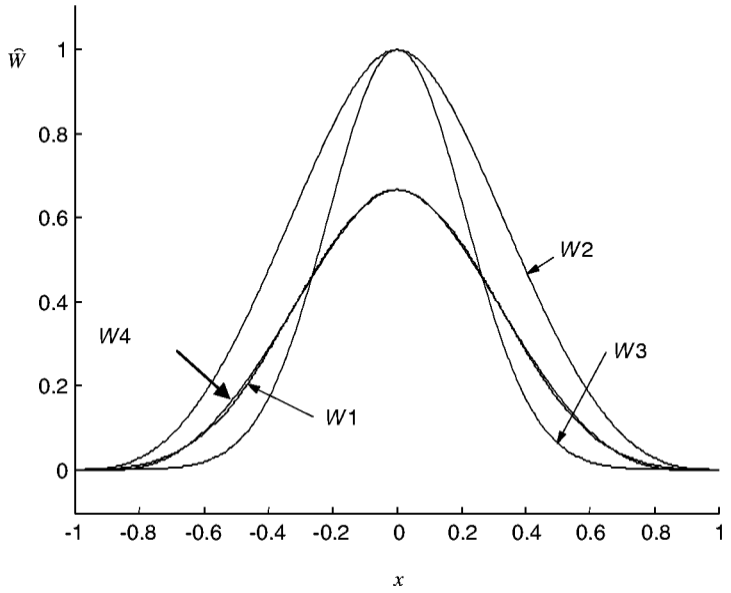


Figure 4.2: Weight functions. W1: cubic spline weight function; W2: quartic spline weight function; W3: exponential weight function ($c\alpha = 0.3$); W4: new quartic weight function

al. (1992) were the first to use MLS approximation to construct shape functions for their diffuse element method (DEM) for mechanics problems. DEM was modified by Belytschko et al. (1994b), who named it the element free Galerkin (EFG) method, where the MLS approximation is also employed.

The MLS approximation has two major features that make it popular:

a. The approximated field function is continuous and smooth in the entire problem domain; and

b. It is capable of producing an approximation with the desired order of consistency. The procedure of constructing shape functions for Mesh free methods using MLS approximation is detailed in following section.

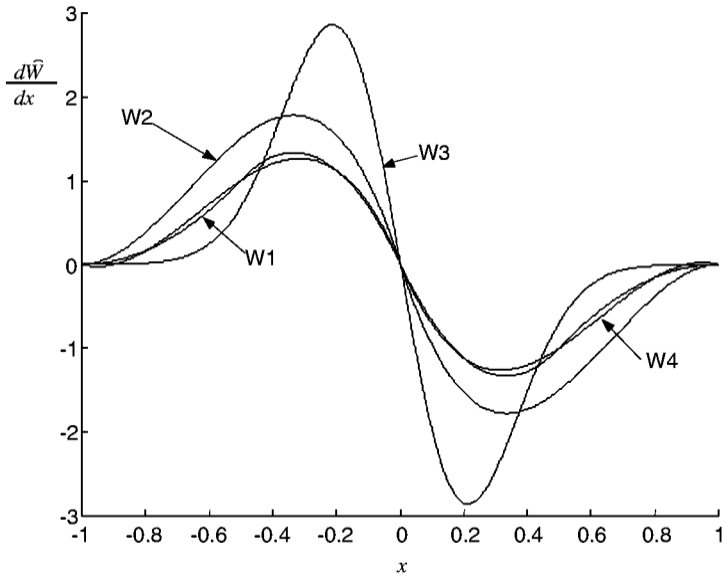


Figure 4.3: The first derivative of weight functions. W1: cubic spline weight function; W2: quartic spline weight function; W3: exponential weight function (a = 0.3); W4: new quartic weight function

4.4 MLS Procedure

Let u(x) be the function of the field variable defined in the domain Ω. The approximation of $u(x)$ at point x is denoted $u^h(x)$. MLS approximation first writes the field function in the form:

$$\mathrm{u^h\,(x) = \sum_{j}^{m} p_j\,(x)\,a_j\,(x) \equiv p^T\,(x)\,a\,(x)} \tag{4.9}$$

Where m is the number of terms of monomials (polynomial basis), and a(x) is a vector of coefficients given by,

$$\mathrm{a^T}\,(x) = \{a_0\,(x)\ \ a_1\,(x)\,,\ \ldots,\ a_m\,(x)\} \tag{4.10}$$

Where functions of x, In eqn. 4.9, p(x) is a vector of basis functions that consists most often of monomials of the lowest orders to ensure minimum completeness. Enhancement functions can, however, be added to achieve better efficiency or to produce stress fields of special characteristics, such as singularity at the crack tip and stress discontinuity at interfaces of different types of materials. Here discussion is for the use of the pure polynomial basis. In 1D space, a complete polynomial basis of order m is given by

$$\mathrm{p}^{\mathrm{T}}(x) = \{p_0(x), p_1(x), \ldots\ldots\ p_m(x)\} = \left\{1, x, x^2, \ldots, x^m\right\} \tag{4.11}$$

and in 2D space,

$$\mathrm{p}^{\mathrm{T}}(x) = p^T(x, y) = \left\{1, x, y, x^2, y^2, \ldots, x^m, y^m\right\} \tag{4.12}$$

In this case, the Pascal triangle can be utilized to build $p^T(x)$, and the number of nodes in the support domain can be chosen accordingly.

In 3D space,

$$\mathrm{p}^{\mathrm{T}}(x) = p^T(x, y, z) = \left\{1, x, y, z, xy, yz, xz, x^2, y^2, z^2, \ldots, x^m, y^m, z^m\right\} \tag{4.13}$$

In this case, the Pascal pyramid can be employed to build $p^T(x)$. The vector of coefficients a(x) in eqn. 4.9 is determined using the function values at a set of nodes that are included in the support domain of x. A support domain of a point x determines the number of nodes that are used locally to approximate the function value at x.

Given a set of n nodal values for the field function $u1, u2, ..., un$, at n nodes $x1, x2, ..., xn$ that are in the support domain, eqn. 4.9 is then used to calculate the approximated values of the field function at these nodes:

$$u^h(x,\ x_I) =\ p^h(x_I)\, a(x), \qquad I = 1,\ 2,\ \ldots,\ n \tag{4.14}$$

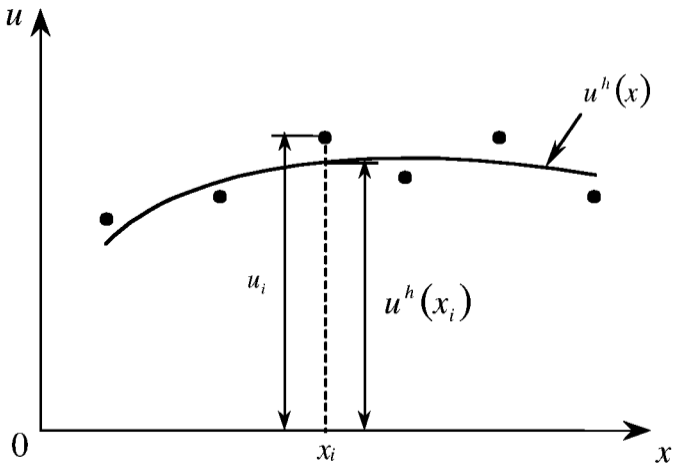


Figure 4.4: The approximation function $u^h(x)$ and the nodal parameters u_i in the MLS approximation

Note that a(x) here is an arbitrary function of x. A functional of weighted residual is constructed using the approximated values of the field function and the nodal parameters, u1 = u(xs), that are shown in fig. 4.4, i.e.,

$$\mathrm{J} = \sum_{I}^{n} \hat{W}\left(x - x_I\right)\left[u^h\left(x,\ x_I\right) - u(x_I)\right]^2 \tag{4.15}$$

$$\mathrm{J} = \sum_{I}^{n} \hat{W}\left(x - x_I\right)\left[p^T\left(\ x_I\right) a\left(x\right) - u_I\right]^2 \tag{4.16}$$

Where $\hat{W}(x - x_I)$ is a weight function, and u_I is the nodal parameter of the field variable at node I.

The stationarity of J in equation with respect to *a(x)* leads to the following linear relation between *a(x)* and *uI*.

$$A\left(x\right) a\left(x\right) = B\left(x\right) u \tag{4.17}$$

$$a\left(x\right) = A^{-1}\left(x\right) B\left(x\right) u \tag{4.18}$$

Where A(x) and B(x) are matrices defined by

$$A(x) = \sum_{I}^{n} \hat{W}_I(x) p^T(x_I) p(x_I), \ldots \hat{W}_I(x) \equiv \hat{W}(x - x_I) \tag{4.19}$$

$$\mathrm{B}\,(x) = \{w_1(x) p\,(x_1)\,, w_2(x) p\,(x_2)\,, \ldots\ldots\ w_n(x) p\,(x_n)\} \tag{4.20}$$

$$u^T = \{u_1, u_2, \ldots,\ u_n\} \tag{4.21}$$

Hence,

$$\mathrm{u^h}\,(x) = \sum_{I}^{n} \sum_{j}^{m} p_j\,(x)\,(A^{-1}\,(x)\,B\,(x))_{Ij} \equiv \sum_{I}^{n} \emptyset_I\,(x)\,u_I \tag{4.22}$$

Where the shape function $\emptyset_I\,(x)$is defined by:

$$\emptyset_I = \sum_{j}^{m} p_j (A^{-1}\,(x)\,B\,(x))_{jI} \tag{4.23}$$

The partial derivatives of $\emptyset_I\,(x)$ can be obtained as follows:

$$\emptyset_{I,i} = \sum_{j}^{m} \left\{ p_{j,i} (A^{-1}\,(x)\,B\,(x))_{jI} +\ p_j (A^{-1}_{,i} B + A^{-1} B_{,i})_{jI} \right\} \tag{4.24}$$

where

$$A^{-1}_{,i} = -A^{-1} A_{,i} A^{-1} \tag{4.25}$$

And the index following a comma is a spatial derivative. It should be noted that the approximation in eqn. 4.22 is no longer a polynomial even if the basis function p(x) are polynomials. However, if $u(x)$ is a polynomial, it is reproduced exactly by $u^h(x)$; see Nayroles et al.

4.5 1D shape functions creation

A plot of a typical 1D MLS weight function and shape function is given in fig. 4.5. The shape function is for the node at x = 0 and is obtained using five nodes evenly distributed in the support domain of [-1, 1].
The quartic spline weight function (W2) is used. It can be seen that the MLS shape function attains a maximum value that is considerably less than 1. For this plot, the quartic weight function eqn. 4.5 is used with dw = 0.45.

4.6 2D shape functions creation

Mesh free shape functions are constructed in a domain of (x, y) [-2, 2] x [-2, 2] using 5 x 5 evenly distributed nodes in the domain. Fig. 4.6 shows the MLS shape function. It is clear that the MLS shape functions do not satisfy the Kronecker delta function. Fig. 4.7 shows the first derivative of MLS shape function w.r.t. x and y.

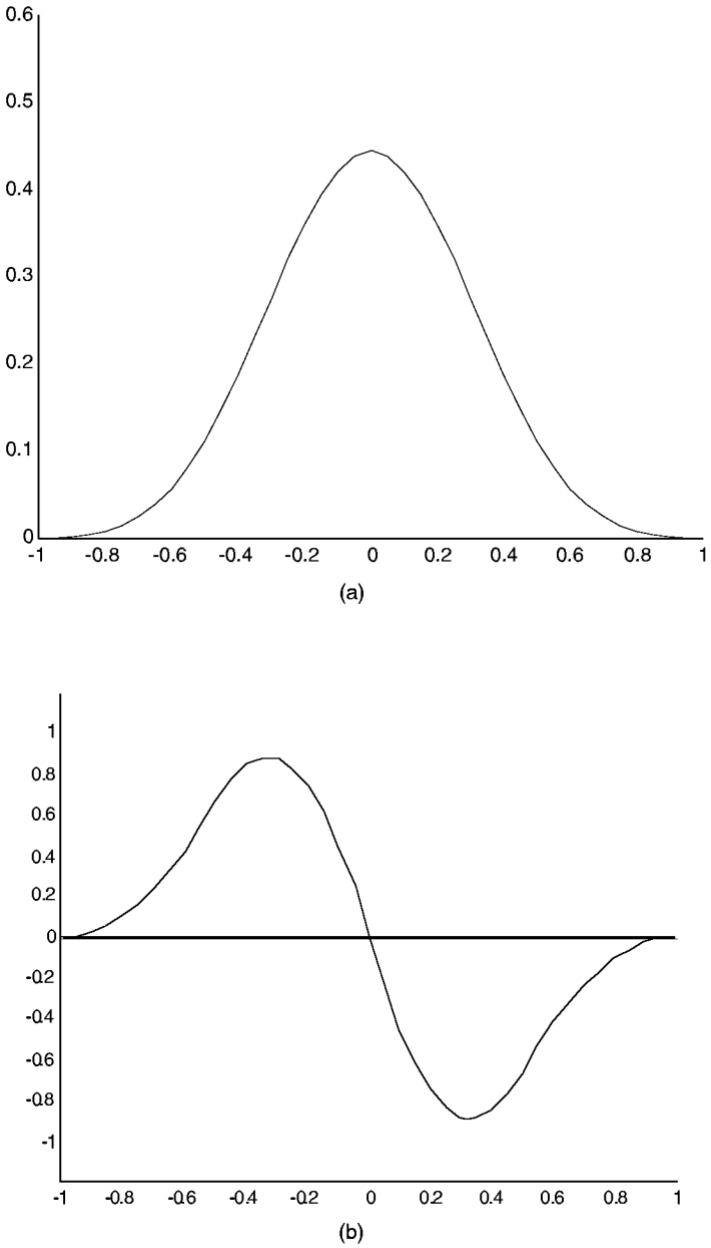


Figure 4.5: MLS shape function in 1D space for the node at x = 0 obtained using five nodes evenly distributed in the support domain of [-1, 1]. Quartic spline weight function (W2) is used. (a) MLS shape function; (b) derivative of the shape function. Note that the MLS shape function does not possess the Kronecker delta function property.

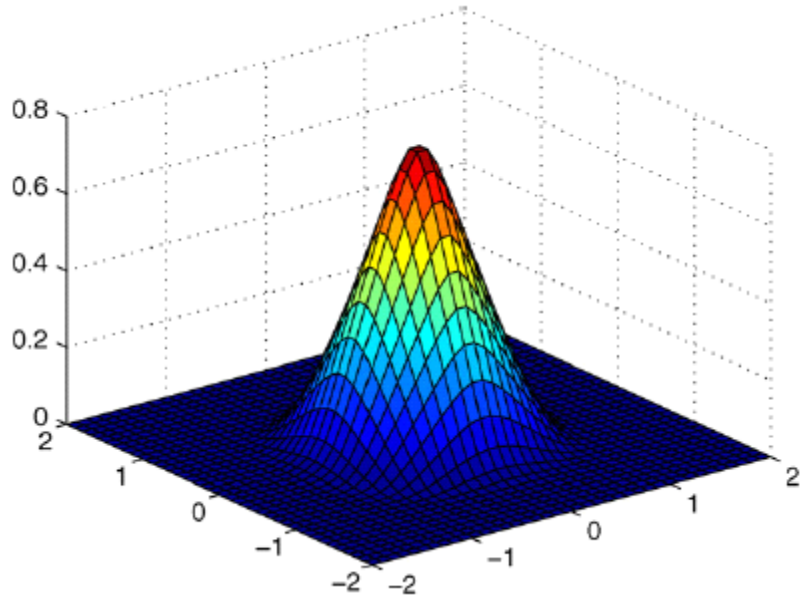


Figure 4.6: MLS shape function for 2D

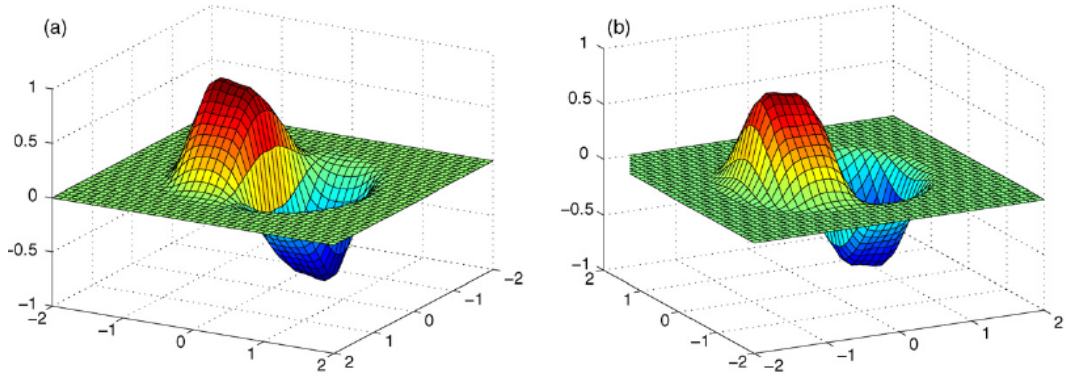


Figure 4.7: First derivative of MLS shape function in x and y direction

Chapter 5

ELEMENT FREE GALERKIN METHOD

5.1 Introduction

The element free Galerkin (EFG) method is an Mesh free method developed by Belytschko et al. (1994b)[28] based on the diffuse elements method (DEM) originated by Nayroles et al. (1992). The major features of the DEM and the EFG method are as follows:

a. Moving least square (MLS) approximation is employed for the construction of the shape function.

b. Galerkin weak form is employed to develop the discretized system equation.

c. Cells of the background mesh for integration are required to carry out the integration to calculate system matrices.

This chapter presents a detailed procedure that leads to the EFG method. Detailed formulation and equations are provided. Issues related to background integration, are examined. A typical benchmark problem of a cantilever beam is considered to illustrate the relationship between the density of the field nodes and the density of

the global background mesh, as well as the number of integration sample points. The findings and remarks on the background integration are applicable to any Mesh free method that requires background integration.

Applications of the EFG method are presented for solving a number of structural engineering problems including linear and nonlinear problems.

It may be noted that the EFG is conforming due to the use of MLS shape functions that are consistent and compatible and the use of the constrained Galerkin approach to impose the essential boundary condition.

5.2 EFG formulation

A two-dimensional (2D) linear solid mechanics problem is used to present the procedure of the EFG method in formulating discretized system equations. The stress components σ_{ij} must satisfy the differential equations of equilibrium in a domain, Ω, of a body under consideration

$$\sigma_{ij,j} + F_i = 0 \qquad \textit{Equilibrium equation in problem domain } \Omega \tag{5.1}$$

where F_i are the components of body force acting on the object. The field equation are to be solved given certain boundary conditions as

$$u_i = \overline{u}_i \qquad \textit{Boundary condition essential boundary } \Gamma_{\mathrm{u}} \tag{5.2}$$

which gives constraints to the field variable of displacement. The natural boundary conditions are given by

$$t_i = \bar{t}_i \qquad \textit{Boundary condition essential boundary } \Gamma_{\mathrm{t}} \tag{5.3}$$

where the traction on a surface with normal nj are given as $t_i = \sigma_{ij} n_j$, and $\overline{u}_i$ and $\bar{t}_i$ are the prescribed values of displacements and tractions on boundaries Γ_{u} and Γ_{t},

respectively. $\Gamma_u \cup \Gamma_t = \Gamma$ is the boundary of the object domain Ω. The constrained Galerkin weak form with Lagrange multipliers for the problem stated by eqns. 5.2 and 5.3 can be given by

$$\int_\Omega \sigma_{ij}\delta v_{i,j}d\Omega - \int_\Omega \delta F_i \delta v_i d\Omega - \int_{\Gamma_t} \bar{t}_i \delta v_i d\Gamma - \int_{\Gamma_u} (u_i - \overline{u}_i)\delta\lambda_i d\Gamma - \int_{\Gamma_u} \lambda_i \delta v_i d\Gamma = 0 \tag{5.4}$$

The MLS approximation described in previous section is now used to express both the trial and test functions at any point of interest x using the nodes in the support domain of the point x. For displacement comment u,

$$\mathrm{u}^{\mathrm{h}}(x) = \sum_I^n \emptyset_I(x)\, u_I \tag{5.5}$$

where n is the number of nodes used in the support domain of the point at x for constructing the MLS shape function $\emptyset_I(x)$. The procedure of constructing $\emptyset_I(x)$ is detailed in Section 4.4, and the formulation of $\emptyset_I(x)$ is given by eqn.4.23.
The Lagrange multiplier λ and its variation $\delta\,\lambda$ are expressed as

$$\begin{aligned} \lambda(x) &= \textstyle\sum_I^n N_I(x)\,\lambda_I, & x &\in \Gamma_u \\ \delta\lambda(x) &= \textstyle\sum_I^n N_I(x)\,\delta\lambda_I, & x &\in \Gamma_u \end{aligned} \tag{5.6}$$

The displacement field, *u= [ux,uy],* consisting of the x and y displacement components for a two-dimensional analysis, is expressed in terms of nodal displacement using eqn. 5.5 as,

$$\left\{ \begin{array}{c} u_x \\ u_y \end{array} \right\} = \left[\begin{array}{ccccccc} \emptyset_1 & 0 & \emptyset_2 & 0 & \dots & \emptyset_n & 0 \\ 0 & \emptyset_1 & 0 & \emptyset_2 & \dots & 0 & \emptyset_n \end{array} \right] \left\{ \begin{array}{c} u_x^1 \\ u_y^1 \\ u_x^2 \\ u_y^2 \\ \vdots \\ u_x^n \\ u_y^n \end{array} \right\} = [\Phi] \{u\} \tag{5.7}$$

The two dimensional strain fields is given by small displacement analysis,

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{array} \right\} = \left\{ \begin{array}{c} \frac{\partial u_x}{\partial x} \\ \frac{\partial u_y}{\partial y} \\ \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \end{array} \right\} \tag{5.8}$$

Where $\gamma_{xy} = 2\varepsilon_{xy}$ is referred to as engineering shear strain, substituting in eqn.5.7 from eqn.5.6, the strain field is written as,

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{array} \right\} = \left[\begin{array}{ccccccc} \emptyset_{1,x} & 0 & \emptyset_{2,x} & 0 & \dots & \emptyset_{n,x} & 0 \\ 0 & \emptyset_{1,y} & 0 & \emptyset_{2,y} & \dots & 0 & \emptyset_{n,y} \\ \emptyset_{1,y} & \emptyset_{1,x} & \emptyset_{2,y} & \emptyset_{2,x} & \dots & \emptyset_{n,y} & \emptyset_{n,x} \end{array} \right] \left\{ \begin{array}{c} u_x^1 \\ u_y^1 \\ u_x^2 \\ u_y^2 \\ \vdots \\ u_x^n \\ u_y^n \end{array} \right\} \tag{5.9}$$

which is expressed in shorthand matrix notation as

$$\{\varepsilon\} = [B] \{u\} \tag{5.10}$$

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \qquad \{u\} = \begin{Bmatrix} u_x^1 \\ u_y^1 \\ u_x^2 \\ u_y^2 \\ \vdots \\ u_x^n \\ u_y^n \end{Bmatrix} \tag{5.11}$$

$$[B] = \begin{bmatrix} [B_1] & [B_2] & \dots & [B_n] \end{bmatrix} \tag{5.12}$$

$$[B_I] = \begin{bmatrix} \emptyset_{I,x} & 0 \\ 0 & \emptyset_{I,y} \\ \emptyset_{I,y} & \emptyset_{I,x} \end{bmatrix} \tag{5.13}$$

The Lagrange multiplier field is expressed in the matrix notation using eqn.5.6 as

$$\begin{Bmatrix} \lambda_x \\ \lambda_y \end{Bmatrix} = \begin{bmatrix} \psi_1 & 0 & \psi_2 & 0 & \dots & \psi_m & 0 \\ 0 & \psi_1 & 0 & \psi_2 & \dots & 0 & \psi_m \end{bmatrix} \begin{Bmatrix} \lambda_x^1 \\ \lambda_y^1 \\ \lambda_x^2 \\ \lambda_y^2 \\ \vdots \\ \lambda_x^m \\ \lambda_y^m \end{Bmatrix} = [N]\{\lambda\} \tag{5.14}$$

Where

$$[N] = \begin{bmatrix} [N_1] & [N_2] & \dots & [N_m] \end{bmatrix} \tag{5.15}$$

$$[N_J] = \begin{bmatrix} \psi_J & 0 \\ 0 & \psi_J \end{bmatrix} \tag{5.16}$$

$$\{\lambda\} = \begin{Bmatrix} \lambda_x^1 \\ \lambda_y^1 \\ \lambda_x^2 \\ \lambda_y^2 \\ \vdots \\ \lambda_x^m \\ \lambda_y^m \end{Bmatrix} \tag{5.17}$$

In eqn.5.17, m is the number of boundary nodes on which the essential boundary conditions are prescribed. Similarly, the variational quantities are expressed using shape function-based descriptions as

$$\{\delta\varepsilon\} = [B]\{\delta v\} \tag{5.18}$$

$$\begin{Bmatrix} \delta\lambda_x \\ \delta\lambda_y \end{Bmatrix} = [N]\{\delta\lambda\} \tag{5.19}$$

With

$$\{\delta v\} = \begin{Bmatrix} \delta v_x^1 \\ \delta v_y^1 \\ \delta v_x^2 \\ \delta v_y^2 \\ \vdots \\ \delta v_x^n \\ \delta v_y^n \end{Bmatrix} \qquad \{\delta\lambda\} = \begin{Bmatrix} \delta\lambda_x^1 \\ \delta\lambda_y^1 \\ \delta\lambda_x^2 \\ \delta\lambda_y^2 \\ \vdots \\ \delta\lambda_x^m \\ \delta\lambda_y^m \end{Bmatrix} \tag{5.20}$$

The stress-strain relationship can be written in the matrix form as

$$\{\sigma\} = [D]\{\varepsilon\} \tag{5.21}$$

Where for two dimensional plane stress analysis,

$$D = \frac{E}{1-\mu^2}\begin{bmatrix} 1 & \mu & 0 \\ \mu & 1 & 0 \\ 0 & 0 & \frac{1-\mu}{2} \end{bmatrix} \tag{5.22}$$

where E and μ are the Young's modulus and Poisson's ratio respectively.
For two dimensions the weak form in eqn.5.5 is now expressed in matrix notation as

$$\begin{aligned} &\int_\Omega \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix}^T \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} d\Omega - \int_\Omega \begin{Bmatrix} \delta v_x \\ \delta v_y \end{Bmatrix}^T \begin{Bmatrix} F_x \\ F_y \end{Bmatrix} d\Omega \\ &- \int_{\Gamma_t} \begin{Bmatrix} \delta v_x \\ \delta v_y \end{Bmatrix}^T \begin{Bmatrix} \bar{t}_x \\ \bar{t}_y \end{Bmatrix} d\Gamma - \int_{\Gamma_u} \begin{Bmatrix} \delta \lambda_x \\ \delta \lambda_y \end{Bmatrix}^T \begin{Bmatrix} u_x \\ u_y \end{Bmatrix} d\Gamma - \int_{\Gamma_u} \begin{Bmatrix} \delta v_x \\ \delta v_y \end{Bmatrix}^T \begin{Bmatrix} \lambda_x \\ \delta \lambda_y \end{Bmatrix} d\Gamma \\ &+ \int_{\Gamma_u} \begin{Bmatrix} \delta \lambda_x \\ \delta \lambda_y \end{Bmatrix}^T \begin{Bmatrix} \overline{u}_x \\ \overline{u}_y \end{Bmatrix} d\Gamma = 0 \end{aligned} \tag{5.23}$$

A few key terms in eqn.5.23 are expanded explicitly to show the process followed to obtain the system of EFG matrix equations. The first term in eqn.5.23 is written as

$$\int_\Omega \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix}^T \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} d\Omega = \int_\Omega \{\delta v\}^T [B]^T [D] [B] \{u\} \, d\Omega \tag{5.24}$$

where substitutions have been made from relations in eqns.5.18, 5.10, and 5.21 in obtaining the right hand side of eqn.5.24. The right-hand side of eqn.5.24 is written as

$$\int_\Omega \{\delta v\}^T [B]^T [D] [B] \{u\} \, d\Omega = \{\delta v\}^T [K] \{u\} \tag{5.25}$$

where stiffness matrix $[K]$ is

$$\begin{gathered} K_{IJ} = \int_\Omega B_I^T D B_J d\Omega \\ I = 1, 2, \ldots, n \qquad\qquad J = 1, 2, \ldots., m \end{gathered} \tag{5.26}$$

The sub-matrix $[KIJ]$ in eqn.5.26 pre multiplies the displacement vector $\{uJ\}$ in eqn.5.25. The second term in eqn.5.23 is now expanded as

$$\int_\Omega \left\{ \begin{array}{c} \delta v_x \\ \delta v_y \end{array} \right\}^T \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega = \int_\Omega \{\delta v\}^T [] \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega \tag{5.27}$$

Where the variational displacement field $\delta v = [\delta v_x, \delta v_y]$, is expressed similar to the expression for displacement field in eqn.5.7. The right-hand side of the eqn.5.27 may be written as

$$\int_\Omega \{\delta v\}^T []^T \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega = \{\delta v\}^T \{F\} \tag{5.28}$$

Where the force vector $\{F\}$ is written as

$$\{F\} = \int_\Omega []^T \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega \tag{5.29}$$

The integrand in eqn.5.29 is further expanded by substituting for [] from eqn.5.5 to obtain

$$[]^T \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} = \left\{ \begin{array}{cc} \emptyset_1 & 0 \\ 0 & \emptyset_1 \\ \emptyset_2 & 0 \\ 0 & \emptyset_2 \\ \vdots & \vdots \\ \emptyset_n & 0 \\ 0 & \emptyset_n \end{array} \right\} \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} = \left\{ \begin{array}{cc} \emptyset_1 & \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} \\ \emptyset_2 & \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} \\ & \vdots \\ \emptyset_n & \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} \end{array} \right\} \tag{5.30}$$

The force vector $\{F\}$ is seen from eqn.5.30 to be composed of sub vectors $\left\{f_J^{\Omega}\right\}$, where

$$\left\{f_J^{\Omega}\right\} = \int_{\Omega} \emptyset_J \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega \qquad\qquad J = 1, 2, ..., n \tag{5.31}$$

The remaining terms in eqn.5.23 may be expanded similarly to obtain

$$\{\delta v\}^T \left[[K_{IJ}] \{u_J\} - \left\{f_J^{\Omega}\right\} - \left\{f_J^{\Gamma}\right\} + [G_{IJ}] \{\lambda_J\} \right] + \{\delta\lambda\}^T \left[[G_{IJ}]^T \{u_J\} - \{q_J\} \right] = 0 \tag{5.32}$$

Since $\{\delta v\}$ and $\{\delta\lambda\}$ are arbitrary variations, obtain two matrix equations by equating the terms that post multiply each of them separately to $\{0\}$ as

$$[K_{IJ}] \{u_J\} - \left\{f_J^{\Omega}\right\} - \left\{f_J^{\Gamma}\right\} + [G_{IJ}] \{\lambda_J\} = \{0\} \tag{5.33}$$

$$[G_{IJ}]^T \{u_J\} - \{q_J\} = \{0\} \tag{5.34}$$

Writing this equations in matrix form,

$$\left[\begin{array}{cc} [K_{IJ}] & [G_{IJ}] \\ [G_{IJ}]^T & [0] \end{array} \right] \left\{ \begin{array}{c} \{u_J\} \\ \{\lambda_J\} \end{array} \right\} = \left\{ \begin{array}{c} \{f_J\} \\ \{q_J\} \end{array} \right\} \tag{5.35}$$

with

$$\mathrm{f}_J = \int_{\Omega} {\mathbf{\Phi}_J}^T \left\{ \begin{array}{c} F_x \\ F_y \end{array} \right\} d\Omega - \int_{\Gamma_t} \delta {\mathbf{\Phi_J}}^T \left\{ \begin{array}{c} \bar{t}_x \\ \bar{t}_y \end{array} \right\} d\Gamma \tag{5.36}$$

$$G_{IJ} = -\int_{\Gamma_u} N_I^T \mathbf{\Phi}_j d\Gamma \tag{5.37}$$

$$q_J = -\int_{\Gamma_u} \mathrm{N}_J^T \left\{ \begin{array}{c} \overline{u}_x \\ \overline{u}_y \end{array} \right\} d\Gamma \tag{5.38}$$

Finally, the contributions of the form of eqn.5.35 for each nodal point is assembled

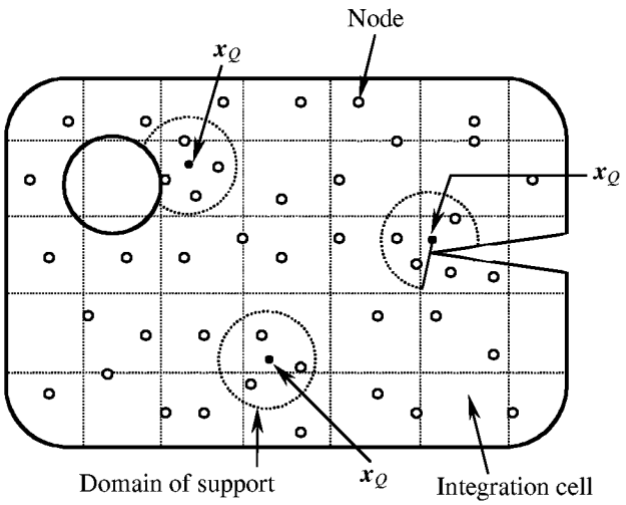


Figure 5.1: Background mesh

to obtain a global system of equation as

$$\begin{bmatrix} [K] & [G] \\ [G^T] & [0] \end{bmatrix} \begin{Bmatrix} \{u\} \\ \{\lambda\} \end{Bmatrix} = \begin{Bmatrix} \{f\} \\ \{q\} \end{Bmatrix} \tag{5.39}$$

In eqn.5.39, $\{u\}$ and $\{\lambda\}$ are the vectors of unknown nodal displacements and Lagrange multipliers, respectively. The system of eqn. 5.39 may be solved using standard linear system solvers to obtain these quantities.

5.3 Flowchart

The solution procedure of the EFG method is similar to that for FEM. The geometry of the problem domain is first modeled, and a set of nodes is generated to represent the problem domain, as shown in Fig. 5.3. The system matrices are assembled via two loops. The outer loop is for all the cells of the background mesh, and the inner loop is for all the Gauss quadrature points in a cell. The flowchart of the algorithm for stress analysis using the EFG method is presented in Fig. 5.2.

5.4 Background integration

In either FEM or EFG, numerical integration is a time-consuming process required for the computation of the stiffness matrix that is established based on the variation method. In FEM, the integration mesh is the same as the element mesh. To obtain accurate results, the element mesh must be sufficiently fine and a sufficient number of integration points have to be used. In EFG, however, the background mesh is required only in performing the integration of computing the stiffness matrix. Therefore, a background mesh of proper density needs to be designed to obtain an approximate solution of desired accuracy. However, this can only be done after performing a detailed investigation to reveal the relationship between the density of the field nodes and the density of the background mesh. The first thing that needs to be addressed is the minimum number of integration points when numerical integration is adopted.

Zienkiewicz (1989) has shown for FEM that, if the number of independent relations provided by all integration points is less than the number of unknowns (displacements at all points in the element), the stiffness matrix K must be singular. This concept should also be applicable, in principle, to EFG. For a 2D problem, the number of unknown variables Nu should be

$$N_u = 2 * n_t - n_f \tag{5.40}$$

where nt and nf are the node number in domain Ω and the number of constrained degrees of freedoms, respectively.

In evaluating the integrand at each quadrature (integration) point, three independent strain relations are used. Therefore, the number of independent equations used in all the quadrature points, NQ, is

$$N_Q = 3 * n_Q \tag{5.41}$$

where nQ is the number of total quadrature points in domain Ω. Therefore, NQ must be larger than Nu, to avoid the singularity in the solution, and the minimum

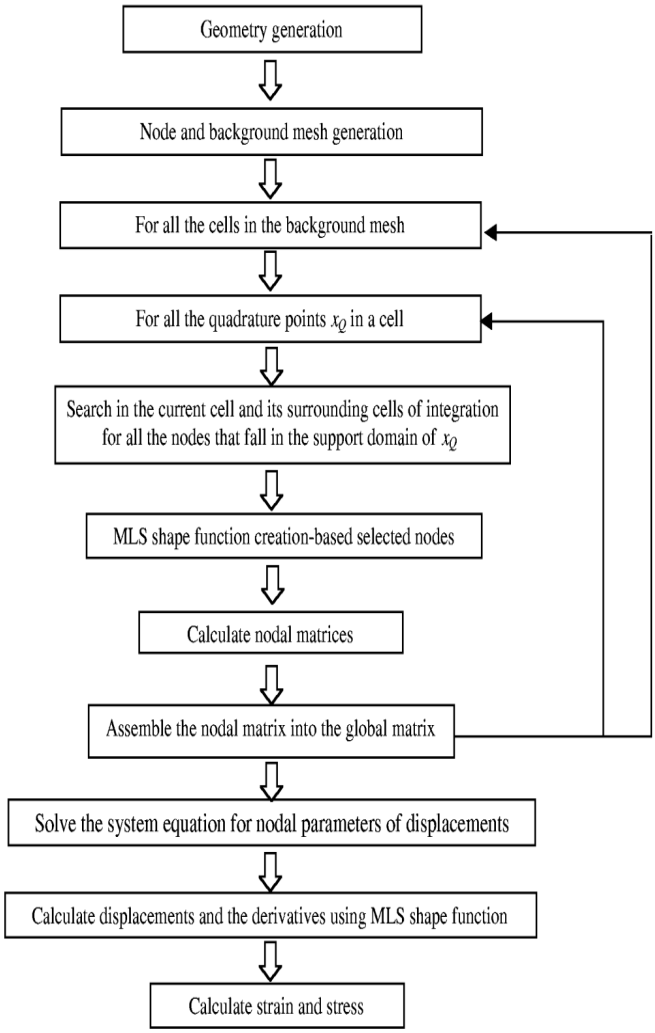


Figure 5.2: Flowchart of EFG method

number of quadrature points must be greater than *Nu/3*. In other words, the total number of quadrature points nQ should be at least two thirds of the total number of unconstrained field nodes in the problem domain, i.e.,

$$N_Q > N_u \approx 2n_t \; or \; n_Q > \frac{2}{3}n_t \qquad for \; 2D \; problem \tag{5.42}$$

Note also that this rule is a necessary requirement, but not necessarily a sufficient requirement. The proper number of quadrature points is studied in the following section using benchmark problems.

5.5 Boundary condition

As described in the previous chapters, the use of MLS approximation produces shape functions that do not possess the Kronecker delta function property, i.e., $\emptyset_I(x_J) \neq \delta_{IJ}$. So that

$$\mathrm{u}^{\mathrm{h}}(x_J) = \sum_{I}^{n} \emptyset_I(x_J)\, u_I \neq u_J \tag{5.43}$$

If u_J, were the prescribed value at a node x_J on Γ_u, the MLS approximant does not reproduce this values at x_J. This causes a serious problem in terms of direct enforcement of essential boundary conditions. A number of techniques are available to circumvent this difficulty. However, each has its limitations and no method to date has received universal acceptance. Some of the techniques available to account for the essential boundary conditions in an EFG formulations are outlined briefly below,

5.5.1 Lagrange multiplier approach

This approach employs Lagrange multipliers for the enforcement of essential boundary conditions. The approach has been described in detail in section 5.1 and results in the system of equations shown in eqn.5.39. This approach has several limitations such as

a. It leads to larger system matrices since additional unknowns corresponding to the Lagrange multipliers are introduced.

b. The bandedness of the system is sacrificed due to the appearance of matrices $[G]$ and $[G]^T$.

c. The matrix, which has to be inverted, possesses zeros at its main diagonals. Solvers with variable bandwidth that do not take advantage of positive definiteness of the matrix need to be employed.

5.6 Penalty parameter approach

In this section, an alternative method "the penalty method" is introduced for the imposition of essential boundary conditions. The use of the penalty method produces equation systems of the same dimensions that conventional FEM produces for the same number of nodes, and the modified stiffness matrix is still positive definite. The problem with the penalty method lies in choosing a penalty parameter that can be used universally for all problems. The penalty method has been used by many researchers; this section follows the formulation reported by G. R. Liu and Yang (1998)[18].

The essential boundary conditions are incorporated into a weak form W3 of the governing equations using a penalty parameter α as shown in eqn.3.15. An analysis of W3 is performed similar to that performed on weak form W2 in section 5.2 results in a set of equations given as

$$[[K] + \ \alpha\,[K^u]]\,\{u\} = \{f\} + \alpha\,\{f^u\} \tag{5.44}$$

Additional terms appearing with the penalty parameter are

$$[K^u_{IJ}] = \int_{\Gamma_u} \emptyset_I\,[S]\,\emptyset_J \ \ d\Gamma \tag{5.45}$$

$$[f_J^u] = \int_{\Gamma_u} \emptyset_J [S] \left\{ \begin{array}{c} \overline{u}_x \\ \overline{u}_y \end{array} \right\} \, d\Gamma \tag{5.46}$$

Where

$$[S] = \left[\begin{array}{cc} S_x & 0 \\ 0 & S_y \end{array} \right] \tag{5.47}$$

$$S_i = \{ \begin{array}{ll} 1 & if\ u_i\ is\ prescribed\ on\ \Gamma_u \\ 0 & if\ u_i\ is\ not\ prescribed\ on\ \Gamma_u \end{array} \tag{5.48}$$

A large value of the penalty parameter in the range $\alpha = 10^3\ to\ 10^{13}$ is employed. The solution of the system of equations may be sensitive to the value of the penalty parameter α. Note that the integration is performed along the essential boundary, and hence matrix K^α will have entries only for the nodes near the essential boundaries Γ_u, which are covered by the support domains of all the quadrature points on Γ_u. Comparing eqn.5.45 with Equation eqn.5.39, the advantages of the penalty method are obvious:

a. The dimension and positive definite property of the matrix are preserved, as long as the penalty parameters chosen are positive.

b. The symmetry and the bandedness of the system matrix are preserved.

These advantages make the penalty method much more efficient and hence much more attractive compared with the Lagrange multipliers method. Studies on implementation of the penalty method and computation of actual application problems have indicated the following minor disadvantages of the penalty method compared with the Lagrange multipliers method.

a. It is necessary to choose penalty parameters that are universally applicable for all kinds of problems. One hopes to use as large as possible penalty parameters, but too large penalty parameters often result in numerical problems.

b. The results obtained are in general less accurate, compared with the method of Lagrange multipliers.

c. An essential boundary condition can never be precisely imposed. It is imposed only approximately.

Despite these minor disadvantages, the penalty method is much more favorable for many researchers. It is also implemented for preparing computer program in C-language (V3) for a two dimensional problem.

5.7 EFG formulation of one dimensional bar

Consider the following one-dimensional problem on the domain *0<x<1*

$$Eu_{,xx} + \ F = 0 \qquad in\ \Omega = (0,1) \tag{5.49}$$

where *u(x)* is the displacement, E is Young's modulus, and F is the body force per unit volume. The following specific boundary conditions are chosen:

$$E, u_{,x} n = \bar{t} \ \ (x = \Gamma_t) \tag{5.50}$$

$$u = \overline{u} \ \ (x = \Gamma_u) \tag{5.51}$$

To obtain the discrete equations it is first necessary to use a weak form of the equilibrium equation and boundary conditions. The following weak form is used: Let trial functions *u(x)∈ H1* and Lagrange multiplier $\lambda \in$ *H0* for all test functions δv(x) $\in$*H1* and $\delta\ \lambda \in$ H0 if

$$\int_0^1 Eu_{,x}\delta v_{,x} d\text{x} - \int_0^1 \delta F_x \delta v_x d\text{x} - \bar{t}_x \delta v_x \,|\Gamma_t - (u_x - \overline{u}_x)\,\delta\lambda_x \,|\Gamma_u - \lambda_x \delta v_x \,|\Gamma_u \tag{5.52}$$

Then the equilibrium condition in eqn.5.44 and boundary conditions in eqn.(5.45 and 5.46) are satisfied. Note that *H1* and *H0* denote the Hilbert's spaces of degree one and zero. In order to obtain the discrete equations from the weak form, the approximate solution u and the test function δv are constructed according to 4.18.

The final discrete equation can be obtained in similar way same a two dimensional way, yielding the following system of linear algebraic equations;

$$\begin{bmatrix} [K] & [G] \\ [G^T] & [0] \end{bmatrix} \begin{Bmatrix} \{u\} \\ \{\lambda\} \end{Bmatrix} = \begin{Bmatrix} \{f\} \\ \{q\} \end{Bmatrix} \tag{5.53}$$

where

$$K_{IJ} = \int_0^1 \Phi_{I,x}^T E \Phi_{J,x} dx \tag{5.54}$$

$$\mathrm{f}_I = \int_0^1 \mathbf{\Phi}_I F_x d\mathrm{x} - \mathbf{\Phi_I}^T \overline{t}_x \tag{5.55}$$

$$G_{IK} = -\mathbf{\Phi}_K \, |\Gamma_{uI} \tag{5.56}$$

$$q_K = -\overline{u}_K \tag{5.57}$$

In eqn.5.50, $\{u\}$ and $\{\lambda\}$ are the vectors of unknown nodal displacements and Lagrange multipliers, respectively. The system of equations 5.50 may be solved using standard linear system solvers to obtain these quantities.

5.7.1 One dimensional bar problem

The element-free Galerkin (EFG) method is a mesh free method because only a set of nodes and a description of the model's boundary are required to generate the discrete equations. The connectivity between the nodes and the approximation functions are completely constructed by the method. The EFG method employs moving least-square (MLS) approximates to approximate the function $u(x)$ with $u^h(x)$. These approximates are constructed from three components:

a weight function of compact support associated with each node, a basis, usually consisting of a polynomial, and a set of coefficients that depend on position. The weight function is nonzero only over a small sub-domain around a node, which is called its support. The support of the weight function defines a nodes domain of influence,

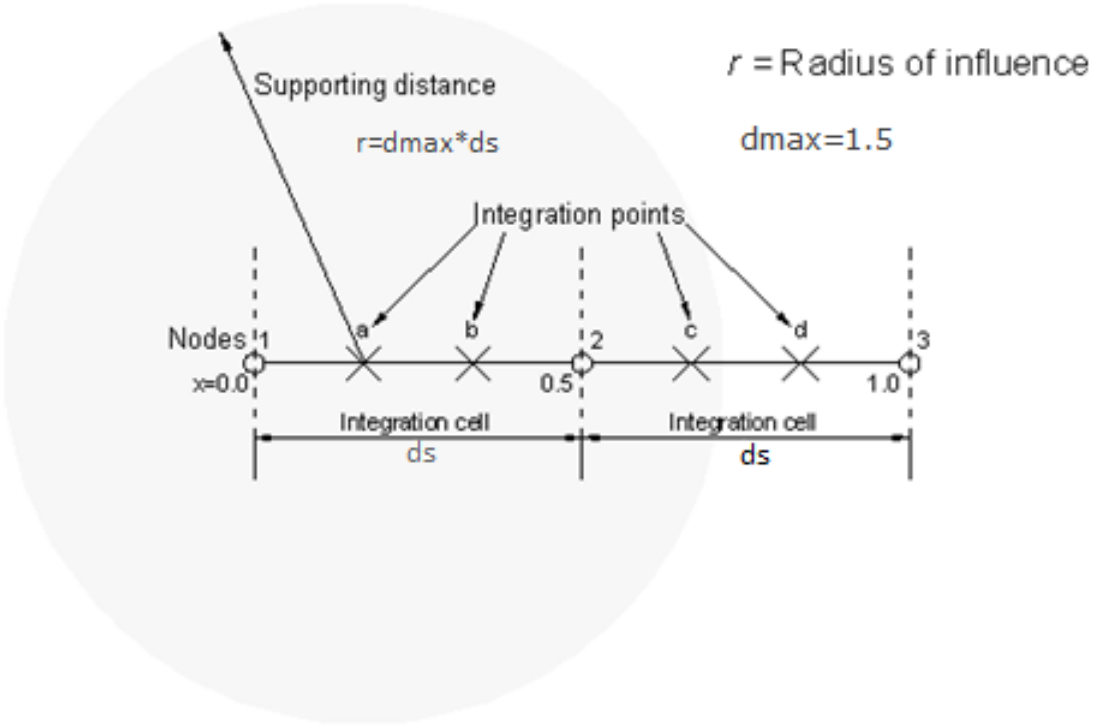


Figure 5.3: Support domain in 1D EFG bar problem

which is the sub-domain over which a particular node contributes to the approximation. The overlap of the nodal domains of influence defines the nodal Connectivity. Fig. 5.7.1 shows various parameters of support domain of a one dimensional bar. In which a support domain with 1.5 radius is shown and two integration cell and four gauss points are shown. For gauss point a node 1 and node 2 are under influence, hence shape function will contributed by this two nodes only. Now, this understanding will be used to analyze a 1D bar problem. To analyze a one dimensional bar a C-program has been prepared (V1), which is subsequently divided in no. of steps. Each step is further described as follows:

a. Input the length, no. of nodes, no. of gauss points, Input dmax, elasticity (E) and area (a). In the first step of the program following input values are inserted as input file in the program. Value of dmax is taken as 2.0 which helps

in deciding the dimension of dmi (Radius of influence).

b. Decide the X-coordinates. Here nodes equally distributed, which is not at all compulsory it can be taken irregularly distributed also.

c. Calculate radius of influence $dmax * avg.distbetweenx - cord.$ Here average spacing between nodes is equal, therefore values of dmi will constant for each gauss point, which is equal to $dmax * avg.Dist.Betweennodes.$

d. Setup gauss point, weight and jacobian. In the program provided, the integration cells are chosen to coincide with the intervals between the nodes, and one-point quadrature is used to integrate the Galerkin weak form. One point quadrature is equivalent to the trapezoidal rule, and it is used in this program for the sake of simplicity. Since the nodes are uniformly spaced, the Jacobian is the same value for each cell. and it is equal to one half the distance between the nodes. The locations of the integration points, their respective weights, and the Jacobian are stored in the array xg. Note that the first value in xg is set to 0.0. This is the point where the essential boundary condition has been enforced, and in the loop over the integration points it will be used to assemble the **G** matrix.

e. Loop over gauss points.

(1) Find distance between gauss point and x-cord. $dif = xg - X - cord.$

(2) Find normalised distance r= absolute(dif)/dmi

(3) Find the weight function (w) and first derivative (dw) of weight function according to cubic or quartic spline function.

(4) Select linear polynomial basis $P[1x]$.

(5) Matrix B is $P * w$.

(6) Matrix A is $w * (P * P^T)$.

(7) Matrix dA is $dw * (P * P^T)$.

(8) Matrix Phi is $P^T * Ainv * B$.

(9) The first derivative of B is $db = P * dw$.

(10) The first derivative of Ainv is $da = -Ainv * dA * Ainv$.

(11) The first derivative of P is $dP[01]$.

(12) The first derivative of Phi is $dPhi = dP * Ainv * B + P(da * B + Ainv * B)$.

(13) The stiffness matrix $K = weight * area * jac * E * (dPhi^T * dPhi)$.

(14) fbody = area*gauss point.

(15) Load vector $f = (weight * fbody * jac) * phi^T$.

(16) Assemble discrete equations.

(17) GG= No. of essential boundary condition.

After loop over all gauss point has finished assemble discrete equation.

f. $[KGG; GGT0]$.

g. $q = 0$.

h. Find displacement $d = K^{-1} * [f^T q]$.

i. Remove extra conditions, find displacements at nodes.

5.8 EFG procedure for 1D beam problem

In 1D beam problem first derivative of x is slope and second derivative of x is displacement. Hence, to compute displacement second derivative of shape function has to be find out. The procedure is same as in 1D bar problem. But the slight change is that, here second derivative of shape function has to be found out for obtaining the nodal stiffness matrix. To understand analysis in a better way, a case of a cantilever

beam with udl is taken as a sample problem in section 6.3. To analyse a 1D beam problem following steps are to be followed.

a. Input the length, no. of nodes and no. of gauss points.

b. Input dmax, Moment of inertia (I) and Elasticity(E).

c. Decide the X-cordinates.

d. Calculate radius of influence $dmax * avg.$ dist between x-cord.

e. Setup gauss point, weight and jacobian.

f. Loop over gauss points.

(1) Find distance between gauss point and x-cord. $dif = xg - X - cord.$

(2) Find the weight function (w), first derivative (dw) of weight function and second derivative of weight function according to cubic or quartic spline function.

(3) Select linear polynomial basis $P[1xx2]$.

(4) Matrix B is $P * w$.

(5) Matrix A is $w * (P * P^T)$.

(6) Matrix dA is $dw * (P * P^T)$.

(7) Matrix ddA is $ddw * (P * P^T)$.

(8) Matrix Phi is $PT * Ainv * B$.

(9) The first derivative of B is $db = P * dw$.

(10) The first derivative of $Ainvisda = -Ainv * dA * Ainv$.

(11) The first derivative of Phi is $dPhi = dP * Ainv * B + P(da * B + Ainv * B)$.

(12) The second derivative of Phi matrix $ddPhi = (ddP * (Ainv * B)) + (P * (dda * B + da * db + da * db + Ainv * ddb)) + (dP * (2 * Ainv * db + 2 * da * B))$

After loop over all gauss point has finished .

(13) Find $Kmatrix = ddPhi * E * I * ddPhi * vg$

(14) Load vector for various loading. $f = phi * udl * vg$
where, udl is applied uniformly distributed load
and vg is distance between gauss points.

g. To incorporate the boundary conditions, For a cantilever beam, two extra rows and columns will be there in stiffness matrix. For a simply supported beam, two extra rows and columns will be incorporated. For a propped cantilever beam, three extra rows and columns will be incorporated. For a fixed beam, four extra rows and columns will be incorporated.

h. Find displacement $d = K^{-1} * f^T$
Remove extra conditions, find displacements at nodes.

i. Solve the equation at gauss points
Displacement at gauss points = $phi * u$
Rotation at gauss points = $dphi * u$

5.9 Programming EFG method

For analysis of structural engineering problems programs have been prepared for one dimensional bar (V1), one dimensional beam (V2) and two dimensional beam (V3). These programs have been prepared in C language. Programs V1 and V2 are programmed using Lagrange multiplier. While preparing program for V3 following experiences were observed are given in section 5.5. so for this part MATLAB has been used for solution of equations. Finally it was decided to use Penalty parameter method for introduction of boundary condition and solution of equation.

Chapter 6

EFG APPLICATIONS

6.1 Introduction

Theory of EFG method has been described in previous chapters. Subsequently based on through study of EFG method, detailed flow chart is presented in this chapter. Computer programs in C-language are developed for 1D bar, 1D beam and 2D plane stress applications. For 2D applications, two methods for introduction of boundary conditions viz. Lagrange method and Penalty method are attempted.
In the following sections different problems are analyzed and results are discussed.

6.2 One dimensional bar problem

Software for analysis of one dimensional elasto-statics problem by EFG method is developed. For the analysis a C-program has been prepared which is given in Appendix-A. For this an example of, one dimensional bar of unit length subjected to linear body force of magnitude x is illustrated.
The displacement at the left end of bar is fixed and right end is traction free. The bar has constant cross sectional area with unit value and, Young's modulus of elasticity E is unit value. Fig. 6.2 shows various parameters of 1D bar problem with discretization considering 11 numbers of nodes. To analyse the bar following procedure is followed.

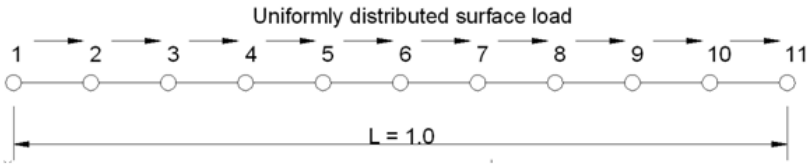


Figure 6.1: Discretization for 1D analysis

Here the program V1 has been used. The computations are done for 4 nodes and 4 gauss points.

a. Input the length, no. of nodes, no. of gauss points, dmax, elasticity (E) and area.

   length = 1, no. of nodes = 4, no. of gauss point = 4.

   dmax = 2.0, E = 1.0, area = 1.0.

b. Decide the X-coordinates.

   X-cord

0	0.333	0.667	1

c. Calculate radius of influence $dmax * avg.$ dist between x-cord.

   $dmi = 2 * 0.333 = 0.667.$

d. Setup gauss point, weight and jacobian.

   gauss points $xg = [0 \;\; 0.167 \;\; 0.5 \;\; 0.833\ ]$

e. Loop over gauss points.

   (1) Find distance between gauss point and x-cord.

   $dif = xg - x - cord.$

   $dif = [0 \quad -0.333 \quad -0.667 \quad -1\ ]$

   (2) Find normalised distance $r = absolute(dif)/dmi$

   $r = [0\ 0.5\ 1\ 1.5\ ]$

(3) Find the weight function (w) and first derivative (dw) of weight function according to cubic or quartic spline function.

w = [0.67 0.17 0 0]

dw = [0 1.5 0 0]

(4) Select linear polynomial basis $P = [1\ x]$.

P^T =

1	1	1	1
0	0.33	0.67	1

(5) $MatrixB = P * w$.

B =

0.667	0.167	0	0
0	0.056	0	0

(6) $MatrixA = w * (P * P^T)$.

A=

0.833	0.056
0.056	0.019

(7) $MatrixdA = dw * (P * P^T)$.

dA =

1.5	0.5
0.5	0.167

(8) $MatrixPhi = P^T * Ainv * B$.

Phi = [1 0 0 0]

(9) The first derivative of B = $db = P * dw$.

$$\text{db} = \begin{bmatrix} 0 & 1.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

(10) The first derivative of $Ainv$ is $da = -Ainv * dA * Ainv$.

da = [-3.13E-07 3.34E-06 4.69E-06 -4.86E+02]

(11) The first derivative of P is dP [0 1].

(12) The first derivative of Phi is $dPhi = dP * Ainv * B + P * (da * B + Ainv * B)$.
dPhi = [-3.00E+00 3.00E+00 0.00E+00 0.00E+00]

(13) The stiffness matrix $K = weight * area * jac * E * (dPhi^T * dPhi)$. K =

0.00E+00	0.00E+00	0.00E+00	0.00E+00
0.00E+00	0.00E+00	0.00E+00	0.00E+00
0.00E+00	0.00E+00	0.00E+00	0.00E+00
0.00E+00	0.00E+00	0.00E+00	0.00E+00

(14) $fbody = area * gausspoint$.

(15) Load vector $f = (weight * fbody * jac) * phi^T$.

$$\text{f} = \begin{bmatrix} 0.00\text{E+}00 \\ 0.00\text{E+}00 \\ 0.00\text{E+}00 \\ 0.00\text{E+}00 \end{bmatrix}$$

(16) Assemble discrete equations.

K =

2.33E+00	-1.71E+00	-5.69E-01	-4.69E-02
-1.71E+00	2.88E+00	-6.01E-01	-5.69E-01
-5.69E-01	-6.01E-01	2.88E+00	-1.71E+00
-4.69E-02	-5.69E-01	-1.71E+00	2.33E+00

(17) GG = No. of essential boundary condition.

Here left end of the bar is fixed for displacement, hence only one essential boundary condition is there. Shape function at the left end of bar is given by,

$GG = [\ -1 \ \ 0 \ \ 0 \ \ 0\ ]$

After loop over all gauss point has finished assemble discrete equation.

f. Impose Lagrange multiplier condition

m =

K	GG
GG^T	0

m =

2.329	-1.714	-0.569	-0.047	-1
-1.714	2.883	-0.601	-0.569	0
-0.569	-0.601	2.884	-1.714	0
-0.047	-0.569	-1.714	2.329	0
-1	0	0	0	0

g. q=0.

h. Find displacement $d = m^{-1} * [f^T q]$.

$$d = \begin{bmatrix} 0 \\ 0.165 \\ 0.3 \\ 0.324 \\ -0.5 \end{bmatrix}$$

i. Remove extra conditions, find displacements at nodes.

$$u = \begin{bmatrix} 0 \\ 0.165 \\ 0.3 \\ 0.324 \end{bmatrix}$$

The exact solution is given by the equation:

$$u(x) = \frac{1}{E}\left[\frac{1}{2}x - \frac{x^3}{6}\right]$$

Table 6.2 shows the x distance, exact displacement, computed displacement by program V1 and error percentage for 6 nodes.

Table 6.2 shows the x distance, exact displacement, computed displacement by program and error percentage for 11 nodes.

Table 6.2 shows the x distance, exact displacement, computed displacement and error percentage for 21 nodes.

X	Exact Disp.	Computed Disp.	% error
0	0	0	0
0.2	0.0987	0.0995	0.840
0.4	0.1893	0.1924	1.618
0.6	0.2640	0.2666	0.977
0.8	0.3147	0.3224	2.455
1	0.3333	0.3325	0.238

Table 6.1: Displacement results for 6 nodes

X	Exact Disp.	Computed Disp.	% error
0	0.0000	0.0000	0
0.1	0.0498	0.0502	0.808
0.2	0.0987	0.0988	0.163
0.3	0.1455	0.1459	0.277
0.4	0.1893	0.1898	0.263
0.5	0.2292	0.2298	0.276
0.6	0.2640	0.2647	0.283
0.7	0.2928	0.2937	0.311
0.8	0.3147	0.3155	0.274
0.9	0.3285	0.3302	0.526
1.0	0.3333	0.3325	0.259

Table 6.2: Displacement results for 11 nodes

Fig. 6.2 shows the displacement results for one dimensional bar by exact analytical and EFG method. For EFG method analysis has been done for 6 nodes, 11 nodes and 21 nodes.

From Table 6.2, 6.2 and 6.2 it can be observed that the in general, the results are having good co-relation with the exact results.

x	Exact Disp.	Computed Disp.	% error
0	0	0	0
0.05	0.0250	0.0251	0.610
0.1	0.0498	0.0498	0.028
0.15	0.0744	0.0745	0.081
0.2	0.0987	0.0987	0.062
0.25	0.1224	0.1225	0.065
0.3	0.1455	0.1456	0.065
0.35	0.1679	0.1680	0.066
0.4	0.1893	0.1895	0.067
0.45	0.2098	0.2100	0.068
0.5	0.2292	0.2293	0.069
0.55	0.2473	0.2474	0.070
0.6	0.2640	0.2642	0.071
0.65	0.2792	0.2794	0.073
0.7	0.2928	0.2931	0.075
0.75	0.3047	0.3049	0.077
0.8	0.3147	0.3149	0.079
0.85	0.3226	0.3229	0.085
0.9	0.3285	0.3287	0.075
0.95	0.3321	0.3326	0.135
1	0.3333	0.3331	0.065

Table 6.4: Displacement results for 21 nodes

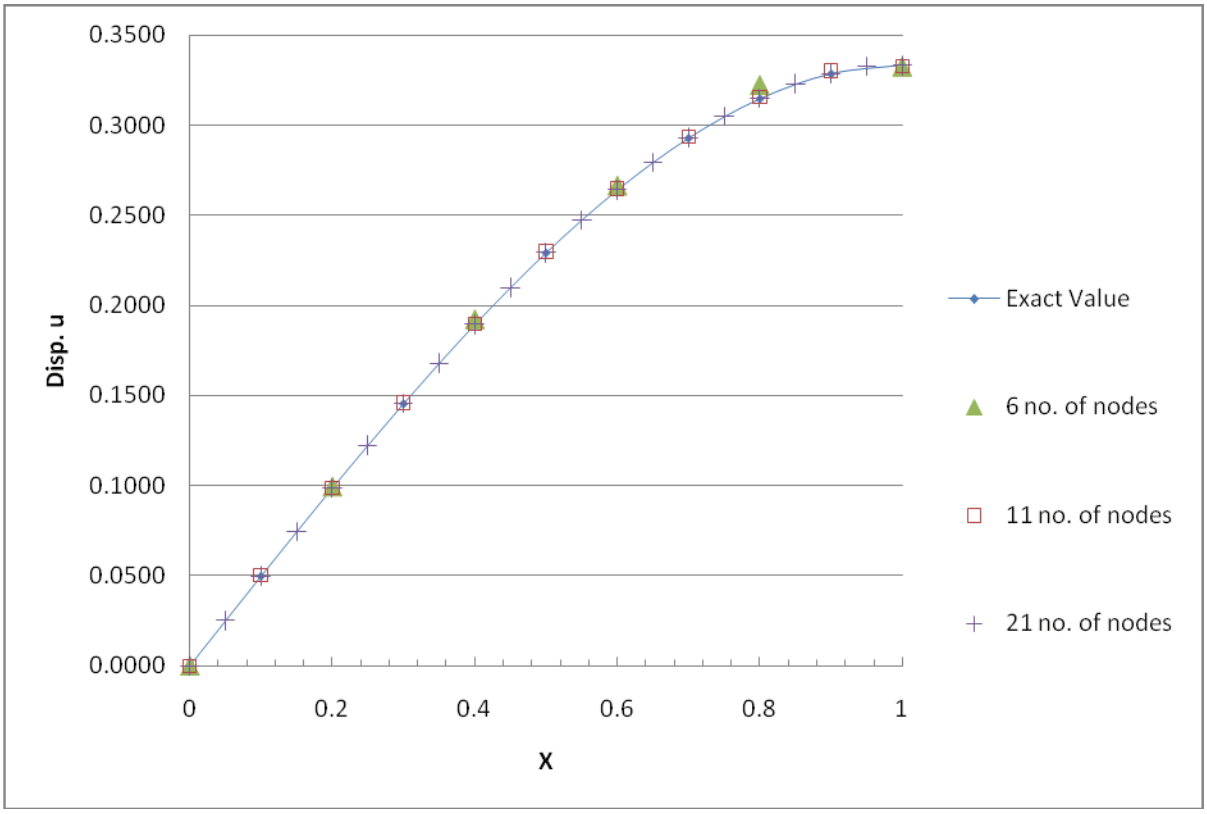


Figure 6.2: Displacement results by exact and EFG method

6.3 Temperature domain problem

A bar as shown in Fig. 6.3 of 1m length with square cross section area of $0.2mx0.2m$ is subjected to a flux of 1000 W/m2 at one end. The other end of bar is fixed at 0^o C. All other surfaces are insulated. Results are shown in the Fig. 6.3 which is the exact solution of the problem and have been checked by software package NISA. EFG procedure remains same as the 1D bar problem with elasticity replaced by conductivity and thermal flux load is applied as point load at the end. Here the stiffness matrix and load vector are given.

Where, point load = 5 x 1000 x 0.22 = 200 W C

The computed nodal stiffness matrix for 5 noded bar is

$K = (w * E * area * jac) * (dphi' * dphi)$

Figure 6.3: 1D heat conduction in a bar

X meters	0	0.2	0.4	0.6	0.8	1
EFG results	0	1.01	2	3	3.99	5
NISA results	0	1	2	3	4	5

Table 6.5: Comparison of temperature distribution along the bar axis

1.55E+02	-1.14E+02	-3.79E+01	-3.13E+00	0.00E+00	0.00E+00
-1.14E+02	1.92E+02	-4.35E+01	-3.13E+01	-3.13E+00	0.00E+00
-3.79E+01	-4.35E+01	1.63E+02	-4.69E+01	-3.13E+01	-3.13E+00
-3.13E+00	-3.13E+01	-4.69E+01	1.63E+02	-4.35E+01	-3.79E+01
0.00E+00	-3.13E+00	-3.13E+01	-4.35E+01	1.92E+02	-1.14E+02
0.00E+00	0.00E+00	-3.13E+00	-3.79E+01	-1.14E+02	1.55E+02

And the load vector is force vector f = $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 200 \end{bmatrix}$

The temperature distribution (generally termed as displacement) is given in Table 6.3. It is observed that results match very well with those obtained from use of software NISA. Fig. 6.3 shows the graph of results.

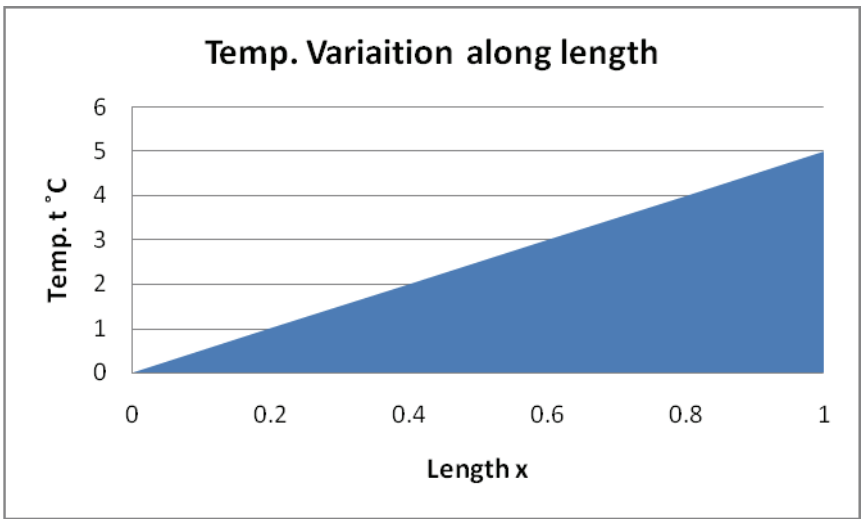


Figure 6.4: Temp. variation along bar length

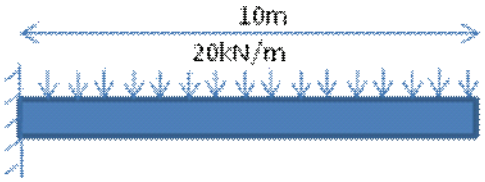


Figure 6.5: Details of cantilever beam

6.4 1D beam problem

One dimensional beam problem has two degree of freedom at each node. A sample problem of one dimensional cantilever beam is taken for analysis. The beam is fixed at left support and right support is traction free (free end). The beam is subjected to uniformly distributed load of 20 kN/m over its entire span. The value of Young's modulus is $1000kN/m^2$ and moment of inertia is $1000m^4$.

To analyze one dimensional beam Problems a C-program has been developed entitled V2, listing of which is given in Appendix-B. The steps involved in analysis are given below, the value of dmax is taken as 3.0. The x coordinates are equally distributed

and number of gauss point are double the number of nodes.

a. Input the length, no. of nodes and no. of gauss points.
   length = 10, no. of nodes = 4, no. of gauss point = 8.

b. Input dmax, Moment of inertia (I) and Elasticity(E).
   dmax = 3, E = 1000.0, I = 1000.0.

c. Decide the X-cordinates.
   X-cord

0	3.33	6.67	10

d. Calculate radius of influence dmax*avg. dist between x-cord.
   $dmi = 3 * 3.33 = 10.$

e. Setup gauss point, weight and jacobian.
   gauss points = [0.62 1.88 3.12 4.38 5.62 6.88 8.12 9.38]

f. Loop over gauss points.

   (1) Find distance between gauss point and x-cord.
   $dif = xg - x - cord.$
   $dif = [0.62 \ -2.71 \ -6.04 \ 9.38]$

   (2) Find the weight function (w), first derivative (dw) of weight function and second derivative of weight function according to cubic or quartic spline function.
   w =[0.9785 0.7027 0.1744 0.0009]
   dw=[-0.0659 0.1728 0.1136 0.0044]
   ddw=[-0.09141 -0.01641 0.03859 0.01359]

   (3) Select linear polynomial basis $P = [1xx^2]$.

1.00000	1.000	1.000	1.000
-0.62500	2.708	6.042	9.375
0.39063	7.335	36.502	87.891

(4) Matrix $B = P * w$.

0.978	0.703	0.174	0.001
-0.612	1.903	1.054	0.009
0.382	5.154	6.367	0.082

(5) Matrix $A = w * (P * P^T)$.

1.857	2.354	11.985
2.354	11.985	52.955
11.985	52.955	277.557

(6) Matrix dA = $dw * (P * P^T)$.

0.225	1.237	5.774
1.237	5.774	32.121
5.774	32.121	194.586

(7) Matrix ddA = $ddw * (P * P^T)$.

0.056	0.373	2.447
0.373	2.447	19.408
2.447	19.408	155.533

(8) Matrix $Phi = P^T * Ainv * B$.

$Phi = [0.738\ 0.335\ -0.071\ -0.002]$

(9) The first derivative of B is db = $P * dw$.

-0.066	0.1728	0.1136	0.0044
0.0412	0.468	0.6863	0.0412
-0.0257	1.2675	4.1464	0.3862

(10) The first derivative of Ainv is $da = -Ainv * dA * Ainv$.

-3.617e-02	-1.297e-01	3.291e-02
-1.297e-01	-1.603e-01	4.552e-02
3.291e-02	4.552e-02	-1.470e-02

(11) The first derivative of Phi is $dPhi = dP * Ainv * B + P(da * B + Ainv * B)$.

dPhi = $[3.834E-01\ 4.563E-01\ 6.259E-02\ 1.03E-02]$

(12) The second derivative of Phi matrix

$ddPhi = (ddP * (Ainv * B)) + (P * (dda * B + da * db + da * db + Ainv * ddb)) + (dP * (2 * Ainv * db + 2 * da * B))$

$ddPhi = [3.834E-01 4.563E-01 6.259E-02 1.039E-02]$

After loop over all gauss point has finished.

(13) Find matrix $K = ddPhi * E * I * ddPhi * vg$.

4.32E+04	-8.90E+04	4.85E+04	-2.67E+03
-8.90E+04	2.27E+05	-1.86E+05	4.85E+04
4.85E+04	-1.86E+05	2.27E+05	-8.90E+04
-2.67E+03	4.85E+04	-8.90E+04	4.32E+04

(14) To incorporate the boundary conditions

For cantilever beam two extra rows and columns will be there in stiffness matrix. Two extra conditions will be at fixed end. Therefore first condition will be shape function at fixed end and second condition will be first derivative of shape function at fixed end.

Hence new nodal stiffness matrix will be

4.32E+04	-8.90E+04	4.85E+04	-2.67E+03	1.00E+00	-4.50E-01
-8.90E+04	2.27E+05	-1.86E+05	4.85E+04	-1.77E-08	6.00E-01
4.85E+04	-1.86E+05	2.27E+05	-8.90E+04	-1.72E-08	-1.50E-01
-2.67E+03	4.85E+04	-8.90E+04	4.32E+04	0.00E+00	0.00E+00
1.00E+00	-1.77E-08	-1.72E-08	0.00E+00	0.00E+00	0.00E+00
-4.50E-01	6.00E-01	-1.50E-01	0.00E+00	0.00E+00	0.00E+00

(15) Find Load vector for various loading.

$f = phi * udl * vg$

where, udl is applied uniformly distributed load

and vg is distance between gauss points.

f = [-24.414 -75.586 -75.586 -24.414 0.000 0.000] here, last two zero are for initial displacement and rotation.

g. Find displacement $d = K^{-1} * f^T$

$$d = \begin{bmatrix} -0.004 \\ -0.004 \\ -0.013 \\ -0.023 \\ -200 \\ -999.999 \end{bmatrix}$$

Remove extra conditions, find displacements at nodes.

$$u = \begin{bmatrix} -0.004 \\ -0.004 \\ -0.013 \\ -0.023 \end{bmatrix}$$

h. solve the equation at gauss points

Displacement at gauss points = phi * u

Rotation at gauss points = dphi * u

Displacement at gauss points =

0.625	-0.000122
1.875	-0.001243
3.125	-0.003271
4.375	-0.005980
5.625	-0.009268
6.875	-0.013015
8.125	-0.017118
9.375	-0.021352

Rotation at gauss points =

0.625	-0.000418
1.875	-0.001313
3.125	-0.001907
4.375	-0.002414
5.625	-0.002830
6.875	-0.003152
8.125	-0.003389
9.375	-0.003323

Fig. 6.4 shows the displacements and rotations at various gauss points.

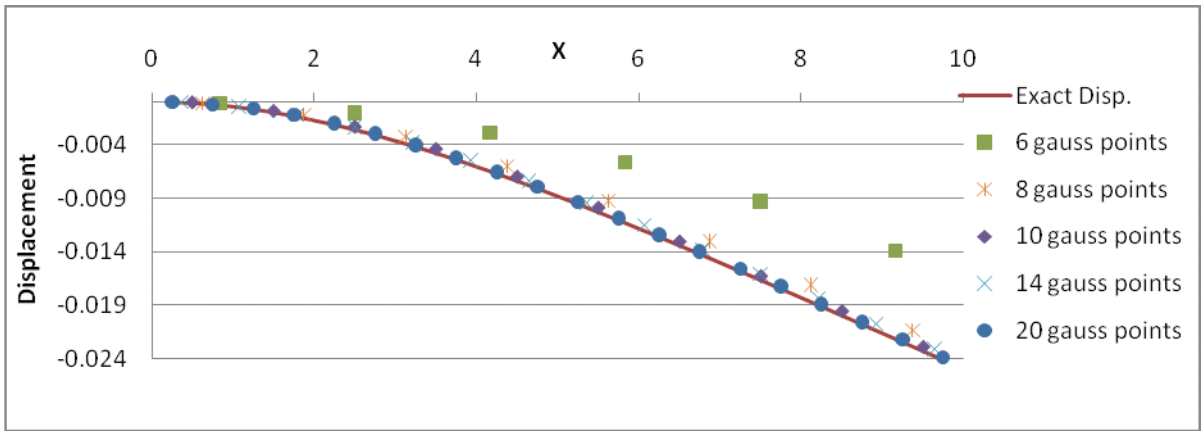
The results are compared with the exact values of displacement and rotation.

In order to test and validate the program V2, a beams with various boundary conditions has been analyzed. Thus results are also plotted for simply supported beam, propped cantilever and fixed beam.

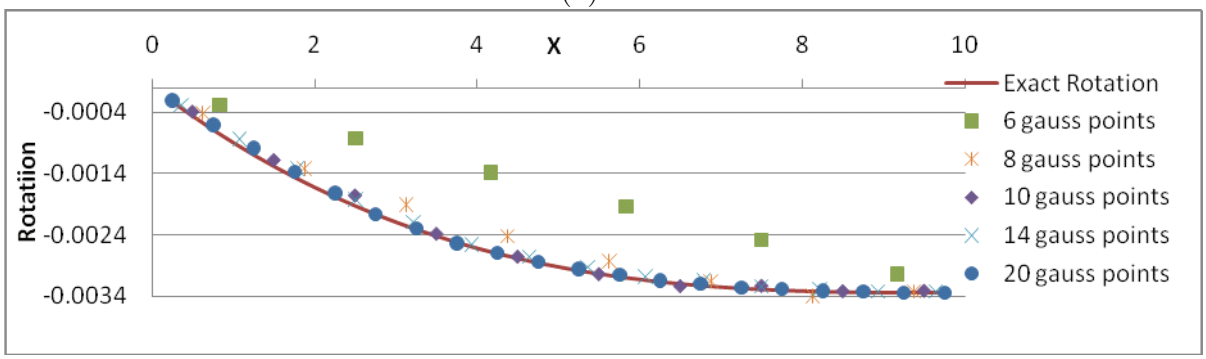
Fig. 6.4 shows a simply supported beam of length 10 m and u.d.l. of 20 kN/m. For simply supported beam exact displacement is 2.603E-03 at mid span and rotation is 8.33E-04 at ends. The percentage error in displacement is 2 % and that in rotation is 0.04 %.

Fig. 6.4 shows a propped cantilever beam with length 10 m and u.d.l. of 20 kN/m. The maximum displacement obtained from EFG method is 1E-03 and exact displacement is 1.08E-03.

Fig. 6.4 shows a fixed beam with length 10 m and 20 kN/m. The maximum displacement obtained by EFG method is 4.4E-04 and exact displacement is 5.21E-04.

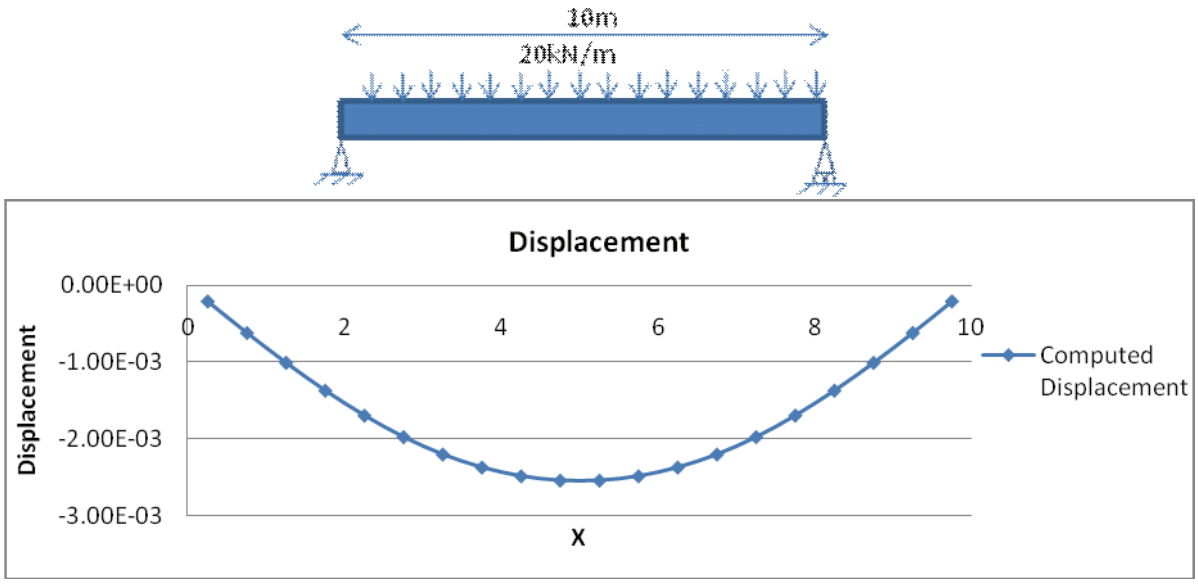


(a)

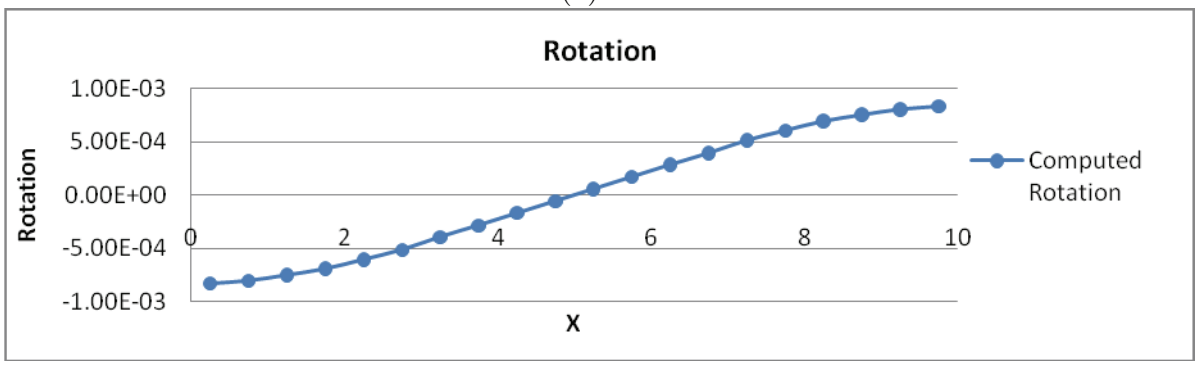


(b)

Figure 6.6: Cantilever beam with U.D.L. (a) Displacement (b)Rotation



(a)



(b)

Figure 6.7: Simply supported beam with U.D.L. (a) Displacement (b) Rotation

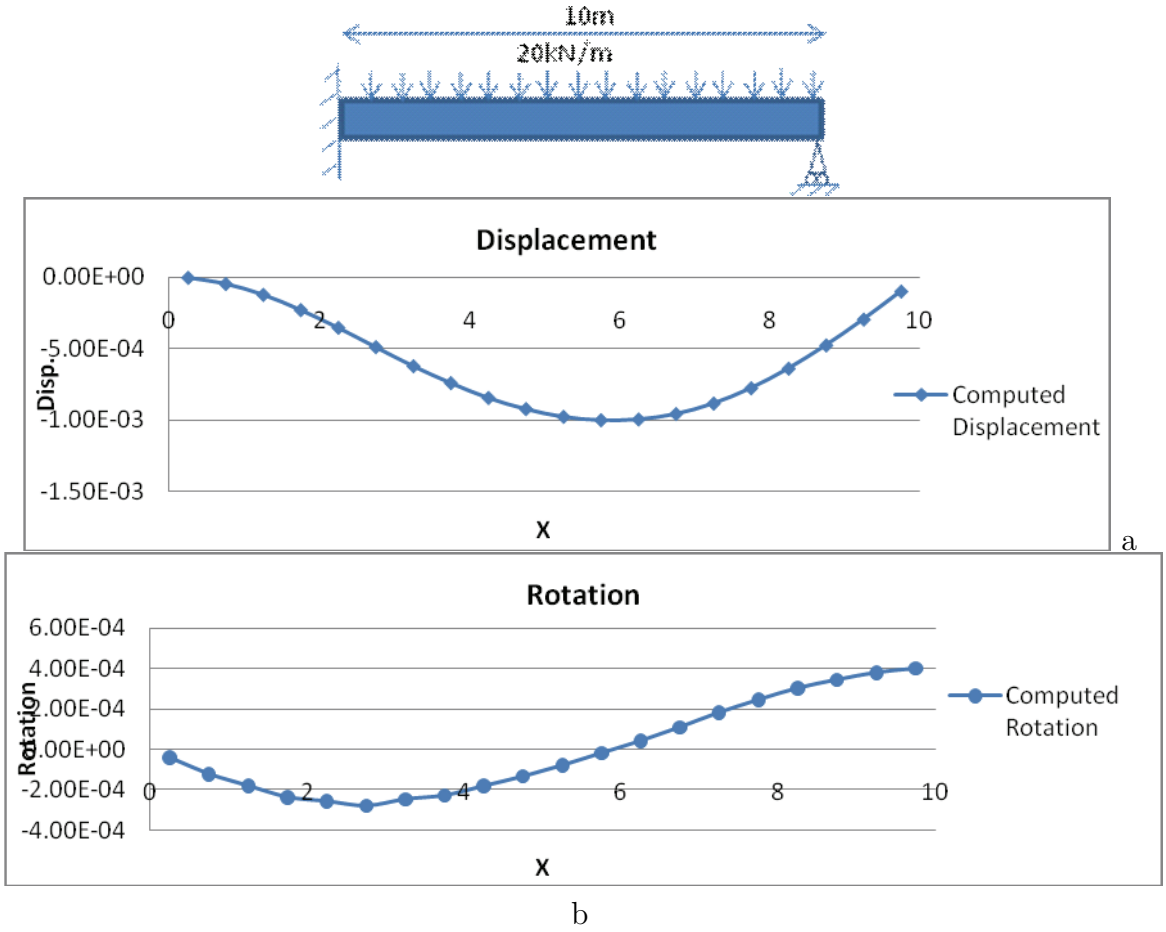
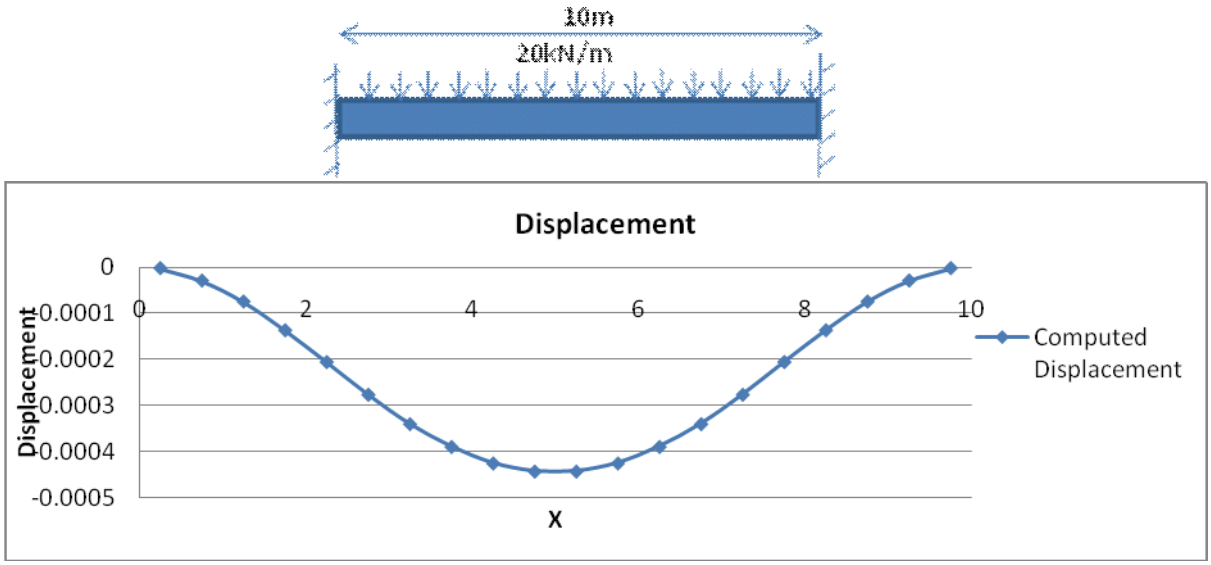
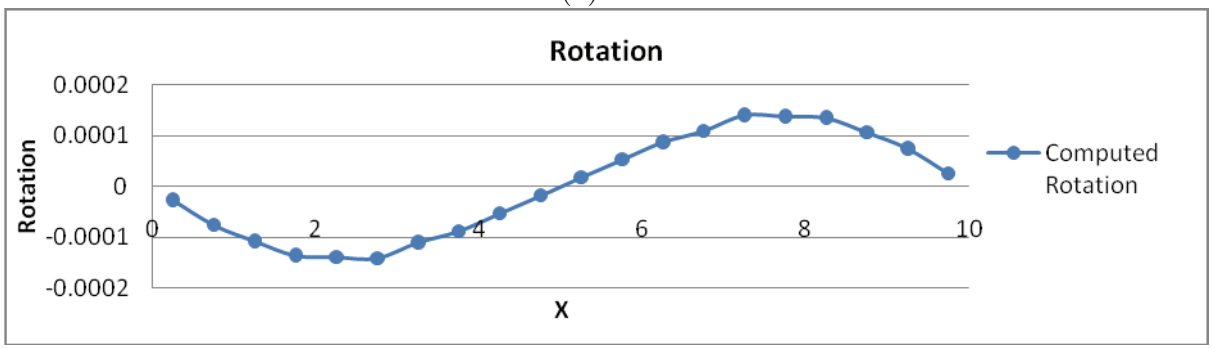


Figure 6.8: Propped cantilever with U.D.L. (a) Displacement (b) Rotation



(a)



(b)

Figure 6.9: Fixed beam with U.D.L. (a) Displacement (b) Rotation

6.5 EFG procedure for 2D beam problem

To solve a problem using the EFG method in 2D domain steps are as follows

a. Insert the Height, length, young's modulus (E), Poisson's ratio (mu) and load values.

b. Construct plane stress matrix (D)

$$\mathrm{D} = E/(1 - mu2) * \begin{bmatrix} 1.0 & \mathrm{mu} & 0 \\ \mathrm{mu} & 1.0 & 0 \\ 0 & 0 & \text{(1-mu)/2} \end{bmatrix}$$

c. Setup nodal coordinates (no. of nodes in row and column). The co-ordinate where the displacements are to be found out are decided it can be randomly distributed or regularly distributed.

d. Determine domain of influence. Domain of influence is calculated as described in section 3.4, it can be a circular or rectangular type of support domain.

e. Set up gauss points, weights, and Jacobian for each cell in each cell there will be 16 gauss points.

   Many forms of quadrature formulas exist; however, the most accurate for polynomial expressions is Gauss-Legendre quadrature. Gauss-Legendre quadrature tables are generally tabulated over the range of coordinates -1 $< \xi <$ 1 (hence the main reason for also choosing many shape function on this interval). Here the 4th order Gauss-Legendre formula is used, hence in each there will be 4 gauss points in each row and column, which makes it to have 16 gauss points in each cell.

   Gaussian points

   ξ = [-0.861 -0.34 0.34 0.8611]

   w = [0.3479 0.6521 0.6521 0.3479]

   Take ξ as (row (s) and column (t)) than,

   N1= $\frac{1}{4}$(1-(s))*(1-(t))

N2= $\frac{1}{4}$(1+(s))*(1-(t))

N3= $\frac{1}{4}$(1+(s))*(1+(t))

N4= $\frac{1}{4}$(1-(s))*(1+(t))

where s=gauss pt(loop 1 to 4) and t=gauss pt (loop 1 to 4)

and weight = weight of s * weight of t.

Coordinate of gauss point will decided from following formula where (x1, x2, x3, x4) are x-coordinates of cell, and (y1, y2, y3, y4) are the y-coordinates of cell.

gs $x = N1 * x1 + N2 * x2 + N3 * x3 + N4 * x4$

gs $y = N1 * y1 + N2 * y2 + N3 * y3 + N4 * y4$

To find the Jacobian of each Gauss point the equations are,

$Npsi1 = *-1*(1-s)$

$Npsi2 = *1*(1-s)$

$Npsi3 = *1*(1+s)$

$Npsi4 = *-1*(1+s)$

$Neta1 = *-1*(1-t)$

$Neta2 = *-1*(1+t)$

$Neta3 = *1*(1+t)$

$Neta4 = *1*(1-t)$

$xpsi = Npsi1 * x1 + Npsi2 * x2 + Npsi3 * x3 + Npsi4 * x4$

$ypsi = Npsi1 * y1 + Npsi2 * y2 + Npsi3 * y3 + Npsi4 * y4$

$xeta = Neta1 * x1 + Neta2 * x2 + Neta3 * x3 + Neta4 * x4$

$yeta = Neta1 * y1 + Neta2 * y2 + Neta3 * y3 + Neta4 * y4$

$jcob = xpsi * yeta - xeta * ypsi$

(1) Loop over gauss points to assemble discrete equations

(2) Determine nodes in neighbourhood of gauss point

(3) Find shape function, first derivative of shape function w.r.t x and first derivative of shape function w.r.t. y.

To find shape function and first derivative of the shape function the procedure is described in section 4.3.

(4) Find Bmat for each node as per

$$\mathrm{B} = \begin{bmatrix} \text{diphix} & 0 \\ 0 & \text{dphiy} \\ \text{dphiy} & \text{dphix} \end{bmatrix}$$

(5) Kmatrix at each node = $weight * jac * (Bmat^T * Dmat * Bmat)$

(6) Determine nodes on boundary, set up boundary conditions

To setup boundary condition between nodes 1D method of determining Gauss points is used. Four gauss points are there between two boundary nodes. Find GG matrix.

(7) Setup gauss point along traction boundary.

Same as boundary element nodes, traction boundary nodes also have four Gauss points between two traction boundary nodes.

(8) Integrate GG matrix and q vector along displacement boundary.

If Lagrange multiplier method is used integration of boundary nodes and initial displacement boundary nodes are to be integrated according to integration condition.

f. Enforce boundary conditions using Lagrange multipliers

The condition of Lagrange multiplier is described in section 5.1 and is as follows,

$$\begin{bmatrix} \mathrm{K} & \mathrm{GG} \\ GG^T & 0 \end{bmatrix} \begin{Bmatrix} \mathrm{f} \\ \mathrm{q} \end{Bmatrix} = \begin{Bmatrix} \mathrm{u} \\ \lambda \end{Bmatrix}$$

It is observed that zero terms appear on the diagonal of stiffness matrix when the EFG approach has been used for getting the solution. Following are the other observations pertaining to this approach[1].

(1) The size of the stiffness matrix increases with the incorporation of the extra matrix G and G^T.

(2) The stiffness matrix which has to be inverted, posses a zero value on its main diagonal, which is very difficult to invert, because of the fact that several limitations creep in Lagrangian multiplier approach as it uses Lagrangian multipliers for the enforcement of essential boundary conditions. Due to these multipliers the bandedness of the system is sacrificed due to appearance of G and G^T as mentioned above. Therefore, to handle these issues MATLAB has been used which has built-in solvers with variable bandwidth which do not take advantage of positive definiteness of the matrix as suggested by (Lu et al. 1994) [9].

g. Solve for output variables - displacements

Output variables u and λ are found out by solving the above equation, from this u and λ are separated.

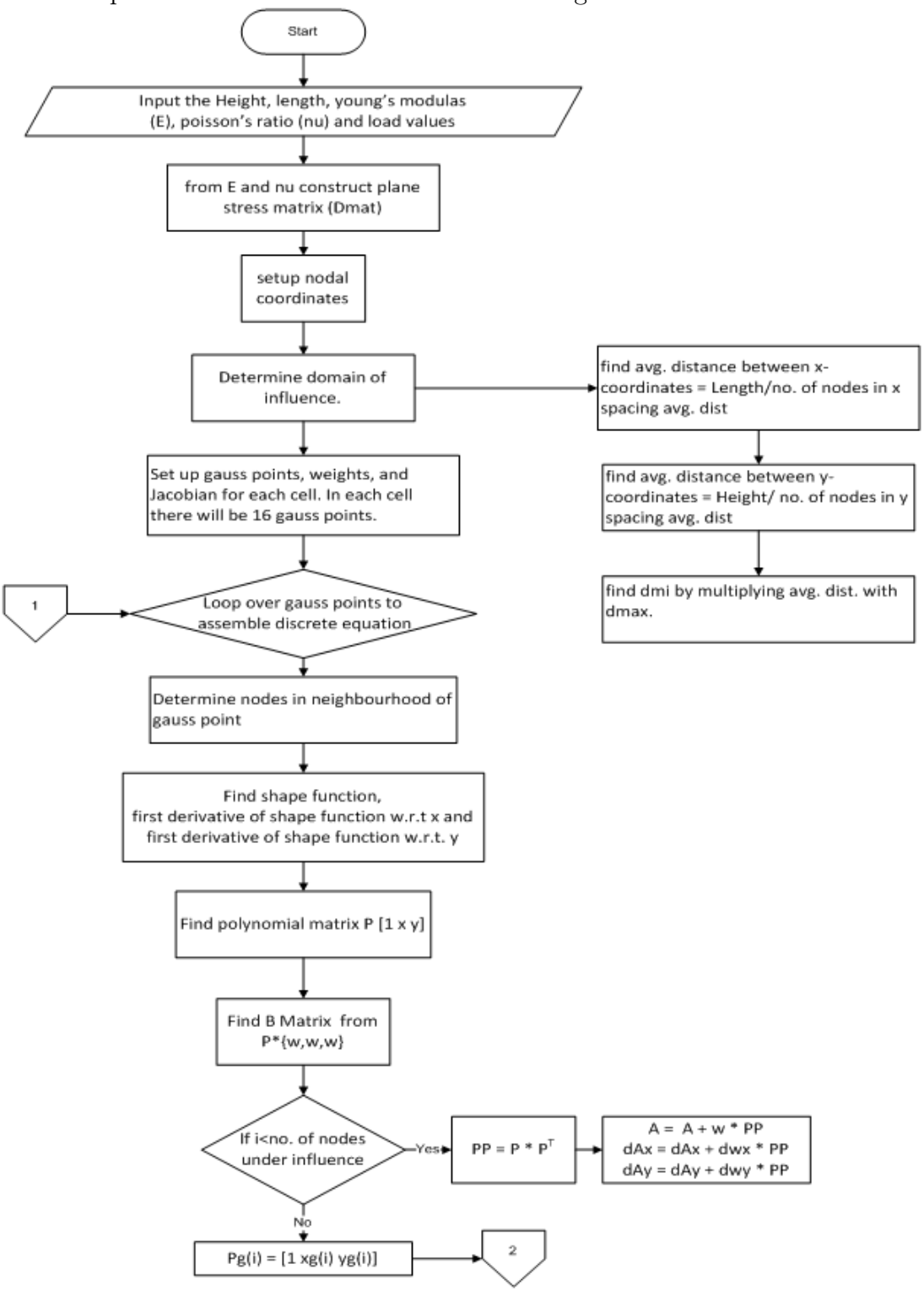
h. Solve for stresses at gauss points

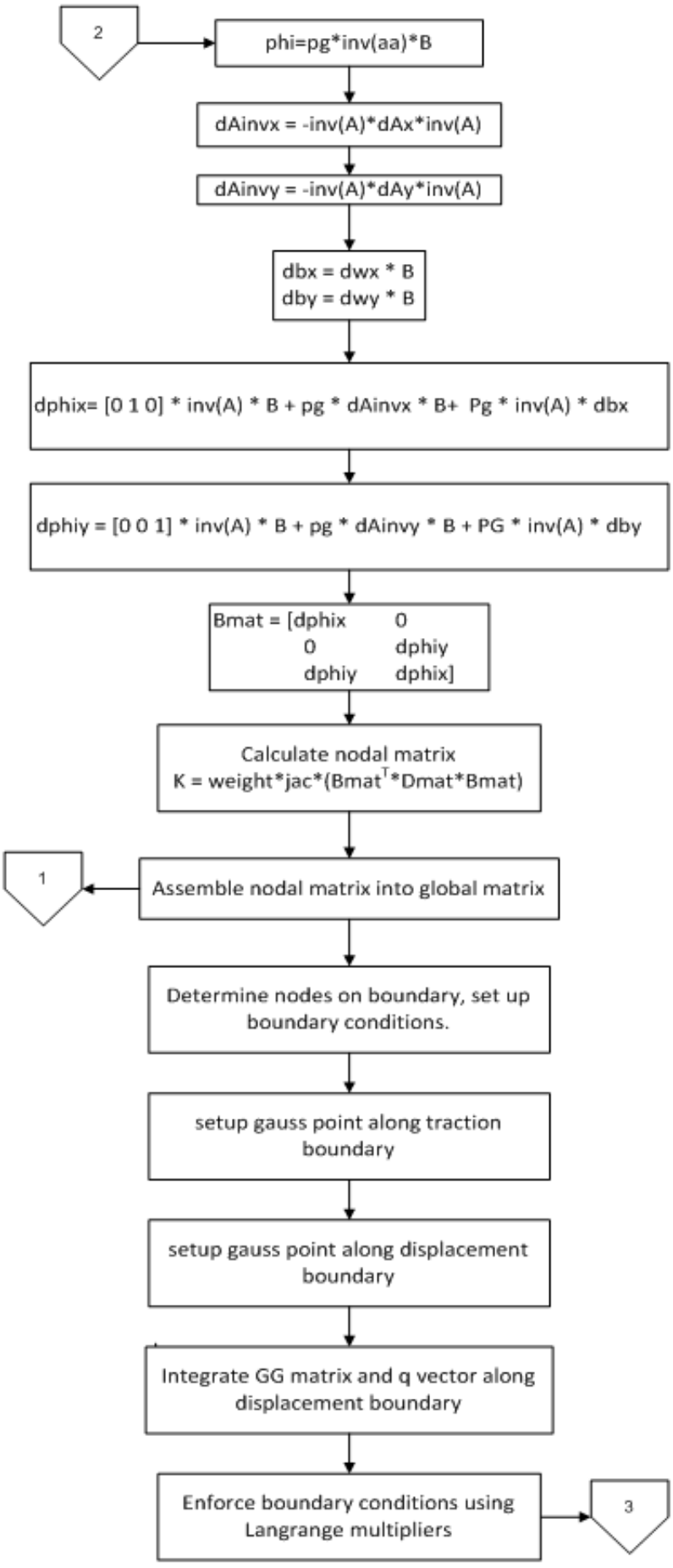
Loop over gauss point

find the domain of influence for each gauss point find shape function at each gauss point find the Bmat at each gauss point stress $\sigma = D * Bmat * u$ Loop end gauss point.

i. Plot the results.

These steps are shown in flowchart from in the Fig. 6.5.




2
phi=pg*inv(aa)*B
dAinvx = -inv(A)*dAx*inv(A)
dAinvy = -inv(A)*dAy*inv(A)
dbx = dwx * B
dby = dwy * B
dphix= [0 1 0] * inv(A) * B + pg * dAinvx * B+ Pg * inv(A) * dbx
dphiy = [0 0 1] * inv(A) * B + pg * dAinvy * B + PG * inv(A) * dby
Bmat = [dphix 0
0 dphiy
dphiy dphix]
Calculate nodal matrix
K = weight*jac*(Bmat^T*Dmat*Bmat)
1
Assemble nodal matrix into global matrix
Determine nodes on boundary, set up boundary conditions.
setup gauss point along traction boundary
setup gauss point along displacement boundary
Integrate GG matrix and q vector along displacement boundary
Enforce boundary conditions using Langrange multipliers
3

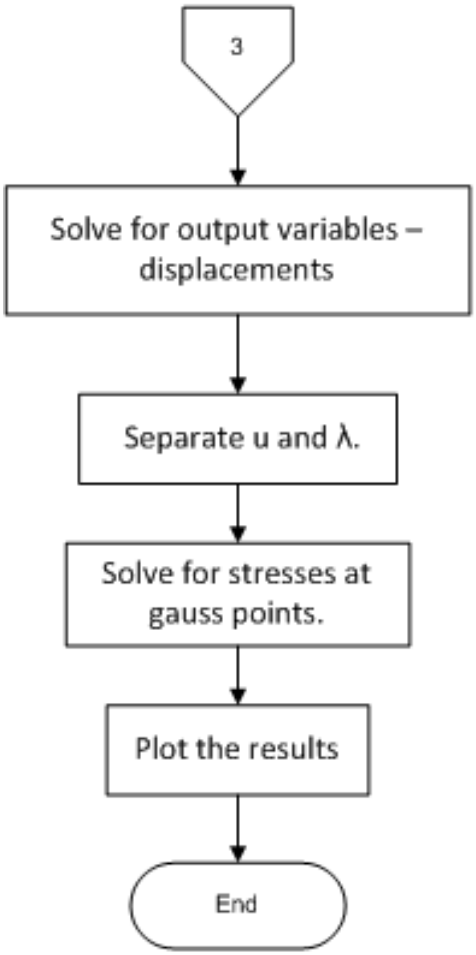


Figure 6.10: Flow of 2D beam by EFG method

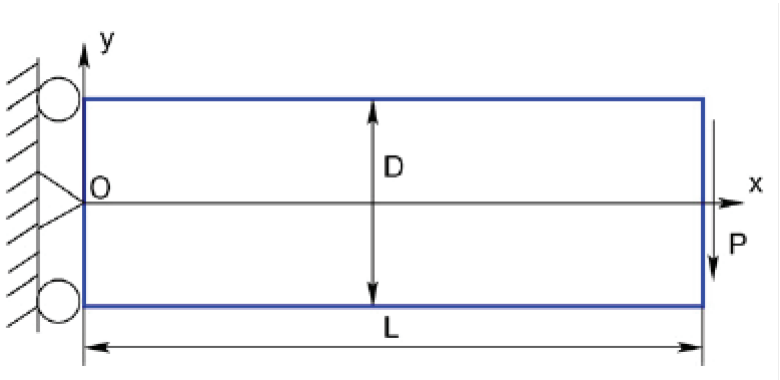


Figure 6.11: The Timoshenko beam details

6.6 Timoshenko beam by Lagrange method

For introduction of boundary conditions, two methods are examined for plane stress problems. First is Lagrange multiplier method and second is Penalty method.
Numerical study is conducted for a cantilever beam, which is often used for benchmarking numerical methods because the exact analytic solution for this problem is known. The purpose here has been to investigate issues related to background integration in the EFG method. There are a number of factors affecting the accuracy of the numerical results of the EFG method. These factors include the number of field nodes n, the background mesh density, and the order of Gauss integration. To provide a quantitative indication of how these factors affect the accuracy of results, a cantilever beam subjected to a load at the free end, as shown in Fig. 6.6, is analyzed in detail using an EFG code.

In this example, the parameters for this cantilever beam are taken as follows:
Loading: P = 1000 N
Young's modulus: $E = 3x10^7 N/m^2$
Poisson's ratio: $v = 0.3$
Height of the beam: D = 12 m
Length of the beam: L = 48 m
The force P is distributed in a form of parabola at the right end of the beam:
In each integration cell, 4 × 4 Gauss quadrature is used. A linear basis and cubic

spline weight function are used in the MLS approximation. Rectangular support of 3.5 times the nodal spacing is employed.

a. Insert the Height, length, young's modulas (E), poisson's ratio (nu) and load values.

   Length = 48

   Height = 12

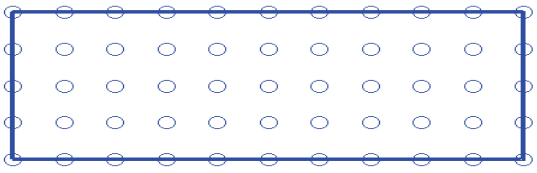
   E = 30E06

   Poisson's ratio = 0.3

b. from E and nu construct plane stress matrix (Dmat).

$$\begin{bmatrix} 3.30E{+}07 & 9.89E{+}06 & 0.00E{+}00 \\ 9.89E{+}06 & 3.30E{+}07 & 0.00E{+}00 \\ 0.00E{+}00 & 0.00E{+}00 & 1.15E{+}07 \end{bmatrix}$$

c. setup nodal coordinates (no. of nodes in row and column).



   No. of nodes in column = 11

   No. of nodes in row = 5

   No. of cells 40

   No. of nodes 55

d. Determine domain of influence.

(1) find avg. distance between x-coordinates = length/no. of nodes in x spacing

avg. dist. = 48/10 = 4.8

(2) find avg. distance between y-coordinates = Height/ no. of nodes in y spacing

avg. dist. = 12/4 = 3.0

(3) find dmi by multiplying avg. dist. with dmax.

$dmax = 3.5.$

$dmiinx = 4.8 * 3.5 = 16.8$

$dmiiny = 3.0 * 3.5 = 10.5$

e. Set up gauss points, weights, and Jacobian for each cell in each cell there will be 16 gauss points.

for loop 1

the gauss points will be

for cell 1 it is,

N1=0.25*(1-(-0.861))*(1-(-0.861)) =0.866

N2=0.25*(1+(-0.861))*(1-(-0.861))=0.0646

N3=0.25*(1+(-0.861))*(1+(-0.861))= 0.0048

N4=0.25*(1-(-0.861))*(1+(-0.861))=0.0646

For the coordinates of the cell 1

x coordinates [0.0 0.0 4.8 4.8]

y coordinates [6.0 3.0 3.0 6.0]

therefore the first gauss point will be

$x = 0.866 * 0.0 + 0.0646 * 0.0 + 0.0048 * 4.8 + 0.0646 * 4.8 = 0.333$

$y = 0.866 * 6.0 + 0.0646 * 3.0 + 0.0048 * 3.0 + 0.0646 * 6.0 = 5.7918$

$weight = 0.3479 * 0.3479 = 0.121$

Npsi1 = -0.4653

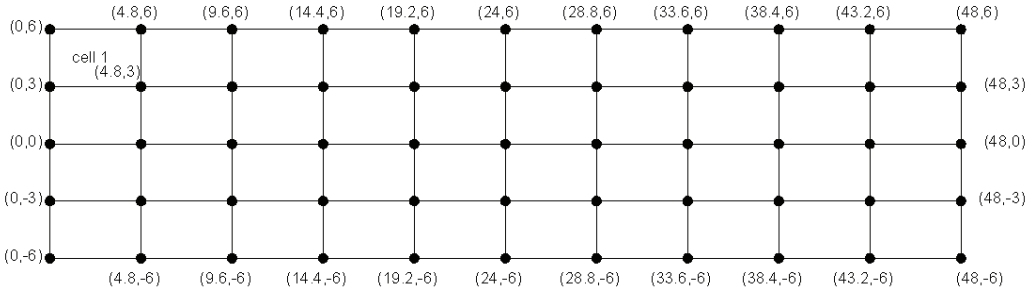


Figure 6.12: Gauss points generation

Npsi2 = 0.4653

Npsi3 = 0.0347

Npsi4 = -0.0347

Neta1 = -0.4653

Neta2 = -0.0347

Neta3 = 0.0347

Neta4 = 0.4653

$xpsi = -0.4653 * 0.0 + 0.4653 * 0 + 0.0347 * 4.8 + -0.0347 * 4.8 = 0.0$

$ypsi = -0.4653 * 6.0 + 0.4653 * 3.0 + 0.0347 * 3.0 + -0.0347 * 6.0 = -1.5$

$xeta = -0.4653 * 0.0 + -0.0347 * 0 + 0.0347 * 4.8 + 0.4653 * 4.8 = 2.4$

$yeta = -0.4653 * 6.0 + -0.0347 * 3.0 + 0.0347 * 3.0 + 0.4653 * 6.0 = 0.0$

$jcob = 3.6$

similarly all the 640 coordinates their corresponding weight and jacobian has to be found out.

f. Loop over gauss points to assemble discrete equations

(1) Determine nodes in neighborhood of gauss point

The size of domain of influence in the rectangular domain is (16.8, 10.5).

Nodes under the domain of influence are 1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19

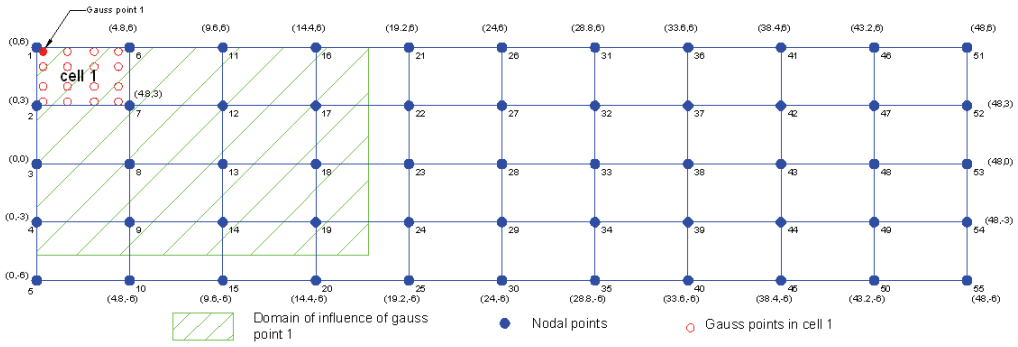


Figure 6.13: Fig. show the domain of influence of gauss point 1

(2) Find shape function, first derivative of shape function w.r.t x and first derivative of shape function w.r.t. y.

First find weight at each gauss point

find difference between gauss point and coordinates

find rx = absolute diff in x/ dmi

Form 1 to 8

[0.020 0.020 0.020 0.020

0.266 0.266 0.266 0.266]

from 9 to 16

[0.552 0.552 0.552 0.552

0.838 0.838 0.838 0.838]

find ry = absolute diff in y / dmi

from 1 to 8

[0.020 0.266 0.552 0.837

0.020 0.266 0.552 0.837]

from 9 1o 16

[0.020 0.266 0.552 0.837

0.020 0.266 0.552 0.837]

find drdx = sign of diff in x / dmi

from 1 to 8

[0.060 0.060 0.060 0.060

− 0.060 − 0.060 − 0.060 − 0.060]

from 9 to 16

[−0.060 − 0.060 − 0.060 − 0.060

− 0.060 − 0.060 − 0.060 − 0.060]

find drdy = sign of diff in y / dmi

from 1 to 8

[−0.095 0.095 0.095 0.095

− 0.095 0.095 0.095 0.095]

from 9 to 16

[−0.095 0.095 0.095 0.095

− 0.095 0.095 0.095 0.095]

form wx matrix

from 1 to 8

[0.6652 0.6652 0.6652 0.6652

0.4588 0.4588 0.4588 0.4588]

from 9 to 16

[0.1201 0.1201 0.1201 0.1201

0.0057 0.0057 0.0057 0.0057]

form dwx matrix

from 1 to 8

[−0.009 − 0.009 − 0.009 − 0.009

0.076 0.076 0.076 0.076]

from 9 to 16

[0.0480.0480.0480.048

0.0060.0060.0060.006]

similarly for wy and dwy

form wy matrix

from 1 to 8

[0.6651 0.4591 0.1202 0.0057

0.6651 0.4591 0.1202 0.0057]

from 9 to 16

[0.6651 0.4591 0.1202 0.0057

0.6651 0.4591 0.1202 0.0057]

form dwy matrix

from 1to 8

$[0.015\ -0.122\ -0.077\ -0.010$

$0.015\ -0.122\ -0.077\ -0.010]$

from 9 to 16

$[0.015\ -0.122\ -0.077\ -0.010$

$0.015\ -0.122\ -0.077\ -0.010]$

form w matrix

from 1 to 8

$[4.42E-01\ 3.05E-01\ 8.00E-02\ 3.82E-03$

$3.05E-01\ 2.11E-01\ 5.52E-02\ 2.63E-03]$

from 9 to 16

$[7.99E-02\ 5.51E-02\ 1.44E-02\ 6.89E-04$

$3.81E-03\ 2.63E-03\ 6.88E-04\ 3.28E-05]$

from 1 to 8

$[-6.04E-03\ -4.17E-03\ -1.09E-03\ -5.21E-05$

$5.06E-02\ 3.50E-02\ 9.15E-03\ 4.37E-04]$

from 9 to 16

$[3.18E-02\ 2.20E-02\ 5.75E-03\ 2.75E-04$

$4.18E-03\ 2.89E-03\ 7.56E-04\ 3.61E-05]$

from 1 to 8

$[9.75E-03\ -8.10E-02\ -5.09E-02\ -6.71E-03$

$6.73E-03\ -5.59E-02\ -3.51E-02\ -4.63E-03]$

from 9 to 16

$[1.76E-03 \ -1.46E-02 \ -9.20E-03 \ -1.21E-03$

$8.39E-05 \ -6.97E-04 \ -4.38E-04 \ -5.77E-05]$

p matrix = [1 x y] =

from 1 to 8

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 4.8 & 4.8 & 4.8 & 4.8 \\ 6 & 3 & 0 & -3 & 6 & 3 & 0 & -3 \end{bmatrix}$$

from 9 to 16

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9.6 & 9.6 & 9.6 & 9.6 & 14.4 & 14.4 & 14.4 & 14.4 \\ 6 & 3 & 0 & -3 & 6 & 3 & 0 & -3 \end{bmatrix}$$

Find B from p*{w,w,w}

from 1 to 8

$$\begin{bmatrix} 0.442 & 0.305 & 0.080 & 0.004 & 0.305 & 0.211 & 0.055 & 0.003 \\ 0.000 & 0.000 & 0.000 & 0.000 & 1.465 & 1.011 & 0.265 & 0.013 \\ 2.654 & 0.916 & 0.000 & -0.011 & 1.831 & 0.632 & 0.000 & -0.008 \end{bmatrix}$$

from 9 to 16

$$\begin{bmatrix} 0.080 & 0.055 & 0.014 & 0.001 & 0.004 & 0.003 & 0.001 & 0.000 \\ 0.767 & 0.529 & 0.139 & 0.007 & 0.055 & 0.038 & 0.010 & 0.000 \\ 0.479 & 0.165 & 0.000 & -0.002 & 0.023 & 0.008 & 0.000 & 0.000 \end{bmatrix}$$

loop over the no of nodes $pp = p * p^T$

$$\begin{bmatrix} 1 & 0 & 6 \\ 0 & 0 & 0 \\ 6 & 0 & 36 \end{bmatrix}$$

aa $= pp * w =$

$$\begin{bmatrix} 0.442 & 0 & 2.654 \\ 0 & 0 & 0 \\ 2.654 & 0 & 15.926 \end{bmatrix}$$

daax $= pp * dwdx =$

$$\begin{bmatrix} -0.006 & 0 & -0.036 \\ 0 & 0 & 0 \\ -0.036 & 0 & -0.217 \end{bmatrix}$$

daay $= pp * dwdy =$

$$\begin{bmatrix} 0.010 & 0.000 & 0.058 \\ 0.000 & 0.000 & 0.000 \\ 0.058 & 0.000 & 0.351 \end{bmatrix}$$

loop ends here

aa matrix

$$\begin{bmatrix} 1.563 & 4.301 & 6.689 \\ 4.301 & 28.563 & 18.409 \\ 6.689 & 18.409 & 35.165 \end{bmatrix}$$

similarly daax and daay matrix are computed

for each gauss point the shape function is

pg = [1, gauss point x co-ord., gauss point y co-ord.] therefore for gauss point 1 it will be pg=[1 0.333 5.7918]

$phi = pg * inv(aa) * B = 0.635$

Find first derivative of inv aa w.r.t. x and w.r.t. y from

$daainvx = -inv(aa) * daax * inv(aa)$

$daainvy = -inv(aa) * daay * inv(aa)$

To find first derivative of shape function w.r.t. x is

$dbx = dwx * B$

$dphix = [010] * inv(aa) * B + pg * daainvx * B + pg * inv(aa) * dbx$

To find first derivative of shape function w.r.t. y is

$dby = dwy*B$ $dphiy = [001]*inv(aa)*B+pg*daainvy*B+pg*inv(aa)*dby$

(3) Find Bmat for each node as per

$$\mathrm{B} = \begin{bmatrix} \text{diphix} & 0 \\ 0 & \text{dphiy} \\ \text{dphiy} & \text{dphix} \end{bmatrix}$$

(4) K matrix at = weight*jac*(BmatT*Dmat*Bmat)

g. Determine nodes on boundary, set up boundary conditions.

h. setup gauss point along traction boundary same as done above

i. Integrate GG matrix and q vector along displacement boundary

j. Enforce bc's using lagrange multipliers

$$\begin{bmatrix} \mathrm{K} & \mathrm{GG} \\ GG^T & 0 \end{bmatrix} \begin{Bmatrix} \mathrm{f} \\ \mathrm{q} \end{Bmatrix} = \begin{Bmatrix} \mathrm{u} \\ \lambda \end{Bmatrix}$$

$m * d = F$

k. Solve for output variables i.e. displacements

$d = m^{-1} * F$

Separate u and λ.

l. Solve for stresses at gauss points.

m. Plot the results

The exact solution of this problem (Timoshenko and Goodier, 1977) is displacement in the x direction,

$$u_x(x,y) = -\frac{Py}{6EI}\left[(6L-3x)\,x + (2+\mu)\left(y^2 - \frac{D^2}{4}\right)\right] \tag{6.1}$$

The displacement in the y direction,

$$u_y(x,y) = \frac{Py}{6EI}\left[3\mu y^2(L-x) + (4+5\mu)\frac{D^2 x}{4} + (3L-x)\,x^2\right] \tag{6.2}$$

The stresses in the beam

$$\sigma_x(x,y) = -\frac{P(L-x)\,y}{I}, \qquad \sigma_{xy}(x,y) = -\frac{P}{2I}\left(\frac{D^2}{4} - y^2\right), \qquad \sigma_y(x,y) = 0 \tag{6.3}$$

Where $I = D^3/12$ is the moment of inertia (second moment of area).
In Table 6.6, the vertical displacement at point (L, 0) calculated by EFG is compared with the exact solution. This table shows excellent agreement -between EFG and the analytical solution. Fig. 6.6 shows distribution of normal stress in the beam. Fig. 6.6 shows the normal stress (σ_{xx}) distribution at dist. L/2 for various dmax values. Fig. 6.6 shows the shear stress (σ_{xy}) distribution at dist. L/2 for various dmax values. From both the figures, it has been noticed that as the value of dmax increases, accuracy also increases. For dmax equal to 3 to 3.5, exact results have been achieved.

Dmax	Nodes	uy exact	uy EFG	% error
1.5	8 x 3	-0.0089	-0.00840	5.60
2	8 x 3	-0.0089	-0.00877	1.45
2.5	8 x 3	-0.0089	-0.00885	0.52
3	8 x 3	-0.0089	-0.00887	0.28
3.5	8 x 3	-0.0089	-0.00892	-0.20

Table 6.6: Comparison of vertical displacement at end

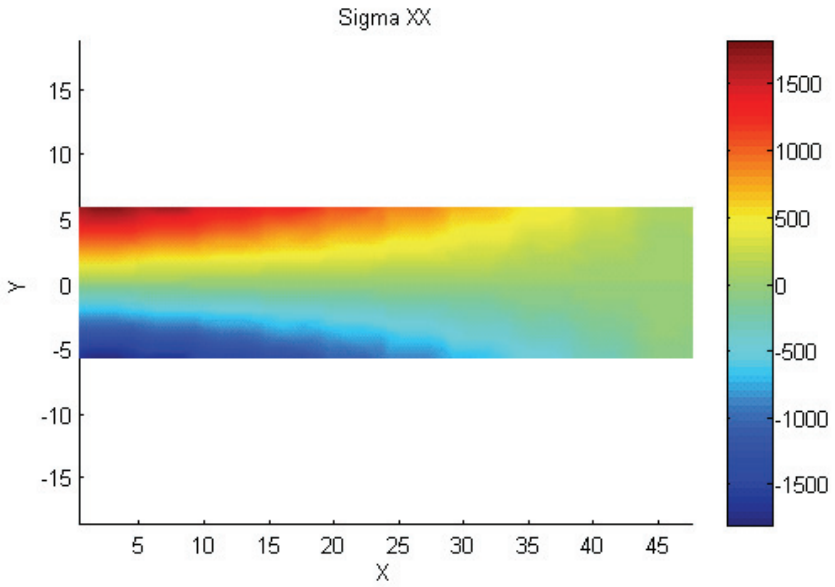


Figure 6.14: Normal stress (σ_{xx}) distribution

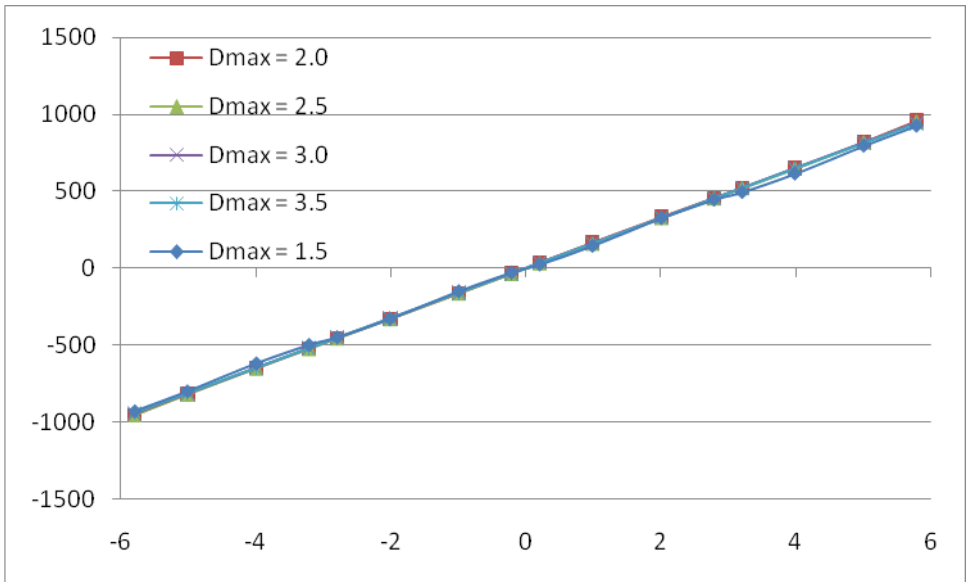


Figure 6.15: Distribution of normal stress (σ_{xx}) on the section of x = L/2 of the cantilever beam.

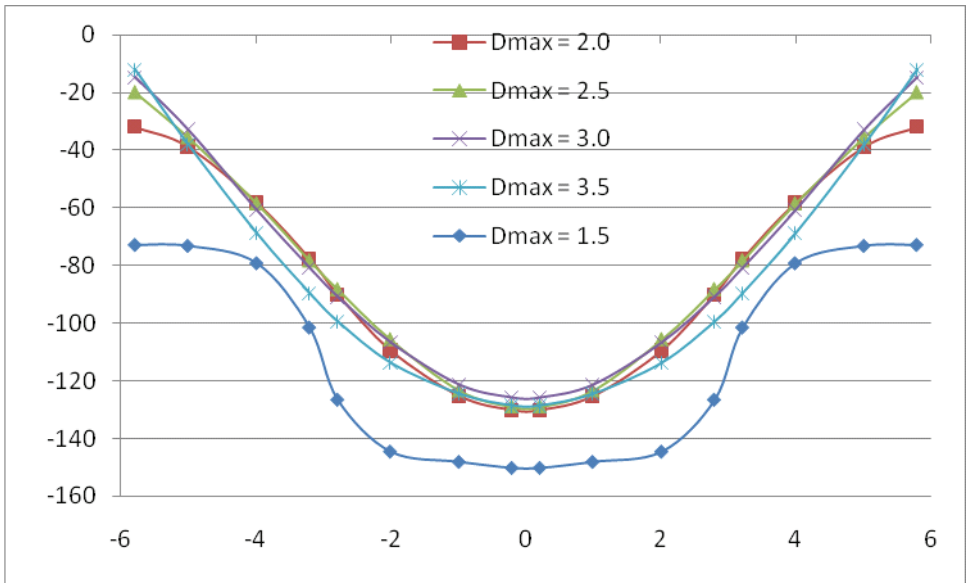


Figure 6.16: Distribution of shear stress(σ_{xy}) on the section of x = L/2 of the cantilever beam

dmax	Nodes	uy exact	uy EFG	% error
1.5	8 x 3	-0.0089	-0.00841	5.55
2	8 x 3	-0.0089	-0.00875	1.69
2.5	8 x 3	-0.0089	-0.00882	0.95
3	8 x 3	-0.0089	-0.00884	0.67
3.5	8 x 3	-0.0089	-0.00885	0.55

Table 6.7: Comparison of vertical displacement at end by penalty method

6.7 Timoshenko beam by Penalty method

The analysis procedure of Timoshenko beam by the penalty method is same as the previous method (Lagrange method) except for the introduction of boundary conditions. Thus, computation of nodal stiffness matrix and nodal load vector is same. The changes to be incorporated have been described in section 5.5 for introduction of boundary conditions by penalty method. The data of the problem is same as taken in section 6.5. Moreover, in this method the value of penalty parameter (α) has to be included. Accuracy of the results varies on two factors dmax and penalty parameter. For this particular case, number of trials varying α was taken, and it was decided to take $\alpha = 2.3\ x\ 10^8$.
Subsequently keeping $\alpha = 2.3\ x\ 10^8$, 24 nodes dmax was varied, results are tabulated in Table 6.4.

α=$\mathbf{2.3 * 10^8}$ It is observed that the displacement at dmax=3.5 and $\alpha = 2.3\ x\ 10^8$, matches very well with the exact results given by Eqn.6.2. Similarly for 55 nodes dmax =3.5 and $\alpha = 3.3\ x\ 10^8$ %error is 0.18%. Fig. 6.7 shows the normal stress (σ_{xx}) distribution at dist. L/2 for various dmax values. Fig. 6.7 shows the shear stress (σ_{xy}) distribution at dist. L/2 for various dmax values.

From Figs. 6.7 and 6.7 it is observed that penalty method shows the same result as obtained by Lagrange method of Figs. 6.6 and 6.6. Only difficulty with penalty method is to choose the penalty parameter which varies with no. of nodes.

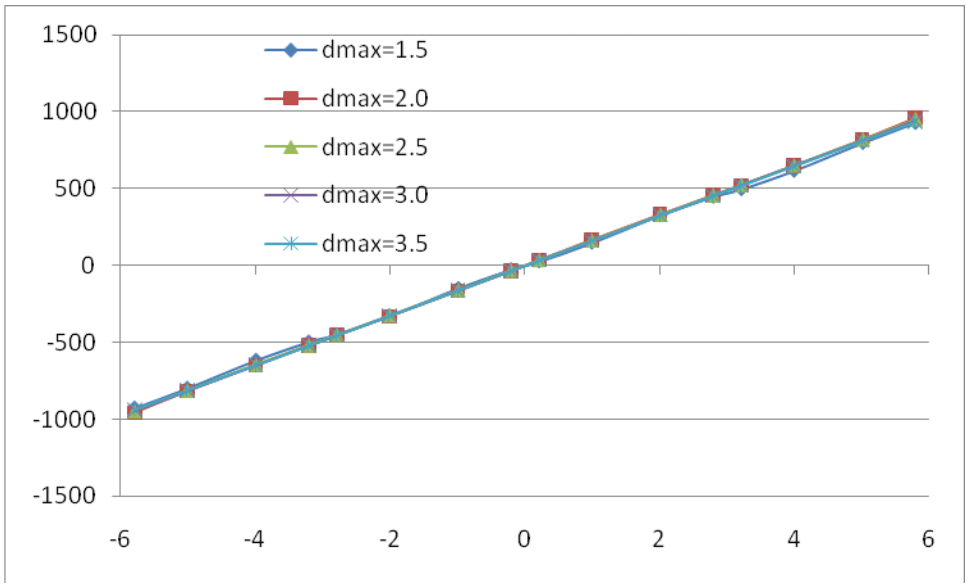


Figure 6.17: Distribution of normal stress σ_{xx} at x = L/2 of the cantilever beam

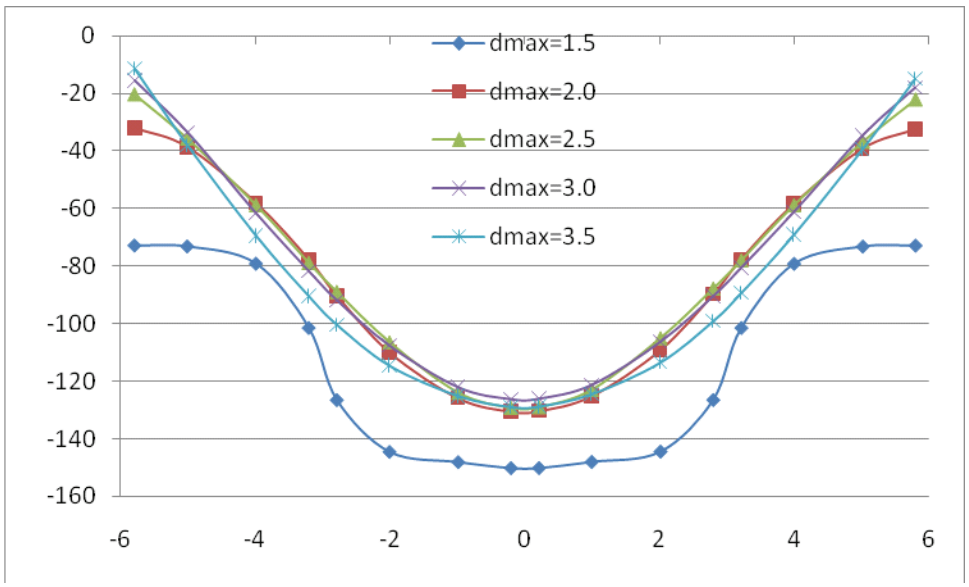


Figure 6.18: Distribution of shear stress σ_{xy} at x = L/2 of the cantilever beam.

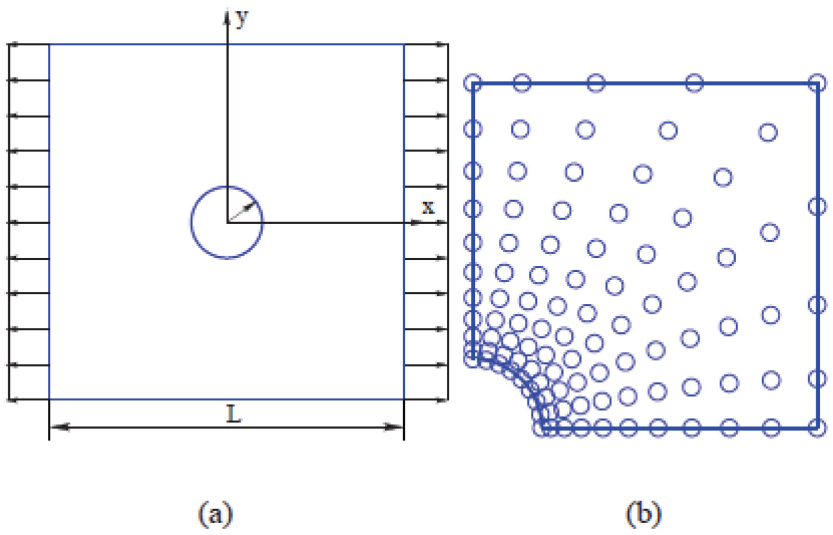


Figure 6.19: The plate with hole : (a) Whole domain; (b) 1/4 model with irregular nodes

6.8 Plate with hole

Consider an infinite plate with a centered hole under unidirectional tension along x direction. The plate dimension is taken to be of L x L and the circle radius is of a. The exact solution is given by,

$$\sigma_x(r,\theta) = 1 - \frac{a^2}{r^2}\left(\frac{3}{2}\cos 2\theta + \cos 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \tag{6.4}$$

$$\sigma_y(r,\theta) = -\frac{a^2}{r^2}\left(\frac{1}{2}\cos 2\theta - \cos 4\theta\right) - \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \tag{6.5}$$

$$\sigma_{xy}(r,\theta) = -\frac{a^2}{r^2}\left(\frac{1}{2}\sin 2\theta + \sin 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\sin 4\theta \tag{6.6}$$

where r, θ are the usual polar coordinates centered at the center of the hole. The boundary conditions include (i) essential boundary conditions on the bottom $uy(x,\ y = 0)$ and left edges $(ux(x = 0,\ x = L,\ y) = 0)$; (ii) natural boundary conditions on

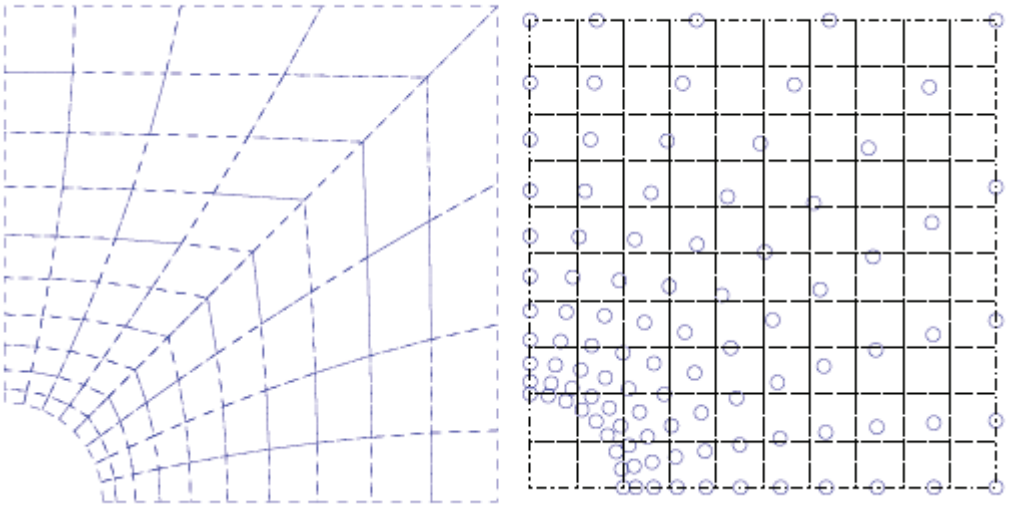


Figure 6.20: Gauss points: (a) Background mesh; (b) Structure cells

the right and top edges on which traction $\mathbf{t}$ computed from the exact stress given in Eqn. 6.4 are applied. In the computation, the material properties are taken as Young modulus of 103, Poisson's ratio equal to 0.3, whereas $L = 10$, $a = 1$. The background mesh used in the numerical integration is given in Fig. 6.8 (a). Another possibility is the structure cell as shown in 6.8 (b). However, the background mesh is chosen because it is better and in addition it is also the finite element mesh from which the particles are extracted. For each background mesh, a $4x4$ Gauss quadrature is employed in fig. 6.22. Concerning weight functions, the cubic spline function is used. The domain of influence for nodes is a circle with radius varied from nodes to nodes. They are chosen such that the support is small for nodes near the hole and bigger for ones near the edges. 6.8 The essential boundary condition is imposed with the boundary point collocation method where collocation points are coincident with nodes along the bottom and left edges. The stress σ_{xx} computed at nodes are plot and compared to the exact solution Fig. 6.8. With a coarse discretization of 99 nodes, results obtained by EFG show good correlation with those of exact results.

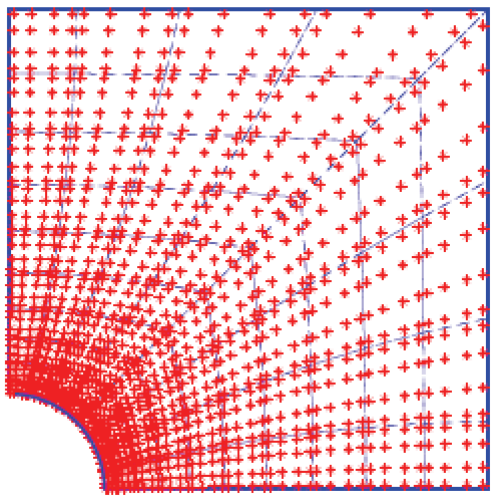
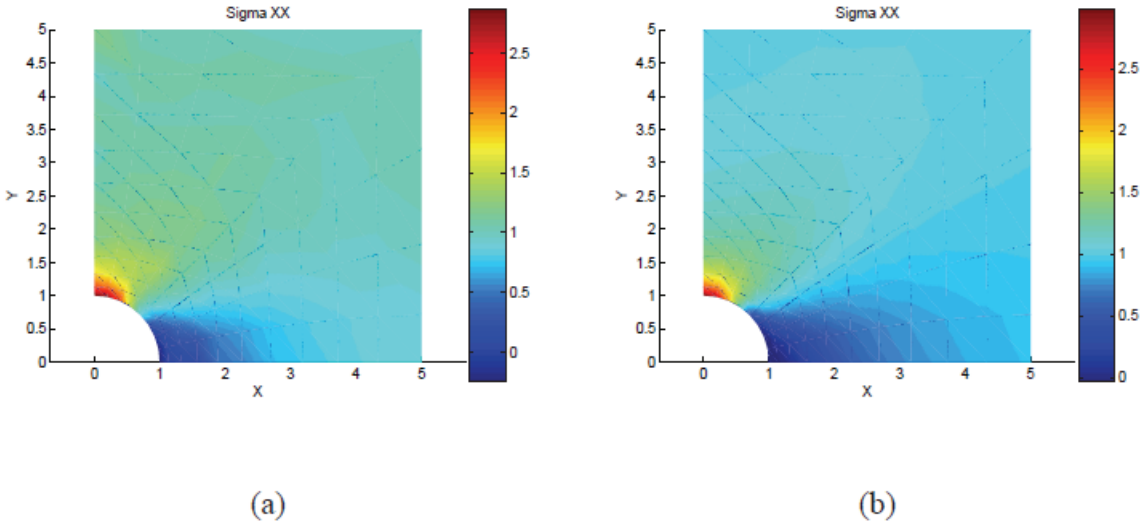


Figure 6.21: Gauss points distribution: 4x4 Gauss quadrature for each sub cell



(a) (b)

Figure 6.22: Stress plot: (a) EFG σ_{xx} stress (b) Exact σ_{xx} stress

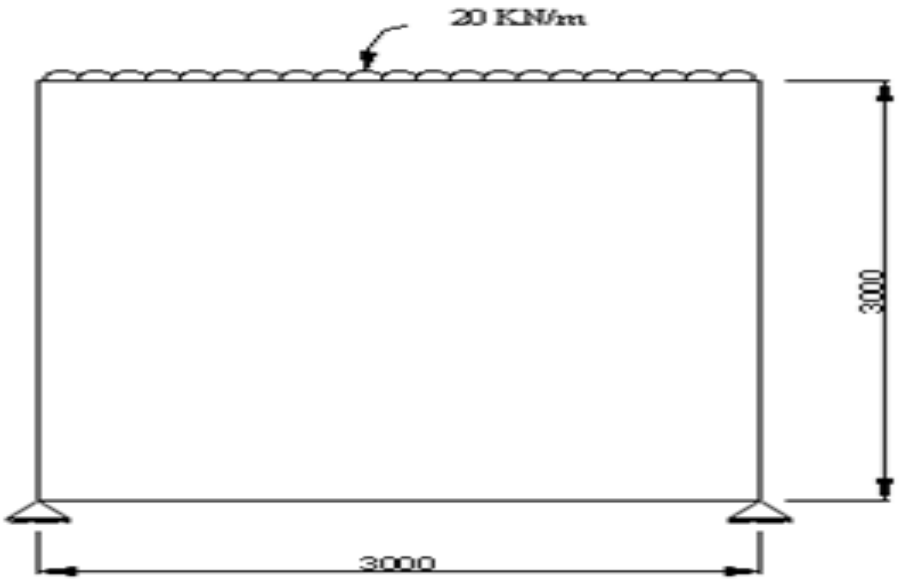


Figure 6.23: Deep beam

6.9 Deep beam problem

Consider a deep beam simply supported subjected to u.d.l. over its entire span in y direction. The dimension of beam is *3m x 3m,* thickness *0.2m,* Young's modulus *2 x 105 kN/m2* and Poisson's ratio *0.3.* This is treated as plane stress problem. Results of this beam are plotted in the fig. 6.9 for σ_{xx}, σ_{yy} and σ_{xy} the results are compared with results given by Krishnamoorthy [14], the same beam is analyzed by FEM by Krishnamoorthy[14] by taking Iso-parametric element and a graph of shear stress has been plotted. The comparison of results of Krishnamoorthy[14], has been done with results obtained from C program V3, comparison is shown in fig. 6.9 which shows that the analysis model is reliable.

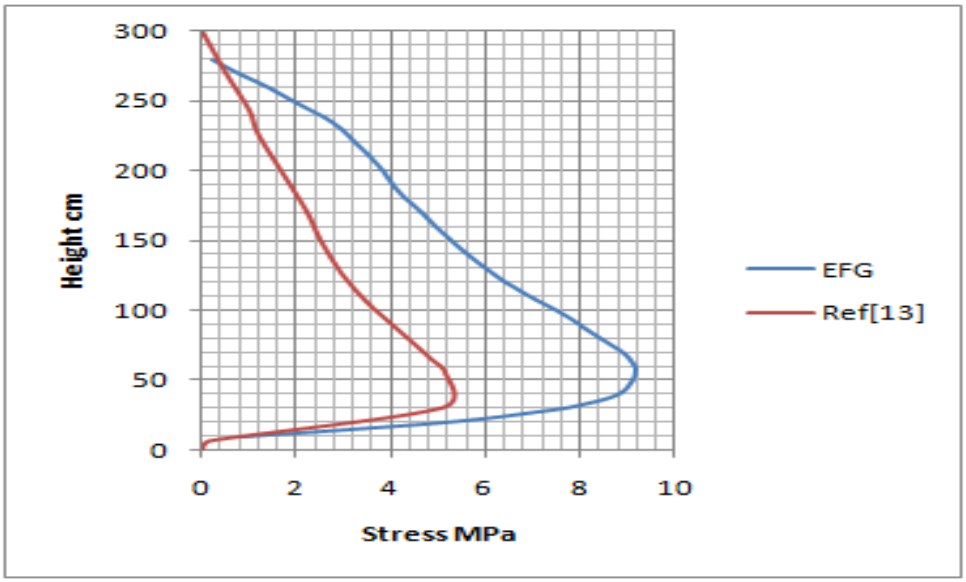


Figure 6.24: Shear stress distribution in deep beam at dist.

Chapter 7

CONCLUSIONS, SUMMARY AND FUTURE SCOPE

7.1 Conclusions

From study of each of the above problems some of the conclusions have been derived as follows:

- For one dimensional bar problem, there is no need for the connection of elements. This is valid when the bar is idealized as a 2-nodded or 3-nodded FEM bar element.
- The results for one dimensional bar problem, converge reasonably well with 0.81 %error for 11 nodes.
- For one dimensional beam problem, the analysis of cantilever beam with uniformly distributed load show that, the number of gauss points are twice the number of coordinate points. The results show very good convergence, for 10 nodes, error percentage is 1.15%.
- The results of a cantilever beam with parabolic shear load at the end shows a very good correlation with the exact analytical solution given by Timoshenko.

- The analysis results of deep beam are compared with those given by Krishnamoorthy [14] and are satisfactory.

- The analysis results of the plate with hole in center, under tensile loading are compared with exact analytical solutions and the results are matching to the tune of 2 % variation.

- While applying the EFG approach by using Lagrange multiplier, some difficulties encountered are as follows:

- The size of the stiffness matrix increases with the incorporation of the extra matrix G and G^T.

- The stiffness matrix which has to be inverted, posses a zero value on its main diagonal, which is very difficult to invert, because of the fact that several limitations creep in Lagrangian multiplier approach as it uses Lagrangian multipliers for the enforcement of essential boundary conditions. Due to these multipliers the bandedness of the system is sacrificed due to appearance of G and G^T as mentioned above. Therefore, to handle these issues MATLAB has been used which has built-in solvers viz, with variable bandwidth which do not take advantage of positive definiteness of the matrix as suggested by (Lu et al. 1994) [28].

- These difficulties of Lagrange multiplier have been resolved by using penalty method.

- Penalty method has been also used for structural analysis of 2D plane stress problems. Some observations from running the program are enumerated as follows:

  a. The accuracy of displacement depends upon value of penalty parameter.

  b. If penalty parameter is taken too high the numerical difficulties are faced.

c. For a particular problem, penalty parameter depends upon material property (Young's modulus) and members on the diagonal of nodal stiffness matrix [18].

7.2 Summary

Chapter 1, presents the introduction and overview of the mesh free approach. It states various difficulties faced in various analysis tools available in market and suitability of mesh free approach for it. It gives objective of study and scope of work of major project.

Chapter 2, presents the literature review of various mesh free methods. Detailed review of EFG method is carried out, so as to apply the method for practical analysis. It highlights various problems attempted and their suitability studied by various authors.

Chapter 3, presents the introduction part of the mesh free approach. It gives understanding about various engineering mechanics problems present in practice. The chapter presents solution procedure followed for analysis of problem by FEM and Mesh free method. It also gives understanding of various terminologies used for mesh free methods. The chapter gives list of various mesh free methods in table form with what type of system of equation it has used and which approximation method has been used for shape function creation.

Various methods used for creation of shape function in mesh free methods are presented in chapter 4. It gives procedure for creation of shape function for one dimensional and two dimensional problems by Moving Least Square method.

Chapter 5, Element Free Galerkin (EFG) method, presents the theory and procedure of the EFG method. Chapter also presents the analysis steps to be followed for one dimensional bar problem, one dimensional beam problem and two dimensional beam problems.

Chapter 6, presents the applications of EFG method to various structural engineer-

ing problems. Analysis of one dimensional bar has been given with sample problems. Analysis of one dimensional beam problem has been given with sample problems. A cantilever beam with parabolic end load is analyzed and studied by parametric studies. Analysis of plate in tension with circular hole in center has been done. Analysis of deep beam problem is done and results have been compared with those given by Krishnamoorthy[14].

Chapter 7 contains the conclusions, summary and future scope of work.

7.3 Future scope

From the experiences of the present study, the future scope of the thesis can be envisaged as follows:

- Detailed study of suitability of various equation solvers and introduction of boundary conditions
- Other applications such as, plate bending and shells using EFG method
- Crack propagation in two dimensional problems by EFG method
- Comparison of various mesh free methods by considering suitable complex structural engineering problems.

Appendix A

1D EFG Bar Program (V1)

```
/*****1D BAR EFG PROGRAM*****/
#include<stdio.h>
#include<conio.h>
#include<math.h>
float detrm(float[50][50],float);
void cofact(float[50][50],float);
void trans(float[50][50],float[50][50],float);
double inv[50][50];
FILE *f1, *f2;
void main()
{
    int nnd, nc, i, j, k, l, gpt, weight, lnd;
    float x[50], le, dmax, E, dm[50], gg[50], jac, K[50][50], f[50], GG[50], xg;
    float dif[50], r[50],drdx[50],we[50],dw[50],B[3][50],p[50][3], A[3][3], dA[3][3];
    float pp[3][3], Ainv[3][3], fac, pd[3], phi[50], Aib[3][50], db[3][50], dphi[50];
float LM[50][50], LMi[50][50], da[3][3], daa[3][3], zone[3], FM[50], fbody, q;
float area;
    double d[50], dit;
    f1=fopen("1dpro.in","r"); /*opening input file*/
```

```
    f2=fopen("1dproout.out","w"); /*opening output file*/
/*********SET UP NODAL COORDINATES ALONG BAR, DETERMINE NUM-
BER OF CELLS*******/
    fscanf(f1,"%f %d %d",&le,&nnd,&gpt);
    fprintf(f2,"Length of bar:\t%f\n",le);
    fprintf(f2,"No. of nodes:\t%d\n",nnd);
    nc= nnd-1;
    fprintf(f2,"No. of cells:\t%d\n",nc);
/*******SET PARAMETERS FOR WEIGHT FUNCTION, MATERIAL PROPERITES*******
    fscanf(f1,"%f %f %f ", &dmax, &E, &area);
    fprintf(f2,"dmax=\t%5.2f\nelasticity=\t%5.2f\narea=\t%5.2f\n", dmax, E, area);
for(i=0;i<nnd;i++) /*node data input*/
    {
    if(i==0)
    x[i]=0.0;
else
    x[i]=(x[i-1]+(le/nc));
fprintf(f2,"x[%d]\t",i);
}
fprintf(f2,"\n");
for(i=0;i<nnd;i++)
    {
    fprintf(f2,"%5.3f\t",x[i]);
}
fprintf(f2,"\n");
/***********DETERMINE DMI FOR EACH NODE***********/
fprintf(f2,"dm= dmax * (diff in x)\n");
for(i=0;i<nnd;i++)
{
```

```
    dm[i]=dmax*(x[2]-x[1]);
fprintf(f2,"dm[%d]\t",i);
    }
fprintf(f2,"\n");
for(i=0;i<nnd;i++)
{
    fprintf(f2,"%6.3f\t",dm[i]);
}
fprintf(f2,"\n");
/**********SET UP GAUSS POINTS, WEIGHTS, AND JACOBIAN FOR EACH
CELL********/
for(i=0;i<gpt;i++)
{
    gg[i]=0.0;
}
jac=(x[2]-x[1])/2;
weight=2;
fprintf(f2,"weight=\t%d \t Jac=\t%5.2f\n",weight,jac);
fprintf(f2,"gauss points where the integration is done\n");
for(i=0;i<gpt;i++)
{
    if(i==0)
    gg[i]=0.0;
    else
    gg[i]=(x[i]+x[i-1])/2.0;
fprintf(f2,"gg[%d]\t",i);
}
fprintf(f2,"\n");
for(i=0;i<gpt;i++)
```

```
{
    fprintf(f2,"%6.3f\t",gg[i]);
}
fprintf(f2,"\n");
for(i=0;i<nnd;i++)
{
f[i]=0.0;
GG[i]=0.0;
dw[i]=0.0;
we[i]=0.0;
for(k=0;k<2;k++)
    {
    B[k][i]=0.0;
}
for(j=0;j<nnd;j++)
{
    K[i][j]=0.0;
}
}
/*************LOOP OVER GAUSS POINTS*********/
for(i=0;i<gpt;i++)
{
fprintf(f2,"The Gauss point %d\ndiff = gg - x\n",i);
    xg=gg[i];
/*******DETERMINE DISTANCE BETWEEN NODES AND GAUSS POINT*******/
for(j=0;j<nnd;j++)
{
    dif[j]=xg-x[j];
fprintf(f2,"dif[%d]\t",j);
```

```
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
{
    fprintf(f2,"%6.3f\t",dif[j]);
}
fprintf(f2,"\n");
/***************SET UP WEIGHTS W AND DW FOR EACH NODE************/
fprintf(f2,"r= fabs(dif) / dm\n");
for(j=0;j<nnd;j++)
{
    fprintf(f2,"r[%d]\t",j);
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
{
if(dm[j]==0.0)
    r[j]=0.0;
else
    r[j]=(fabs(dif[j]))/dm[j];
fprintf(f2,"%5.2f\t",r[j]);
if(dif[j]==0.0 ||dm[j]==0.0)
    drdx[j]=0.0;
else
    drdx[j]=fabs(dif[j])/(dm[j]*dif[j]);
if(r[j]<=0.5)
{
    we[j]=(2.0/3.0)+(-4.0*r[j]*r[j])+(4.0*pow(r[j],3));
dw[j]=(-8.0*r[j]+12.0*r[j]*r[j])*drdx[j];
```

```
}
else
{
    if(0.5<r[j] && r[j]<=1.0)
{
    we[j]=(4.0/3.0)-4.0*r[j]+4.0*r[j]*r[j]-(4.0/3.0)*pow(r[j],3);
dw[j]=(-4.0+8.0*r[j]-4.0*r[j]*r[j])*drdx[j];
}
else
{
    we[j]=0.0;
dw[j]=0.0;
}
}
}
fprintf(f2,"\nThe weight function w:\n");
for(j=0;j<nnd;j++)
{
    fprintf(f2,"w[%d]\t",j);
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
{
    fprintf(f2,"%5.2f\t",we[j]);
}
fprintf(f2,"\n");
fprintf(f2,"drdx\n");
for(j=0;j<nnd;j++)
{
```

```
    fprintf(f2,"%5.2f\t",drdx[j]);
}
fprintf(f2,"\nThe first derivative of weight function: dw\n");
for(j=0;j<nnd;j++)
{
fprintf(f2,"%5.2f\t",dw[j]);
p[j][0]=1.0;
p[j][1]=x[j];
}
fprintf(f2,"\nP matrix \n");
    for(j=0;j<nnd;j++)
    {
    fprintf(f2,"1\t");
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
    {
    fprintf(f2,"x%d\t",j);
}
fprintf(f2,"\n");
for(k=0;k<2;k++)
{
    for(j=0;j<nnd;j++)
    {
    fprintf(f2,"%5.2f\t",p[j][k]);
}
fprintf(f2,"\n");
}
/*********SET UP SHAPE FUNCTIONS AND DERIVATIVES**********/
```

```
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
{
    for(k=0;k<2;k++)
{
    if(k==0)
    B[k][j]=(1.0*we[j]);
else
    B[k][j]=(x[j]*we[j]);
}
}
fprintf(f2,"The matrix B is:\n");
for(j=0;j<nnd;j++)
    {
    fprintf(f2,"1*w%d\t",j);
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
    {
    fprintf(f2,"x%d*w%d\t",j,j);
}
fprintf(f2,"\n");
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",B[j][k]);
}
fprintf(f2,"\n");
```

```
}
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    A[j][k]=0.0;
dA[j][k]=0.0;
}
}
for(j=0;j<nnd;j++)
{
    for(k=0;k<2;k++)
{
    for(l=0;l<2;l++)
{
    pp[k][l]=p[j][k]*p[j][l];
A[k][l]=A[k][l]+we[j]*pp[k][l];
dA[k][l]=dA[k][l]+dw[j]*pp[k][l];
}
}
}
fprintf(f2,"The matrix A = A + w*p'*p\n");
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    fprintf(f2,"%6.3f\t",A[j][k]);
}
fprintf(f2,"\n");
```

```
}
fprintf(f2,"The matrix dA = dA + dw*p'*p\n");
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    fprintf(f2,"%6.3f\t",dA[j][k]);
}
fprintf(f2,"\n");
}
fac=(A[0][0]*A[1][1])-(A[0][1]*A[1][0]);
Ainv[0][0]=(A[1][1])/fabs(fac);
Ainv[1][1]=(A[0][0])/fabs(fac);
Ainv[0][1]=(-A[1][0])/fabs(fac);
Ainv[1][0]=(-A[0][1])/fabs(fac);
fprintf(f2,"The inverse of matrix A is:\n");
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    fprintf(f2,"%6.3f\t",Ainv[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<nnd;j++)
{
    for(k=0;k<2;k++)
{
    Aib[j][k]=0.0;
```

```
}
}
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
    for(l=0;l<2;l++)
{
    Aib[j][k]=Aib[j][k]+(Ainv[j][l]*B[l][k]);
}
}
}
/*fprintf(f2,"The Matrix Aib is:\n");
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",Aib[j][k]);
}
fprintf(f2,"\n");
} */
for(k=0;k<nnd;k++)
{
    phi[k]=(0.0);;
}
fprintf(f2,"The Matrix Phi = [1 gg]*(Ainv * B)\n");
pd[0]=1.0;
pd[1]=(xg);
//fprintf(f2,"%f\t %f\t %f\n",pd[0],pd[1],xg);
```

```
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
    for(l=0;l<2;l++)
{
    phi[k]=phi[k]+pd[j]*(Ainv[j][l]*B[l][k]);
}
}
}
for(k=0;k<nnd;k++)
{
    fprintf(f2,"phi[%d]\t",k);
}
fprintf(f2,"\n");
for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",phi[k]);
}
/*for(k=0;k<nnd;k++)
{
    phi[k]=0.0;
}
for(k=0;k<nnd;k++)
{
    for(l=0;l<2;l++)
{
    phi[k]=phi[k]+(pd[l]*Aib[l][k]);
}
```

```
}
fprintf(f2,"The Matrix phi is:\n");
for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",phi[k]);
}*/
fprintf(f2,"\n");
fprintf(f2,"\nThe Matrix db is:\n");
for(j=0;j<nnd;j++)
    {
    fprintf(f2,"1*dw%d\t",j);
}
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
    {
    fprintf(f2,"x%d*dw%d\t",j,j);
}
fprintf(f2,"\n");
for(k=0;k<nnd;k++)
{
db[0][k]=1.0*dw[k];
    db[1][k]=x[k]*dw[k];
}
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
fprintf(f2,"%6.4f\t ",db[j][k]);
}
```

```
fprintf(f2,"\n");
}
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    daa[j][k]=0.0;
da[j][k]=0.0;
}
}
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    for(l=0;l<2;l++)
{
    daa[j][k]=daa[j][k]+dA[j][l]*Ainv[l][k];
}
}
}
fprintf(f2,"\nThe Matrix da = -Ainv * dA * Ainv\n");
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    for(l=0;l<2;l++)
{
    da[j][k]=da[j][k]-(Ainv[j][l])*daa[l][k];
}
```

```
}
}
for(j=0;j<2;j++)
{
    for(k=0;k<2;k++)
{
    fprintf(f2,"%6.3e\t",da[j][k]);
}
}
for(k=0;k<nnd;k++)
{
    dphi[k]=0.0;
}
fprintf(f2,"\nThe Matrix dphi is: dphi = [0 1]*Ainv*B+[1 gg]*(da*B+Ainv*db)\n");
zone[0]=0.0;
zone[1]=1.0;
for(j=0;j<2;j++)
{
    for(k=0;k<nnd;k++)
{
    for(l=0;l<2;l++)
{
    dphi[k] = dphi[k]+(zone[j]*(Ainv[j][l]*B[l][k]))+(pd[j] * (da[j][l]*B[l][k]+Ainv[j][l]*db[l][k]));
}
}
}
for(j=0;j<nnd;j++)
{
fprintf(f2,"%6.3e\t",dphi[j]);
```

```
    }
fprintf(f2,"\nThe matrix GG = -phi\n");
/**************ASSEMBLE DISCRETE EQUATIONS***************/
if(i==0)
{
    for(j=0;j<nnd;j++)
{
    GG[j]=(-phi[j]);
}
}
else
{
    for(j=0;j<nnd;j++)
{
    for(k=0;k<nnd;k++)
{
K[j][k]=K[j][k]+(weight*E*area*jac)*(dphi[j]*dphi[k]);
}
}
fbody=0.0;
    for(j=0;j<nnd;j++)
    {
    f[j]=f[j]+(weight*fbody*jac)*phi[j];
    }
}
f[5]=200;
for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3e\t",GG[k]);
```

```
}
fprintf(f2,"\nK=(w*E*area*jac)*(dphi'*dphi)\n");
for(j=0;j<nnd;j++)
{
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3e\t",K[j][k]);
}
fprintf(f2,"\n");
}
fprintf(f2,"\nforce vector\n");
for(k=0;k<nnd;k++)
{
    fprintf(f2,"f[%d]=\t%6.3e\n",k,f[k]);
}
fprintf(f2,"\n");
}
lnd=nnd+1;
/*******ENFORCE BOUNDARY CONDITIONS USING LAGRANGE MULTIPLI-
ERS*******/
fprintf(f2,"ENFORCE BOUNDARY CONDITIONS USING LAGRANGE MULTI-
PLIERS\n");
fprintf(f2,"K\t G\t \t u\t \t f\n G'\t 0\t \t lam\t \t q\n");
for(j=0;j<lnd;j++)
{
    for(k=0;k<lnd;k++)
{
    LM[j][k]=0.0;
}
```

```
d[j]=0.0;
}
for(j=0;j<lnd;j++)
{
    for(k=0;k<lnd;k++)
{
    if(j<nnd && k<nnd)
    LM[j][k]=K[j][k];
else
    if(j==lnd-1 && k<lnd-1)
    LM[j][k]=GG[k];
else
    if(k==lnd-1 && j<lnd-1)
    LM[j][k]=GG[j];
else
    LM[j][k]=0.0;
}
}
for(j=0;j<lnd;j++)
{
    for(k=0;k<lnd;k++)
{
    fprintf(f2,"%6.3f\t",LM[j][k]);
}
fprintf(f2,"\n");
}
q=0.0;
for(j=0;j<lnd;j++)
{
```

```
if(j<nnd)
    FM[j]=f[j];
else
    FM[j]=q;
}
for(k=0;k<lnd;k++)
{
    fprintf(f2,"FM[%d]=\t%6.3f \n",k,FM[k]);
}
//ff=lnd;
/************SOLVE FOR NODAL PARAMETERS***********/
/*********SOLUTION OF EQUATION LMinv*FM ***********/
dit=detrm(LM,lnd);
    fprintf(f2,"THE DETERMINANT IS=%f",dit);
    if(dit==0)
    fprintf(f2,"\nMATRIX IS NOT INVERSIBLE\n");
else
    cofact(LM,lnd);
fprintf(f2,"\nTHE INVERSE OF THE MATRIX:\n");
    for(i=0;i<lnd;i++)
    {
    for(j=0;j<lnd;j++)
    {
LMi[i][j]=inv[i][j];
    fprintf(f2,"%5.3f\t",LMi[i][j]);
}
fprintf(f2,"\n");
}
for(i=0;i<lnd;i++)
```

```
    {
    for(j=0;j<lnd;j++)
    {
d[i]=d[i]+LMi[i][j]*FM[j];
}
}
for(k=0;k<lnd;k++)
{
    fprintf(f2,"d[%d]=\t%6.2f \n",k,d[k]);
}
/*******FUNCTION TO FIND THE DETERMINANT OF THE MATRIX**********/
float detrm(float a[50][50],float k)
{
    float s=1,det=0,b[50][50];
    int i,j,m,n,c;
if(k==1)
    return(a[0][0]);
else
{
    det=0;
for(c=0;c<k;c++)
{
    m=0;
n=0;
for(i=0;i<k;i++)
{
    for(j=0;j<k;j++)
{
    b[i][j]=0;
```

```
if(i!=0&&j!=c)
{
    b[m][n]=a[i][j];
if(n<(k-2))
n++;
else
{
    n=0;
m++;
}
}
}
}
det=det+s*(a[0][c]*detrm(b,k-1));
s=-1*s;
}
}
return(det);
}
/********FUNCTION TO FIND COFACTOR***********/
void cofact(float num[50][50],float f)
{
    float b[50][50],fac[50][50];
int p,q,m,n,i,j;
for(q=0;q<f;q++)
{
    for(p=0;p<f;p++)
{
    m=0;
```

```
n=0;
for(i=0;i<f;i++)
{
    for(j=0;j<f;j++)
{
    b[i][j]=0;
if(i!=q&&j!=p)
{
    b[m][n]=num[i][j];
if(n<(f-2))
    n++;
else
{
    n=0;
m++;
}
}
}
}
fac[q][p]=pow(-1,q+p)*detrm(b,f-1);
}
}
trans(num,fac,f);
}
/******FUNCTION TO FIND TRANSPOSE AND INVERSE OF A MATRIX********/
void trans(float num[50][50],float fac[50][50],float r)
{
    int i,j;
float b[50][50],d;
```

```
for(i=0;i<r;i++)
{
    for(j=0;j<r;j++)
{
    b[i][j]=fac[j][i];
}
}
d=detrm(num,r);
inv[i][j]=0;
for(i=0;i<r;i++)
{
    for(j=0;j<r;j++)
{
    inv[i][j]=b[i][j]/d;
}
}
return(0);
}
```

Appendix B

1D Beam EFG Program (V2)

```
/******1D BEAM EFG PROGRAM******/
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
void matrixinverse(float [9] ,int);
void stiffbeam(float , float , float [100][100], float [100], float [100], float [100], int,
int);
void inversemat(float [100][100], int matsize );
float inv[100][100];
float K[100][100], Min[9], Mout[9];
FILE *f1, *f2;
void main()
{
    int nnd, nc, i, j, k, l, gpt, weight, lnd, matsize;
    float x[100], le, dmax, E, dm[100], gg[100], jac, f[100], GG[100], xg;
    float dif[100], r[100],drdx[100],we[100],dw[100],B[3][100],p[100][3], A[3][3], dA[3][3];
    float pp[3][3], Ainv[3][3], pd[3], phi[100][100], Aib[3][100], db[3][100], dphi[100][100];
float da[3][3], daa[3][3], zone[3], q;
```

```
float I, u0, thta0, M0, Q0, vx[100], vg[100], sign, dwx[100], ddA[3][3],ddb[3][100];
float temp[3][3], temp1[3][3], temp2[3][3], dda[3][3];
float ddphi[100][100], ztwo[3], phi0[100], dphi0[100], ddphi0[100], phin[100], dphin[100];
float ddphin[100], Kinv[100][100], tem;
    float d[100];
    f1=fopen("efgbeamin.in","r"); /*opening input file*/
    f2=fopen("efgbeamout.out","w"); /*opening output file*/
/*********SET UP NODAL COORDINATES ALONG BAR,
DETERMINE NUMBER OF CELLS*******/
    fscanf(f1,"%f %d",&le,&nnd);
    fprintf(f2,"Length of bar:\t%5.3f\n",le);
    fprintf(f2,"No. of nodes:\t%d\n",nnd);
    nc= nnd-1;
    fprintf(f2,"No. of cells:\t%d\n",nc);
/****SET PARAMETERS FOR WEIGHT FUNCTION, MATERIAL PROPER-
TIES********/
    fscanf(f1,"%f %f %f %f", &dmax, &E, &I, &q);
    fprintf(f2,"dmax=\t%5.2f\t\telasticity=\t%5.2f\t\tMoment of Inertia=\t%5.2f\t\tLoad=\t%
dmax, E, I, q);
fscanf(f1,"%f %f %f %f", &u0, &thta0, &M0, &Q0);
fprintf(f2,"U0=\t%15.3f\n",u0);
    fprintf(f2,"Theta0=\t%15.3f\n",thta0);
fprintf(f2,"M0=\t%15.3f\n",M0);
fprintf(f2,"Q0=\t%15.3f\n",Q0);
fscanf(f1,"%d",&nnd);
for(i=0;i<nnd;i++) /*node data input*/
    {
    x[i]=(i*(le/nc));
vx[i]= le/nc;
```

```
fprintf(f2,"x[%d]=\t%5.2f\t",i,x[i]);
}
fprintf(f2,"\n");
/***********DETERMINE DMI FOR EACH NODE***********/
for(i=0;i<nnd;i++)
{
    dm[i]=dmax*(x[2]-x[1]);
fprintf(f2,"dm[%d]=\t%5.2f\t",i,dm[i]);
    }
fprintf(f2,"\n");
/**********SET UP GAUSS POINTS, WEIGHTS, AND JACOBIAN FOR EACH
CELL********/
gpt=2*nnd;
for(i=0;i<gpt;i++)
{
    gg[i]=0.0;
}
jac=(x[2]-x[1])/2;
weight=2;
fprintf(f2,"weight= \t%d , Jac=\t%5.2f\t",weight,jac);
fprintf(f2,"\n");
for(i=0;i<gpt;i++)
{
    gg[i]=(i+0.5)*le/gpt;
vg[i]= le/gpt;
fprintf(f2,"gg[%d]:%5.2f\t",i,gg[i]);
}
fprintf(f2,"\n");
fprintf(f2,"\n");
```

```
for(i=0;i<nnd;i++)
{
GG[i]=0.0;
dw[i]=0.0;
we[i]=0.0;
for(k=0;k<3;k++)
    {
    B[k][i]=0.0;
}
for(j=0;j<nnd;j++)
{
    K[i][j]=0.0;
}
}
for(i=0;i<gpt+2;i++)
    for(j=0;j<nnd;j++)
    phi[i][k]=0.0;
/**********LOOP OVER GAUSS POINTS*********/
for(i=0;i<gpt+2;i++)
{
fprintf(f2,"\nThe Gauss pdoint %d\n",i);
if(i==gpt)
    xg=0.0;
else
    if(i==gpt+1)
    xg=le;
else
    xg=gg[i];
/*******DETERMINE DISTANCE BETWEEN NODES AND GAUSS POINT*******/
```

```
for(j=0;j<nnd;j++)
{
    dif[j]=xg-x[j];
fprintf(f2,"dif[%d]:%5.2f\t",j,dif[j]);
}
fprintf(f2,"\n");
/***************SET UP WEIGHTS W AND DW FOR EACH NODE************/
fprintf(f2,"The weight function:\n");
for(j=0;j<nnd;j++)
{
if(dm[j]==0.0)
    r[j]=0.0;
else
    r[j]=(fabs(dif[j]))/dm[j];
//fprintf(f2,"%5.2f\t",r[j]);
if(dif[j]<0)
    sign=-1;
else
    sign=1;
if(dif[j]==0.0 ||dm[j]==0.0)
    drdx[j]=0.0;
else
    drdx[j]=fabs(dif[j])/(dm[j]*dif[j]);
if(r[j]<=1.0)
{
    we[j]=1.0 - 6.0 * r[j]*r[j] + 8.0 * pow(r[j],3) - 3.0 * pow(r[j],4) ;
    dw[j]=(1.0/dm[j])*(-12.0 * r[j] + 24.0 * pow(r[j],2) - 12.0 * pow(r[j],3)) * sign;
    dwx[j] = pow((1/dm[j]),2) *(-12.0 + 48.0 * r[j] - 36.0 * pow(r[j],2));
}
```

```
else
{
    we[j]=0.0;
dw[j]=0.0;
dwx[j]=0.0;
}
fprintf(f2,"%7.4f\t",we[j]);
}
fprintf(f2,"\n");
fprintf(f2,"The first derivative of weight function:\n");
for(j=0;j<nnd;j++)
{
fprintf(f2,"%7.4f\t",dw[j]);
p[j][0]=1.0;
p[j][1]=(-(xg-x[j]));
p[j][2]=pow((xg-x[j]),2);
}
/*********SET UP SHAPE FUNCTIONS AND DERIVATIVES**********/
fprintf(f2,"\n");
for(j=0;j<nnd;j++)
{
    B[0][j]=(1.0*we[j]);
    B[1][j]=(-(xg-x[j])*we[j]);
B[2][j]=(pow((xg-x[j]),2))*we[j];
}
fprintf(f2,"\n");
fprintf(f2,"The matrix B is:\n");
for(j=0;j<3;j++)
{
```

```
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",B[j][k]);
}
fprintf(f2,"\n");
}
fprintf(f2,"\nThe Matrix db is:\n");
for(k=0;k<nnd;k++)
{
db[0][k]=1.0*dw[k];
    db[1][k]=(-(xg-x[k]))*dw[k];
db[2][k]=pow(xg-x[k],2)*dw[k];
}
for(j=0;j<3;j++)
{
    for(k=0;k<nnd;k++)
{
fprintf(f2,"%6.4f\t ",db[j][k]);
}
fprintf(f2,"\n");
}
fprintf(f2,"\nThe Matrix ddB is:\n");
for(k=0;k<nnd;k++)
{
ddb[0][k]=1.0*dwx[k];
    ddb[1][k]=(-(xg-x[k]))*dwx[k];
ddb[2][k]=pow(xg-x[k],2)*dwx[k];
}
for(j=0;j<3;j++)
```

```
{
    for(k=0;k<nnd;k++)
{
fprintf(f2,"%6.4f\t ",ddb[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    A[j][k]=0.0;
dA[j][k]=0.0;
ddA[j][k]=0.0;
}
}
for(j=0;j<nnd;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
{
    pp[k][l]=p[j][k]*p[j][l];
A[k][l]=A[k][l]+we[j]*pp[k][l];
dA[k][l]=dA[k][l]+dw[j]*pp[k][l];
ddA[k][l]=ddA[k][l]+dwx[j]*pp[k][l];
}
}
}
```

```
l=0;
fprintf(f2,"The matrix A is:\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
Min[l]=A[j][k];
Mout[l]=0.0;
    fprintf(f2,"A[%d][%d]=%6.3f\t",j,k,A[j][k]);
l++;
}
fprintf(f2,"\n");
}
fprintf(f2,"The matrix dA is:\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"dA[%d][%d]=%6.3f\t",j,k,dA[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"ddA[%d][%d]=%6.3f\t",j,k,ddA[j][k]);
}
fprintf(f2,"\n");
```

```
}
k=3;
matrixinverse(Min,k);
fprintf(f2,"The inverse of matrix A is:\n");
l=0;
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    Ainv[j][k]=Mout[l];
    fprintf(f2,"Ainv[%d][%d]=%6.3f\t",j,k,Ainv[j][k]);
l++;
}
fprintf(f2,"\n");
}
for(j=0;j<3;j++)
{
    for(k=0;k<nnd;k++)
{
    Aib[j][k]=0.0;
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<nnd;k++)
{
    for(l=0;l<3;l++)
{
    Aib[j][k]=Aib[j][k]+(Ainv[j][l]*B[l][k]);
```

```
}
}
}
fprintf(f2,"The Matrix Aib is:\n");
for(j=0;j<3;j++)
{
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"%6.3f\t",Aib[j][k]);
}
fprintf(f2,"\n");
}
pd[0]=1.0;
pd[1]=0.0;
pd[2]=0.0;
fprintf(f2,"%f\t %f\t %f\n",pd[0],pd[1],pd[2]);
for(k=0;k<nnd;k++)
{
    for(j=0;j<3;j++)
    {
    for(l=0;l<3;l++)
{
    phi[i][k] += (pd[j]*(Ainv[j][l]*B[l][k]));
}
}
}
for(k=0;k<nnd;k++)
{
    fprintf(f2,"phi[%d][%d]=%6.3f\n",i,k,phi[i][k]);
```

```
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
daa[j][k]=0.0;
    for(l=0;l<3;l++)
{
    daa[j][k]=daa[j][k]+dA[j][l]*Ainv[l][k];
}
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
da[j][k]=0.0;
    for(l=0;l<3;l++)
{
    da[j][k]=da[j][k]-(Ainv[j][l])*daa[l][k];
}
}
}
fprintf(f2,"\nThe first derivative of Ainv is\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"da[%d][%d]=%6.3e\t",j,k,da[j][k]);
```

```
}
fprintf(f2,"\n");
}
fprintf(f2,"\n");
zone[0]=0.0;
zone[1]=1.0;
zone[2]=0.0;
for(k=0;k<nnd;k++)
{
dphi[i][k]=0.0;
    for(j=0;j<3;j++)
    {
    for(l=0;l<3;l++)
{
    dphi[i][k] = dphi[i][k] + (zone[j] * (Ainv[j][l] * B[l][k])) + (pd[j] * (da[j][l] * B[l][k] +
Ainv[j][l] * db[l][k]));
}
}
}
for(j=0;j<nnd;j++)
{
fprintf(f2,"dphi[%d][%d]=%6.3e\n",i,j,dphi[i][j]);
    }
fprintf(f2,"\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    temp[j][k]=0.0;
```

```
temp1[j][k]=0.0;
temp2[j][k]=0.0;
dda[j][k]=0.0;
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
{
    temp[j][k]=temp[j][k]+(Ainv[j][l])*daa[l][k];
}
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"temp[%d][%d]=%6.3f\t",j,k,temp[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
{
```

```
    temp1[j][k]=temp1[j][k]+(dA[j][l])*temp[l][k];
}
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"temp1[%d][%d]=%6.3f\t",j,k,temp1[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
{
    temp2[j][k]=temp2[j][k]+(ddA[j][l])*Ainv[l][k];
}
}
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"temp2[%d][%d]=%6.3f\t",j,k,temp2[j][k]);
}
fprintf(f2,"\n");
```

```
}
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
    {
    dda[j][k]=dda[j][k]+(2*(Ainv[j][l]*temp1[l][k])-(Ainv[j][l]*temp2[l][k]));
    }
}
}
fprintf(f2,"\nThe Second derivative of Ainv is\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"dda[%d][%d]=%6.3e\t",j,k,dda[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<nnd;j++)
{
    ddphi[j][i]=0.0;
}
ztwo[0]=0.0;
ztwo[1]=0.0;
ztwo[2]=2.0;
for(j=0;j<3;j++)
{
```

```
    for(k=0;k<nnd;k++)
{
    for(l=0;l<3;l++)
{
    ddphi[k][i] = ddphi[k][i]+(ztwo[j]*(Ainv[j][l]*B[l][k]))+pd[j]*(dda[j][l]*B[l][k]+
da[j][l] * db[l][k] + da[j][l] * db[l][k] + Ainv[j][l] * ddb[l][k])) + (zone[j] * (2 * Ainv[j][l] *
db[l][k] + 2 * da[j][l] * B[l][k]));
}
}
}
fprintf(f2,"\nThe second derivative of phi\n");
    for(k=0;k<nnd;k++)
{
    fprintf(f2,"ddphi[%d][%d]=%6.3e\n",k,i,ddphi[k][i]);
}
if(i==gpt)
{
    for(k=0;k<nnd;k++)
{
    phi0[k]=phi[i][k];
dphi0[k]=dphi[i][k];
ddphi0[k]=ddphi[k][i];
}
}
if(i==gpt+1)
{
    for(k=0;k<nnd;k++)
{
    phin[k]=phi[i][k];
```

```
dphin[k]=dphi[i][k];
ddphin[k]=ddphi[k][i];
}
}
}
/**************ASSEMBLE DISCRETE EQUATIONS***************/
lnd=nnd+2;
stiffbeam(E, I, ddphi, phi0, dphi0, vg, nnd, gpt);
fprintf(f2,"\nThe stifness matrix is \n");
    for(j=0;j<lnd;j++)
{
    for(k=0;k<lnd;k++)
{
    fprintf(f2,"%6.3e\t",K[j][k]);
}
fprintf(f2,"\n");
}
matsize=lnd;
//inversemat(K,lnd);
for(i=0;i<matsize;i++) //automatically initialize the unitmatrix, e.g.
    for(j=0;j<matsize;j++) // - -
    if(i==j) // |1 0 0 |
    Kinv[i][j]=1; // |0 1 0 |
    else // |0 0 1 |
    Kinv[i][j]=0; // - -
/*————————LoGiC starts here————————*/ //procedure to make the matrix A
to unit matrix
    for(k=0;k<matsize;k++) //by some row operations,and the same rowoperations
of
```

```
{ //Unit mat. I gives the inverse of matrix A
tem=K[k][k]; //'temp' stores the A[k][k] value so that A[k][k] will not change
for(j=0;j<matsize;j++) //during the operation A[i][j]/=A[k][k] when i=j=k
{
K[k][j]/=tem; //it performs the following row operations to make A to unit matrix
Kinv[k][j]/=tem; //R0=R0/A[0][0],similarly for I alsoR0=R0/A[0][0]
} //R1=R1-R0*A[1][0] similarly for I
for(i=0;i<matsize;i++) //R2=R2-R0*A[2][0] ,,
{
tem=K[i][k]; //R1=R1/A[1][1]
for(j=0;j<matsize;j++) //R0=R0-R1*A[0][1]
{ //R2=R2-R1*A[2][1]
if(i==k)
break; //R2=R2/A[2][2]
K[i][j]-=K[k][j]*tem; //R0=R0-R2*A[0][2]
Kinv[i][j]-=Kinv[k][j]*tem; //R1=R1-R2*A[1][2]
}
}
}
/*————————LoGiC ends here————————*/
fprintf(f2,"The inverse of the matrix is:\n"); //Print the matrix I that now contains the inverse of mat. A
for(i=0;i<matsize;i++)
{
for(j=0;j<matsize;j++)
{
fprintf(f2,"%10.4e\t",Kinv[i][j]);
}
fprintf(f2,"\n");
```

```
    }
for(j=0;j<nnd;j++)
{
f[j]=0.0;
    for(k=0;k<gpt;k++)
{
    f[j] = f[j] + phi[k][j]*q*vg[k];
}
    //f[j] = f[j] + phin[j]*Q0 - dphin[j]*M0;
    }
    f[nnd] = u0;
    f[nnd+1] = thta0;
for(k=0;k<lnd;k++)
{
    fprintf(f2,"f[%d]=%6.3f\n",k,f[k]);
}
fprintf(f2,"\n");
for(i=0;i<lnd;i++)
    {
d[i]=0.0;
    for(j=0;j<lnd;j++)
    {
d[i]=d[i]+Kinv[i][j]*f[j];
}
}
for(k=0;k<lnd;k++)
{
    fprintf(f2,"d[%d]=\t%6.3f \n",k,d[k]);
}
```

```
}
/*****************FUNCTION TO FIND THE DETERMINANT OF THE MA-
TRIX**********************/
void matrixinverse(float Min[9], int actualsize)
{
/* This function calculates the inverse of a square matrix
*
* matrix_inverse(double *Min, double *Mout, int actualsize)
*
* Min : Pointer to Input square Double Matrix
* Mout : Pointer to Output (empty) memory space with size of Min
* actualsize : The number of rows/columns
*
* Notes:
* - the matrix must be invertible
* - there's no pivoting of rows or columns, hence,
* accuracy might not be adequate for your needs.
*
* Code is rewritten from c++ template code Mike Dinolfo
*/
/* Loop variables */
int i, j, k;
/* Sum variables */
double sum,x;
/* Copy the input matrix to output matrix */
for(i=0; i<actualsize*actualsize; i++)
{
Mout[i]=Min[i];
}
```

```
/* Add small value to diagonal if diagonal is zero */
for(i=0; i<actualsize; i++)
{
j=i*actualsize+i;
if((Mout[j]<1e-10)&&(Mout[j]>-1e-10))
    { Mout[j]=1e-10; }
}
/* Matrix size must be larger than one */
if (actualsize <= 1)
    return;
for (i=1; i < actualsize; i++)
{
Mout[i] /= Mout[0]; /* normalize row 0 */
}
for (i=1; i < actualsize; i++)
{
for (j=i; j < actualsize; j++)
{ /* do a column of L */
sum = 0.0;
for (k = 0; k < i; k++) {
sum += Mout[j*actualsize+k] * Mout[k*actualsize+i];
}
Mout[j*actualsize+i] -= sum;
}
if (i == actualsize-1) continue;
for (j=i+1; j < actualsize; j++)
{ /* do a row of U */
sum = 0.0;
for (k = 0; k < i; k++)
```

```
{
sum += Mout[i*actualsize+k]*Mout[k*actualsize+j];
}
Mout[i*actualsize+j] = (Mout[i*actualsize+j]-sum) / Mout[i*actualsize+i];
}
}
for ( i = 0; i < actualsize; i++ ) /* invert L */
{
for ( j = i; j < actualsize; j++ )
{
x = 1.0;
if ( i != j )
{
x = 0.0;
for ( k = i; k < j; k++ )
{
x -= Mout[j*actualsize+k]*Mout[k*actualsize+i];
}
}
Mout[j*actualsize+i] = x / Mout[j*actualsize+j];
}
}
for ( i = 0; i < actualsize; i++ ) /* invert U */
{
for ( j = i; j < actualsize; j++ )
{
if ( i == j )
continue;
sum = 0.0;
```

```
for ( k = i; k < j; k++ )
{
sum += Mout[k*actualsize+j]*( (i==k) ? 1.0 : Mout[i*actualsize+k] );
}
Mout[i*actualsize+j] = -sum;
}
}
for ( i = 0; i < actualsize; i++ ) /* final inversion */
{
for ( j = 0; j < actualsize; j++ )
{
sum = 0.0;
for ( k = ((i>j)?i:j); k < actualsize; k++ )
{
sum += ((j==k)?1.0:Mout[j*actualsize+k])*Mout[k*actualsize+i];
}
Mout[j*actualsize+i] = sum;
}
}
}
void stiffbeam(float E, float I, float ddphi[100][100], float phi0[100], float dphi0[100],
float vg[100], int nnd, int gpt)
{
    int i, j, k;
float K1[100][100], K2[100][100];
for(i=0;i<nnd+2;i++)
{
for(j=0;j<nnd+2;j++)
{
```

```
    K1[i][j]=0.0;
K2[i][j]=0.0;
K[i][j]=0.0;
}
}
    for(i=0;i<nnd;i++)
    {
for(j=0;j<nnd;j++)
{
for(k=0;k<gpt;k++)
{
    K1[i][j] = K1[i][j]+ E*I*ddphi[i][k]*ddphi[j][k]*vg[k];
}
    }
    }
// Lagrange multipliers: at clamped
    for (i=0;i<nnd;i++)
{
    K2[i][nnd]= phi0[i];
K2[i][nnd+1]= dphi0[i];
    K2[nnd][i]= phi0[i];
K2[nnd+1][i]= dphi0[i];
}
for(i=0;i<nnd+2;i++)
    {
    for(j=0;j<nnd+2;j++)
{
    K[i][j] = K[i][j] + K1[i][j] + K2[i][j];
}
```

```
}
}
void inversemat(float K[100][100], int matsize )
{
    float I[50][50],temp;
    int i,j,k;
    /*A=(float **)malloc(matsize*sizeof(float *)); //allocate memory dynamically for
matrix A(matsize X matsize)
    for(i=0;i<matsize;i++)
    A[i]=(float *)malloc(matsize*sizeof(float)); */
    /*I=(float **)malloc(matsize*sizeof(float *)); //memory allocation for indentity
matrix I(matsize X matsize)
    for(i=0;i<matsize;i++)
    I[i]=(float *)malloc(matsize*sizeof(float)); */
    //printf("Enter the matrix:"); // ask the user for matrix A
/* for(i=0;i<matsize;i++)
    for(j=0;j<matsize;j++)
    A[i][j]=Mat[i][j]; */
    for(i=0;i<matsize;i++) //automatically initialize the unitmatrix, e.g.
    for(j=0;j<matsize;j++) // - -
    if(i==j) // |1 0 0 |
    I[i][j]=1; // |0 1 0 |
    else // |0 0 1 |
    I[i][j]=0; // - -
/*————————LoGiC starts here————————*/ //procedure to make the matrix A
to unit matrix
    for(k=0;k<matsize;k++) //by some row operations,and the same rowoperations
of
    { //Unit mat. I gives the inverse of matrix A
```

```
    temp=K[k][k]; //'temp' stores the A[k][k] value so that A[k][k] will not change
    for(j=0;j<matsize;j++) //during the operation A[i][j]/=A[k][k] when i=j=k
    {
    K[k][j]/=temp; //it performs the following row operations to make A to unit
matrix
    I[k][j]/=temp; //R0=R0/A[0][0],similarly for I alsoR0=R0/A[0][0]
    } //R1=R1-R0*A[1][0] similarly for I
    for(i=0;i<matsize;i++) //R2=R2-R0*A[2][0] ,,
    {
    temp=K[i][k]; //R1=R1/A[1][1]
    for(j=0;j<matsize;j++) //R0=R0-R1*A[0][1]
    { //R2=R2-R1*A[2][1]
    if(i==k)
    break; //R2=R2/A[2][2]
    K[i][j]-=K[k][j]*temp; //R0=R0-R2*A[0][2]
    I[i][j]-=I[k][j]*temp; //R1=R1-R2*A[1][2]
    }
    }
    }
/*————————LoGiC ends here————————————*/
    printf("The inverse of the matrix is:"); //Print the matrix I that now contains the
inverse of mat. A
    for(i=0;i<matsize;i++)
    {
    for(j=0;j<matsize;j++)
{
inv[i][j]=I[i][j];
    fprintf(f2,"%f ",I[i][j]);
}
```

```
    fprintf(f2," \n ");
    }
}
```

Appendix C

2D Plane Stress EFG Program (V3)

```
/**************2D PLANE STRESS EFG PROGRAM************/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void mesh2(float,float,int,int);
void pgauss(int);
void egauss(float[2][200],int[200][200],float[2][4],int);
void domain(float [2], float [2][200], float [2][200], int);
void shape(float [2], float,float [2][200], int[200],float [2][200]);
void lowerupper(float [3][3], int);
void chole(float [3][3], int);
int numcell, numq, conn[200][200], b, c[200];
float x[2][200], dm[2][200], v[4][4], Min[200], Mout[200], sff[3][3], Lu[3][3], Uu[3][3],
perm[3][3];
float dphix[200], dphiy[200], phi[200], Imo, GG[200][200];
double gs[4][1000];
```

```
FILE *f1, *f2;
void main()
{
    int i, j, k, l, ndivl, ndivw, en[200], conn2, numnod, ndivlq, ndivwq, quado, connc[200][200], numq2;
int conn1[200][200], kkk, ind, ind1, ind2, m1, m2, n, c1, c2, c3, c4, c5, c6, matsize, tem ;
float L, D, E, nu, P, Dmat[3][3], x1[2][200], dmax, xspac, Bmat[3][200];
float yspac, xc[2][200], gauss[2][4], K[200][200], gpos[2], DB[200][200];
float gg[2][200], weight, jac, nnu[2][10], nt[2][10], lthu, ltht;
float ubar[200], f[200], ycen, jcob, mark[200], gst[4][200], gsu[4][200], qk[200];
float force[200], tx, ty, y1, y2, len, fac11, fac2, xp1, yp1, uxex1, uyex1, N1;
float N2, G1[2][2], G2[2][2], LM[200][200], LMinv[200][200];
f1=fopen("2din.in","r"); /*opening input file*/
f2=fopen("2dout.out","w"); /*opening output file*/
/*********SET UP NODAL COORDINATES ALONG BAR, DETERMINE NUMBER OF CELLS*******/
fscanf(f1,"%f %f %f %f %f ",&L, &D, &E, &nu, &P);
fprintf(f2,"Length:\t%6.2f\n",L);
fprintf(f2,"Height:\t%6.2f\n",D);
fprintf(f2,"Elasticity:\t%6.2e\n",E);
fprintf(f2,"Poission ratio:\t%6.2f\n",nu);
fprintf(f2,"Load:\t%6.2f\n",P);
fscanf(f1,"%d %d", &ndivl, &ndivw);
fprintf(f2,"No. division in length:\t%d\n",ndivl);
fprintf(f2,"No. division in height:\t%d\n",ndivw);
Dmat[0][0]=Dmat[1][1]=E/(1-(nu*nu));
Dmat[0][1]=Dmat[1][0]=E/(1-(nu*nu))*nu;
Dmat[0][2]=Dmat[1][2]=Dmat[2][0]=Dmat[2][1]=0.0;
```

```
Dmat[2][2]=E/(1-(nu*nu))*(1-nu)/2;
fprintf(f2,"The D matrix is\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
    fprintf(f2,"%7.2e\t",Dmat[i][j]);
}
fprintf(f2,"\n");
}
mesh2(L,D,ndivl,ndivw);
conn2=numcell;
numnod=numq;
for(i=0;i<2;i++)
{
for(j=0;j<numnod;j++)
{
x1[i][j]=x[i][j];
fprintf(f2,"%7.2f\t",x[i][j]);
}
fprintf(f2,"\n");
}
fprintf(f2,"\n");
for(i=0;i<4;i++)
{
for(j=0;j<conn2;j++)
{
conn1[i][j]=conn[i][j];
fprintf(f2,"%7d\t",conn1[i][j]);
```

```
}
fprintf(f2,"\n");
}
fprintf(f2,"%d\t%d\n",conn2, numnod);
//DETERMINE DOMAINS OF INFLUENCE - UNIFORM NODAL SPACING
dmax=3.5;
xspac = L/ndivl;
    yspac = D/ndivw;
for(j=0;j<numnod;j++)
{
    dm[0][j]=dmax*xspac;
    dm[1][j]=dmax*yspac;
}
for(i=0;i<2;i++)
{
for(j=0;j<numnod;j++)
{
    fprintf(f2,"%5.2f\t",dm[i][j]);
}
fprintf(f2,"\n");
}
//SET UP QUADRATURE CELLS
ndivlq = 10;
ndivwq = 4;
    mesh2(L,D,ndivlq,ndivwq);
for(i=0;i<2;i++)
{
for(j=0;j<numq;j++)
{
```

```
xc[i][j]=x[i][j];
fprintf(f2,"%7.2f\t",xc[i][j]);
}
fprintf(f2,"\n");
}
fprintf(f2,"\n");
for(i=0;i<4;i++)
{
for(j=0;j<numcell;j++)
{
connc[i][j]=conn[i][j];
fprintf(f2,"%7d\t",connc[i][j]);
}
fprintf(f2,"\n");
}
fprintf(f2,"%d\t%d\n",numcell, numq);
fprintf(f2,"\n");
    // SET UP GAUSS POINTS, WEIGHTS, AND JACOBIAN FOR EACH CELL
    quado = 4;
pgauss(quado);
for(i=0;i<2;i++)
{
    for(j=0;j<4;j++)
{
    gauss[i][j]=v[i][j];
fprintf(f2,"%5.4f\t",gauss[i][j]);
}
fprintf(f2,"\n");
}
```

```
numq2 = numcell*quadoˆ2;
for(i=0;i<4;i++)
{
for(j=0;j<numq2;j++)
{
    gs[i][j] = 0.0;
}
}
egauss(xc, conn, gauss, numcell);
for(i=0;i<4;i++)
{
kkk=39;
for(j=0;j<numq2;j++)
{
    //fprintf(f2,"%5.4f\t",gs[i][j]);
if(j==kkk)
{
// fprintf(f2,"\n");
kkk= kkk+40;
}
}
fprintf(f2,"\n");
}
for(i=0;i<numnod*2;i++)
{
    for(j=0;j<numnod*2;j++)
{
    K[i][j]=0.0;
}
```

```
}
for(i=0;i<numq2;i++)
{
    gpos[0] = gs[0][i];
gpos[1] = gs[1][i];
    weight = gs[2][i];
    jac = gs[3][i];
    domain(gpos,x,dm,numnod);
fprintf(f2,"\nfor gauss point %d\n",i);
for(j=0;j<b;j++)
{
    fprintf(f2,"%d\t",c[j]);
}
for(j=0;j<2*b;j++)
{
    en[j]=0.0;
}
shape(gpos,dmax,x,c,dm);
for(j=0;j<2*b;j++)
{
    for(k=0;k<3;k++)
{
    Bmat[k][j]=0.0;
}
}
for(j=0;j<b;j++)
{
    for(k=0;k<3;k++)
{
```

```
    if(k==0)
{
    Bmat[k][(2*j)-1+1]=dphix[j];
Bmat[k][2*j+1]=0.0;
}
else
{
    if(k==1)
{
Bmat[k][(2*j)-1+1]=0.0;
    Bmat[k][2*j+1]=dphiy[j];
}
else
{
    Bmat[k][(2*j)-1+1]=dphiy[j];
Bmat[k][2*j+1]=dphix[j];
}
}
}
}
for(j=0;j<3;j++)
{
for(k=0;k<2*b;k++)
{
    fprintf(f2,"%7.3f\t",Bmat[j][k]);
}
fprintf(f2,"\n");
}
for(j=0;j<b;j++)
```

```
{
    en[(2*j)-1+1] = 2*c[j]-1+1;
    en[(2*j)+1] = 2*c[j]+1;
    }
/*for(k=0;k<2*b;k++)
{
    fprintf(f2,"%7.3f\t",en[k]);
} */
fprintf(f2,"\n");
for(k=0;k<2*b;k++)
{
    for(j=0;j<3;j++)
    {
DB[j][k]=0.0;
for(l=0;l<3;l++)
{
    DB[j][k] += (Dmat[j][l]*Bmat[l][k]);
}
    }
    }
/*for(j=0;j<3;j++)
    {
    for(k=0;k<2*b;k++)
    {
    fprintf(f2,"%7.3e\t",DB[j][k]);
    }
    fprintf(f2,"\n");
    } */
for(j=0;j<2*b;j++)
```

```
    {
    for(k=0;k<2*b;k++)
    {
for(l=0;l<3;l++)
{
    K[en[j]][en[k]] += (Bmat[l][j]*DB[l][k])*(weight*jac);
}
    }
    }
/*fprintf(f2,"\n");
for(j=0;j<2*b;j++)
    {
    for(k=0;k<2*b;k++)
    {
    fprintf(f2,"%7.3e\t",K[en[j]][en[k]]);
    }
    fprintf(f2,"\n");
    } */
}
fprintf(f2,"\nThe K matrix is \n");
/*for(j=0;j<2*numnod;j++)
{
for(k=0;k<2*numnod;k++)
    {
    fprintf(f2,"%7.3e\t",K[j][k]);
}
fprintf(f2,"\n");
}*/
//DETERMINE NODES ON BOUNDARY, SET UP BC'S
```

```
ind1 = 0.0; ind2 = 0.0;
for(j=0;j<numnod;j++)
{
    if(x[0][j]==0.0)
{
    nnu[0][ind1] = x[0][j];
    nnu[1][ind1] = x[1][j];
ind1=ind1+1;
    }
    if(x[0][j]==L)
{
    nt[0][ind2] = x[0][j];
    nt[1][ind2] = x[1][j];
ind2=ind2+1;
    }
    }
for(k=0;k<ind1;k++)
{
    fprintf(f2,"%f\t%f\t",nnu[0][k],nnu[1][k]);
}
for(k=0;k<ind1;k++)
{
    fprintf(f2,"%f\t%f\t",nt[0][k],nt[1][k]);
}
lthu = ind1;
    ltht = ind2;
for(j=0;j<lthu*2;j++)
{
    ubar[j]=0.0;
```

```
}
for(j=0;j<numnod*2;j++)
{
    f[j]=0.0;
}
//SET UP GAUSS POINTS ALONG TRACTION BOUNDARY
ind=0;
    for(i=0;i<(ltht-1);i++)
{
    ycen=(nt[1][i+1]+nt[1][i])/2;
    jcob = fabs(((nt[1][i+1])-nt[1][i])/2);
    for(j=0;j<quado;j++)
{
    mark[j] = ycen-gauss[0][j]*jcob;
gst[0][ind]=nt[0][i];
    gst[1][ind]=mark[j];
    gst[2][ind]=gauss[1][j];
    gst[3][ind]=jcob;
ind = ind+1;
    }
    }
fprintf(f2,"\n");
for(j=0;j<4;j++)
{
for(k=0;k<ind;k++)
{
    fprintf(f2,"%7.3f\t",gst[j][k]);
}
fprintf(f2,"\n");
```

```
}
//SET UP GAUSS POINTS ALONG DISPLACEMENT BOUNDARY
for(j=0;j<4;j++)
{
for(k=0;k<ind;k++)
{
    if(j==0)
    gsu[j][k]=0.0;
else
    gsu[j][k]=gst[j][k];
}
}
for(i=0;i<2*lthu;i++)
    qk[i] = 0.0;
Imo = (1.0/12.0)*pow(D,3);
for(i=0;i<ind;i++)
{
    gpos[0] = gst[0][i];
gpos[1] = gst[1][i];
    weight=gst[2][i];
    jac = gst[3][i];
domain(gpos,x,dm,numnod);
fprintf(f2,"\nfor gauss point %d\n",i);
for(j=0;j<b;j++)
{
    fprintf(f2,"%d\t",c[j]);
}
for(j=0;j<2*b;j++)
{
```

```
    en[j]=0.0;
force[j]=0.0;
}
shape(gpos,dmax,x,c,dm);
    tx = 0.0;
    ty = -(P/(2.0*Imo))*((D*D)/4.0-(pow(gpos[1],2)));
    for(j=0;j<b;j++)
{
    en[2*j-1+1] = 2*c[j]-1+1;
    en[2*j+1] = 2*c[j]+1;
    force[2*j-1+1]=tx*phi[j];
    force[2*j+1] = ty*phi[j];
    }
for(j=0;j<2*b;j++)
{
    f[en[j]] = f[en[j]] + jac*weight*force[j];
}
    }
for(i=0;i<numnod;i++)
{
    fprintf(f2,"%7.3f\t%7.3f\n",f[2*i-1+1],f[2*i+1]);
}
for(i=0;i<numnod*2;i++)
{
    for(j=0;j<lthu*2;j++)
{
    GG[i][j] = 0.0;
}
}
```

```
    ind1=0; ind2=0;
for(i=0;i<(lthu-1);i++)
{
    m1 = ind1; m2 = m1+1;
    y1 = nnu[1][m1]; y2 = nnu[1][m2];
    len = y1-y2;
ind1=ind1+1;
for(j=0;j<quado;j++)
{
gpos[0] = gsu[0][ind2];
gpos[1] = gsu[1][ind2];
    weight = gsu[2][j];
    jac = gsu[3][j];
ind2=ind2+1;
    fac11 = (-P*nnu[1][m1])/(6*E*Imo);
    fac2 = P/(6.0*E*Imo);
    xp1 = gpos[0];
    yp1 = gpos[1];
    uxex1 = (6.0*L-3.0*xp1)*xp1 + (2+nu)*(pow(yp1,2) - pow((D/2),2));
    uxex1 = uxex1*fac11;
uyex1 = 3.0*nu*yp1*yp1*(L-xp1)+0.25*(4.0+5.0*nu)*xp1*D*D+(3.0*L-xp1)*xp1*xp1;
    uyex1 = uyex1*fac2;
N1 = (gpos[1]-y2)/len; N2 = 1-N1;
    qk[2*m1-1+1] = qk[2*m1-1+1]-weight*jac*N1*uxex1;
    qk[2*m1+1] = qk[2*m1+1] - weight*jac*N1*uyex1;
    qk[2*m2-1+1] = qk[2*m2-1+1] -weight*jac*N2*uxex1;
    qk[2*m2+1] = qk[2*m2+1] - weight*jac*N2*uyex1;
domain(gpos,x,dm,numnod);
    shape(gpos,dmax,x,c,dm);
```

```
for(n=0;n<b;n++)
{
    G1[0][0] = G1[1][1]= -weight*jac*phi[n]*N1;
G1[0][1] = G1[1][0] = 0.0;
    G2[0][0] = G2[1][1]= -weight*jac*phi[n]*N2;
G2[0][1] = G2[1][0] = 0.0;
    c1=2*c[n]-1+1;
c2=2*c[n]+1;
c3=2*m1-1+1;
c4=2*m1+1;
    c5=2*m2-1+1;
c6=2*m2+1;
    GG[c1][c3]=GG[c1][c3]+ G1[0][0];
GG[c2][c4]=GG[c2][c4]+G1[1][1];
GG[c1][c4]=GG[c1][c4]+G1[0][1];
GG[c2][c3]=GG[c2][c3]+G1[1][0];
GG[c1][c5]=GG[c1][c5]+ G2[0][0];
GG[c2][c6]=GG[c2][c6]+G2[1][1];
GG[c1][c6]=GG[c1][c6]+G2[0][1];
GG[c2][c5]=GG[c2][c5]+G2[1][0];
    }
}
}
for(i=0;i<numnod*2;i++)
{
    for(j=0;j<lthu*2;j++)
{
    fprintf(f2,"%7.4f\t",GG[i][j]);
}
```

```
fprintf(f2,"\n");
}
for(i=numnod*2;i<(numnod*2+lthu*2);i++)
{
    f[i]=0.0;
}
for(i=numnod*2;i<(numnod*2+lthu*2);i++)
{
    f[i]= -qk[i];
}
for(i=0;i<(numnod+lthu);i++)
{
    fprintf(f2,"%7.3e\t%7.3e\n",f[2*i-1+1],f[2*i+1]);
}
for(j=0;j<(numnod+lthu);j++)
{
    for(k=0;k<(numnod+lthu);k++)
{
    if(j<numnod && k<numnod)
    LM[j][k]=K[j][k];
else
    if(j>=numnod && k<lthu)
    LM[j][k]=GG[j-numnod][k];
else
    if(k>=numnod && j<lthu)
    LM[j][k]=GG[k-numnod][j];
else
    LM[j][k]=0.0;
}
```

```
}
matsize=(numnod+lthu);
//inversemat(K,lnd);
for(i=0;i<matsize;i++)
    for(j=0;j<matsize;j++)
    if(i==j)
    LMinv[i][j]=1;
    else
    LMinv[i][j]=0;
    fprintf(f2,"The Lagrange multiplier matrix is:\n");
    for(i=0;i<(numnod+lthu);i++)
    {
    for(j=0;j<(numnod+lthu);j++)
{
    fprintf(f2,"%10.4e\t",LM[i][j]);
}
    fprintf(f2,"\n");
    }
}
void mesh2(float length, float height, int ndivl, int ndivw)
{
    int i, j, elemn;
    float nodet[200][4];
    numcell= ndivw*ndivl;
    numq = (ndivl+1)*(ndivw+1);
    for(i=0;i<(ndivl+1);i++)
    {
    for(j=0;j<(ndivw+1);j++)
    {
```

```
    x[0][((ndivw+1)*(i)+j)]= (length/ndivl)*(i);
    x[1][((ndivw+1)*(i)+j)]= (-(height/ndivw)*(j)+height/2);
    }
    }
    for(j=0;j<ndivl;j++)
    {
    for(i=0;i<ndivw;i++)
{
    elemn = (j)*ndivw + i+1;
    nodet[elemn-1][0] = elemn + (j);
    nodet[elemn-1][1] = nodet[elemn-1][0] + 1;
    nodet[elemn-1][2] = nodet[elemn-1][1]+ndivw+1;
    nodet[elemn-1][3] = nodet[elemn-1][2]-1;
    }
    }
    for(j=0;j<elemn;j++)
    {
    for(i=0;i<4;i++)
{
conn[i][j]=nodet[j][i];
}
    }
}
void pgauss(int k)
{
// This function returns a matrix with 4 gauss points and their weights
    if (k==4)
{
    v[0][0] =-.861136311594052575224;
```

```
    v[0][1] =-.339981043584856264803;
    v[0][2] =(-v[0][1]);
    v[0][3] = (-v[0][0]);
    v[1][0] =.347854845137453857373;
    v[1][1] =.652145154862546142627;
    v[1][2] = v[1][1];
    v[1][3] = v[1][0];
    }
}
void egauss(float xc[2][200], int conn[200][200], float gauss[2][4], int numcell)
{
int index, e, i, j, k, l, one[50], je;
float psiJ[4], etaJ[4], xe[200], ye[200], eta, psi, N[50], NJpsi[50], NJeta[50];
float xpsi, ypsi, xeta, yeta, jcob, xq, yq;
    // routine to set up gauss points, jacobian, and weights
    index=0;
for(i=0;i<4;i++)
{
    one[i]=1;
}
    psiJ[0] = -1;
psiJ[1] = 1;
psiJ[2] = 1;
psiJ[3] = -1;
etaJ[0] = -1;
etaJ[1] = -1;
etaJ[2] = 1;
etaJ[3] = 1;
l = 4;
```

```
    for(e=0;e<numcell;e++)
{
    // DETERMINE NODES IN EACH CELL
    for(j=0;j<4;j++)
{
    je=conn[j][e];
xe[j]=xc[0][je-1];
ye[j]=xc[1][je-1];
    }
    for (i=0;i<l;i++)
{
    for(j=0;j<l;j++)
{
    index = index+1;
    eta=gauss[0][i];
psi=gauss[0][j];
for(k=0;k<4;k++)
{
    N[k] = .25*(one[k]+psi*psiJ[k])*(one[k]+eta*etaJ[k]);
    NJpsi[k]=.25*psiJ[k]*(one[k]+eta*etaJ[k]);
    NJeta[k]=.25*etaJ[k]*(one[k]+psi*psiJ[k]);
}
xpsi = ypsi = xeta = yeta= xq = yq =0.0;
for(k=0;k<4;k++)
{
    xpsi= xpsi + NJpsi[k]*xe[k];
    ypsi= ypsi + NJpsi[k]*ye[k];
    xeta= xeta + NJeta[k]*xe[k];
    yeta= yeta + NJeta[k]*ye[k];
```

```
xq = xq + N[k]*xe[k];
    yq = yq + N[k]*ye[k];
}
    jcob=xpsi*yeta-xeta*ypsi;
gs[0][index-1] = xq;
gs[1][index-1] = yq;
    gs[2][index-1] = gauss[1][i]*gauss[1][j];
    gs[3][index-1] = jcob;
    }
    }
    }
}
void domain(float gpos[2], float x1[2][200], float ds[2][200], int nnodes)
{
int i,j;
float dif[2][200], a[2][200];
for(j=0;j<nnodes;j++)
{
    dif[0][j]= gpos[0]-x1[0][j];
dif[1][j]= gpos[1]-x1[1][j];
}
for(i=0;i<2;i++)
{
    for(j=0;j<nnodes;j++)
{
    a[i][j]=ds[i][j]-fabs(dif[i][j]);
}
}
b=0;
```

```
for(j=0;j<nnodes;j++)
{
    if(a[0][j]>0 && a[1][j]>0)
{
    c[b]=j;
b++;
}
}
}
void shape(float gpos[2], float dmax,float x1[2][200], int c1[200],float ds[2][200])
{
int i,j,k,l;
    float nv[2][200], p[3][200], pp[3][3],aa[3][3],daax[3][3],daay[3][3];
float B[3][200], dif[2][200], t[2][200], signx, signy, rx, ry, drdx, drdy;
float w[200], dwdx[200], dwdy[200], wx, wy, dwx, dwy, a1[3][3];
float pd[3], pd1[3], pd2[3], C[3], gam[3][3], D1, D2, D3, temp[3];
float dbx[3][200], dby[3][200];
for(i=0;i<b;i++)
{
    nv[0][i]=x1[0][c1[i]];
nv[1][i]=x1[1][c1[i]];
}
for(i=0;i<b;i++)
{
p[0][i]=1.0;
    p[1][i]=nv[0][i];
p[2][i]=nv[1][i];
}
for(j=0;j<b;j++)
```

```
{
    dif[0][j]= gpos[0]-nv[0][j];
dif[1][j]= gpos[1]-nv[1][j];
}
for(j=0;j<b;j++)
{
    t[0][j]= ds[0][j]/dmax;
t[1][j]= ds[1][j]/dmax;
}
wx=0.0;
wy=0.0;
l=0.0;
// CUBIC SPLINE WEIGHT FUNCTION
    for(j=0;j<=b;j++)
{
if(dif[0][j]<l)
signx = (-1.0);
else
    signx=1.0;
if(dif[1][j]<l)
    signy= (-1.0);
else
    signy=1.0;
    drdx = signx/ds[0][c1[j]];
    drdy = signy/ds[1][c1[j]];
    rx = fabs(dif[0][j])/ds[0][c1[j]];
fprintf(f2,"rx=%6.3f\t",rx);
    ry = fabs(dif[1][j])/ds[1][c1[j]];
    if (rx>0.5)
```

```
{
    wx = (4.0/3.0)- (4.0*rx ) + (4.0*rx*rx) -((4.0/3.0)*pow(rx,3));
    dwx = (-4.0 + (8.0*rx)-(4.0*rx*rx))*drdx;
    }
else
{
    wx = (2.0/3.0) - 4.0*rx*rx + (4.0*pow(rx,3));
    dwx = (-8.0*rx + 12.0*rx*rx)*drdx;
    }
if (ry>0.5)
{
    wy = (4.0/3.0)-4.0*ry+4.0*ry*ry -((4.0/3.0)*pow(ry,3));
    dwy = (-4.0 + 8.0*ry-4.0*ry*ry)*drdy;
}
    else
{
    wy = (2.0/3.0) - 4.0*ry*ry + (4.0*pow(ry,3));
    dwy = (-8.0*ry + 12.0*ry*ry)*drdy;
    }
w[j] = wx*wy;
    dwdx[j] = wy*dwx;
    dwdy[j] = wx*dwy;
    }
for(j=0;j<b;j++)
{
    B[0][j]=p[0][j]*w[j];
B[1][j]=p[1][j]*w[j];
B[2][j]=p[2][j]*w[j];
}
```

```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
{
    pp[i][j]=0.0;
aa[i][j]=0.0;
daax[i][j]=0.0;
daay[i][j]=0.0;
}
}
for(j=0;j<b;j++)
{
    for(k=0;k<3;k++)
{
    for(l=0;l<3;l++)
{
    pp[k][l]=p[k][j]*p[l][j];
aa[k][l]=aa[k][l]+w[j]*pp[k][l];
daax[k][l]=daax[k][l]+dwdx[j]*pp[k][l];
daay[k][l]=daay[k][l]+dwdy[j]*pp[k][l];
}
}
}
fprintf(f2,"\nThe matrix aa = aa + w*p'*p\n");
for(j=0;j<3;j++)
{
    for(k=0;k<3;k++)
{
    fprintf(f2,"%6.3f\t",aa[j][k]);
```

```
}
fprintf(f2,"\n");
}
pd[0]=1.0; pd[1]=gpos[0]; pd[2]=gpos[1];
pd1[0]=0.0; pd1[1]=1.0; pd1[2]=0.0;
pd2[0]=0.0; pd2[1]=0.0; pd2[2]=1.0;
//-----------------------------------------------
lowerupper(aa,3);
//-----------------------------------------
for(k=0;k<3;k++)
{
    temp[k]=0.0;
for(l=0;l<3;l++)
{
    temp[k] += (pd2[l]-daay[k][l]*gam[l][0]);
}
}
//-------------------
for(j=0;j<3;j++)
{
    if(j==0)
{
    for(k=0;k<3;k++)
{
C[k]=0.0;
    for(l=0;l<3;l++)
{
    C[k] += perm[k][l]*pd[l];
}
```

```
}
}
else
    if(j==1)
{
    for(k=0;k<3;k++)
{
C[2-k]=0.0;
    for(l=0;l<3;l++)
{
    C[2-k] += (perm[k][l]*pd1[l]-daax[k][l]*gam[l][0]);
}
}
}
else
{
    for(k=0;k<3;k++)
{
    temp[k]=0.0;
    for(l=0;l<3;l++)
    {
    temp[k] += daay[k][l]*gam[l][0];
    }
    }
    for(k=0;k<3;k++)
{
C[k]=0.0;
    for(l=0;l<3;l++)
{
```

```
    C[k] += perm[k][l]*(pd2[l]-temp[l]);
}
}
}
D1=C[0];
D2=C[1]-(Lu[1][0]*D1);
D3=C[2]-Lu[2][0]*D1-Lu[2][1]*D2;
gam[2][j] = D3/Uu[2][2];
gam[1][j] = (D2 - Uu[1][2]*gam[2][j])/(Uu[1][1]);
    gam[0][j] = (D1 - Uu[0][1]*gam[1][j]-Uu[0][2]*gam[2][j])/Uu[0][0];
}
for(j=0;j<b;j++)
{
phi[j]=0.0;
    for(k=0;k<3;k++)
{
    phi[j] += (gam[k][0]*B[k][j]);
}
}
fprintf(f2,"\nThe matrix phi\n");
for(j=0;j<b;j++)
{
    fprintf(f2,"%7.4e\t",phi[j]);
}
fprintf(f2,"\n");
//dbx = p.*[dwdx;dwdx;dwdx];
for(j=0;j<b;j++)
{
    dbx[0][j]=p[0][j]*dwdx[j];
```

```
dbx[1][j]=p[1][j]*dwdx[j];
dbx[2][j]=p[2][j]*dwdx[j];
}
//dby = p.*[dwdy;dwdy;dwdy];
for(j=0;j<b;j++)
{
    dby[0][j]=p[0][j]*dwdy[j];
dby[1][j]=p[1][j]*dwdy[j];
dby[2][j]=p[2][j]*dwdy[j];
}
    //dphix = gam(1:3,2)*B + gam(1:3,1)*dbx;
for(j=0;j<b;j++)
{
dphix[j]=0.0;
    for(k=0;k<3;k++)
{
    dphix[j] += (gam[k][1]*B[k][j])+(gam[k][0]*dbx[k][j]);
}
}
    //dphiy = gam(1:3,3)*B + gam(1:3,1)*dby;
for(j=0;j<b;j++)
{
dphiy[j]=0.0;
    for(k=0;k<3;k++)
{
    dphiy[j] += (gam[k][2]*B[k][j])+(gam[k][0]*dby[k][j]);
}
}
fprintf(f2,"\nThe matrix dphix\n");
```

```
for(j=0;j<b;j++)
{
    fprintf(f2,"%7.4e\t",dphix[j]);
}
fprintf(f2,"\nThe matrix dphiy\n");
for(j=0;j<b;j++)
{
    fprintf(f2,"%7.4e\t",dphiy[j]);
}
fprintf(f2,"\n");
}
//---------------------------------------------
void lowerupper(float sff[3][3], int ip)
{
    int i,j,k;
    float a1[3][3];
    for(i=0;i<ip;i++)
    for(j=0;j<ip;j++)
    perm[i][j] = 0.0;
    for(i=0;i<ip;i++)
    perm[ip-i-1][i] = 1.0;
    /*fprintf(f2,"P-Matrix = \n");
    for(i=0;i<ip;i++)
    {
    for(j=0;j<ip;j++)
    fprintf(f2,"%f ",perm[i][j]);
    fprintf(f2,"\n");
    } */
    for(i=0;i<ip;i++)
```

```
    {
    for(j=0;j<ip;j++)
    {
    a1[i][j]=0.0;
    for(k=0;k<ip;k++)
    a1[i][j] += perm[i][k]*sff[k][j];
    }
    }
    /*fprintf(f2,"A-Matrix = \n");
    for(i=0;i<ip;i++)
    {
    for(j=0;j<ip;j++)
    fprintf(f2,"%f ",a1[i][j]);
    fprintf(f2,"\n");
    } */
    chole(a1,ip);
}
void chole(float a1[3][3], int n)
{
int i,j,k,m;
float sum;
    for(i=0;i<n;i++)
    {
    for(j=0;j<n;j++)
    {
    Uu[i][j]=0.0;
    Lu[i][j]=0.0;
    }
    }
```

```
    Uu[0][0]=a1[0][0];
    Lu[0][0]=1.0;
    for(j=1;j<n;j++)
    {
    Lu[j][0]=a1[j][0]/Uu[0][0];
    Uu[0][j]=a1[0][j];
    for(i=0;i<j;i++)
    {
    sum=0.0;
    for(m=0;m<i;m++)
    sum+=Lu[j][m]*Uu[m][i];
    Lu[j][i]=(a1[j][i]-sum)/Uu[i][i];
    sum=0.;
    for(m=0;m<i;m++)
    sum+=Lu[i][m]*Uu[m][j];
    Uu[i][j]=a1[i][j]-sum;
    }
    Lu[j][j]=1.0;
    sum=0.0;
    for(m=0;m<j;m++)
    sum+=Lu[j][m]*Uu[m][j];
    Uu[j][j]=a1[j][j]-sum;
}
}
double Determinant(double **a,int n)
{
int i,j,j1,j2;
double det = 0;
double **m = NULL;
```

```
if (n < 1) { /* Error */
} else if (n == 1) { /* Shouldn't get used */
det = a[0][0];
} else if (n == 2) {
det = a[0][0] * a[1][1] - a[1][0] * a[0][1];
} else {
det = 0;
for (j1=0;j1<n;j1++) {
m = malloc((n-1)*sizeof(double *));
for (i=0;i<n-1;i++)
m[i] = malloc((n-1)*sizeof(double));
for (i=1;i<n;i++) {
j2 = 0;
for (j=0;j<n;j++) {
if (j == j1)
continue;
m[i-1][j2] = a[i][j];
j2++;
}
}
det += pow(-1.0,j1+2.0) * a[0][j1] * Determinant(m,n-1);
for (i=0;i<n-1;i++)
free(m[i]);
free(m);
}
}
return(det);
}
/*
```

```
Find the cofactor matrix of a square matrix
*/
void CoFactor(double **a,int n,double **b)
{
int i,j,ii,jj,i1,j1;
double det;
double **c;
c = malloc((n-1)*sizeof(double *));
for (i=0;i<n-1;i++)
c[i] = malloc((n-1)*sizeof(double));
for (j=0;j<n;j++) {
for (i=0;i<n;i++) {
/* Form the adjoint a_ij */
i1 = 0;
for (ii=0;ii<n;ii++) {
if (ii == i)
continue;
j1 = 0;
for (jj=0;jj<n;jj++) {
if (jj == j)
continue;
c[i1][j1] = a[ii][jj];
j1++;
}
i1++;
}
/* Calculate the determinate */
det = Determinant(c,n-1);
/* Fill in the elements of the cofactor */
```

```
b[i][j] = pow(-1.0,i+j+2.0) * det;
}
}
for (i=0;i<n-1;i++)
free(c[i]);
free(c);
}
/*
Transpose of a square matrix, do it in place
*/
void Transpose(double **a,int n)
{
int i,j;
double tmp;
for (i=1;i<n;i++) {
for (j=0;j<i;j++) {
tmp = a[i][j];
a[i][j] = a[j][i];
a[j][i] = tmp;
}
}
}
```

Appendix D

Web-pages

Some of the web pages which have been used for study related to this thesis work are:

a. www.sicedirect.com

b. www.concrete.org

c. www.imechanica.org

d. www.pdf-search-engine.com

e. www.knovel.com

f. www.hq-uploads.com

References

[1] K. P. Chong A. P. Boresi and S. Saigal. *Approximate solution methods in engineering mechanics*. John Wiley and sons, Inc, second edition, 2003.

[2] S. Atluri and S. Shengping. The meshless local petrov-galerkin (mlpg) method: A simple & less-costly alternative to the finite element an boundary element methods. *Computational Mechanics*, 24:348– 372, 2002.

[3] S. N. Atluri and T. Zhu. New meshless local petrov-galerkin (mlpg) approach in computational mechanics. *Comput. Mech.*, 22(2):117 – 127, 1998.

[4] J. Belinha and L.M.J.S. Dinis. Nonlinear analysis of plates and laminates using the element free galerkin method. *Composite Structures*, 78:337350, 2007.

[5] M. D. Buhmann. *Radial Basis Functions*. Cambridge: Cambridge University Press, 2003.

[6] X. Liu C. K. Lee and S. C. Fan. Local multiquadric approximation for solving boundary value problems. *Mechanics*, 30:396–409, 2003.

[7] O. C. Zienkiewicz E. Onate, S. Idelsohn and R. L. Taylor. A finite point method in computational mechanics. application to convective transport and fluid flow. *Int. J. Numer. Methods Eng*, 39:3839–3866, 1996.

[8] E. Onate e.t. al. A stabilized finite point method for analysis of fluid mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 139:315–346, 1996.

[9] W. K. Liu e.t. al. Reproducing kernel particle methods for structural dynamics. *Int. J. Numer. Methods Eng.*, 38:1655 – 1679, 1995.

[10] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. R. astr. Soc.*, 181:375–389, 1977.

[11] T. Belytchko J. Dolbow. An introduction to programming the meshless element free galerkin method. *Archives of Computational methods in engineering*, 5, 3:207–241, 1998.

[12] E. J. Kansa. Multiquadrics - a scattered data approximation scheme with applications to computational fluid-dynamics - i. *Computers & Mathematics with Applications*, 19(8/9):127–145, 1990.

[13] E. J. Kansa. Multiquadrics - a scattered data approximation scheme with applications to computational fluid-dynamics - ii,. *Computers & Mathematics with Applications*, 19(8/9):147–161, 1990.

[14] C. S. Krishnamoorthy. *Finite Element Analysis Theory and Programming*. TATS Mcgraw hill, second edition, 1994.

[15] S. Li and W. K. Liu. Meshless and particle methods and their applications. *Applied Mechanics Review*, 55:1–34, 2002.

[16] T. Liszka and J. Orkisz. The finite difference method at arbitrary irregular grids and its application in applied mechanics. *Cornput. Struct*, 11:83–95, 1980.

[17] G. R. Liu. A point assembly method for stress analysis for solid, in impact response of materials & structures. *Shim, V. P. W. et al., Eds.,*, pages 475–480, 1999.

[18] G. R. Liu. *Mesh Free Methods: Moving beyond the Finite Element Method*. CRC press, 2003.

[19] G. R. Liu and Y. T. Gu. A point interpolation method for two dimensional solids. *Int. J. Numer. Methods Eng.*, 50:937–951, 2001.

[20] M. B. Liu and G. R. Liu. *Smoothed Particle Hydrodynamics:A Meshfree Particle Method*. World Scientific Publishing, 2003.

[21] B. L. Lucy. A numerical approach to testing the fission hypothesis. *Astron. J*, 82(12):1013–1924, December 1977.

[22] J.-M. Savignat P. Breitkopf, A. Rassineux and P. Villon. Integration constraint in diffuse element method. *Computer Methods in Applied Mechanics and Engineering*, 193:1203–1220, 2004.

[23] T. Belytchko P. Krysl. Analysis of thin plates by the element free galerkin method. *Computational Mechanics*, 17:26–35, 1996.

[24] T. Belytchko P. Krysl. Analysis of thin shells by the element free galerkin method. *Int. J. Solids Structures*, 33(20-22):3057–3080, 1996.

[25] T. Belytchko S. Beissel. Nodal integration of the element-free galerkin method. *Computer methods in applied mechanics and engineering*, 139:49–74, 1996.

[26] B. N. Rao S. Rahman. An element free galerkin method for probabilistic mechanics and reliability. *International journal for solids and structures*, 38:9313–9330, 2001.

[27] D. Organ T. Belystchko, Y. Krongauz and M. Fleming. Meshless method: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139:3–47, 1996.

[28] Y. Y. Lu T. Belytschko and L. Gu. Element free galerkin methods. *International journal for numerical methods in engineering*, 37:229–256, 1994.

[29] S. Bordas V. Nguyen, T. Rabczuk and M. Duflot. Meshless methods: A review and computer implementation aspects. *Mathematics and Computers in Simulation*, 79:763813, 2008.

[30] S. Li W. K. Liu and T. Belytschko. Moving least-square reproducing kernel methods (ii) - fourier analysis. *Computer Methods in Applied Mechanics and Engineering*, 139:159194, 1996.

[31] S. Li W. K. Liu and T. Belytschko. Moving least-square reproducing kernel methods (i) - methodology and convergence. *Computer Methods in Applied Mechanics and Engineering*, 143:113–154, 1997.

[32] H. Wendland. Meshless galerkin methods using radial basis functions. *Mathematics of Computation*, 68(228):1521–1531, 1999.

[33] T. Belytchko Y. Krongauz. Enforcement of essential boundary conditions in meshless approximations using finite elements. *Comput. Methods Appl. Mech. Engrg.*, 131:133–145, 1996.

[34] T. Belytchko Y. Krongauz. Efg approximation with discontinous derivatives. *International journal for numerical methods in engineering*, 41:1215–1233, 1998.