



ADDIS ABABA UNIVERSITY  
SCHOOL OF GRADUATE STUDIES  
ADDIS ABABA INSTITUTE OF TECHNOLOGY  
School of Civil and Environmental Engineering

## Development of a Finite Element Software for Computing Stresses and Deformations in Layered Soils

By

Hiruy Dagnew  
B.Sc. in Civil Engineering  
Addis Ababa University, 2006

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the  
Requirement for Degree of Master of Science in Geotechnical Engineering

Advisor  
Prof. Alemayehu Teferra

July 2014  
Addis Ababa  
Ethiopia

ADDIS ABABA UNIVERSITY  
SCHOOL OF GRADUATE STUDIES  
ADDIS ABABA INSTITUTE OF TECHNOLOGY  
School of Civil and Environmental Engineering

Development of a Finite Element Software for Computing Stresses  
and Deformations in Layered Soils

By

Hiruy Dagnew  
B.Sc. in Civil Engineering  
Addis Ababa University, 2006

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the  
Requirement for Degree of Master of Science in Geotechnical Engineering

Approved by Board of Examiners:

Prof. Alemayehu Teferra  
*Advisor*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*

\_\_\_\_\_  
*External Examiner*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*

\_\_\_\_\_  
*Internal Examiner*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*

\_\_\_\_\_  
*Chair Person*

\_\_\_\_\_  
*Signature*

\_\_\_\_\_  
*Date*

## DECLARATION

I, the undersigned, declare that this thesis is my original work performed under the supervision of my research advisor Prof. Alemayehu Teferra and has not been presented as a thesis for a degree in any other university. All sources of materials used for this thesis have also been duly acknowledged.

Name           Hiruy Dagnew

Signature       \_\_\_\_\_

Place           Addis Ababa Institute of Technology,  
                    Addis Ababa University,  
                    Addis Ababa

Date:           July 2014

## **ACKNOWLEDGEMENTS**

Compilation of this thesis has taken me through a journey longer than I expected at first. Thanks to the Lord it has now made it to the final stages. Through the course of my undertaking, I have been receiving the assistance and encouragement of many people. A special gratitude goes to my advisor Professor Alemayehu Teferra. His guidelines and suggestions have added huge value to my work and directed it into a more practical track. Similarly, other faculty members at the Civil Engineering Department have contributed to my work at various occasions. Friends, colleagues and classmates have all been providing much needed encouragement throughout my postgraduate sessions. Well, nothing else would have been enough if I didn't have the support of a loving family.

In addition to the above, I shouldn't leave without mentioning the significant role played by availability of online references to my work. I would like to express my appreciation to all who are contributing their part in promoting free transfer of knowledge.

# Table of Contents

<b>ACKNOWLEDGEMENTS .....</b>	<b>i</b>
<b>ABSTRACT.....</b>	<b>iv</b>
<b>LIST OF COMMONLY USED SYMBOLS AND ABBREVIATIONS.....</b>	<b>v</b>
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>1.1 Background .....</b>	<b>1</b>
<b>1.2 Statement of the Problem .....</b>	<b>2</b>
<b>1.3 Aim and Objectives.....</b>	<b>2</b>
<b>1.4 Scope of Study .....</b>	<b>3</b>
<b>1.5 Presentation of the Thesis.....</b>	<b>3</b>
<b>2. LITERATURE REVIEW .....</b>	<b>5</b>
<b>2.1.General.....</b>	<b>5</b>
<b>2.2.Derivation of the Finite Element Method Equations.....</b>	<b>6</b>
<b>2.2.1.Derivation of Equations for Plane Strain Problems .....</b>	<b>6</b>
<b>2.2.1.1. Selecting the Type of Element.....</b>	<b>8</b>
<b>2.2.1.2. Selecting Displacement Functions.....</b>	<b>9</b>
<b>2.2.1.3. Defining the Strain-Displacement and Stress-Strain Relationships .....</b>	<b>12</b>
<b>2.2.1.4. Derivation of the Elemental Stiffness Matrix Using the Total Potential                     Energy Approach.....</b>	<b>15</b>
<b>2.2.2.Considerations for Axisymmetric Problems.....</b>	<b>17</b>
<b>2.2.3.Summary of Useful Equations .....</b>	<b>19</b>
<b>3. FEM Software Development.....</b>	<b>20</b>
<b>3.1. Program Flow Chart.....</b>	<b>20</b>
<b>3.2. Program Modules Incorporated in the Software .....</b>	<b>22</b>
<b>3.2.1. The SOIL DATA ENTRY Program Module.....</b>	<b>22</b>
<b>3.2.2. The LOADING DATA ENTRY Program Module .....</b>	<b>22</b>
<b>3.2.3. The GENERATE MESH Program Module.....</b>	<b>22</b>
<b>3.2.4. The RUN ANALYSIS Program Module.....</b>	<b>23</b>
<b>3.3. User Interfaces and Application Example.....</b>	<b>28</b>
<b>4. ANALYSIS AND COMPARISON .....</b>	<b>36</b>
<b>4.1. Introduction.....</b>	<b>36</b>

<b>4.2. Comparison of Analysis Results Against Available Equations and Other Software ..</b>	<b>37</b>
<b>4.2.1. Single Layer (Elastic Isotropic Half-Space) .....</b>	<b>37</b>
<b>4.2.1.1. Surface Loads .....</b>	<b>37</b>
<b>4.2.1.1.1. Point Load .....</b>	<b>37</b>
<b>4.2.1.1.2. Line Load.....</b>	<b>40</b>
<b>4.2.1.1.3. Uniform Strip Load .....</b>	<b>43</b>
<b>4.2.1.1.4. Uniform Load of Circular Plan Area.....</b>	<b>46</b>
<b>4.2.1.2. Loads in Half-Space .....</b>	<b>49</b>
<b>4.2.1.2.1. Point Load .....</b>	<b>49</b>
<b>4.2.1.2.2. Line Load.....</b>	<b>52</b>
<b>4.2.1.2.3. Uniform Strip Load .....</b>	<b>55</b>
<b>4.2.1.2.4. Uniform Load of Circular Plan Area.....</b>	<b>58</b>
<b>4.2.2. Two Layers- Surface Loads .....</b>	<b>61</b>
<b>4.2.2.1.1. Point Load .....</b>	<b>61</b>
<b>4.2.2.1.2. Line Load.....</b>	<b>64</b>
<b>4.2.2.1.3. Uniform Strip Load .....</b>	<b>67</b>
<b>4.2.2.1.4. Uniform Load of Circular Plan Area.....</b>	<b>70</b>
<b>4.2.3. Three Layers- Surface Loads.....</b>	<b>73</b>
<b>4.2.3.1. Point Load .....</b>	<b>73</b>
<b>4.2.3.2. Line Load.....</b>	<b>76</b>
<b>4.2.3.3. Uniform Strip Load .....</b>	<b>79</b>
<b>4.2.3.4. Uniform Load of Circular Plan Area.....</b>	<b>82</b>
<b>5. CONCLUSION AND RECOMMENDATIONS .....</b>	<b>85</b>
<b>5.1. Conclusion .....</b>	<b>85</b>
<b>5.2. Recommendations .....</b>	<b>86</b>
<b>6. REFERENCES.....</b>	<b>87</b>
<b>ANNEXES .....</b>	<b>88</b>
<b>ANNEX 1: Codes Used for the Development of the Software.....</b>	<b>89</b>

## **ABSTRACT**

Stress and deformation are the two most basic parameters that a geotechnical engineer deals with while analyzing or recommending foundations for any type of structure. The two correlated parameters express the reaction a soil medium exhibits upon application of loading. In the design stage, these parameters need to be estimated accurately and be checked that they lie within acceptable limits as stated in design standard codes so that the structure to be built will be able to serve its intended purpose in a safe and serviceable manner.

A number of equations have been provided to compute stresses and deformations of foundation soils. These equations provide stress and deformation values as a function of the load being applied, the elastic properties of the soil, and the location of the point where it is required to find out the stress and deformation values. While computation of the unknowns is relatively easy for homogeneous soils, the problem gets more complicated for multiple-layered foundation soils. In cases of such multiple-layered occurrences, very limited equations are available and hence a geotechnical engineer is usually forced to use less accurate approximation techniques such as averaging the characteristics of the various layers into a single value. The finite element technique provides an alternative approach whereby analysis is conducted through numerical methods which give results that are comparable to the closed form solutions. In addition to its accuracy, the finite element technique has an advantage in enabling the analysis of a wide range of problem types because of its more generalized approach.

This research is an applied type of research where a finite element software that enables computation of the stress and deformation values for multi-layered soils has been developed. The analysis considers linearly elastic material model for each layer. The software uses linear displacement functions and can analyze the two dimensional problems of plane strain and axisymmetric conditions. The output of the software has been compared against results of closed form solutions where available and also against output of a commercially available software. The comparison has shown that, with proper modeling and data entry, the finite element software can generate an output of very good accuracy.

The role of the computer software will be limited to collecting the input parameters in a user friendly interface, solving the large number of simultaneous equations that are generated in the finite element procedure and displaying the output in a number of handy alternatives. For the whole process, the user will be expected to have a thorough knowledge of geotechnical engineering concepts and the steps being followed by the software.

## LIST OF COMMONLY USED SYMBOLS AND ABBREVIATIONS

$A$	-	Area of an element in a finite element mesh
$[B]$	-	B-Matrix (Strain-Deformation Matrix)
$\gamma_{xz}$	-	Engineering Shear Strain in the x-z Plane
$\gamma_{rz}$	-	Engineering Shear Strain in the r-z Plane of Cylindrical Coordinate System
$[D]$	-	Stress-Strain Matrix
$\{d\}$	-	Displacement Vector/Matrix
DOF	-	Degree of Freedom
$E$	-	Young's Modulus of Elasticity
$\epsilon_x, \epsilon_y, \epsilon_z$	-	Normal Strains in Cartesian Coordinate System
$\epsilon_r, \epsilon_\theta, \epsilon_z$	-	Normal Strains in Cylindrical Coordinate System (for Axisymmetric problems)
$\epsilon_{xz}$	-	Pure Shear Strain in the x-z Plane
$\epsilon_{rz}$	-	Pure Shear Strain in the r-z Plane of the Cylindrical Coordinate System
$\{f\}$	-	Force Vector/Matrix
FEM	-	Finite Element Method
$G$	-	Shear Modulus
$i, j, m$	-	The three nodes of a triangular element
$[k]$	-	Elemental Stiffness Matrix
$[K]$	-	Global Stiffness Matrix
$N_i, N_j, N_m$	-	Shape functions
$\pi_p$	-	Total Potential Energy
$\sigma_x, \sigma_y, \sigma_z$	-	Normal Stresses in the x, y and z directions
$\sigma_r, \sigma_\theta, \sigma_z$	-	Normal Stresses in Cylindrical Coordinate System
$\tau_{xz}$	-	Shear Stress in the x-z Plane
$\tau_{rz}$	-	Shear Stress in the r-z Plane of Cylindrical Coordinate System
$U$	-	Strain Energy
$u, v$	-	Displacements in the horizontal and vertical directions, respectively
$\nu$	-	Poisson's Ratio
$\psi_i$	-	Displacement function
$\Omega_b, \Omega_p, \Omega_s$	-	Potential energies of the body, concentrated and distributed forces, respectively



## **1. INTRODUCTION**

### **1.1 Background**

Various equations have been proposed to estimate immediate settlement, a basic concern in the process of foundations' design. While most equations are applicable to homogenous soil cases, some methods have been recommended to be used for analyzing the case of two layers.

An equation has been provided by H.F. Winterkorn/H. Fans to estimate settlement of the center of a uniformly loaded area on an elastic layer underlain by a rigid base. Another equation by Burmister(1965) estimates elastic settlement of a stiff layer of finite thickness underlain by a soft layer of great depth. [5]

As can be inferred from above, the available equations can be used for a very limited variation of problems. Apparently, it is not manageable to come up with a closed form generalized solution for computation of elastic settlement in multilayered soils.

In this thesis, it has been attempted to provide a numerical solution to the stress and strain analysis of multi-layered soils using the finite element technique which is a numerical method that has been gaining popularity in recent years. This method simplifies the solution procedure as it does not try to come up with a closed form equation but rather involves discretizing the soil domain into fine elements and to express the problem in terms of what is referred to as global stiffness equation which can be solved to give the unknown force and displacement values. The current advancement in computation capacity has made it possible to handle the large memory and processing requirements of this technique thereby making it a more attractive alternative than the closed form solutions for such problems.

A few Finite Element software (such as PLAXIS and GeoStudio) are available for use in the global market. But the prices of this software are very high and the industry is known to make use of cracked versions of the software in the country. Such practice is known to pose a great hazard in the engineering world. The output from a cracked/manipulated software could be highly erroneous, to say the least. Through this research work, a Finite Element Software that will be affordable, user friendly (interactive), and open sourced is being provided and made available to answer one of the basic needs of the local geotechnical engineering professional community.

## **1.2 Statement of the Problem**

Foundation soils are usually found in a stratified arrangement. This is attributed to geological processes such as volcanic action, erosion/sedimentation and weathering. Limited techniques are available for estimating the stress and deformation parameters in such multi-layered systems. It is apparent that closed form equations will not be practical for analyzing such systems due to an extremely wide range of problem combinations that could possibly be encountered when one starts trying to handle non homogeneous cases. Hence, the remaining option is looking into the finite element technique and develop an application that fulfills the following qualities.

The application should be able to accept the necessary inputs (elastic properties and loading magnitude) and compute the stress and deformation values for the given model problem.

Accuracy of the finite element technique is related with the subdivisions (meshing arrangement) that are applied to the modeled problem. Hence, after developing the software, a number of trial runs should be conducted to compare outputs of the program against known results and recommendation will be given concerning what kind of meshing arrangement gives the most accurate result.

The software should be user friendly and the user should be able to get computational output at each step so that the analysis may be checked independently.

## **1.3 Aim and Objectives**

This study is aimed at developing an application software that can determine stresses and deformation parameters for single-layered and multi-layered soils. The following objectives are set forth in order to reach the aim of the study:

1. To gain deeper understanding of the basic principles in the theory of elasticity that govern stress and deformation parameters
2. To model the process using the Finite Element Method (FEM) and to apply the model into a computer application software.
3. To develop the application software in such a manner that it can analyze problems involving various types of loading, various types of boundary conditions, and various types of foundation material arrangements.
4. To analyze and compare the results of the software output against closed form equations (for cases where available) and against other software.

## **1.4 Scope of Study**

This study is limited to developing a Finite Element Method based application that can estimate stresses and deformations (strains and displacements) applicable within the elastic range of geotechnical problems. Hence, estimations of long term settlements, i.e. consolidation and creep settlements, will not be covered here.

The software is being prepared to analyze two dimensional problems. In geotechnical engineering, two dimensional analyses are practically applicable for problems of plane-strain and axisymmetric conditions where the problem is simplified due to the fact that a number of the strain components are zero in these two cases.

While bound by the above listed conditions, the software shall undertake the analysis of various types of loading and stratification arrangements as follows.

- i. Application of any numbers of or combinations of point loads, continuous line loads, circular loads and uniform strip loads;
- ii. Application of any number of layers that can be represented by the elastic properties Young's modulus of Elasticity and Poisson's ratio.

The types of loading arrangements that can be analyzed by the software are shown in Figure 1.1.

## **1.5 Presentation of the Thesis**

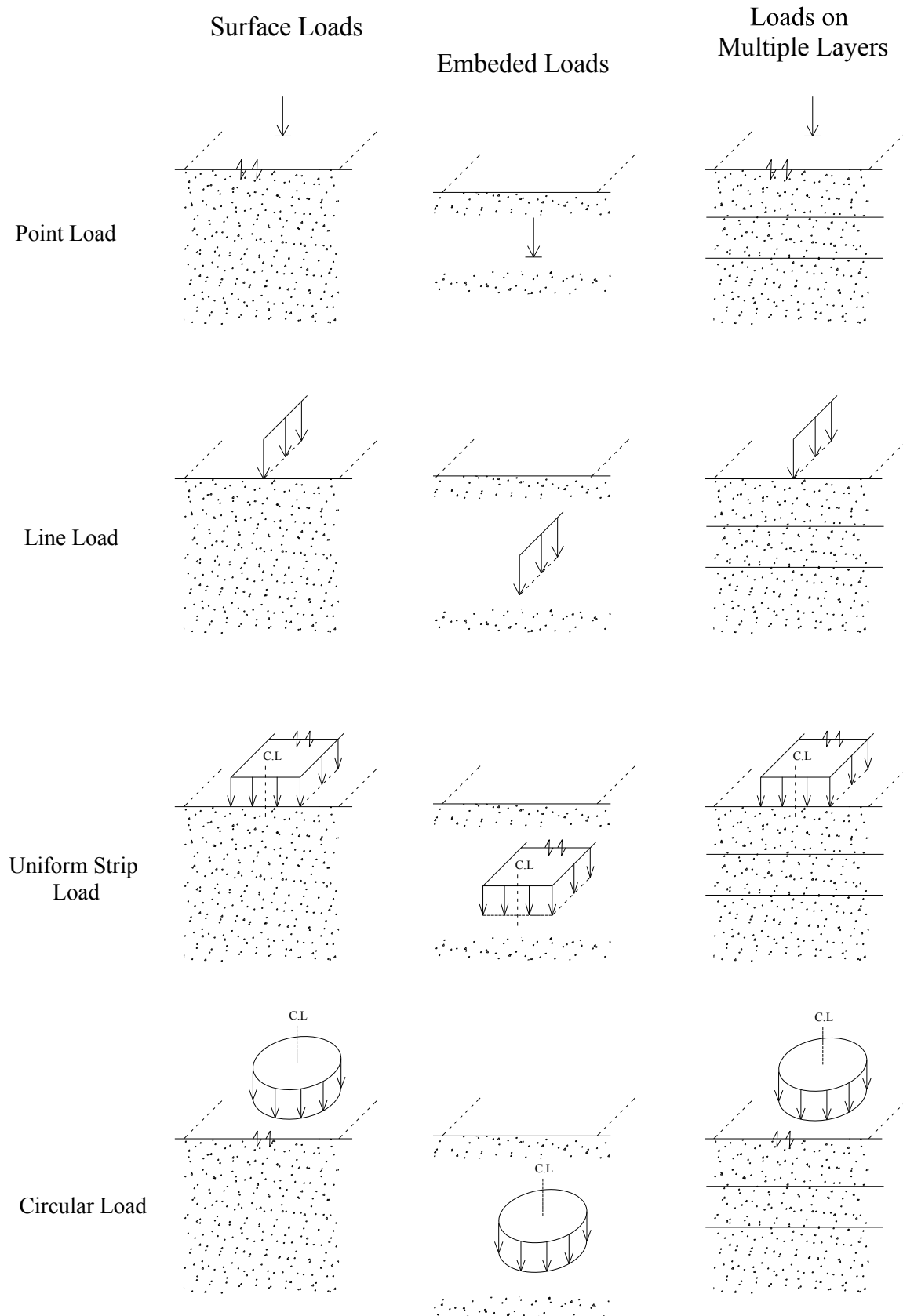
This thesis work comprises of five Chapters. The first Chapter introduces the main issue that will be covered by this paper. The basic objectives and the scope of the research are also presented here.

The second Chapter is dedicated to literature review. In this portion, highlights are given concerning previous works conducted under the topic of stress and deformation computations. Also in this Chapter, the derivation of the basic finite element equations will be discussed.

The third Chapter is dedicated to discussion of the software development procedures taken. Here, the basic programming modules incorporated in the software and flowchart of the solution process will be covered. Also in this Chapter an example problem will be solved in order to display the software's usage.

The analysis and comparison section consists of the forth Chapter. Here, output of the finite element software is compared with output from other software and also with results of closed form solutions where available.

On the final Chapter, conclusion and recommendation are forwarded based on lessons taken from the research work.



**Figure 1.1:** Types of Loads that can be Analyzed by the FEM Software

## 2. LITERATURE REVIEW

### 2.1. General

The basic question of how an elastic medium will react to a load has been motivating various professional to work on what are referred to as fundamental solutions of elasticity.

Earlier, solutions for the case of a homogeneous elastic material being acted on by either a concentrated load or distributed load have been discussed by Boussinesq, Flament, Kelvin, Cerrutti, and Mindlin. The works of these individuals are solutions bound by governing equations of constitutive law, conservation of energy, and continuity (compatibility) conditions. The solutions provided by each of the professionals have been developed for different sets of problems as follows. [3][7]

- Joseph Boussinesq provided equations to respond to the problem of a point load acting normal to the surface of an elastic half space.
- Alfred Flamant used Boussinesq's solution to answer the question of how the continuum would react if a line load was applied normal to the surface of an elastic half-space.
- Willam Thompson (Lord Kelvin) solved the problem of a point load acting within an infinite elastic continuum.
- Cerrutti Solved the problem of a horizontal point load acting along the surface of elastic half-space.
- Raymond Mindlin solved problems of a point load acting vertically and horizontally within an elastic half space.

While the solutions provided by each of the above scientists are suited for the cases of a homogeneous material extending indefinitely at least in depth and width, their use for geotechnical engineering has been rather limited due to their applicability to only ideal scenarios. Nevertheless, these solutions have laid the foundation for further manipulation to develop solutions to the cases of non homogeneous problems.

The equation provided by H.F. Winterkorn/H. Fans involves a shape factor  $J'$  to estimate settlement of the center of a uniformly loaded area on an elastic layer underlain by a rigid base. [5]

Alternatively, the equation of Burmister estimates elastic settlement of a stiff layer of finite thickness underlain by a soft layer of great depth. [5]

While the earlier attempts have been focusing on providing mathematical solutions to estimate stress and deformation distribution for a series of cases, all the closed form solutions remain bound to very limited combination of loads and arrangement of elastic media. The finite element technique provides an alternative numerical approach to solving such geotechnical problems. This technique which has been gaining popularity in recent years with the advent of high performance computers has a number of advantages over the closed form type solutions. Its main advantage lies in its implementation of a generalized set of procedures in solving problems with any type of loading condition and geometrical arrangement. [6]

## **2.2. Derivation of the Finite Element Method Equations**

### **2.2.1. Derivation of Equations for Plane Strain Problems**

The stiffness approach is used in developing a solution strategy for the finite element software. The stiffness approach is a robust means of computing for unknown forces and displacement values in matrix equation format. [1][6]

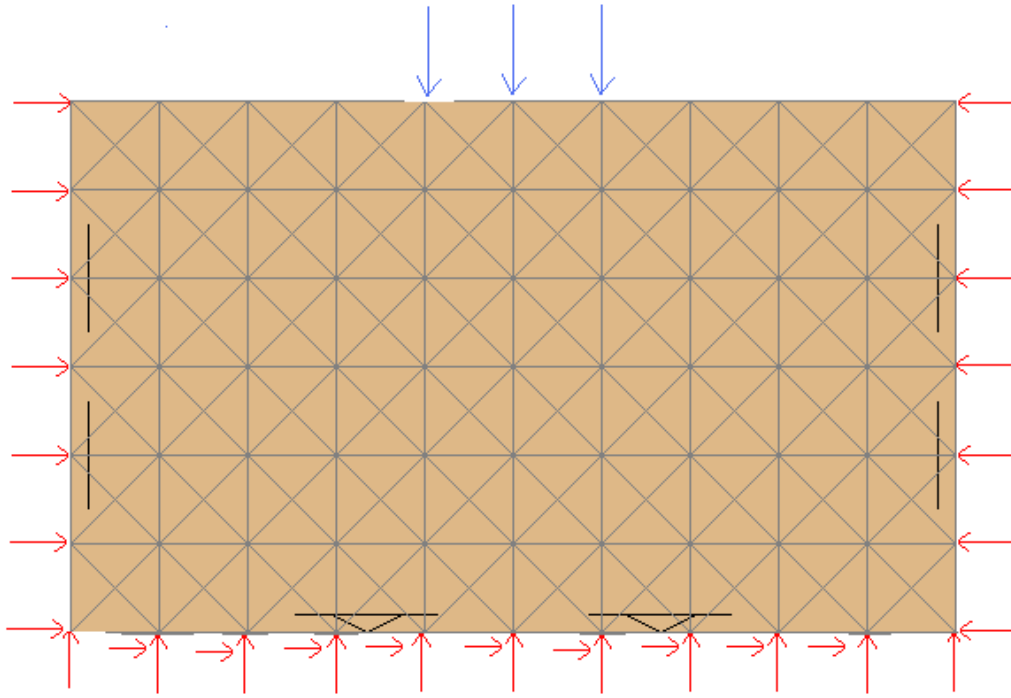
$$\text{The stiffness equation:} \quad \{F\} = [K]\{d\} \quad (2.1)$$

where  $\{F\}$  = global external forces vector

$[K]$  = global stiffness matrix

$\{d\}$  = structural displacement vector

All displacements and forces are associated with the nodes of a finite element mesh.



**Figure 2.1:** Nodal Forces on a Finite Element Mesh

The stiffness equation attributes forces induced at each node as a reaction to the combined action of displacements occurring at the node itself and every other node in the system.

$$\{F\} = [K] \{d\} \quad (2.1)$$

$$\begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{bmatrix} * \begin{Bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{Bmatrix}$$

$$F_1 = K_{11}d_1 + K_{12}d_2 + K_{13}d_3 + \dots + K_{1n}d_n$$

$$F_n = K_{n1}d_1 + K_{n2}d_2 + K_{n3}d_3 + \dots + K_{nn}d_n$$

=> Force induced at a node= sum of the products of displacement values of each node with a respective coefficient

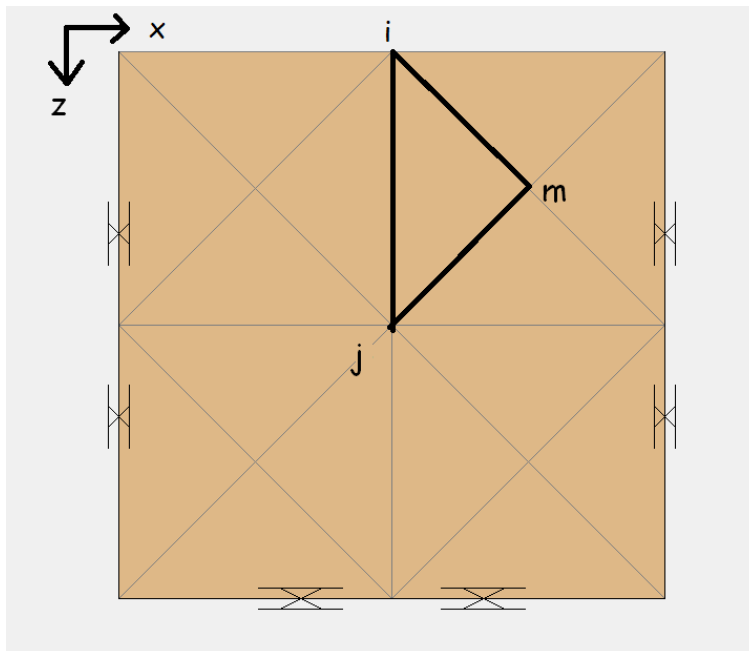
The global stiffness matrix,  $[K]$ , holds coefficients equivalent to the force induced at a node as a consequence of a unit displacement acting at each respective node of the system

Developing the stiffness matrix involves the following steps

- Selecting the type of element used to discretize the system (in this case a triangle)
- Selecting the displacement function (linear displacement function is used here)
- Defining the Strain-Displacement and Stress-Strain Relationship
- Deriving the elemental stiffness matrix and equations using the Total Potential Energy approach

#### 2.2.1.1. Selecting the Type of Element

The two dimensional soil matrix is discretized into sets of triangular elements. Each element has three nodes denoted  $i$ ,  $j$ , and  $m$ . Each node has 2 DOFs (displacement in  $x$  and  $z$  directions).



**Figure 2.2:** A Triangular Element in a Finite Element Mesh

Triangular elements have been chosen because triangles are the basic two dimensional shapes and also because the finite element expressions related to triangular element are simple.



Let  $u_i$  and  $v_i$  represent the displacement components of node  $i$  in the  $x$  and  $z$  directions, respectively.

The nodal displacements for an element with nodes  $i, j$ , and  $m$  are:

$$\{d\} = \begin{Bmatrix} d_i \\ d_j \\ d_m \end{Bmatrix} \quad \text{where} \quad \{d_i\} = \begin{Bmatrix} u_i \\ v_i \end{Bmatrix}$$

$$\text{Therefore: } \{d\} = \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \quad (2.2)$$

### 2.2.1.2. Selecting Displacement Functions

In order to compute the deformation parameters such as the normal and shear strains, the distribution of displacements throughout the element should get represented by a displacement function. Here, a linear displacement function is allocated for each triangular element, defined as [1]

$$\{\psi_i\} = \begin{Bmatrix} u(x, z) \\ v(x, z) \end{Bmatrix} = \begin{Bmatrix} a_1 + a_2x + a_3z \\ a_4 + a_5x + a_6z \end{Bmatrix} \quad (2.3)$$

$$\{\psi_i\} = \begin{Bmatrix} u(x, z) \\ v(x, z) \end{Bmatrix} = \begin{bmatrix} 1 & x & z & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & z \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix}$$

A linear displacement function ensures that the displacements along each edge of the element and the nodes shared by adjacent elements are equal.

To obtain the values for the “a”s, the coordinates of the nodal points are substituted into the above equations.

$$\begin{aligned} u_i &= a_1 + a_2x_i + a_3z_i & v_i &= a_4 + a_5x_i + a_6z_i \\ u_j &= a_1 + a_2x_j + a_3z_j & v_j &= a_4 + a_5x_j + a_6z_j \\ u_m &= a_1 + a_2x_m + a_3z_m & v_m &= a_4 + a_5x_m + a_6z_m \end{aligned} \quad (2.4)$$

Solving for the “a”s and writing the results in matrix form gives:

$$\begin{Bmatrix} u_i \\ u_j \\ u_m \end{Bmatrix} = \begin{bmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_m & z_m \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} \Rightarrow \{a\} = [x]^{-1} \{u\} \quad (2.5)$$

The inverse of the [x] matrix is:

$$[x]^{-1} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \quad (2.6)$$

Where

$$2A = \begin{vmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_m & z_m \end{vmatrix} \text{ is the determinant of } [x]$$

$$2A = x_i(z_j - z_m) + x_j(z_m - z_i) + x_m(z_i - z_j)$$

Where A is the area of the triangle and

$$\begin{aligned} \alpha_i &= x_j z_m - z_j x_m & \beta_i &= z_j - z_m & \gamma_i &= x_m - x_j \\ \alpha_j &= x_i z_m - z_i x_m & \beta_j &= z_m - z_i & \gamma_j &= x_i - x_m \\ \alpha_m &= x_i z_j - z_i x_j & \beta_m &= z_i - z_j & \gamma_m &= x_j - x_i \end{aligned} \quad (2.7)$$

The values of {a} may be written in matrix form as:

$$\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \\ u_m \end{Bmatrix} \quad \text{and} \quad \begin{Bmatrix} a_4 \\ a_5 \\ a_6 \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \begin{Bmatrix} v_i \\ v_j \\ v_m \end{Bmatrix} \quad (2.8)$$

then, deriving the displacement function in terms of the coordinates x and z gives;

$$\{u\} = [1 \ x \ z] \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix}$$

Substituting the values for “a” into the above equation gives:

$$\{u\} = \frac{1}{2A} \begin{bmatrix} 1 & x & z \end{bmatrix} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \\ u_m \end{Bmatrix}$$

$$\{u\} = \frac{1}{2A} \begin{bmatrix} 1 & x & z \end{bmatrix} \begin{bmatrix} \alpha_i u_i + \alpha_j u_j + \alpha_m u_m \\ \beta_i u_i + \beta_j u_j + \beta_m u_m \\ \gamma_i u_i + \gamma_j u_j + \gamma_m u_m \end{bmatrix}$$

Performing the matrix multiplication gives;

$$u(x,z) = \frac{1}{2A} \{ (\alpha_i + \beta_i x + \gamma_i z) u_i + (\alpha_j + \beta_j x + \gamma_j z) u_j + (\alpha_m + \beta_m x + \gamma_m z) u_m \}$$

a similar expression can be obtained for the z displacement

$$v(x,z) = \frac{1}{2A} \{ (\alpha_i + \beta_i x + \gamma_i z) v_i + (\alpha_j + \beta_j x + \gamma_j z) v_j + (\alpha_m + \beta_m x + \gamma_m z) v_m \}$$

the displacements can be written in a more convenient form as:

$$u(x,z) = N_i u_i + N_j u_j + N_m u_m \quad v(x,z) = N_i v_i + N_j v_j + N_m v_m \quad (2.9)$$

where

$$N_i = \frac{1}{2A} (\alpha_i + \beta_i x + \gamma_i z) \quad N_j = \frac{1}{2A} (\alpha_j + \beta_j x + \gamma_j z) \quad N_m = \frac{1}{2A} (\alpha_m + \beta_m x + \gamma_m z)$$

The elemental displacements can be summarized as;

$$\{\psi_i\} = \begin{Bmatrix} U(x,z) \\ V(x,z) \end{Bmatrix} = \begin{Bmatrix} N_i u_i + N_j u_j + N_m u_m \\ N_i v_i + N_j v_j + N_m v_m \end{Bmatrix} \quad (2.10)$$

In another form the above equations are;

$$\{\psi_i\} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix}$$

$$\{\Psi\} = [N] \{d\} \quad \text{where} \quad \{N\} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix}$$

### 2.2.1.3. Defining the Strain-Displacement and Stress-Strain Relationships

Elemental Strains: - For Plane Strain problems, the non-zero strain components are  $\varepsilon_x$ ,  $\varepsilon_z$  and  $\gamma_{xz}$ . These strain components are defined as;

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_z \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial z} \\ \frac{\partial u}{\partial z} + \frac{\partial v}{\partial x} \end{Bmatrix} \quad (2.11)$$

Substituting the approximation for the displacement gives;

$$\frac{\partial u}{\partial x} = u_{,x} = \frac{\partial}{\partial x} (N_i u_i + N_j u_j + N_m u_m)$$

$$u_{,x} = N_{i,x} u_i + N_{j,x} u_j + N_{m,x} u_m$$

the derivatives of the interpolation functions are;

$$N_{i,x} = \frac{1}{2A} \frac{\partial}{\partial x} (\alpha_i + \beta_i x + \gamma_i z) = \frac{\beta_i}{2A} \quad N_{j,x} = \frac{\beta_j}{2A} \quad N_{m,x} = \frac{\beta_m}{2A} \quad (2.12)$$

therefore,

$$\frac{\partial u}{\partial x} = \frac{1}{2A} (\beta_i u_i + \beta_j u_j + \beta_m u_m) \quad (2.13)$$

In a similar manner, the remaining strain terms can be expressed as;

$$\frac{\partial v}{\partial z} = \frac{1}{2A} (\gamma_i v_i + \gamma_j v_j + \gamma_m v_m) \quad (2.14)$$

$$\frac{\partial u}{\partial z} + \frac{\partial v}{\partial x} = \frac{1}{2A} (\gamma_i u_i + \beta_i v_i + \gamma_j u_j + \beta_j v_j + \gamma_m u_m + \beta_m v_m) \quad (2.15)$$

One can write the strains in matrix form as

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_z \\ \gamma_{xz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \quad (2.16)$$

For axisymmetric analysis, the  $\varepsilon_\theta$  strain component which is analogous to the  $\varepsilon_y$  of plane strain problems is non-zero. To keep a general format for both kinds of problems the  $\varepsilon_y$  component can be incorporated as follows;

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_z \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \quad (2.17)$$

These equations can be written in matrix form as;

$$\{\varepsilon\} = [B]\{d\} \quad (2.18)$$

where

$$[B] = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix}$$

### Stress-Strain Relationship:

The generalized Hooke's law for isotropic materials is expressed in matrix form as follows: [10]

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \varepsilon_{xy} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \end{bmatrix} = \begin{bmatrix} 1/E & -\nu/E & -\nu/E & 0 & 0 & 0 \\ -\nu/E & 1/E & -\nu/E & 0 & 0 & 0 \\ -\nu/E & -\nu/E & 1/E & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2G & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2G & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2G \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix}$$

considering that

$$G = \frac{E}{2(1+\nu)} \quad \text{and} \quad \gamma = 2\varepsilon \quad \dots \text{for the shear strains;}$$

the above set of equations could be rearranged into

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5-\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5-\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5-\nu \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix}$$

For Plane strain, the strains  $\varepsilon_y$ ,  $\gamma_{xy}$ , and  $\gamma_{yz}$  are equal to zero

The above equation could be simplified and rearranged into;

$$\begin{Bmatrix} \sigma_x \\ \sigma_z \\ \sigma_y \\ \tau_{xz} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & 0.5-\nu \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_z \\ \varepsilon_y \\ \gamma_{xz} \end{Bmatrix}$$

or;

$$\begin{Bmatrix} \sigma_x \\ \sigma_z \\ \sigma_y \\ \tau_{xz} \end{Bmatrix} = [D] \begin{Bmatrix} \varepsilon_x \\ \varepsilon_z \\ \varepsilon_y \\ \gamma_{xz} \end{Bmatrix} \quad (2.19)$$

Where,

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & 0.5-\nu \end{bmatrix}; \text{ Stress-strain matrix} \quad \dots(2.20)$$

In short, stresses can be related to strains by the equation;

$$\{\sigma\} = [D]\{\varepsilon\} \quad (2.21)$$

**Note:** Though the  $\sigma_y$  is not necessarily zero for plane strain cases, references generally skip this component and express the  $D$  matrix in a three by three format to display components of the equation within a single plane. In this presentation, however, the  $\sigma_y$  component is maintained and the  $D$  matrix is provided in a four by four format which is analogous to the case of axi-symmetric cases.

#### 2.2.1.4. Derivation of the Elemental Stiffness Matrix Using the Total Potential Energy Approach

The aim of this subchapter is to derive the stiffness matrix which is a component of the stiffness equation given by;

$$\{F\}=[K]\{d\} \quad (2.1)$$

where  $\{F\}$  = external forces vector

$[K]$  = stiffness matrix

$\{d\}$  = displacements vector

The derivation will be carried out using the total potential energy approach.[1]

The total potential energy is defined as the sum of the internal strain energy  $U$  and the potential energy of the external forces  $\Omega$ :

$$\pi_p = U + \Omega_b + \Omega_p + \Omega_s \quad (2.22)$$

Where the strain energy is ;

$$U = \frac{1}{2} \int_v \{\varepsilon\}^T \{\sigma\} dV \quad \Rightarrow \quad U = \frac{1}{2} \int_v \{\varepsilon\}^T [D] \{\varepsilon\} dV \quad (2.23)$$

The potential energy of the body force term is;

$$\begin{aligned} \Omega_b &= - \int_v \{\Psi\}^T \{X\} dV \\ &= - \int_v \{d\}^T [N]^T \{X\} dV \end{aligned} \quad (2.24)$$

Where  $\{\Psi\}$  is the general displacement function, and  $\{X\}$  is the body weight per unit volume.

Where the potential energy of the concentrated forces is ;

$$\Omega_p = -\{d\}^T \{P\} \quad (2.25)$$

Where  $\{P\}$  are the concentrated forces, and  $\{d\}$  are the nodal displacements.

The potential energy of the distributed loads is;

$$\begin{aligned} \Omega_s &= - \int_s \{\Psi\}^T \{T\} dS \\ &= - \int_s \{d\}^T [N]^T \{T\} dS \end{aligned} \quad (2.26)$$

Where  $\{\Psi\}$  is the general displacement function, and  $\{T\}$  are the surface tractions.

Then the total potential energy expression becomes:

$$\pi_p = \frac{1}{2} \int_v \{d\}^T [B]^T [D] [B] \{d\} dV - \int_v \{d\}^T [N]^T \{X\} dV - \{d\}^T \{P\} - \int_s \{d\}^T [N]^T \{T\} dS$$

The nodal displacements  $\{d\}$  are independent of the general x-z coordinates, therefore

$$\pi_p = \frac{1}{2} \{d\}^T \int_v [B]^T [D] [B] dV \{d\} - \{d\}^T \int_v [N]^T \{X\} dV - \{d\}^T \{P\} - \{d\}^T \int_s [N]^T \{T\} dS$$

The last three terms can be defined as:

$$\{f\} = \int_v [N]^T \{X\} dV + \{P\} + \int_s [N]^T \{T\} dS$$

Therefore:

$$\pi_p = \frac{1}{2} \{d\}^T \int_v [B]^T [D] [B] dV \{d\} - \{d\}^T \{f\} \quad (2.27)$$

Stability occurs when the system's potential energy is at its minimum. At the minimum potential energy the first derivate should be equal to zero.

Minimization of  $\pi_p$  with respect to each nodal displacement requires that;

$$\frac{\partial \pi_p}{\partial \{d\}} = \int_v [B]^T [D] [B] dV \{d\} - \{f\} = 0$$

The above relationship requires;

$$\{f\} = \int_v [B]^T [D] [B] dV \{d\}$$

The stiffness matrix can be defined as:

$$[k] = \int_v [B]^T [D] [B] dV$$

For a two dimensional element, it is convenient to use a thickness of one unit, the above integral reduces to;

$$[k] = \int_A [B]^T [D] [B] dx dz$$

For the case of linear interpolation functions, the integrand in the above equation is not a function of x or z (global coordinates); therefore, the integration reduces to:

$$[k] = [B]^T [D] [B] \int_A dx dz$$

$$[k] = A [B]^T [D] [B] \quad (2.28)$$

Where “A” is the area of the triangular element.



### 2.2.2. Considerations for Axisymmetric Problems

Axisymmetric problems involve conditions that are symmetrical about a central-vertical axis. The mathematical problems presented are similar to those of plane strain due to the fact that both conditions are two dimensional. One basic difference is that in axisymmetric problems one deals with cylindrical coordinate system and hence applies the r-z coordinate system rather than the x-z coordinate system. Consequently, the terminologies used for the non-zero strain and stress components are;

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} \quad \text{and} \quad \boldsymbol{\sigma} = \begin{Bmatrix} \sigma_r \\ \sigma_z \\ \sigma_\theta \\ \tau_{rz} \end{Bmatrix}, \text{ respectively.} \quad (2.29)$$

Furthermore, unlike its plane strain counterpart, the  $\varepsilon_\theta$  strain component is not necessarily zero. For the axisymmetric case, the strain components are [4]

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial v}{\partial z} \\ \frac{u}{r} \\ \frac{\partial u}{\partial z} + \frac{\partial v}{\partial r} \end{Bmatrix} \quad (2.30)$$

from  $\{\varepsilon\} = [B]\{d\}$  and following similar steps to that of the plane strain case;

$$[B] = \frac{1}{2A} \begin{bmatrix} \frac{\partial N_i}{\partial r} & 0 & \frac{\partial N_j}{\partial r} & 0 & \frac{\partial N_m}{\partial r} & 0 \\ 0 & \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_j}{\partial z} & 0 & \frac{\partial N_m}{\partial z} \\ \frac{N_i}{r} & 0 & \frac{N_j}{r} & 0 & \frac{N_m}{r} & 0 \\ \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial r} & \frac{\partial N_j}{\partial z} & \frac{\partial N_j}{\partial r} & \frac{\partial N_m}{\partial z} & \frac{\partial N_m}{\partial r} \end{bmatrix}$$

$$[B] = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ \frac{\alpha_i}{r} + \beta_i + \frac{\gamma_i z}{r} & 0 & \frac{\alpha_j}{r} + \beta_j + \frac{\gamma_j z}{r} & 0 & \frac{\alpha_m}{r} + \beta_m + \frac{\gamma_m z}{r} & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix} \quad (2.31)$$

With the **B** matrix now involving the variable coordinates  $r$  and  $z$ , the strains are no longer constant within an element as in the plane strain case. In order to overcome the complications involved in the subsequent volume integral evaluation,  $r$  and  $z$  can be substituted by the centroidal coordinates  $\bar{r}$  and  $\bar{z}$ .

The **D** matrix remains the same as the in the case of plane strain problems;

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & 0.5-\nu \end{bmatrix} \quad (2.20)$$

The stiffness matrix of the element can now be computed following similar steps as in the case of plane strain. Remembering that the volume integral has to be taken over the whole ring of material gives;

$$[k] = 2\pi \int [B]^T [D] [B] r dr dz$$

The integration cannot be performed as simply as in the plane strain case due to involvement of the non constant  $r$  and  $z$  variables. A concise and accurate enough simplification involves substituting the centroidal coordinates in place of the variable ones.

$$\bar{r} = \frac{\bar{r}_i + \bar{r}_j + \bar{r}_m}{3} \quad \text{and} \quad \bar{z} = \frac{\bar{z}_i + \bar{z}_j + \bar{z}_m}{3}$$

with the above simplification,

$$[k] = 2\pi [\bar{B}]^T [D] [\bar{B}] \bar{r} A \quad (2.32)$$

### 2.2.3. Summary of Useful Equations

The stiffness equation (the force-displacement relationship)

$$\{F\}=[K]\{d\}$$

Where,  $[k] = \int_v [B]^T [D] [B] dV$

For two dimensional elements with linear displacement functions;

$$[k] = A[B]^T [D] [B] \quad \text{for plane strain problems} \quad (2.28)$$

$$[k] = 2\pi\bar{r} A [\bar{B}]^T [D] [\bar{B}] \quad \text{for axisymmetric problems} \quad (2.32)$$

The strain-displacement relationship

$$\{\varepsilon\} = [B]\{d\} \quad (2.18)$$

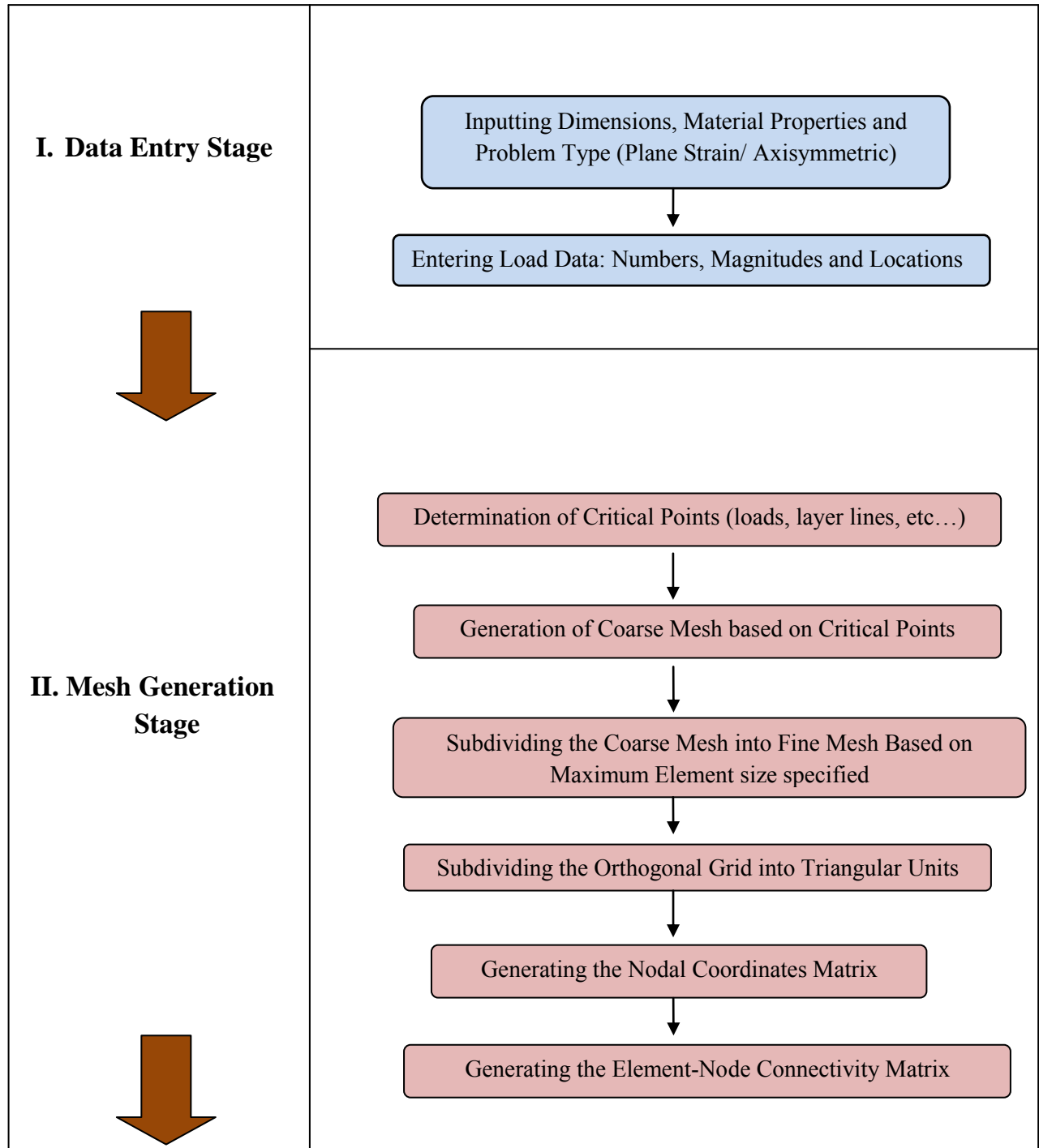
The stress-strain relationship

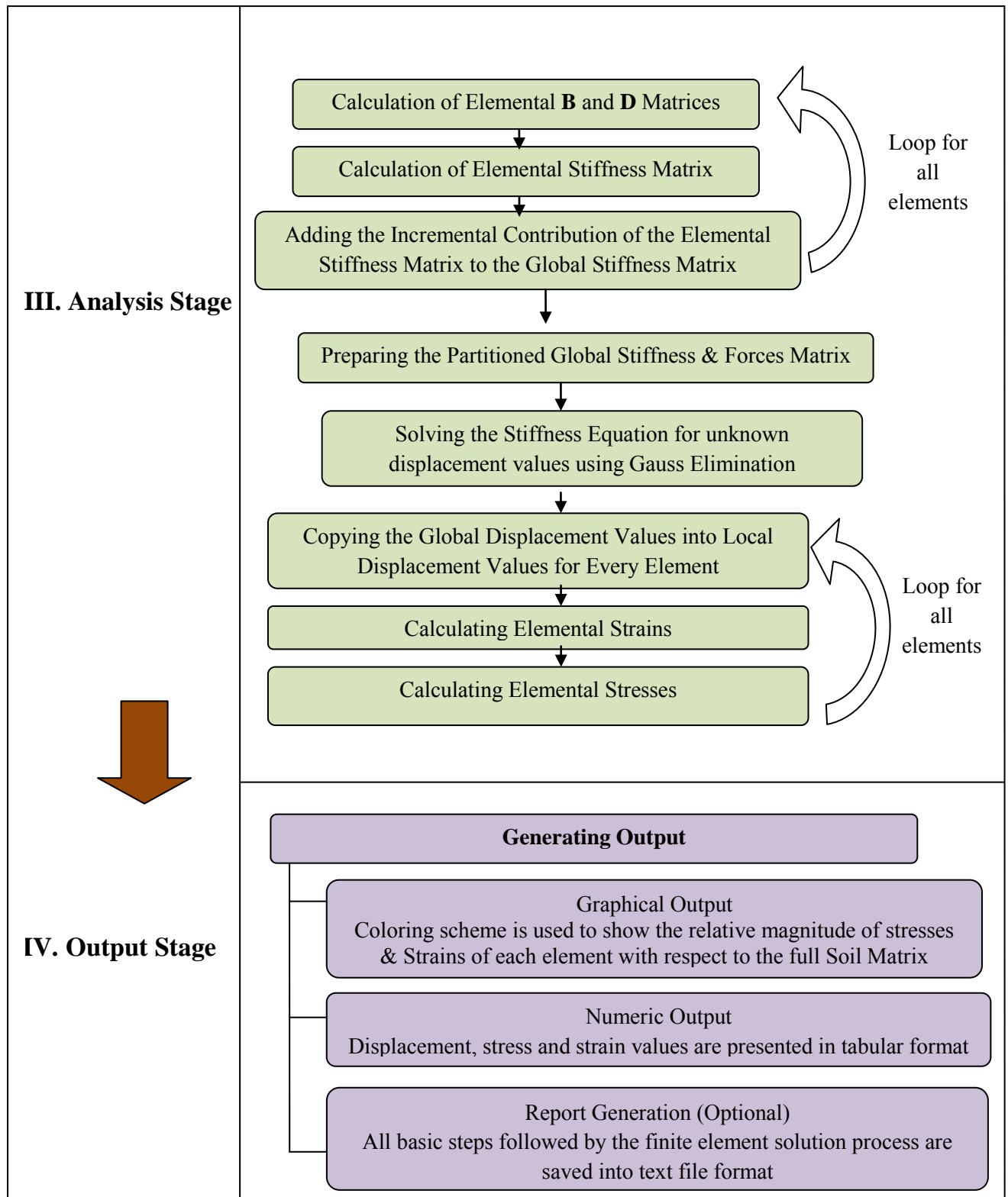
$$\{\sigma\} = [D][\varepsilon] \quad (2.21)$$

### 3. FEM Software Development

#### 3.1. Program Flow Chart

The finite element software developed implements the following flow chart.





**Figure 3.1:** Flowchart of the Finite Element Software

### **3.2. Program Modules Incorporated in the Software**

#### **3.2.1. The SOIL DATA ENTRY Program Module**

Data Entry is the first step taken by the finite element software. Here, the program prompts the user to enter

- dimensions of the soil matrix to be used for analysis,
- total number of soil layers in the problem, and
- thickness and elastic properties ( $E$  and  $\nu$ ) of each layer

Here, the user specifies whether the problem type is plane strain or axisymmetric. In this stage there is also an option of selecting between two types of boundary conditions. In either case the boundaries are fixed against lateral movement ( $x$  or  $r$  directions for plane strain and axis-symmetric cases, respectively) while it is possible to either fix or release the left & right boundaries in the vertical ( $z$ ) direction. For the default case, the program considers laterally and vertically restrained boundaries.

After the various data are entered, the values are stored in respective variables.

#### **3.2.2. The LOADING DATA ENTRY Program Module**

Loading Data Entry is the second step taken by the finite element software. Here, the program prompts the user to enter

- number of concentrated loads to be applied,
- magnitude and location of each concentrated load,
- number of distributed loads to be applied, and
- magnitude and location of each distributed load

After the various data are entered, the values are stored in respective variables.

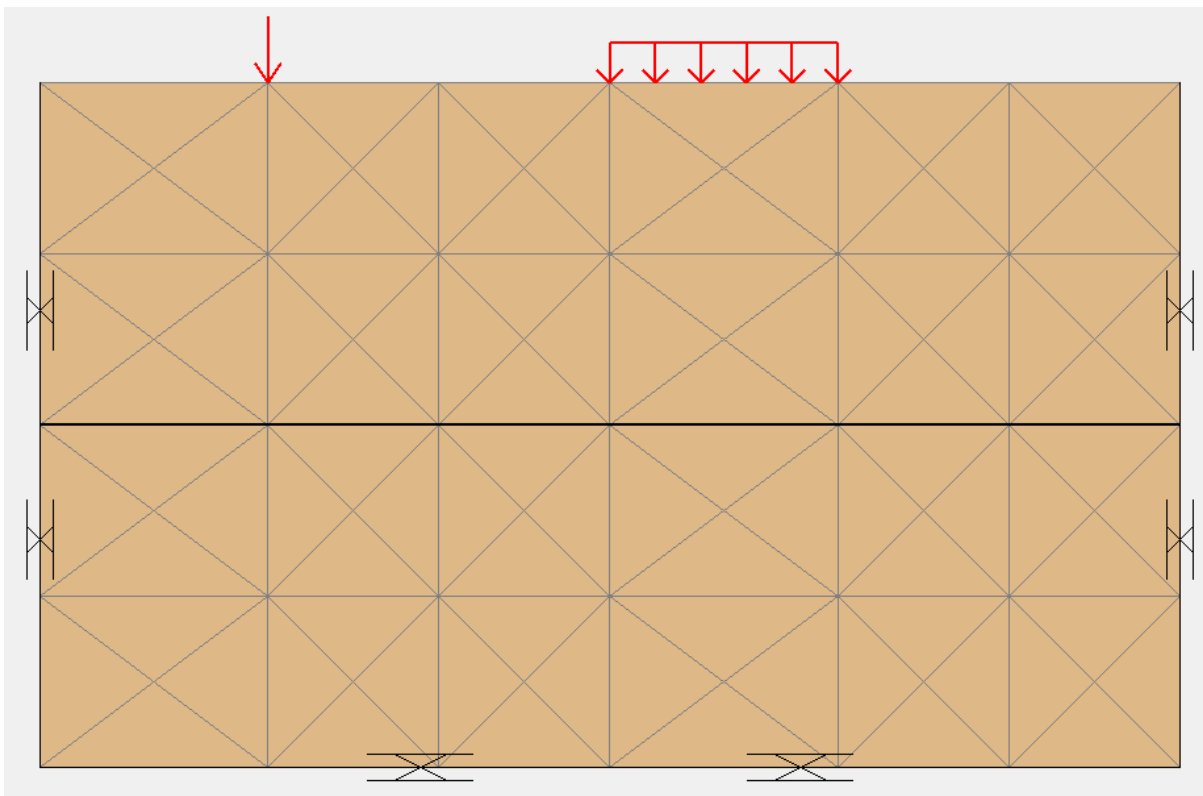
#### **3.2.3. The GENERATE MESH Program Module**

The third step taken by the software is mesh generation. Here, the following set of procedures is executed.

- Critical points such as location of loads and layer lines are identified.
- A coarse mesh is generated based on the critical points identified.
- The minimum dimension between girds of the coarse mesh is identified.
- The coarse mesh is further subdivided into fine mesh based on either the maximum element size specified by the user or the minimum dimension identified between any two critical points (coarse mesh), whichever is smaller.

While the default maximum mesh dimension is the largest aspect divided by 40, the user has the option of specifying a smaller mesh dimension.

- The forces matrix is generated based on the concentrated and distributed loads applied. The distributed loads are converted to equivalent point loads that are applied at the nearest nodes.
- The fine orthogonal mesh generated in previous steps is further subdivided into triangular elements. Following, the nodal coordinates list is generated.
- A list of the unconstrained degrees of freedom is generated to be used later for partitioning the stiffness and forces matrices.



**Figure 3.2:** Sample Case of Discretized Soil Matrix

#### **3.2.4. The RUN ANALYSIS Program Module**

The Analysis program executes the main finite element method procedures. This involves the following three steps.

Step 1. Calculation of Elemental Stiffness Matrices and Assembly of the Global Stiffness matrix.

This step involves a loop that calculates the stiffness matrix of each element. Here the elemental **B** matrix is calculated using the Equations (2.18) & (2.31):

$$[B] = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix}$$

and

$$[B] = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ \frac{\alpha_i}{r} + \beta_i + \frac{\gamma_i z}{r} & 0 & \frac{\alpha_j}{r} + \beta_j + \frac{\gamma_j z}{r} & 0 & \frac{\alpha_m}{r} + \beta_m + \frac{\gamma_m z}{r} & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix}$$

for plane strain and axisymmetric problems, respectively.

Then the elemental **D** matrix is calculated using Equation (2.20):-

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & 0.5-\nu \end{bmatrix}$$

When there is more than one layer, elements within different layers will have different **D** matrix values. Otherwise, the D matrix will be the same for all elements.

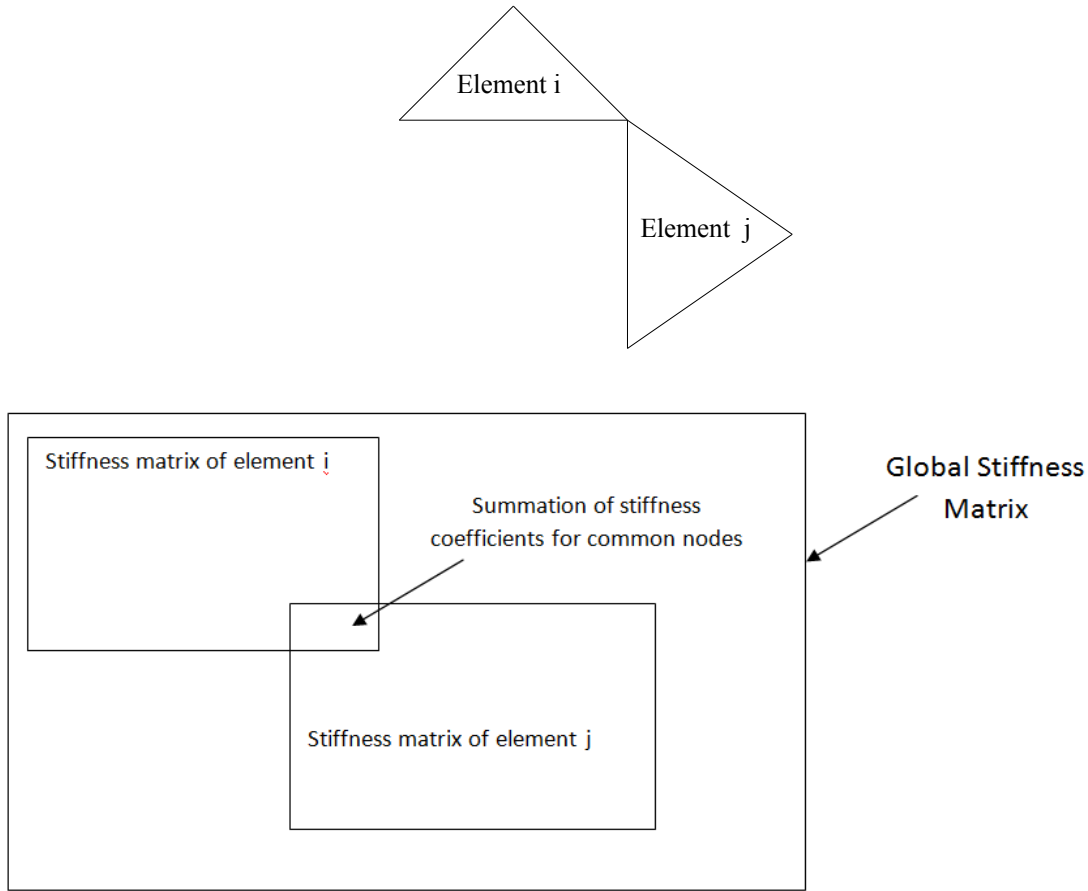
Following, the elemental stiffness matrix, **k**, is calculated using the equations:-

$$[k] = A[B]^T[D][B] \quad \text{for plane strain problems and}$$

$$[k] = 2\pi\bar{r} A [\bar{B}]^T[D][\bar{B}] \quad \text{for axisymmetric problems}$$

With every loop, the incremental contribution of the elemental stiffness matrix is added to the global stiffness matrix, **K**, to give the final assembled global stiffness matrix upon completion of the looping procedure.





**Figure 3.3:** Summation of Stiffness Coefficients for Common Nodes

### Step 2. Solving the Stiffness Equation for the Unknown Displacement Values

The next step involves generating the partitioned global stiffness and forces matrices. The partitioned matrices involve only the components of the unrestrained DOFs. Since it is only at the unrestrained DOFs that the displacement values are not known, it is sufficient to express the stiffness equation in terms of the unrestrained DOFs. The program collects elements that correspond to unrestrained DOFs from the stiffness and forces matrices to construct the partitioned stiffness and forces matrices.

$$\begin{array}{c} \text{known forces} \\ \text{unknown forces} \end{array} \left\{ \begin{array}{c} F_1 \\ F_2 \\ \vdots \\ F_n \end{array} \right\} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{bmatrix} * \begin{array}{c} \text{Unknown} \\ \text{displacements} \\ \text{known} \\ \text{displacement} \\ \text{values} \end{array} \left\{ \begin{array}{c} d_1 \\ d_2 \\ \vdots \\ d_n \end{array} \right\}$$

**Figure 3.4:** Partitioning the Stiffness Equation

Taking the partitioned matrices leaves an equation of the form;

$$\{f\}_{\text{known}} = [k] \{d\}_{\text{unknown}}$$

Following, the stiffness equation is solved by using gauss elimination procedure. Since most components of the stiffness matrix are zero, the Gauss elimination provides a convenient procedure for solving the matrix equation and determining the unknown displacements matrix, **d**.

### Step 3. Calculating Elemental Strains and Stresses

In order to calculate the elemental stresses and strains, the local displacement value of each element's DOFs should be referred from the global displacement matrix. A looping procedure copies values of local displacements, calculates the elemental strains and then calculates the elemental stresses for each element.

The elemental strains are calculated using the equation:-







$$\{\varepsilon\} = [B]\{d\} \quad (2.18)$$

The elemental stresses are calculated using the equation:-

$$\{\sigma\} = [D][\varepsilon] \quad (2.19)$$

### **3.2.5. The OUTPUT Program Module**

The OUPUT program module presents the calculation output of the analysis stage to the user. Output is provided in graphical, numeric and text report formats. In the graphical presentation, the maximum stress and strain values are determined and a color scheme is used to show the relative magnitudes of stresses and strains of each element with respect to the full soil matrix. To this effect, the module identifies the maximum stress or strain and categorizes the remaining elements into six groups as follows.

Group	Color Code	Description
I.		>75% of the maximum stress/strain
II.		between 50% and 75% of the maximum stress/strain
III.		between 25% and 50% of the maximum stress/strain
IV.		between 0% and 25% of the maximum stress/strain
V.		Stress/strain equals zero
VI.		Stress/strain is less than zero (applied to normal stresses and strains only)

**Table 3.1:** Color Code of the Finite Element Software Graphic Output

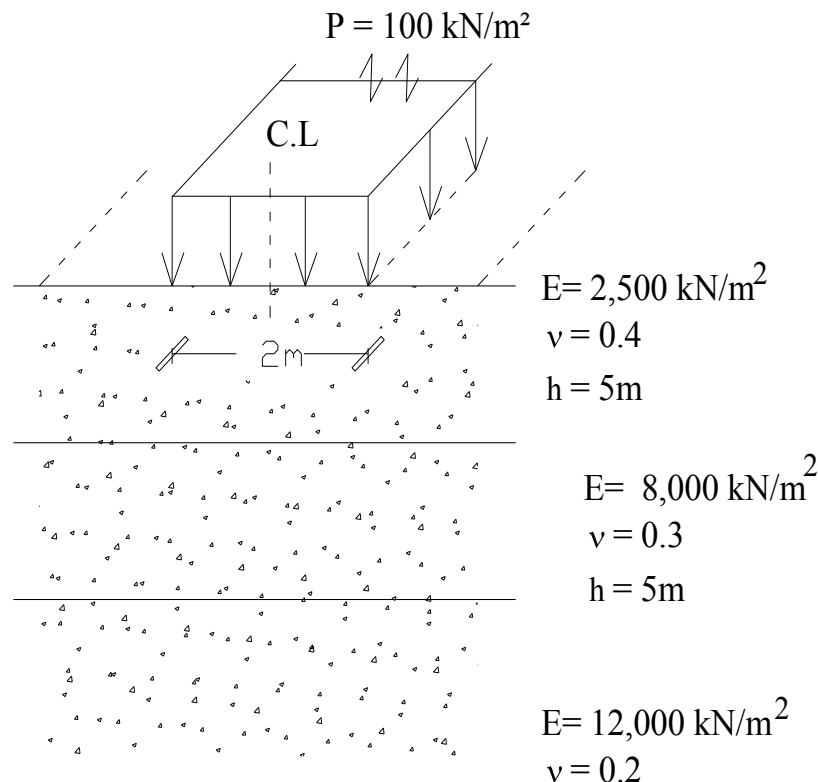
Along with the graphical output, a numeric output is presented in tabular format displaying values of displacement, stresses and strains. Furthermore, the user can get settlement, stress and strain output for any location on the soil matrix model just by pointing the computer mouse at the desired location.

In addition, the user has an option of getting a text report generated regarding the analysis conducted. The report, generated in text file format, displays the value of each variable at each calculation step of the analysis stage.

### 3.3. User Interfaces and Application Example

In this sub chapter, the usage of the finite element software will be demonstrated through an example problem. The problem involves a strip load acting upon the surface of a triple layered soil system as shown in the figure below.

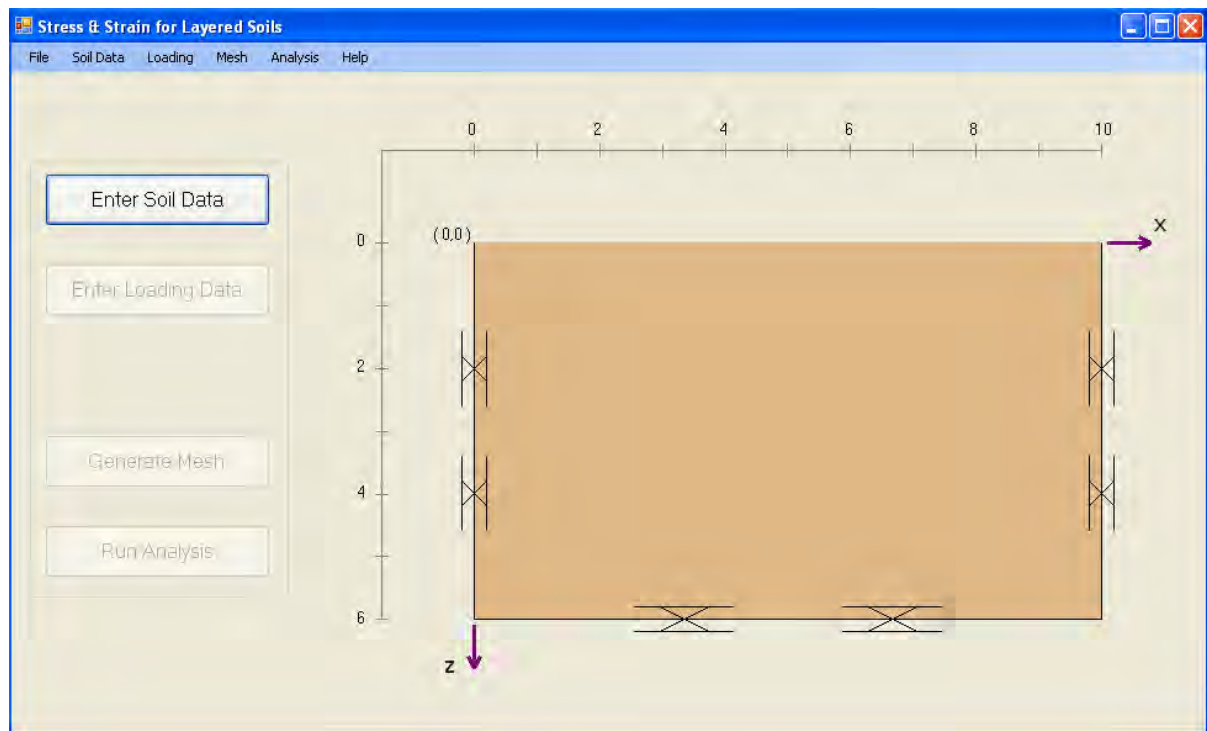
It is required to know the settlement, stress and strain distributions through the foundation soil.



**Figure 3.5:** A Strip Load Acting on a Three Layered System Problem

Prior to beginning analysis, the user should have the software copied on a computer. There is no installation procedure for the software as it is sufficient to copy the folder containing the executable software file and other associated files to any desired directory. However, in order to run the executable file, the computer should have the support component, .NET framework 4, or a more recent version installed on the computer. This support component is provided along the FEM software and can be installed by double clicking its icon. Once the .NET framework is installed, the FEM software could be run as many times as needed without having to reinstall the support component.

To begin analysis of the problem, the user starts the finite element software by double clicking the FEM software icon. Upon execution, the home screen of the user interface appears as shown in Figure 3.6



**Figure 3.6:** The Home Screen of the FEM Software

The home screen of the finite element software consists of the menu bar, the four basic application buttons and the graphical display area. The four buttons are the “Enter Soil Data” button, the “Enter Loading Data” button, the “Generate Mesh” button and the “Run Analysis” button. The buttons are prepared to let a user make a simple use of the application with default setting of the software. A user who intends to manipulate more advanced features of the software can access them through the menu bar. As an initial step, the user begins by clicking the “Enter Soil Data Button”. Following the button click, the soil data entry interface appears.

This interface has been prepared to let the user provide parameters regarding size of soil matrix to be used for analysis and material properties for each layer. The dimensions of soil matrix should be large enough that the stresses in the boundaries will be much lower than maximum stress expected to occur on elements closest to the load. On the other hand, if one takes very large dimensions, it won’t be possible to get output data in close intervals. For this specific problem, a width of 20m and a total depth of 15 meters will be used. Here, though the depth of the bottom layer is supposed to extend indefinitely, a depth of 5m will be assigned to the program. The total depth taken by the model is many times larger than the width of the load and hence the output is not expected to be significantly affected by allocation of a finite depth to the problem.

After entering the given information from the problem and keeping the default Plane Strain Setting of the software, the Soil Data Entry interface will look like the one shown in Figure 3.7.

**Dimensions of Soil Matrix**

Width =  m      Depth =  m

**Soil Properties**

Number of Soil Layers=

Layer	Thickness (m)	Modulus of Elasticity(kN/m <sup>2</sup> )	Poisson's Ratio
1	5	2500	0.4
2	5	8000	0.3
3	5	12000	0.2

**Type of Problem**

☒ Plane Strain  
☐ Axisymmetric

Apply      Cancel

**Figure 3.7:** The Soil Data Entry Interface of the FEM Software

Following soil data entry, the user clicks the apply button and gets returned to the home screen. After the soil data entry, the user clicks the ENTER LOADING DATA button which initiates the loading data entry interface to appear. On this interface, the user provides the magnitude and location of loads. The strip load of the example problem falls under the distributed load category and it is most convenient to apply the load at the center of the soil matrix model. After entry of the required information, the interface looks like the one shown in Figure 3.8.

**Loading Data Input**

**Loading**

**Concentrated Load**

Number of Load Cases	Magnitude (kN/m)	Location (m)	Depth (m)
0			

**Distributed Load**

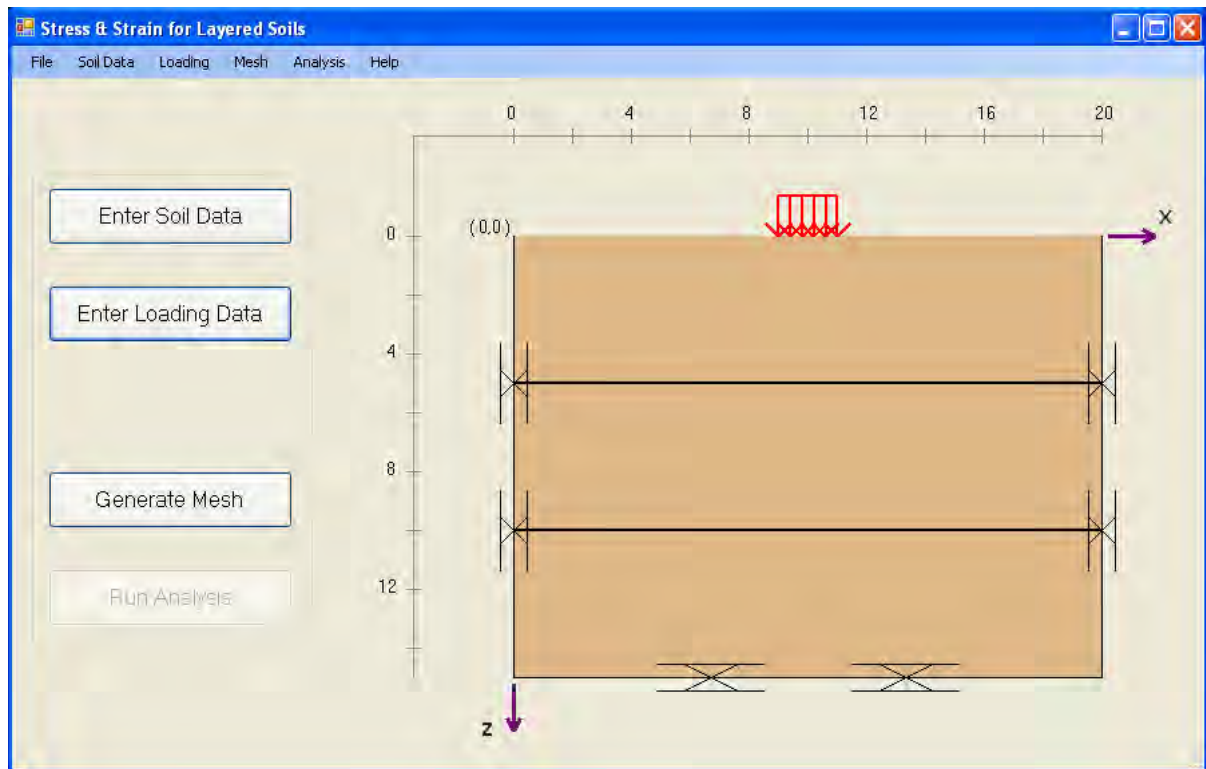
Number of Load Cases	Magnitude (kN/m²)	Beginning (m)	End (m)	Depth (m)
1	100	9	11	0

Note: The Concentrated Load Shown here is equivalent to Continuous Line Loading  
The Distributed Load shown here is equivalent to Uniform Strip Load

Apply Cancel

**Figure 3.8:** The Loading Data Entry Interface of the FEM Software

Following, the user clicks the apply button which leads back to the home screen of the software. Here it is possible to see that the layer lines have been marked and the load has been drawn as shown in Figure 3.9.

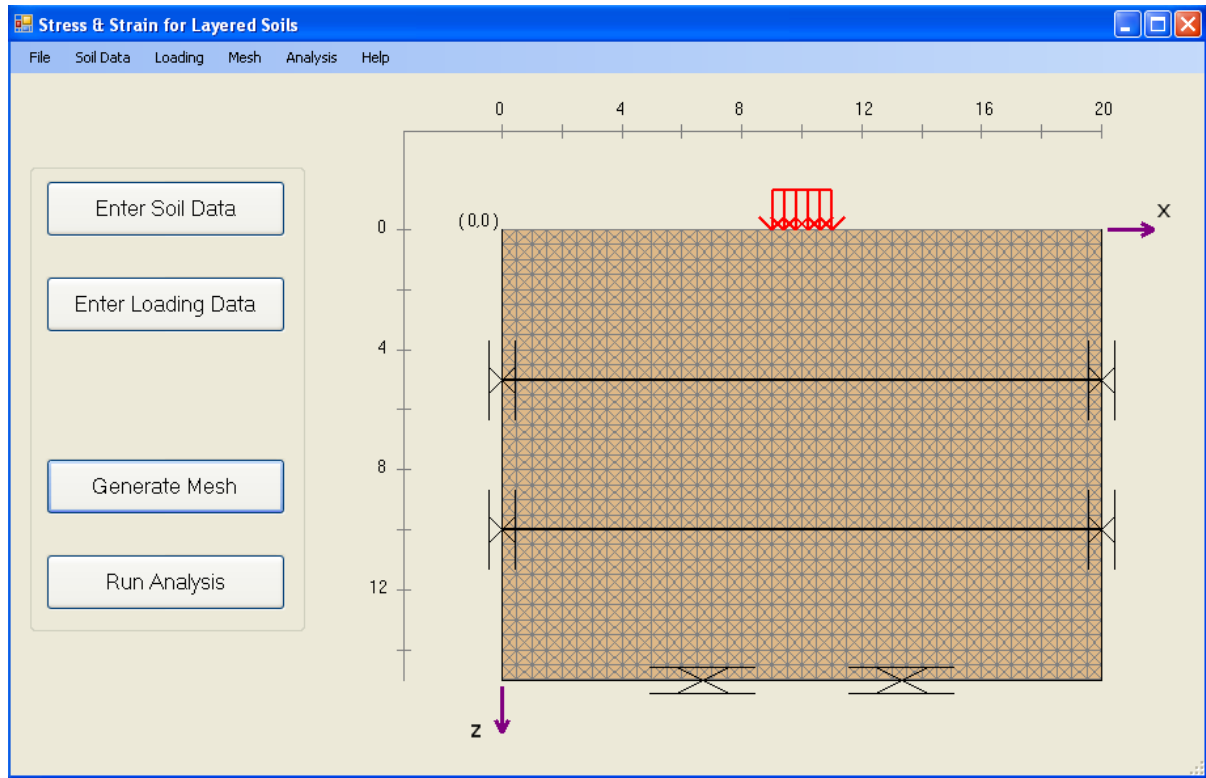


**Figure 3.9:** The Home Screen after Soil Data and Loading Data Entry

Next the user clicks the Generate Mesh button. For the default case, the software will generate a mesh with elements of maximum size equal to the largest dimension of the model divided by 40. Alternatively, the user can click on the “Mesh” menu item and select the “Edit Mesh Size” option to modify maximum mesh dimension prior to generating the mesh.

After the mesh generation, the home screen will look like the one shown in Figure 3.10.

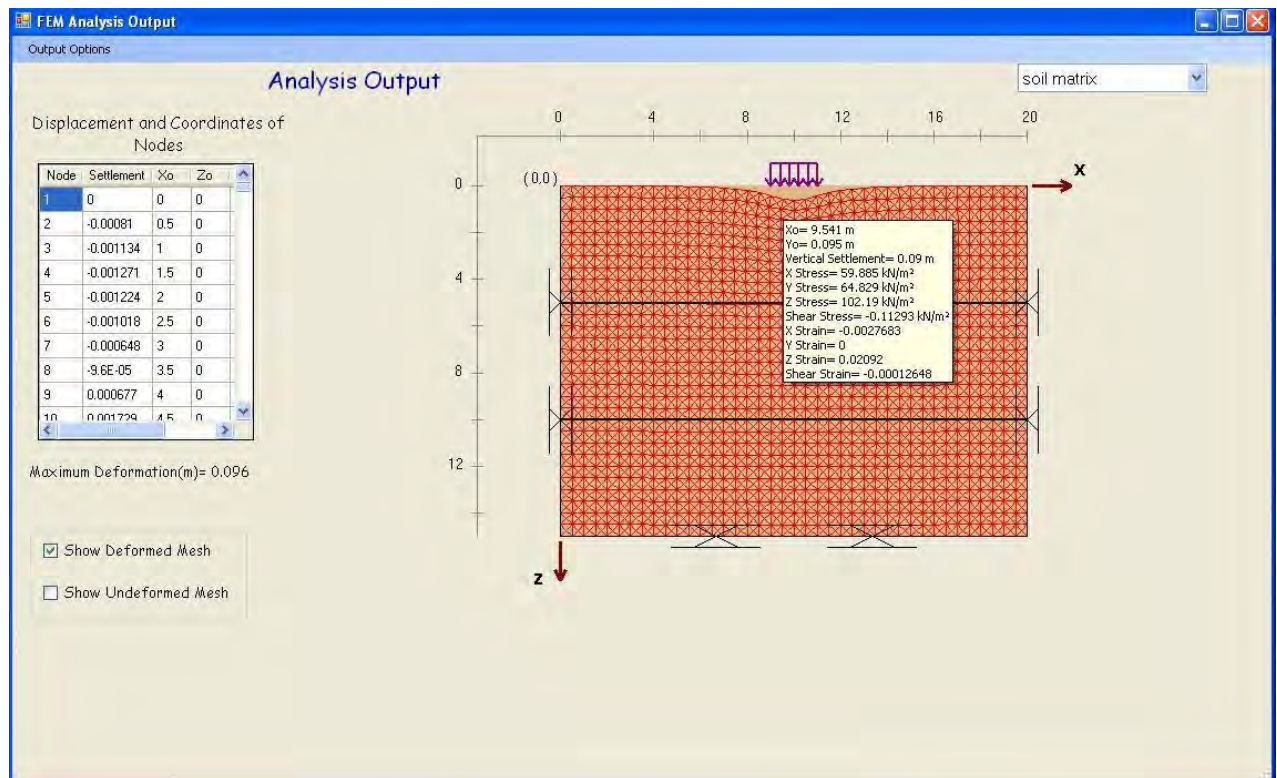




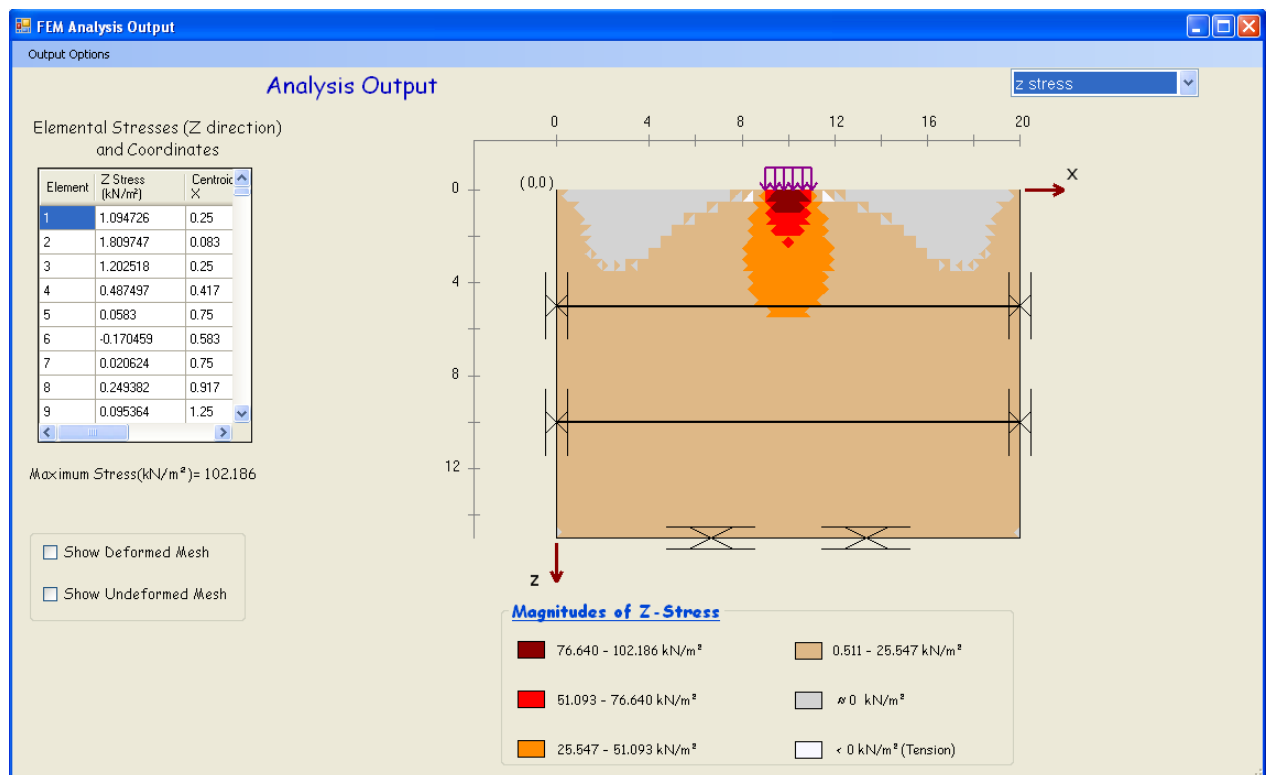
**Figure 3.10:** The Home Screen after Mesh Generation

At this stage, the software has collected all the necessary information and is ready to conduct the finite element analysis. Clicking the RUN ANALYSIS button will start the solution procedure. When the program finishes, the output interface appears by displaying the deformed mesh and the numeric output of the nodal deformations as shown in Figure 3.11.

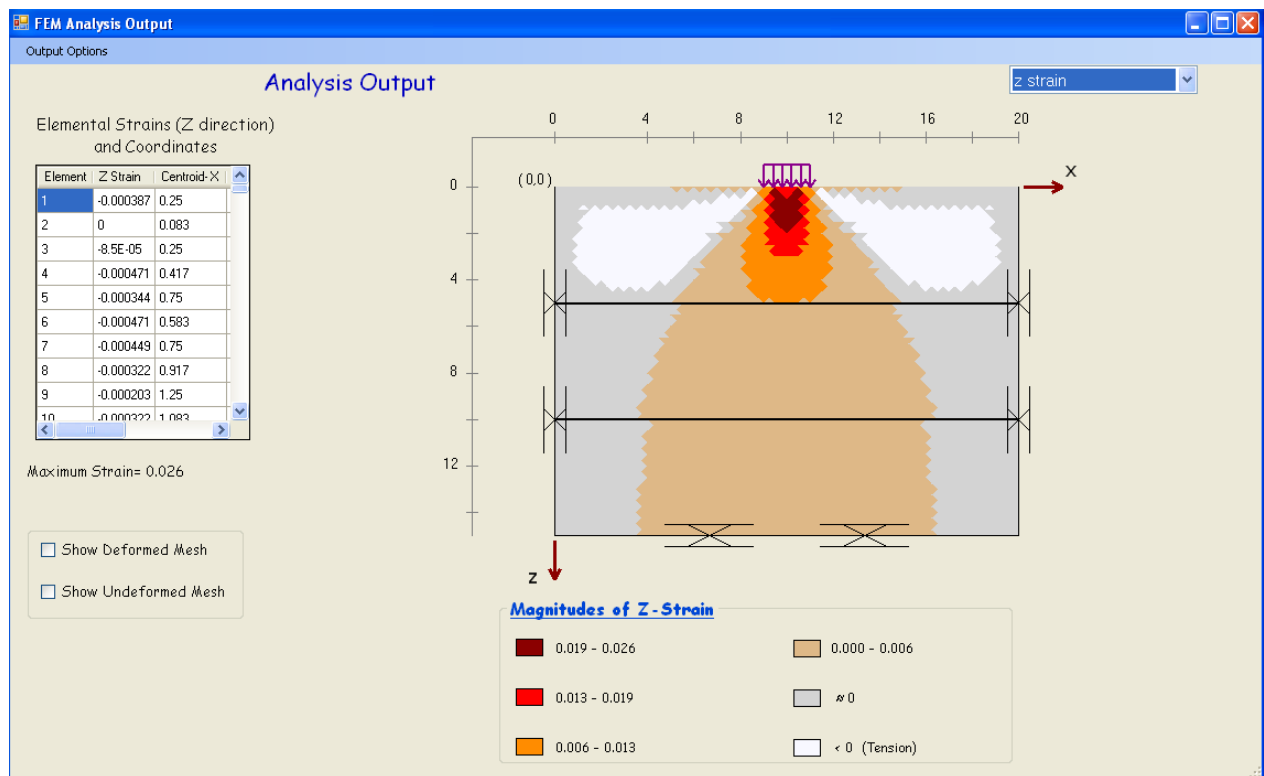
Alternatively, the user could have any of the stress and strain outputs displayed. As a sample, the vertical stress output and the vertical strain output are displayed in Figures 3.12 and 3.13.



**Figure 3.11:** The Output Interface Displaying Deformed Mesh and Output Data at the Mouse Pointer



**Figure 3.12:** The Output Interface Displaying Stress in the z direction



**Figure 3.13:** The Output Interface Displaying z-Strain Distribution

Furthermore, the user could request for an analysis report to be generated by the software. This is achieved by clicking the output options menu item and selecting the GENERATE REPORT option. Upon receiving this command, the software will generate a report of the calculations taken. The report displays the value of the finite element equation variables at each calculation step and other relevant information such as the nodal coordinates and the element-node connectivity matrix.

A comparison of this software's output against the output of the software GeoStudio 2007 has been conducted for this problem along with other problems in Chapter 4.

## **4. ANALYSIS AND COMPARISON**

### **4.1. Introduction**

In this chapter the output of the Finite element software is compared with output from other software and solutions of closed form equations where available. Comparison is performed for sample problems involving point loads, line loads and uniform strip loads. Comparison is conducted for cases of one, two and three layers. For the case of one layer, in addition to surface loads, loads embedded within the soil medium are also considered. The output parameters used for comparison are;

- Vertical displacement,
- Vertical stress,
- Vertical strain, and
- Shear Stress

The commercially available software, GeoStudio 2007, is used for comparing the analysis results. While the SIGMA/W module of GeoStudio 2007 can compute stress and deformation values of geotechnical problems, the application is currently available in student licenses and is limited to using a maximum of 500 elements only.

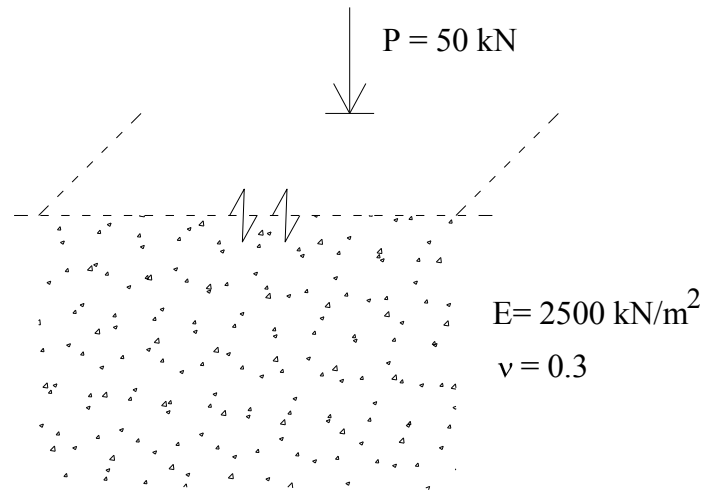
## 4.2. Comparison of Analysis Results Against Available Equations and Other Software

### 4.2.1. Single Layer (Elastic Isotropic Half-Space)

#### 4.2.1.1. Surface Loads

##### 4.2.1.1.1. Point Load

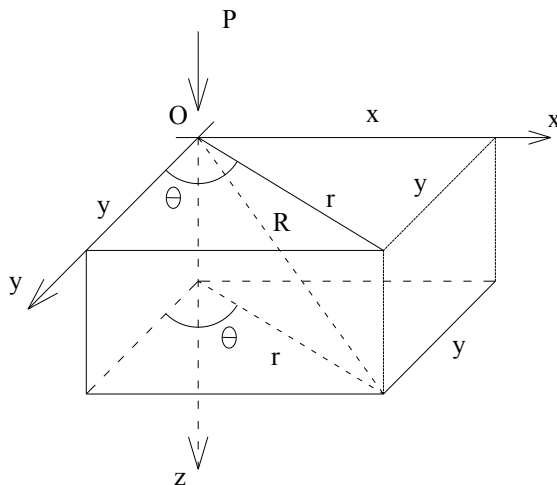
Sample problem 1: Point Load Acting on the Surface of a Single Layer



**Figure 4.1**

For the problem of a point load acting on the surface of a single layer system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Furthermore, the output results for vertical and shear stresses are compared with available equations. Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of results is performed for a plane at a radial distance of 1.5m from the point load.

Equations of Gray [2] for a point load acting on the surface of an elastic half space are given;



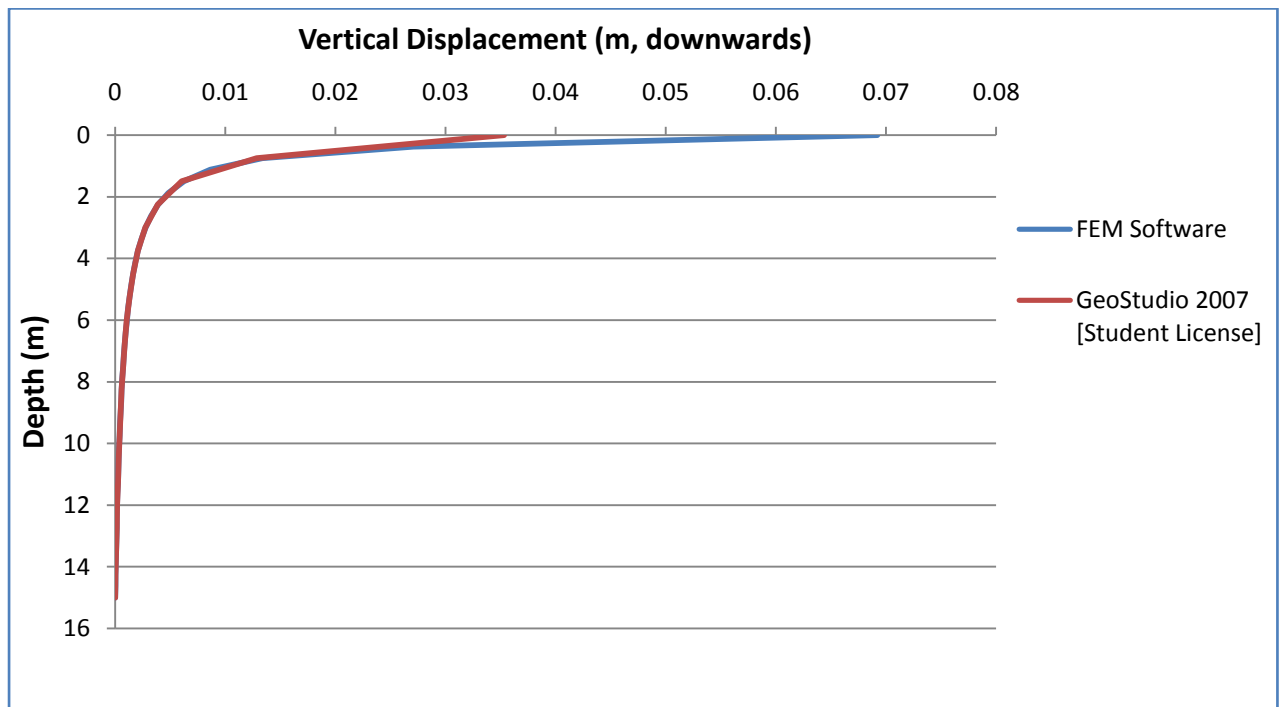
$$\sigma_r = \frac{P}{2\pi} \left[ 3 \frac{r^2 z}{R^5} - \frac{m-2}{m} \left\{ \frac{R-z}{Rr^2} \right\} \right]$$

$$\sigma_\theta = \frac{P}{2\pi} \left[ \frac{m-2}{m} \left\{ \frac{1}{r^2} - \frac{z}{R^3} - \frac{z}{Rr^2} \right\} \right]$$

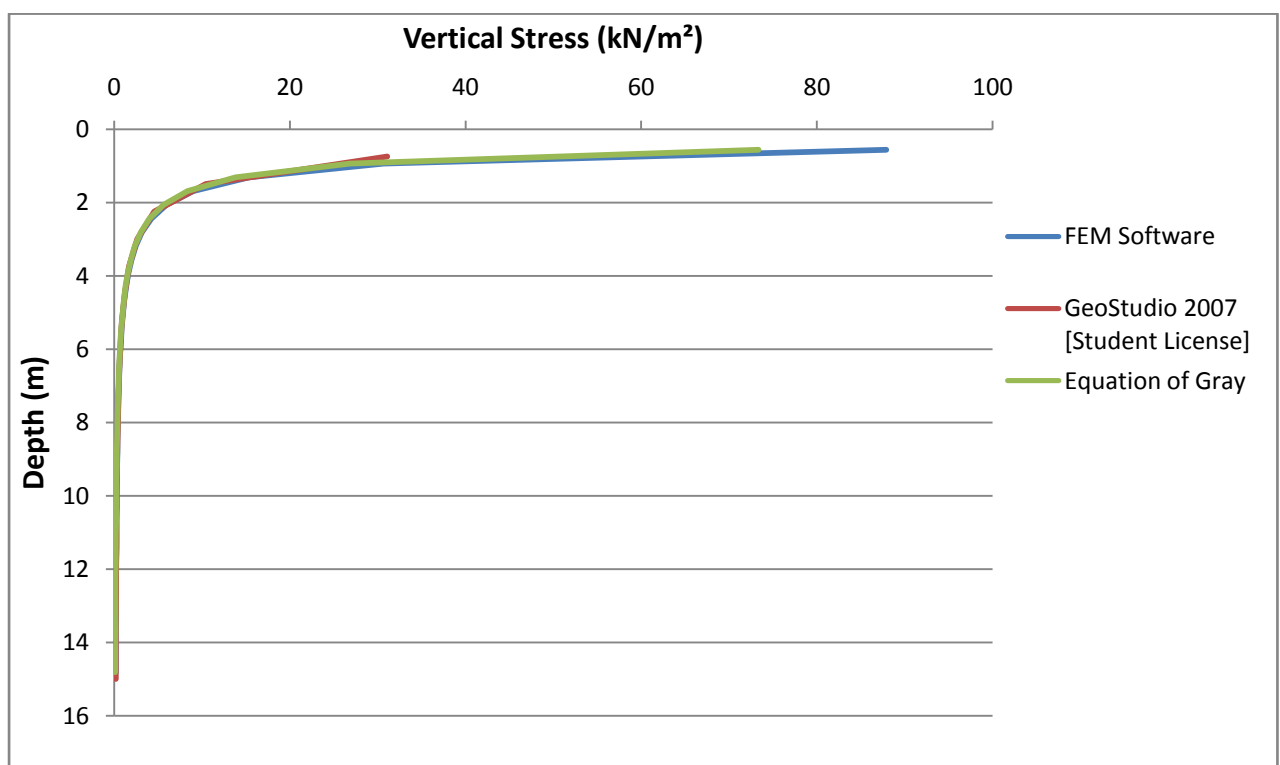
$$\sigma_z = \frac{3P}{2\pi} * \frac{z^3}{R^5}$$

$$\tau_{rz} = \frac{3P}{2\pi} * \frac{rz^2}{R^5}$$

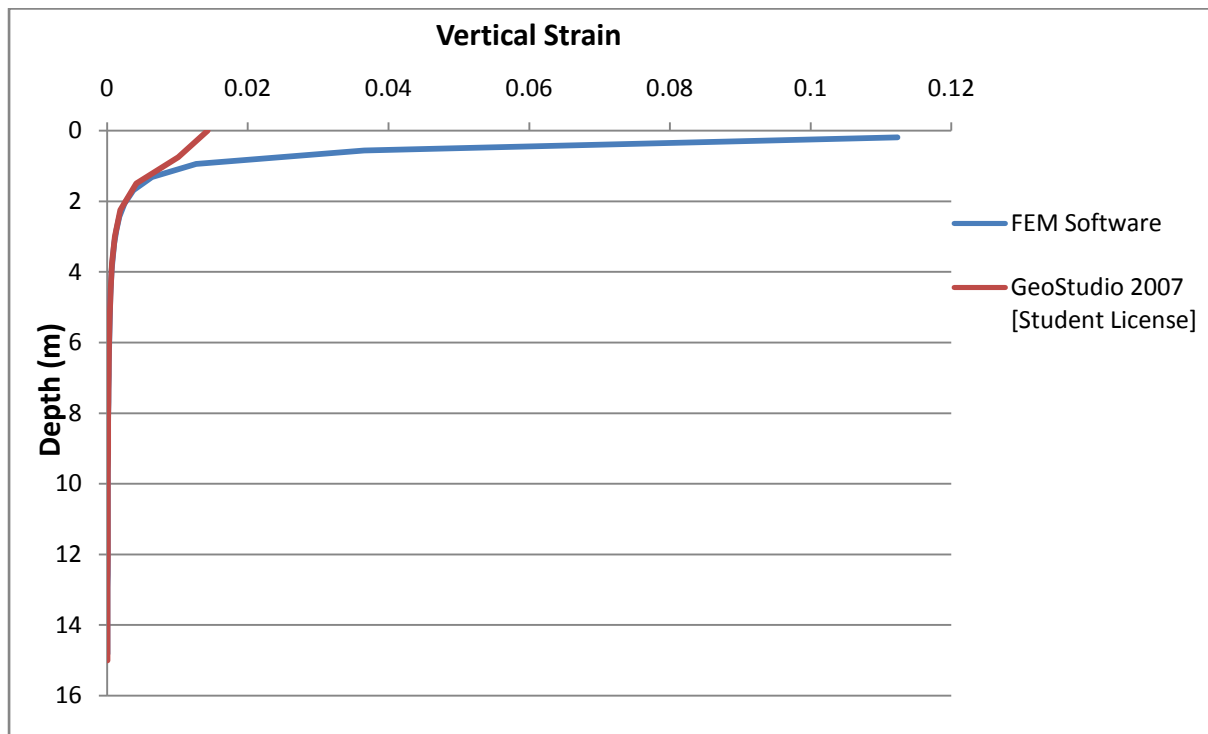
$m = \text{Poisson's number}$



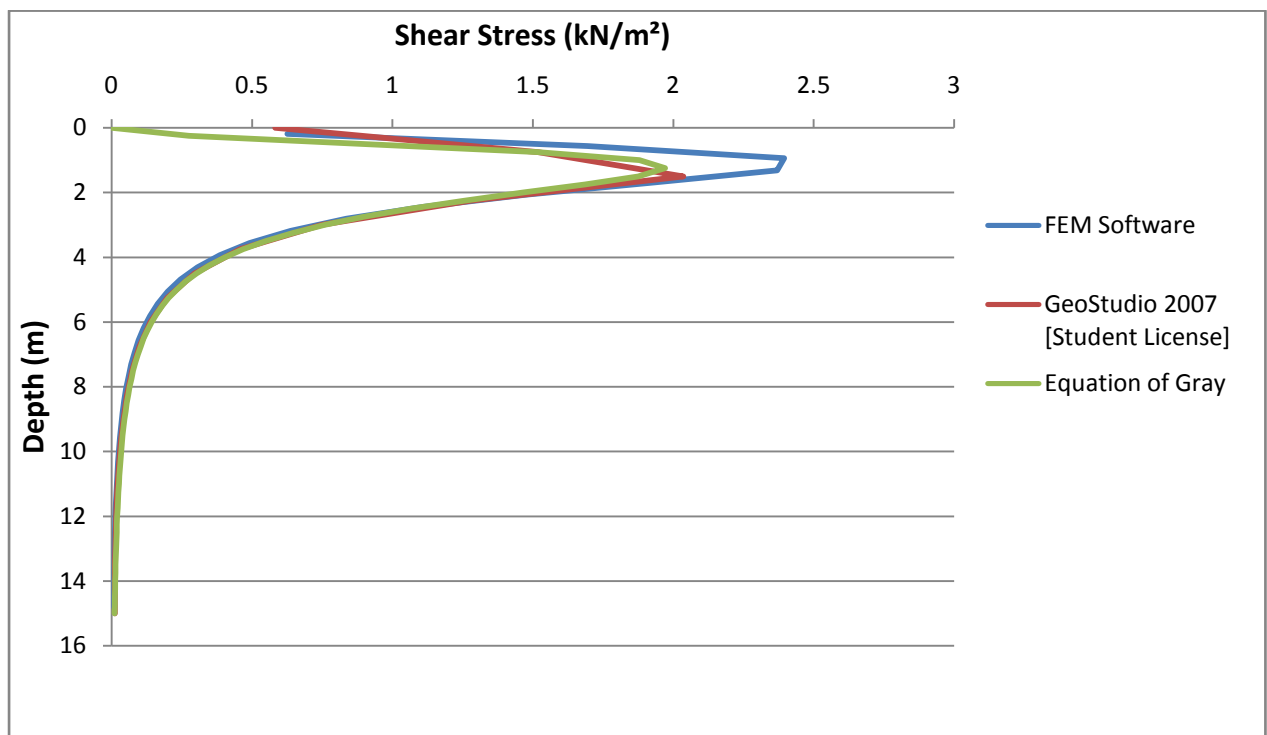
**Figure 4.2:** Distribution of Vertical Displacement along the Axis of a Point Load Acting on the Surface of a Single Layered System



**Figure 4.3:** Distribution of Vertical Stress along the Axis of a Point Load Acting on the Surface of a Single Layered System



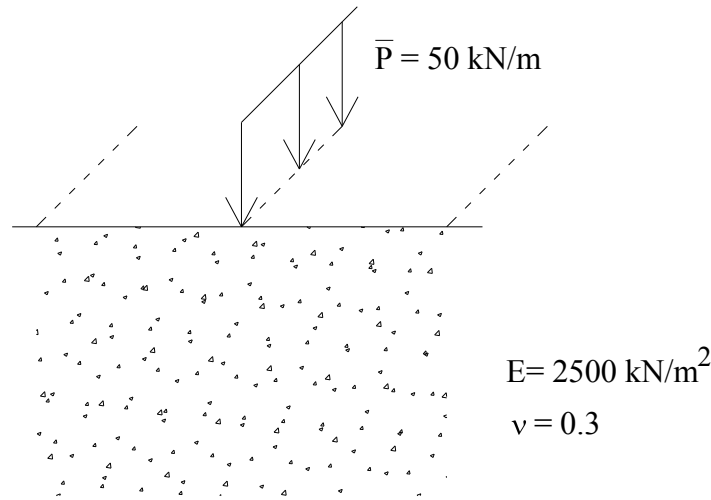
**Figure 4.4:** Distribution of Vertical Strain along the Axis of a Point Load Acting on the Surface of a Single Layered System



**Figure 4.5:** Distribution of Shear Stress along a Plane at a radial distance of 1.5m from a Point Load Acting on the Surface of a Single Layered System

#### 4.2.1.1.2. Line Load

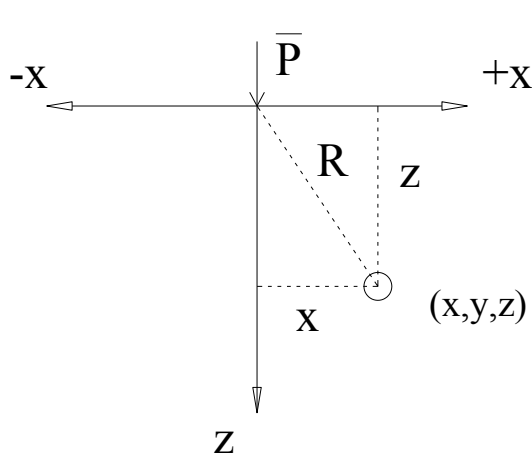
##### Sample problem 2: Line Load Acting on the Surface of a Single Layer



**Figure 4.6**

For the problem of a line load acting on the surface of a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Furthermore, the output results for vertical and shear stresses are compared with available equations. Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis parallel to the plane of loading at a distance of 1.5 meters.

Equations of Gray [2] for a line load acting on the surface of an elastic half space are given;



$$\sigma_x = \frac{2\bar{P}}{\pi} * \frac{x^2 z}{R^4}$$

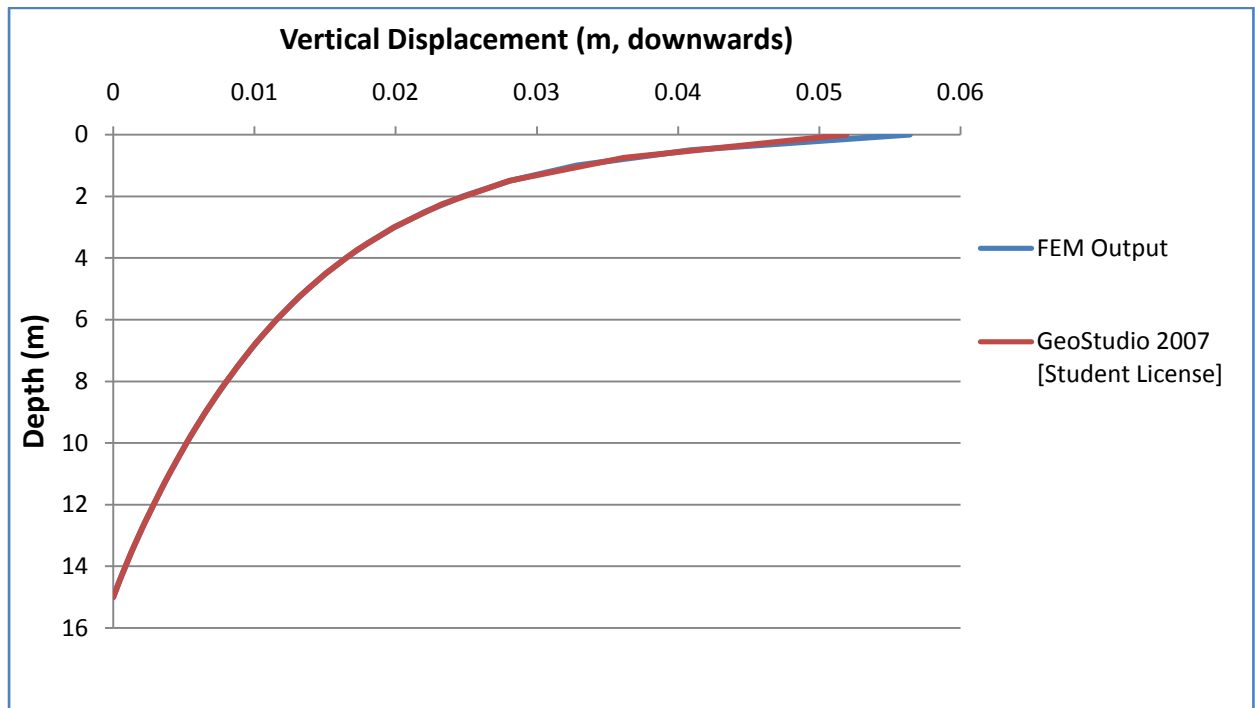
$$\sigma_y = \frac{2\bar{P}}{\pi m} * \frac{z}{R^2}$$

$$\sigma_z = \frac{2\bar{P}}{\pi} * \frac{z^3}{R^4}$$

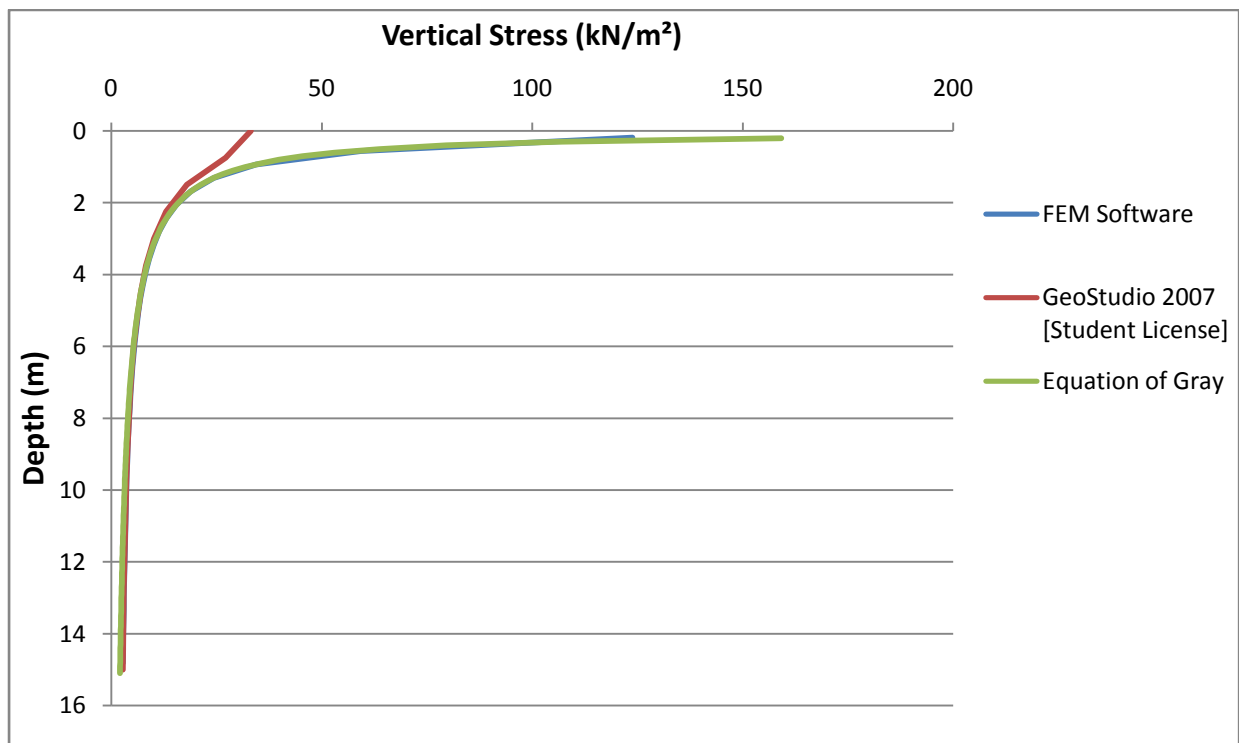
$$\tau_{xy} = \frac{2\bar{P}}{\pi} * \frac{xz^2}{R^4}$$

m=Poisson's number

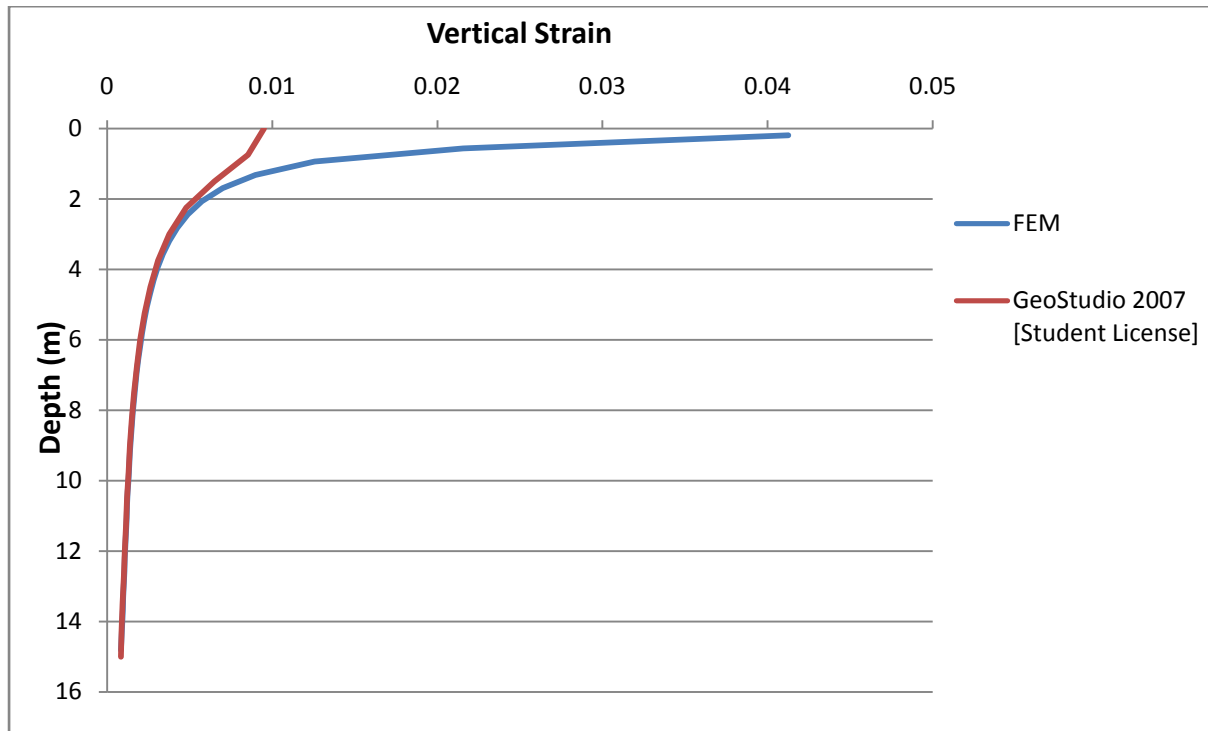




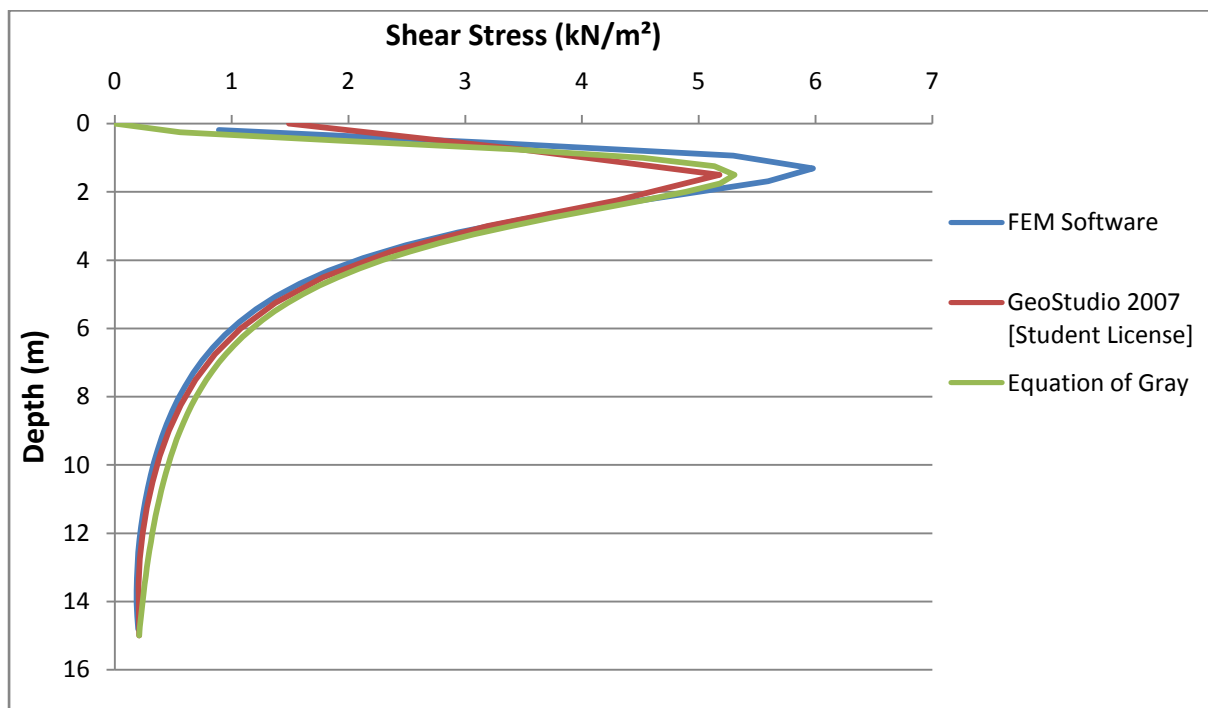
**Figure 4.7:** Distribution of Vertical Displacement along the Axis of Line Load Acting on the Surface of a Single Layered System



**Figure 4.8:** Distribution of Vertical Stress along the Axis of a Line Load Acting on the Surface of a Single Layered System



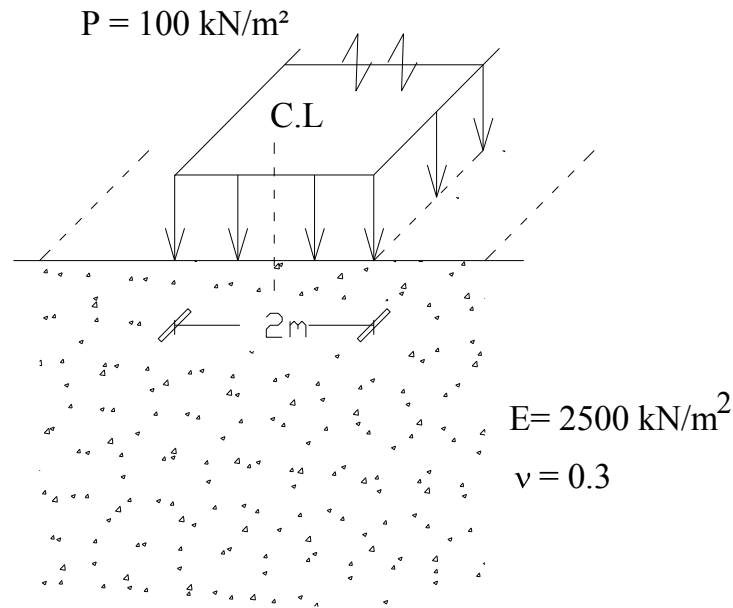
**Figure 4.9:** Distribution of Vertical Strain along the Axis of a Line Load Acting on the Surface of a Single Layered System



**Figure 4.10:** Distribution of Shear Stress along an Axis 1.5m Offset from a Line Load Acting on the Surface of a Single Layered System

#### 4.2.1.1.3. Uniform Strip Load

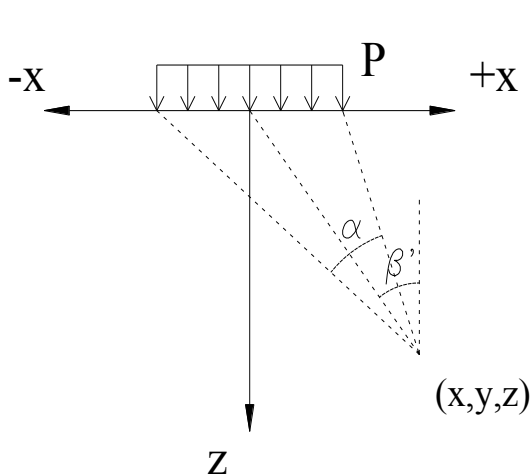
Sample problem 3: Strip Load Acting on the Surface of a Single Layer



**Figure 4.11**

For the problem of a uniformly distributed strip load acting on the surface of a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Furthermore, the output results for vertical and shear stresses are compared with available equations. Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of strip loading.

Equations of Gray [2] for a strip load acting on the surface of an elastic half space;



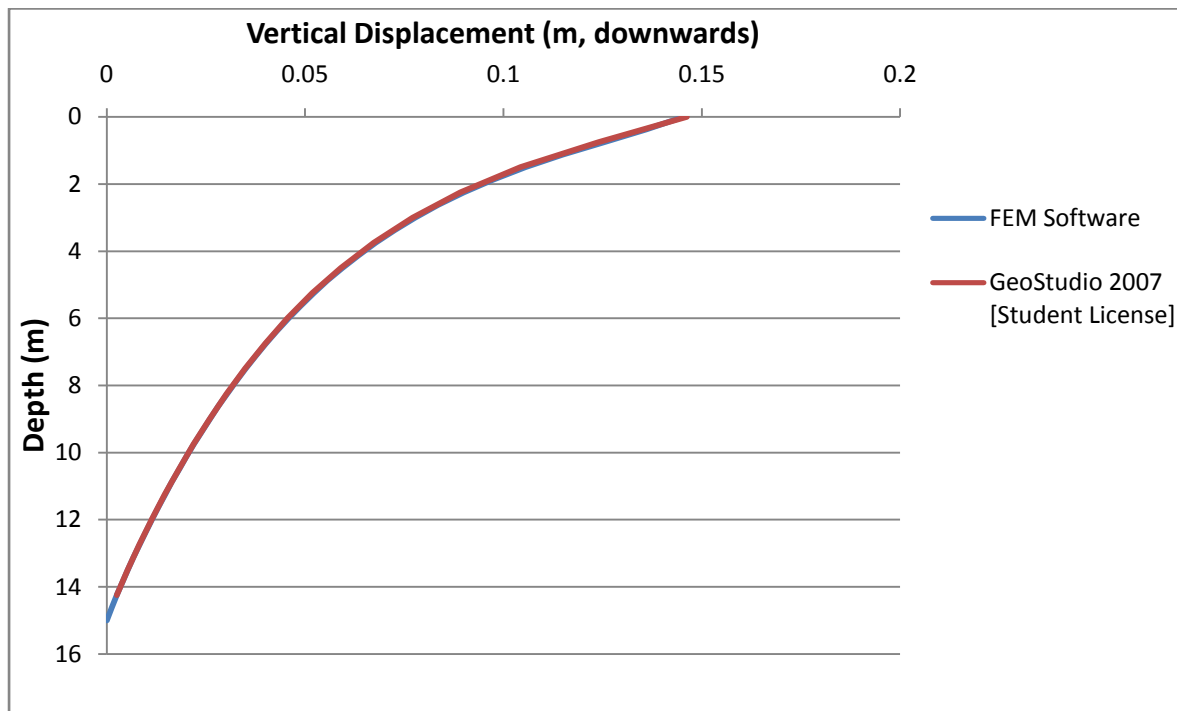
$$\sigma_x = \frac{P}{\pi} [\alpha - \sin \alpha \cdot \cos 2\beta']$$

$$\sigma_z = \frac{P}{\pi} [\alpha + \sin \alpha \cdot \cos 2\beta']$$

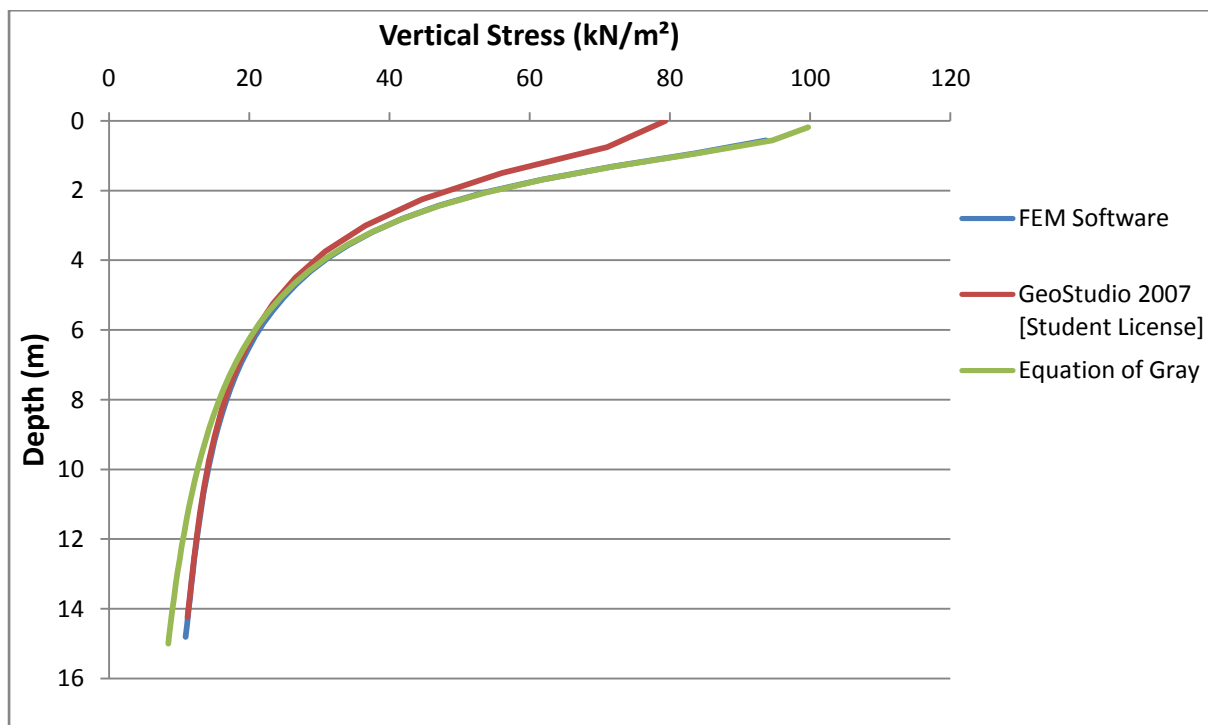
$$\sigma_y = \frac{2P}{\pi m} \alpha$$

$$\tau_{xz} = \frac{P}{\pi} \sin \alpha \cdot \sin 2\beta'$$

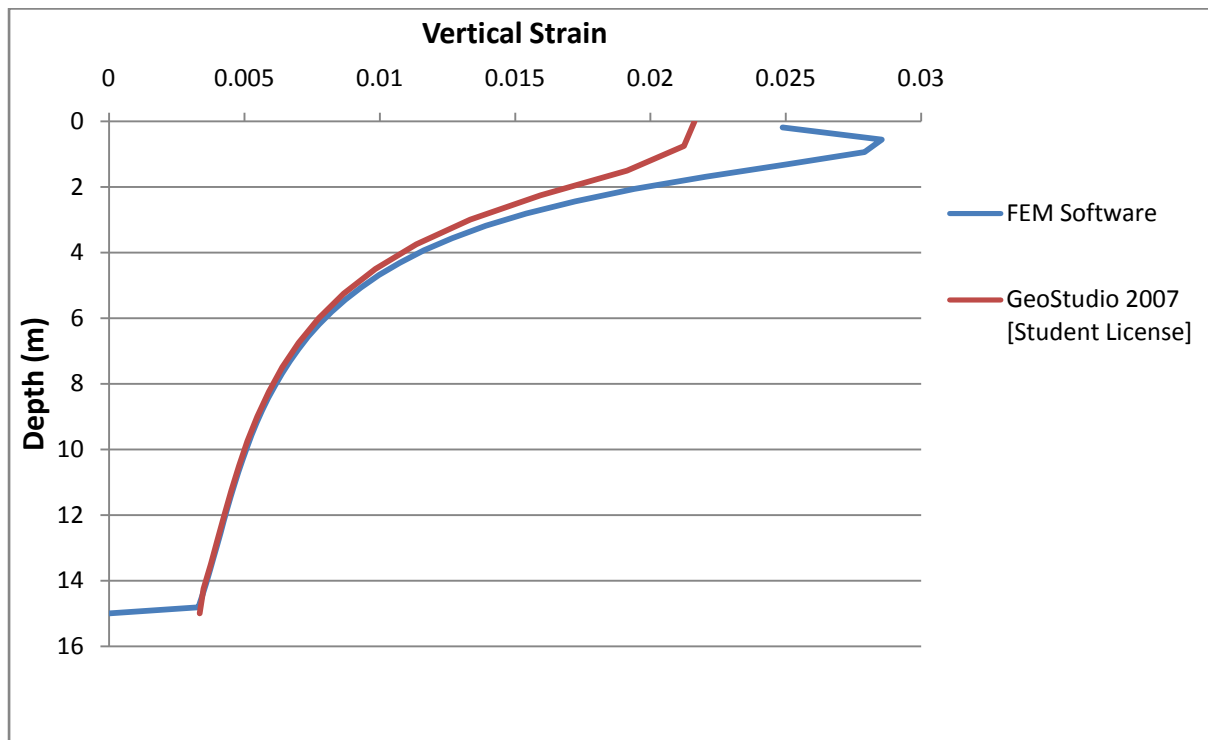
$m = \text{Poisson's number}$



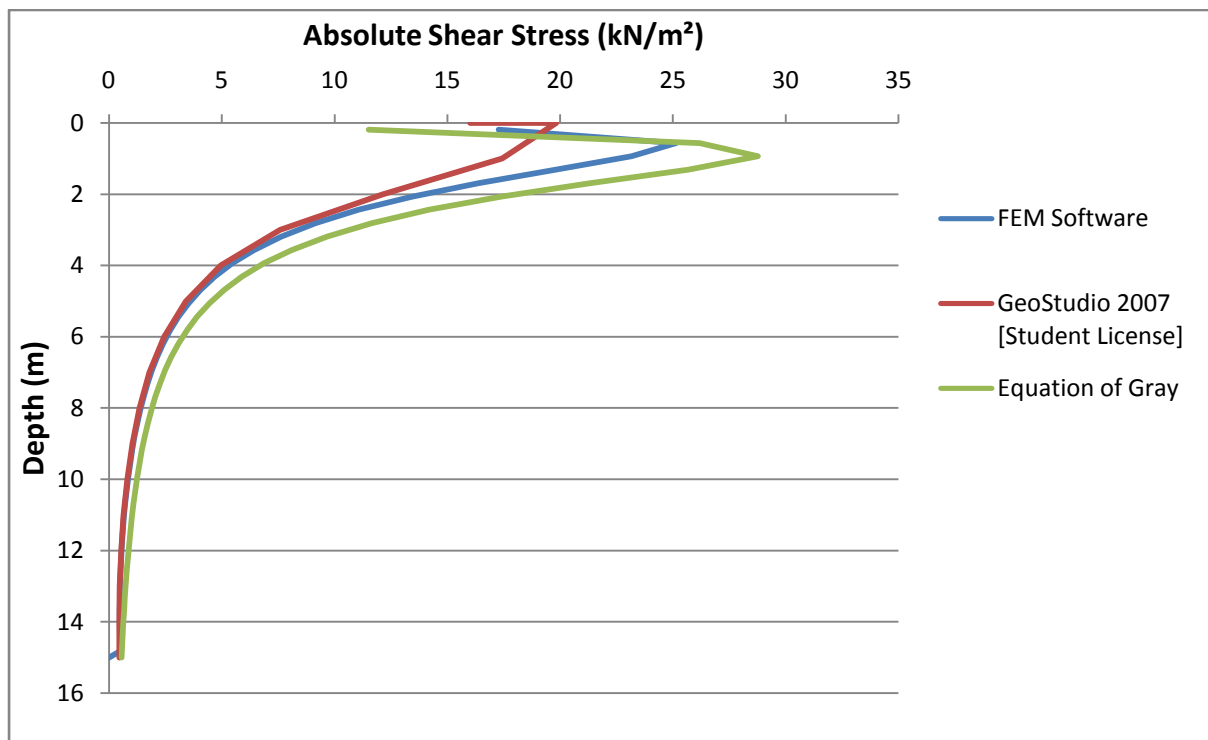
**Figure 4.12:** Distribution of Vertical Displacement along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Single Layered System



**Figure 4.13:** Distribution of Vertical Stress along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Single Layered System



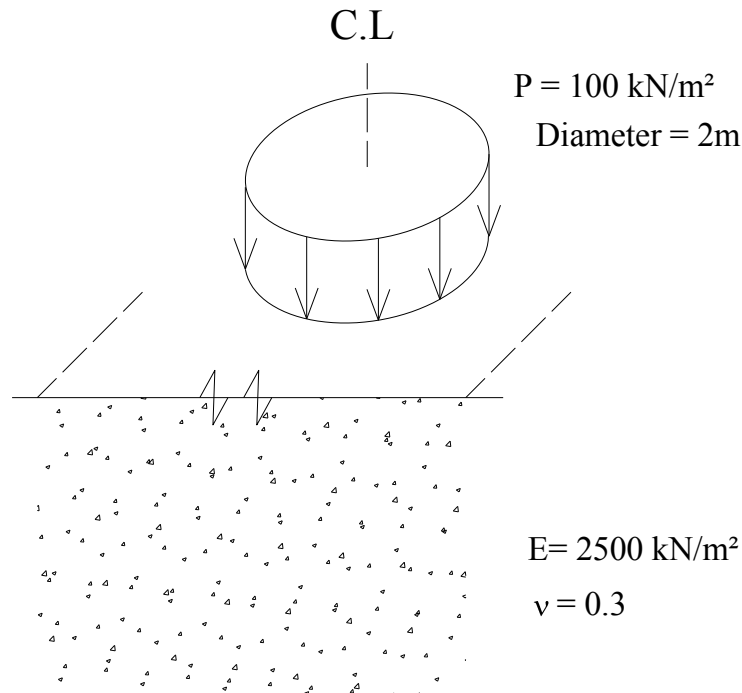
**Figure 4.14:** Distribution of Vertical Strain along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Single Layered System



**Figure 4.15:** Distribution of Shear Stress along the Edge of a Uniformly Distributed Strip Load Acting on the Surface of a Single Layered System

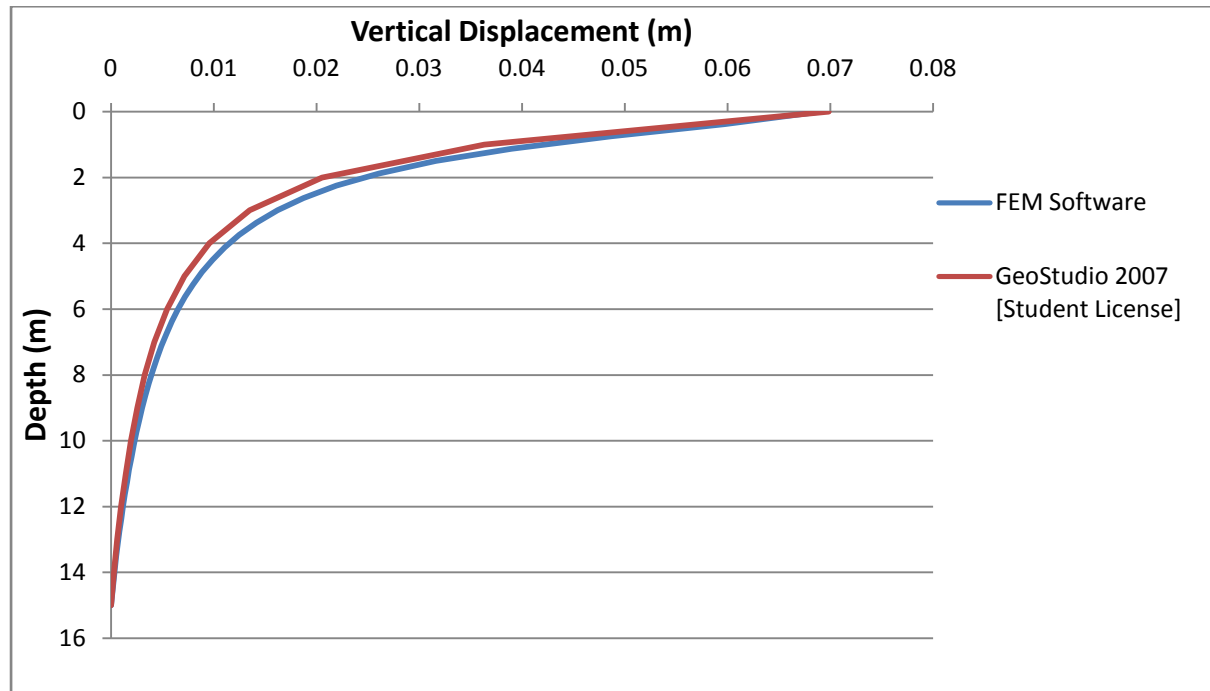
**4.2.1.1.4. Uniform Load of Circular Plan Area**

Sample problem 4: Uniform Load of Circular Plan Area Acting on the Surface of a Single Layer

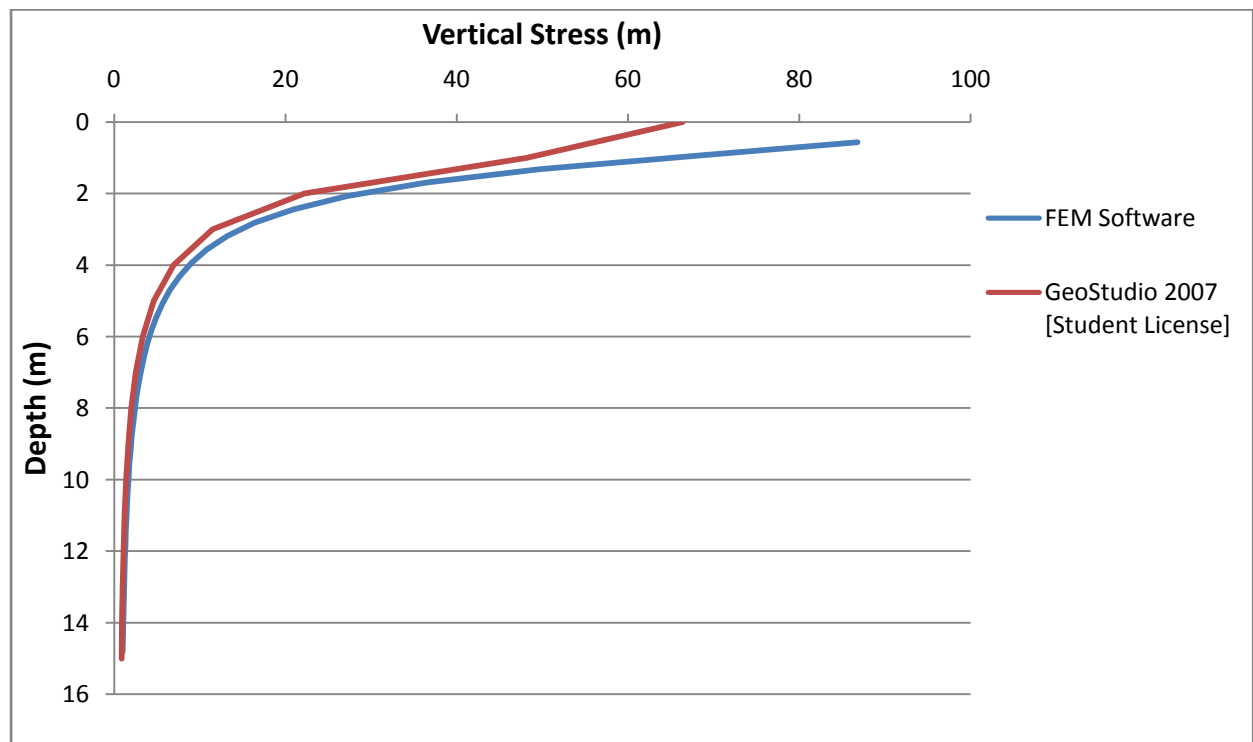


**Figure 4.16**

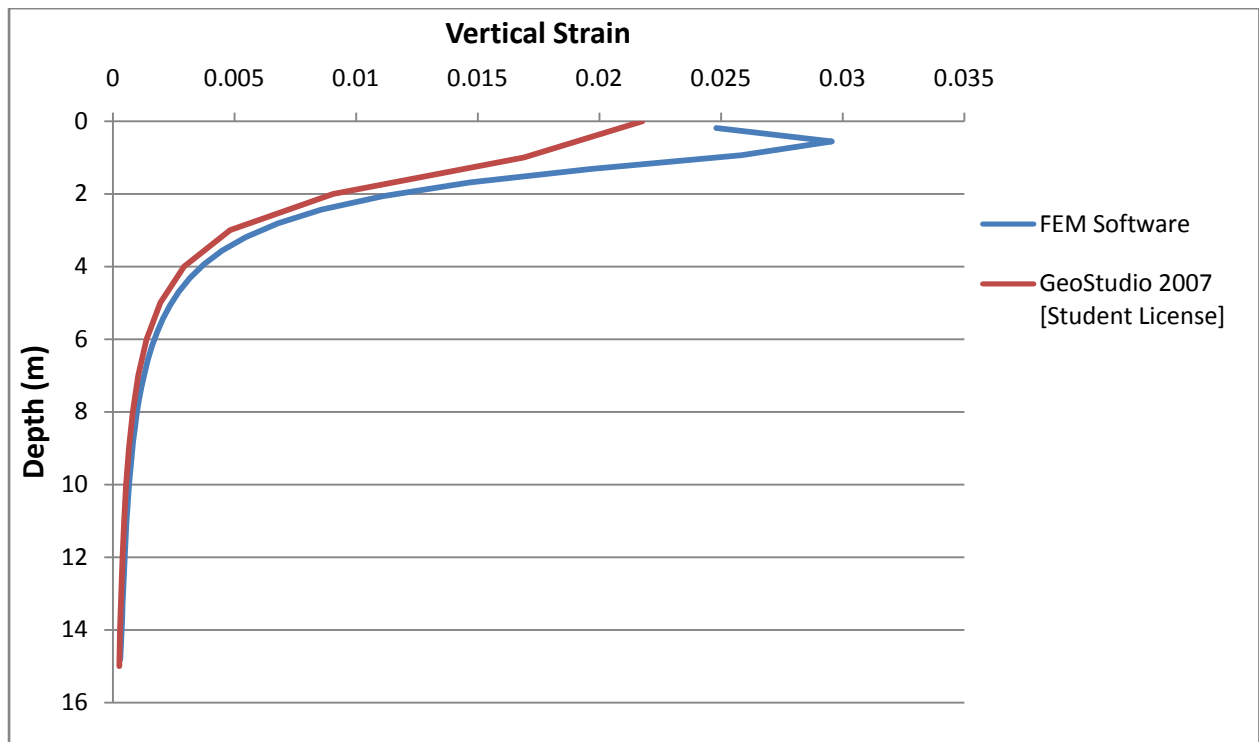
For the problem of a uniform load of circular plan area acting on the surface of a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of the circular loading.



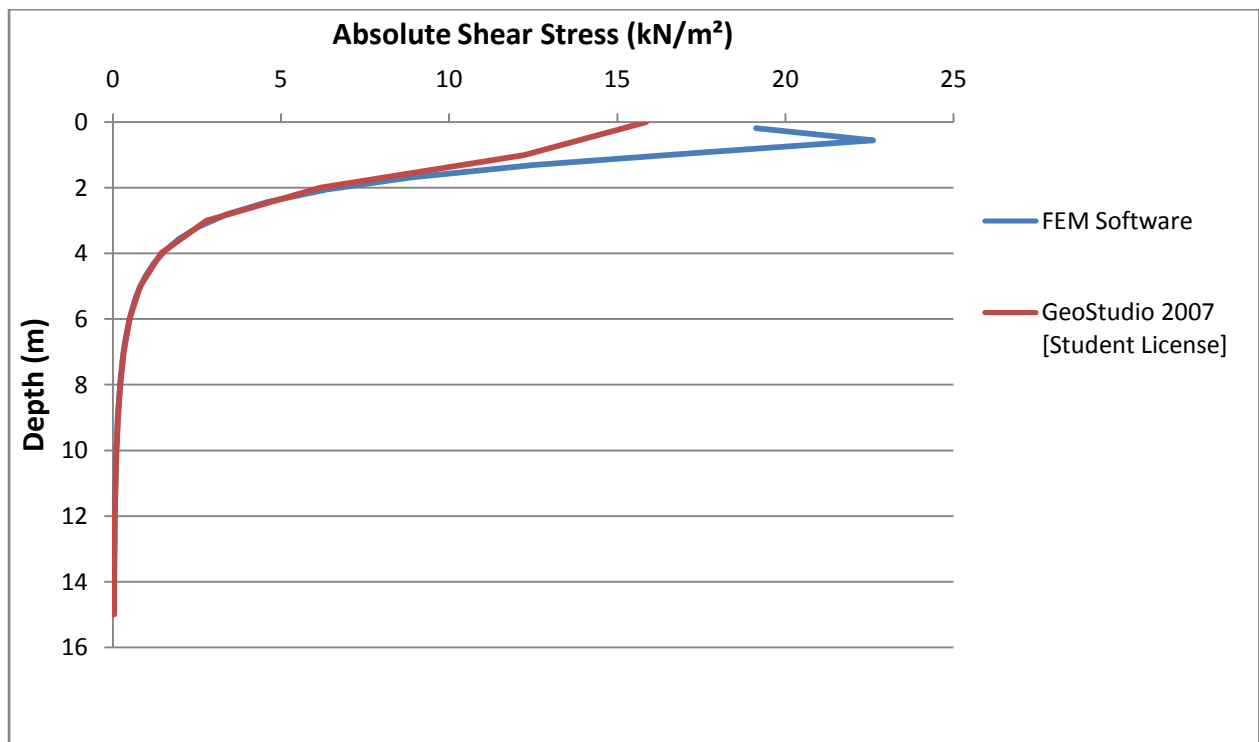
**Figure 4.17:** Distribution of Vertical Displacement along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Single Layered System



**Figure 4.18:** Distribution of Vertical Stress along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Single Layered System



**Figure 4.19:** Distribution of Vertical Strain along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Single Layered System



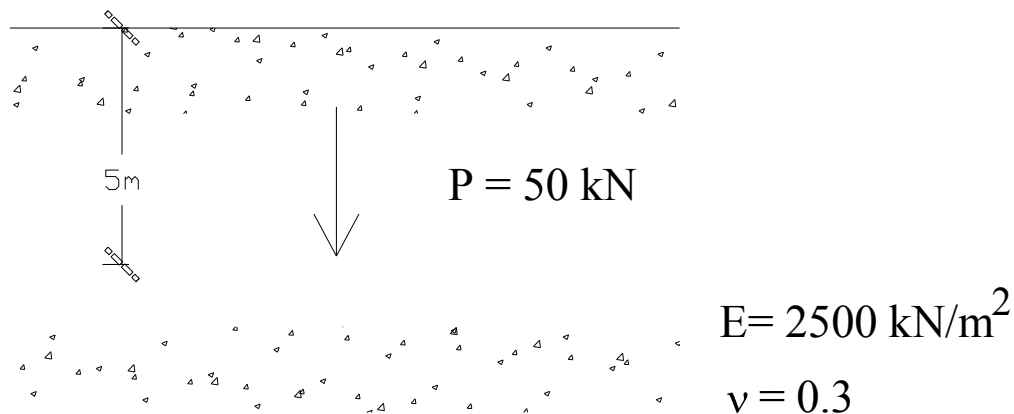
**Figure 4.20:** Distribution of Shear Stress along the Edge of a Uniform Load of Circular Plan Area Acting on the Surface of a Single Layered System



#### 4.2.1.2. Loads in Half-Space

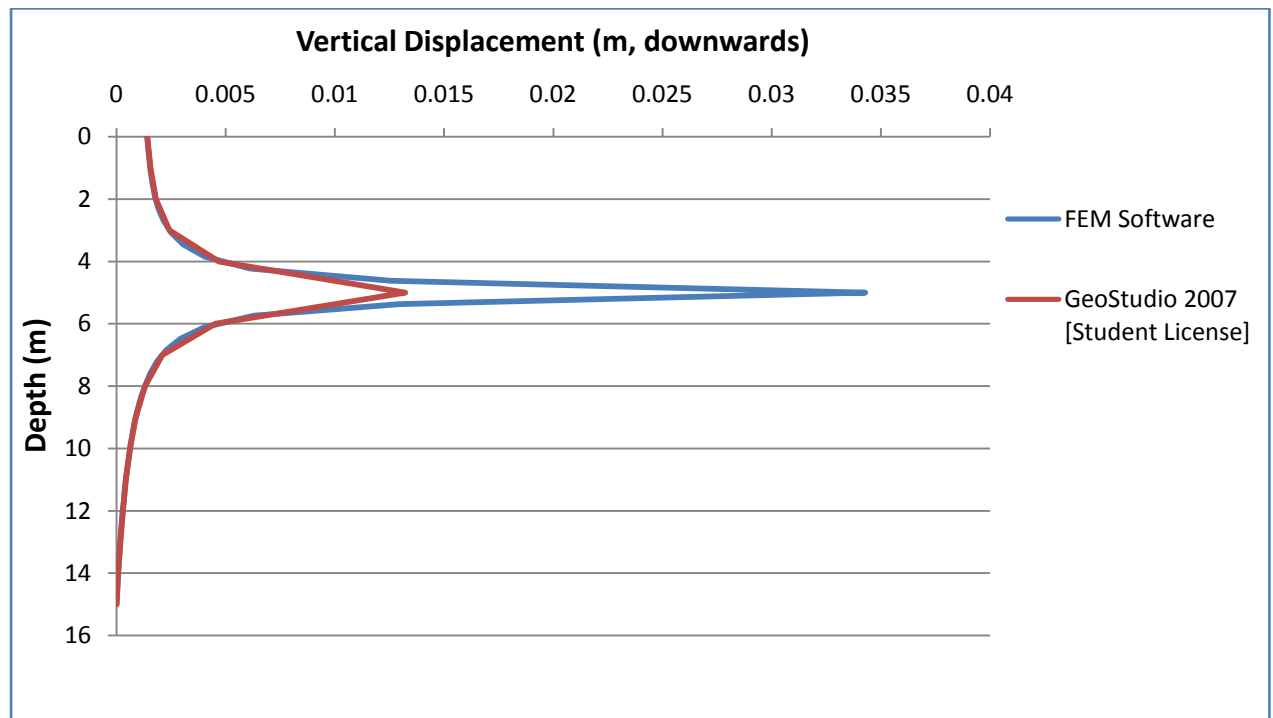
##### 4.2.1.2.1. Point Load

Sample problem 5: Point Load Acting within a Single Layer

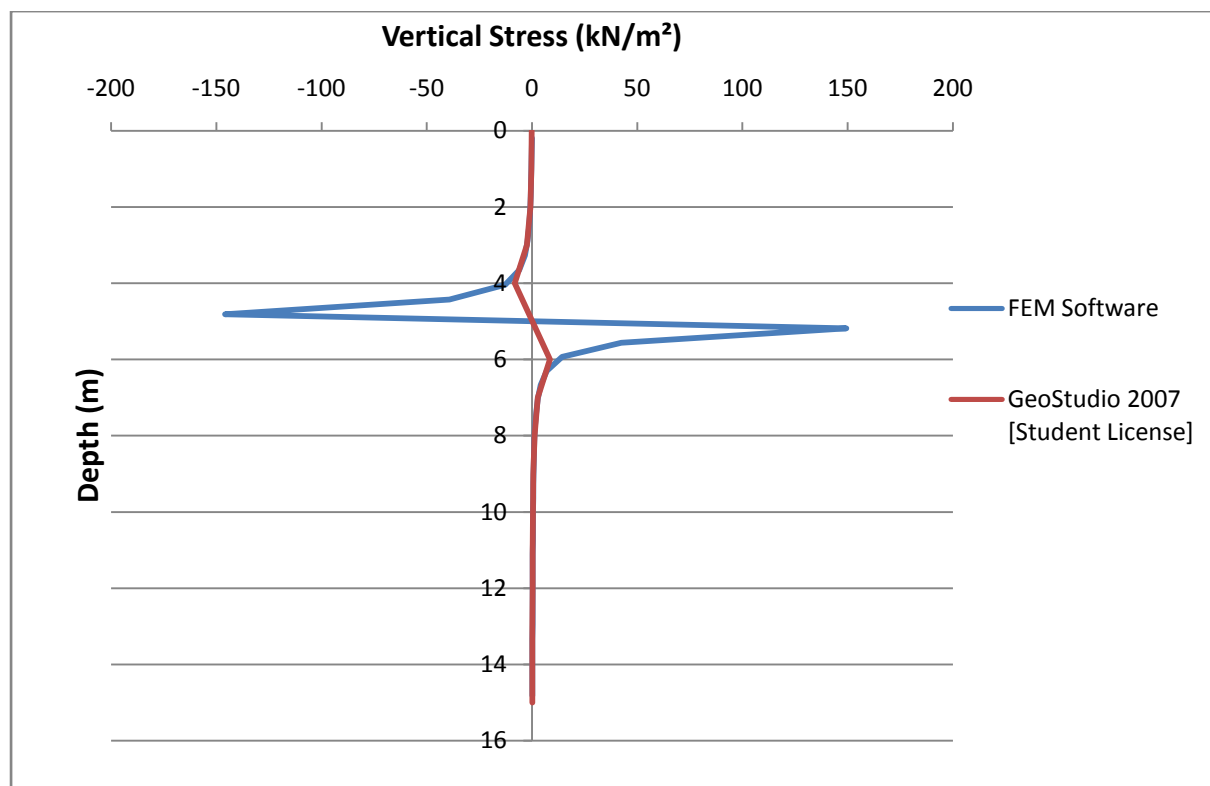


**Figure 4.21**

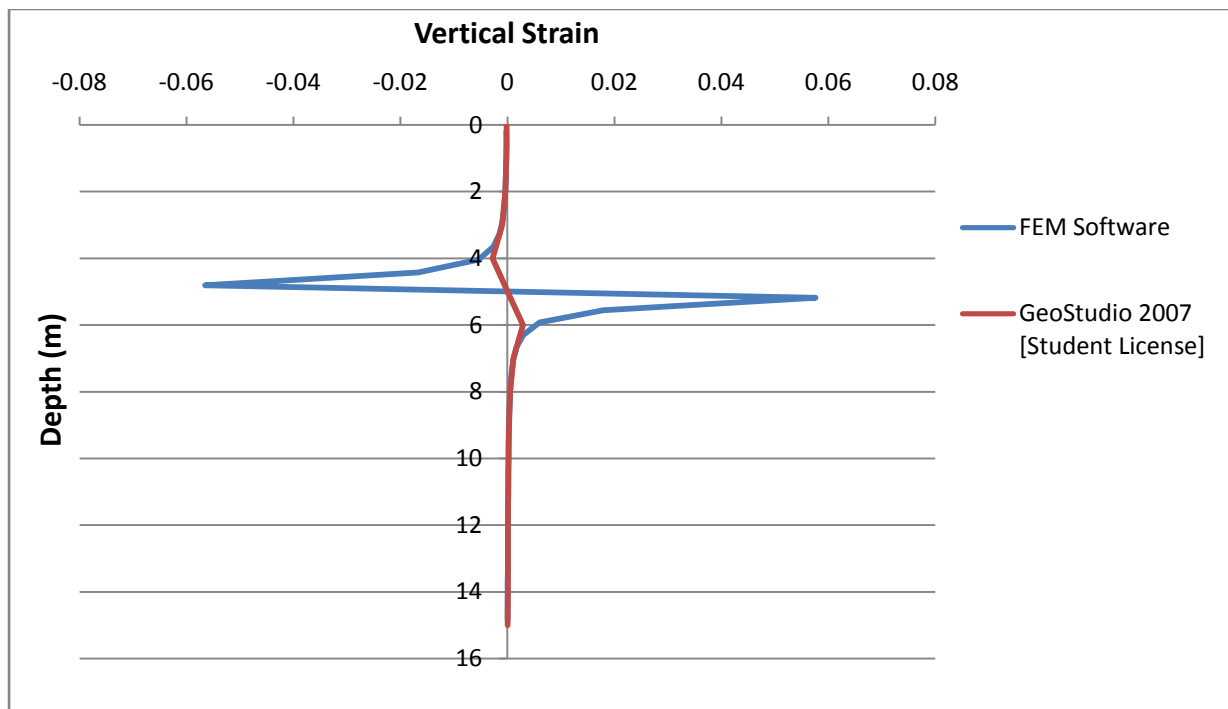
For the problem of a point load acting within a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of results is performed for a plane at a radial distance of 1m from the point load.



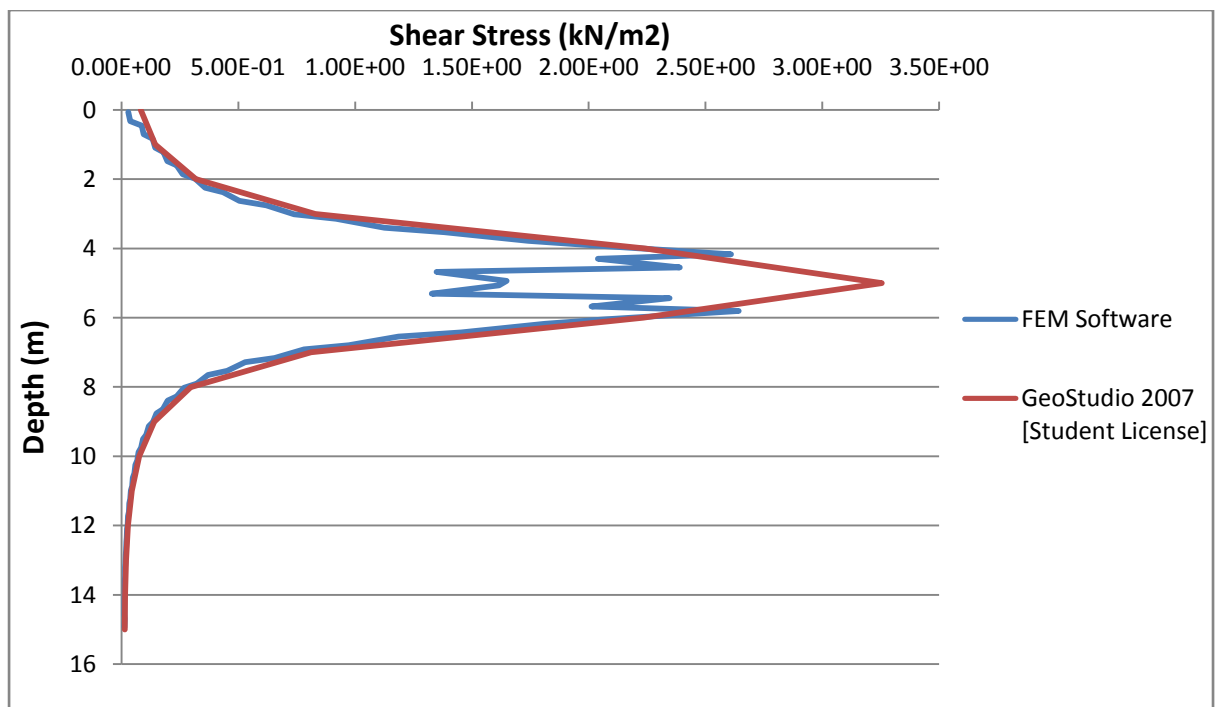
**Figure 4.22:** Distribution of Vertical Displacement along the Axis of a Point Load Acting within a Single Layered System



**Figure 4.23:** Distribution of Vertical Stress along the Axis of a Point Load Acting within a Single Layered System



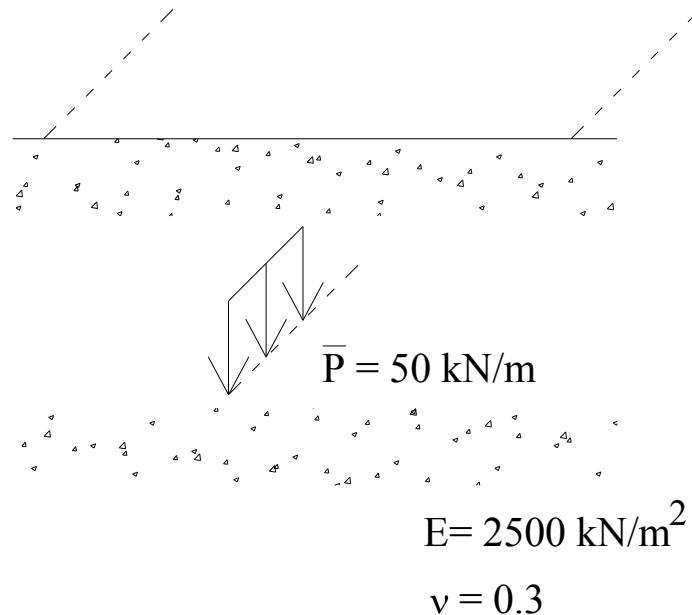
**Figure 4.24:** Distribution of Vertical Strain along the Axis of a Point Load Acting within a Single Layered System



**Figure 4.25:** Distribution of Shear Stress at a Plane of 1m Radial Distance from a Point Load Acting within a Single Layered System

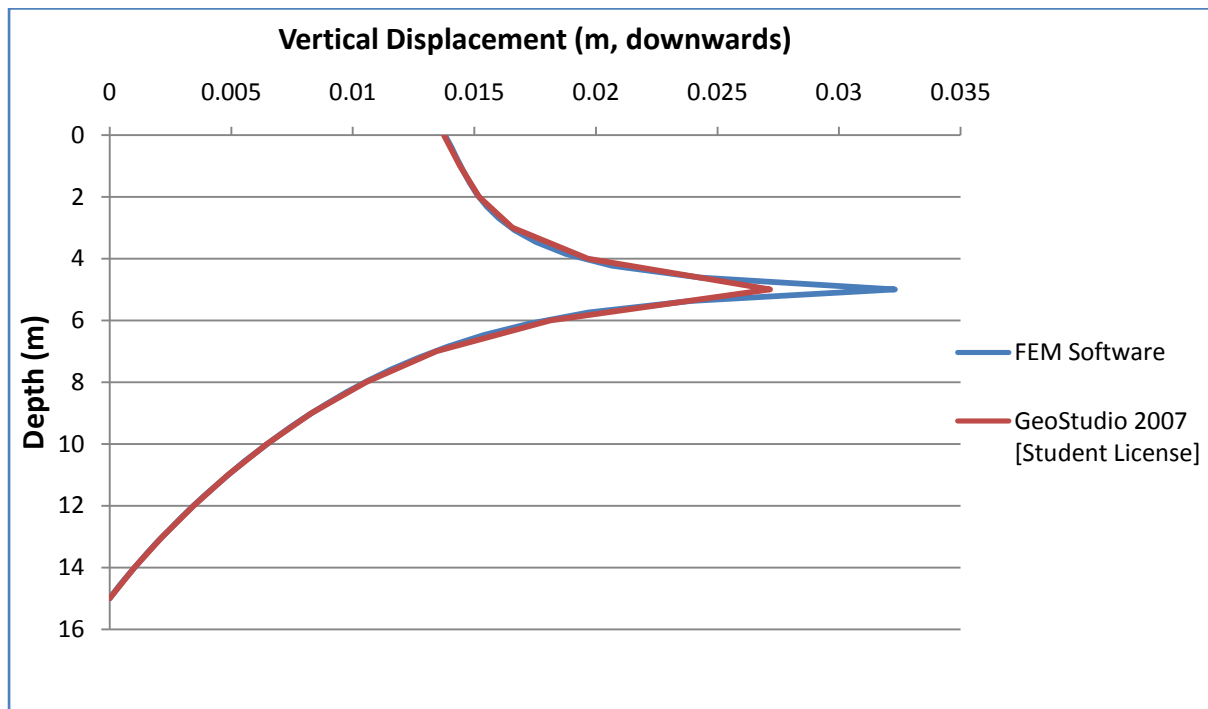
#### 4.2.1.2.2. Line Load

##### Sample problem 6: Line Load Acting within a Single Layer

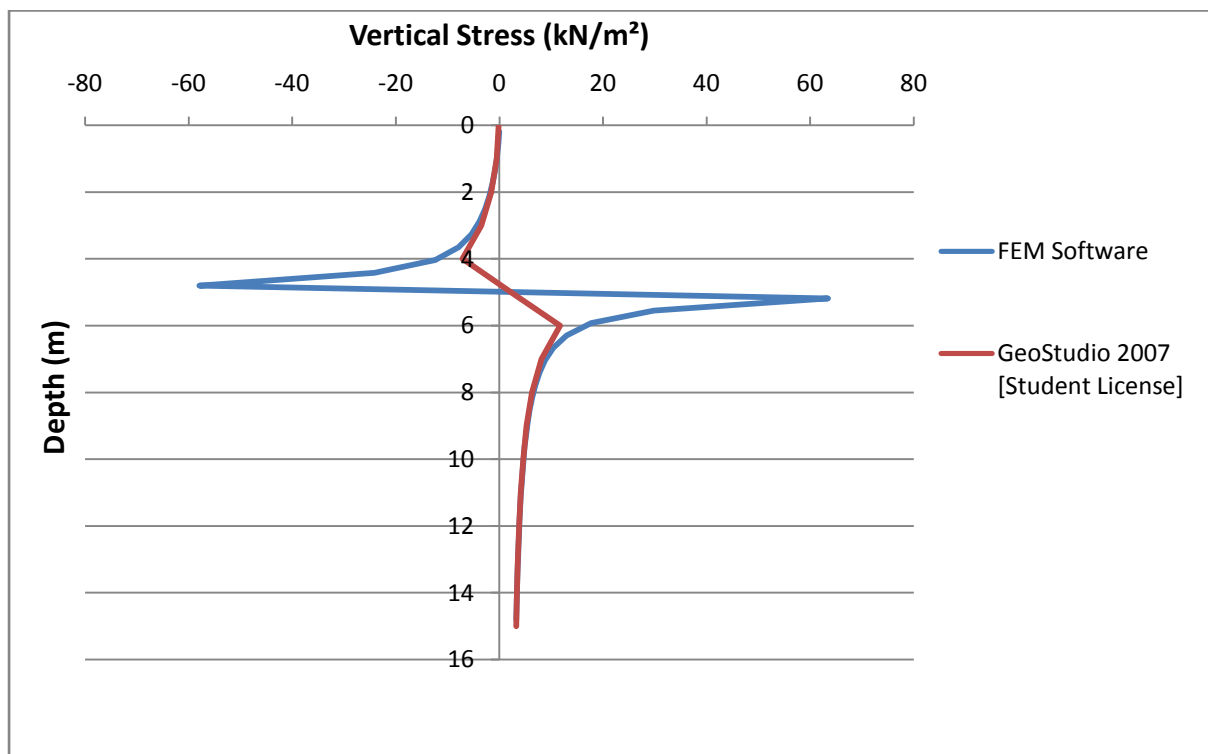


**Figure 4.26**

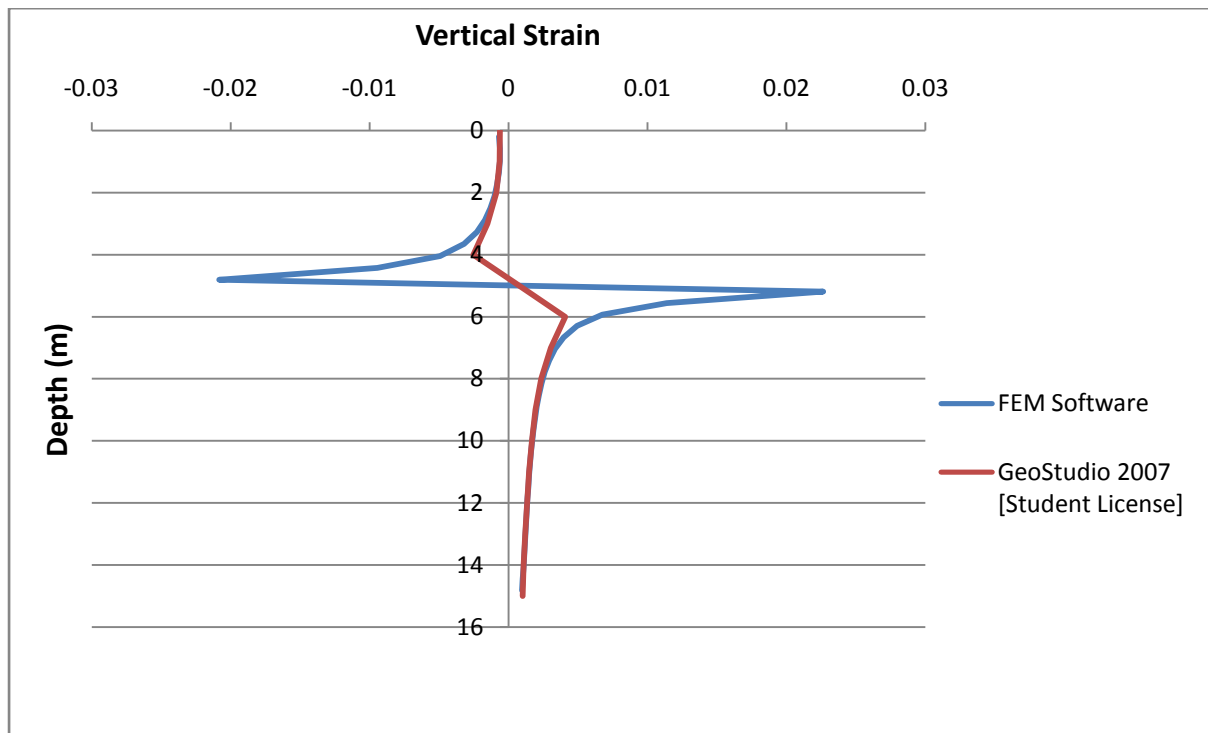
For the problem of a line load acting within a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis parallel to the plane of loading at a distance of 1m.



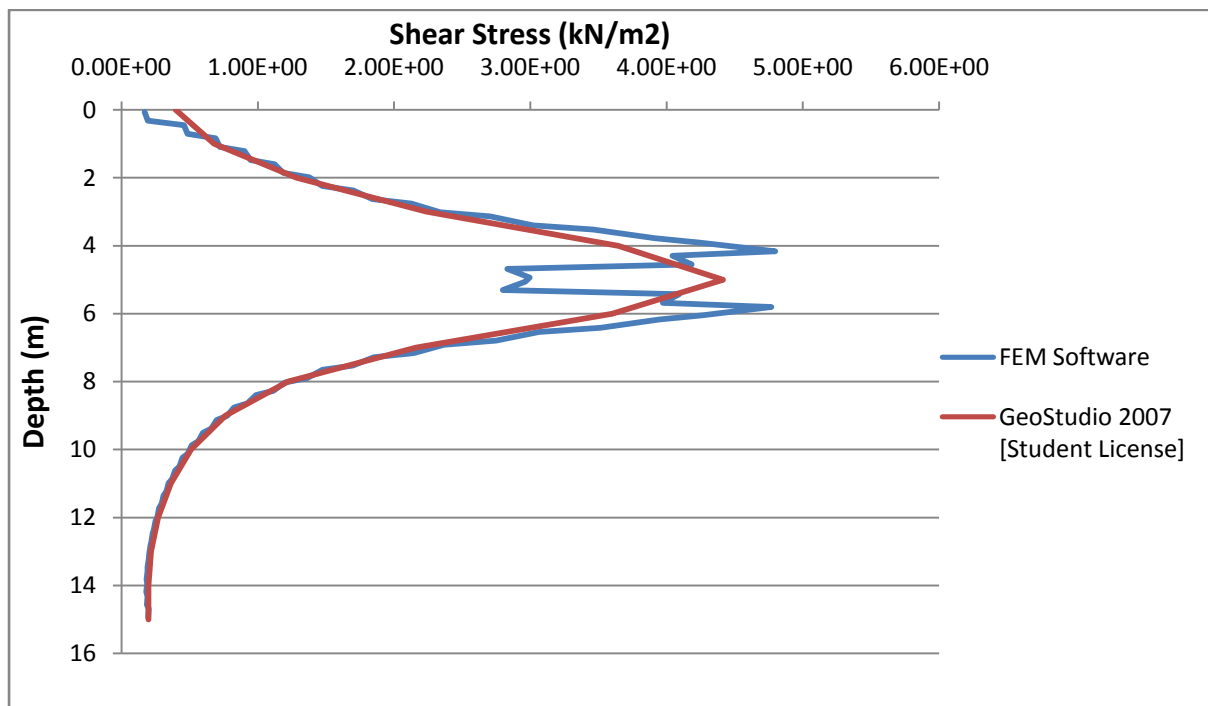
**Figure 4.27:** Distribution of Vertical Displacement along the Axis of a Line Load Acting within a Single Layered System



**Figure 4.28:** Distribution of Vertical Stress along the Axis of a Line Load Acting within a Single Layered System



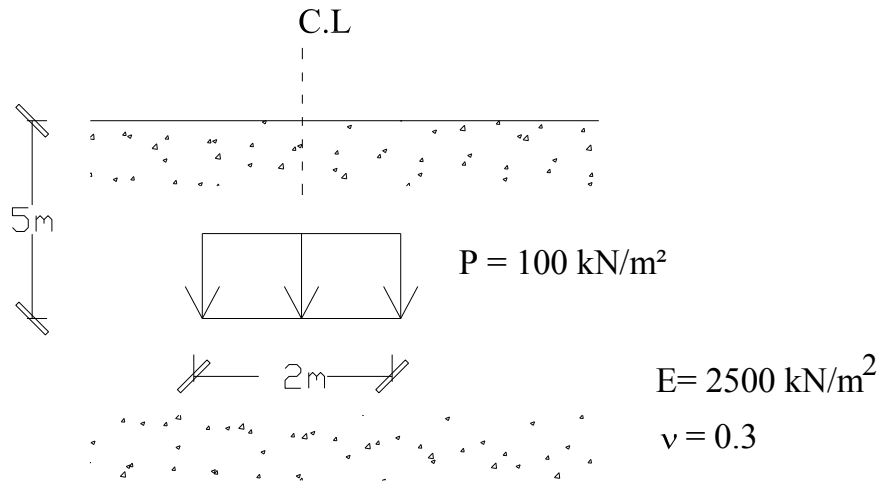
**Figure 4.29:** Distribution of Vertical Strain along the Axis of a Line Load Acting within a Single Layered System



**Figure 4.30:** Distribution of Shear Stress at an Axis 1m Offset from a Line Load Acting within a Single Layered System

#### 4.2.1.2.3. Uniform Strip Load

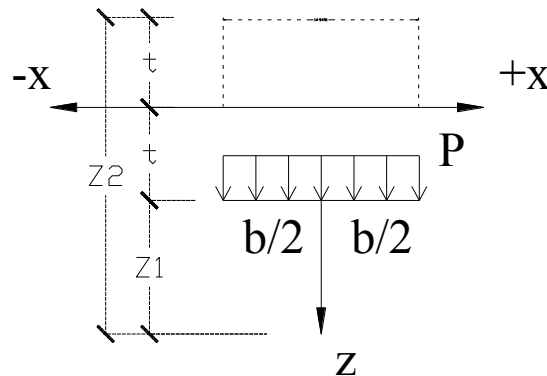
Sample problem 7: Strip Load Acting within a Single Layer



**Figure 4.31**

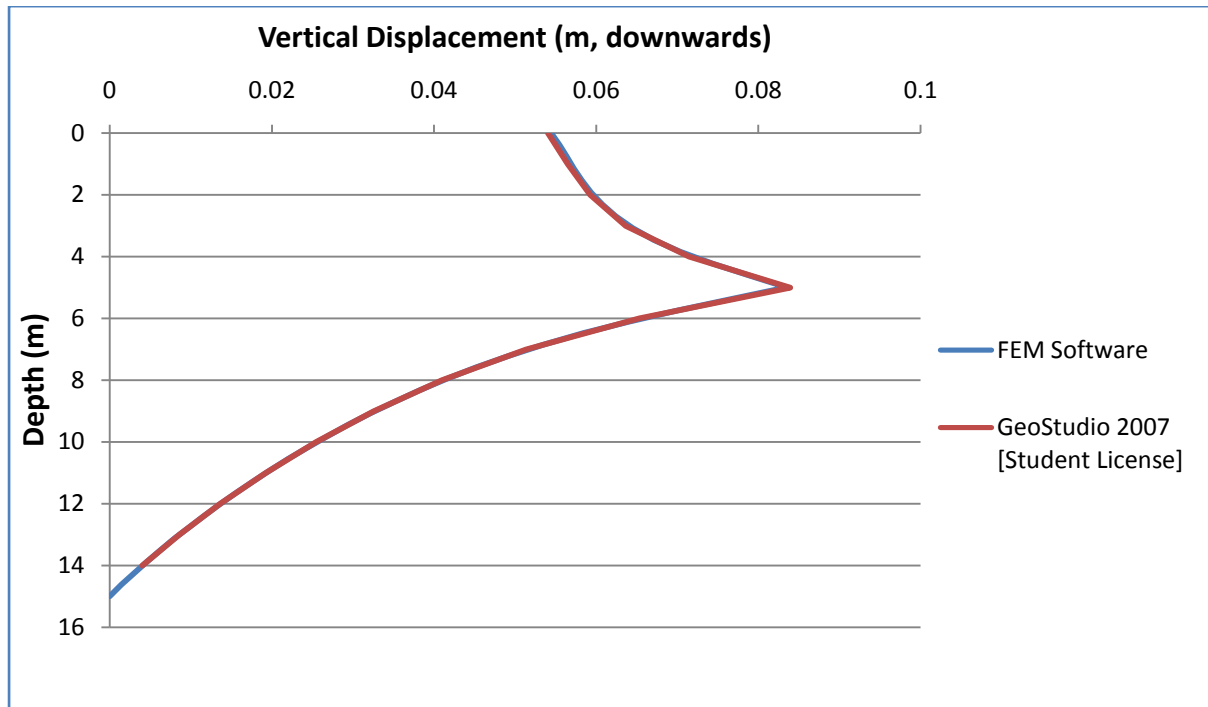
For the problem of a uniformly distributed strip load acting within a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Furthermore, the output results for vertical and shear stresses are compared with available equations. Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of strip loading.

Equations of Kezdi for a strip load acting on the surface of an elastic half space;

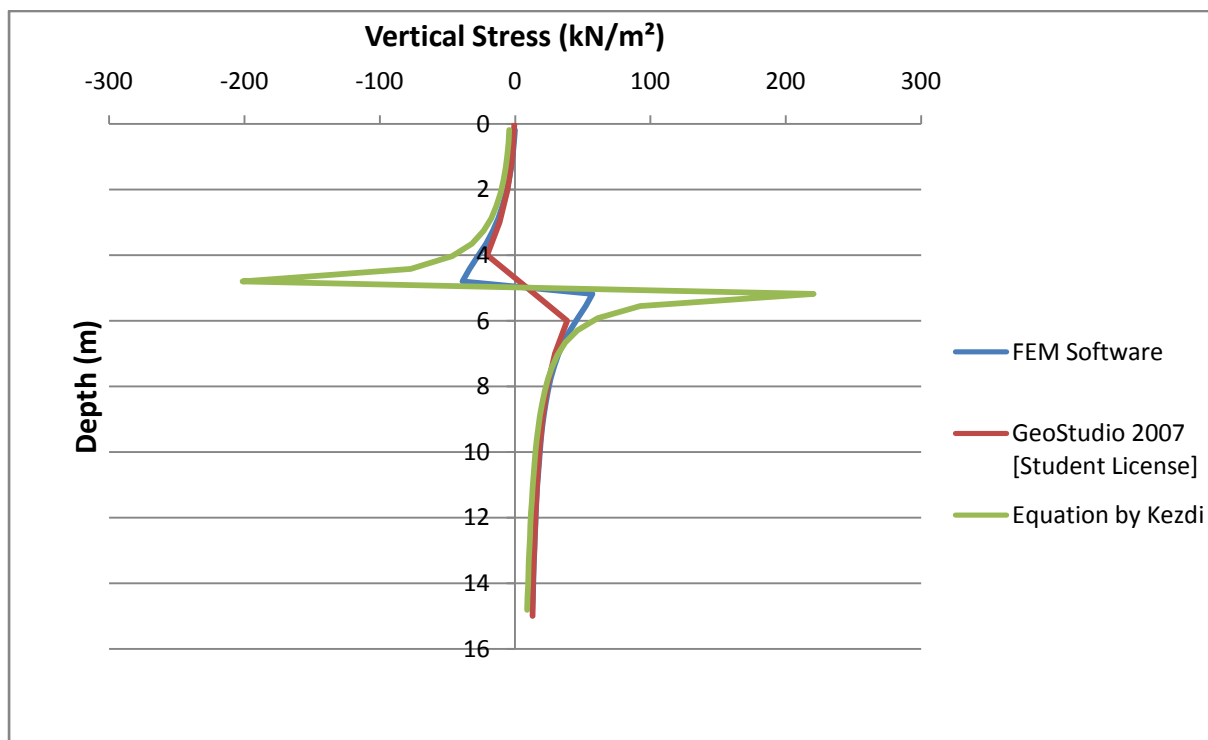


$$\sigma_z = \frac{P}{\pi} \left[ \frac{(bZ_1/2)}{z_1^2 + b^2/4} + \arctan \frac{b}{2Z_1} + \frac{(bZ_2/2)}{z_2^2 + b^2/4} + \arctan \frac{b}{2Z_2} - \frac{1-\mu}{2} * Z_1 \left\{ \frac{(b/2)}{z_1^2 + \frac{b^2}{4}} \right. \right. \\ \left. \left. - \frac{(b/2)}{z_2^2 + b^2/4} \right\} + \frac{1+\mu}{2} * \frac{Z_1 t}{Z_2} * \frac{(b/2)}{z_2^2 + b^2/4} \right]$$

$\mu = \text{Poisson's ratio}$

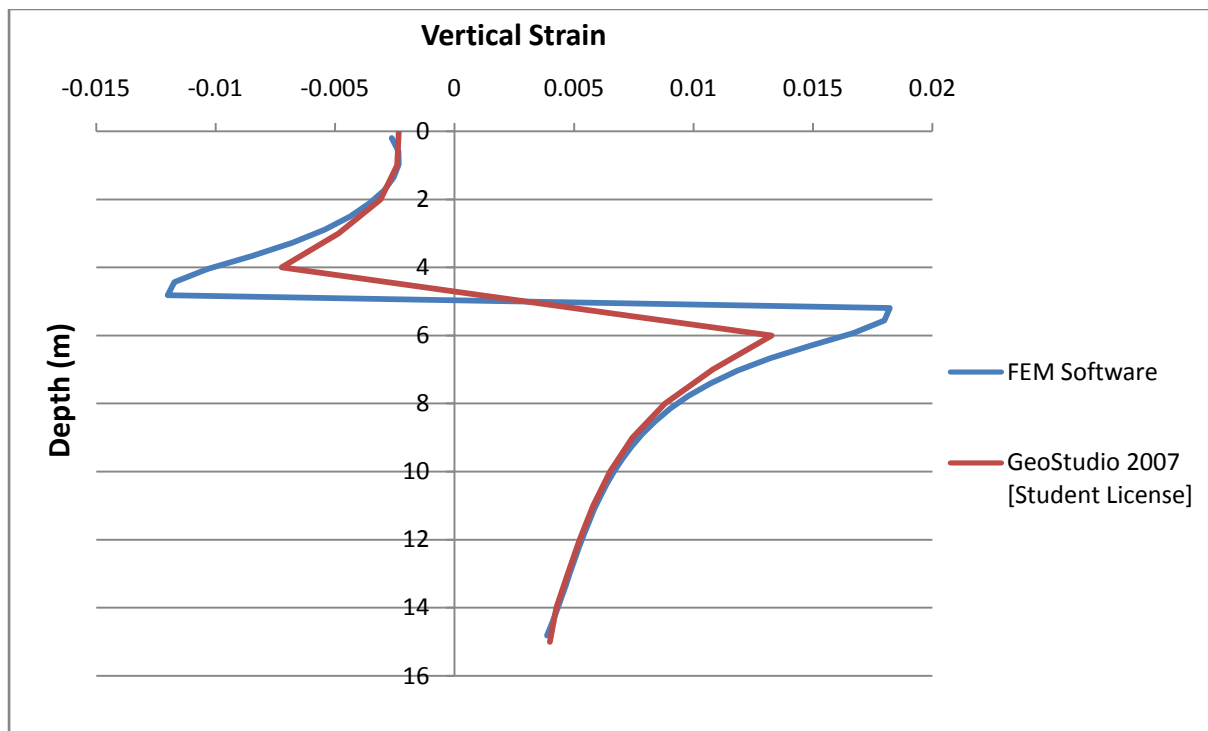


**Figure 4.32:** Distribution of Vertical Displacement along the Centerline of a Uniformly Distributed Strip Load Acting within a Single Layered System

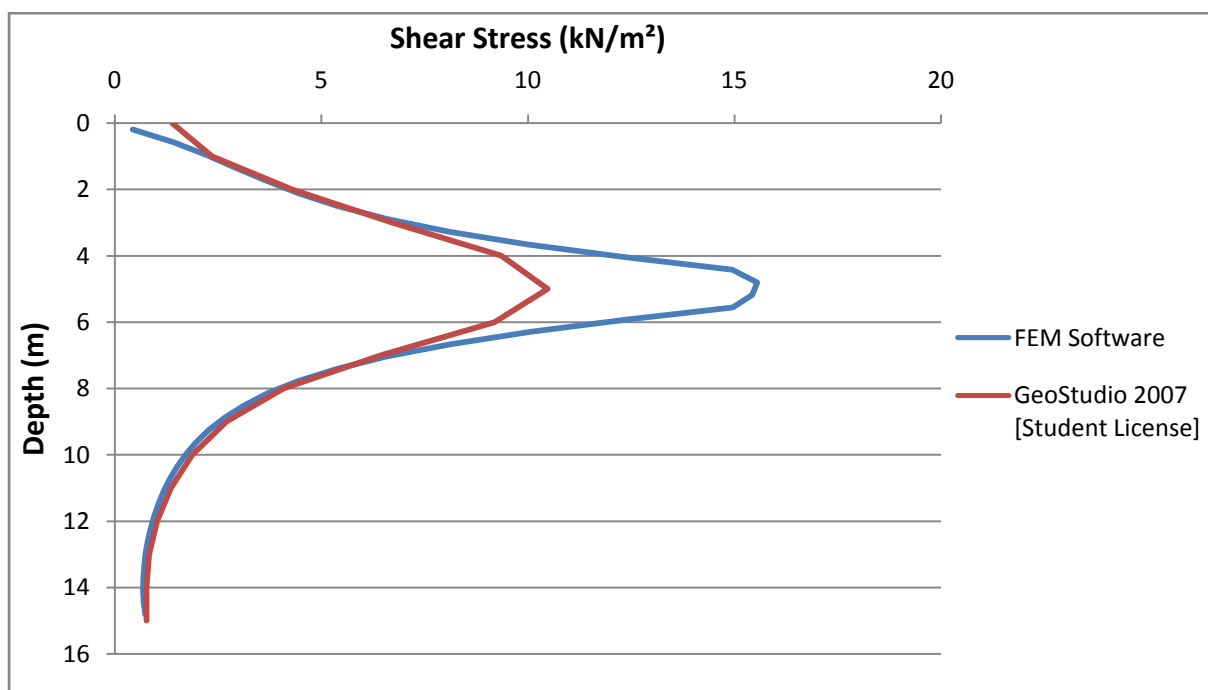


**Figure 4.33:** Distribution of Vertical Stress along the Centerline of a Uniformly Distributed Strip Load Acting within a Single Layered System





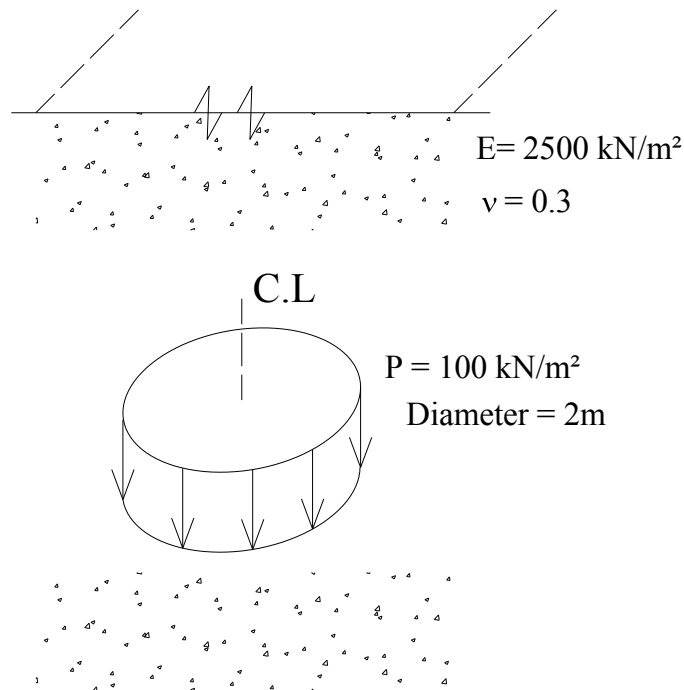
**Figure 4.34:** Distribution of Vertical Strain along the Centerline of a Uniformly Distributed Strip Load Acting within a Single Layered System



**Figure 4.35:** Distribution of Shear Stress along the Edge of a Uniformly Distributed Strip Load Acting within a Single Layered System

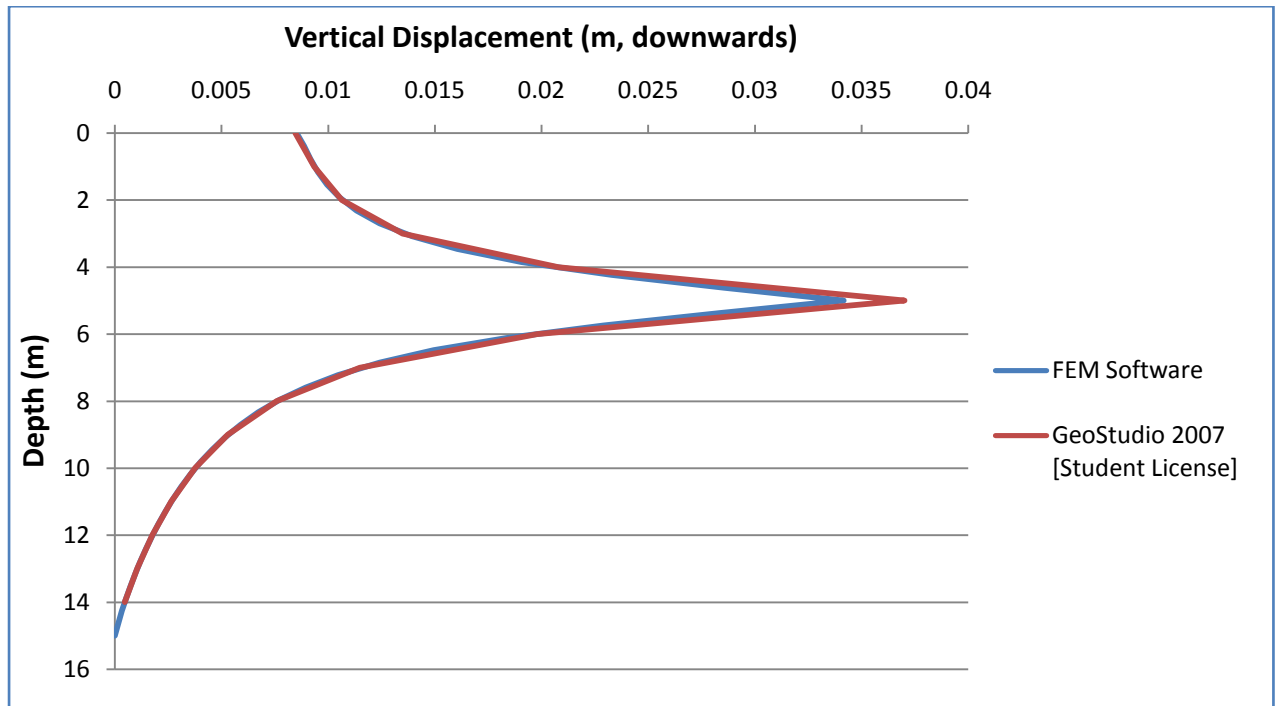
#### 4.2.1.2.4. Uniform Load of Circular Plan Area

Sample problem 8: Uniform Load of Circular Plan Area Acting within a Single Layer

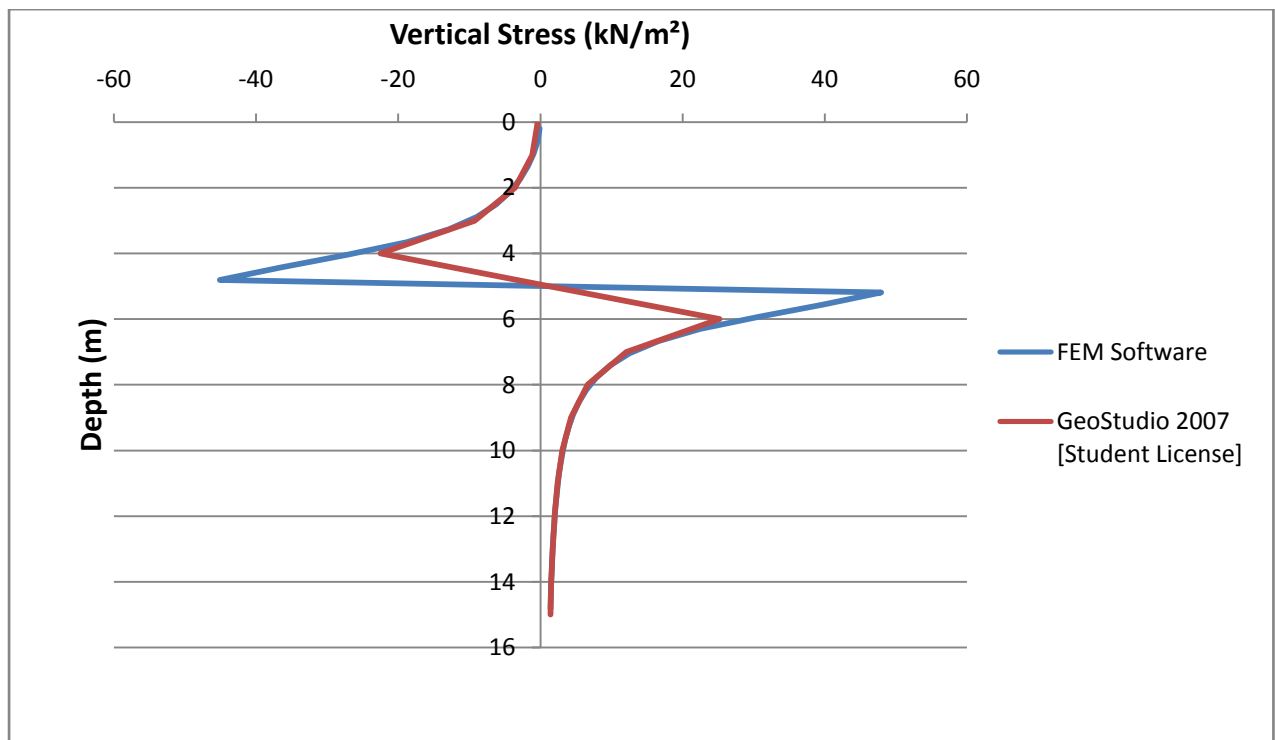


**Figure 4.36**

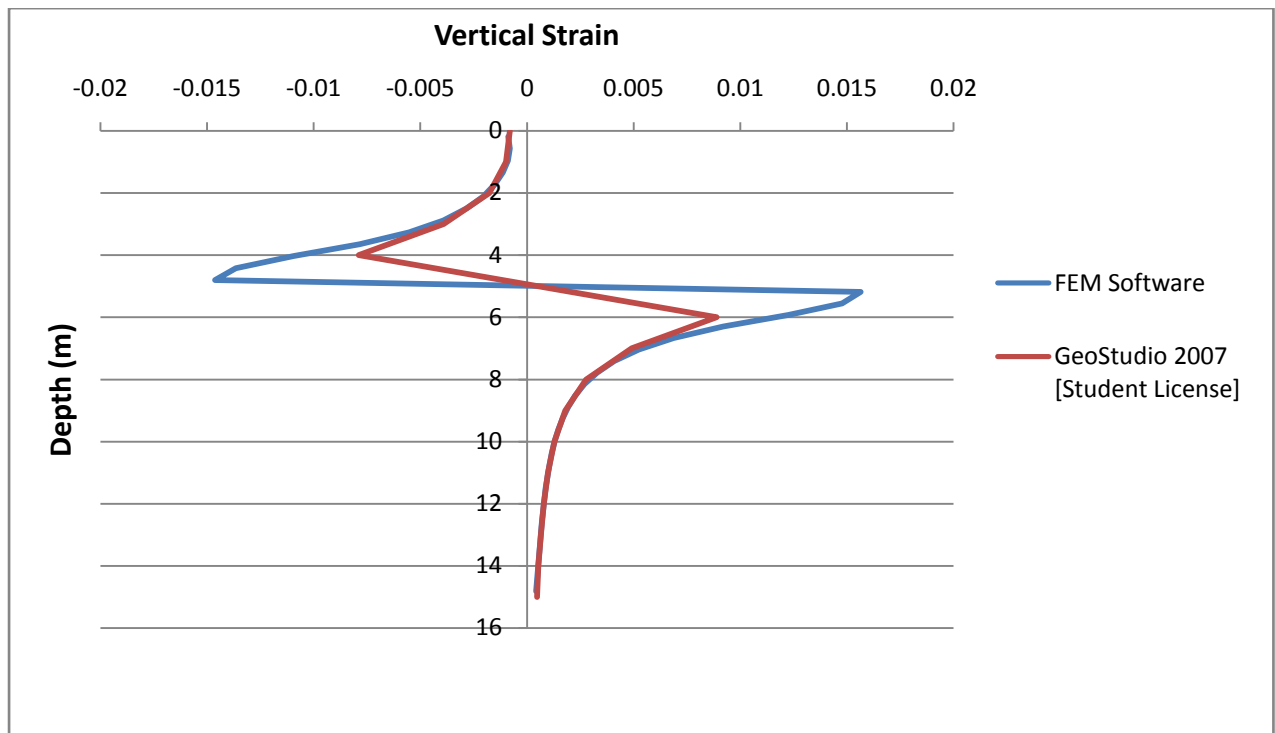
For the problem of a uniform load of circular plan area acting within a single layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of the circular loading.



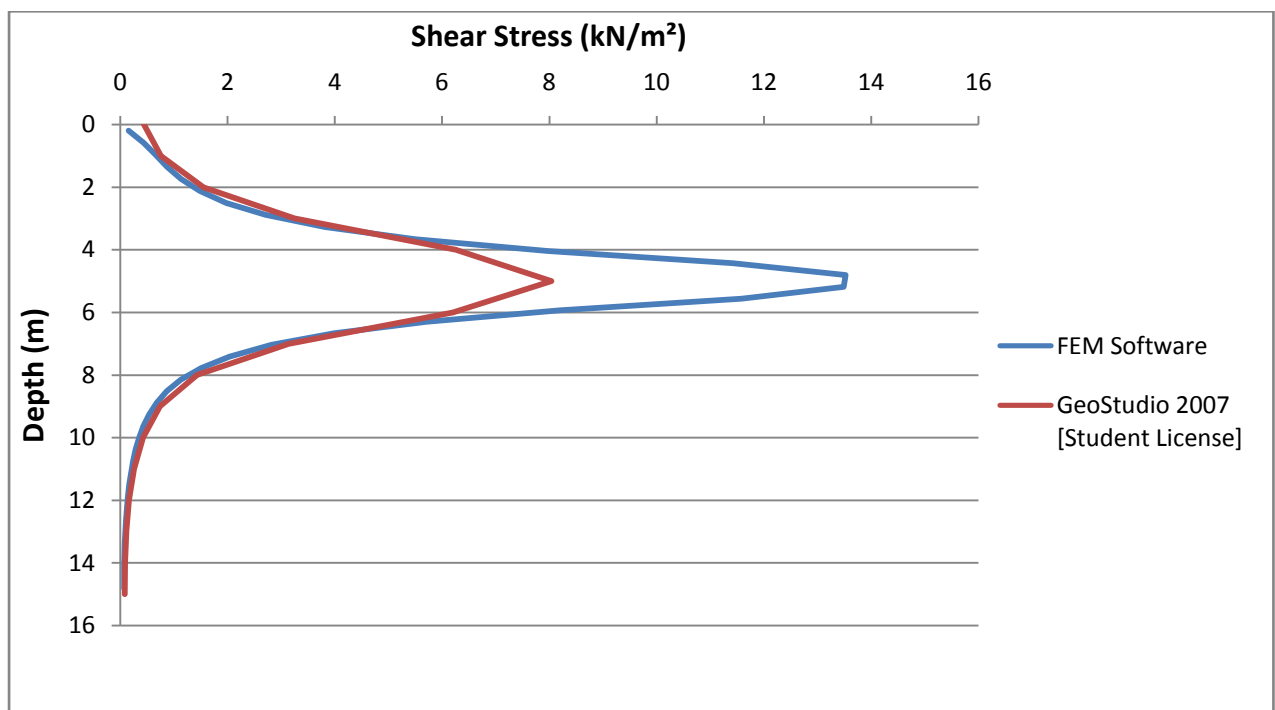
**Figure 4.37:** Distribution of Vertical Displacement along the Centerline of a Uniform Load of Circular Plan Area Acting within a Single Layered System



**Figure 4.38:** Distribution of Vertical Stress along the Centerline of a Uniform Load of Circular Plan Area Acting within a Single Layered System



**Figure 4.39:** Distribution of Vertical Strain along the Centerline of a Uniform Load of Circular Plan Area Acting within a Single Layered System

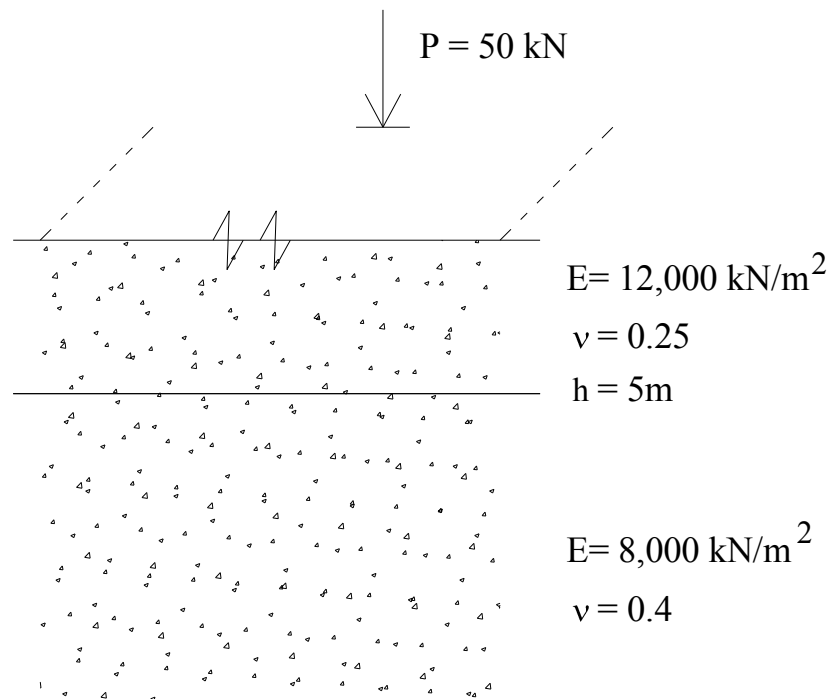


**Figure 4.40:** Distribution of Shear Stress along the Edge of a Uniform Load of Circular Plan Area Acting within a Single Layered System

## 4.2.2. Two Layers- Surface Loads

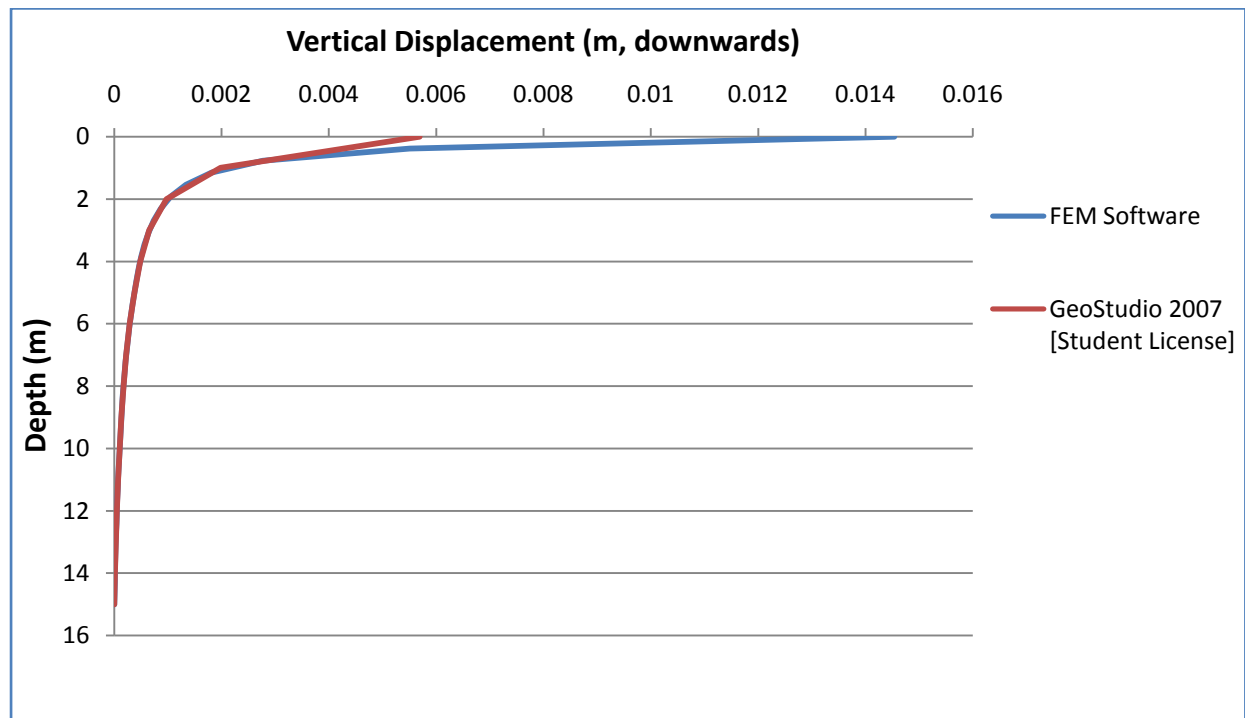
### 4.2.2.1.1. Point Load

Sample problem 9: Point Load Acting on the Surface of a Double Layered System

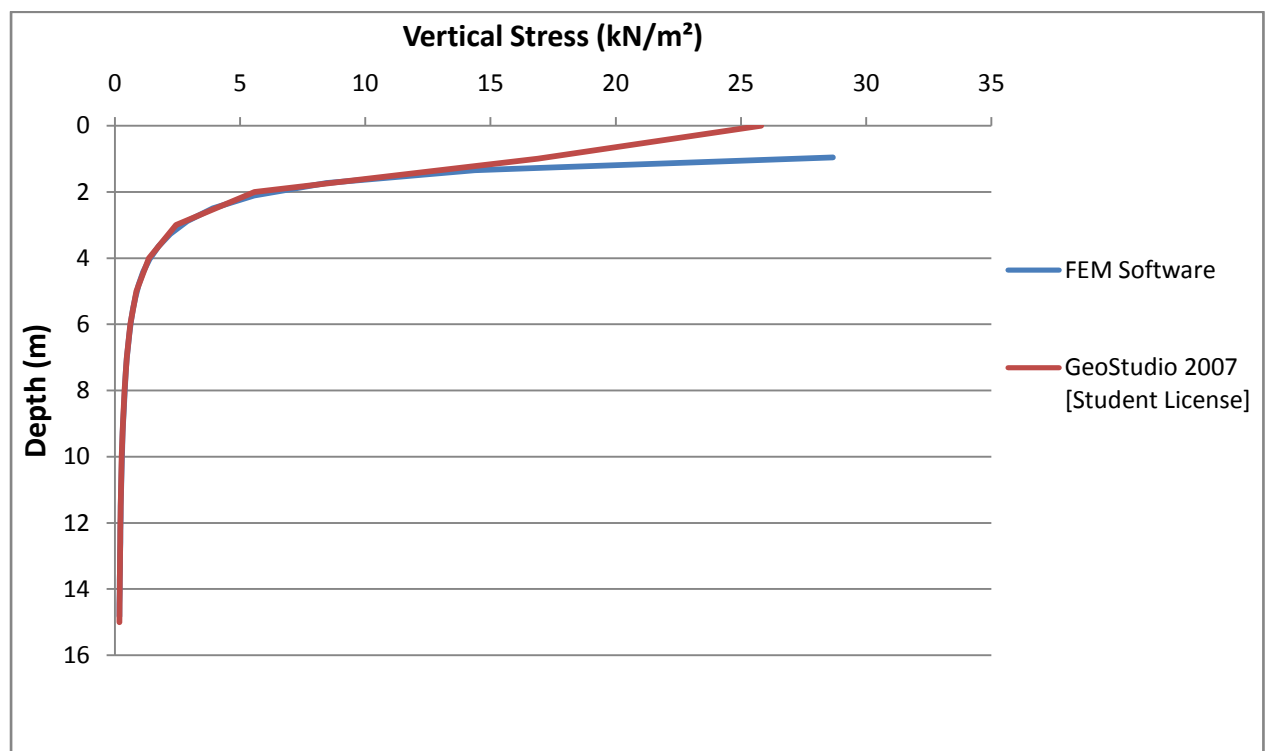


**Figure 4.41**

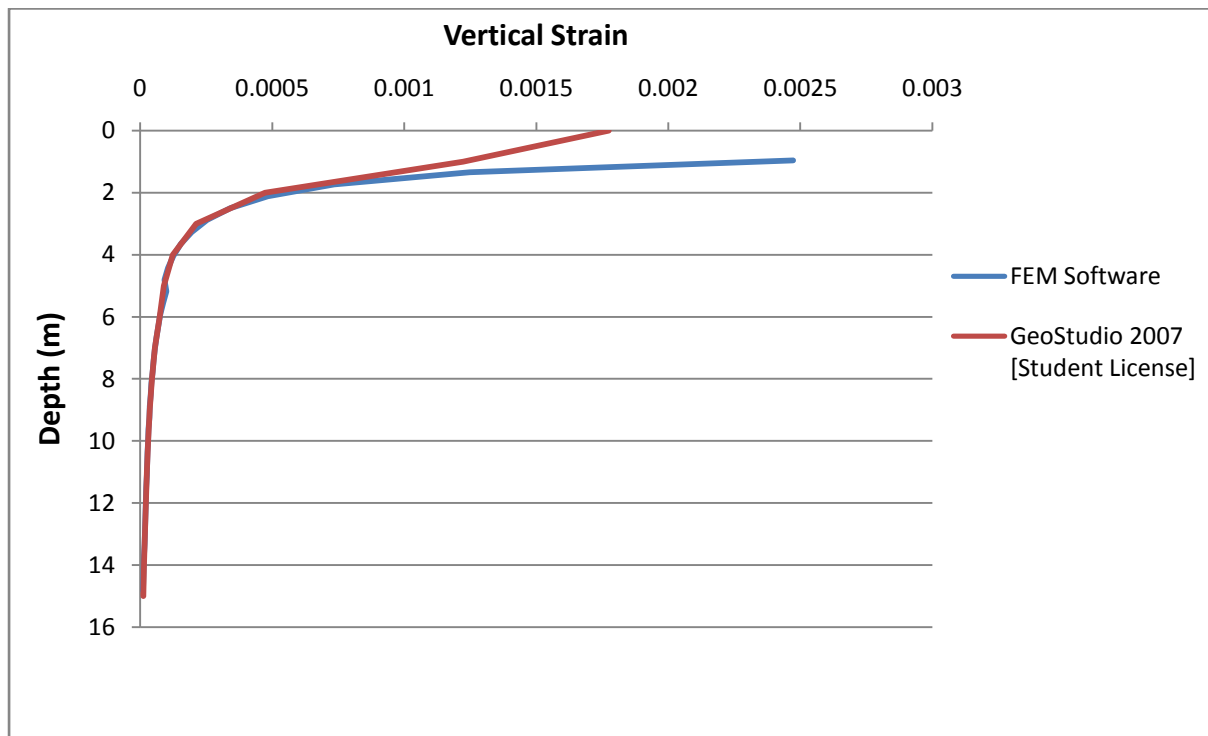
For the problem of a point load acting on the surface of a two layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for a plane at a radial distance of 1m.



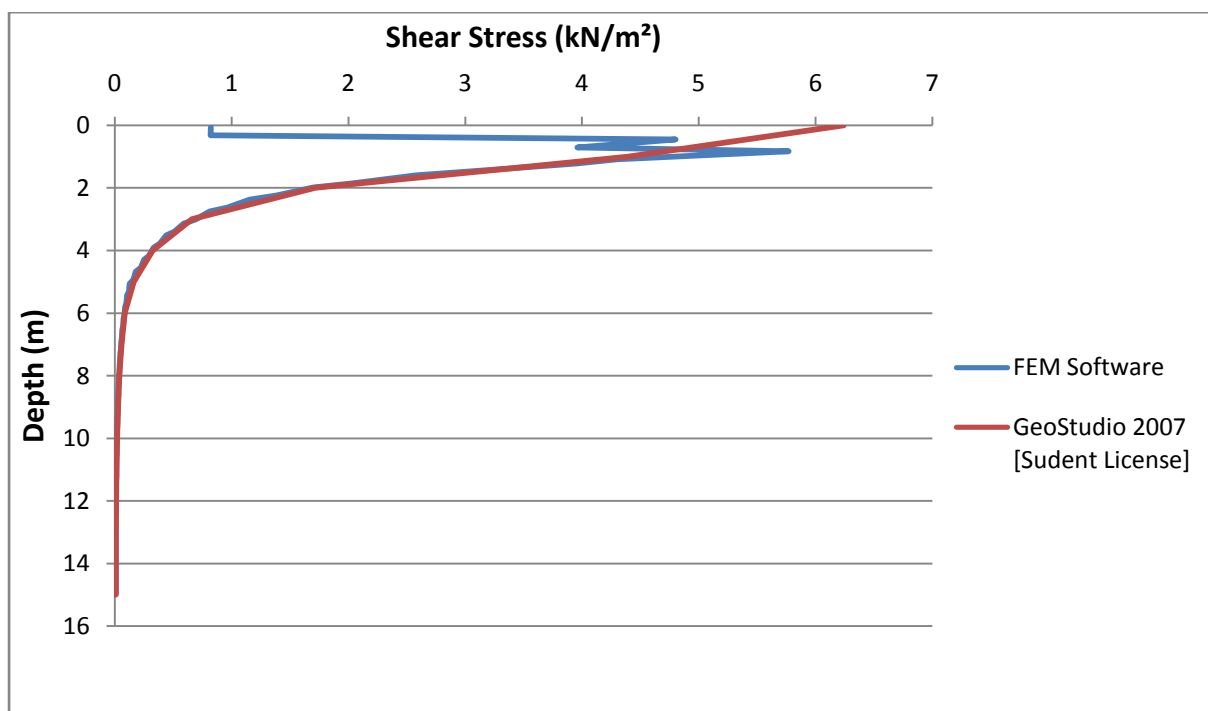
**Figure 4.42:** Distribution of Vertical Displacement along the Axis of a Point Load Acting on the Surface of a Two-Layered System



**Figure 4.43:** Distribution of Vertical Stress along the Axis of a Point Load Acting on the Surface of a Two-Layered System



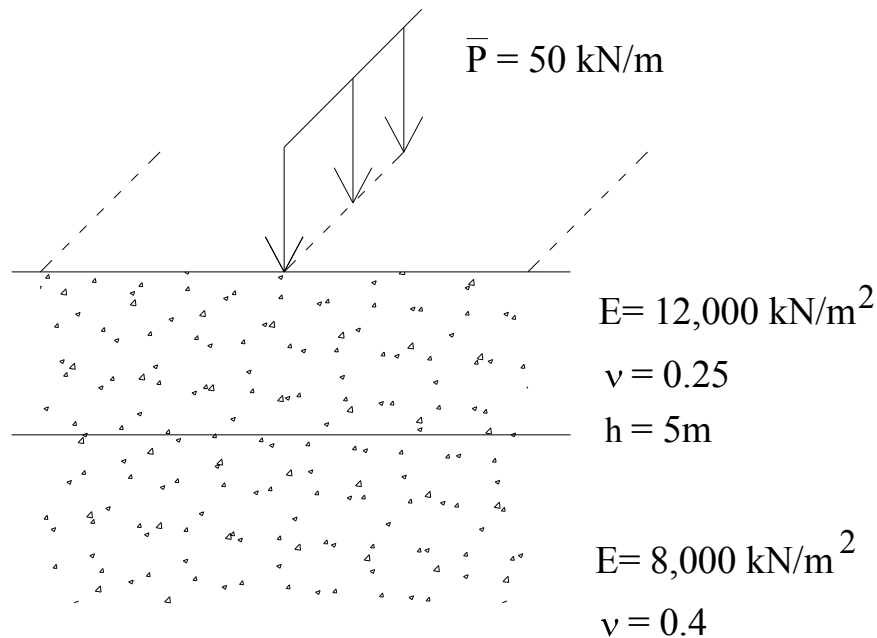
**Figure 4.44:** Distribution of Vertical Strain along the Axis of a Point Load Acting on the Surface of a Two-Layered System



**Figure 4.45:** Distribution of Shear Stress along a Plane at 1m Offset from a Point Load Acting on the Surface of a Two-Layered System

#### 4.2.2.1.2. Line Load

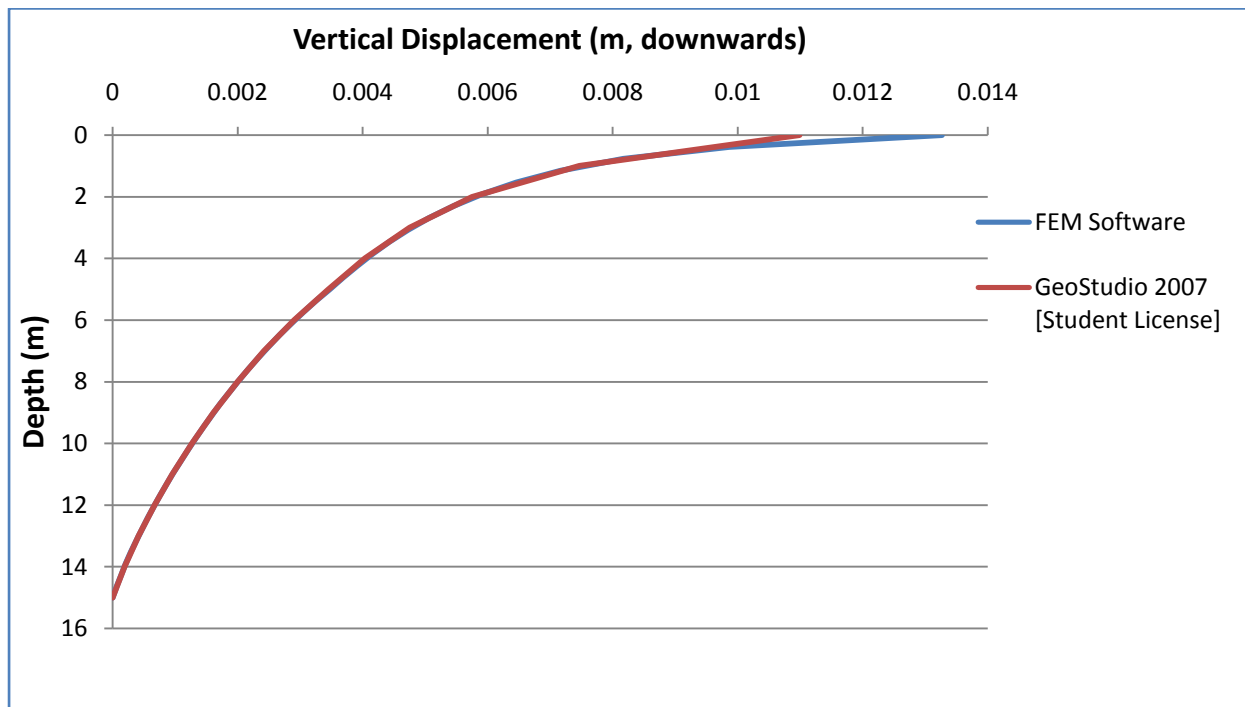
Sample problem 10: Line Load Acting on the Surface of a Double Layered System



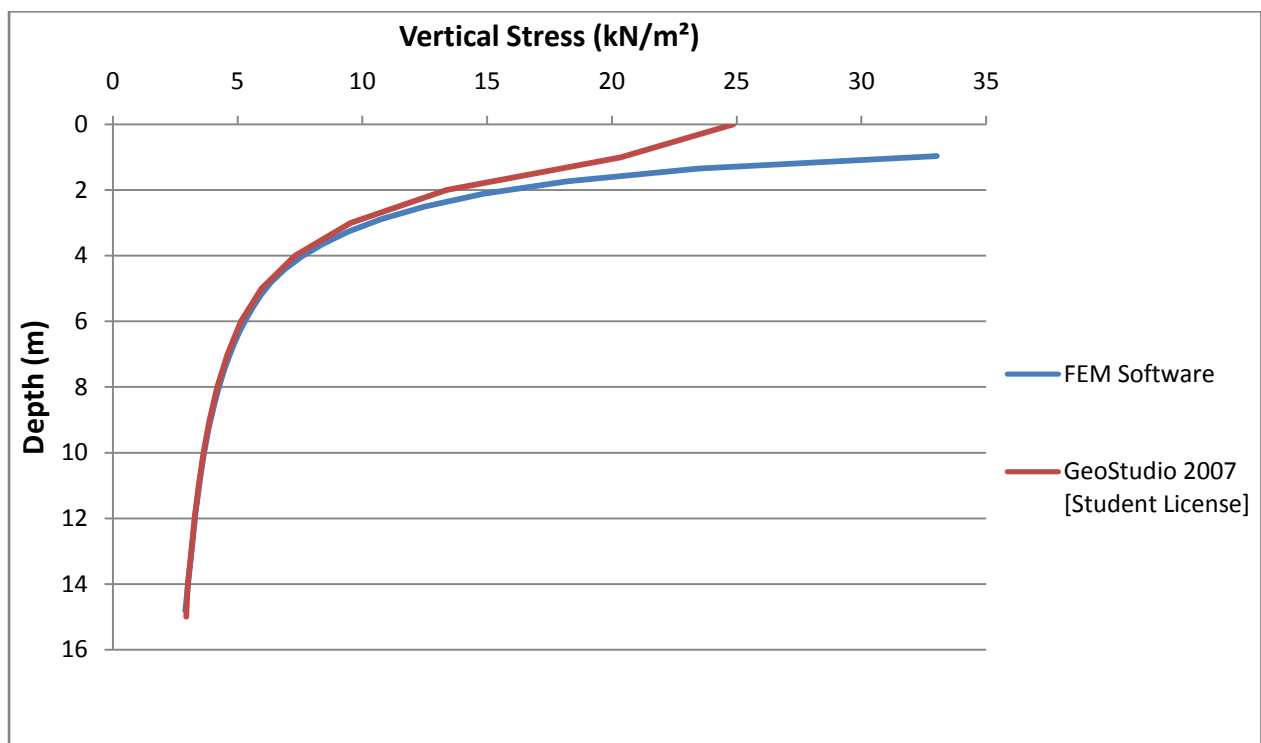
**Figure 4.46**

For the problem of a continuous line load acting on the surface of a two layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis parallel to the plane of loading at a distance of 1m.

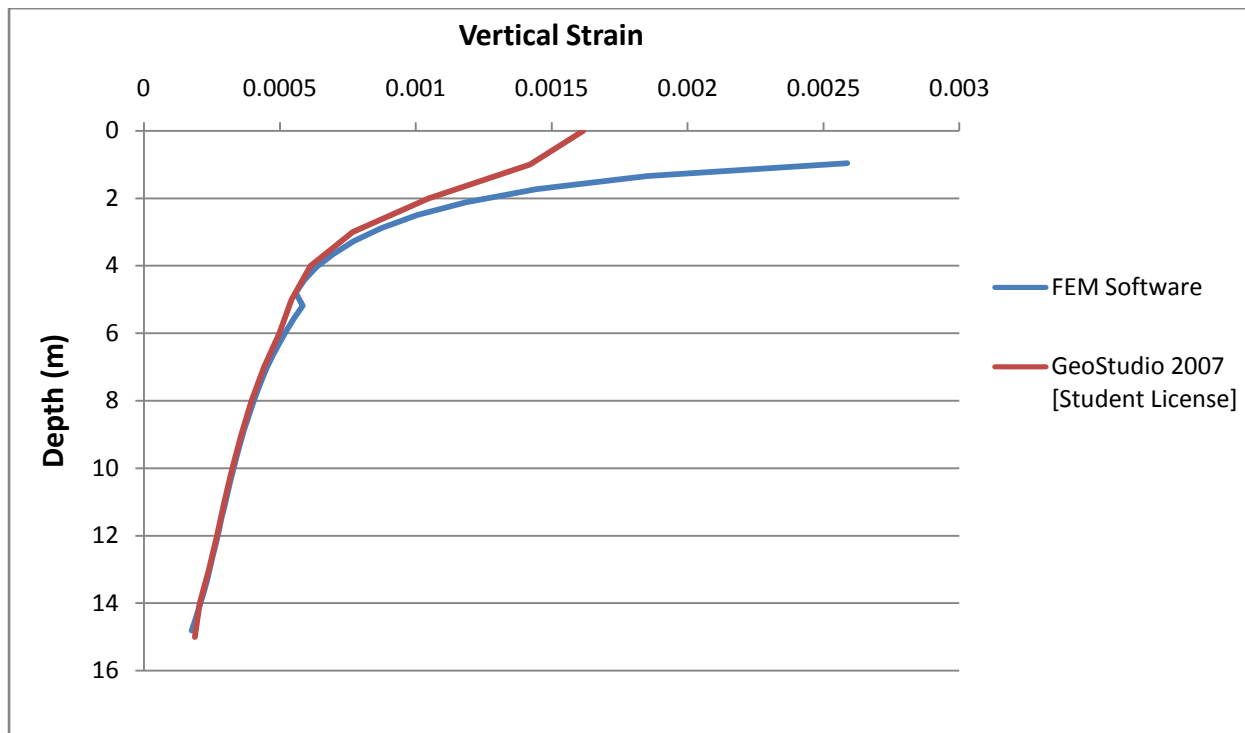




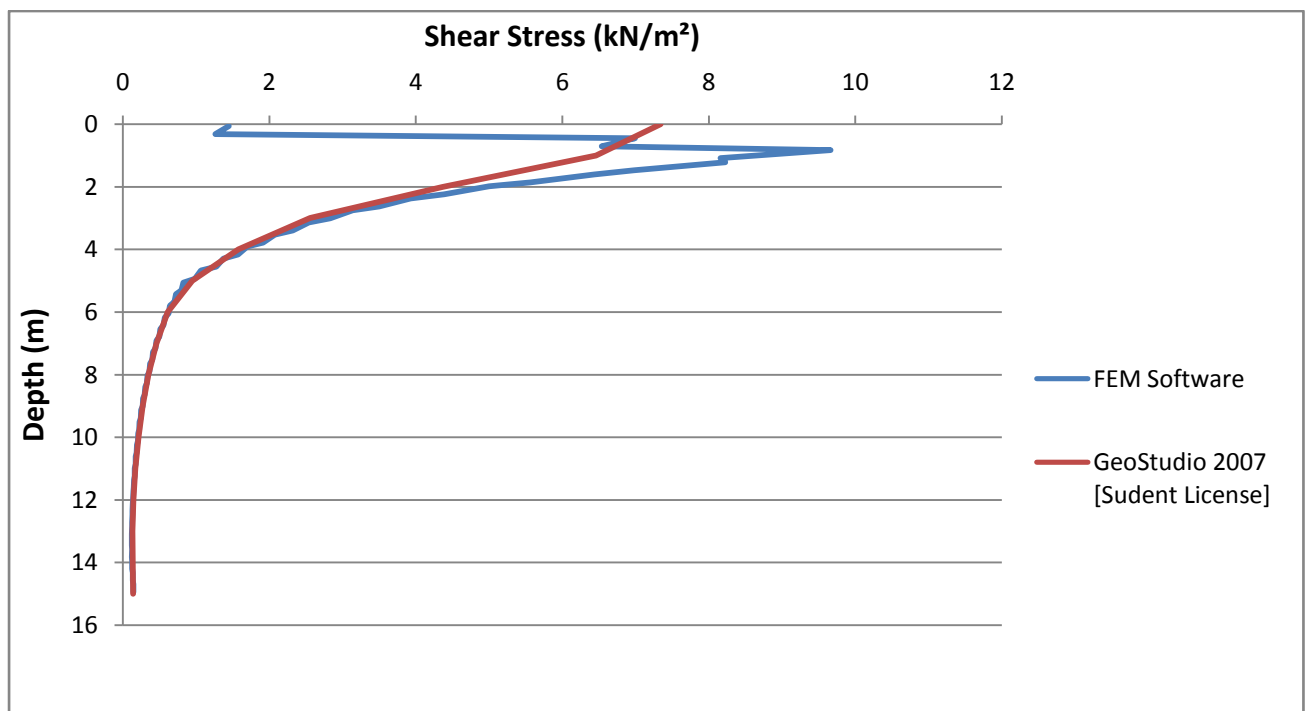
**Figure 4.47:** Distribution of Vertical Displacement along the Axis of a Continuous Line Load Acting on the Surface of a Two-Layered System



**Figure 4.48:** Distribution of Vertical Stress along the Axis of a Continuous Line Load Acting on the Surface of a Two-Layered System



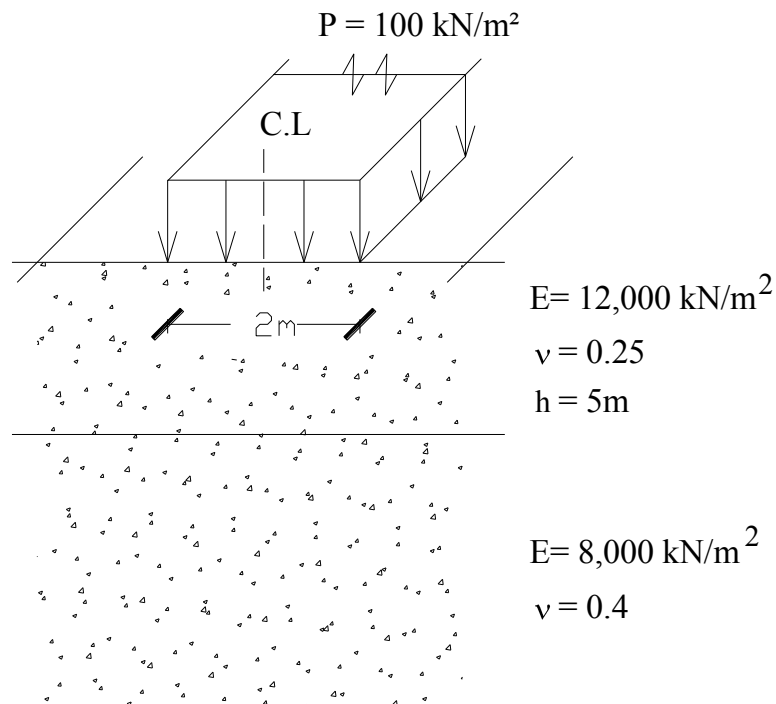
**Figure 4.49:** Distribution of Vertical Strain along the Axis of a Continuous Line Load Acting on the Surface of a Two-Layered System



**Figure 4.50:** Distribution of Shear Stress at an Axis 1m Offset from a Continuous Line Load Acting on the Surface of a Two-Layered System

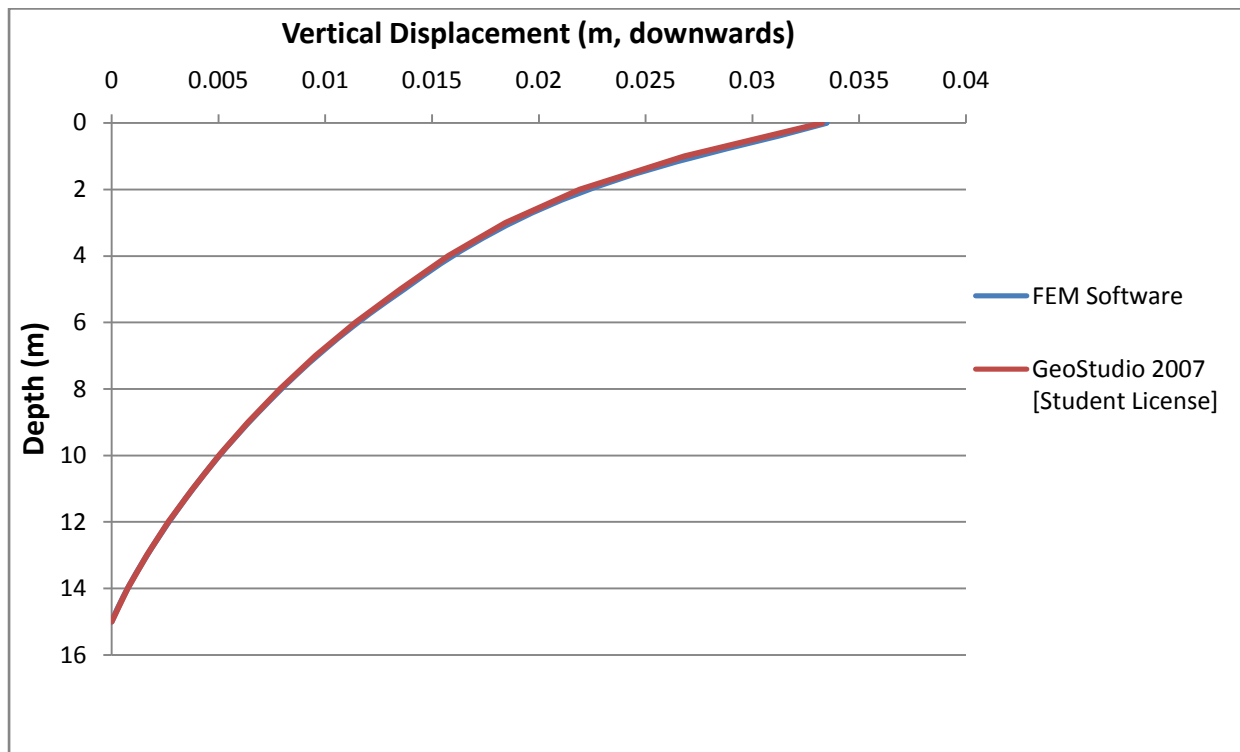
#### 4.2.2.1.3. Uniform Strip Load

Sample problem 11: Strip Load Acting on the Surface of a Double Layered System

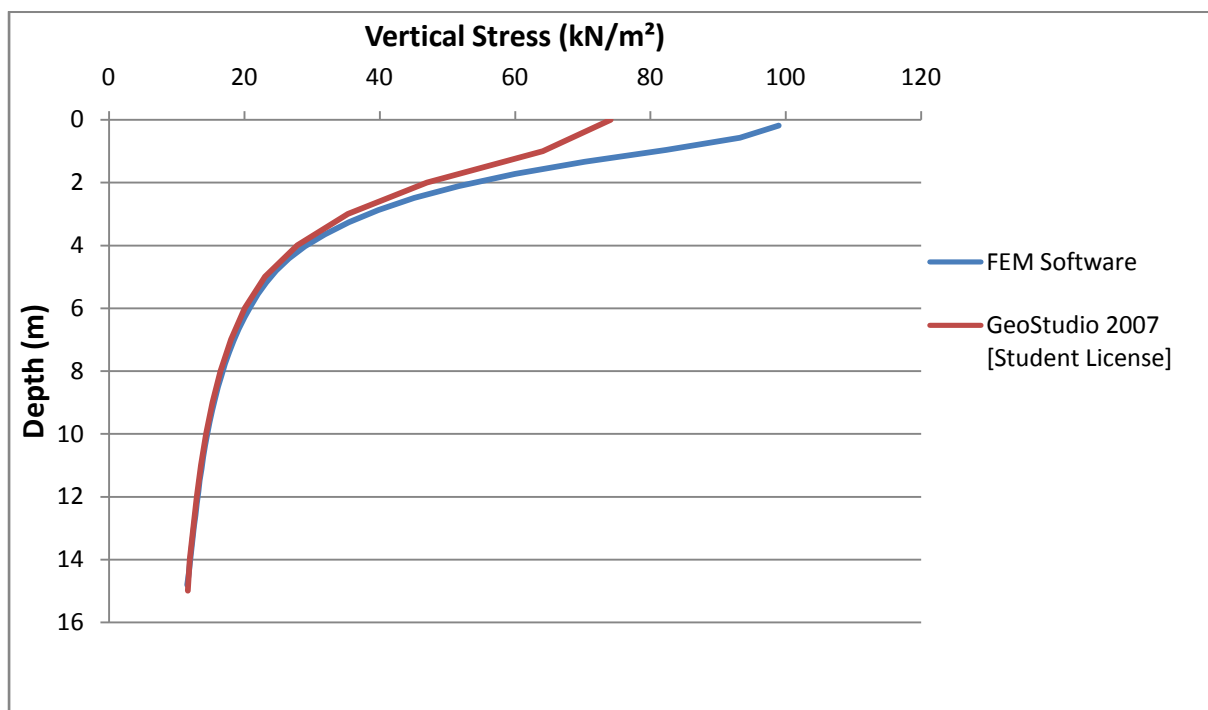


**Figure 4.51**

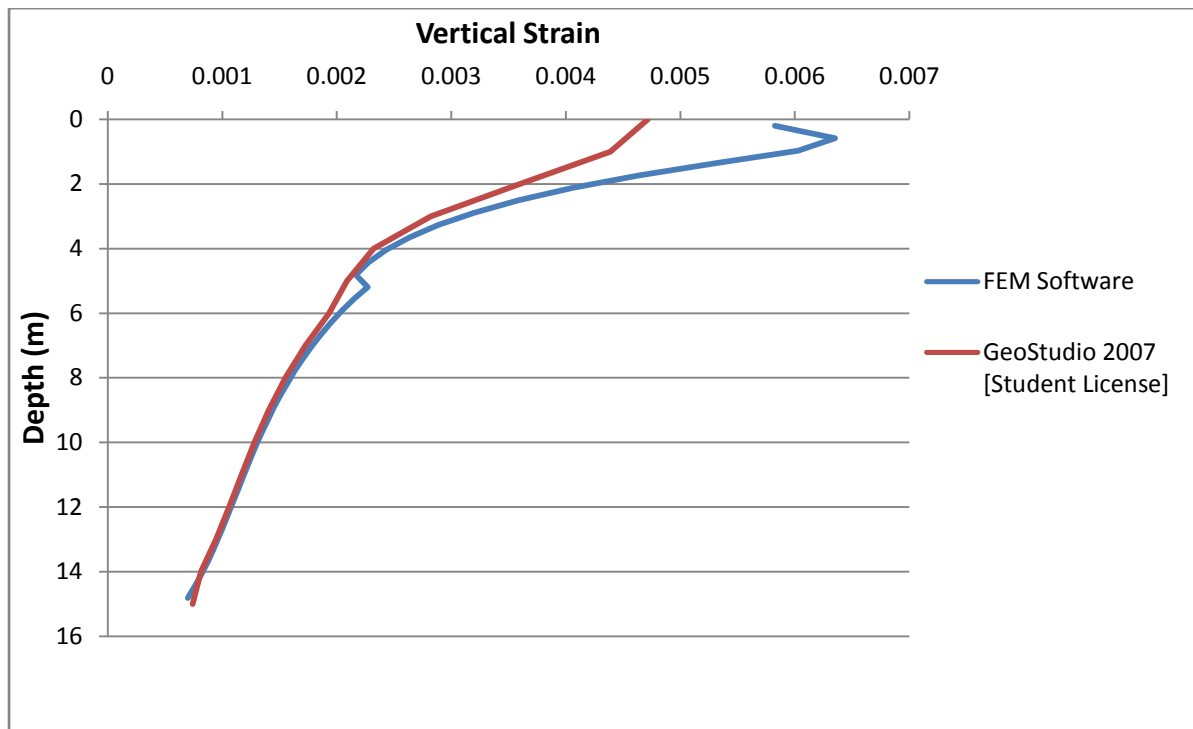
For the problem of a uniformly distributed strip load acting on the surface of a two layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the centerline of loading. For the shear stress distribution, comparison of result is performed at the edge of loading.



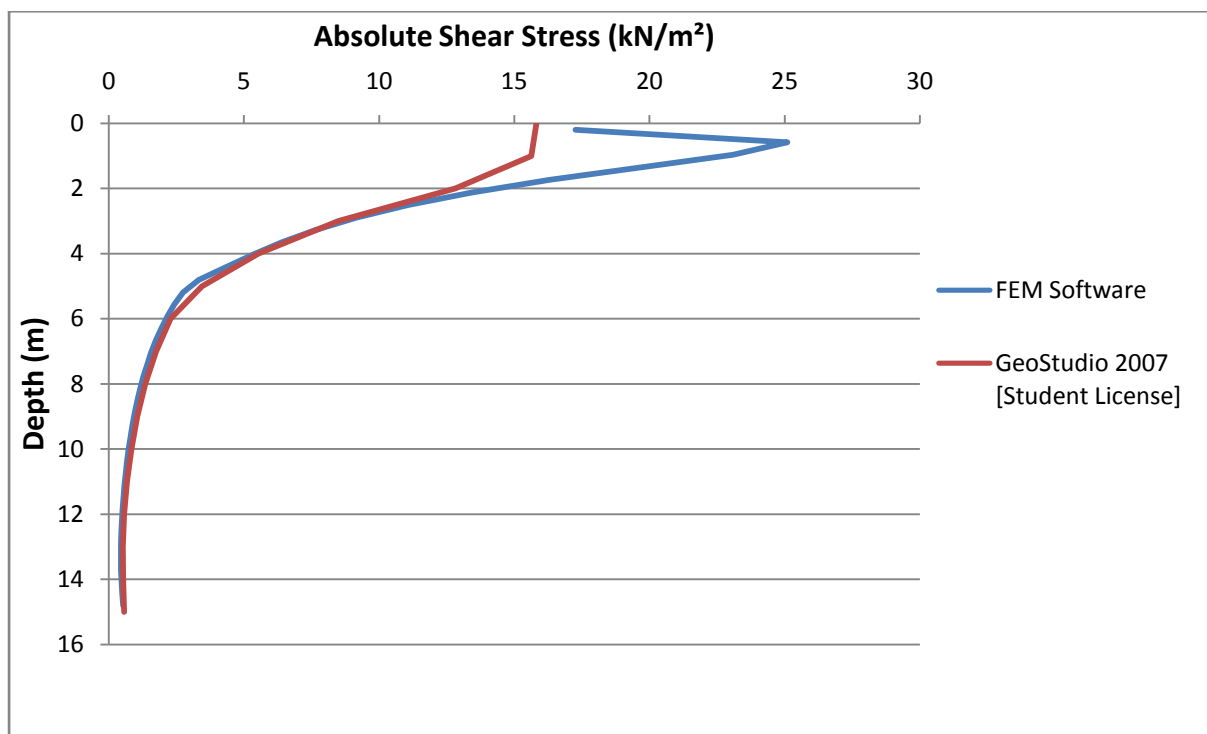
**Figure 4.52:** Distribution of Vertical Displacement along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Two-Layered System



**Figure 4.53:** Distribution of Vertical Stress along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Two-Layered System



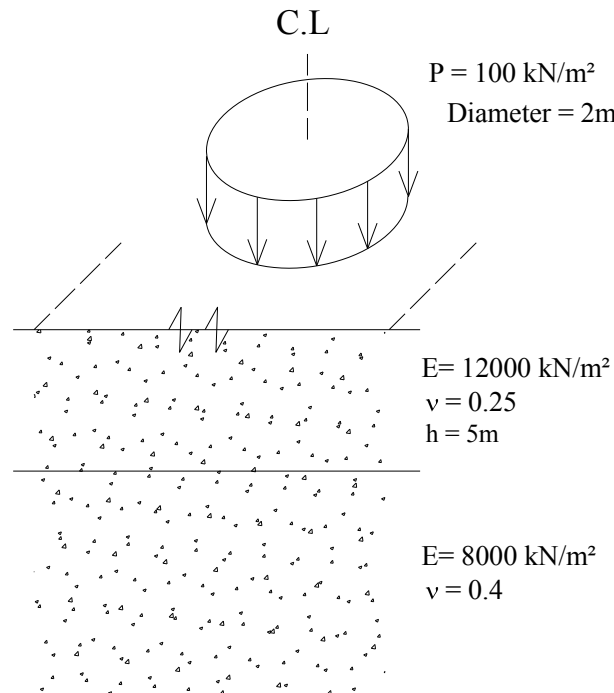
**Figure 4.54:** Distribution of Vertical Strain along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Two-Layered System



**Figure 4.55:** Distribution of Shear Stress along the Edge of a Uniformly Distributed Strip Load Acting on the Surface of a Two-Layered System

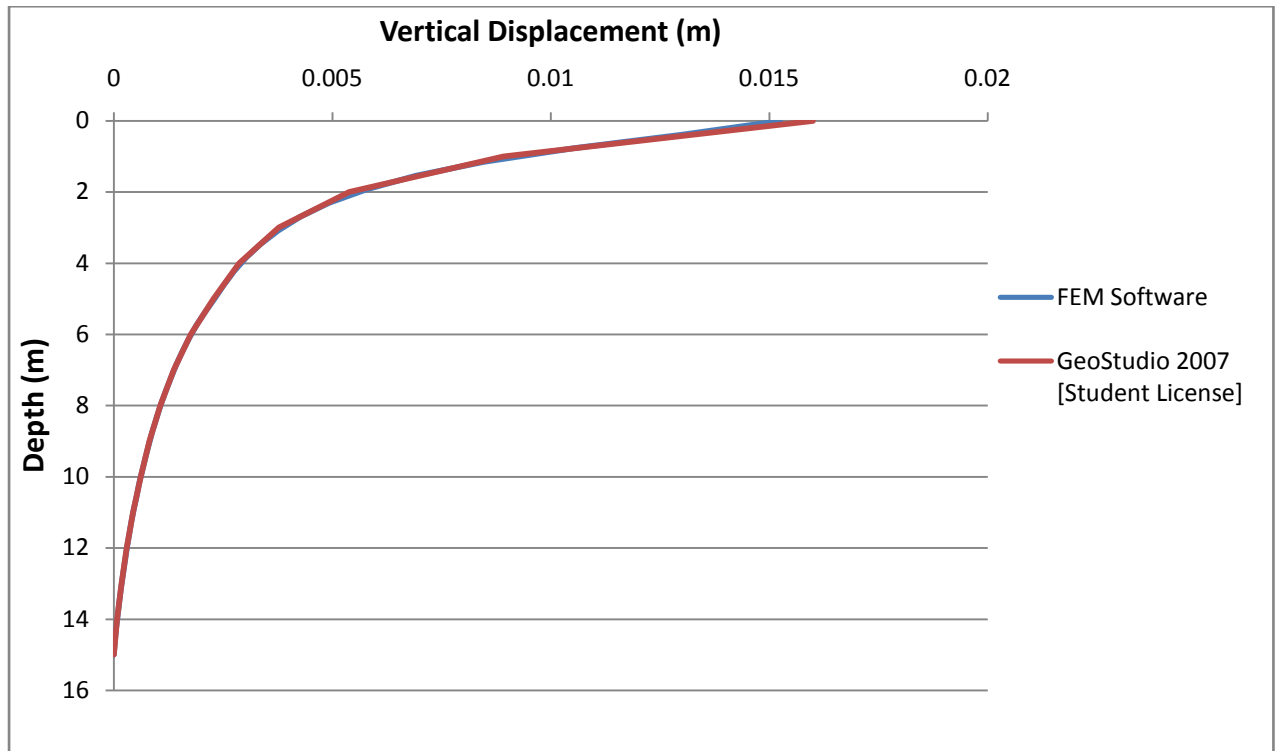
#### 4.2.2.1.4. Uniform Load of Circular Plan Area

Sample problem 12: Uniform Load of Circular Plan Area Acting on the Surface of a Double Layered System

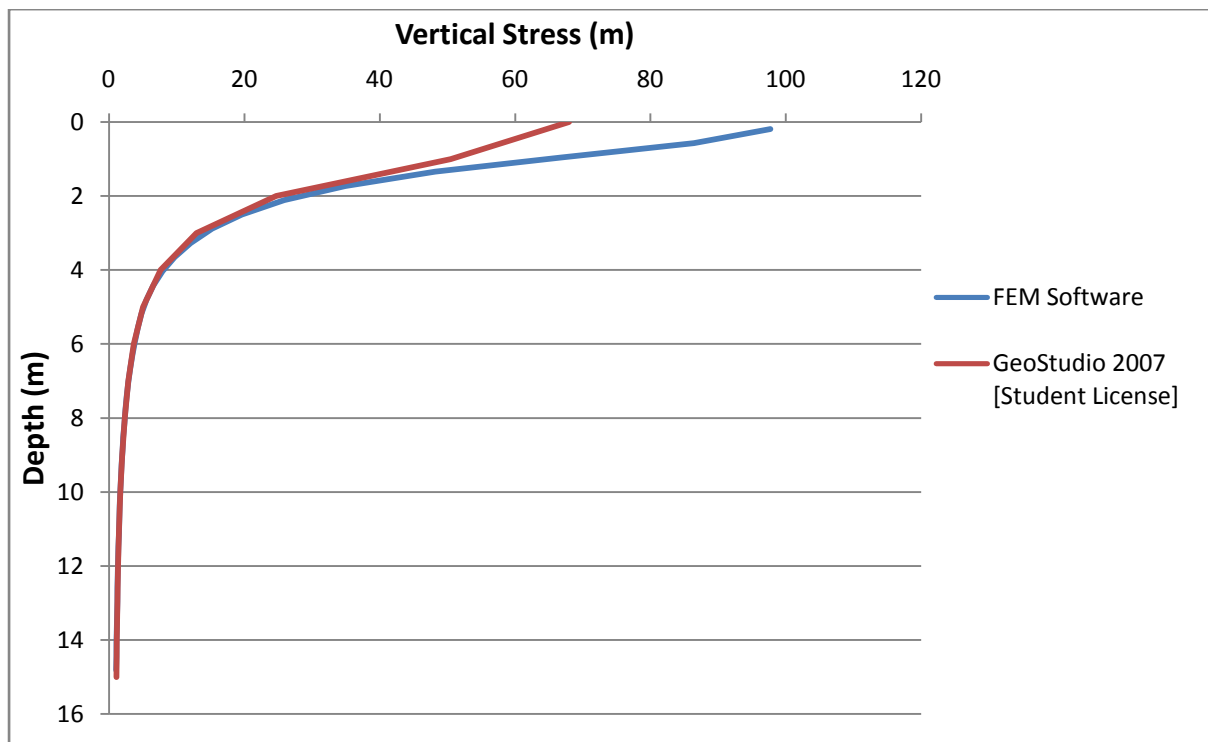


**Figure 4.56**

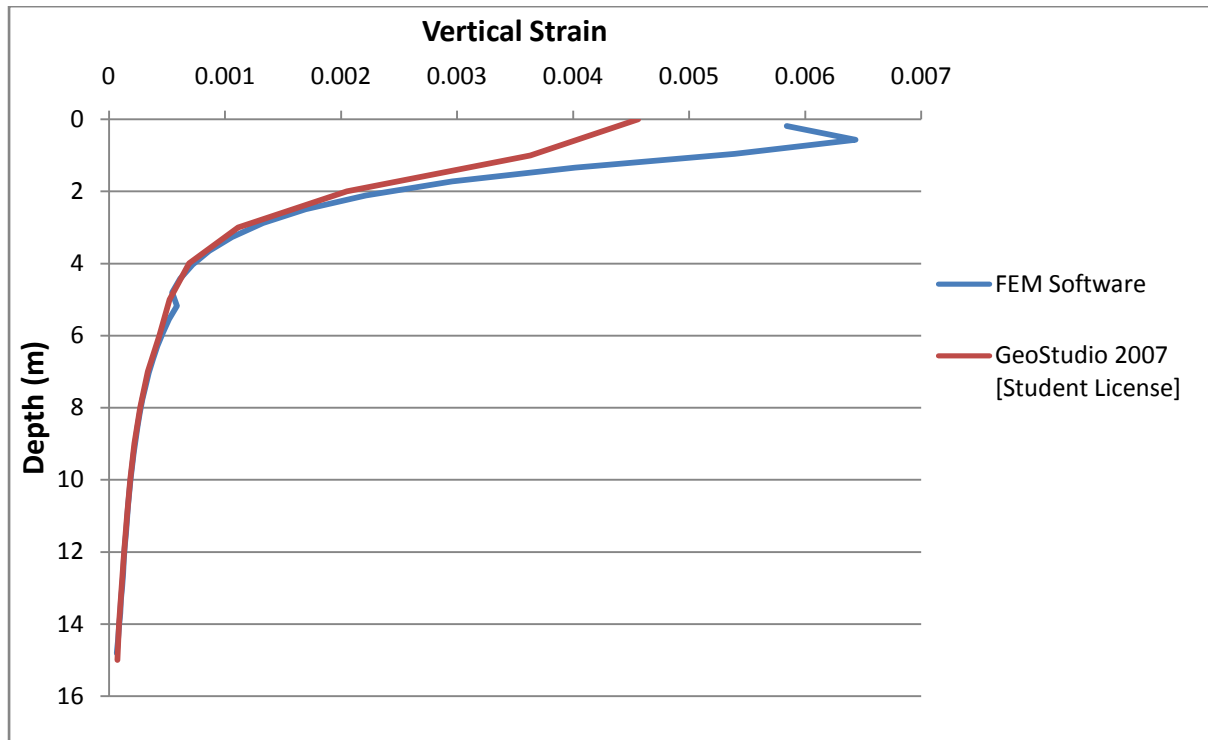
For the problem of a uniform load of circular plan area acting on the surface of a double layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the centerline of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of circular loading.



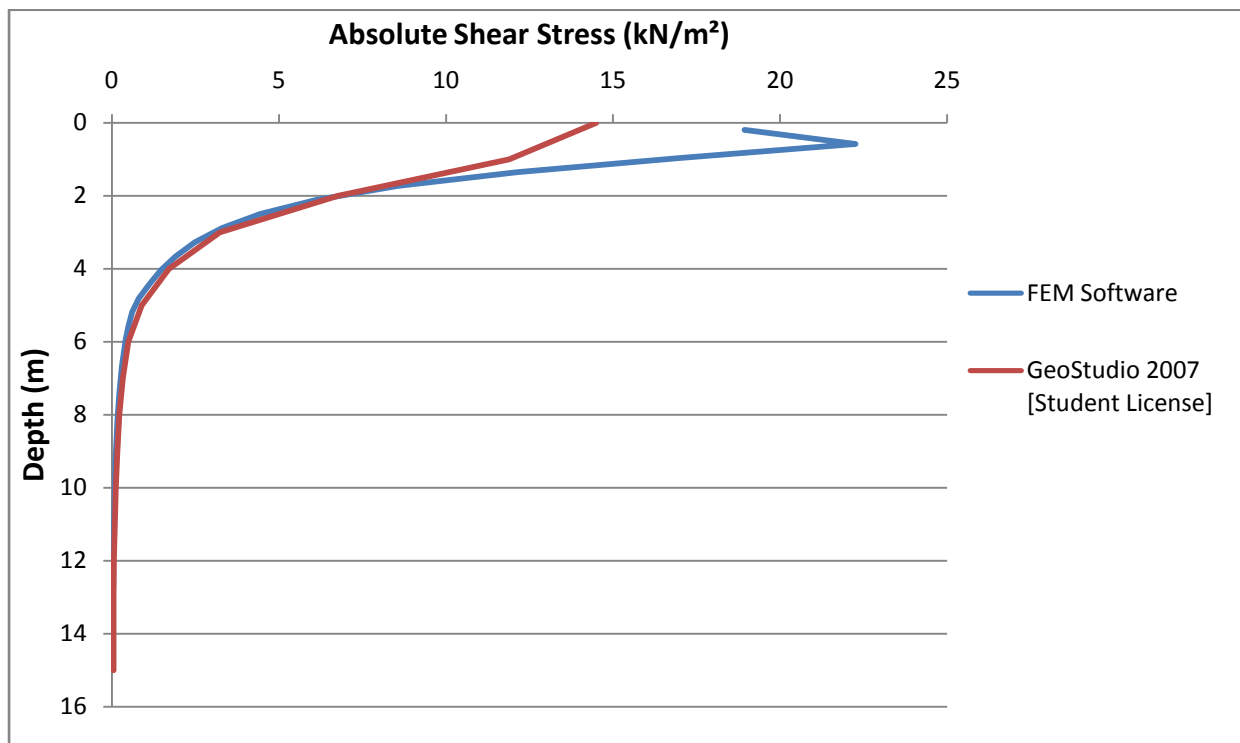
**Figure 4.57:** Distribution of Vertical Displacement along the Centerline of a Uniform Load of Circular Plan Area Acting on a Double Layered System



**Figure 4.58:** Distribution of Vertical Stress along the Centerline of a Uniform Load of Circular Plan Area Acting on a Double Layered System



**Figure 4.59:** Distribution of Vertical Strain along the Centerline of a Uniform Load of Circular Plan Area Acting on a Double Layered System



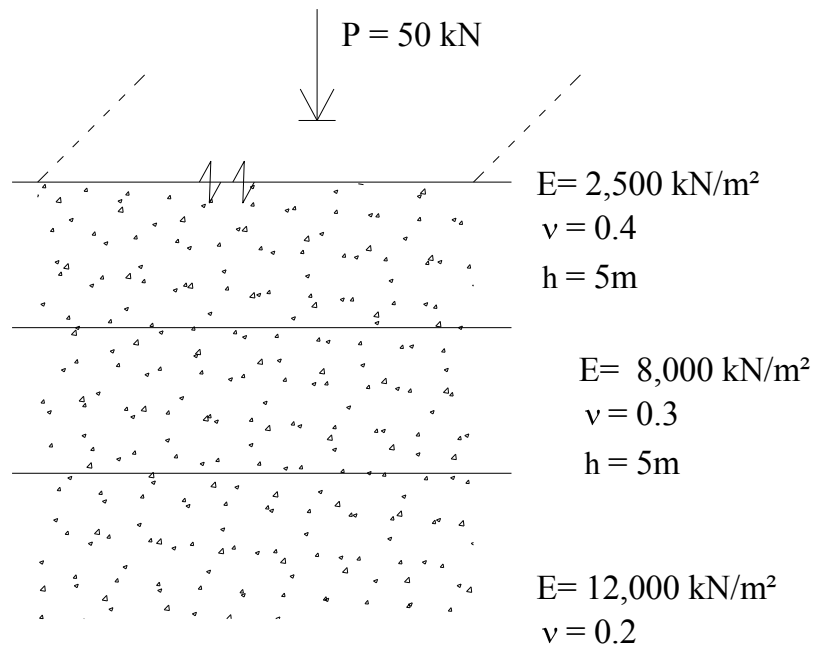
**Figure 4.60:** Distribution of Shear Stress along the Centerline of a Uniform Load of Circular Plan Area Acting on a Double Layered System



### 4.2.3. Three Layers- Surface Loads

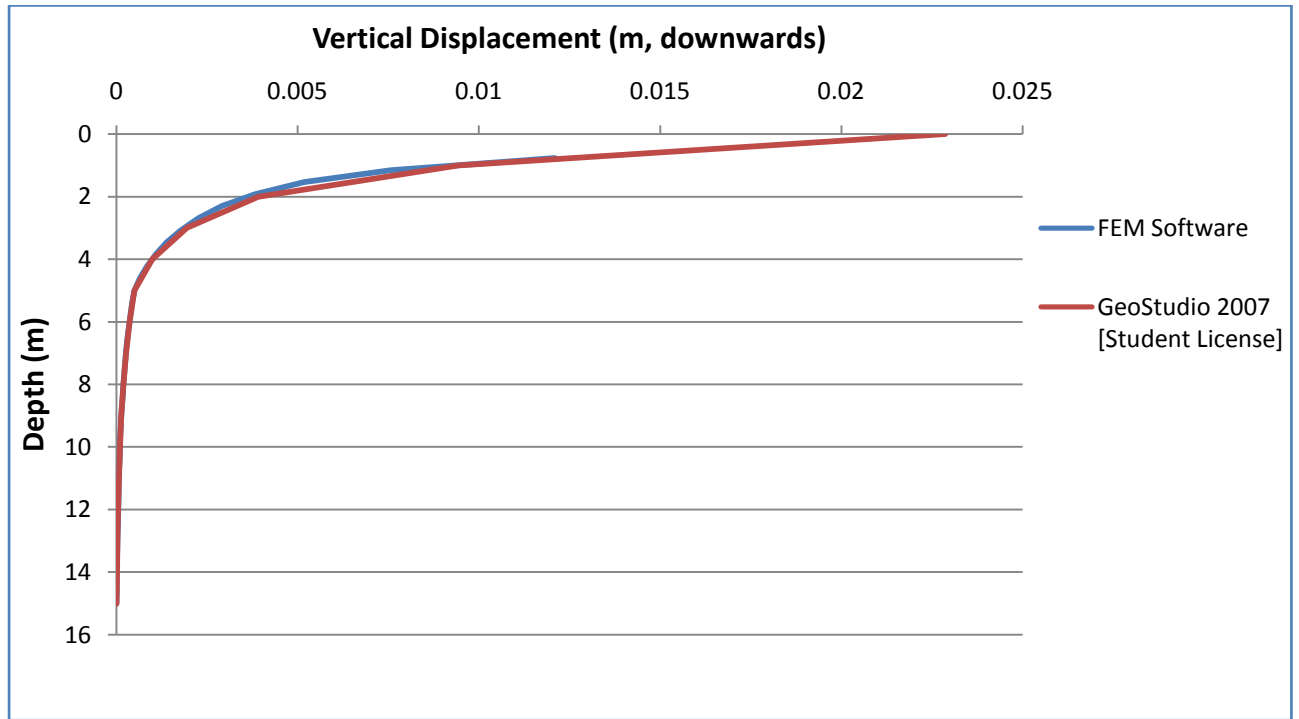
#### 4.2.3.1. Point Load

Sample problem 13: Point Load Acting on the Surface of a Triple Layered System

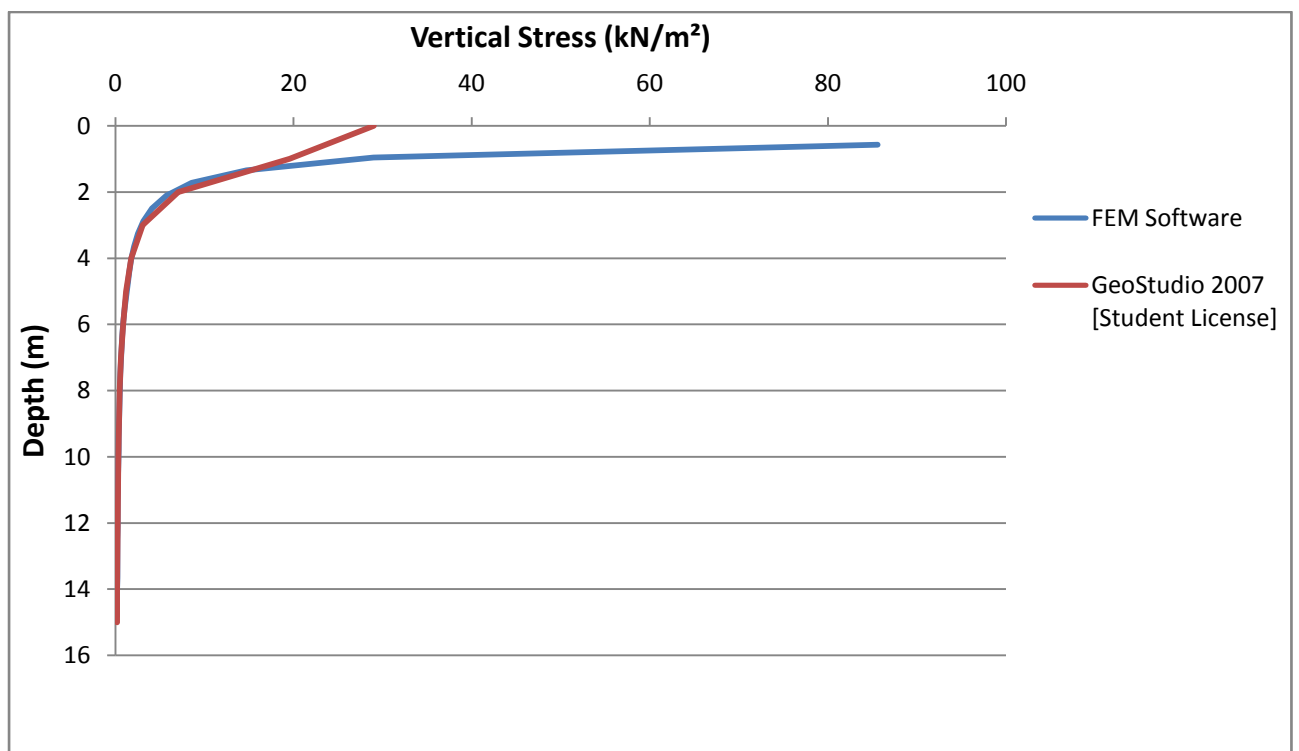


**Figure 4.61**

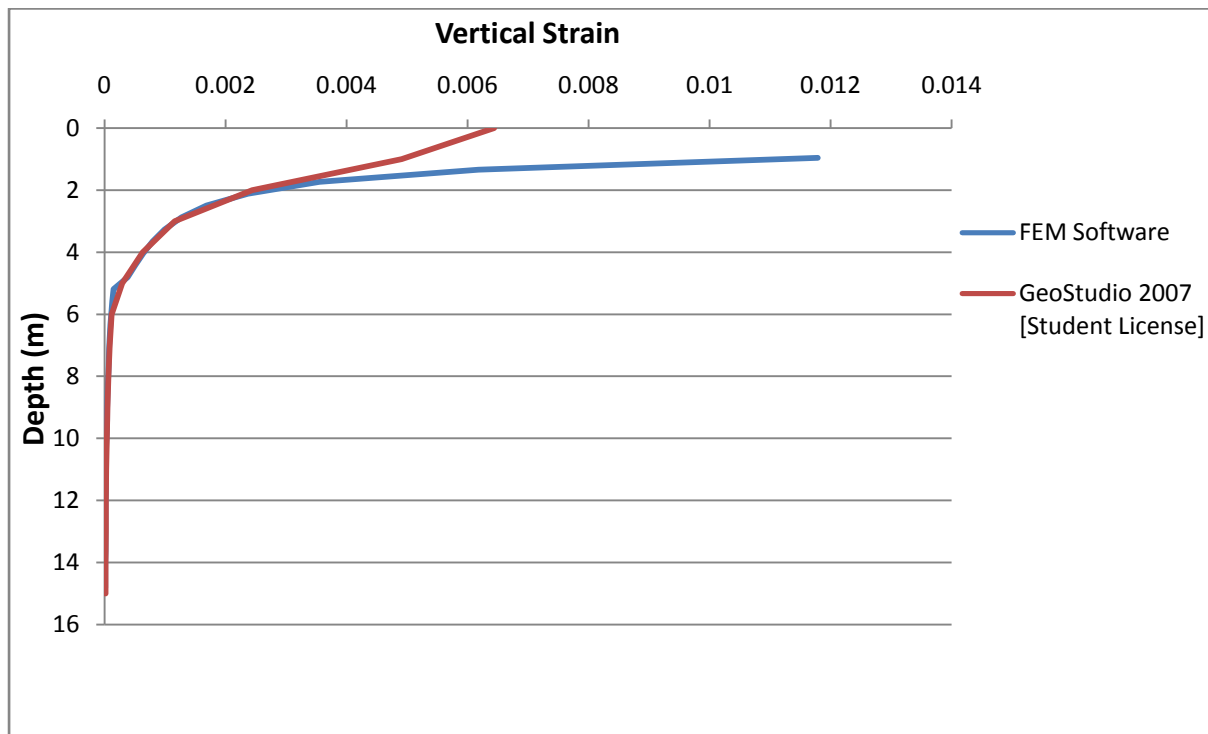
For the problem of a point load acting on the surface of a three layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed along a plane at a 1m radial distance from the point load.



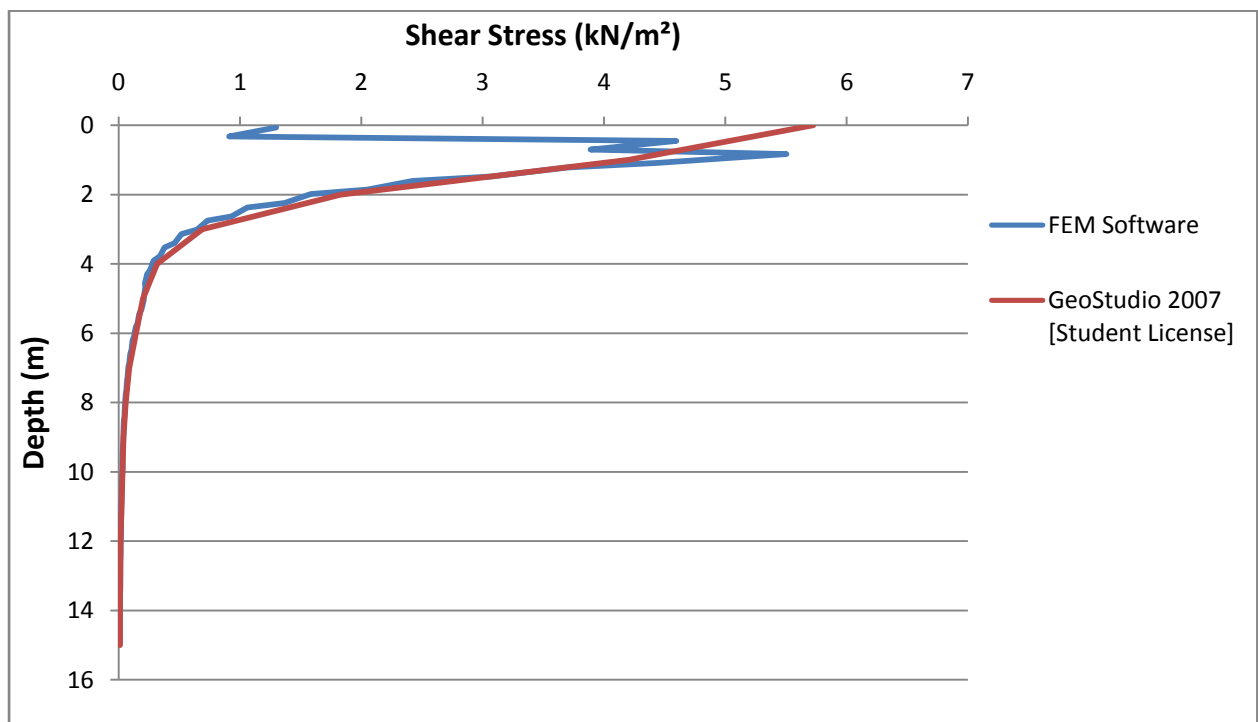
**Figure 4.62:** Distribution of Vertical Displacement along the Axis of a Point Load Acting on the Surface of a Three-Layered System



**Figure 4.63:** Distribution of Vertical Stress along the Axis of a Point Load Acting on the Surface of a Three-Layered System



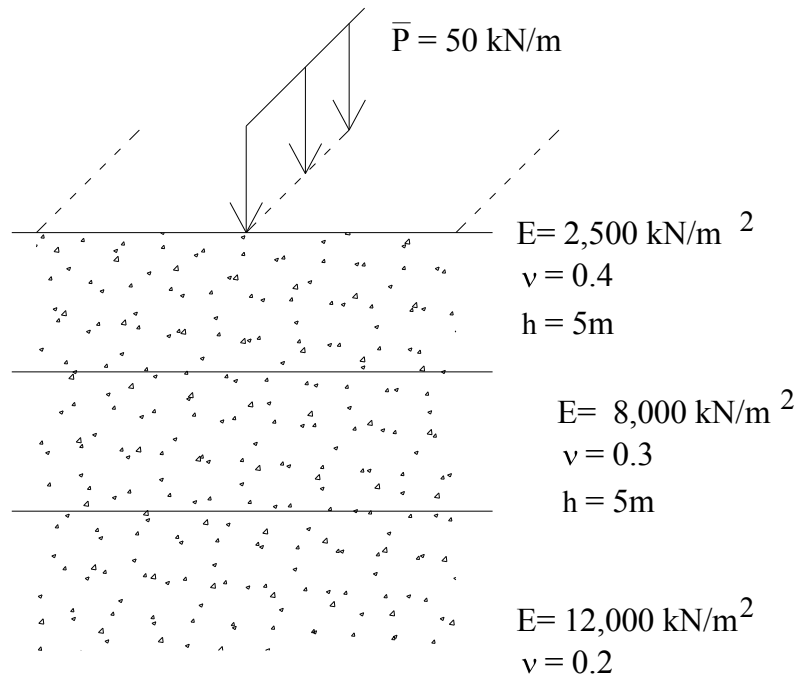
**Figure 4.64:** Distribution of Vertical Strain along the Axis of a Point Load Acting on the Surface of a Three-Layered System



**Figure 4.65:** Distribution of Shear Stress along the Axis of a Point Load Acting on the Surface of a Three-Layered System

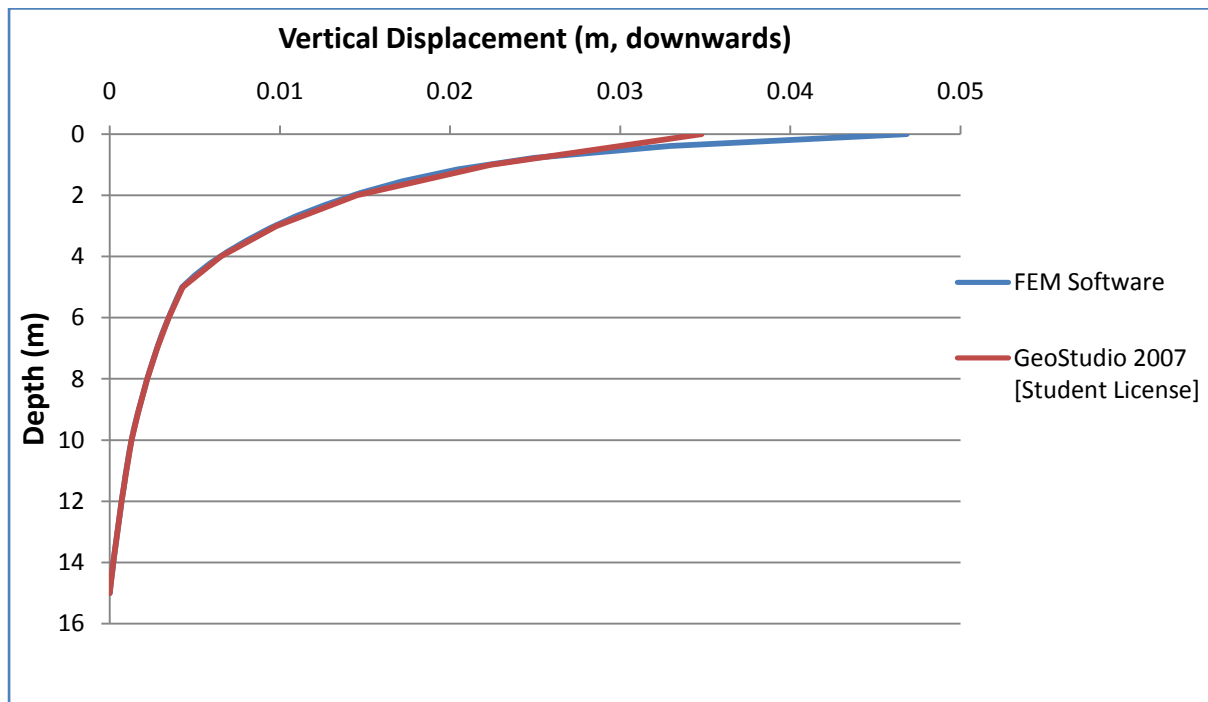
#### 4.2.3.2. Line Load

##### Sample problem 14: Line Load Acting on the Surface of a Triple Layered System

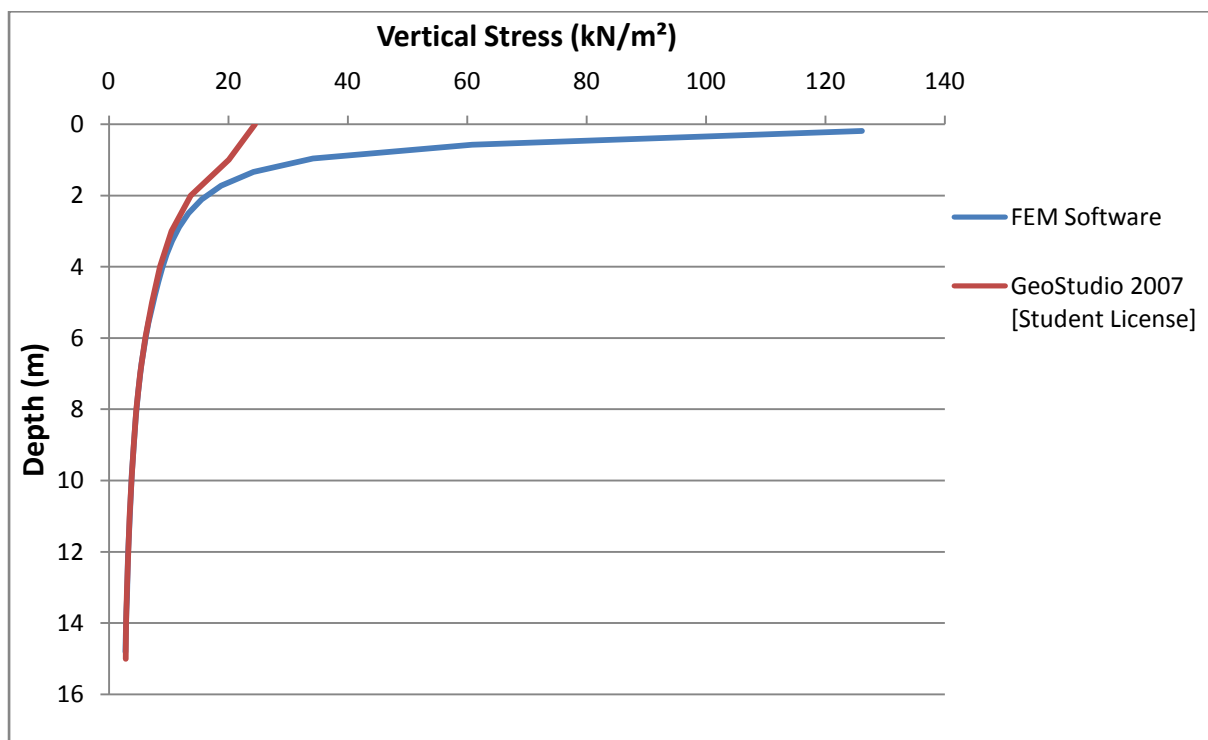


**Figure 4.66**

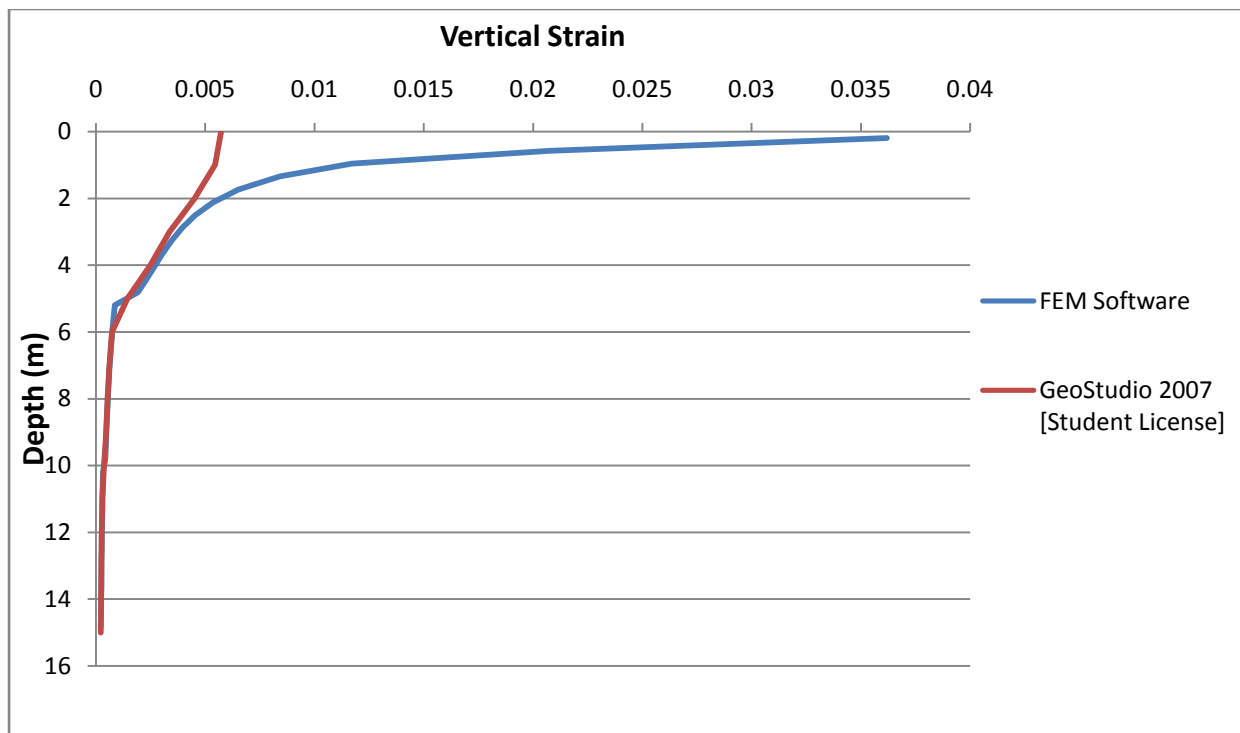
For the problem of a continuous line load acting on the surface of a three layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis parallel to the plane of loading at a distance of 1m.



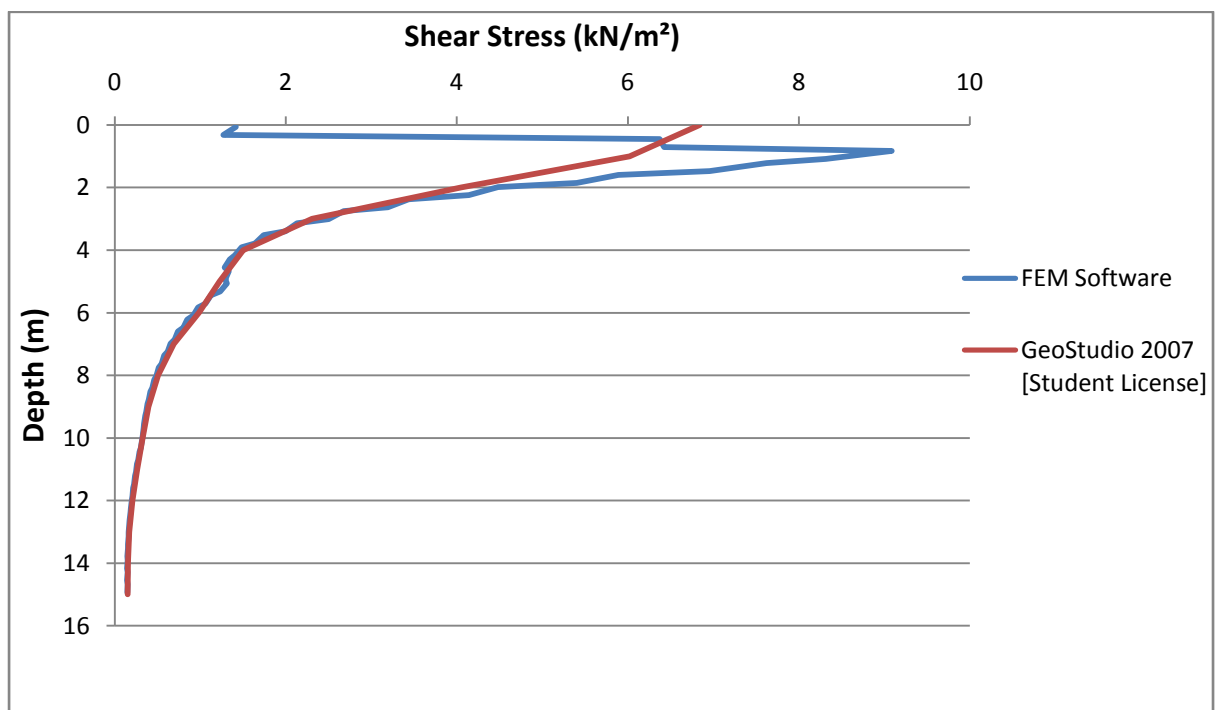
**Figure 4.67:** Distribution of Vertical Displacement along the Axis of a Continuous Line Load Acting on the Surface of a Three-Layered System



**Figure 4.68:** Distribution of Vertical Stress along the Axis of a Continuous Line Load Acting on the Surface of a Three-Layered System



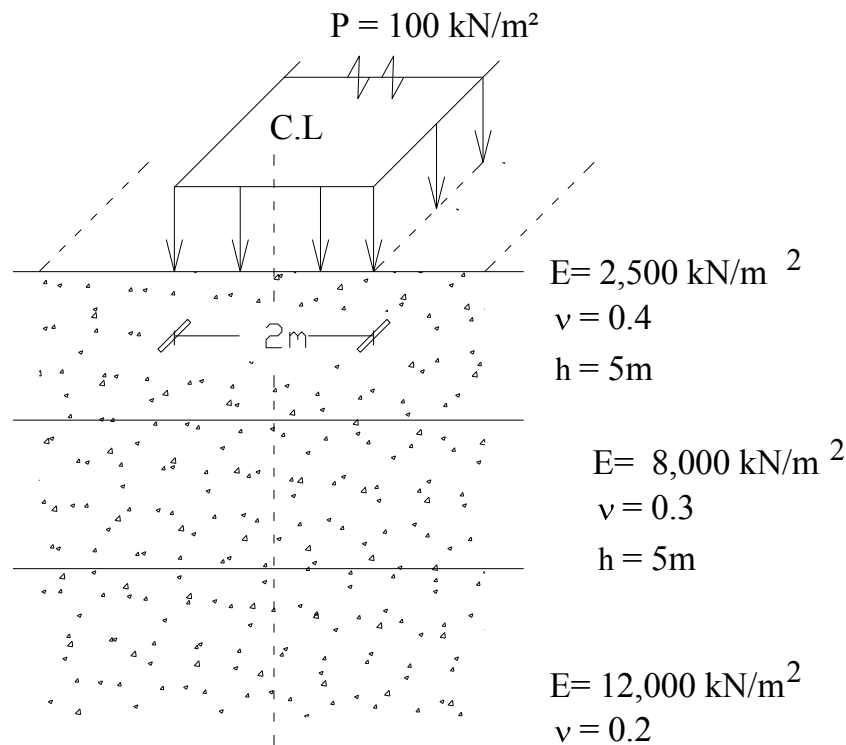
**Figure 4.69:** Distribution of Vertical Strain along the Axis of a Continuous Line Load Acting on the Surface of a Three-Layered System



**Figure 4.70:** Distribution of Shear Stress at an Axis 1m Offset from a Continuous Line Load Acting on the Surface of a Three-Layered System

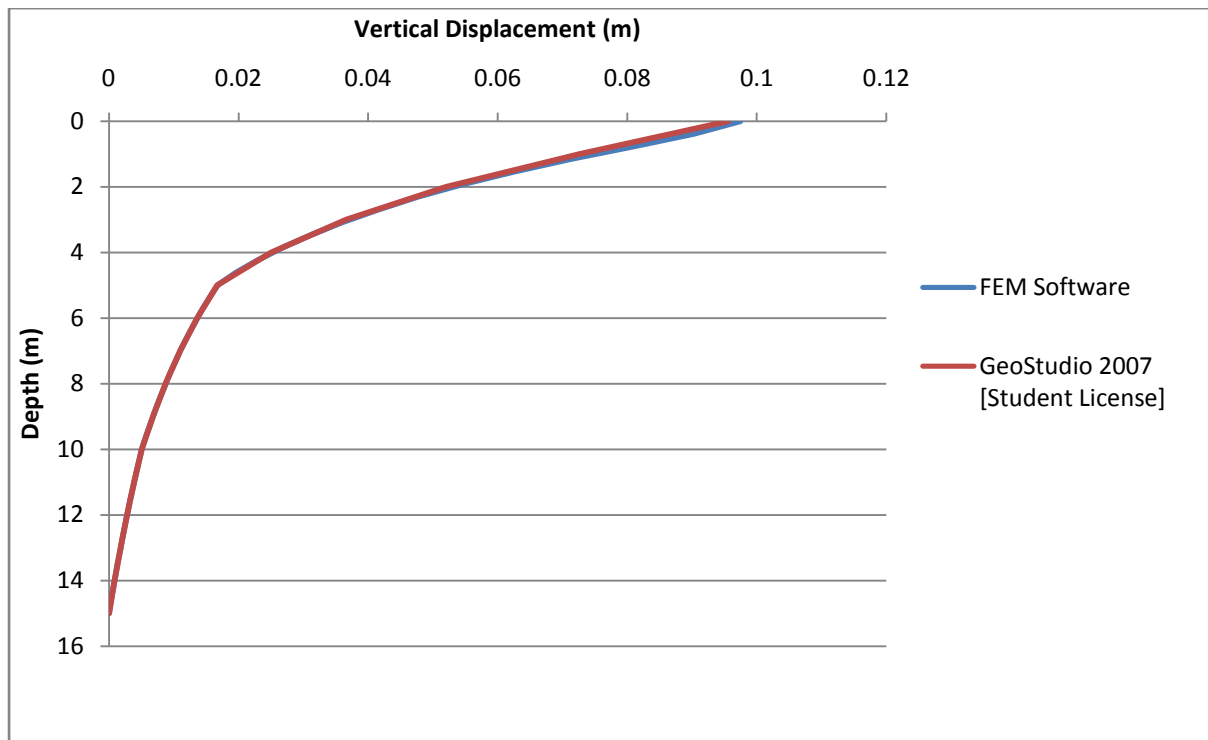
#### 4.2.3.3. Uniform Strip Load

Sample problem 15: Strip Load Acting on the Surface of a Triple Layered System

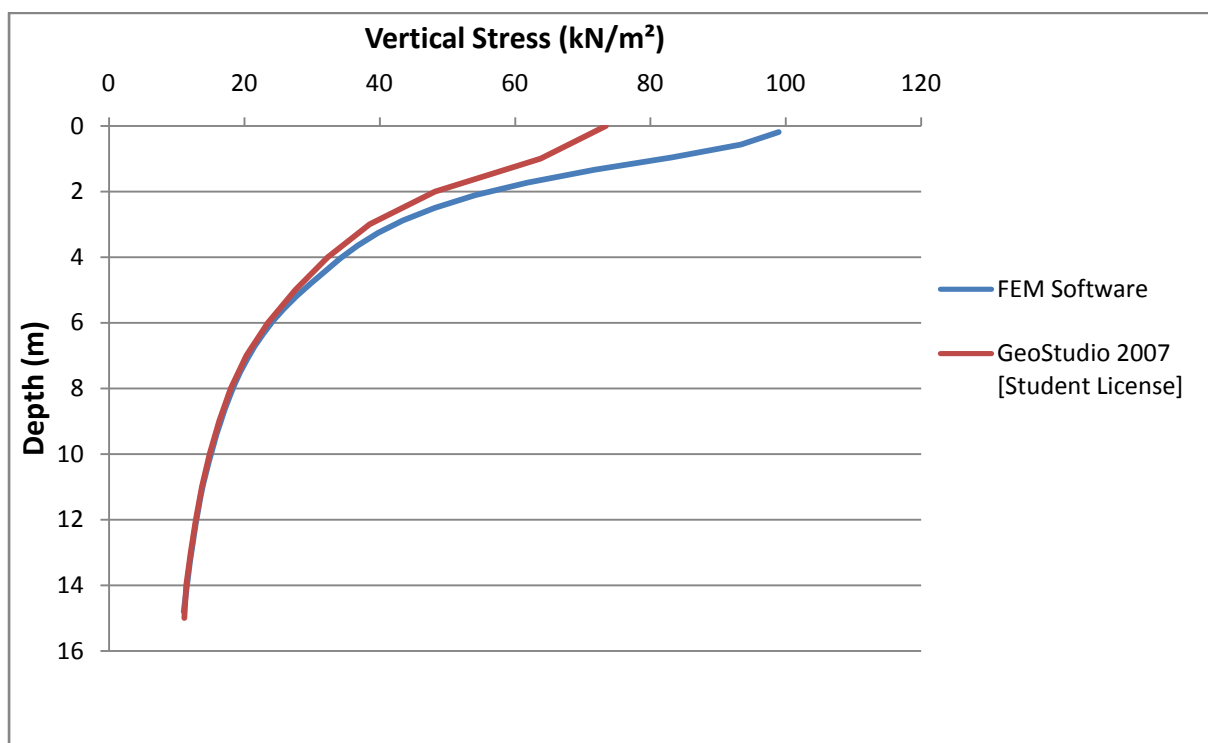


**Figure 4.71**

For the problem of a uniformly distributed strip load acting on the surface of a three layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed at the edge of the uniform strip load.

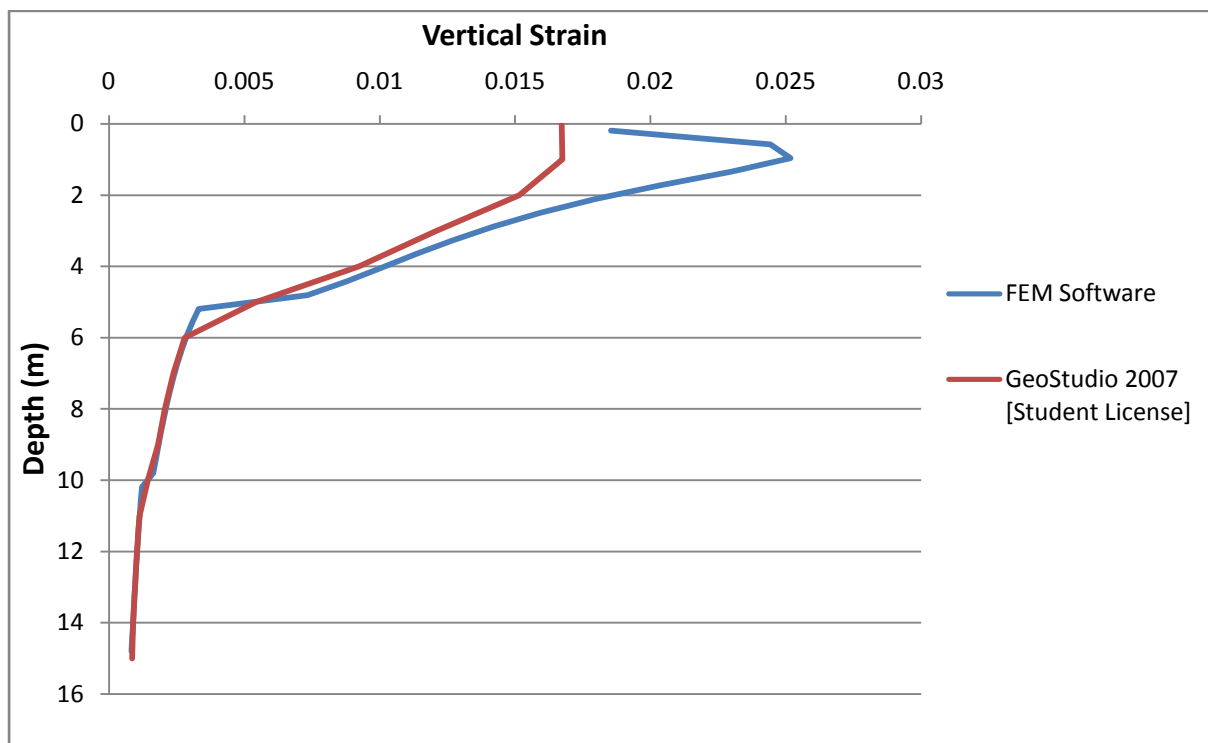


**Figure 4.72:** Distribution of Vertical Displacement along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Three-Layered System

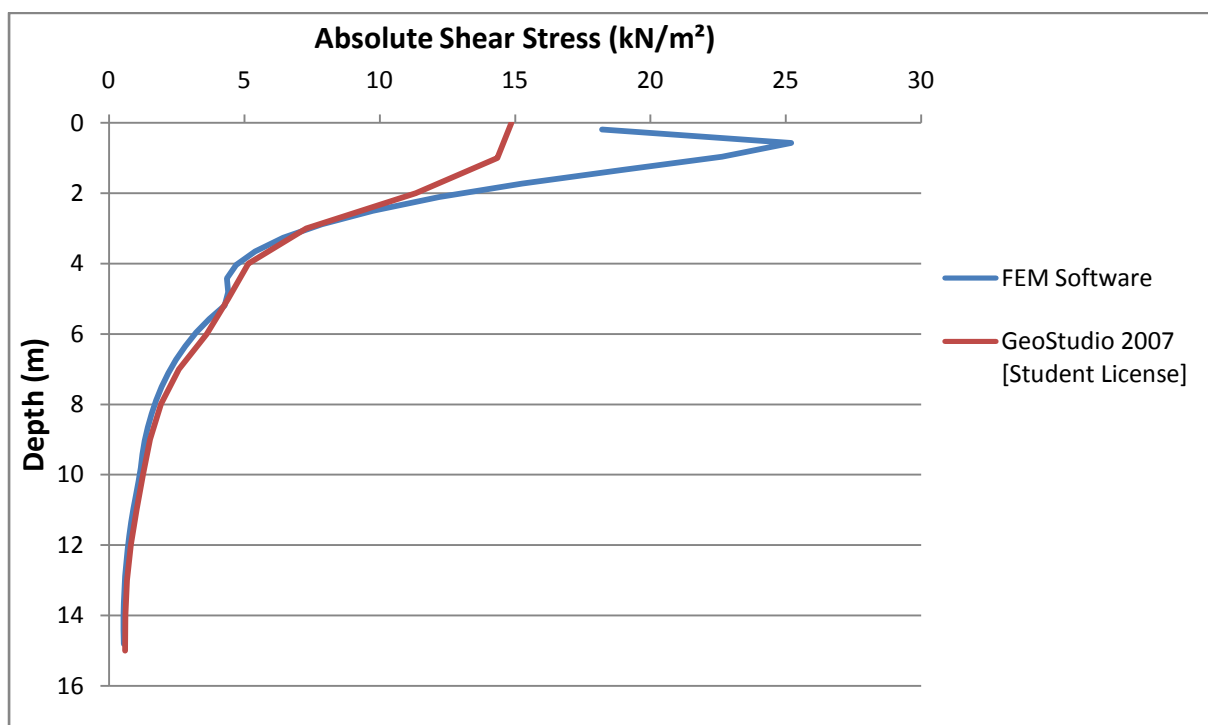


**Figure 4.73:** Distribution of Vertical Stress along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Three -Layered System





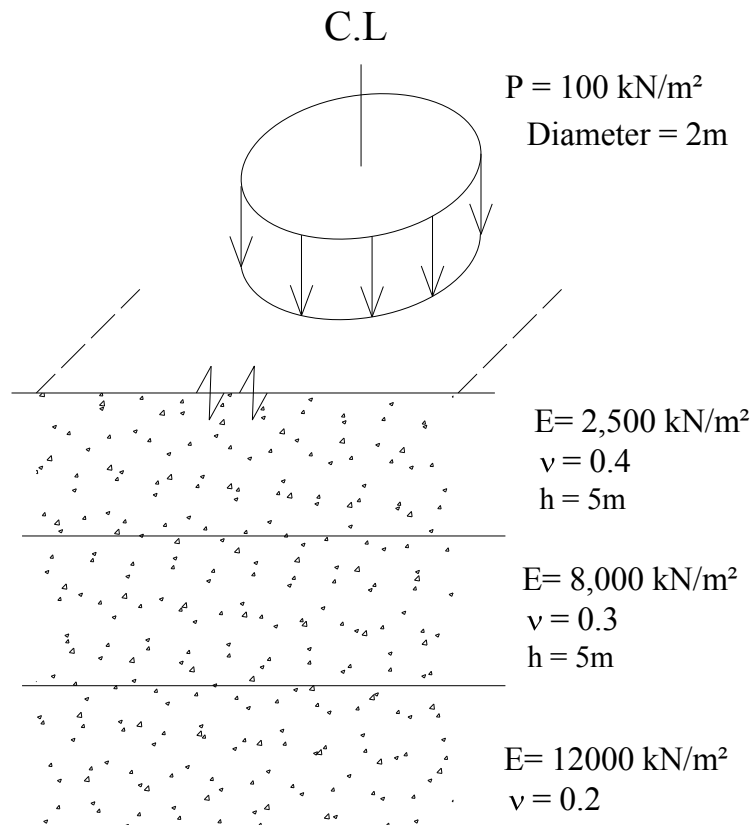
**Figure 4.74:** Distribution of Vertical Strain along the Centerline of a Uniformly Distributed Strip Load Acting on the Surface of a Three -Layered System



**Figure 4.75:** Distribution of Shear Stress at the Edge of a Uniformly Distributed Strip Load Acting on the Surface of a Three -Layered System

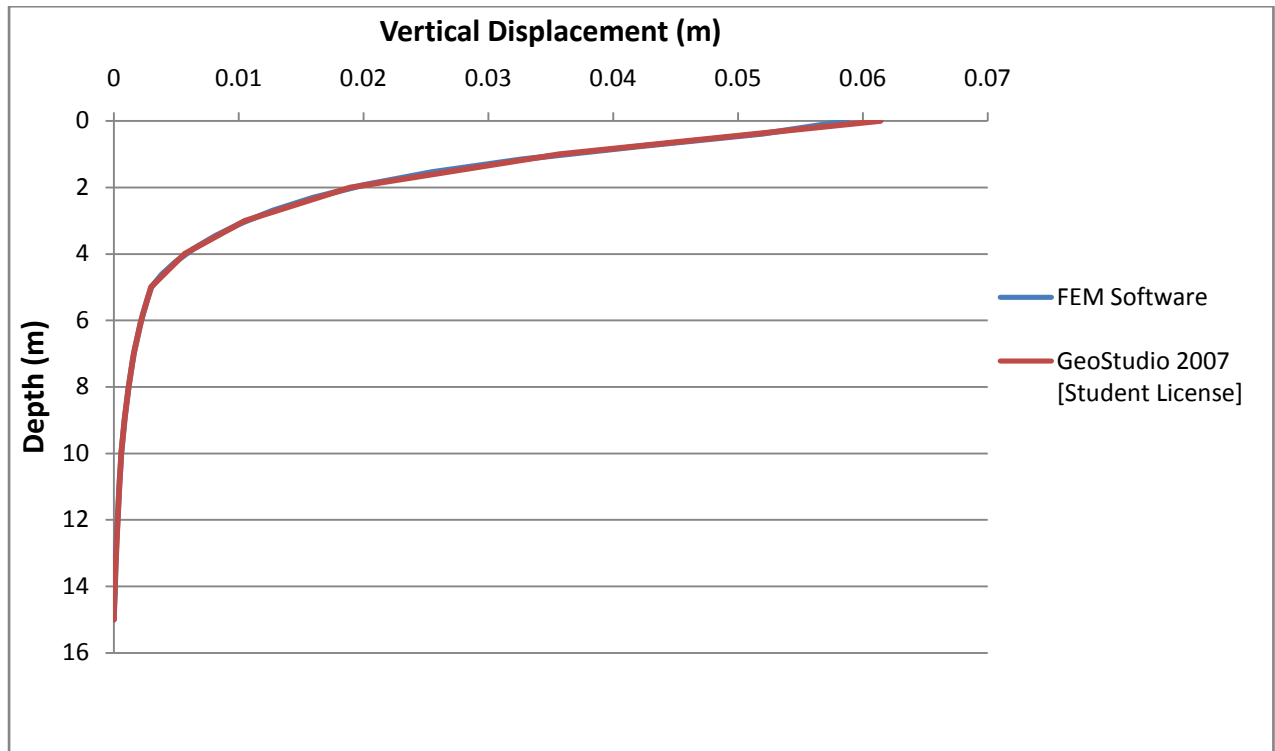
#### 4.2.3.4. Uniform Load of Circular Plan Area

Sample problem 16: Uniform Load of Circular Plan Area Acting on the Surface of a Triple Layered System

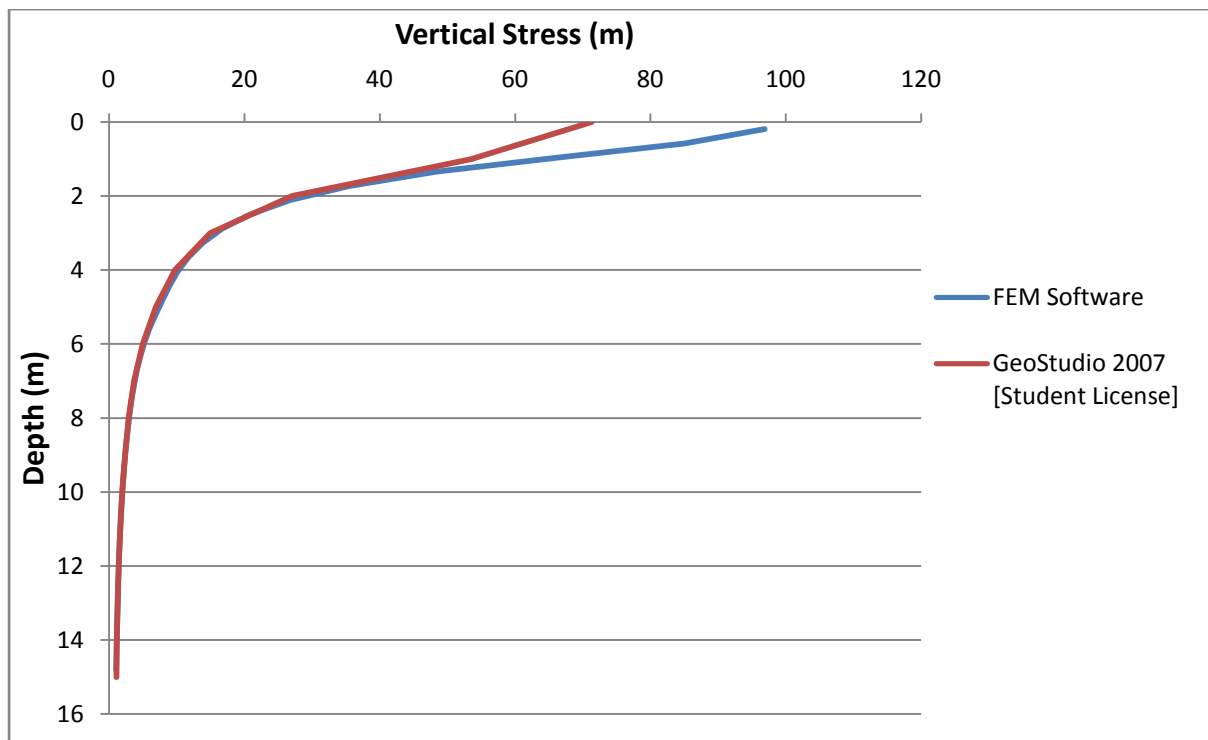


**Figure 4.76**

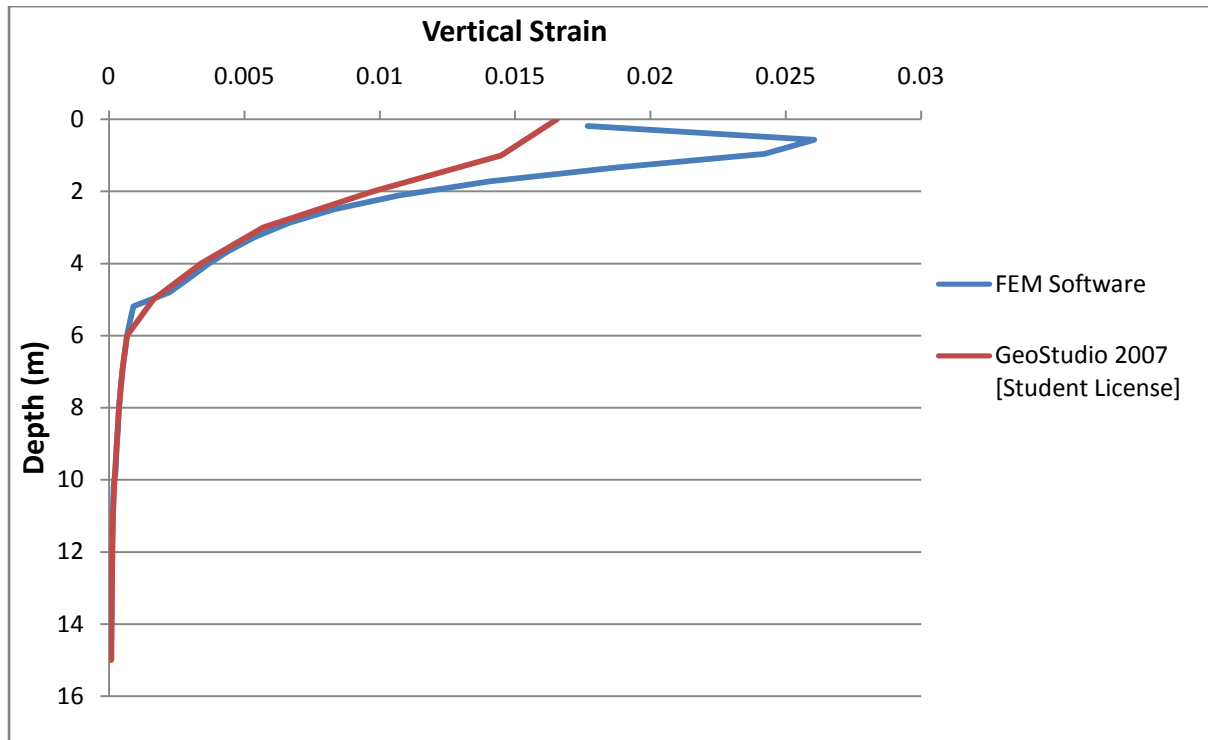
For the problem of a uniform load of circular plan area acting on a triple layered system, finite element analysis has been carried out using the software developed herewith and using the software GeoStudio 2007 (student license version). Except for the shear stress distribution, all other parameters are computed along the axis of loading. For the shear stress distribution, comparison of result is performed for an axis passing through the edge of circular loading.



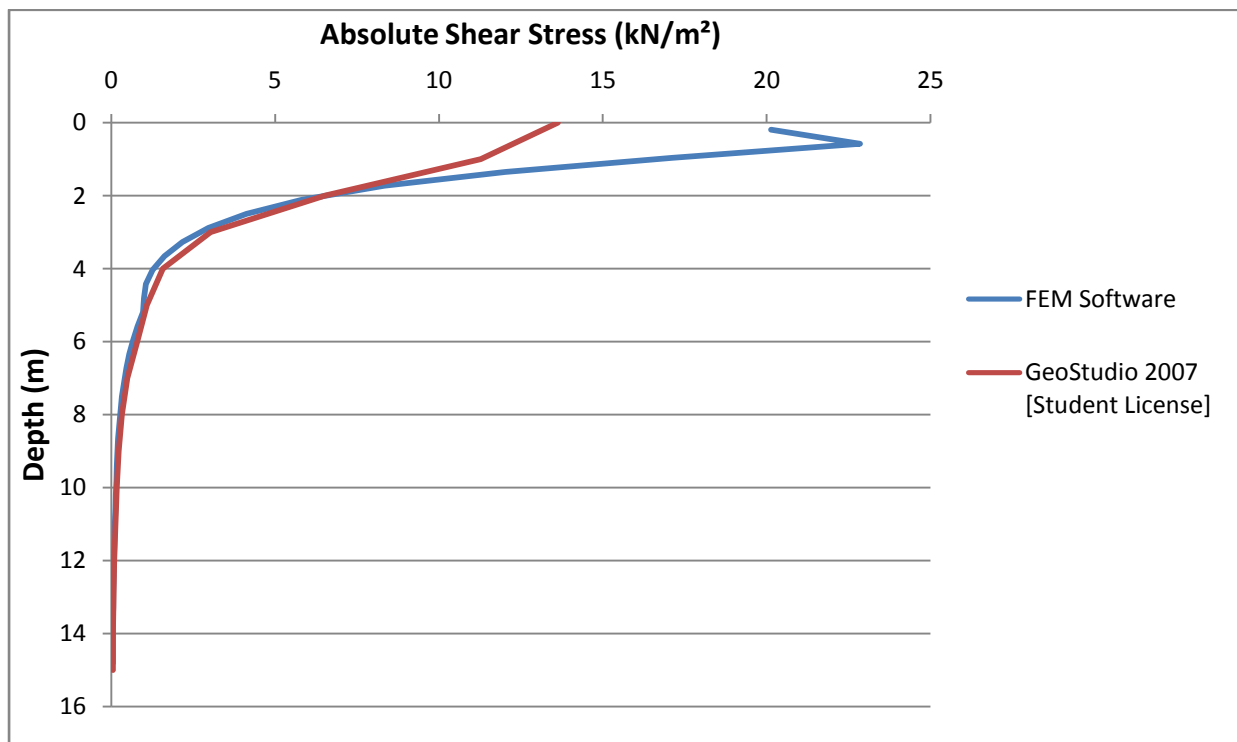
**Figure 4.77:** Distribution of Vertical Displacement along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Triple Layered System



**Figure 4.78:** Distribution of Vertical Stress along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Triple Layered System



**Figure 4.79:** Distribution of Vertical Strain along the Centerline of a Uniform Load of Circular Plan Area Acting on the Surface of a Triple Layered System



**Figure 4.80:** Distribution of Shear Stress along the Edge of a Uniform Load of Circular Plan Area Acting on the Surface of a Triple Layered System

## **5. CONCLUSION AND RECOMMENDATIONS**

### **5.1. Conclusion**

So far, the basic steps used in the development of this finite element software for computing stress and deformation have been presented along with comparison of the software's output against other software and available equations. Based on the observations made in due course of this research, the following conclusions are drawn.

1. Through utilization of the proper finite element procedures, it is possible to compute, to an acceptable degree of accuracy, stress and deformation values for a wide range of geotechnical problems.
2. The accuracy of a finite element solution is highly dependent on the number of elements used for analysis. Some of the output by the student license version of the software GeoStudio 2007 is certainly less accurate than that of the software developed herewith (please refer Figures 4.9, 4.33, 4.63, 4.64, and 4.68). This is due to the fact that the student license version of GeoStudio 2007 is limited to usage of 500 elements only. This has forced larger elements to be used for covering the model area and hence has resulted in less accurate results.
3. The Finite Element software developed in this research uses linear functions to interpolate displacement values. Using linear displacement functions simplifies the derivation of the finite element equations. However, the stress and strain values computed through such mechanism will be constant within an element. Hence, higher order interpolation functions are required to display variation of stress and strain within an element.
4. Usage of commercially available software is limited due to high cost involved in their procurement. Trial versions which are available for free or at lower costs are usually limited in processing capacity. Hence, the software developed herewith provides an opportunity to make use of the Finite Element technique at a much convenient and economical manner.

## **5.2. Recommendations**

The Finite Element Technique provides an opportunity to perform analysis over a wide range of geotechnical problems as well as in many other streams. The application developed herewith has taken an initial step in making use of available computational capacity and finite element concepts to enable analysis of stresses and deformations for two dimensional geotechnical problems. Either by upgrading this application or developing a similar application with additional features, it is possible to expand the application's use for almost all geotechnical problems. Among areas of possible expansion, the following are the major ones.

- Application of higher order interpolation functions to get a more detailed and accurate output;
- Analysis of three dimensional problems;
- Analysis of primary (consolidation) and creep settlements; and
- Various analyses of more complicated problems such as dynamic analysis.

## 6. REFERENCES

- [1] Punurai, W. - 2D Finite Elements  
Faculty of Engineering, Mahidol University  
[www.egmu.net/civil/wonsiri/fe6.pdf](http://www.egmu.net/civil/wonsiri/fe6.pdf)
- [2] Alemayehu Teferra and Edgar Schultz - Formulae, Charts and Tables Soil Mechanics and Foundation Engineering, Stresses in Soils  
A.A. Balkema, Rotterdam 1988
- [3] Zietsman, Christiaan Abraham - A Hierarchical Linear Elastic Boundary Element Solver for Lenticular Ore Bodies  
University of Stellenbosch 2007  
<http://scholar.sun.ac.za/handle/10019.1/1686>
- [4] O.C. Zienkiewicz, R.L. Taylor - The Finite Element Method; Fifth Edition; Volume 1: The Basis  
Butterworth-Heinemann 2000
- [5] Alemayehu Teferra - Lecture Notes, Advanced Soil Mechanics (2010)
- [6] Hadush Seged - Lecture Notes, Advanced Computational Methods in Geotechnical Engineering (2011)
- [7] H.G. Poulos E.H. Davis - Elastic Solutions for Soil and Rock Mechanics  
John Wiley & Sons Inc. 1974 USA
- [8] Michael Halvorson - Visual Basic 2010 Step by Step  
Microsoft Press, 2010
- [9] Tim Patrick - ADO.NET 4 Step by Step
- [10] Hadush Seged - Lecture Notes, Theory of Elasticity in Geotechnical Engineering (2009)

### **Software Used**

GeoStudio 2007 (Student License)

Visual Basic 2010

# ANNEXES



## ANNEX 1: Codes Used for the Development of the Software

Following is a presentation of the complete programming codes used for developing the software in the Visual Basic 2010 Application Development Environment.

Note:

1. Statements that begin with a double asterisk (\*\*) are descriptions of the code and not parts of the code.
2. Some of the lines within the codes have been wrapped to the next line to fit the width of the page in this presentation.

### \*\*Codes Used for Declaring Frequently Used Variables

```
Imports System.IO
Imports System.Data
Module Calculation_Module

    Public ScaledDeformationMatrix() As Single
    'Public StressMatrix(3) As Single
    Public WidthW, DepthD As Single
    Public LargestDeformation As Single

    Public LayerThicknesses(,) As Single
    Public PoissonsRatio(,) As Single
    Public ElasticModulus(,) As Single

    Public NodalCoordinates(,) As Single
    Public ScaledDeformedCoordinates(,) As Single
    Public ElementNodes(,) As UInteger
    Public ElementCentroids(,) As Single

    Public BMatrix3D(,,), BTranspose(,,) As Single
    Public DMatrix(,,) As Double

    Public ShortStiffness(,) As Double
    Public ShortStiffnessCopy(,) As Single
    Public Shortforces() As Double
    Public ShortforcesCopy() As Double
    Public GlobalStiffnessMatrix(,)

    Public GlobalDisplacements() As Double
    Public FixityStatus() As Byte
    Public UnknownDOFList() As UInteger
    Public DOF As UInteger

    Public TestFixity As String

    Public UDLMatrix(,), PointLoadMatrix(,) As Single
    Public LoadingDataEntered As Boolean = False

    Public VerticalGridMatrix(,), HorizontalGridMatrix() As Single
```

```
Public TempVGridMatrix(,), TempHGridMatrix() As Single
Public ForcesMatrix() As Single
Public NumberSoilLayers, NumberPointLoads, NumberUniformLoads As UInteger

Public VerticalStress(), HorizontalStress(), StressY() As Double
Public ShearStress(), AbsoluteShearStress() As Double

Public VerticalStrain(), HorizontalStrain(), StrainY() As Double
Public ShearStrain(), AbsoluteShearStrain() As Double

Public MaxVerticalStress, MaxHorizontalStress, MaxYstress, MaxShearStress As
Double
Public MaxVerticalStrain, MaxHorizontalStrain, MaxYStrain, MaxShearStrain As
Double

Public Reducer As Decimal

'Public MaxGraphicsWidth As UInteger = 500
'Public MaxGraphicsDepth As UInteger = 300

Public MaximumMeshDimension As Single
Public DefaultMeshOn As Boolean = True
'Public DefaultMeshDimension As Boolean = False
Public ElasticProperties As String

Public ProjectDataSet As New DataSet("FiniteElementSoftwareDataSet")
Public BasicProjectDataTable As New DataTable("BasicProjectDataTable")
Public SoilPropertiesTable As New DataTable("SoilPropertiesTable")
Public PointLoadTable As New DataTable("PointLoadTable")
Public UniformLoadTable As New DataTable("UniformLoadTable")
Public VerticalGridMatrixTable As New DataTable("VerticalGridMatrixTable")
Public HorizontalGridMatrixTable As New DataTable("HorizontalGridMatrixTable")
Public ElementNodesTable As New DataTable("ElementNodesTable")
Public NodalCoordinatesAndLoadsTable As New
DataTable("NodalCoordinatesAndLoadsTable")
Public FixityStatusTable As New DataTable("FixityStatusTable")
Public BasicProjectDataRow As DataRow

'Dim table As DataTable = New DataTable("ParentTable")

Public LeftEndVerticallyRestrained As Boolean = True
Public RightEndVerticallyRestrained As Boolean = True
Public PlaneStrainAnalysis As Boolean = True

Public OutputReportText As String
Public ReportStreamToWrite As StreamWriter

Public AnalysisBegin, AnalysisEnd, ReportBegin, ReportEnd As String

Public SaveFileDialog1 As New SaveFileDialog

Public DecimalTest As Decimal
Public IntegerTest As Integer

Public SoilPropetiesEntered As Boolean = False
Public LoadsApplied As Boolean = False
Public MeshGenerated As Boolean = False
Public AnalysisRun As Boolean = False
End Module
```

**\*\*Codes Used for Managing the Functions on the Home Screen (Data Entry, Mesh Generation and Analysis Execution)**

```
Imports System.Math
Imports System.IO
Imports System.Data
Imports System.Drawing.Printing

Public Class MainForm
    Dim MeshVisibility As Boolean = False
    Dim SoilDataEntered As Boolean = False

    Private PrintPageSettings As New PageSettings
    Dim memoryImage As Bitmap

    Dim NumberVerticalGrid As UInteger
    Dim NumberHorizontalGrid As UInteger
    Dim MeshGraphicPoints As Point()
    Dim GraphicsScale As Single
    Dim GraphicsWidth As UInteger
    Dim GraphicsDepth As UInteger
    Dim OriginX As UInteger = 300 + 100
    Dim OriginY As UInteger = 90 + 50
    Dim ElementalElasticModulus, ElementalPoissonsRatio, ElementArea As Single

    Dim BMatrix2D(2, 5) As Single

    Dim SimpleSaveEnabled As Boolean = False
    Dim SavedFileName As String

    Dim MaxGraphicsWidth As UInteger = 500
    Dim MaxGraphicsDepth As UInteger = 300

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        Dim i As Short
        Dim j As UShort

        Try

            If Me.Size.Width > 380 Then

                'adjusting graphics scale
                MaxGraphicsWidth = Me.Size.Width * (1 - 0.1) - 300 - 70
                MaxGraphicsDepth = Me.Size.Height * (1 - 0.2) - 75 - 50

                'setting the width & Depth
                Try
                    WidthW = BasicProjectDataRow.Item("Width")
                    DepthD = BasicProjectDataRow.Item("Depth")
                Catch ex As Exception
                    WidthW = 10
                    DepthD = 6
                End Try

                'Selecting Governing Scale from WidthScale & DepthScale
                If WidthW / DepthD > MaxGraphicsWidth / MaxGraphicsDepth Then
                    GraphicsScale = MaxGraphicsWidth / WidthW
                Else
                    GraphicsScale = MaxGraphicsDepth / DepthD
                End If
            End If
        End Try
    End Sub
End Class
```

```

End If
GraphicsDepth = DepthD * GraphicsScale
GraphicsWidth = WidthW * GraphicsScale
OriginX = 300 + 75 + 0.5 * (MaxGraphicsWidth - GraphicsWidth)
OriginY = 90 + 50 + 0.5 * (MaxGraphicsDepth - GraphicsDepth)

'Arranging Graphics for Soil Mesh
Dim counter As Integer = 0
Dim MeshDirection As Byte = 1
Dim MeshGraphics As Graphics
MeshGraphics = Me.CreateGraphics
Dim MeshPen As New Pen(Color.Gray)

If MeshVisibility = True Then

    'Drawing mesh step by step for every element
    'Dim Element As UInteger
    'For Element = 0 To UBound(ElementNodes)
'100 * NodalCoordinates(ElementNodes(Element, 0), 0), 100 *
NodalCoordinates(ElementNodes(Element, 0), 1)
    'MeshGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 1))
    'MeshGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 1))
    'MeshGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 1))
    'Next

    'Preparing Mesh Coordinates for drawlines command (avoids
redundant lines)
    MeshGraphicPoints = {New Point(OriginX, OriginY)} 'Preparing the
coordinates for multi line
    For j = 0 To UBound(VerticalGridMatrix, 2) - 1 'Loop along
the horizontal (X-direction)
        For i = 1 To UBound(HorizontalGridMatrix) 'Loop along
the vertical (Y-direction)
            counter = counter + 1
            ReDim Preserve MeshGraphicPoints(counter)
            MeshGraphicPoints(counter) = New Point(OriginX +
GraphicsScale * VerticalGridMatrix(0, j + MeshDirection), OriginY + GraphicsScale *
HorizontalGridMatrix(i))
            MeshDirection = Abs(MeshDirection - 1)
        Next
        For i = UBound(HorizontalGridMatrix) To 0 Step -1 'Y-
coordinates- upwards
            counter = counter + 1
            ReDim Preserve MeshGraphicPoints(counter)
            MeshGraphicPoints(counter) = New Point(OriginX +
GraphicsScale * VerticalGridMatrix(0, j + MeshDirection), OriginY + GraphicsScale *
HorizontalGridMatrix(i))
            MeshDirection = Abs(MeshDirection - 1)
        Next
        MeshDirection = Abs(MeshDirection - 1)
    Next

```

```
        Next
    End If

    'Soil graphics paint procedure
    Dim SoilGraphics As Graphics
    SoilGraphics = Me.CreateGraphics
    Dim SoilColor As New SolidBrush(Color.BurlyWood)
    SoilGraphics.FillRectangle(SoilColor, OriginX, OriginY, GraphicsWidth,
GraphicsDepth)

    If MeshVisibility = True Then
        'Drawing the mesh lines
        For j = 1 To UBound(VerticalGridMatrix, 2) - 1      'Draw vertical
grid lines
            MeshGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
VerticalGridMatrix(0, j), OriginY, OriginX + GraphicsScale * VerticalGridMatrix(0, j),
OriginY + GraphicsScale * DepthD)
        Next
        For i = 0 To UBound(HorizontalGridMatrix) - 1      'Draw
horizontal grid lines
            MeshGraphics.DrawLine(MeshPen, OriginX, OriginY +
GraphicsScale * HorizontalGridMatrix(i), OriginX + GraphicsScale * WidthW, OriginY +
GraphicsScale * HorizontalGridMatrix(i))
        Next
        MeshGraphics.DrawLines(MeshPen, MeshGraphicPoints) 'draws the non
orthogonal mesh lines
    End If

    'Graphics for Soil Boundary
    Dim SoilBoundaryPen As New Pen(Color.Black, 1)

    'Drawing Soil Matrix Boundary
    SoilGraphics.DrawLine(SoilBoundaryPen, OriginX, OriginY, OriginX,
OriginY + GraphicsDepth) 'Left
    SoilGraphics.DrawLine(SoilBoundaryPen, OriginX, OriginY +
GraphicsDepth, OriginX + GraphicsWidth, OriginY + GraphicsDepth) 'Bottom
    SoilGraphics.DrawLine(SoilBoundaryPen, OriginX + GraphicsWidth,
OriginY + GraphicsDepth, OriginX + GraphicsWidth, OriginY) 'Right

    Dim EndConditionPen As New Pen(Color.Black)
    'End conditions left
    SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) - 30, OriginX - 10, OriginY + CInt(GraphicsDepth / 3) + 30)
    SoilGraphics.DrawLine(EndConditionPen, OriginX + 10, OriginY +
CInt(GraphicsDepth / 3) - 30, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) + 30)
    If LeftEndVerticallyRestrained Then
        SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) - 10, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) + 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) + 10, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) - 10)
    End If

    SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 30, OriginX - 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 30)
    SoilGraphics.DrawLine(EndConditionPen, OriginX + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 30, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 30)
    If LeftEndVerticallyRestrained Then
        SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 10, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 10)
```

```
        SoilGraphics.DrawLine(EndConditionPen, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 10, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
- 10)
        End If

        'End Condition Bottom with crossing
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth /
3) - 40, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth / 3) + 40, OriginY
+ GraphicsDepth - 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth /
3) - 40, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth / 3) + 40, OriginY
+ GraphicsDepth + 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth /
3) - 20, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth / 3) + 20, OriginY
+ GraphicsDepth + 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth /
3) - 20, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth / 3) + 20, OriginY
+ GraphicsDepth - 10)

        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth *
2 / 3) - 40, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 40,
OriginY + GraphicsDepth - 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth *
2 / 3) - 40, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 40,
OriginY + GraphicsDepth + 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth *
2 / 3) - 20, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 20,
OriginY + GraphicsDepth + 10)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + CInt(GraphicsWidth *
2 / 3) - 20, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 20,
OriginY + GraphicsDepth - 10)

        'End Condition Right
        SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth / 3) - 30, OriginX + GraphicsWidth - 10, OriginY +
CInt(GraphicsDepth / 3) + 30)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth + 10,
OriginY + CInt(GraphicsDepth / 3) - 30, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) + 30)
        If RightEndVerticallyRestrained = True Then
            SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth -
10, OriginY + CInt(GraphicsDepth / 3) - 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) + 10)
            SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth -
10, OriginY + CInt(GraphicsDepth / 3) + 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) - 10)
        End If

        SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth * 2 / 3) - 30, OriginX + GraphicsWidth - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 30)
        SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth + 10,
OriginY + CInt(GraphicsDepth * 2 / 3) - 30, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 30)
        If RightEndVerticallyRestrained = True Then
            SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth -
10, OriginY + CInt(GraphicsDepth * 2 / 3) - 10, OriginX + GraphicsWidth + 10, OriginY
+ CInt(GraphicsDepth * 2 / 3) + 10)
            SoilGraphics.DrawLine(EndConditionPen, OriginX + GraphicsWidth -
10, OriginY + CInt(GraphicsDepth * 2 / 3) + 10, OriginX + GraphicsWidth + 10, OriginY
+ CInt(GraphicsDepth * 2 / 3) - 10)
```

```

End If

'Graphics for Layer Lines
Dim PenForLayers As New Pen(Color.Black, 2)
Dim CumulativeThickness As Single = 0
'The row LayerThickness(1,_) holds cumulative layer thicknesses
For i = 1 To NumberSoilLayers - 1
    SoilGraphics.DrawLine(PenForLayers, OriginX, OriginY +
GraphicsScale * LayerThicknesses(1, i - 1), OriginX + GraphicsWidth, OriginY +
GraphicsScale * LayerThicknesses(1, i - 1))
Next

'Drawign Load Arrows
If LoadingDataEntered = True Then
    Dim PenForUDL As New Pen(Color.Red, 2)
    'Dim BackgroundBlank As New SolidBrush(Color.White)

    If NumberUniformLoads <> 0 Then
        For j = 0 To NumberUniformLoads - 1
            If UDLMatrix(j, 0) <> 0 Then
                'SoilGraphics.FillRectangle(BackgroundBlank, OriginX +
GraphicsScale * UDLMatrix(j, 1) - 10, OriginY + GraphicsScale * UDLMatrix(j, 3) - 35,
GraphicsScale * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) + 20, 35)
                SoilGraphics.DrawLine(PenForUDL, OriginX +
GraphicsScale * UDLMatrix(j, 1), OriginY + GraphicsScale * UDLMatrix(j, 3) - 30,
OriginX + GraphicsScale * UDLMatrix(j, 2), OriginY + GraphicsScale * UDLMatrix(j, 3) -
30)

                For i = 0 To 5
                    SoilGraphics.DrawLine(PenForUDL, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5, OriginY + GraphicsScale * UDLMatrix(j, 3) - 30, OriginX + GraphicsScale *
UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5, OriginY
+ GraphicsScale * UDLMatrix(j, 3))
                    SoilGraphics.DrawLine(PenForUDL, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5 - 10, OriginY + GraphicsScale * UDLMatrix(j, 3) - 10, OriginX + GraphicsScale
* UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5,
OriginY + GraphicsScale * UDLMatrix(j, 3))
                    SoilGraphics.DrawLine(PenForUDL, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5 + 10, OriginY + GraphicsScale * UDLMatrix(j, 3) - 10, OriginX + GraphicsScale
* UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5,
OriginY + GraphicsScale * UDLMatrix(j, 3))
                Next
            End If
        Next
    End If

    Dim PointLoadsPen As New Pen(Color.Red, 2)
    If NumberPointLoads <> 0 Then
        For j = 0 To NumberPointLoads - 1 Step 1
            If PointLoadMatrix(j, 0) <> 0 Then
                SoilGraphics.DrawLine(PointLoadsPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1), OriginY + GraphicsScale * PointLoadMatrix(j, 2)
- 50, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY + GraphicsScale *
PointLoadMatrix(j, 2))
                SoilGraphics.DrawLine(PointLoadsPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1) - 10, OriginY + GraphicsScale *
PointLoadMatrix(j, 2) - 15, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY +
GraphicsScale * PointLoadMatrix(j, 2))
                SoilGraphics.DrawLine(PointLoadsPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1) + 10, OriginY + GraphicsScale *

```



```
PointLoadMatrix(j, 2) - 15, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY +
GraphicsScale * PointLoadMatrix(j, 2))
    End If
Next
End If
End If

'Drawing Coordinate Axes and Directional Arrows
Dim CoordinatesPen As New Pen(Color.Purple, 3)
Dim AxisPen As New Pen(Color.Gray, 1)
Dim font As New Font(Height, 14)
Dim OriginFont As New Font(Height, 10)
Dim brush As Brush = Brushes.Black
Dim format As New StringFormat(StringFormatFlags.NoClip)
Dim Centralformat As New StringFormat(StringFormatFlags.NoClip)
Dim RightAlignFormat As New StringFormat(LeftRightAlignment.Right)

    SoilGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 5,
OriginY, OriginX + GraphicsWidth + 40, OriginY)
    SoilGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 30,
OriginY - 5, OriginX + GraphicsWidth + 40, OriginY)
    SoilGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 30,
OriginY + 5, OriginX + GraphicsWidth + 40, OriginY)
    If PlaneStrainAnalysis Then
        SoilGraphics.DrawString("x", font, brush, OriginX + GraphicsWidth
+ 40, OriginY - 28, format)
    Else
        SoilGraphics.DrawString("r", font, brush, OriginX + GraphicsWidth
+ 40, OriginY - 28, format)
    End If

    SoilGraphics.DrawLine(CoordinatesPen, OriginX, OriginY + GraphicsDepth
+ 5, OriginX, OriginY + GraphicsDepth + 40)
    SoilGraphics.DrawLine(CoordinatesPen, OriginX - 5, OriginY +
GraphicsDepth + 30, OriginX, OriginY + GraphicsDepth + 40)
    SoilGraphics.DrawLine(CoordinatesPen, OriginX + 5, OriginY +
GraphicsDepth + 30, OriginX, OriginY + GraphicsDepth + 40)
    SoilGraphics.DrawString("z", font, brush, OriginX - 27, OriginY +
GraphicsDepth + 25, format)

    SoilGraphics.DrawString("( 0,0 )", OriginFont, brush, OriginX - 35,
OriginY - 40 + 25, format)

    SoilGraphics.DrawLine(AxisPen, OriginX - 75, OriginY - 75, OriginX +
GraphicsWidth, OriginY - 75)
    SoilGraphics.DrawLine(AxisPen, OriginX - 75, OriginY - 75, OriginX -
75, OriginY + GraphicsDepth)

    Dim Marker As Single
    i = 1
    Do While Marker <= WidthW
        SoilGraphics.DrawLine(AxisPen, OriginX + CInt(GraphicsScale *
Marker), OriginY - 80, OriginX + CInt(GraphicsScale * Marker), OriginY - 70)
        Marker = i * Max(WidthW, DepthD) / 10
        i += 1
    Loop
    Marker = 0
    i = 1
    Do While Marker <= DepthD
        SoilGraphics.DrawLine(AxisPen, OriginX - 80, OriginY +
CInt(GraphicsScale * Marker), OriginX - 70, OriginY + CInt(GraphicsScale * Marker))
        Marker = i * Max(WidthW, DepthD) / 10
```



```
        i += 1
    Loop
    Marker = 0
    Do While Marker <= WidthW
        SoilGraphics.DrawString(Marker, OriginFont, brush, OriginX +
GraphicsScale * Marker - 7, OriginY - 100, CentralFormat)
        Marker += (Max(DepthD, WidthW) / 5)
    Loop
    Marker = 0
    Do While Marker <= DepthD
        SoilGraphics.DrawString(Marker, OriginFont, brush, OriginX - 85,
OriginY + GraphicsScale * Marker - 10, RightAlignFormat)
        Marker += (Max(DepthD, WidthW) / 5)
    Loop

    End If
Catch ex As Exception

End Try

End Sub

Private Sub EnterSoilData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SoilDataButton.Click

    My.Forms.InputForm.ShowDialog()
    If InputForm.DialogResult = DialogResult.OK Then
        Refresh() 'without the refresh command, clicking of button doesnt erase
previous graphics
        LoadingDataButton.Enabled = True
        EnterLoadingDataToolStripMenuItem.Enabled = True
        MeshSizeToolStripMenuItem.Enabled = True
        SoilDataEntered = True
        BasicProjectDataRow.Item("ProgramStage") = 1
    End If

End Sub

Private Sub EnterLoadingData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LoadingDataButton.Click

    My.Forms.Loading_Data_Input.ShowDialog()
    If Loading_Data_Input.DialogResult = DialogResult.OK Then
        My.Forms.InputForm.ApplySoilDataButton.Enabled = False
        MeshVisibility = False
        GenerateMeshButton.Enabled = True
        RunAnalysisButton.Enabled = False
        RunAnalysisToolStripMenuueItem.Enabled = False
        GenerateMeshToolStripMenuItem.Enabled = True
        BasicProjectDataRow.Item("ProgramStage") = 2
    End If

End Sub

Private Sub GenerateMesh_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles GenerateMeshButton.Click

    Dim MinimumWidth As Single
    Dim MinimumThickness As Single

    Dim NumberFineGrid() As UInteger
    Dim TotalVerticalGrids As UInteger = 0
```

```
Dim Counter As UInteger = 1
Dim a, b, c, d, f As UInteger
Dim NumberHorizSubgrid As UInteger
Dim ThicknessHorizSubgrid As Single
Dim LayerCounter As UInteger

Dim i As UShort
Dim j As UShort
Dim k As Short

Dim GridUnavailableCheck As Boolean
Dim NumberVerticalGrid As UShort
Dim NumberHorizontalGrid As UShort

Dim SortRank(,) As UInteger
Dim SortCount As UInteger

If BasicProjectDataRow.Item("MeshDefaultOn") Then
    If WidthW > DepthD Then
        MaximumMeshDimension = WidthW / 40
    Else
        MaximumMeshDimension = DepthD / 40
    End If
Else
    MaximumMeshDimension = BasicProjectDataRow.Item("MaximumMeshSize")
End If

Try
    'Generating the Major Vertical & Horizontal Grids
    ReDim VerticalGridMatrix(1, 1 + 2 * NumberUniformLoads + NumberPointLoads)
    ReDim HorizontalGridMatrix(NumberSoilLayers + NumberPointLoads +
NumberUniformLoads)

    'Generating Horizontal Grid at Interfaces (further grid to be added on
Loading)
    For i = 1 To NumberSoilLayers
        HorizontalGridMatrix(i) = HorizontalGridMatrix(i - 1) +
LayerThicknesses(0, i - 1)
    Next
    'HorizontalGridMatrix(NumberSoilLayers) = DepthD
    'MsgBox(DepthD)
    NumberHorizontalGrid = NumberSoilLayers

    'Generating Major Vertical & Horizontal Grids(on critical points such as
soil edges & Loading Points)
    VerticalGridMatrix(0, 1) = WidthW
    NumberVerticalGrid = 1

    If NumberUniformLoads > 0 Then
        For k = 0 To NumberUniformLoads - 1
            '1. for every uniform load
            case
                For i = 1 To 2
                    'for the beginning & end locations
                    j = 0
                    'For every member of the
                    vertical grid(starting from 0)
                    GridUnavailableCheck = True
                    While GridUnavailableCheck = True And j <= NumberVerticalGrid
                        If UDLMatrix(k, i) = VerticalGridMatrix(0, j) Then
                            GridUnavailableCheck = False
                            'vertical grid has already been created on the
location
                        End If
                    End While
                Next i
            Next k
        End If
    End If
```

```

        j = j + 1
    End While
    If GridUnavailableCheck = True Then
        NumberVerticalGrid = NumberVerticalGrid + 1
        VerticalGridMatrix(0, NumberVerticalGrid) = UDLMatrix(k,
i)

    End If
Next

    j = 0                                     'For every member of the horizontal
grid(starting from 0)
    GridUnavailableCheck = True
    While GridUnavailableCheck = True And j <= NumberHorizontalGrid
        If UDLMatrix(k, 3) = HorizontalGridMatrix(j) Then
            GridUnavailableCheck = False 'vertical grid has already
been created on the location
        End If
        j = j + 1
    End While
    If GridUnavailableCheck = True Then
        NumberHorizontalGrid = NumberHorizontalGrid + 1
        HorizontalGridMatrix(NumberHorizontalGrid) = UDLMatrix(k, 3)

    End If
Next
End If

If NumberPointLoads > 0 Then
    For i = 0 To NumberPointLoads - 1 '2. For every point load
        j = 0                          'For every member of the vertical
grid(starting from 0)
        GridUnavailableCheck = True
        While GridUnavailableCheck = True And j <= NumberVerticalGrid
            If PointLoadMatrix(i, 1) = VerticalGridMatrix(0, j) Then
                GridUnavailableCheck = False
                VerticalGridMatrix(1, j) = VerticalGridMatrix(1, j) +
PointLoadMatrix(i, 0)
            End If
            j = j + 1
        End While
        If GridUnavailableCheck = True Then
            NumberVerticalGrid = NumberVerticalGrid + 1
            VerticalGridMatrix(0, NumberVerticalGrid) = PointLoadMatrix(i,
1)
            VerticalGridMatrix(1, j) = VerticalGridMatrix(1, j) +
PointLoadMatrix(i, 0)
        End If

        j = 0                                     'For every member of the
horizontal grid(starting from 0)
        GridUnavailableCheck = True
        While GridUnavailableCheck = True And j <= NumberHorizontalGrid

            If PointLoadMatrix(i, 2) = HorizontalGridMatrix(j) Then
                GridUnavailableCheck = False
                'VerticalGridMatrix(1, j) = VerticalGridMatrix(1, j) +
PointLoadMatrix(i, 0)
            End If
            j = j + 1
        End While

```

```
        If GridUnavailableCheck = True Then
            NumberHorizontalGrid = NumberHorizontalGrid + 1
            HorizontalGridMatrix(NumberHorizontalGrid) =
PointLoadMatrix(i, 2)
                'MsgBox("Point Load")
                'MsgBox(PointLoadMatrix(i, 2))
                'VerticalGridMatrix(1, j) = VerticalGridMatrix(1, j) +
PointLoadMatrix(i, 0)
        End If
    Next
End If

ReDim Preserve VerticalGridMatrix(1, NumberVerticalGrid)
ReDim Preserve HorizontalGridMatrix(NumberHorizontalGrid)

'Sorting the major vertical Grid locations in ascending order
ReDim SortRank(2, NumberVerticalGrid) 'row-0:rank, row-1:number of
subgrids
ReDim TempVGridMatrix(3, NumberVerticalGrid) 'row-0:location, row-
1:force magnitude, row-2:WIDTH, row-3:fine widths
For i = 0 To NumberVerticalGrid
    SortCount = 0
    For j = 0 To NumberVerticalGrid
        If VerticalGridMatrix(0, i) > VerticalGridMatrix(0, j) Or
(VVerticalGridMatrix(0, i) = VerticalGridMatrix(0, j) And i < j) Then
            SortCount = SortCount + 1
        End If
    Next
    SortRank(0, i) = SortCount
Next
For i = 0 To NumberVerticalGrid
    TempVGridMatrix(0, SortRank(0, i)) = VerticalGridMatrix(0, i)
    TempVGridMatrix(1, SortRank(0, i)) = VerticalGridMatrix(1, i)
Next

'Sorting the major horizontal Grid locations in ascending order
ReDim SortRank(1, NumberHorizontalGrid) 'row-0:rank, row-1:number of
subgrids
ReDim TempHGridMatrix(NumberHorizontalGrid)
For i = 0 To NumberHorizontalGrid
    SortCount = 0
    For j = 0 To NumberHorizontalGrid
        If HorizontalGridMatrix(i) > HorizontalGridMatrix(j) Or
(HorizontalGridMatrix(i) = HorizontalGridMatrix(j) And i < j) Then
            SortCount = SortCount + 1
        End If
    Next
    SortRank(0, i) = SortCount
Next
For i = 0 To NumberHorizontalGrid
    TempHGridMatrix(SortRank(0, i)) = HorizontalGridMatrix(i)
    'TempVGridMatrix(1, SortRank(0, i)) = VerticalGridMatrix(1, i)
Next

'My.Forms.Loading_Data_Input.Button1.Enabled = False
NumberVerticalGrid = UBound(TempVGridMatrix, 2)
NumberHorizontalGrid = UBound(TempHGridMatrix)
ReDim NumberFineGrid(NumberVerticalGrid - 1)
```

```

'Determining minimum width and Minimum Thickness To Determine Minimum Mesh
Dimension
MinimumWidth = WidthW
For i = 0 To NumberVerticalGrid - 1
    TempVGridMatrix(2, i) = TempVGridMatrix(0, i + 1) - TempVGridMatrix(0,
i)
    'MsgBox("widths" & TempVGridMatrix(2, i))
    If TempVGridMatrix(2, i) < MinimumWidth Then
        MinimumWidth = TempVGridMatrix(2, i)
    End If
Next

If MinimumWidth < MaximumMeshDimension Then
    MaximumMeshDimension = MinimumWidth
End If

MinimumThickness = DepthD
For i = 0 To NumberHorizontalGrid - 1
    If TempHGridMatrix(i + 1) - TempHGridMatrix(i) < MinimumThickness Then
        MinimumThickness = TempHGridMatrix(i + 1) - TempHGridMatrix(i)
    End If
Next

If MinimumThickness < MaximumMeshDimension Then
    MaximumMeshDimension = MinimumThickness
End If

'Determining number of fine grids for each segment (Vertical)
For i = 0 To NumberVerticalGrid - 1
    NumberFineGrid(i) = CInt(TempVGridMatrix(2, i) / MaximumMeshDimension)

    TempVGridMatrix(3, i) = TempVGridMatrix(2, i) / NumberFineGrid(i)
    TotalVerticalGrids = TotalVerticalGrids + NumberFineGrid(i)
Next
ReDim VerticalGridMatrix(0, TotalVerticalGrids)

'Generating the complete vetical grid data
VerticalGridMatrix(0, 0) = TempVGridMatrix(0, 0)      'setting the
coordinate of leftoutermost point
'ForcesMatrix(1) = TempVGridMatrix(1, 0)              'setting the y
component of the force on the leftoutermost point
For i = 0 To NumberVerticalGrid - 1
    For j = 1 To NumberFineGrid(i) - 1
        VerticalGridMatrix(0, Counter) = TempVGridMatrix(0, i) + j *
TempVGridMatrix(3, i)
        'MsgBox("Gridmatrix(" & Counter & ")= " & VerticalGridMatrix(0,
Counter))
        Counter = Counter + 1
    Next
    VerticalGridMatrix(0, Counter) = TempVGridMatrix(0, i + 1)
    'MsgBox("Gridmatrix(" & Counter & ")= " & VerticalGridMatrix(0,
Counter))
    'ForcesMatrix(2 * Counter + 1) = TempVGridMatrix(1, i + 1) (We dont
use row 1 of TempVGridMatrix to carry forces any more)
    Counter = Counter + 1
Next

'Generating the horizontal grid data
ReDim HorizontalGridMatrix(0)
HorizontalGridMatrix(0) = 0
Counter = 1
LayerCounter = 0

```

```

NumberHorizontalGrid = 0
For i = 0 To UBound(TempHGridMatrix) - 1
    NumberHorizSubgrid = CInt((TempHGridMatrix(i + 1) -
TempHGridMatrix(i)) / MaximumMeshDimension) 'number of subgrids
    ThicknessHorizSubgrid = (TempHGridMatrix(i + 1) - TempHGridMatrix(i))
/ NumberHorizSubgrid 'Thickness of Subgrids
    NumberHorizontalGrid = NumberHorizontalGrid + NumberHorizSubgrid
    ReDim Preserve HorizontalGridMatrix(NumberHorizontalGrid)
    For j = 1 To NumberHorizSubgrid - 1
        HorizontalGridMatrix(Counter) = TempHGridMatrix(i) + j *
ThicknessHorizSubgrid
        Counter = Counter + 1
    Next
    HorizontalGridMatrix(Counter) = TempHGridMatrix(i + 1)
    'MsgBox("H Grid Matrix(" & Counter & ")= " &
HorizontalGridMatrix(Counter))
    Counter = Counter + 1
    If HorizontalGridMatrix(Counter - 1) = LayerThicknesses(1,
LayerCounter) And LayerCounter <> NumberSoilLayers - 1 Then
        LayerCounter = LayerCounter + 1
        ElasticModulus(LayerCounter, 1) = 4 * NumberHorizontalGrid *
TotalVerticalGrids
        PoissonsRatio(LayerCounter, 1) = 4 * NumberHorizontalGrid *
TotalVerticalGrids
    End If
Next

ReDim ForcesMatrix((2 * NumberHorizontalGrid + 1) * (2 *
TotalVerticalGrids + 1))

'applying loads due to UDL on the respective nodes
Dim R1 As Single
Dim R2 As Single
If NumberUniformLoads <> 0 Then
    For i = 0 To UBound(UDLMatrix)
        For k = 0 To NumberHorizontalGrid
            If UDLMatrix(i, 3) = HorizontalGridMatrix(k) Then
                For j = 0 To TotalVerticalGrids - 1
                    If UDLMatrix(i, 1) <= VerticalGridMatrix(0, j) And
VerticalGridMatrix(0, j) < UDLMatrix(i, 2) Then
                        If PlaneStrainAnalysis Then
                            ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 1) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 1) +
UDLMatrix(i, 0) * (VerticalGridMatrix(0, j + 1) - VerticalGridMatrix(0, j)) / 2
                            ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 3) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 3) +
UDLMatrix(i, 0) * (VerticalGridMatrix(0, j + 1) - VerticalGridMatrix(0, j)) / 2
                        Else
                            R1 = VerticalGridMatrix(0, j)
                            R2 = VerticalGridMatrix(0, j + 1)
                            ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 1) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 1) + PI
/ 3 * UDLMatrix(i, 0) * (R2 - R1) * (2 * R1 + R2)
                            ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 3) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 3) + PI
/ 3 * UDLMatrix(i, 0) * (R2 - R1) * (R1 + 2 * R2)
                        End If
                    End If
                Next
                'MsgBox("Force Matrix" & ForcesMatrix(2 * j + 1) & ", " &
ForcesMatrix(2 * j + 3))
            End If
        Next
    Next

```

```

        Next
    Next
End If

'applying loads due to Point Loads on the respective nodes
If NumberPointLoads <> 0 Then
    For i = 0 To UBound(PointLoadMatrix, 1)
        For k = 0 To NumberHorizontalGrid
            If PointLoadMatrix(i, 2) = HorizontalGridMatrix(k) Then
                For j = 0 To TotalVerticalGrids
                    If PointLoadMatrix(i, 1) = VerticalGridMatrix(0, j)
Then
                        If PlaneStrainAnalysis Or PointLoadMatrix(i, 1) =
0 Then
                                ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 1) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 1) +
PointLoadMatrix(i, 0)
                                Else
                                        ForcesMatrix(2 * k * (2 * TotalVerticalGrids +
1) + 2 * j + 1) = ForcesMatrix(2 * k * (2 * TotalVerticalGrids + 1) + 2 * j + 1) +
PointLoadMatrix(i, 0) * 2 * PI * PointLoadMatrix(i, 1)
                                End If
                        End If
                    Next
                    'MsgBox("Force Matrix" & ForcesMatrix(2 * j + 1) & "," &
ForcesMatrix(2 * j + 3))
                End If
            Next
        Next
    End If

'Generating Nodal Coordinates
NumberVerticalGrid = UBound(VerticalGridMatrix, 2)
ReDim NodalCoordinates(NumberHorizontalGrid * NumberVerticalGrid + (1 +
NumberHorizontalGrid) * (1 + NumberVerticalGrid) - 1, 1) '****got to check this

Counter = 0
For i = 0 To NumberHorizontalGrid
    For j = 0 To NumberVerticalGrid
        NodalCoordinates(Counter, 0) = VerticalGridMatrix(0, j)
        NodalCoordinates(Counter, 1) = HorizontalGridMatrix(i)
        'MsgBox(Counter & "," & NodalCoordinates(Counter, 0) & "," &
NodalCoordinates(Counter, 1))
        Counter = Counter + 1
    Next

    For j = 0 To NumberVerticalGrid - 1
        If i <> NumberHorizontalGrid Then
            'MsgBox("counter= " & Counter)
            NodalCoordinates(Counter, 0) = 0.5 * (VerticalGridMatrix(0, j)
+ VerticalGridMatrix(0, j + 1))
            NodalCoordinates(Counter, 1) = 0.5 * (HorizontalGridMatrix(i)
+ HorizontalGridMatrix(i + 1))
            'MsgBox(Counter & "," & NodalCoordinates(Counter, 0) & "," &
NodalCoordinates(Counter, 1))
            Counter = Counter + 1
        End If
    Next
Next

```

```

'Generating the Element-Node Connectivity Matrix
ReDim ElementNodes(4 * NumberHorizontalGrid * NumberVerticalGrid - 1, 2)

Counter = 0
For i = 0 To NumberHorizontalGrid - 1
    For j = 0 To NumberVerticalGrid - 1

        a = i * (2 * NumberVerticalGrid + 1) + j
'Top Left Node
        b = i * (2 * NumberVerticalGrid + 1) + j + 1
'Top Right Node
        c = i * (2 * NumberVerticalGrid + 1) + j + 2 * NumberVerticalGrid
+ 2 'Bottom Right Node
        d = i * (2 * NumberVerticalGrid + 1) + j + 2 * NumberVerticalGrid
+ 1 'Bottom Left Node
        f = i * (2 * NumberVerticalGrid + 1) + j + NumberVerticalGrid + 1
'Central Node

        ElementNodes(Counter, 0) = a
        ElementNodes(Counter, 1) = b
        ElementNodes(Counter, 2) = f
        'MsgBox("a,b,f= " & a & ", " & b & ", " & f)

        ElementNodes(Counter + 1, 0) = a
        ElementNodes(Counter + 1, 1) = f
        ElementNodes(Counter + 1, 2) = d
        'MsgBox("b,f,c= " & b & ", " & f & ", " & c)

        ElementNodes(Counter + 2, 0) = f
        ElementNodes(Counter + 2, 1) = c
        ElementNodes(Counter + 2, 2) = d
        'MsgBox("f,d,c= " & f & ", " & d & ", " & c)

        ElementNodes(Counter + 3, 0) = b
        ElementNodes(Counter + 3, 1) = c
        ElementNodes(Counter + 3, 2) = f
        'MsgBox("a,f,d= " & a & ", " & f & ", " & d)

        Counter = Counter + 4
    Next
Next

ReDim FixityStatus(2 * UBound(NodalCoordinates) + 1)
'MsgBox("#fixity status=" & UBound(NodalCoordinates) + 1)

'Introducing vertical fix for Left and right side support
For i = 0 To 2 * NumberHorizontalGrid * (2 * NumberVerticalGrid + 1) + 1
Step 2 * (2 * NumberVerticalGrid + 1)
    FixityStatus(i) = 1
    FixityStatus(i + 2 * NumberVerticalGrid) = 1

    If LeftEndVerticallyRestrained = True Then
        FixityStatus(i + 1) = 1
    End If
    If RightEndVerticallyRestrained = True Then
        FixityStatus(i + 2 * NumberVerticalGrid + 1) = 1
    End If
Next

For i = 2 * NumberHorizontalGrid * (2 * NumberVerticalGrid + 1) + 1 To 2 *
UBound(NodalCoordinates) + 1 Step 2
    FixityStatus(i) = 1

```



```
Next

For i = 2 * NumberHorizontalGrid * (2 * NumberVerticalGrid + 1) + 2 To 2 *
UBound(NodalCoordinates) - 2 Step 2
    FixityStatus(i) = 1
Next

Catch ex As OutOfMemoryException
    MsgBox("System is out of memory because too many elements are beign
generated." & vbCrLf & _
        "Please check the input parameters and the Maximum Mesh Size
specified")
Exit Sub
Catch ex As Exception
    MsgBox("Mesh could not be generated, please check the input parameters")
Exit Sub
End Try

MeshVisibility = True
RunAnalysisButton.Enabled = True
RunAnalysisToolStripMenueItem.Enabled = True
BasicProjectDataRow("ProgramStage") = 3
Refresh()
End Sub

Private Sub RunAnalysis_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RunAnalysisButton.Click

    ReDim GlobalDisplacements(2 * UBound(NodalCoordinates) + 1)
    ReDim ScaledDeformedCoordinates(2 * UBound(NodalCoordinates) + 1, 2 *
UBound(NodalCoordinates) + 1)

    Try
        AnalysisBegin = TimeString

        Dim Element As UInteger
        Dim i, j As UShort
        'Dim k As Short

        Dim LocalDisplacements(5) As Double

        Dim RNodeNumber, CNodeNumber As UInteger
        Dim RDirectionNumber, CDirectionNumber As Byte
        Dim ElementStiffnessMatrix(5, 5) As Double
        ReDim GlobalStiffnessMatrix(UBound(FixityStatus), UBound(FixityStatus))
        Dim ArrayProduct(5, 3) As Single

        ReDim ElementCentroids(UBound(ElementNodes), 1)
        ReDim BMatrix3D(3, 5, UBound(ElementNodes))
        ReDim BTranspose(5, 3, UBound(ElementNodes))
        ReDim DMatrix(3, 3, UBound(ElementNodes))

        For Element = 0 To UBound(ElementNodes)
            ElementCentroids(Element, 0) = (NodalCoordinates(ElementNodes(Element,
0), 0) + NodalCoordinates(ElementNodes(Element, 1), 0) +
NodalCoordinates(ElementNodes(Element, 2), 0)) / 3
            ElementCentroids(Element, 1) = (NodalCoordinates(ElementNodes(Element,
0), 1) + NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 2), 1)) / 3

            'calculating B and D matrices(Element)
```

```
Dim AlphaI, AlphaJ, AlphaK As Single
Dim BetaI, BetaJ, BetaK As Single
Dim GammaI, GammaJ, GammaK As Single
Dim LayerNumber As UInteger

Dim l As Decimal

ElementArea = 0.5 * Abs((NodalCoordinates(ElementNodes(Element, 0), 0)
* NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 1), 0) * NodalCoordinates(ElementNodes(Element,
2), 1) + NodalCoordinates(ElementNodes(Element, 2), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1)) -
(NodalCoordinates(ElementNodes(Element, 1), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1) + NodalCoordinates(ElementNodes(Element,
2), 0) * NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 0), 0) * NodalCoordinates(ElementNodes(Element,
2), 1)))

ElementalElasticModulus = ElasticModulus(0, 0)
ElementalPoissonsRatio = PoissonsRatio(0, 0)
For i = 0 To NumberSoilLayers - 1
    If Element >= PoissonsRatio(i, 1) Then
        ElementalElasticModulus = ElasticModulus(i, 0)
        ElementalPoissonsRatio = PoissonsRatio(i, 0)
        LayerNumber = i
    End If
Next

l = ElementalElasticModulus / ((1 + ElementalPoissonsRatio) * (1 - 2 *
ElementalPoissonsRatio))

DMatrix(0, 0, Element) = l * (1 - ElementalPoissonsRatio)
DMatrix(0, 1, Element) = l * (ElementalPoissonsRatio)
DMatrix(0, 2, Element) = l * (ElementalPoissonsRatio)
DMatrix(1, 0, Element) = l * (ElementalPoissonsRatio)
DMatrix(1, 1, Element) = l * (1 - ElementalPoissonsRatio)
DMatrix(1, 2, Element) = l * (ElementalPoissonsRatio)
DMatrix(2, 0, Element) = l * (ElementalPoissonsRatio)
DMatrix(2, 1, Element) = l * (ElementalPoissonsRatio)
DMatrix(2, 2, Element) = l * (1 - ElementalPoissonsRatio)
DMatrix(3, 3, Element) = l * (0.5 - ElementalPoissonsRatio)

BetaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 1) -
NodalCoordinates(ElementNodes(Element, 2), 1))
BetaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 1) -
NodalCoordinates(ElementNodes(Element, 0), 1))
BetaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 1) -
NodalCoordinates(ElementNodes(Element, 1), 1))

GammaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 0) -
NodalCoordinates(ElementNodes(Element, 1), 0))
GammaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 0) -
NodalCoordinates(ElementNodes(Element, 2), 0))
GammaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 0) -
NodalCoordinates(ElementNodes(Element, 0), 0))
```

```
        BMatrix2D = {{BetaI, 0, BetaJ, 0, BetaK, 0}, {0, GammaI, 0, GammaJ, 0,
GammaK}, {0, 0, 0, 0, 0, 0}, {GammaI, BetaI, GammaJ, BetaJ, GammaK, BetaK}}
        If PlaneStrainAnalysis = False Then
            AlphaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 0) *
NodalCoordinates(ElementNodes(Element, 2), 1) - NodalCoordinates(ElementNodes(Element,
2), 0) * NodalCoordinates(ElementNodes(Element, 1), 1))
            AlphaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1) - NodalCoordinates(ElementNodes(Element,
0), 0) * NodalCoordinates(ElementNodes(Element, 2), 1))
            AlphaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 0) *
NodalCoordinates(ElementNodes(Element, 1), 1) - NodalCoordinates(ElementNodes(Element,
1), 0) * NodalCoordinates(ElementNodes(Element, 0), 1))
            BMatrix2D(2, 0) = AlphaI / ElementCentroids(Element, 0) + BetaI +
GammaI * ElementCentroids(Element, 1) / ElementCentroids(Element, 0)
            BMatrix2D(2, 2) = AlphaJ / ElementCentroids(Element, 0) + BetaJ +
GammaJ * ElementCentroids(Element, 1) / ElementCentroids(Element, 0)
            BMatrix2D(2, 4) = AlphaK / ElementCentroids(Element, 0) + BetaK +
GammaK * ElementCentroids(Element, 1) / ElementCentroids(Element, 0)
        End If

        For i = 0 To 3
            For j = 0 To 5
                BMatrix3D(i, j, Element) = BMatrix2D(i, j)
                BTranspose(j, i, Element) = BMatrix2D(i, j)
            Next
        Next

        'End of B and D calculation

        'calculating [B transpose]*[D]
        For i = 0 To 5
            For j = 0 To 3
                ArrayProduct(i, j) = BTranspose(i, 0, Element) * DMatrix(0, j,
Element) + BTranspose(i, 1, Element) * DMatrix(1, j, Element) + BTranspose(i, 2,
Element) * DMatrix(2, j, Element) + BTranspose(i, 3, Element) * DMatrix(3, j, Element)
            Next
        Next

        'Calculation of Elemental Stiffness [k]
        For i = 0 To 5
            For j = 0 To 5
                ElementStiffnessMatrix(i, j) = ElementArea *
(((ArrayProduct(i, 0)) * (BMatrix3D(0, j, Element))) + ((ArrayProduct(i, 1)) *
(BMatrix3D(1, j, Element))) + ((ArrayProduct(i, 2)) * (BMatrix3D(2, j, Element))) +
((ArrayProduct(i, 3)) * (BMatrix3D(3, j, Element))))
                If PlaneStrainAnalysis = False Then
                    ElementStiffnessMatrix(i, j) = ElementStiffnessMatrix(i,
j) * 2 * PI * ElementCentroids(Element, 0)
                End If

                RNodeNumber = i \ 2
                RDirectionNumber = i Mod 2
                CNodeNumber = j \ 2
                CDirectionNumber = j Mod 2

                GlobalStiffnessMatrix(2 * ElementNodes(Element, RNodeNumber) +
RDirectionNumber, 2 * ElementNodes(Element, CNodeNumber) + CDirectionNumber) =
```

```
GlobalStiffnessMatrix(2 * ElementNodes(Element, RNodeNumber) + RDirectionNumber, 2 *
ElementNodes(Element, CNodeNumber) + CDirectionNumber) + ElementStiffnessMatrix(i, j)
    Next
Next
Next

ReDim UnknownDOFList(UBound(FixityStatus))

DOF = 0
For i = 0 To UBound(FixityStatus)
    If FixityStatus(i) = 0 Then
        DOF = DOF + 1
    End If
Next

ReDim UnknownDOFList(DOF - 1)
ReDim ShortStiffness(DOF - 1, DOF - 1)
ReDim ShortStiffnessCopy(DOF - 1, DOF - 1)

DOF = 0
For i = 0 To UBound(FixityStatus)
    If FixityStatus(i) = 0 Then
        DOF = DOF + 1
        UnknownDOFList(DOF - 1) = i
    End If
Next

'Preparing the partitioned forces and stiffness matrices
ReDim Shortforces(DOF - 1)
ReDim ShortforcesCopy(DOF - 1)

For i = 0 To DOF - 1
    For j = 0 To DOF - 1
        ShortStiffness(i, j) = GlobalStiffnessMatrix(UnknownDOFList(i),
UnknownDOFList(j))
        ShortStiffnessCopy(i, j) = ShortStiffness(i, j)
    Next
    Shortforces(i) = ForcesMatrix(UnknownDOFList(i))
Next

'Solving the Matrix Equation
My.Forms.Gauss_Elimination.ShowDialog()

If Gauss_Elimination.DialogResult = DialogResult.OK Then
    'calculation of elemental Strain and stresses
    ReDim HorizontalStrain(UBound(ElementNodes))
    ReDim VerticalStrain(UBound(ElementNodes))
    ReDim StrainY(UBound(ElementNodes))
    ReDim ShearStrain(UBound(ElementNodes))
    ReDim AbsoluteShearStrain(UBound(ElementNodes))

    ReDim HorizontalStress(UBound(ElementNodes))
    ReDim VerticalStress(UBound(ElementNodes))
    ReDim StressY(UBound(ElementNodes))
    ReDim ShearStress(UBound(ElementNodes))
    ReDim AbsoluteShearStress(UBound(ElementNodes))

    MaxVerticalStrain = 0
    MaxHorizontalStrain = 0
    MaxShearStrain = 0
    MaxVerticalStress = 0
    MaxHorizontalStress = 0

```

```
MaxShearStress = 0

For Element = 0 To UBound(ElementNodes)
    For i = 0 To 2
        For j = 0 To 1
            direction
                LocalDisplacements(2 * i + j) = GlobalDisplacements(2 *
ElementNodes(Element, i) + j)
            Next
        Next

        'calculateBandDmatrices(Element)

        ' strain=[B]{d}
        For j = 0 To 5
            HorizontalStrain(Element) = HorizontalStrain(Element) -
BMatrix3D(0, j, Element) * LocalDisplacements(j)
            VerticalStrain(Element) = VerticalStrain(Element) -
BMatrix3D(1, j, Element) * LocalDisplacements(j)
            StrainY(Element) = StrainY(Element) - BMatrix3D(2, j, Element)
* LocalDisplacements(j)
            ShearStrain(Element) = ShearStrain(Element) - BMatrix3D(3, j,
Element) * LocalDisplacements(j)
            AbsoluteShearStrain(Element) = Abs(ShearStrain(Element))
        Next

        'Determining the Maximum horizontal and vertical Strains (for
Graphing purposes later)
        If VerticalStrain(Element) > MaxVerticalStrain Then
            MaxVerticalStrain = VerticalStrain(Element)
        End If
        If HorizontalStrain(Element) > MaxHorizontalStrain Then
            MaxHorizontalStrain = HorizontalStrain(Element)
        End If
        If StrainY(Element) > MaxYStrain Then
            MaxYStrain = StrainY(Element)
        End If
        If AbsoluteShearStrain(Element) > MaxShearStrain Then
            MaxShearStrain = AbsoluteShearStrain(Element)
        End If

        ' Stress=[D]{Strain}
        HorizontalStress(Element) = DMatrix(0, 0, Element) *
HorizontalStrain(Element) + DMatrix(0, 1, Element) * VerticalStrain(Element) +
DMatrix(0, 2, Element) * StrainY(Element)
        VerticalStress(Element) = DMatrix(1, 0, Element) *
HorizontalStrain(Element) + DMatrix(1, 1, Element) * VerticalStrain(Element) +
DMatrix(1, 2, Element) * StrainY(Element)
        StressY(Element) = DMatrix(2, 0, Element) *
HorizontalStrain(Element) + DMatrix(2, 1, Element) * VerticalStrain(Element) +
DMatrix(2, 2, Element) * StrainY(Element)
        ShearStress(Element) = DMatrix(3, 3, Element) *
ShearStrain(Element)
        AbsoluteShearStress(Element) = Abs(ShearStress(Element))

        If HorizontalStress(Element) > MaxHorizontalStress Then
            MaxHorizontalStress = HorizontalStress(Element)
        End If
        If VerticalStress(Element) > MaxVerticalStress Then
            MaxVerticalStress = VerticalStress(Element)
        End If
        If StressY(Element) > MaxYstress Then
```

```

        MaxYstress = StressY(Element)
    End If
    If AbsoluteShearStress(Element) > MaxShearStress Then
        MaxShearStress = AbsoluteShearStress(Element)
    End If
Next

LargestDeformation = GlobalDisplacements(0)
For i = 1 To 2 * UBound(NodalCoordinates) + 1 Step 2
    If (Abs(GlobalDisplacements(i))) > Abs(LargestDeformation) Then
        LargestDeformation = GlobalDisplacements(i)
    End If
Next

'copying displacement values for Graphing purposes later
ReDim ScaledDeformationMatrix(2 * UBound(NodalCoordinates) + 1)
For i = 0 To UBound(ScaledDeformationMatrix)
    ScaledDeformationMatrix(i) = 0.15 * WidthW / 5 *
GlobalDisplacements(i) / Abs(LargestDeformation)
    RNodeNumber = i \ 2
    RDirectionNumber = i Mod 2
    ScaledDeformedCoordinates(RNodeNumber, RDirectionNumber) =
NodalCoordinates(RNodeNumber, RDirectionNumber) + ScaledDeformationMatrix(i)
Next
AnalysisEnd = TimeString

My.Forms.Output.Show()
End If
Catch
    MsgBox("Analysis could not be completed, please check the input
parameters.")
Exit Sub
End Try
End Sub

Private Sub MeshSizeToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MeshSizeToolStripMenuItem.Click
    My.Forms.Max_Mesh_Dim.Show()
End Sub

Private Sub ManageBoundaryConditionsToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ManageBoundaryConditionsToolStripMenuItem.Click
    My.Forms.EndConditionForm.ShowDialog()
    'If EndConditionForm.DialogResult = DialogResult.OK Then
    Refresh() 'without the refresh command, clicking of button doesnt erase
previous graphics
    'End If
End Sub

Private Sub EnterSoilDataToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles EnterSoilDataToolStripMenuItem.Click
    SoilDataButton.PerformClick()
End Sub

Private Sub EnterLoadingDataToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles EnterLoadingDataToolStripMenuItem.Click
    LoadingDataButton.PerformClick()
End Sub

Private Sub GenerateMeshToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GenerateMeshToolStripMenuItem.Click

```

```
GenerateMeshButton.PerformClick()
End Sub

Private Sub RunAnalysisToolStripMenuItemClick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RunAnalysisToolStripMenuItemClick
RunAnalysisButton.PerformClick()
End Sub

Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CloseToolStripMenuItem.Click
Close()
End Sub

Private Sub AboutToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AboutToolStripMenuItem.Click
MsgBox("Finite Element Software for Computing Stresses and Deformations in
Layered Soils" & vbCrLf & vbCrLf & _
"A Thesis Submitted to the School of Graduate Studies in Partial
Fullfillment of the" & vbCrLf & _
"Requirement for Degree of Master of Science in Geotechnical
Engineering" & vbCrLf & _
vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf & _
"hiruyd@yahoo.com" & vbCrLf & _
vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbCrLf & "July 2014")
End Sub

Private Sub RestartToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles RestartToolStripMenuItem.Click
Application.Restart()
End Sub

Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

Try
BasicProjectDataTable.Columns.Add("ProgramStage", GetType(Byte))
BasicProjectDataTable.Columns.Add("PlaneStrainOn", GetType(Boolean))
BasicProjectDataTable.Columns.Add("LeftEndVerticallyRestrained",
GetType(Boolean))
BasicProjectDataTable.Columns.Add("RightEndVerticallyRestrained",
GetType(Boolean))
BasicProjectDataTable.Columns.Add("MeshDefaultOn", GetType(Boolean))

BasicProjectDataTable.Columns.Add("Width", GetType(Single))
BasicProjectDataTable.Columns.Add("Depth", GetType(Single))
BasicProjectDataTable.Columns.Add("numberOfLayers", GetType(UInteger))
BasicProjectDataTable.Columns.Add("MaximumMeshSize", GetType(Single))

BasicProjectDataTable.Columns.Add("NumberPointLoads", GetType(UInteger))
BasicProjectDataTable.Columns.Add("NumberUniformLoads", GetType(UInteger))

BasicProjectDataTable.Rows.Add({1})
BasicProjectDataRow = BasicProjectDataTable.Rows(0)

'The initial(default) conditions
BasicProjectDataRow.Item("ProgramStage") = 0
BasicProjectDataRow.Item("PlaneStrainOn") = True
BasicProjectDataRow.Item("LeftEndVerticallyRestrained") = True
BasicProjectDataRow.Item("RightEndVerticallyRestrained") = True
BasicProjectDataRow.Item("MeshDefaultOn") = True
```

```
        Catch ex As Exception

    End Try
End Sub

Private Sub SaveAsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaveAsToolStripMenuItem.Click

    SaveFileDialog1.Filter = "xml files (*.xml)|*.xml"

    If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
        SavedFileName = SaveFileDialog1.FileName
        Try
            FileSaveProcedure()
        Catch ex As Exception
            MsgBox("Error! The file could not be saved." & vbCrLf & ex.Message)
        End Try
        SimpleSaveEnabled = True
    End If

End Sub

Private Sub OpenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles OpenToolStripMenuItem.Click
    Dim ScanRow As DataRow
    Dim i As UShort

    OpenFileDialog1.Filter = "xml files (*.xml)|*.xml"
    OpenFileDialog1.ShowDialog()

    If OpenFileDialog1.FileName <> "" Then
        SavedFileName = OpenFileDialog1.FileName
        Try
            ProjectDataSet = New DataSet
            ProjectDataSet.ReadXml(SavedFileName)

            BasicProjectDataTable =
ProjectDataSet.Tables("BasicProjectDataTable").Copy
            BasicProjectDataRow = BasicProjectDataTable.Rows(0)

            'Converting database into program variables

            MeshVisibility = False

            ''Loading the problem type (Plane Strain/Axisymmetric)
            If BasicProjectDataRow.Item("PlaneStrainOn") Then
                PlaneStrainAnalysis = True
            Else
                PlaneStrainAnalysis = False
            End If

            ''Loading Left & Right vertical restraint conditions
            If BasicProjectDataRow.Item("LeftEndVerticallyRestrained") Then
                LeftEndVerticallyRestrained = True
            Else
                LeftEndVerticallyRestrained = False
            End If
            If BasicProjectDataRow.Item("RightEndVerticallyRestrained") Then
                RightEndVerticallyRestrained = True
            Else
                RightEndVerticallyRestrained = False
            End If
        End Try
    End If
End Sub
```



```
''Loading mesh conditions
If BasicProjectDataRow.Item("MeshDefaultOn") Then
    DefaultMeshOn = True
Else
    DefaultMeshOn = False
    MaximumMeshDimension = BasicProjectDataRow.Item("MaximumMeshSize")
End If

'MsgBox("Basic data entered")
If BasicProjectDataRow.Item("ProgramStage") = 0 Then
    BasicProjectDataTable.Columns.Add("Width", GetType(Single))
    BasicProjectDataTable.Columns.Add("Depth", GetType(Single))
    BasicProjectDataTable.Columns.Add("numberOfLayers",
GetType(UInteger))
    BasicProjectDataTable.Columns.Add("MaximumMeshSize",
GetType(Single))
    BasicProjectDataTable.Columns.Add("NumberPointLoads",
GetType(UInteger))
    BasicProjectDataTable.Columns.Add("NumberUniformLoads",
GetType(UInteger))
    Exit Try
End If

''Loading the soil properties
SoilPropertiesTable = New DataTable
SoilPropertiesTable =
ProjectDataSet.Tables("SoilPropertiesTable").Copy

WidthW = BasicProjectDataRow.Item("width")
DepthD = BasicProjectDataRow.Item("depth")
NumberSoilLayers = BasicProjectDataRow.Item("numberOfLayers")

ReDim ElasticModulus(NumberSoilLayers - 1, 1)
ReDim PoissonsRatio(NumberSoilLayers - 1, 1)
ReDim LayerThicknesses(1, NumberSoilLayers - 1)
Dim CumulativeThickness As Single

i = 0
For Each ScanRow In SoilPropertiesTable.Rows
    'MsgBox(i)
    LayerThicknesses(0, i) = ScanRow.Item(1)
    CumulativeThickness += LayerThicknesses(0, i)
    LayerThicknesses(1, i) = CumulativeThickness

    ElasticModulus(i, 0) = ScanRow.Item(2)
    PoissonsRatio(i, 0) = ScanRow.Item(3)
    i = i + 1
Next

LoadingDataButton.Enabled = True
EnterLoadingDataToolStripMenuItem.Enabled = True
MeshSizeToolStripMenuItem.Enabled = True
SoilDataEntered = True

'MsgBox("Soil Properties Entered")
If BasicProjectDataRow.Item("ProgramStage") = 1 Then
    BasicProjectDataTable.Columns.Add("NumberPointLoads",
GetType(UInteger))
    BasicProjectDataTable.Columns.Add("NumberUniformLoads",
GetType(UInteger))
    Exit Try
```

```
End If

'Loading Point & Uniform Load Datas
NumberPointLoads = BasicProjectDataRow.Item("NumberPointLoads")
NumberUniformLoads = BasicProjectDataRow.Item("NumberUniformLoads")

PointLoadTable = New DataTable
UniformLoadTable = New DataTable
If NumberPointLoads <> 0 Then PointLoadTable =
ProjectDataSet.Tables("PointLoadTable").Copy
If NumberUniformLoads <> 0 Then UniformLoadTable =
ProjectDataSet.Tables("UniformLoadTable").Copy

ReDim PointLoadMatrix(NumberPointLoads - 1, 2) 'Column-0 =magnitude,
Column-1 =location, Column-2 =depth
ReDim UDLMatrix(NumberUniformLoads - 1, 3) 'Column-0 =magnitude,
Column-1 =beginning, column-2 =end, Column-3 =depth
LoadingDataEntered = True

i = 0
For Each ScanRow In PointLoadTable.Rows()
    PointLoadMatrix(i, 0) = ScanRow!Magnitude
    PointLoadMatrix(i, 1) = ScanRow!Location
    PointLoadMatrix(i, 2) = ScanRow!Depth
    i = i + 1
Next

i = 0
For Each ScanRow In UniformLoadTable.Rows()
    UDLMatrix(i, 0) = ScanRow!Magnitude
    UDLMatrix(i, 1) = ScanRow!Beginning
    UDLMatrix(i, 2) = ScanRow!End
    UDLMatrix(i, 3) = ScanRow!Depth
    i = i + 1
Next

LoadingDataEntered = True

My.Forms.InputForm.ApplySoilDataButton.Enabled = False
GenerateMeshButton.Enabled = True
RunAnalysisButton.Enabled = False
RunAnalysisToolStripMenueItem.Enabled = False
GenerateMeshToolStripMenuItem.Enabled = True
'MsgBox("Loading Data Entered")
If BasicProjectDataRow.Item("ProgramStage") = 2 Then
    Exit Try
End If

VerticalGridMatrixTable = New DataTable
VerticalGridMatrixTable =
ProjectDataSet.Tables("VerticalGridMatrixTable").Copy
i = 0
ReDim VerticalGridMatrix(0,
BasicProjectDataRow.Item("NumberVerticalGrid") - 1)
For Each ScanRow In VerticalGridMatrixTable.Rows
    VerticalGridMatrix(0, i) = ScanRow.Item(0)
    i += 1
Next

HorizontalGridMatrixTable = New DataTable
HorizontalGridMatrixTable =
ProjectDataSet.Tables("HorizontalGridMatrixTable").Copy
```

```
        i = 0
        ReDim
HorizontalGridMatrix(BasicProjectDataRow.Item("NumberHorizontalGrid") - 1)
        For Each ScanRow In HorizontalGridMatrixTable.Rows
            HorizontalGridMatrix(i) = ScanRow.Item(0)
            i += 1
        Next

        ElementNodesTable = New DataTable
        ElementNodesTable = ProjectDataSet.Tables("ElementNodesTable").Copy
        i = 0
        ReDim ElementNodes(BasicProjectDataRow.Item("numberOFelements") - 1,
2)

        For Each ScanRow In ElementNodesTable.Rows
            ElementNodes(i, 0) = ScanRow.Item(0)
            ElementNodes(i, 1) = ScanRow.Item(1)
            ElementNodes(i, 2) = ScanRow.Item(2)
            i += 1
        Next

        NodalCoordinatesAndLoadsTable = New DataTable
        NodalCoordinatesAndLoadsTable =
ProjectDataSet.Tables("NodalCoordinatesAndLoadsTable").Copy
        i = 0
        ReDim NodalCoordinates(BasicProjectDataRow.Item("numberOFnodes") - 1,
1)

        ReDim ForcesMatrix(2 * BasicProjectDataRow.Item("numberOFnodes") - 1)
        For Each ScanRow In NodalCoordinatesAndLoadsTable.Rows
            NodalCoordinates(i, 0) = ScanRow.Item(0)
            NodalCoordinates(i, 1) = ScanRow.Item(1)
            ForcesMatrix(2 * i + 1) = ScanRow.Item(2)
            i += 1
        Next

        FixityStatusTable = New DataTable
        FixityStatusTable = ProjectDataSet.Tables("FixityStatusTable").Copy
        i = 0
        ReDim FixityStatus(BasicProjectDataRow.Item("DegreesOfFreedom") - 1)
        For Each ScanRow In FixityStatusTable.Rows
            FixityStatus(i) = ScanRow.Item(0)
            i += 1
        Next
        MeshVisibility = True
        RunAnalysisButton.Enabled = True
        RunAnalysisToolStripMenuueItem.Enabled = True

        Catch ex As Exception
            MsgBox("Error Opening File!" & vbCrLf & ex.Message)
        End Try
        Refresh()
        SavedFileName = OpenFileDialog1.FileName
        SimpleSaveEnabled = True
    End If
End Sub

Private Sub PrintToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles PrintToolStripMenuItem.Click
    Try
        CaptureScreen()
        PrintDocument1.Print()
    Catch ex As Exception
        'Display error message
    End Try
```

```
        MessageBox.Show(ex.Message)
    End Try
End Sub

Private Sub CaptureScreen()

    PrintPageSettings.Landscape = True

    Dim myGraphics As Graphics = Me.CreateGraphics()
    Dim s As Size = Me.Size
    s = New Size(120 + GraphicsWidth + 65, 85 + GraphicsDepth + 85)
    memoryImage = New Bitmap(s.Width, s.Height, myGraphics)
    Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
    memoryGraphics.CopyFromScreen(Me.Location.X + OriginX - 120, Me.Location.Y +
OriginY - 85, 0, 0, s)
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
    Dim PrintableWidth As Single = e.MarginBounds.Width
    Dim PrintableHeight As Single = e.MarginBounds.Height

    Dim rectDraw As RectangleF

    Try
        '(e.MarginBounds.Left, e.MarginBounds.Top, e.MarginBounds.Width,
e.MarginBounds.Height)

        If PrintableWidth / PrintableHeight > ((GraphicsWidth + 185) /
(GraphicsDepth + 170)) Then
            rectDraw = New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
GraphicsWidth * e.MarginBounds.Height /
GraphicsDepth, e.MarginBounds.Height)
        Else
            rectDraw = New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
e.MarginBounds.Width, GraphicsDepth *
e.MarginBounds.Width / GraphicsWidth)
        End If

        e.Graphics.DrawImage(memoryImage, rectDraw)
        'e.Graphics.DrawImage(
Catch ex As Exception

    End Try

End Sub

Private Sub PrintPreviewToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles PrintPreviewToolStripMenuItem.Click
    Try
        CaptureScreen()

        'Specify current page settings
        PrintDocument1.DefaultPageSettings = PrintPageSettings

        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()
    Catch ex As Exception
        'Display error message
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

```

Private Sub FileSaveProcedure()
    Dim i As UInteger

    'saving mesh data
    VerticalGridMatrixTable = New DataTable("VerticalGridMatrixTable")
    VerticalGridMatrixTable.Columns.Add("LocationX", GetType(Single))

    HorizontalGridMatrixTable = New DataTable("HorizontalGridMatrixTable")
    HorizontalGridMatrixTable.Columns.Add("LocationY", GetType(Single))

    ElementNodesTable = New DataTable("ElementNodesTable")
    ElementNodesTable.Columns.Add("NodeI", GetType(Single))
    ElementNodesTable.Columns.Add("NodeJ", GetType(Single))
    ElementNodesTable.Columns.Add("NodeM", GetType(Single))

    NodalCoordinatesAndLoadsTable = New DataTable("NodalCoordinatesAndLoadsTable")
    NodalCoordinatesAndLoadsTable.Columns.Add("CoordinateX", GetType(Single))
    NodalCoordinatesAndLoadsTable.Columns.Add("CoordinateY", GetType(Single))
    NodalCoordinatesAndLoadsTable.Columns.Add("VerticalLoad", GetType(Single))

    FixityStatusTable = New DataTable("FixityStatusTable")
    FixityStatusTable.Columns.Add("FixityStatus", GetType(UInteger))

    Try
        BasicProjectDataTable.Columns.Add("NumberHorizontalGrid",
        GetType(UInteger))
        BasicProjectDataTable.Columns.Add("NumberVerticalGrid", GetType(UInteger))
        BasicProjectDataTable.Columns.Add("numberOfElements", GetType(UInteger))
        BasicProjectDataTable.Columns.Add("numberOfNodes", GetType(UInteger))
        BasicProjectDataTable.Columns.Add("DegreesOfFreedom", GetType(UInteger))
    Catch ex As Exception
    End Try

    If BasicProjectDataRow.Item("ProgramStage") > 2 Then
        'saving the mesh data if mesh has been generated
        For i = 0 To UBound(VerticalGridMatrix, 2)
            VerticalGridMatrixTable.Rows.Add({i})
            VerticalGridMatrixTable.Rows(i).Item(0) = VerticalGridMatrix(0, i)
        Next
        BasicProjectDataRow.Item("NumberVerticalGrid") =
        UBound(VerticalGridMatrix, 2) + 1
        For i = 0 To UBound(HorizontalGridMatrix)
            HorizontalGridMatrixTable.Rows.Add({i})
            HorizontalGridMatrixTable.Rows(i).Item(0) = HorizontalGridMatrix(i)
        Next
        BasicProjectDataRow.Item("NumberHorizontalGrid") =
        UBound(HorizontalGridMatrix) + 1
        For i = 0 To UBound(ElementNodes)
            ElementNodesTable.Rows.Add({i})
            ElementNodesTable.Rows(i).Item(0) = ElementNodes(i, 0)
            ElementNodesTable.Rows(i).Item(1) = ElementNodes(i, 1)
            ElementNodesTable.Rows(i).Item(2) = ElementNodes(i, 2)
        Next
        BasicProjectDataRow.Item("numberOfElements") = UBound(ElementNodes) + 1
        For i = 0 To UBound(NodalCoordinates)
            NodalCoordinatesAndLoadsTable.Rows.Add({i})
            NodalCoordinatesAndLoadsTable.Rows(i).Item(0) = NodalCoordinates(i, 0)
            NodalCoordinatesAndLoadsTable.Rows(i).Item(1) = NodalCoordinates(i, 1)
            NodalCoordinatesAndLoadsTable.Rows(i).Item(2) = ForcesMatrix(2 * i +
1)

        Next
        BasicProjectDataRow.Item("numberOfNodes") = UBound(NodalCoordinates) + 1

```

```
For i = 0 To UBound(FixityStatus)
    FixityStatusTable.Rows.Add({i})
    FixityStatusTable.Rows(i).Item(0) = FixityStatus(i)
Next
BasicProjectDataRow.Item("DegreesOfFreedom") = UBound(FixityStatus) + 1
End If

ProjectDataSet = New DataSet("FiniteElementSoftwareDataSet")
ProjectDataSet.Tables.Add(BasicProjectDataTable)
ProjectDataSet.Tables.Add(SoilPropertiesTable)
ProjectDataSet.Tables.Add(PointLoadTable)
ProjectDataSet.Tables.Add(UniformLoadTable)
ProjectDataSet.Tables.Add(VerticalGridMatrixTable)
ProjectDataSet.Tables.Add(HorizontalGridMatrixTable)
ProjectDataSet.Tables.Add(ElementNodesTable)
ProjectDataSet.Tables.Add(NodalCoordinatesAndLoadsTable)
ProjectDataSet.Tables.Add(FixityStatusTable)
'MsgBox(SavedFileName)
ProjectDataSet.WriteXml(SavedFileName)

ProjectDataSet.Tables.Remove(BasicProjectDataTable)
ProjectDataSet.Tables.Remove(SoilPropertiesTable)
ProjectDataSet.Tables.Remove(PointLoadTable)
ProjectDataSet.Tables.Remove(UniformLoadTable)
ProjectDataSet.Tables.Remove(VerticalGridMatrixTable)
ProjectDataSet.Tables.Remove(HorizontalGridMatrixTable)
ProjectDataSet.Tables.Remove(ElementNodesTable)
ProjectDataSet.Tables.Remove(NodalCoordinatesAndLoadsTable)
ProjectDataSet.Tables.Remove(FixityStatusTable)
End Sub

Private Sub SaveToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SaveToolStripMenuItem.Click
    If SimpleSaveEnabled Then
        Try
            FileSaveProcedure()
        Catch ex As Exception
            MsgBox("Error! The file could not be saved." & vbCrLf & ex.Message)
        End Try
    Else
        SaveAsToolStripMenuItem.PerformClick()
    End If
End Sub

End Class
```

### \*\*Codes Used for Managing Functions Associated with Soil Date Entry Interface

```
Imports System.Math
Imports System.Data
Public Class InputForm
    Public EnteredData() As Boolean = {False, True, False, False, False, False, False,
False, False, False, False, False}
    Dim AllEntered As Boolean = True
    Dim SoilPropertiesTableCreated As Boolean = False
    Dim NumberSoilLayersEntered As UInteger
    Dim TempoSoilTable As DataTable

    Private Sub InputForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
Try
    'Loading the saved width and depth
    TextBox2.Text = BasicProjectDataRow.Item("Width")
    TextBox1.Text = BasicProjectDataRow.Item("Depth")
    SoilProperties.DataSource = SoilPropertiesTable

    'Loading the number of layers
    NumberSoilLayersEntered = BasicProjectDataRow.Item("numberOFLayers")
    TextBox3.Text = BasicProjectDataRow.Item("numberOFLayers")

    'Loading the type of problem (Plane Strian Vs Axisymmetric)
    If BasicProjectDataRow.Item("PlaneStrainOn") Then
        PlaneStrainButton.Checked = True
    Else
        AxisymmetricButton.Checked = True
    End If

Catch ex As Exception
    'There is no saved database.
    'Creating soil properties table when parameters have not been saved yet

    If SoilPropertiesTableCreated = False Then
        SoilPropertiesTable.Columns.Add("Layer", GetType(String))
        SoilPropertiesTable.Columns.Add("Thickness (m)", GetType(String))
        SoilPropertiesTable.Columns.Add("Modulus of Elasticity(kN/m2)",
GetType(String))
        SoilPropertiesTable.Columns.Add("Poisson's Ratio", GetType(String))
        SoilPropertiesTable.Columns(0).ReadOnly = True
        SoilProperties.DataSource = SoilPropertiesTable
        TextBox3.Text = "1"
        PlaneStrainButton.Checked = True
        SoilPropertiesTableCreated = True
    End If
End Try

'making a backup copy of soil properties to enable reversing changes
TempoSoilTable = SoilPropertiesTable.Copy()
End Sub

Private Sub TextBox3_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TextBox3.TextChanged
    Dim i As UShort

    'Arranging the soil properties DataGridView
    If TextBox3.Text <> "" Then
        Try
            DecimalTest = 1 / Sqrt(TextBox3.Text)
            If TextBox3.Text > NumberSoilLayersEntered Then
                For i = NumberSoilLayersEntered + 1 To TextBox3.Text
                    SoilPropertiesTable.Rows.Add({i})
                Next
            Else
                For i = TextBox3.Text + 1 To NumberSoilLayersEntered Step 1
                    SoilPropertiesTable.Rows.RemoveAt(TextBox3.Text)
                Next
            End If
            NumberSoilLayersEntered = TextBox3.Text
        Catch ex As Exception
            MsgBox("The entry for number of layers is not valid!")
            TextBox3.Text = ""
        Exit Sub
    End If
End Sub
```

```
        End Try
    End If
End Sub

Private Sub SoilProperties_CellValueChanged(ByVal sender As Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles
SoilProperties.CellValueChanged
    Dim ScanRow As DataRow
    Dim CumulativeThickness As Single = 0

    Try
        For Each ScanRow In SoilPropertiesTable.Rows
            If (IsDBNull(ScanRow.Item(1)) = False) Then
                CumulativeThickness += ScanRow.Item(1)
                TextBox1.Text = CumulativeThickness
            End If
        Next ScanRow
    Catch ex As Exception
        MsgBox("Invalid Depth Input!")
    Exit Sub
End Try
End Sub

Private Sub CancelSoilDataButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CancelSoilDataButton.Click
    NumberSoilLayersEntered = NumberSoilLayers
    SoilPropertiesTable = TempoSoilTable.Copy()
    SoilProperties.DataSource = SoilPropertiesTable
    Me.DialogResult = DialogResult.Cancel
End Sub

Private Sub ApplySoilDataButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ApplySoilDataButton.Click
    'Converting Database Input Into Array Input (upon clicking APPLY at the Input
    Form)

    Dim ScanRow As DataRow
    Dim i As UShort = 0

    'testing the validity of width input
    Try
        DecimalTest = 1 / Sqrt(TextBox2.Text)
    Catch ex As Exception
        MsgBox("The entry for Width is not valid!")
    Exit Sub
End Try

    'testing the validity of number of soil layers input
    Try
        DecimalTest = 1 / Sqrt(TextBox3.Text)
    Catch ex As Exception
        MsgBox("The entry for number of layers is not valid!")
    Exit Sub
End Try

    'ReDim Preserve LayerThicknesses(1, TextBox3.Text - 1)
    ReDim ElasticModulus(TextBox3.Text - 1, 1)
    ReDim PoissonsRatio(TextBox3.Text - 1, 1)
    Dim TempoLayerThicknesses(1, TextBox3.Text - 1) As Single
    Dim CumulativeThickness As Single

    'testing the validity of soil properties input, saves the input if valid
```



```

Try
    i = 0
    For Each ScanRow In SoilPropertiesTable.Rows
        DecimalTest = 1 / Sqrt(ScanRow.Item(1)) 'checking that thickness
is positive
        DecimalTest = 1 / Sqrt(ScanRow.Item(2)) 'checking that elasticity
is positive
        DecimalTest = Sqrt(ScanRow.Item(3)) 'checking that poissons is
not negative
        If ScanRow.Item(3) >= 0.5 Then 'checking that poissons is
less than 0.5
            MsgBox("Layer " & i + 1 & ": Poisson's ratio should be less than
0.5")
            Exit Sub
        End If

        TempoLayerThicknesses(0, i) = ScanRow.Item(1)
        CumulativeThickness += TempoLayerThicknesses(0, i)
        TempoLayerThicknesses(1, i) = CumulativeThickness

        ElasticModulus(i, 0) = ScanRow.Item(2)
        PoissonsRatio(i, 0) = ScanRow.Item(3)
        i = i + 1
    Next
Catch ex As Exception
    MsgBox("The soil properties table is either incomplete or contains invalid
input!")
    Exit Sub
End Try

'saving values in variables and database
WidthW = TextBox2.Text
DepthD = TextBox1.Text
NumberSoilLayers = TextBox3.Text
LayerThicknesses = TempoLayerThicknesses.Clone()

If IsDBNull(BasicProjectDataRow.Item("ProgramStage")) Then
    BasicProjectDataRow.Item("ProgramStage") = 1
End If
BasicProjectDataRow.Item("Width") = TextBox2.Text
BasicProjectDataRow.Item("Depth") = TextBox1.Text
BasicProjectDataRow.Item("numberOfLayers") = NumberSoilLayers

If PlaneStrainButton.Checked = True Then
    PlaneStrainAnalysis = True
    BasicProjectDataRow.Item("PlaneStrainOn") = 1
Else
    PlaneStrainAnalysis = False
    BasicProjectDataRow.Item("PlaneStrainOn") = 0
    LeftEndVerticallyRestrained = False
End If

BasicProjectDataRow.Item("LeftEndVerticallyRestrained") =
LeftEndVerticallyRestrained
BasicProjectDataRow.Item("RightEndVerticallyRestrained") =
RightEndVerticallyRestrained

If BasicProjectDataRow.Item("MeshDefaultOn") Then
    If WidthW > DepthD Then
        BasicProjectDataRow.Item("MaximumMeshSize") = WidthW / 40
    Else
        BasicProjectDataRow.Item("MaximumMeshSize") = DepthD / 40
    End If
End If

```

```
        End If
    End If

    Me.DialogResult = DialogResult.OK
End Sub

End Class
```

### \*\*Codes Used for Managing Functions Associated with Loading Date Entry Interface

```
Imports System.IO
Imports System.Math
Imports System.Data
Public Class Loading_Data_Input
    Dim EnteredData = {False, False, False, False, False, False}
    Dim AllEntered As Boolean
    'Dim LoadingDataTablesCreated As Boolean = False
    Dim BackupPointLoadsTable As New DataTable
    Dim BackupUniformLoadsTable As New DataTable
    Dim TempoPointLoadNumbers As UInteger
    Dim TempoUniformLoadNumbers As UInteger

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        'Clicking the apply button

        If TextBox1.Text = "" Then
            MsgBox("Entry for number of concentrated loads is invalid!")
            Exit Sub
        End If
        If TextBox2.Text = "" Then
            MsgBox("Entry for number of distributed loads is invalid!")
            Exit Sub
        End If
        If TextBox1.Text = 0 And TextBox2.Text = 0 Then
            MsgBox("No loads have been applied!")
            Exit Sub
        End If

        Dim i As UShort
        Dim ScanRow As DataRow

        Try
            i = 0
            For Each ScanRow In PointLoadTable.Rows()
                DecimalTest = ScanRow!Magnitude
                DecimalTest = ScanRow!Location
                DecimalTest = Sqrt(ScanRow!Location)
                DecimalTest = Sqrt(WidthW - ScanRow!Location)
                DecimalTest = ScanRow!Depth
                DecimalTest = Sqrt(ScanRow!Depth)
                DecimalTest = Sqrt(DepthD - ScanRow!Depth)
                i = i + 1
            Next
        Catch ex As Exception
            MsgBox("The concentrated loads table is either incomplete of contains
invalid input!")
            Exit Sub
        End Try
    End Sub
End Class
```

```

Try
    i = 0
    For Each ScanRow In UniformLoadTable.Rows()
        DecimalTest = ScanRow!Magnitude
        DecimalTest = ScanRow!Beginning
        DecimalTest = Sqrt(ScanRow!Beginning)
        DecimalTest = Sqrt(WidthW - ScanRow!Beginning)
        DecimalTest = ScanRow!End
        DecimalTest = Sqrt(ScanRow!end)
        DecimalTest = Sqrt(WidthW - ScanRow!end)
        DecimalTest = Sqrt(ScanRow!end - ScanRow!Beginning)
        DecimalTest = ScanRow!Depth
        DecimalTest = Sqrt(ScanRow!Depth)
        DecimalTest = Sqrt(DepthD - ScanRow!Depth)
        i = i + 1
    Next
Catch ex As Exception
    MsgBox("The distributed loads table is either incomplete of contains
invalid input!")
Exit Sub
End Try
NumberPointLoads = TextBox1.Text
NumberUniformLoads = TextBox2.Text

BasicProjectDataRow.Item("NumberPointLoads") = TextBox1.Text
BasicProjectDataRow.Item("NumberUniformLoads") = TextBox2.Text
BackupPointLoadsTable = PointLoadTable.Copy
BackupUniformLoadsTable = UniformLoadTable.Copy

'Converting Database Input Into Array Input
ReDim PointLoadMatrix(NumberPointLoads - 1, 2) 'Column-0 =magnitude, Column-1
=location, Column-2 =depth
ReDim UDLMatrix(NumberUniformLoads - 1, 3)      'Column-0 =magnitude, Column-1
=beginning, column-2 =end, Column-3 =depth
LoadingDataEntered = True

Try
    i = 0
    For Each ScanRow In PointLoadTable.Rows()
        PointLoadMatrix(i, 0) = ScanRow!Magnitude
        PointLoadMatrix(i, 1) = ScanRow!Location
        PointLoadMatrix(i, 2) = ScanRow!Depth
        i = i + 1
    Next
Catch ex As Exception
    MsgBox("The concentrated loads table is either incomplete or contains
invalid input!")
Exit Sub
End Try

Try
    i = 0
    For Each ScanRow In UniformLoadTable.Rows()
        UDLMatrix(i, 0) = ScanRow!Magnitude
        UDLMatrix(i, 1) = ScanRow!Beginning
        UDLMatrix(i, 2) = ScanRow!End
        UDLMatrix(i, 3) = ScanRow!Depth
        i = i + 1
    Next
Catch ex As Exception

```

```
        MsgBox("The distributed loads table is either incomplete or contains  
invalid input!")  
        Exit Sub  
    End Try  
  
    Me.DialogResult = DialogResult.OK  
End Sub  
  
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button2.Click  
    'cancel button clicked  
    TempoPointLoadNumbers = NumberPointLoads  
    TempoUniformLoadNumbers = NumberUniformLoads  
    PointLoadTable = BackupPointLoadsTable.Copy  
    UniformLoadTable = BackupUniformLoadsTable.Copy  
    TextBox1.Text = NumberPointLoads  
    TextBox2.Text = NumberUniformLoads  
    Me.DialogResult = DialogResult.Cancel  
End Sub  
  
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles TextBox1.TextChanged  
    'Event Procedure when the number of POINT loads has been changed  
  
    Dim i As UShort  
  
    If TextBox1.Text <> "" Then  
        Try  
            If TextBox1.Text > TempoPointLoadNumbers Then  
                For i = TempoPointLoadNumbers + 1 To TextBox1.Text  
                    PointLoadTable.Rows.Add({i})  
                Next  
            Else  
                For i = TextBox1.Text + 1 To TempoPointLoadNumbers Step 1  
                    PointLoadTable.Rows.RemoveAt(TextBox1.Text)  
                Next  
            End If  
  
            TempoPointLoadNumbers = TextBox1.Text  
            If TempoPointLoadNumbers <> TextBox1.Text Then  
                MsgBox("Entry for number of concentrated loads is invalid!")  
                TextBox1.Text = TempoPointLoadNumbers  
                Exit Try  
            End If  
        Catch ex As Exception  
            MsgBox("Entry for number of concentrated loads is invalid!")  
            TextBox1.Text = ""  
        End Try  
    End If  
End Sub  
  
Private Sub TextBox2_TextChanged(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles TextBox2.TextChanged  
    'Event Procedure when the number of UNIFORM loads has been changed  
  
    Dim i As UShort  
    'Dim TempoUniformLoadNumbers As UInteger  
  
    If TextBox2.Text <> "" Then  
        Try  
            If TextBox2.Text > TempoUniformLoadNumbers Then
```

```
        For i = TempoUniformLoadNumbers + 1 To TextBox2.Text
            UniformLoadTable.Rows.Add({i})
        Next
    Else
        For i = TextBox2.Text + 1 To TempoUniformLoadNumbers Step 1
            UniformLoadTable.Rows.RemoveAt(TextBox2.Text)
        Next
    End If
    TempoUniformLoadNumbers = TextBox2.Text
    If TempoUniformLoadNumbers <> TextBox2.Text Then
        MsgBox("Entry for number of distributed loads is invalid!")
        TextBox2.Text = TempoUniformLoadNumbers
    End If
Catch ex As Exception
    MsgBox("Entry for number of distributed loads is invalid!")
    TextBox2.Text = ""
End Try
End If

End Sub

Private Sub Loading_Data_Input_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim Note As String

    TempoPointLoadNumbers = NumberPointLoads
    TempoUniformLoadNumbers = NumberUniformLoads

    Try
        'Case: there is a saved database
        TextBox1.Text = BasicProjectDataRow("NumberPointLoads")
        Try
            'there is at least one point load
            PointLoadData.DataSource = PointLoadTable
        Catch ex As Exception
            'there is no point load (database saved is only for uniform load)
            BuildPointLoadTable()
            PointLoadData.DataSource = PointLoadTable
        End Try
        BackupPointLoadsTable = PointLoadTable.Copy
    Catch ex As Exception
        'Database doesn't exist for loads. It gets created here.
        Try
            BuildPointLoadTable()
        Catch
        End Try
        PointLoadData.DataSource = PointLoadTable
        BackupPointLoadsTable = PointLoadTable.Copy
        TextBox1.Text = 1
    End Try

    Try
        'Case: there is a saved database
        TextBox2.Text = BasicProjectDataRow("NumberUniformLoads")
        Try
            'there is at least one uniform load
            UniformLoadData.DataSource = UniformLoadTable
        Catch ex As Exception
            'there is no point load (database saved is only for point load)
            BuildUniformLoadTable()
            UniformLoadData.DataSource = UniformLoadTable
        End Try
    End Try
```

```
        BackupUniformLoadsTable = UniformLoadTable.Copy
    Catch ex As Exception
        'Database doesn't exist for loads. It gets created here.
    Try
        BuildUniformLoadTable()
    Catch
    End Try
    UniformLoadData.DataSource = UniformLoadTable
    BackupUniformLoadsTable = UniformLoadTable.Copy
    TextBox2.Text = 1
End Try

If BasicProjectDataRow.Item("PlaneStrainOn") Then
    Note = "Note:" & "          The Concentrated Load Shown here is equivalent to
Continious "
    Note = Note & "Line Loading" & vbCrLf & vbCrLf
    Note = Note & "          The Distributed Load shown here is equivalen
to Uniform "
    Note = Note & "Strip Load"
    Me.Label14.Location = New System.Drawing.Point(60, 340)
    Label17.Text = "kN/m"
Else
    Note = "Note:" & "          The Concentrated Load Shown in this Axisymmetric
model is equivalent to " & vbCrLf
    Note = Note & "          "
    Note = Note & "- a Point Load if R=0 (the unit is kN)" & vbCrLf
    Note = Note & "          "
    Note = Note & "- a circumferential line load if R>0 (the unit is kN/m)" &
vbCrLf & vbCrLf
    Note = Note & "          The Distributed Load shown here is equivalen
to Uniform "
    Note = Note & "Load distributed over a circular area"
    Me.Label14.Location = New System.Drawing.Point(60, 310)
    Label17.Text = "(kN or kN/m)"
End If
Label14.Text = Note

End Sub

Private Sub BuildPointLoadTable()
    PointLoadTable.Columns.Add("LoadCase", GetType(String))
    PointLoadTable.Columns.Add("Magnitude", GetType(String))
    PointLoadTable.Columns.Add("Location", GetType(String))
    PointLoadTable.Columns.Add("Depth", GetType(String))
    PointLoadTable.Columns(0).ReadOnly = True
End Sub

Private Sub BuildUniformLoadTable()
    UniformLoadTable.Columns.Add("LoadCase", GetType(String))
    UniformLoadTable.Columns.Add("Magnitude", GetType(String))
    UniformLoadTable.Columns.Add("Beginning", GetType(String))
    UniformLoadTable.Columns.Add("End", GetType(String))
    UniformLoadTable.Columns.Add("Depth", GetType(String))
    UniformLoadTable.Columns(0).ReadOnly = True
End Sub

End Class
```

\*\*Codes Used for Managing the End Conditions of the Soil Model

```
Imports System.Data
Public Class EndConditionForm

    Private Sub CancelEndCondition_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CancelEndCondition.Click
        Try
            LeftEndVerticallyRestrained =
BasicProjectDataRow("LeftEndVerticallyRestrained")
            RightEndVerticallyRestrained =
BasicProjectDataRow("RightEndVerticallyRestrained")
        Catch ex As Exception
            LeftEndVerticallyRestrained = True
            RightEndVerticallyRestrained = True
        End Try
        Me.DialogResult = DialogResult.Cancel
    End Sub

    Private Sub EndConditionForm_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        If LeftEndVerticallyRestrained = True Then
            LeftRestrainedRadioButton.Checked = True
        Else
            LeftUnrestrictedRadioButton.Checked = True
        End If
        If RightEndVerticallyRestrained = True Then
            RightRestrainedButton.Checked = True
        Else
            RightUnrestrictedButton.Checked = True
        End If
    End Sub

    Private Sub EndConditionForm_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim BoundaryConditionGraphics As Graphics
        BoundaryConditionGraphics = GroupBox1.CreateGraphics

        Dim SoilColor As New SolidBrush(Color.BurlyWood)
        BoundaryConditionGraphics.FillRectangle(SoilColor, 170, 80, 250, 100)

        Dim BoundaryLinePen As New Pen(Color.Black)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 170, 80, 170, 180)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 170, 180, 420, 180)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 420, 180, 420, 80)

        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 220, 173, 265, 173)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 220, 187, 265, 187)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 235, 173, 250, 187)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 235, 187, 250, 173)

        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 325, 173, 370, 173)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 325, 187, 370, 187)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 340, 173, 355, 187)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 340, 187, 355, 173)

        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 160, 105, 160, 150)
        BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 180, 105, 180, 150)
```

```
If LeftEndVerticallyRestrained = True Then
    BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 160, 115, 180, 140)
    BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 160, 140, 180, 115)
End If

BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 410, 105, 410, 150)
BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 430, 105, 430, 150)
If RightEndVerticallyRestrained = True Then
    BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 410, 115, 430, 140)
    BoundaryConditionGraphics.DrawLine(BoundaryLinePen, 410, 140, 430, 115)
End If

End Sub

Private Sub LeftRestrainedRadioButton_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
LeftRestrainedRadioButton.CheckedChanged
    If LeftRestrainedRadioButton.Checked = True Then
        LeftEndVerticallyRestrained = True
    Else
        LeftEndVerticallyRestrained = False
    End If
    Refresh()
End Sub

Private Sub RightRestrainedButton_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles RightRestrainedButton.CheckedChanged
    If RightRestrainedButton.Checked = True Then
        RightEndVerticallyRestrained = True
    Else
        RightEndVerticallyRestrained = False
    End If
    Refresh()
End Sub

Private Sub ApplyButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ApplyButton.Click
    BasicProjectDataRow.Item("LeftEndVerticallyRestrained") =
LeftEndVerticallyRestrained
    BasicProjectDataRow.Item("RightEndVerticallyRestrained") =
RightEndVerticallyRestrained
    Me.DialogResult = DialogResult.OK
End Sub

End Class
```

### \*\*Codes Used for Managing the MAXIMUM MESH DIMENSION

```
Imports System.Math
Imports System.Data
Public Class Max_Mesh_Dim

    Private Sub CheckBox1_CheckStateChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles CheckBox1.CheckStateChanged
        If CheckBox1.Checked Then
            TextBox1.Enabled = False
            If WidthW > DepthD Then
                TextBox1.Text = WidthW / 40
            Else
                TextBox1.Text = DepthD / 40
            End If
        End If
    End Sub

End Class
```



```
        End If
    Else
        TextBox1.Enabled = True
    End If
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    'OK button clicked
    Try
        DecimalTest = 1 / Sqrt(TextBox1.Text)
        MaximumMeshDimension = TextBox1.Text

        If CheckBox1.Checked Then
            BasicProjectDataRow.Item("MeshDefaultOn") = True
            DefaultMeshOn = True
        Else
            BasicProjectDataRow.Item("MeshDefaultOn") = False
            DefaultMeshOn = False
        End If
        BasicProjectDataRow.Item("MaximumMeshSize") = TextBox1.Text
        My.Forms.MainForm.GenerateMeshButton.PerformClick()
    Catch ex As Exception
        MsgBox("Entry is not valid for maximum mesh dimension")
    Exit Sub
End Try
Close()
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    DefaultMeshOn = True
    Close()
End Sub

Private Sub Max_Mesh_Dim_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Try
        If BasicProjectDataRow.Item("MeshDefaultOn") Then
            CheckBox1.Checked = True

            TextBox1.Enabled = False
            If WidthW > DepthD Then
                TextBox1.Text = WidthW / 40
            Else
                TextBox1.Text = DepthD / 40
            End If
        Else
            CheckBox1.Checked = False

            TextBox1.Enabled = True
            TextBox1.Text = BasicProjectDataRow.Item("MaximumMeshSize")
        End If
    Catch ex As Exception
        CheckBox1.Checked = True

        TextBox1.Enabled = False
        If WidthW > DepthD Then
            TextBox1.Text = WidthW / 40
        Else
            TextBox1.Text = DepthD / 40
        End If
    End Try
```

```
End Try

End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TextBox1.TextChanged
    If TextBox1.Text = "" Then
        Button2.Enabled = False
    Else
        Button2.Enabled = True
    End If
End Sub

End Class
```

### \*\*Codes Used for Solving the System of Linear Equations

```
Imports System.Math
Public Class Gauss_Elimination

    Private Sub Gauss_Elimination_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        End Sub

    Private Sub Gauss_Elimination_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Shown
        Dim NonZeroList() As UInteger
        Dim i, j As UShort
        Dim k As Short
        Dim NumberOfCaluculations, progress As ULong
        Dim ProgressGraphics As Graphics = Me.CreateGraphics
        Dim StringFont As New Font(Height, 12)
        Dim brush As Brush = Brushes.Black

        ProgressGraphics.DrawString("Processing...", StringFont, brush, New Point(70,
15))

        ProgressBar1.Minimum = 0
        ProgressBar1.Maximum = 100
        NumberOfCaluculations = Round(DOF * (DOF + 1) * (2 * DOF + 1) / 6, 0)

        'Gauss Elimination
        For i = 0 To DOF - 2                                     'reducer row
            ReDim NonZeroList(0)
            For j = i + 1 To (DOF - 1) 'for every element of the reducer row
                If ShortStiffness(i, j) <> 0.0 Then 'columns containing nonzero
elements(within the reducer row) are listed
                    ReDim Preserve NonZeroList(UBound(NonZeroList) + 1)
                    NonZeroList(UBound(NonZeroList)) = j
                End If
            Next
            For j = i + 1 To (DOF - 1) 'row to be reduced
                If ShortStiffness(j, i) <> 0.0 Then
                    Reducer = ShortStiffness(j, i) / ShortStiffness(i, i)
                    For k = 0 To UBound(NonZeroList) 'every element of the
row to be reduced(as per the non zero list)
                        ShortStiffness(j, NonZeroList(k)) = ShortStiffness(j,
NonZeroList(k)) - Reducer * ShortStiffness(i, NonZeroList(k))
                    Next
                End If
            Next
        Next
    End Sub
```

```
        Shortforces(j) = Shortforces(j) - Reducer * Shortforces(i)
    End If
Next
progress += (DOF - i - 1) * (DOF - i)
ProgressBar1.Value = (progress / NumberOfCaluculations) * 100
'ProgressPercentage.Text = Round((progress / NumberOfCaluculations), 0) &
" %"
Next

'Backsubstitution and Solution
For i = 1 To DOF
    Reducer = 0
    For j = 1 To i - 1
        Reducer = Reducer + GlobalDisplacements(UnknownDOFList(DOF - j)) *
ShortStiffness(DOF - i, DOF - j)
    Next
    GlobalDisplacements(UnknownDOFList(DOF - i)) = (Shortforces(DOF - i) -
Reducer) / ShortStiffness(DOF - i, DOF - i)
    progress += i
    ProgressBar1.Value = (progress / NumberOfCaluculations) * 100
    'ProgressPercentage.Text = Round((progress * 100 / NumberOfCaluculations),
0) & " %"
Next

Me.DialogResult = DialogResult.OK
End Sub
End Class
```

### \*\*Codes Used for Managing the Output Interface

```
Imports System.Math
Imports System.IO
Imports System.Drawing.Printing

Public Class Output
    Private PrintPageSettings As New PageSettings
    Dim memoryImage As Bitmap

    Dim MeshGraphicPoints() As Point
    Dim DeformedMeshGraphicPoints() As Point
    Dim GraphicsScale As Single
    Dim GraphicsWidth As UInteger
    Dim GraphicsDepth As UInteger
    Dim OriginX As UInteger = 300
    Dim OriginY As UInteger = 120
    Dim LegendSpace As UInteger = 180

    Dim Top25Percent() As UInteger
    Dim UpperMiddle25Percent() As UInteger
    Dim LowerMiddle25Percent() As UInteger
    Dim Least25Percent() As UInteger
    Dim ZeroPercentList() As UInteger
    Dim TensileList() As UInteger

    Dim HighestOccurance As UInteger
    Dim LeadingCategory As Byte
    Dim OutputUnit As String
    Dim MaximumValue As Single = 0
    Dim ReadyToRefresh As Boolean = True
```

```
Dim PreviousX, PreviousY As UInteger
Dim Ordinate, Perpendicular As String

Dim MaxGraphicsWidth As UInteger = 500
Dim MaxGraphicsDepth As UInteger = 300
Dim FullGraphicDepth As UInteger
Dim FullGraphicWidth As UInteger

Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CheckBox2.CheckedChanged
    If CheckBox2.Checked = False Then
        If ReadyToRefresh = True Then
            Refresh()
        End If
    Else
        DrawUndeformedMesh()
    End If
End Sub

Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CheckBox1.CheckedChanged
    If CheckBox1.Checked = False Then
        If ReadyToRefresh = True Then
            Refresh()
        End If
    Else
        DrawDeformedMesh()
    End If
End Sub

Private Sub Output_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    'Me.Controls.Add(OutPutListBox)
    'OutPutListBox.Items.Add("")
    'OutPutListBox.CausesValidation = False

    Try
        If PlaneStrainAnalysis Then
            Ordinate = "X"
            Perpendicular = "Y"
        Else
            Ordinate = "R"
            Perpendicular = "θ"
        End If

        If PlaneStrainAnalysis Then
            ComboBox1.Items(1) = "x Stress"
            ComboBox1.Items(2) = "y Stress"
            ComboBox1.Items(4) = "shear stress (xz)"
            ComboBox1.Items(5) = "x Strain"
            ComboBox1.Items(6) = "y Strain"
            ComboBox1.Items(8) = "shear strain (xz)"
        Else
            ComboBox1.Items(1) = "r stress"
            ComboBox1.Items(2) = "θ stress"
            ComboBox1.Items(4) = "shear stress (rz)"
            ComboBox1.Items(5) = "r Strain"
            ComboBox1.Items(6) = "θ Strain"
            ComboBox1.Items(8) = "shear strain (rz)"
        End If
        ComboBox1.SelectedIndex = 0
    End Try
```

```
        PrintPageSettings.Landscape = True
    Catch ex As Exception

    End Try

End Sub

Private Sub Output_Mousemove(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.MouseMove

    Dim p As New Point(MousePosition.X, MousePosition.Y)
    Dim PointerX, PointerY As Single

    Dim RowNumber, ColumnNumber As Integer
    Dim i As Integer = 0
    Dim j As Integer = 0
    Dim ElementsPreciding As UInteger
    Dim Element As UInteger
    Dim TooltipText As String

    Try
        PointerX = PointToClient(p).X
        PointerY = PointToClient(p).Y

        If PointerX >= OriginX And PointerX <= OriginX + GraphicsWidth And
PointerY >= OriginY And PointerY <= OriginY + GraphicsDepth Then
            PointerX = (PointerX - OriginX) / GraphicsScale
            PointerY = (PointerY - OriginY) / GraphicsScale

            Do Until PointerX >= VerticalGridMatrix(0, i) And (PointerX <=
VerticalGridMatrix(0, i + 1) Or i = UBound(VerticalGridMatrix, 2) - 1)
                i += 1
            Loop
            ColumnNumber = i

            Do Until PointerY >= HorizontalGridMatrix(j) And (PointerY <=
HorizontalGridMatrix(j + 1) Or j = UBound(HorizontalGridMatrix) - 1)
                j += 1
            Loop
            RowNumber = j

            ElementsPreciding = 4 * RowNumber * UBound(VerticalGridMatrix, 2) + 4
* ColumnNumber

            If PointerY >= HorizontalGridMatrix(j) + (HorizontalGridMatrix(j + 1)
- HorizontalGridMatrix(j)) / (VerticalGridMatrix(0, i + 1) - VerticalGridMatrix(0, i))
* (PointerX - VerticalGridMatrix(0, i)) Then
                If PointerY >= HorizontalGridMatrix(j + 1) +
(HorizontalGridMatrix(j) - HorizontalGridMatrix(j + 1)) / (VerticalGridMatrix(0, i +
1) - VerticalGridMatrix(0, i)) * (PointerX - VerticalGridMatrix(0, i)) Then
                    Element = ElementsPreciding + 2
                Else
                    Element = ElementsPreciding + 1
                End If
            Else
                If PointerY >= HorizontalGridMatrix(j + 1) +
(HorizontalGridMatrix(j) - HorizontalGridMatrix(j + 1)) / (VerticalGridMatrix(0, i +
1) - VerticalGridMatrix(0, i)) * (PointerX - VerticalGridMatrix(0, i)) Then
                    Element = ElementsPreciding + 3
                Else
                    Element = ElementsPreciding + 0
                End If
            End If
        End Try
    End Sub
```

```
End If

If PreviousX = MousePosition.X And PreviousY = MousePosition.Y Then

Else

    Dim AlphaI, AlphaJ, AlphaK As Single
    Dim BetaI, BetaJ, BetaK As Single
    Dim GammaI, GammaJ, GammaK As Single
    Dim ElementArea As Single
    Dim Vi, Vj, Vm As Single
    Dim Settlement As Single

    ElementArea = 0.5 * Abs((NodalCoordinates(ElementNodes(Element,
0), 0) * NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 1), 0) * NodalCoordinates(ElementNodes(Element,
2), 1) + NodalCoordinates(ElementNodes(Element, 2), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1)) -
(NodalCoordinates(ElementNodes(Element, 1), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1) + NodalCoordinates(ElementNodes(Element,
2), 0) * NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 0), 0) * NodalCoordinates(ElementNodes(Element,
2), 1)))

    BetaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 1) -
NodalCoordinates(ElementNodes(Element, 2), 1))
    BetaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 1) -
NodalCoordinates(ElementNodes(Element, 0), 1))
    BetaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 1) -
NodalCoordinates(ElementNodes(Element, 1), 1))

    GammaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 0) -
NodalCoordinates(ElementNodes(Element, 1), 0))
    GammaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 0) -
NodalCoordinates(ElementNodes(Element, 2), 0))
    GammaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 0) -
NodalCoordinates(ElementNodes(Element, 0), 0))

    AlphaI = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 1), 0) *
NodalCoordinates(ElementNodes(Element, 2), 1) - NodalCoordinates(ElementNodes(Element,
2), 0) * NodalCoordinates(ElementNodes(Element, 1), 1))
    AlphaJ = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 2), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1) - NodalCoordinates(ElementNodes(Element,
0), 0) * NodalCoordinates(ElementNodes(Element, 2), 1))
    AlphaK = (1 / 2 / ElementArea) *
(NodalCoordinates(ElementNodes(Element, 0), 0) *
NodalCoordinates(ElementNodes(Element, 1), 1) - NodalCoordinates(ElementNodes(Element,
1), 0) * NodalCoordinates(ElementNodes(Element, 0), 1))

    Vi = GlobalDisplacements(2 * ElementNodes(Element, 0) + 1)
    Vj = GlobalDisplacements(2 * ElementNodes(Element, 1) + 1)
    Vm = GlobalDisplacements(2 * ElementNodes(Element, 2) + 1)
```

```
Settlement = (AlphaI * Vi + AlphaJ * Vj + AlphaK * Vm) + PointerX
* (BetaI * Vi + BetaJ * Vj + BetaK * Vm) + PointerY * (GammaI * Vi + GammaJ * Vj +
GammaK * Vm)

TooltipText = "Xo= " & Round(PointerX, 3) & " m"
TooltipText = TooltipText & vbCrLf & "Yo= " & Round(PointerY, 3) &
" m"

TooltipText = TooltipText & vbCrLf & "Vertical Settlement= " &
Round(Settlement, 3) & " m"
TooltipText = TooltipText & vbCrLf & Ordinate & " Stress= " &
Format(HorizontalStress(Element), "g5") & " kN/m²"
TooltipText = TooltipText & vbCrLf & Perpendicular & " Stress= " &
Format(StressY(Element), "g5") & " kN/m²"
TooltipText = TooltipText & vbCrLf & "Z Stress= " &
Format(VerticalStress(Element), "g5") & " kN/m²"
TooltipText = TooltipText & vbCrLf & "Shear Stress= " &
Format(ShearStress(Element), "g5") & " kN/m²"
TooltipText = TooltipText & vbCrLf & Ordinate & " Strain= " &
Format(HorizontalStrain(Element), "g5")
TooltipText = TooltipText & vbCrLf & Perpendicular & " Strain= " &
Format(StrainY(Element), "g5")
TooltipText = TooltipText & vbCrLf & "Z Strain= " &
Format(VerticalStrain(Element), "g5")
TooltipText = TooltipText & vbCrLf & "Shear Strain= " &
Format(ShearStrain(Element), "g5")

ToolTip1.SetToolTip(Me, TooltipText)
End If
PreviousX = MousePosition.X
PreviousY = MousePosition.Y
Else
    ToolTip1.SetToolTip(Me, "")
End If
Catch ex As Exception

End Try

End Sub

Private Sub Output_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

Try
    If Me.Size.Width > 380 Then
        Dim i As Short
        Dim j As UShort
        'Dim k As Short
        'Dim l As Single

        'TextBox1.Select(0, 0)

        'Graphics for soil matrix
        Dim OutputGraphics As Graphics
        OutputGraphics = Me.CreateGraphics

        'Selecting Governing Scale from WidthScale & DepthScale

        MaxGraphicsWidth = Me.Size.Width * (1 - 0.1) - 300 - 70
        MaxGraphicsDepth = Me.Size.Height - 115 - 95 - LegendSpace
```

```

    If WidthW / DepthD > MaxGraphicsWidth / MaxGraphicsDepth Then
        GraphicsScale = MaxGraphicsWidth / WidthW
    Else
        GraphicsScale = MaxGraphicsDepth / DepthD
    End If
    GraphicsDepth = DepthD * GraphicsScale
    GraphicsWidth = WidthW * GraphicsScale
    OriginX = 300 + 75 + 0.5 * (MaxGraphicsWidth - GraphicsWidth)
    OriginY = 90 + 45 + 0.5 * (MaxGraphicsDepth - GraphicsDepth)

    Dim SoilColorDarkRed As New SolidBrush(Color.DarkRed)           ' > 75%
red
    Dim SoilColorRed As New SolidBrush(Color.Red)                   ' > 50%
    Dim SoilColorOrange As New SolidBrush(Color.DarkOrange)         '
>25%
    Dim SoilColorBrown As New SolidBrush(Color.BurlyWood)           ' >0%
lightgray
    Dim SoilColorLightGray As New SolidBrush(Color.LightGray)       ' ≈0%
(0.5% to -0.5%)
    Dim SoilColorWhite As New SolidBrush(Color.GhostWhite)          ' <0%
(tension)
    Dim SoilColor As New SolidBrush(Color.BurlyWood)                'Regular
soil color

    If ComboBox1.SelectedIndex <> 0 Then
        Select Case LeadingCategory
            Case 1
                OutputGraphics.FillRectangle(SoilColorDarkRed, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
            Case 2
                OutputGraphics.FillRectangle(SoilColorRed, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
            Case 3
                OutputGraphics.FillRectangle(SoilColorOrange, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
            Case 4
                OutputGraphics.FillRectangle(SoilColorBrown, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
            Case 5
                OutputGraphics.FillRectangle(SoilColorLightGray, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
            Case 6
                OutputGraphics.FillRectangle(SoilColorWhite, OriginX,
OriginY, GraphicsWidth, GraphicsDepth)
        End Select

        If LeadingCategory <> 1 Then
            For i = 1 To UBound(Top25Percent)
Top25Percent(i)
                Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 0), 1))
                Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 1), 1))
                Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Top25Percent(i), 2), 1))
                Dim TrianglePoints As Point() = {point1, point2, point3}
                OutputGraphics.FillPolygon(SoilColorDarkRed,
TrianglePoints)
            
```



```

        Next
    End If
    If LeadingCategory <> 2 Then
        For i = 1 To UBound(UpperMiddle25Percent)
UpperMiddle25Percent(i)
            Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 0), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 0), 1))
            Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 1), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 1), 1))
            Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 2), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(UpperMiddle25Percent(i), 2), 1))
            Dim TrianglePoints As Point() = {point1, point2, point3}
            OutputGraphics.FillPolygon(SoilColorRed, TrianglePoints)
        Next
    End If
    If LeadingCategory <> 3 Then
        For i = 1 To UBound(LowerMiddle25Percent)
LowerMiddle25Percent(i)
            Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 0), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 0), 1))
            Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 1), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 1), 1))
            Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 2), 0), OriginY + GraphicsScale *
* NodalCoordinates(ElementNodes(LowerMiddle25Percent(i), 2), 1))
            Dim TrianglePoints As Point() = {point1, point2, point3}
            OutputGraphics.FillPolygon(SoilColorOrange,
TrianglePoints)
        Next
    End If
    If LeadingCategory <> 4 Then
        For i = 1 To UBound(Least25Percent)
Least25Percent(i)
            Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 0), 1))
            Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 1), 1))
            Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Least25Percent(i), 2), 1))
            Dim TrianglePoints As Point() = {point1, point2, point3}
            OutputGraphics.FillPolygon(SoilColorBrown, TrianglePoints)
        Next
    End If
    If LeadingCategory <> 5 Then
        For i = 1 To UBound(ZeroPercentList)
Least25Percent(i)
            Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 0), 1))
            Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 1), 1))

```

```

        Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(ZeroPercentList(i), 2), 1))
        Dim TrianglePoints As Point() = {point1, point2, point3}
        OutputGraphics.FillPolygon(SoilColorLightGray,
TrianglePoints)
    Next
End If
If LeadingCategory <> 5 Then
    For i = 1 To UBound(TensileList)
Least25Percent(i)
        Dim point1 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 0), 1))
        Dim point2 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 1), 1))
        Dim point3 As New Point(OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(TensileList(i), 2), 1))
        Dim TrianglePoints As Point() = {point1, point2, point3}
        OutputGraphics.FillPolygon(SoilColorWhite, TrianglePoints)
    Next
End If

'OutputGraphics.FillRectangle(SoilColor, OriginX, OriginY,
GraphicsWidth, GraphicsDepth)

Dim LegendRectangle As New Pen(Color.Gray)
'Dim LegendGraphics As Graphics
'LegendGraphics = Me.CreateGraphics

'Graphics for Legend
If ComboBox1.SelectedIndex <> 0 Then

    Label5.Text = CStr(Format(0.75 * MaximumValue, "0.000")) & " -
" & CStr(Format(MaximumValue, "0.000")) & OutputUnit
    Label6.Text = CStr(Format(MaximumValue * 0.5, "0.000")) & " -
" & CStr(Format(0.75 * MaximumValue, "0.000")) & OutputUnit
    Label7.Text = CStr(Format(MaximumValue * 0.25, "0.000")) & " -
" & CStr(Format(0.5 * MaximumValue, "0.000")) & OutputUnit
    Label8.Text = CStr(Format(0.005 * MaximumValue, "0.000")) & "
- " & CStr(Format(0.25 * MaximumValue, "0.000")) & OutputUnit
    Label9.Text = " ≈ 0 " & OutputUnit
    Label10.Text = " < 0 " & OutputUnit & "(Tension)"

    LegendGroupBox.Visible = True
    If ComboBox1.SelectedIndex = 4 Or ComboBox1.SelectedIndex = 8
Then
        Panel4.Visible = False
        Label10.Visible = False
    Else
        Panel4.Visible = True
        Label10.Visible = True
    End If
End If
Else
    OutputGraphics.FillRectangle(SoilColor, OriginX, OriginY,
GraphicsWidth, GraphicsDepth)
    LegendGroupBox.Visible = False
End If

```

```
'Graphics for soil mesh
If CheckBox2.Checked = True Then
    DrawUndeformedMesh()
End If

'Dim RedPen As New Pen(Color.Red)
'Dim MagentaPen As New Pen(Color.DarkMagenta)

Dim LoadingPen As New Pen(Color.DarkMagenta, 2)
Dim Pencolor As New Pen(Color.Black)

'Graphics for deformed mesh
If CheckBox1.Checked = True Then
    DrawDeformedMesh()
End If

'Drawign Load Arrows
If LoadingDataEntered = True Then
    'Dim BackgroundBlank As New SolidBrush(Color.White)

    If NumberUniformLoads <> 0 Then
        For j = 0 To NumberUniformLoads - 1
            If UDLMatrix(j, 0) <> 0 Then
                'OutputGraphics.FillRectangle(BackgroundBlank, OriginX
+ GraphicsScale * UDLMatrix(j, 1) - 10, OriginY + GraphicsScale * UDLMatrix(j, 3) -
35, GraphicsScale * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) + 20, 35)
                OutputGraphics.DrawLine(LoadngPen, OriginX +
GraphicsScale * UDLMatrix(j, 1), OriginY + GraphicsScale * UDLMatrix(j, 3) - 20,
OriginX + GraphicsScale * UDLMatrix(j, 2), OriginY + GraphicsScale * UDLMatrix(j, 3) -
20)

                For i = 0 To 5
                    OutputGraphics.DrawLine(LoadngPen, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5, OriginY + GraphicsScale * UDLMatrix(j, 3) - 20, OriginX + GraphicsScale *
UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5, OriginY
+ GraphicsScale * UDLMatrix(j, 3))
                    OutputGraphics.DrawLine(LoadngPen, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5 - 5, OriginY + GraphicsScale * UDLMatrix(j, 3) - 7, OriginX + GraphicsScale *
UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5, OriginY
+ GraphicsScale * UDLMatrix(j, 3))
                    OutputGraphics.DrawLine(LoadngPen, OriginX +
GraphicsScale * UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j,
1)) / 5 + 5, OriginY + GraphicsScale * UDLMatrix(j, 3) - 7, OriginX + GraphicsScale *
UDLMatrix(j, 1) + GraphicsScale * i * (UDLMatrix(j, 2) - UDLMatrix(j, 1)) / 5, OriginY
+ GraphicsScale * UDLMatrix(j, 3))
                Next
            End If
        Next
    End If

    'Dim PencolorRed As New Pen(Color.Red)
    If NumberPointLoads <> 0 Then
        For j = 0 To NumberPointLoads - 1 Step 1
            If PointLoadMatrix(j, 0) <> 0 Then
                OutputGraphics.DrawLine(LoadngPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1), OriginY + GraphicsScale * PointLoadMatrix(j, 2)
- 30, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY + GraphicsScale *
PointLoadMatrix(j, 2))
            End If
        Next
    End If
End If
```

```
        OutputGraphics.DrawLine>LoadingPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1) - 7, OriginY + GraphicsScale *
PointLoadMatrix(j, 2) - 10, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY +
GraphicsScale * PointLoadMatrix(j, 2))
        OutputGraphics.DrawLine>LoadingPen, OriginX +
GraphicsScale * PointLoadMatrix(j, 1) + 7, OriginY + GraphicsScale *
PointLoadMatrix(j, 2) - 10, OriginX + GraphicsScale * PointLoadMatrix(j, 1), OriginY +
GraphicsScale * PointLoadMatrix(j, 2))
        End If
    Next
End If
End If

'Graphics for Layer Lines
Dim PenColorLayers As New Pen(Color.Black, 2)
For i = 0 To (UBound(LayerThicknesses, 2) - 1)
    OutputGraphics.DrawLine(PenColorLayers, OriginX, OriginY +
GraphicsScale * LayerThicknesses(1, i), OriginX + GraphicsWidth, OriginY +
GraphicsScale * LayerThicknesses(1, i))
Next

'Graphics for Soil Boundary
Dim SoilBoundaryPen As New Pen(Color.Black, 1)

'Drawing Soil Matrix Boundary
OutputGraphics.DrawLine(SoilBoundaryPen, OriginX, OriginY, OriginX,
OriginY + GraphicsDepth) 'Left
OutputGraphics.DrawLine(SoilBoundaryPen, OriginX, OriginY +
GraphicsDepth, OriginX + GraphicsWidth, OriginY + GraphicsDepth) 'Bottom
OutputGraphics.DrawLine(SoilBoundaryPen, OriginX + GraphicsWidth,
OriginY + GraphicsDepth, OriginX + GraphicsWidth, OriginY) 'Right

'End conditions left
OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) - 30, OriginX - 10, OriginY + CInt(GraphicsDepth / 3) + 30)
OutputGraphics.DrawLine(Pencolor, OriginX + 10, OriginY +
CInt(GraphicsDepth / 3) - 30, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) + 30)

OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 30, OriginX - 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 30)
OutputGraphics.DrawLine(Pencolor, OriginX + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 30, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 30)

If LeftEndVerticallyRestrained = True Then
    OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) - 10, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) + 10)
    OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth / 3) + 10, OriginX + 10, OriginY + CInt(GraphicsDepth / 3) - 10)

    OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 10, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
+ 10)
    OutputGraphics.DrawLine(Pencolor, OriginX - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 10, OriginX + 10, OriginY + CInt(GraphicsDepth * 2 / 3)
- 10)
End If
```

```
'End Condition Bottom with crossing
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth / 3) -
40, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth / 3) + 40, OriginY +
GraphicsDepth - 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth / 3) -
40, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth / 3) + 40, OriginY +
GraphicsDepth + 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth / 3) -
20, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth / 3) + 20, OriginY +
GraphicsDepth + 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth / 3) -
20, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth / 3) + 20, OriginY +
GraphicsDepth - 10)

    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth * 2 /
3) - 40, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 40,
OriginY + GraphicsDepth - 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth * 2 /
3) - 40, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 40,
OriginY + GraphicsDepth + 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth * 2 /
3) - 20, OriginY + GraphicsDepth - 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 20,
OriginY + GraphicsDepth + 10)
    OutputGraphics.DrawLine(Pencolor, OriginX + CInt(GraphicsWidth * 2 /
3) - 20, OriginY + GraphicsDepth + 10, OriginX + CInt(GraphicsWidth * 2 / 3) + 20,
OriginY + GraphicsDepth - 10)

'End Condition Right
    OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth / 3) - 30, OriginX + GraphicsWidth - 10, OriginY +
CInt(GraphicsDepth / 3) + 30)
    OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth + 10,
OriginY + CInt(GraphicsDepth / 3) - 30, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) + 30)

    OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth * 2 / 3) - 30, OriginX + GraphicsWidth - 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 30)
    OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth + 10,
OriginY + CInt(GraphicsDepth * 2 / 3) - 30, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 30)

    If RightEndVerticallyRestrained = True Then
        OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth / 3) - 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) + 10)
        OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth / 3) + 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth / 3) - 10)

        OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth * 2 / 3) - 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) + 10)
        OutputGraphics.DrawLine(Pencolor, OriginX + GraphicsWidth - 10,
OriginY + CInt(GraphicsDepth * 2 / 3) + 10, OriginX + GraphicsWidth + 10, OriginY +
CInt(GraphicsDepth * 2 / 3) - 10)
    End If

'Drawing Coordinate Axes and Directional Arrows
Dim CoordinatesPen As New Pen(Color.DarkRed, 3)
```

```
Dim AxisPen As New Pen(Color.Gray, 1)
Dim font As New Font(Height, 14)
Dim OriginFont As New Font(Height, 10)
Dim brush As Brush = Brushes.Black
Dim TextFormat As New StringFormat(StringFormatFlags.NoClip)
Dim Centralformat As New StringFormat(StringFormatFlags.NoClip)
Dim RightAlignFormat As New StringFormat(LeftRightAlignment.Right)

    OutputGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 5,
OriginY, OriginX + GraphicsWidth + 40, OriginY)
    OutputGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 30,
OriginY - 5, OriginX + GraphicsWidth + 40, OriginY)
    OutputGraphics.DrawLine(CoordinatesPen, OriginX + GraphicsWidth + 30,
OriginY + 5, OriginX + GraphicsWidth + 40, OriginY)
    If PlaneStrainAnalysis Then
        OutputGraphics.DrawString("x", font, brush, OriginX +
GraphicsWidth + 40, OriginY - 28, TextFormat)
    Else
        OutputGraphics.DrawString("r", font, brush, OriginX +
GraphicsWidth + 40, OriginY - 28, TextFormat)
    End If

    OutputGraphics.DrawLine(CoordinatesPen, OriginX, OriginY +
GraphicsDepth + 5, OriginX, OriginY + GraphicsDepth + 40)
    OutputGraphics.DrawLine(CoordinatesPen, OriginX - 5, OriginY +
GraphicsDepth + 30, OriginX, OriginY + GraphicsDepth + 40)
    OutputGraphics.DrawLine(CoordinatesPen, OriginX + 5, OriginY +
GraphicsDepth + 30, OriginX, OriginY + GraphicsDepth + 40)
    OutputGraphics.DrawString("z", font, brush, OriginX - 27, OriginY +
GraphicsDepth + 25, TextFormat)

    OutputGraphics.DrawString("( 0,0 )", OriginFont, brush, OriginX - 35,
OriginY - 40 + 25, TextFormat)

    OutputGraphics.DrawLine(AxisPen, OriginX - 75, OriginY - 45, OriginX +
GraphicsWidth, OriginY - 45)
    OutputGraphics.DrawLine(AxisPen, OriginX - 75, OriginY - 45, OriginX -
75, OriginY + GraphicsDepth)

    Dim Marker As Single

    Do While Marker <= WidthW
        OutputGraphics.DrawLine(AxisPen, OriginX + CInt(GraphicsScale *
Marker), OriginY - 50, OriginX + CInt(GraphicsScale * Marker), OriginY - 40)
        Marker = i * Max(WidthW, DepthD) / 10
        i += 1
    Loop
    Marker = 0
    i = 1
    Do While Marker <= DepthD
        OutputGraphics.DrawLine(AxisPen, OriginX - 80, OriginY +
CInt(GraphicsScale * Marker), OriginX - 70, OriginY + CInt(GraphicsScale * Marker))
        Marker = i * Max(WidthW, DepthD) / 10
        i += 1
    Loop
    Marker = 0
    Do While Marker <= WidthW
        OutputGraphics.DrawString(Marker, OriginFont, brush, OriginX +
GraphicsScale * Marker - 7, OriginY - 70, Centralformat)
        Marker += (Max(DepthD, WidthW) / 5)
    Loop
    Marker = 0
```

```
        Do While Marker <= DepthD
            OutputGraphics.DrawString(Marker, OriginFont, brush, OriginX - 85,
OriginY + GraphicsScale * Marker - 10, RightAlignFormat)
            Marker += (Max(DepthD, WidthW) / 5)
        Loop

        ReadyToRefresh = True

    End If
Catch ex As Exception

End Try

End Sub

Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
    Me.Cursor = Cursors.WaitCursor
    Dim i As Integer = 0
    Dim OutputTableTitles() As String

    ReDim Top25Percent(0)
    ReDim UpperMiddle25Percent(0)
    ReDim LowerMiddle25Percent(0)
    ReDim Least25Percent(0)
    ReDim ZeroPercentList(0)
    ReDim TensileList(0)

    Try
        CheckBox2.Checked = False

        Select Case ComboBox1.SelectedIndex
            Case 0
                'Soil Matrix Selected
                If PlaneStrainAnalysis Then
                    OutputTableTitles = {"Node", "Settlement", "Xo", "Zo", "X f",
"Z f"}
                Else
                    OutputTableTitles = {"Node", "Settlement", "Ro", "Zo", "R f",
"Z f"}
                End If

                DataGridView1.DataSource = BuildOutputTable(GlobalDisplacements,
OutputTableTitles, 1)

                ReadyToRefresh = False
                CheckBox1.Checked = True
                CheckBox2.Checked = False
                MaximumValue = LargestDeformation
                Label4.Text = "Displacement and Coordinates of Nodes"
                'TextBox1.Text = DeformationOutput
                Label2.Text = "Maximum Deformation(m)= " &
CStr(Format(Abs(MaximumValue), "0.000"))
                Refresh()

            Case 1
                'X Stress Selected
                If PlaneStrainAnalysis Then
                    OutputTableTitles = {"Element", "X Stress (kN/m²)", "Centroid-
X", "Centroid- Z", "Node- i", "Node- J", "Node- m"}
                    Label4.Text = "Elemental Stresses (X direction) and
Coordinates"
                    LegendGroupBox.Text = "Magnitudes of X-Stress"
                Else
```



```
        OutputTableTitles = {"Element", "r stress (kN/m2)", "Centroid-  
r", "Centroid- z", "Node- i", "Node- j", "Node- m"}  
        Label4.Text = "Elemental Stresses (r direction) and  
Coordinates"  
        LegendGroupBox.Text = "Magnitudes of r-stress"  
    End If  
  
    DataGridView1.DataSource = BuildOutputTable(HorizontalStress,  
OutputTableTitles, 2)  
  
    ReadyToRefresh = False  
    CheckBox1.Checked = False  
    CheckBox2.Checked = False  
    MaximumValue = MaxHorizontalStress  
    RankProcedure(HorizontalStress)  
    'TextBox1.Text = HorizontalStressOutput  
    Label2.Text = "Maximum Stress(kN/m2)= " &  
CStr(Format(Abs(MaximumValue), "0.000"))  
    OutputUnit = " kN/m2"  
    Refresh()  
  
    Case 2 'Y or 0 Stress Selected  
        If PlaneStrainAnalysis Then  
            OutputTableTitles = {"Element", "Y Stress (kN/m2)", "Centroid-  
X", "Centroid- Z", "Node- i", "Node- j", "Node- m"}  
            Label4.Text = "Elemental Stresses (Y direction) and  
Coordinates"  
            LegendGroupBox.Text = "Magnitudes of Y-Stress"  
        Else  
            OutputTableTitles = {"Element", "0 stress (kN/m2)", "Centroid-  
r", "Centroid- z", "Node- i", "Node- j", "Node- m"}  
            Label4.Text = "Elemental Stresses (0 direction) and  
Coordinates"  
            LegendGroupBox.Text = "Magnitudes of 0-stress"  
        End If  
  
        DataGridView1.DataSource = BuildOutputTable(StressY,  
OutputTableTitles, 2)  
  
        ReadyToRefresh = False  
        CheckBox1.Checked = False  
        CheckBox2.Checked = False  
        MaximumValue = MaxYstress  
        RankProcedure(StressY)  
        'TextBox1.Text = VerticalStressOutput  
        Label2.Text = "Maximum Stress(kN/m2)= " &  
CStr(Format(Abs(MaximumValue), "0.000"))  
        OutputUnit = " kN/m2"  
        Refresh()  
  
        Case 3 'Z Stress Selected  
            If PlaneStrainAnalysis Then  
                OutputTableTitles = {"Element", "Z Stress (kN/m2)", "Centroid-  
X", "Centroid- Z", "Node- i", "Node- j", "Node- m"}  
            Else  
                OutputTableTitles = {"Element", "Z Stress (kN/m2)", "Centroid-  
r", "Centroid- z", "Node- i", "Node- j", "Node- m"}  
            End If  
            Label4.Text = "Elemental Stresses (Z direction) and Coordinates"  
            DataGridView1.DataSource = BuildOutputTable(VerticalStress,  
OutputTableTitles, 2)
```



```
ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxVerticalStress
RankProcedure(VerticalStress)
'TextBox1.Text = VerticalStressOutput
Label2.Text = "Maximum Stress(kN/m²)= " &
CStr(Format(Abs(MaximumValue), "0.000"))
LegendGroupBox.Text = "Magnitudes of Z-Stress"
OutputUnit = " kN/m²"
Refresh()

Case 4      'Shear Stress Selected
If PlaneStrainAnalysis Then
    OutputTableTitles = {"Element", "Shear Stress (kN/m²)",
"Centroid- X", "Centroid- Z", "Node- i", "Node- J", "Node- m"}
Else
    OutputTableTitles = {"Element", "Shear Stress (kN/m²)",
"Centroid- r", "Centroid- z", "Node- i", "Node- J", "Node- m"}
End If
DataGridView1.DataSource = BuildOutputTable(ShearStress,
OutputTableTitles, 2)

ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxShearStress
RankProcedure(AbsoluteShearStress)
Label4.Text = "Elemental Shear Stresses" & vbCrLf & "and
Coordinates"
'TextBox1.Text = ShearStressOutput
Label2.Text = "Maximum Stress(kN/m²)= " &
CStr(Format(Abs(MaximumValue), "0.000"))
LegendGroupBox.Text = "Magnitudes of Shear Stress (Absolute
Value)"
OutputUnit = " kN/m²"
Refresh()

Case 5      'X or r Strain Selected
If PlaneStrainAnalysis Then
    OutputTableTitles = {"Element", "X Strain", "Centroid- X",
"Centroid- Z", "Node- i", "Node- J", "Node- m"}
Label4.Text = "Elemental Strains (X direction) and
Coordinates"
LegendGroupBox.Text = "Magnitudes of X-Strain"
Else
    OutputTableTitles = {"Element", "r strain", "Centroid- r",
"Centroid- z", "Node- i", "Node- J", "Node- m"}
Label4.Text = "Elemental Strains (r direction) and
Coordinates"
LegendGroupBox.Text = "Magnitudes of r-strain"
End If

DataGridView1.DataSource = BuildOutputTable(HorizontalStrain,
OutputTableTitles, 2)

ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxHorizontalStrain
RankProcedure(HorizontalStrain)
'TextBox1.Text = HorizontalStrainOutput
```

```
Label2.Text = "Maximum Strain= " & CStr(Format(Abs(MaximumValue),
"0.000"))

OutputUnit = " "
Refresh()

Case 6      'Y or  $\theta$  Strain Selected
If PlaneStrainAnalysis Then
    OutputTableTitles = {"Element", "Y Strain ", "Centroid- X",
"Centroid- Z", "Node- i", "Node- J", "Node- m"}
    Label4.Text = CStr("Elemental Strains (Y Direction)" & vbCrLf
& "and Coordinates")
    LegendGroupBox.Text = "Magnitudes of Y-Strain"
Else
    OutputTableTitles = {"Element", " $\theta$  strain ", "Centroid- r",
"Centroid- z", "Node- i", "Node- J", "Node- m"}
    Label4.Text = CStr("Elemental Strains ( $\theta$  Direction)" & vbCrLf
& "and Coordinates")
    LegendGroupBox.Text = "Magnitudes of  $\theta$ -strain"
End If
DataGridView1.DataSource = BuildOutputTable(StrainY,
OutputTableTitles, 2)

ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxVerticalStrain
RankProcedure(StrainY)
'TextBox1.Text = VerticalStrainOutput
Label2.Text = "Maximum Strain= " & CStr(Format(Abs(MaximumValue),
"0.000"))

OutputUnit = " "
Refresh()

Case 7      'Z Strain Selected
If PlaneStrainAnalysis Then
    OutputTableTitles = {"Element", "Z Strain ", "Centroid- X",
"Centroid- Z", "Node- i", "Node- J", "Node- m"}
Else
    OutputTableTitles = {"Element", "Z Strain ", "Centroid- r",
"Centroid- z", "Node- i", "Node- J", "Node- m"}
End If
DataGridView1.DataSource = BuildOutputTable(VerticalStrain,
OutputTableTitles, 2)

ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxVerticalStrain
RankProcedure(VerticalStrain)
Label4.Text = CStr("Elemental Strains (Z direction)" & vbCrLf &
"and Coordinates")
'TextBox1.Text = VerticalStrainOutput
Label2.Text = "Maximum Strain= " & CStr(Format(Abs(MaximumValue),
"0.000"))

LegendGroupBox.Text = "Magnitudes of Z-Strain"
OutputUnit = " "
Refresh()

Case 8      'Shear Strain Selected
If PlaneStrainAnalysis Then
    OutputTableTitles = {"Element", "Shear Strain ", "Centroid-
X", "Centroid- Z", "Node- i", "Node- J", "Node- m"}
```

```

Else
    OutputTableTitles = {"Element", "Shear Strain ", "Centroid-
r", "Centroid- Z", "Node- i", "Node- J", "Node- m"}
End If
DataGridView1.DataSource = BuildOutputTable(ShearStrain,
OutputTableTitles, 2)

ReadyToRefresh = False
CheckBox1.Checked = False
CheckBox2.Checked = False
MaximumValue = MaxShearStrain
RankProcedure(AbsoluteShearStrain)
Label4.Text = CStr("Elemental Shear Strains" & vbCrLf & "and
Coordinates")

'TextBox1.Text = ShearStrainOutput
Label2.Text = "Maximum Strain= " & CStr(Format(Abs(MaximumValue),
"0.000"))

LegendGroupBox.Text = "Magnitudes of Shear Strain (Absolute
Value)"

OutputUnit = " "
Refresh()
End Select

i = 0
Dim OutputColumn As DataGridViewColumn
For Each OutputColumn In DataGridView1.Columns
    'Me.DataGridView1.Columns(i).SortMode =
DataGridViewColumnSortMode.NotSortable
    OutputColumn.SortMode = DataGridViewColumnSortMode.NotSortable
    i = i + 1
Next

Me.Cursor = Cursors.Default
Catch ex As Exception

End Try

End Sub

Sub RankProcedure(ByVal ParameterMatrix)
    Dim i As Short
    HighestOccurance = 0

    For i = 0 To UBound(ParameterMatrix)
        If Round(ParameterMatrix(i), 3) > Round(0.75 * MaximumValue, 3) Then
            ReDim Preserve Top25Percent(UBound(Top25Percent) + 1)
            Top25Percent(UBound(Top25Percent)) = i
            If UBound(Top25Percent) > HighestOccurance Then
                HighestOccurance = UBound(Top25Percent)
                LeadingCategory = 1
            End If
        ElseIf Round(ParameterMatrix(i), 3) > Round(0.5 * MaximumValue, 3) Then
            ReDim Preserve UpperMiddle25Percent(UBound(UpperMiddle25Percent) + 1)
            UpperMiddle25Percent(UBound(UpperMiddle25Percent)) = i
            If UBound(UpperMiddle25Percent) > HighestOccurance Then
                HighestOccurance = UBound(UpperMiddle25Percent)
                LeadingCategory = 2
            End If
        ElseIf Round(ParameterMatrix(i), 3) > Round(0.25 * MaximumValue, 3) Then
            ReDim Preserve LowerMiddle25Percent(UBound(LowerMiddle25Percent) + 1)
            LowerMiddle25Percent(UBound(LowerMiddle25Percent)) = i

```

```

        If UBound(LowerMiddle25Percent) > HighestOccurance Then
            HighestOccurance = UBound(LowerMiddle25Percent)
            LeadingCategory = 3
        End If
    ElseIf Round(ParameterMatrix(i), 3) > Round(0.005 * MaximumValue, 3) Then
        ReDim Preserve Least25Percent(UBound(Least25Percent) + 1)
        Least25Percent(UBound(Least25Percent)) = i
        If UBound(Least25Percent) > HighestOccurance Then
            HighestOccurance = UBound(Least25Percent)
            LeadingCategory = 4
        End If
    ElseIf Round(ParameterMatrix(i), 3) >= Round(-0.005 * MaximumValue, 3)
Then
        ReDim Preserve ZeroPercentList(UBound(ZeroPercentList) + 1)
        ZeroPercentList(UBound(ZeroPercentList)) = i
        If UBound(ZeroPercentList) > HighestOccurance Then
            HighestOccurance = UBound(ZeroPercentList)
            LeadingCategory = 5
        End If
    Else
        ReDim Preserve TensileList(UBound(TensileList) + 1)
        TensileList(UBound(TensileList)) = i
        If UBound(TensileList) > HighestOccurance Then
            HighestOccurance = UBound(TensileList)
            LeadingCategory = 6
        End If
    End If
End If
Next
End Sub
Sub DrawUndeformedMesh()
    Dim MeshPen As New Pen(Color.Gray)
    Dim counter As Integer
    Dim MeshDirection As Byte
    Dim i As Short
    Dim j As UShort
    Dim Element As UInteger = 0
    Dim UndeformedMeshGraphics As Graphics

    Try
        UndeformedMeshGraphics = Me.CreateGraphics

        'Previous method; connecting each node of each element involving redundant
lines;
        'For Element = 0 To UBound(ElementNodes)
        'OutputGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 1))
        'OutputGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 1), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 1))
        'OutputGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 2), 1), OriginX + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 0), OriginY + GraphicsScale *
NodalCoordinates(ElementNodes(Element, 0), 1))
        'Next

        'Drawing mesh by using drawlines command (avoids redundant lines)

```

```

        For j = 1 To UBound(VerticalGridMatrix, 2) - 1      'Draw vertical grid
lines
            UndeformedMeshGraphics.DrawLine(MeshPen, OriginX + GraphicsScale *
VerticalGridMatrix(0, j), OriginY, OriginX + GraphicsScale * VerticalGridMatrix(0, j),
OriginY + GraphicsScale * DepthD)
        Next
        For i = 0 To UBound(HorizontalGridMatrix) - 1      'Draw horizontal grid
lines
            UndeformedMeshGraphics.DrawLine(MeshPen, OriginX, OriginY +
GraphicsScale * HorizontalGridMatrix(i), OriginX + GraphicsScale * WidthW, OriginY +
GraphicsScale * HorizontalGridMatrix(i))
        Next

        'Drawing the Criss Crossing mesh lines Undeformed
        counter = 0
        MeshDirection = 1
        MeshGraphicPoints = {New Point(OriginX, OriginY)}
        For j = 0 To UBound(VerticalGridMatrix, 2) - 1      'X-coordinates
            For i = 1 To UBound(HorizontalGridMatrix)      'Y-coordinates-
downwards
                counter = counter + 1
                ReDim Preserve MeshGraphicPoints(counter)
                MeshGraphicPoints(counter) = New Point(OriginX + GraphicsScale *
VerticalGridMatrix(0, j + MeshDirection), OriginY + GraphicsScale *
HorizontalGridMatrix(i))
                MeshDirection = Abs(MeshDirection - 1)
            Next
            For i = UBound(HorizontalGridMatrix) To 0 Step -1      'Y-
coordinates- upwards
                counter = counter + 1
                ReDim Preserve MeshGraphicPoints(counter)
                MeshGraphicPoints(counter) = New Point(OriginX + GraphicsScale *
VerticalGridMatrix(0, j + MeshDirection), OriginY + GraphicsScale *
HorizontalGridMatrix(i))
                MeshDirection = Abs(MeshDirection - 1)
            Next
            MeshDirection = Abs(MeshDirection - 1)
        Next
        UndeformedMeshGraphics.DrawLines(MeshPen, MeshGraphicPoints)
    Catch ex As Exception

    End Try

End Sub
Sub DrawDeformedMesh()
    Dim i As Short
    Dim j As UShort
    Dim counter As Integer
    Dim Element As UInteger = 0
    Dim MeshDirection As Byte
    Dim DeformedMeshPen As New Pen(Color.Red)
    Dim DeformedMeshGraphics As Graphics

    Try
        DeformedMeshGraphics = Me.CreateGraphics

        'Drawing the deformed mesh using DrawLines method
        counter = 1
        MeshDirection = 0

```

```

        DeformedMeshGraphicPoints = {New Point(OriginX + GraphicsScale *
ScaledDeformedCoordinates(0, 0), OriginY + GraphicsScale *
ScaledDeformedCoordinates(0, 1))}
        For j = 0 To UBound(HorizontalGridMatrix) - 1      'along the Vertical
            For i = 0 To UBound(VerticalGridMatrix, 2) - 1  'along the
Horizontal
                ReDim Preserve
DeformedMeshGraphicPoints(UBound(DeformedMeshGraphicPoints) + 7)
                DeformedMeshGraphicPoints(counter) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 1, 1), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 1, 1), 1))
                DeformedMeshGraphicPoints(counter + 1) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 1, 2), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 1, 2), 1))
                DeformedMeshGraphicPoints(counter + 2) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 0), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 0), 1))
                DeformedMeshGraphicPoints(counter + 3) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 1), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 1), 1))
                DeformedMeshGraphicPoints(counter + 4) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 2), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 0, 2), 1))
                DeformedMeshGraphicPoints(counter + 5) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 3, 1), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 3, 1), 1))
                DeformedMeshGraphicPoints(counter + 6) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 3, 0), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 3, 0), 1))
                Element = Element + 4
                counter = counter + 7
            Next
            If j < UBound(HorizontalGridMatrix) - 1 Then
                counter = counter - 1
                ReDim Preserve
DeformedMeshGraphicPoints(UBound(DeformedMeshGraphicPoints) +
UBound(VerticalGridMatrix, 2) - 1)
                For i = UBound(VerticalGridMatrix, 2) - 1 To 0 Step -1
'Returning to the lefternmost edge
                    Element = Element - 4
                    DeformedMeshGraphicPoints(counter) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 2, 2), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(ElementNodes(Element + 2, 2), 1))
                    counter = counter + 1
                Next
                Element = Element + 4 * UBound(VerticalGridMatrix, 2)
            Else
                ReDim Preserve
DeformedMeshGraphicPoints(UBound(DeformedMeshGraphicPoints) + 1)
                DeformedMeshGraphicPoints(counter) = New Point(OriginX +
GraphicsScale * ScaledDeformedCoordinates(UBound(VerticalGridMatrix, 2), 0), OriginY +
GraphicsScale * ScaledDeformedCoordinates(UBound(VerticalGridMatrix, 2), 1))
            End If
        Next
        DeformedMeshGraphics.DrawLine(DeformedMeshPen, DeformedMeshGraphicPoints)

    Catch ex As Exception

    End Try

End Sub

```

```
Private Function BuildOutputTable(ByVal OutputArray, ByVal LeadingColumnTitles,
ByVal OutputSetupType) As DataTable

    Dim i As Short
    Dim j As Short
    Dim OutputTable As New DataTable
    Dim TableRow As DataRow

    OutputTable.Columns.Add("ColumnZero")
    OutputTable.Columns.Add("ColumnOne")
    OutputTable.Columns.Add("ColumnTwo")
    OutputTable.Columns.Add("ColumnThree")
    OutputTable.Columns.Add("ColumnFour")
    OutputTable.Columns.Add("ColumnFive")

    If OutputSetupType = 1 Then
        For i = 0 To UBound(NodalCoordinates)
            OutputTable.Rows.Add()
            Next

            i = 0
            j = 1
            For Each TableRow In OutputTable.Rows
                TableRow!ColumnZero = i + 1
                TableRow!ColumnOne = Round(OutputArray(j), 6)
                TableRow!columnTwo = NodalCoordinates(i, 0)
                TableRow!columnThree = NodalCoordinates(i, 1)
                TableRow!columnFour = Round(NodalCoordinates(i, 0) +
GlobalDisplacements(j - 1), 3)
                TableRow!columnFive = Round(NodalCoordinates(i, 1) +
GlobalDisplacements(j), 3)
                i += 1
                j += 2
            Next
        Else
            OutputTable.Columns.Add("ColumnSix")
            For i = 0 To UBound(OutputArray)
                OutputTable.Rows.Add()
                Next

                i = 0
                For Each TableRow In OutputTable.Rows
                    TableRow!ColumnZero = i + 1
                    TableRow!ColumnOne = Round(OutputArray(i), 6)
                    TableRow!columnTwo = Round(ElementCentroids(i, 0), 3)
                    TableRow!columnThree = Round(ElementCentroids(i, 1), 3)
                    TableRow!columnFour = ElementNodes(i, 0) + 1
                    TableRow!columnFive = ElementNodes(i, 1) + 1
                    TableRow!ColumnSix = ElementNodes(i, 2) + 1
                    i += 1
                Next
            End If

            For i = 0 To UBound(LeadingColumnTitles)
                OutputTable.Columns(i).ColumnName = LeadingColumnTitles(i)
                OutputTable.Columns(i).AllowDBNull = True
            Next

            Return OutputTable

        End Function
```

```
Private Sub CopyToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CopyToolStripMenuItem.Click
    Clipboard.SetDataObject(Me.DataGridView1.GetClipboardContent())

    'Clipboard.SetDataObject(DataGridView1.SelectedCells)
End Sub

Private Sub CopyAllToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles CopyAllToolStripMenuItem.Click
    DataGridView1.SelectAll()
    Clipboard.SetDataObject(Me.DataGridView1.GetClipboardContent())
End Sub

Private Sub GraphicOutputToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GraphicOutputToolStripMenuItem.Click
    My.Forms.NumericOutputForm.Show()
End Sub

Private Sub GenerateReportToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GenerateReportToolStripMenuItem.Click
    Try
        SaveFileDialog1.Filter = "Text files(*.txt)|*.txt"
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            My.Forms.GenerateReport.Show()
        End If
    Catch ex As Exception

    End Try

End Sub

Private Sub PrintToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles PrintToolStripMenuItem.Click
    Try
        CaptureScreen()
        PrintDocument1.Print()
    Catch ex As Exception

    End Try
End Sub

Private Sub CaptureScreen()

    Dim myGraphics As Graphics = Me.CreateGraphics()
    Dim s As Size = Me.Size

    FullGraphicDepth = Me.Size.Height - 50
    FullGraphicWidth = Me.Size.Width - 260

    s = New Size(FullGraphicWidth, FullGraphicDepth)

    memoryImage = New Bitmap(s.Width, s.Height, myGraphics)
    Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
    memoryGraphics.CopyFromScreen(Me.Location.X + 260, Me.Location.Y + 50, 0, 0,
s)
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
    Dim PrintableWidth As Single = e.MarginBounds.Width
    Dim PrintableHeight As Single = e.MarginBounds.Height
```



```
Dim rectDraw As RectangleF
'(e.MarginBounds.Left, e.MarginBounds.Top, e.MarginBounds.Width,
e.MarginBounds.Height)

Try
    If PrintableWidth / PrintableHeight > (FullGraphicWidth /
FullGraphicDepth) Then
        rectDraw = New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
FullGraphicWidth * e.MarginBounds.Height /
FullGraphicDepth, e.MarginBounds.Height)
    Else
        rectDraw = New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
e.MarginBounds.Width, FullGraphicDepth *
e.MarginBounds.Width / FullGraphicWidth)
    End If

    e.Graphics.DrawImage(memoryImage, rectDraw)
'e.Graphics.DrawImage(
Catch ex As Exception

End Try

End Sub

Private Sub PrintPreviewToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles PrintPreviewToolStripMenuItem.Click
Try
    CaptureScreen()
    PrintDocument1.DefaultPageSettings = PrintPageSettings

    PrintPreviewDialog1.Document = PrintDocument1
    PrintPreviewDialog1.ShowDialog()
Catch ex As Exception
'Display error message
MessageBox.Show(ex.Message)
End Try
End Sub

End Class
```

### \*\*Codes Used for Managing the Report Generation

```
Imports System.IO
Imports System.Math
Imports System.Data

Public Class GenerateReport

    Private Sub GenerateReport_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Shown

        Try

            Me.Cursor = Cursors.WaitCursor

            Dim i As Integer
            Dim j As UShort

            ReportBegin = TimeString
            ReportStreamToWrite = New StreamWriter(SaveFileDialog1.FileName)
```

```
OutputReportText = vbCrLf & vbTab & vbTab & "Finite Element Software for  
Computing Stress & Deformation in Layered Soils" & vbCrLf & vbCrLf & vbTab & vbTab & "  
*****" & vbCrLf  
OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab & vbTab &  
vbTab & "Analysis Report" & vbCrLf  
OutputReportText = OutputReportText & vbTab & vbTab & vbTab & vbTab &  
vbTab & vbTab & vbTab & vbTab & vbTab & DateString & " " & TimeString & vbCrLf &  
vbCrLf & vbCrLf  
  
OutputReportText = OutputReportText & " Given:" & vbCrLf  
OutputReportText = OutputReportText & vbTab & "Soil Matrix:-" & vbTab &  
"Width= " & WidthW & "m" & vbTab & "Total Depth= " & DepthD & "m" & vbCrLf & vbCrLf  
  
For i = 0 To UBound(LayerThicknesses, 2)  
    OutputReportText = OutputReportText & vbTab & " Layer-" & i + 1 &  
vbTab & "Thickness= " & LayerThicknesses(0, i) & "m" & vbTab & "Modulus of Elasticity,  
E=" & ElasticModulus(i, 0) & "kN/m2" & vbTab & "Poisson's Ratio= " & PoissonsRatio(i,  
0) & vbCrLf  
Next  
OutputReportText = OutputReportText & vbCrLf & vbCrLf & vbTab & "Load  
Cases:-" & vbCrLf  
For i = 0 To UBound(UDLMatrix)  
    OutputReportText = OutputReportText & vbTab & " Uniform Load-" & i +  
1 & ":" & vbTab & UDLMatrix(i, 0) & "kN/m" & " acting from X= " & UDLMatrix(i, 1) &  
"m" & " to X= " & UDLMatrix(i, 2) & "m, at " & UDLMatrix(i, 3) & "m depth" &  
vbCrLf  
Next  
OutputReportText = OutputReportText & vbCrLf  
For i = 0 To UBound(PointLoadMatrix)  
    OutputReportText = OutputReportText & vbTab & " Point Load-" & i + 1  
& ":" & vbTab & PointLoadMatrix(i, 0) & "kN" & vbTab & "acting on X= " &  
PointLoadMatrix(i, 1) & "m" & " at " & PointLoadMatrix(i, 2) & "m depth" & vbCrLf  
Next  
  
OutputReportText = OutputReportText & vbCrLf  
If LeftEndVerticallyRestrained Then  
    OutputReportText = OutputReportText & vbCrLf & vbTab & "-Left boundary  
is Vertically Restrained" & vbCrLf  
Else  
    OutputReportText = OutputReportText & vbCrLf & vbTab & "-Left boundary  
is Vertically Unrestrained" & vbCrLf  
End If  
If RightEndVerticallyRestrained Then  
    OutputReportText = OutputReportText & vbTab & "-Right boundary is  
Vertically Restrained" & vbCrLf  
Else  
    OutputReportText = OutputReportText & vbTab & "-Right boundary is  
Vertically Unrestrained" & vbCrLf  
End If  
  
OutputReportText = OutputReportText & vbCrLf  
If PlaneStrainAnalysis Then  
    OutputReportText = OutputReportText & vbCrLf & vbTab & "The problem  
type is- Plane Strain" & vbCrLf  
Else  
    OutputReportText = OutputReportText & vbCrLf & vbTab & "The problem  
type is- Axisymmetric" & vbCrLf  
End If  
  
OutputReportText = OutputReportText & vbCrLf
```

```

OutputReportText = OutputReportText & vbCrLf & " Finite Element Method
Steps:" & vbCrLf & vbCrLf
OutputReportText = OutputReportText & vbTab & "Maximum Mesh Dimension
Used= " & MaximumMeshDimension & "m" & vbCrLf
OutputReportText = OutputReportText & vbTab & "Total Number of Nodes = " &
UBound(NodalCoordinates) + 1 & vbCrLf
OutputReportText = OutputReportText & vbCrLf & vbTab & "Nodal Coordinates
and Degrees of Freedom:-" & vbCrLf
ReportStreamToWrite.Write(OutputReportText)

For i = 0 To UBound(NodalCoordinates)
    If PlaneStrainAnalysis Then
        OutputReportText = vbTab & " Node " & i + 1 & vbTab & "X= " &
(NodalCoordinates(i, 0).ToString("f4")) & vbTab & " Z= " & (NodalCoordinates(i,
1).ToString("f4")) & " " & vbTab & "DOF_Id {x,z}= { " & 2 * i + 1 & " , " & 2 * (i
+ 1) & " }" & vbCrLf
    Else
        OutputReportText = vbTab & " Node " & i + 1 & vbTab & "r= " &
(NodalCoordinates(i, 0).ToString("f4")) & vbTab & " z= " & (NodalCoordinates(i,
1).ToString("f4")) & " " & vbTab & "DOF_Id {r,z}= { " & 2 * i + 1 & " , " & 2 * (i
+ 1) & " }" & vbCrLf
    End If
    ReportStreamToWrite.Write(OutputReportText)
Next
OutputReportText = vbCrLf & vbCrLf & vbCrLf & vbTab & "Total Number of
Elements Formed= " & UBound(ElementNodes) + 1 & vbCrLf
OutputReportText = OutputReportText & vbCrLf & vbTab & "Element-Node
Connectivity:-" & vbCrLf
ReportStreamToWrite.Write(OutputReportText)

For i = 0 To UBound(ElementNodes)
    OutputReportText = vbTab & " Element " & i + 1 & vbTab & " Nodes= {"
& ElementNodes(i, 0) + 1 & " , " & ElementNodes(i, 1) + 1 & " , " & ElementNodes(i, 2) + 1
& " }" & vbCrLf
    ReportStreamToWrite.Write(OutputReportText)
Next

OutputReportText = vbCrLf & vbCrLf & vbCrLf & vbCrLf
OutputReportText = OutputReportText & vbTab & "Assembly of elemental
stiffnesses" & vbCrLf & vbCrLf

If PlaneStrainAnalysis Then
    OutputReportText = OutputReportText & vbTab & " - for Plane Strain
problems the elemental stiffness matrix is given by;" & vbCrLf
    OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab & vbTab &
" T" & vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & "[k] = tA [B]
[D][B]" & vbCrLf
    OutputReportText = OutputReportText & vbCrLf & vbTab & "where;" &
vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & " t = thickness
(for the 2 dimensional case of plane strain problems, t=1 unit)" & vbCrLf & vbCrLf
    ReportStreamToWrite.Write(OutputReportText)

    OutputReportText = "
r
γ" &
vbCrLf
    OutputReportText = OutputReportText & "
| βi 0 βj 0
βm 0 | where the Beta and Gamma variables represent orthogonal distances
between the nodes i,j, and m as follows" & vbCrLf

```

```

        OutputReportText = OutputReportText & "          1|
    |" & vbCrLf
        OutputReportText = OutputReportText & "          [B] = —| 0  γi  0
γj  0  γm |          βi = Yj-Ym  βj = Ym-Yi  βk = Yi-Yj" & vbCrLf
        OutputReportText = OutputReportText & "          2A|
    |" & vbCrLf
        OutputReportText = OutputReportText & "          | 0  0  0  0
0  0 |  γi = Xm-Xj  γj = Xi-Xm  γk = Xj-Xi" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          | γi  βi  γj  βj
γm  βm |" & vbCrLf
        OutputReportText = OutputReportText & "          L
    |" & vbTab & "          A= area of the element " & vbCrLf
Else
    OutputReportText = OutputReportText & vbTab & " - for axisymmetric
problems the elemental stiffness matrix is approximated by;" & vbCrLf
    OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab & vbTab &
" _ _ T _" & vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & "[k] = 2π.r.A
[B] [D][B]" & vbCrLf
    OutputReportText = OutputReportText & vbCrLf & vbTab & "where;" &
vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & " _" & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & " r = radius
(distance from Axis of symmetry to centroid of element)" & vbCrLf & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & " A = area of
the element" & vbCrLf & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & " _" & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & " [B] = the
value of the strain-displacement matrix at the centroidal point" & vbCrLf & vbCrLf
        ReportStreamToWrite.Write(OutputReportText)

    OutputReportText = "          r
γ" & vbCrLf
        OutputReportText = OutputReportText & "          |  βi
0  βj  0  βm  0 |" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          |  0
γi  0  γj  0  γm |" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          |
        ai+βi.r+γi.z  0  aj+βj.r+γj.z  0  am+βm.r+γm.z  0 |" & vbCrLf
        OutputReportText = OutputReportText & "          [B] = —|
    |" & vbCrLf
        OutputReportText = OutputReportText & "          2A |  r
r          r |" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          |
    |" & vbCrLf
        OutputReportText = OutputReportText & "          |  γi
βi  γj  βj  γm  βm |" & vbCrLf
        OutputReportText = OutputReportText & "          L
    |" & vbTab & vbCrLf
        OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab & "
where; " & vbCrLf & vbCrLf

```

```

        OutputReportText = OutputReportText & vbTab & vbTab & vbTab & "       $\alpha_i$ 
= rj.Zm-rm.Zj  $\alpha_j$  = rm.Zi-ri.Zm       $\alpha_k$  = ri.Zj-rj.Zi" & vbCrLf & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & vbTab & "       $\beta_i$ 
= Yj-Ym       $\beta_j$  = Ym-Yi       $\beta_k$  = Yi-Yj" & vbCrLf & vbCrLf
        OutputReportText = OutputReportText & vbTab & vbTab & vbTab & "       $\gamma_i$ 
= Xm-Xj       $\gamma_j$  = Xi-Xm       $\gamma_k$  = Xj-Xi" & vbCrLf & vbCrLf

    End If
    ReportStreamToWrite.Write(OutputReportText)

    OutputReportText = vbCrLf & vbCrLf & vbTab & vbTab & vbTab & "the Stress-
Strain matrix (Constitutive martix), [D], is given by;" & vbCrLf & vbCrLf
    OutputReportText = OutputReportText & "       $\begin{bmatrix} 1-v & v & 0 \\ v & 1-v & 0 \\ 0 & 0 & 0.5-v \end{bmatrix}$  where " & vbCrLf
    OutputReportText = OutputReportText & "       $E = \text{modulus of elasticity for the respective element}$  &
vbCrLf
    OutputReportText = OutputReportText & "      [D] =  $\frac{E}{(1+v)(1-2v)}$  | v
1-v      v      0      " & vbCrLf
    OutputReportText = OutputReportText & "      v= Poisson's ratio for the respective element" & vbCrLf
    OutputReportText = OutputReportText & "       $\begin{bmatrix} 1-v & 0 \\ v & 1-v & 0 \\ 0 & 0 & 0.5-v \end{bmatrix}$  " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    OutputReportText = OutputReportText & "      " & vbCrLf
    ReportStreamToWrite.Write(OutputReportText)

    '
    Dim Element As UInteger
    Dim LocalDisplacements(5) As Single

    Dim ElementStiffnessMatrix(5, 5) As Double
    'Dim GlobalStiffnessMatrix(UBound(FixityStatus), UBound(FixityStatus)) As
Double
    Dim ArrayProduct(5, 3) As Double
    Dim ElementArea As Double
    Dim ElementalElasticModulus As Single
    Dim ElementalPoissonsRatio As Single

    For Element = 0 To UBound(ElementNodes)

        ElementArea = 0.5 * Abs((NodalCoordinates(ElementNodes(Element, 0), 0)
* NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 1), 0) * NodalCoordinates(ElementNodes(Element,
2), 1) + NodalCoordinates(ElementNodes(Element, 2), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1)) -
(NodalCoordinates(ElementNodes(Element, 1), 0) *
NodalCoordinates(ElementNodes(Element, 0), 1) + NodalCoordinates(ElementNodes(Element,
2), 0) * NodalCoordinates(ElementNodes(Element, 1), 1) +
NodalCoordinates(ElementNodes(Element, 0), 0) * NodalCoordinates(ElementNodes(Element,
2), 1)))

        ElementalElasticModulus = ElasticModulus(0, 0)
        ElementalPoissonsRatio = PoissonsRatio(0, 0)

        For i = 0 To NumberSoilLayers - 1

```

```

        If Element >= PoissonsRatio(i, 1) Then
            ElementalElasticModulus = ElasticModulus(i, 0)
            ElementalPoissonsRatio = PoissonsRatio(i, 0)
        End If
    Next

    For i = 0 To 5
        For j = 0 To 3
            ArrayProduct(i, j) = BTranspose(i, 0, Element) * DMatrix(0, j,
Element) + BTranspose(i, 1, Element) * DMatrix(1, j, Element) + BTranspose(i, 2,
Element) * DMatrix(2, j, Element) + BTranspose(i, 3, Element) * DMatrix(3, j, Element)
        Next
    Next

    'Calculation of Elemental Stiffness
    For i = 0 To 5
        For j = 0 To 5
            ElementStiffnessMatrix(i, j) = ElementArea *
(((ArrayProduct(i, 0)) * (BMatrix3D(0, j, Element))) + ((ArrayProduct(i, 1)) *
(BMatrix3D(1, j, Element))) + ((ArrayProduct(i, 2)) * (BMatrix3D(2, j, Element))) +
((ArrayProduct(i, 3)) * (BMatrix3D(3, j, Element))))
            If PlaneStrainAnalysis = False Then
                ElementStiffnessMatrix(i, j) = ElementStiffnessMatrix(i,
j) * 2 * PI * ElementCentroids(Element, 0)
            End If
        Next
    Next

    OutputReportText = vbCrLf & vbTab & "Element " & Element + 1 & vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & "      E= " &
ElementalElasticModulus & " kN/m2" & vbTab & "v= " & ElementalPoissonsRatio & "      A= " &
ElementArea & " m2" & "      Elemental Nodes= {" & ElementNodes(Element, 0) + 1 & "," &
ElementNodes(Element, 1) + 1 & "," & ElementNodes(Element, 2) + 1 & "}" & "
DOF_List= {" & 2 * ElementNodes(Element, 0) + 1 & "," & 2 * ElementNodes(Element, 0) +
2 & "," & 2 * ElementNodes(Element, 1) + 1 & "," & 2 * ElementNodes(Element, 1) + 2 &
"," & 2 * ElementNodes(Element, 2) + 1 & "," & 2 * ElementNodes(Element, 2) + 2 & "}" &
vbCrLf & vbCrLf

    'OutputReportText = OutputReportText & vbTab & vbTab & " [D]= " & "[[
" & Round(DMatrix(0, 0, Element), 5) & "      " & Round(DMatrix(0, 1, Element), 5) & "      0
" & "[ " & Round(DMatrix(1, 0, Element), 5) & "      " & Round(DMatrix(1, 1, Element),
5) & "      " & "0 ]" & "[ 0 0 " & Round(DMatrix(2, 2, Element), 5) & " ]]" & vbCrLf &
vbCrLf

    'OutputReportText = OutputReportText & vbTab & vbTab & " [B]= " & "[[
" & BMatrix3D(0, 0, Element) & " 0 " & BMatrix3D(0, 2, Element) & " 0 " &
BMatrix3D(0, 4, Element) & " 0 ]" & "[ " & 0 " & BMatrix3D(1, 1, Element) & " 0 " &
BMatrix3D(1, 3, Element) & " " & BMatrix3D(1, 5, Element) & " ]" & "[ " &
BMatrix3D(2, 0, Element) & " " & BMatrix3D(2, 1, Element) & " " & BMatrix3D(2, 2,
Element) & " " & BMatrix3D(2, 3, Element) & " " & BMatrix3D(2, 4, Element) & " " &
BMatrix3D(2, 5, Element) & " ]]" & vbCrLf

    OutputReportText = OutputReportText & vbTab & vbTab & " [D]= " & "[["
    For i = 0 To 3
        OutputReportText = OutputReportText & "[ "
        For j = 0 To 3
            OutputReportText = OutputReportText & Round(DMatrix(i, j,
Element), 5) & "      "
        Next
        OutputReportText = OutputReportText & "]"
    Next
    OutputReportText = OutputReportText & "]" & vbCrLf & vbCrLf

```

```
OutputReportText = OutputReportText & vbTab & vbTab & " [B]= " & "["
For i = 0 To 3
    OutputReportText = OutputReportText & "[ "
    For j = 0 To 5
        OutputReportText = OutputReportText & Round(BMatrix3D(i, j,
Element), 5) & " "
    Next
    OutputReportText = OutputReportText & "]" "
Next
OutputReportText = OutputReportText & "]" & vbCrLf & vbCrLf

OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab & "=>

[k]= "

OutputReportText = OutputReportText & " ["
For i = 0 To 5
    OutputReportText = OutputReportText & " [ "
    ReportStreamToWrite.Write(OutputReportText)

    For j = 0 To 5
        OutputReportText = Round(ElementStiffnessMatrix(i, j), 5) & "

        ReportStreamToWrite.Write(OutputReportText)
    Next
    OutputReportText = "]"
Next
OutputReportText = OutputReportText & " ] " & vbCrLf & vbCrLf
ReportStreamToWrite.Write(OutputReportText)

'
Next

OutputReportText = vbCrLf & vbCrLf & vbTab & "List of Unconstrained
Degrees of Freedom (with Unknown displacement values)" & vbCrLf & vbCrLf & vbTab &
vbTab & "{ "
ReportStreamToWrite.Write(OutputReportText)

DOF = 0
For i = 0 To UBound(FixityStatus)
    If FixityStatus(i) = 0 Then
        OutputReportText = i + 1 & " "
        If (DOF + 1) Mod 30 = 0 Then
            OutputReportText = OutputReportText & vbCrLf & vbTab & vbTab &

            " "

            End If
            ReportStreamToWrite.Write(OutputReportText)
            DOF = DOF + 1
        End If
    End If
Next

OutputReportText = "}" & vbCrLf & vbCrLf & vbCrLf & vbCrLf & vbTab & "The
Global Stiffness Matrix is assembled by superposing the elemental stiffness matrix
elements at the respective degrees of freedom." & vbCrLf & vbTab & "In order to solve
for the unknown displacement values, we use the Partitioned Global Stiffness matrix
formed by taking from the Global" & vbCrLf & vbTab & "Stiffness Matrix the rows &
columns corresponding to unconstrained degrees of freedom (whose displacement value is
not known)." & vbCrLf & vbCrLf & vbCrLf & vbCrLf
ReportStreamToWrite.Write(OutputReportText)

For i = 0 To DOF - 1
```

```
        OutputReportText = vbTab & "Partitioned Global Stiffness Matrix" &
vbTab & "Row " & i + 1 & " (DOF " & UnknownDOFList(i) + 1 & ")" & vbCrLf & vbCrLf
& vbTab & " " & "[[ "
        ReportStreamToWrite.Write(OutputReportText)

        For j = 0 To DOF - 1
            OutputReportText = Round(ShortStiffnessCopy(i, j), 5) & " "
            ReportStreamToWrite.Write(OutputReportText)
        Next

        OutputReportText = "]]" & vbCrLf & vbCrLf & vbCrLf
        ReportStreamToWrite.Write(OutputReportText)
    .
Next

    OutputReportText = vbCrLf & vbCrLf & vbCrLf & vbTab & "The partitioned
forces matrix, [F], corresponding to the unconstrained degrees of freedom is given
by;" & vbCrLf & vbCrLf & vbTab & vbTab & "[[ "
    ReportStreamToWrite.Write(OutputReportText)

    For i = 0 To DOF - 1
        OutputReportText = ForcesMatrix(UnknownDOFList(i)) & " "
        ReportStreamToWrite.Write(OutputReportText)
    Next

    ReportStreamToWrite.Write("]]" & vbCrLf & vbCrLf)
    .
    OutputReportText = vbCrLf & vbCrLf & vbTab & "From the Stiffness equation;
[F]=[K]{d} where [K] is the stiffness matrix" & vbCrLf & vbCrLf & vbTab & vbTab &
vbTab & vbTab & vbTab & vbTab & vbTab & "{d} is the displacement vector" & vbCrLf &
vbTab & vbTab & vbTab & vbTab & vbTab & "-1" & vbCrLf & vbTab & vbTab & vbTab & " =>
{d} = [K] [F]" & vbCrLf & vbCrLf & vbCrLf
    OutputReportText = OutputReportText & vbCrLf & vbCrLf & vbTab & "Solving
the matrix equation for the unknown global displacement values gives;" & vbCrLf &
vbCrLf
    ReportStreamToWrite.Write(OutputReportText)

    OutputReportText = vbTab & vbTab & "{d}= {"
    ReportStreamToWrite.Write(OutputReportText)
    For i = 0 To DOF - 1
        OutputReportText = Round(GlobalDisplacements(UnknownDOFList(i)), 5) &
" "
        ReportStreamToWrite.Write(OutputReportText)
    Next

    OutputReportText = "}" & vbCrLf & vbCrLf

    OutputReportText = OutputReportText & vbCrLf & vbCrLf & vbCrLf & vbTab &
"Calculation of Elemental Stresses and Strains" & vbCrLf & vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & "The elemental
displacement matrix is expressed as" & vbCrLf & vbCrLf & vbTab & vbTab &
"{d}={Ui,Vi,Uj,Vj,Uk,Vk} where U and V are displacements along the X and along the Y
respectively" & vbCrLf & vbCrLf
    OutputReportText = OutputReportText & vbTab & vbTab & vbTab & vbTab &
vbTab & vbTab & "their magnitudes are referred from the global displacement matrix
using the degree of freedom ID" & vbCrLf & vbCrLf & vbCrLf
    ReportStreamToWrite.Write(OutputReportText)
```



```

        For Element = 0 To UBound(ElementNodes)
            For i = 0 To 2
                For j = 0 To 1
                    'for every node
                    'for X direction & Y
                    direction
                    LocalDisplacements(2 * i + j) = GlobalDisplacements(2 *
ElementNodes(Element, i) + j)
                Next
            Next

            OutputReportText = vbTab & "Element " & Element + 1 & vbCrLf & vbCrLf
            OutputReportText = OutputReportText & vbTab & vbTab & vbTab &
"DOF_List= {" & 2 * ElementNodes(Element, 0) + 1 & "," & 2 * ElementNodes(Element, 0)
+ 2 & "," & 2 * ElementNodes(Element, 1) + 1 & "," & 2 * ElementNodes(Element, 1) + 2
& "," & 2 * ElementNodes(Element, 2) + 1 & "," & 2 * ElementNodes(Element, 2) + 2 &
"}" & vbTab & "=> {d}= {" & Round(LocalDisplacements(0), 5) & " " &
Round(LocalDisplacements(1), 5) & " " & Round(LocalDisplacements(2), 5) & " " &
Round(LocalDisplacements(3), 5) & " " & Round(LocalDisplacements(4), 5) & " " &
Round(LocalDisplacements(5), 5) & "}" & vbCrLf & vbCrLf

            OutputReportText = OutputReportText & vbTab & vbTab & " [D]= " & "["
            For i = 0 To 3
                OutputReportText = OutputReportText & "[ "
                For j = 0 To 3
                    OutputReportText = OutputReportText & Round(DMatrix(i, j,
Element), 5) & " "
                Next
                OutputReportText = OutputReportText & "]"
            Next
            OutputReportText = OutputReportText & "]" & vbCrLf & vbCrLf

            OutputReportText = OutputReportText & vbTab & vbTab & " [B]= " & "["
            For i = 0 To 3
                OutputReportText = OutputReportText & "[ "
                For j = 0 To 5
                    OutputReportText = OutputReportText & Round(BMatrix3D(i, j,
Element), 5) & " "
                Next
                OutputReportText = OutputReportText & "]"
            Next
            OutputReportText = OutputReportText & "]" & vbCrLf & vbCrLf & vbCrLf

            ' strain=[B]{d}

            OutputReportText = OutputReportText & vbTab & vbTab & " Strains: {ε}
=[B]{d} => {ε} = {" & Round(HorizontalStrain(Element), 5) & " " &
Round(StrainY(Element), 5) & " " & Round(VerticalStrain(Element), 5) & " " &
Round(ShearStrain(Element), 5) & "}" & vbCrLf & vbCrLf
            If PlaneStrainAnalysis Then
                OutputReportText = OutputReportText & vbTab & vbTab & vbTab &
vbTab & vbTab & vbTab & "εx = " & Round(HorizontalStrain(Element), 5) & vbTab & "εy =
" & Round(StrainY(Element), 5) & vbTab & "εz = " & Round(VerticalStrain(Element), 5) &
vbTab & "γxz = " & Round(ShearStrain(Element), 5) & vbCrLf & vbCrLf
            Else
                OutputReportText = OutputReportText & vbTab & vbTab & vbTab &
vbTab & vbTab & vbTab & "εr = " & Round(HorizontalStrain(Element), 5) & vbTab & "εθ =
" & Round(StrainY(Element), 5) & vbTab & "εz = " & Round(VerticalStrain(Element), 5) &
vbTab & "γrz = " & Round(ShearStrain(Element), 5) & vbCrLf & vbCrLf
            End If
            OutputReportText = OutputReportText & vbCrLf
        Next
    End Sub

```

```
' Stress=[D]{Strain}

    OutputReportText = OutputReportText & vbTab & vbTab & " Stresses: {σ}
=[D]{ε} => {σ} = {" & Round(HorizontalStress(Element), 5) & " " &
Round(StressY(Element), 5) & " " & Round(VerticalStress(Element), 5) & " " &
Round(ShearStress(Element), 5) & "}" & vbCrLf & vbCrLf
    If PlaneStrainAnalysis Then
        OutputReportText = OutputReportText & vbTab & vbTab & vbTab &
vbTab & vbTab & vbTab & "σx = " & Round(HorizontalStress(Element), 5) & vbTab & "σy =
" & Round(StressY(Element), 5) & vbTab & "σz = " & Round(VerticalStress(Element), 5) &
vbTab & "τxz = " & Round(ShearStress(Element), 5) & vbTab & "kN/m²" & vbCrLf & vbCrLf
    Else
        OutputReportText = OutputReportText & vbTab & vbTab & vbTab &
vbTab & vbTab & vbTab & "σr = " & Round(HorizontalStress(Element), 5) & vbTab & "σθ =
" & Round(StressY(Element), 5) & vbTab & "σz = " & Round(VerticalStress(Element), 5) &
vbTab & "τrz = " & Round(ShearStress(Element), 5) & vbTab & "kN/m²" & vbCrLf & vbCrLf
    End If
    ReportStreamToWrite.Write(OutputReportText)

Next

    ReportEnd = TimeString
    OutputReportText = vbCrLf & vbCrLf & " Analysis Duration = " &
AnalysisBegin & " to " & AnalysisEnd
    OutputReportText = OutputReportText & vbCrLf & " Reporting Duration= " &
ReportBegin & " to " & ReportEnd
    ReportStreamToWrite.Write(OutputReportText)

    ReportStreamToWrite.Close()
    Me.Cursor = Cursors.Default
    Me.Visible = False
    MsgBox("Report Generation Completed")
    Me.DialogResult = DialogResult.OK
End If
Catch ex As Exception
    MsgBox("Error! Report could not be generated.")
End Try

End Sub
End Class
```

### \*\*Codes Used for Managing the Cross-Section based Numeric Output

```
Imports System.Math

Public Class NumericOutputForm

    Private Sub NumericOutputComboBox_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
NumericOutputComboBox.SelectedIndexChanged
        Dim i As UInteger
        Dim NumericOutputTable As New DataTable
        Dim ScanRow As DataRow
        Dim DOF_index As UInteger = 1
        Dim ColumnStagger As Byte = 0
        Dim ColumnNumber As Integer = 2

    Try
```

```

        Select Case NumericOutputComboBox.SelectedIndex
        Case 0                                     'Vertical Deformation
Selectd
        For i = 0 To 2 * UBound(VerticalGridMatrix, 2) + 2 'preparing the
columns
            NumericOutputTable.Columns.Add()
        Next
        For i = 0 To 2 * UBound(HorizontalGridMatrix)      'preparing the
rows
            NumericOutputTable.Rows.Add()
        Next

        For Each ScanRow In NumericOutputTable.Rows
            ColumnNumber = ColumnNumber + ColumnStagger
            Do While ColumnNumber <= 2 * UBound(VerticalGridMatrix, 2) + 2
                ScanRow.Item(ColumnNumber) =
GlobalDisplacements(DOF_index)
                ColumnNumber += 2
                DOF_index += 2
            Loop
            ColumnNumber = 2
            ColumnStagger = 1 - ColumnStagger
        Next

        NumericOutputTable.Rows(0).Item(0) = HorizontalGridMatrix(0)
        Dim RowNumber As Integer = 1
        For i = 1 To UBound(HorizontalGridMatrix)
            NumericOutputTable.Rows(RowNumber).Item(0) = 0.5 *
(HorizontalGridMatrix(i - 1) + HorizontalGridMatrix(i))
            NumericOutputTable.Rows(RowNumber + 1).Item(0) =
HorizontalGridMatrix(i)
            RowNumber += 2
        Next

        ColumnNumber = 2
        NumericOutputTable.Columns(0).ColumnName = CStr(" ")
        If PlaneStrainAnalysis Then
            NumericOutputTable.Columns(1).ColumnName = CStr("x
coordinates")
        Else
            NumericOutputTable.Columns(1).ColumnName = CStr("r
coordinates")
        End If

        NumericOutputTable.Columns(2).ColumnName =
CStr(VerticalGridMatrix(0, 0))
        For i = 1 To UBound(VerticalGridMatrix, 2)
            NumericOutputTable.Columns(ColumnNumber + 1).ColumnName =
CStr(0.5 * (VerticalGridMatrix(0, i - 1) + VerticalGridMatrix(0, i)))
            NumericOutputTable.Columns(ColumnNumber + 2).ColumnName =
CStr(VerticalGridMatrix(0, i))
            ColumnNumber += 2
        Next

        Dim InsertRow As DataRow = NumericOutputTable.NewRow()
        InsertRow.Item(0) = "z coordinates"
        NumericOutputTable.Rows.InsertAt(InsertRow, 0)
        DataGridView1.DataSource = NumericOutputTable

        Case 1                                     'Horizontal
Deformation Selectd
        For i = 0 To 2 * UBound(VerticalGridMatrix, 2) + 2

```

```

        NumericOutputTable.Columns.Add()
    Next
    For i = 0 To 2 * UBound(HorizontalGridMatrix)
        NumericOutputTable.Rows.Add()
    Next

    For Each ScanRow In NumericOutputTable.Rows
        ColumnNumber = ColumnNumber + ColumnStagger
        Do While ColumnNumber <= 2 * UBound(VerticalGridMatrix, 2) + 2
            ScanRow.Item(ColumnNumber) = GlobalDisplacements(DOF_index
- 1)

            ColumnNumber += 2
            DOF_index += 2
        Loop
        ColumnNumber = 2
        ColumnStagger = 1 - ColumnStagger
    Next

    NumericOutputTable.Rows(0).Item(0) = HorizontalGridMatrix(0)
    Dim RowNumber As Integer = 1
    For i = 1 To UBound(HorizontalGridMatrix)
        NumericOutputTable.Rows(RowNumber).Item(0) = 0.5 *
(HorizontalGridMatrix(i - 1) + HorizontalGridMatrix(i))
        NumericOutputTable.Rows(RowNumber + 1).Item(0) =
HorizontalGridMatrix(i)
        RowNumber += 2
    Next

    'Labelling X coordinates on the Column headers
    ColumnNumber = 2
    NumericOutputTable.Columns(0).ColumnName = CStr(" ")
    If PlaneStrainAnalysis Then
        NumericOutputTable.Columns(1).ColumnName = CStr("x
coordiridates")
    Else
        NumericOutputTable.Columns(1).ColumnName = CStr("r
coordiridates")
    End If
    NumericOutputTable.Columns(2).ColumnName =
CStr(VerticalGridMatrix(0, 0))
    For i = 1 To UBound(VerticalGridMatrix, 2)
        NumericOutputTable.Columns(ColumnNumber + 1).ColumnName =
CStr(0.5 * (VerticalGridMatrix(0, i - 1) + VerticalGridMatrix(0, i)))
        NumericOutputTable.Columns(ColumnNumber + 2).ColumnName =
CStr(VerticalGridMatrix(0, i))
        ColumnNumber += 2
    Next

    'inserting the blank row below the column header
    Dim InsertRow As DataRow = NumericOutputTable.NewRow()
    InsertRow.Item(0) = "z coordiridates"
    NumericOutputTable.Rows.InsertAt(InsertRow, 0)
    DataGridView1.DataSource = NumericOutputTable

    Case 2
        DataGridView1.DataSource =
BuildStressStrainTables(HorizontalStress)
        'x Stress Selectd

    Case 3
        DataGridView1.DataSource = BuildStressStrainTables(StressY)
        'Y- Stress Selectd

```

```
Case 4 'Z Stress Selectd
    DataGridView1.DataSource = BuildStressStrainTables(VerticalStress)

Case 5 'Shear Stress Selectd
    DataGridView1.DataSource = BuildStressStrainTables(ShearStress)

Case 6 'x Strain Selectd
    DataGridView1.DataSource =
BuildStressStrainTables(HorizontalStrain)

Case 7 'Y Strain Selectd
    DataGridView1.DataSource = BuildStressStrainTables(StrainY)

Case 8 'Z Strain Selectd
    DataGridView1.DataSource = BuildStressStrainTables(VerticalStrain)

Case 9 'Shear Strain Selectd
    DataGridView1.DataSource = BuildStressStrainTables(ShearStrain)
End Select
DataGridView1.Columns(0).Frozen = True
DataGridView1.Columns(1).Frozen = True
DataGridView1.Rows(0).Frozen = True

Dim OutputColumn As DataGridViewColumn
For Each OutputColumn In DataGridView1.Columns
    'Me.DataGridView1.Columns(i).SortMode =
DataGridViewColumnSortMode.NotSortable
    OutputColumn.SortMode = DataGridViewColumnSortMode.NotSortable
Next
Catch ex As Exception

End Try

End Sub

Private Function BuildStressStrainTables(ByVal OutputParameter)
    Dim OutputTable As New DataTable
    Dim i As Integer
    Dim ScanRow As DataRow
    Dim ColumnSpacing As Byte
    Dim ColumnNumber As Integer = 2
    Dim ElementSeries1 As UInteger = 0
    Dim ElementSeries2 As UInteger = 1
    Dim ElementSeries3 As UInteger = 2
    Dim RowNumber As UInteger
    'Dim ElementIncrement As Integer = -2

    For i = 1 To 3 * UBound(VerticalGridMatrix, 2) + 2
        OutputTable.Columns.Add()
    Next

    For i = 1 To 3 * UBound(HorizontalGridMatrix)
        OutputTable.Rows.Add()
    Next

    RowNumber = 1
    For Each ScanRow In OutputTable.Rows
        'ScanRow.Item(0) = HorizontalGridMatrix(RowNumber - 1)

        If RowNumber Mod 3 = 1 Then
```

```

        ColumnNumber = 3
        Do While ColumnNumber <= 3 * UBound(VerticalGridMatrix, 2) - 1 + 2
            ScanRow.Item(ColumnNumber) = OutputParameter(ElementSeries1)
            ColumnNumber += 3
            ElementSeries1 += 4
        Loop

    ElseIf RowNumber Mod 3 = 2 Then
        ColumnNumber = 2
        ColumnSpacing = 2
        Do While ColumnNumber <= 3 * UBound(VerticalGridMatrix, 2) - 1 + 2
            ScanRow.Item(ColumnNumber) = OutputParameter(ElementSeries2)
            ColumnNumber += ColumnSpacing
            ColumnSpacing = 3 - ColumnSpacing
            ElementSeries2 += 2
            'ElementIncrement = 4 - ElementIncrement
        Loop

    ElseIf RowNumber Mod 3 = 0 Then
        ColumnNumber = 1 + 2
        Do While ColumnNumber <= 3 * UBound(VerticalGridMatrix, 2) - 1 + 2
            ScanRow.Item(ColumnNumber) = OutputParameter(ElementSeries3)
            ColumnNumber += 3
            ElementSeries3 += 4
        Loop

    End If
    RowNumber += 1
Next

RowNumber = 0
For i = 1 To UBound(HorizontalGridMatrix)
    OutputTable.Rows(RowNumber).Item(0) = Round(5 / 6 * HorizontalGridMatrix(i
- 1) + 1 / 6 * HorizontalGridMatrix(i), 3)
    OutputTable.Rows(RowNumber + 1).Item(0) = Round(0.5 *
(HorizontalGridMatrix(i - 1) + HorizontalGridMatrix(i)), 3)
    OutputTable.Rows(RowNumber + 2).Item(0) = Round(1 / 6 *
HorizontalGridMatrix(i - 1) + 5 / 6 * HorizontalGridMatrix(i), 3)
    RowNumber += 3
Next

ColumnNumber = 2
OutputTable.Columns(0).ColumnName = CStr(" ")
If PlaneStrainAnalysis Then
    OutputTable.Columns(1).ColumnName = CStr("x coordinates")
Else
    OutputTable.Columns(1).ColumnName = CStr("r coordinates")
End If

For i = 1 To UBound(VerticalGridMatrix, 2)
    OutputTable.Columns(ColumnNumber).ColumnName = CStr(" " & Round((5 / 6 *
VerticalGridMatrix(0, i - 1) + 1 / 6 * VerticalGridMatrix(0, i)), 3))
    OutputTable.Columns(ColumnNumber + 1).ColumnName = CStr(" " & Round(0.5 *
(VVerticalGridMatrix(0, i - 1) + VerticalGridMatrix(0, i)), 3))
    OutputTable.Columns(ColumnNumber + 2).ColumnName = CStr(" " & Round((1 / 6
* VerticalGridMatrix(0, i - 1) + 5 / 6 * VerticalGridMatrix(0, i)), 3))
    'OutputTable.Columns(ColumnNumber).ColumnName = 1 / 5 * ColumnNumber
    'OutputTable.Columns(ColumnNumber + 1).ColumnName = 1 / 5 * ColumnNumber +
1
    'OutputTable.Columns(ColumnNumber + 2).ColumnName = 1 / 5 * ColumnNumber +
2
    ColumnNumber += 3

```

```
Next

Dim InsertRow As DataRow = OutputTable.NewRow()
InsertRow.Item(0) = "z coordinates"
OutputTable.Rows.InsertAt(InsertRow, 0)
Return (OutputTable)
End Function

Private Sub NumericOutputForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    Try
        If PlaneStrainAnalysis Then
            NumericOutputComboBox.Items(2) = "x stress"
            NumericOutputComboBox.Items(3) = "y stress"
            NumericOutputComboBox.Items(5) = "shear stress (xz)"
            NumericOutputComboBox.Items(6) = "x strain"
            NumericOutputComboBox.Items(7) = "y strain"
            NumericOutputComboBox.Items(9) = "shear strain (xz)"
        Else
            NumericOutputComboBox.Items(2) = "r stress"
            NumericOutputComboBox.Items(3) = "θ stress"
            NumericOutputComboBox.Items(5) = "shear stress (rz)"
            NumericOutputComboBox.Items(6) = "r strain"
            NumericOutputComboBox.Items(7) = "θ strain"
            NumericOutputComboBox.Items(9) = "shear strain (rz)"
        End If
        NumericOutputComboBox.SelectedIndex = 0
    Catch ex As Exception

    End Try

End Sub
End Class
```