To the Graduate Council:

I am submitting herewith a thesis written by Pasquale Rinaldi entitled "A Web-Based Interactive Finite Element Test Bed and Tutorial." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science.

Dr. Christopher D. Pionke, Major Professor

We have read this thesis and
recommend its acceptance:

Dr. N. Allen Yu

Dr. Rapinder S. Sawhney

Acceptance for the Council:

Vice Provost and Dean of Graduate Studies

A Web-Based Interactive Finite Element Test Bed and Tutorial

A Thesis

Presented for the

Master's of Science Degree

The University of Tennessee, Knoxville

Pasquale John Rinaldi, Jr.

December 2002

Abstract

This thesis outlines the development of a web-based, interactive tutorial and computational test bed for students in their first course on finite elements. The program provides analytical and numerical results using the finite element method on various engineering problems in solid mechanics and the fluid-thermal sciences.

Before developing the program, it was necessary to research current software packages already available and the different teaching methods used. The research showed that there was a need for a program like the one developed in this thesis and the best software to calculate analytical solutions with would be *Maple*. The best format to develop the program is through the web using HTML, JavaScript, and PhP. This format allows the most users access to the program.

The program's website is frame based which allows easy access to any area of the site. The **Home** section provides information on the scope of the Thesis. The **Program Information** section provides information on the website's features. The **Tutorial Examples** section provides in-depth solutions to example problems. The **Interactive TestBed** section is where the user can develop, solve, and compare various problems using the finite element method. The **TestBed Help** section provides an interactive help guide to using the **TestBed** section.

At the time of publication, the website is located at `http://www.engr.utk.edu/~pionke/webifett/`. This site provides information regarding introductory finite elements in an analytical and numerical way. It is a resource which provides a good reference for any course discussing introductory finite elements. This site helps fill in the gaps that may have been overlooked during coursework. It can also reinforce concepts that were covered. The interactive aspect of this website provides the user with the ability to quickly solve different problems with little difficulty.

Contents

List of Figures

Chapter 1

Introduction

The purpose of this thesis is to design a web-based, interactive tutorial and computational test bed for students in their first course on finite elements (FE). The program will provide both analytical and numerical results using the finite element method on various engineering problems in solid mechanics and the fluid-thermal sciences. This program bridges the gap between numerical and theoretical finite element courses by presenting not only numerical solutions to one and two dimensional finite element problems but also presents closed form analytical solutions. At the time of publication, the website is located at `http://www.engr.utk.edu/~pionke/webifett/`.

The analytical solutions provide insight into how the element size, type of elements, material, and geometric properties affect the problem solution. The numerical solutions provide comparisons to the analytical solutions to demonstrate how effective the finite element method is at predicting approximate solutions. This effectiveness is then applied to problems where no exact solution is available. Before getting into the specifics of this program, it is important to give some background into current FE software, teaching techniques, and why this program is useful.

1.1 Background Research

Before developing the tutorial, it was necessary to research software packages that are already available, different teaching methods available and the way material is presented to students. Also it was important to research software to use to develop the tutorial, how to provide the program to the students, and finally convenience. As for convenience, if there is software that can do something needed for the tutorial, then there is no need to redevelop it. With these ideas in mind, researching current software packages began.

1.1.1 Current Software Packages

After beginning the search, it became apparent that there are hundreds of software packages available. They vary from extremely specific to very general in the types of FE problems they solve. The first thing was to try to find software that would solve finite element problems analytically, since this is what a major part of the tutorial is focused on. After extensive research of software packages, it was found that most of the available software programs do not do analytical solutions to FE problems.

The reason for most programs not performing symbolic solutions to FE problems is simple. If it is possible solve an FE problem analytically, then this FE problem has a closed form solution. When a problem has a closed form solution, then the finite element method is not needed to approximate the solution. Most software packages are written to solve problems with real world applications and these problems usually do not contain a closed form solution. With this fact, some examples of software packages are described below.

One of the first programs researched was *Structural Mechanics*[8] which was developed by the makers of *Mathematica*. This program solves varying structural mechanics problems analytically, which is part of what the tutorial will do, but it does not include fluid-thermal problems. This program has a small learning curve with its well designed graphical user interface (GUI) and good graphical output of results. The program only

allows the user to select from a handful of problems though and does not allow the student to modify the selected governing differential equation. Also, this program is not web based and would therefore not be freely avaiable. As mentioned earlier, due its limited types of problems, *Structural Mechanics's* would not be a good program to use when developing the tutorial.

Another program that is of interest is *ANSYS* [2]. This is a "typical" FE program and most others such as Cosmos [3], Abaqus [1], etc are similar in design and use. This program does not solve FE problems analytically. It does solve varying FE problems with very good graphical results. The version of the program that was researched is available on Unix environments, and has a very harsh learning curve. If the user has very little knowledge of the FE method, then using this program will be very difficult. Therefore, since *ANSYS* does not provide analytical solution methods and has a very difficult learning curve, it is not sufficient to develop the tutorial. While *ANSYS* has its disadvantages for this tutorial, it is a very good FE program that can solve many advanced structural problems.

The program entitled *PDELab* [10] enables a user to solve a PDE system numerically across the Internet. *PDELab* uses a Unix server running an X-client to enable the user to access the GUI interface to enter their problem and develop a solution. It uses a pre- and post-processor in which to display the graphics back to the user for their desired input and solution outputs. This program was well developed but requires a moderate learning curve and does not provide analytical solutions. Also, there are a limited number of X-clients on their servers, so at times; the program is unavailable to the user.

Matlab [7] is another mainstream program that provides both analytical and symbolic solutions to finite element problems. The symbolic library and analytical capabilities in *Matlab* implements the analytical engine from *Maple* [6], which is the biggest drawback to *Matlab*. Why use *Matlab*, which gets some of its capabilities from *Maple* when *Maple* is available for use. Another drawback is that *Matlab* does not provide good support for exporting its equations to other formats. Other than these drawbacks, *Matlab* has a

very robust and well developed programming environment in which to develop the FE tutorial program. With this review of some of the many different software packages on the Internet, the next step was to research different methods used to teach finite elements to students with computer interaction.

1.1.2 Current Tutorial Methods on the Web

The first place that was researched for current teaching methods was the finite element course provided at the University of Tennessee, Engineering Science 551w, taught by Professor Baker [4]. This is a course that the author has taken first hand. This course is typical to other courses taught at other universities. The course material deals with varying engineering disciplines from structural to fluid-thermal fields. Along with solving problems by hand from the book, there are also a number of lab assignments that are used through Dr. Baker's own finite element problem-solving environment (fempse) developed in *Matlab*. The labs dealt with modifying templates written in *Matlab*. To modify these templates require some basic knowledge of *Matlab*, so there exists a moderate learning curve for students who have no previous experience with *Matlab*.

The process developed by Professor Baker involves learning through numerical solutions, the FE process. The method involves solving three classes of problems, verification, benchmark, and validation problems. Verification problems deal with comparing the results of the FE experiment with the analytical solution to a simplified problem. Benchmark problems deal with comparison of the FE results compared to the results obtained from another known and accepted method or code. The last class of problems was validation problems and these deal with comparison of the results to experimental data previously obtained. This process allows definitive information to support the accuracy of the FE method chosen so it may be applied to any problem chosen and provide accurate results that can be trusted.

Using *Matlab* templates enables the student to learn about the FE process and gain some insight into the complexities with which to write such dynamic code. As with all

source code that comes with many programs, it is very difficult to modify someone else's work, no matter how well documented, and this can cause some frustration when trying to solve the different labs. It is also easy to get lost wondering what all the components do and mean when working through the labs. Another problem is that all the solutions are purely numerical, and though some analytical solutions are derived by hand, this process is very limited and time consuming. Although a huge amount of information is learned in this class, providing more insight into the analytical results through the FE tutorial will help the student.

Other courses are purely theoretical, have excellent introductory textbooks [5] and require all the course work done by hand. This a useful teaching aid, but can only be performed on certain sets of problems. This limitation does not allow students to see how the FE method can approximate other problems without solutions and see the true capabilities that using the FE method can provide. Some courses provide software packages such as *ANSYS* and others like it, but programs such as these only provide numerical solutions to problems along with other limitations already addressed.

There are other courses still that present the FE method through the use of pure programming languages such as `FORTRAN` and `C++`. These programs have huge learning curves and require the student to fully understand how to write, edit, and compile programs to apply the FE method to various engineering problems. Courses such as this allow the student to easily get lost in the programming and fail to learn any of the FE process. Finally, the last type of classes that teach the FE method are ones that teach the analytical and theoretical methods to students. After the students have a good grasp on these methods, they are introduced to commercial packages such as *ANSYS* that solve, numerically, the FE problem. The students get an overview on when to use different packages and an introduction in how to use them. These courses present the foundation of the FE method to the students with hand solved theoretical analyses and then provides real world applications through the use of various commercial codes available.

With all of these courses analyzed, there is a need for the FE tutorial program that is being developed to help bridge the gap between numerical and analytical courses with little to no learning curve.

1.2 Format of the Tutorial

There are many formats available today to provide software to the public. These include platform-dependent, platform-independent, server-side web based, client-side web based, etc. To provide the FE tutorial program to the most people, on a large variety of computer formats, with little or no learning curve involved, the choice was the World Wide Web, or the Internet. Providing the bulk of the GUI as a web-based interface will require use of HyperText Markup Language (HTML), HyperText PreProcessor (PHP), and Javascript.

Programming the GUI in these formats allow any user on any computer using most of the available web browsers access to the program. The tutorial will allow the user to enter all the information on their browser, send the information to the server where all the computational work is done through a mathematical software package and then the results are sent back to the user on their browser. This format will allow the program to reach the most amount of people with the least amount of user-end computer requirements. A good Internet connection with a recent Internet browser should be the bulk of the requirements needed to run the FE program. The next step was to determine the mathematical package that would solve the FE problems given.

1.3 Software to Perform the FE Modeling

After the extensive research mentioned above, the program of choice to develop the finite element tutorial was *Maple*. *Maple* provides analytical and numerical solution capabilities. It provides a detailed programming environment in which to design the FE tutorial program. *Maple* provides various formats with which to export solutions

and provides various formats with which to represent inputs into the program. The next important aspect was to easily import information from the Internet and export that information back to the Internet. *Maple* is able to do this easily by providing direct commands to translate information to and from MathML, the math typesetting language for HTML. The ability to easily transfer information from the Internet to *Maple* and back again was a huge factor in choosing *Maple*.

1.4 Program Objectives

In developing the tutorial, it is intended to provide the user with:

- Ability to select a one or two dimensional engineering problem class and then develop the solution step by step.
- Ability to choose between the type and size of the mesh that the user wants to use, where applicable.
- Ability to choose how the user wants to solve the given problem such as through the use of full or under integration.
- Ability to calculate answers analytically and numerically through the use of normalized graphs and charts.
- In-depth solutions and explanations to these problems on how the finite element method works and how the user's choice, such as integration type, can effect the solution.
- Extensive pre- and post-processing graphics to visualize the problem, the choice of mesh, and convergence results .
- An extensive set of already developed solutions for some standard problems, allowing the user to choose the problem he wants solved.

- An extensive help file detailing the workings of the program, what different terms mean, and one tutorial leading the student through the interactive testbed section.
- Details demonstrating the finer points of the finite element method such as convergence studies and error checks.

The first item in the objectives is one of the most important parts of the program. For the user to be able to select from a variety of $1D$ and $2D$ problem classes, which will be developed step by step. This process allows the user the greatest amount of flexibility while minimizing any errors the student may encounter. Along with selecting their problem with ease, the GUI provides a set of fields to inform the student of their specific engineering problem selected. The fields include the partial differential equation, boundary and initial conditions, type and number of elements, etc. By providing these and other problem fields, maximum readabiltity and ease of use throughout the Tutorial is achieved.

The user will be able to obtain the solution to a specific problem both analytically and numerically through the use of normalized graphs, while being able to experiment with the different solution methods. The ability to experiment with a problem is one of the major features lacking from the majority of FE codes. By simply changing a few menu choices, the user can get a completely different result without having to code these changes himself. This ability to experiment with the problem will allow the user to see the differences in the results based off of choosing different FE methods and compare these results with the analytical solution obtaining a true understanding of the intricacies involved in the FE solution method.

The ability to choose between the type and number of elements, where applicable, that the user wants to use is also essential to the learning process. By being able to choose different types and numbers of elements the user is allowed to experiment with the problem and learn how the FE process works through mesh refinement or some other process. For instance, for the 1D Beam Problem Class the user only has one choice of element type, while for the 2D Plane-Stress Problem Class, the user can also choose

between different element types and integration method.

Being able to choose different ways to solve the problem also allows the user to experiment with different solution methods in the FE process. Whether they may want to perform, for example under or full integration, the user is able to learn how these and other finite element techniques can affect solution convergence. By allowing the user to calculate the solution analytically, they can use the analytical results to provide direct comparisons on how the solution depends on element size and element geometry while using the numerical results to view graphics representing these same comparisons.

Providing the user with in-depth solutions and explanations of certain processes and terminology along the way will help to shed light on some finite element nuances that the user may not have fully grasped yet. The goal is to lead the user through each step of the solution process with full information. By providing in-depth solutions explaining output from the problem entered, the user can learn what they just told the program to do, along with ways that changing the problem might affect the solution.

By providing an array of graphics for both pre- and post-processing, the testbed will help the user to visualize the problem at hand, along with the results. For any problem selected, the user will able to see pictures of the problem and the style of mesh chosen for the problem. With the results obtained, the user will able to visualize how the problem's solution has changed due to the governing differential equation and convergence results for various different FE methods chosen. Along with the analytical and numerical results, the graphics will help in visually explaining the given problem.

With all programs, providing full examples with solutions is important to get a feel for how the program works, and some concrete examples to look at when trouble arises. The solutions will be interactive, i.e. the user can change a parameter of the solution and recompute the answer, or read an example in the help file. The help file will include an extensive glossary, a tutorial of a worked testbed example, and a detailed topical explanation of the FE method on selected problems. Help files are important in that they allow the user to get answers to questions about the program, and about any

specifics about what the program is doing.

Chapter 2

User Interface Development

2.1 Layout and Design

Before starting any program, a layout is needed. After deciding to use the world wide web (www) for this tutorial, the layout of this program was designed. The interface was developed with the user in mind. This meant that the layout of the website had to be uniform, straightforward and simple to follow. The first way to achieve this was to start out with a header, shown in Figure 2.1. The header is located inside a frame and consists of a set of links that appear above every page so that the user can navigate his way around the entire website easily.

Along with the navigational header for the entire website, there is a navigation bar, shown in Figure 2.2 on some of the longer webpages embedded in that page. This webpage navigation bar allows the user to go back and forth within the single webpage making it easier to access the information that would otherwise require an excessive amount of scrolling up or down.

When the user is inside the page, there are links, shown in Figure 2.3 strategically placed throughout the webpage that allow the user to click the link and be transported to the top of the webpage immediately, giving the user access to navigational bars along the header. From here, it was necessary to develop the easiest way to receive input from the user.

Figure 2.1: Navigational Header Throughout WebSite

Figure 2.2: Navigational Header For a Single WebPage

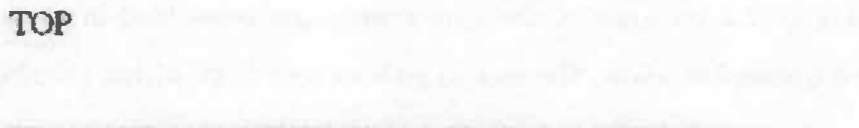

Figure 2.3: Links Taking User back to the Top of a Long Page

2.2 Format to Receive Input

Deciding on the format to receive input from the user was a daunting task. It was necessary to make sure that the website provided both text and images to best convey the subject matter. To better define structure throughout, tables were used to maintain a logical structure and order. To best receive data from the user, a restrictive format was used. The restrictive format consisted of breaking up the problem into many parts and then stepping the user through each part of the problem piece by piece. To further enhance the ease of use, each time the user advances one step, the results from the previous selections are displayed to remind the user of the problem that they are developing. The format to receive input is shown in Figure 2.4 below.

The choices are presented to the user using HTML forms and JavaScript. This allows for the fastest page loading time along with the simplest and easiest interface for the user to follow. All of the selections that the user has the ability to choose from have been predefined. Originally, the user was going to be allowed to dynamically enter a governing differential equation (GDE) into an equation editor and then have *Maple* dynamically calculate the FE solution. However, allowing the user to enter any equation they want may lead to many problems with format, usefulness, reasonableness, and feasability. Numerous errors could occur in this appoach and it is highly probable the problem they

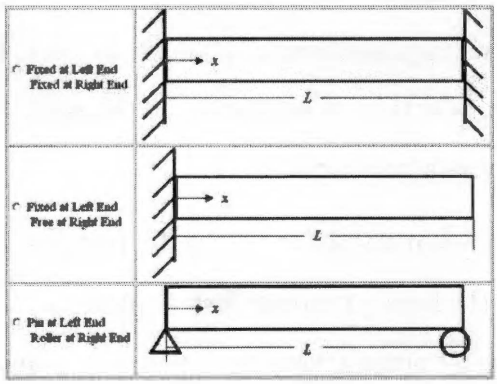

Figure 2.4: Example of User Input

entered does not have a solution at all.

To handle this problem, the restrictive format was chosen. This allows for useful and educationally beneficial problems to be interactively developed, solved, and compared while making sure the user doesn't get frustrated trying to come up with a useful problem. From here, how to compute the data was the next format that was tackled.

2.3 Format to Compute Data

The format to compute the data that was given to the user is achieved by using *Maple*. *Maple* is able to analytically compute solutions to the exact and finite element problem formulations. By using the selections that the user enters, the program goes to a set page where the already computed information is displayed on that page. The reason that all the calculations are performed beforehand and just loaded based off of the user's selections is a matter of the current state of technology.

Currently, *Maple* and servers cannot handle the required computation time required to run these simulations. For example, a 16 element one dimensional Euler-Bernoulli beam with a constant load and constant geometric properties takes approximately 45 minutes to run. A server with one user on the website will have trouble handling this load, not to mention if a whole classroom of students is using the website. To cut down server requirements, a set selection of problems for the user to choose from along with the precomputed results was a necessity.

While at first this may seem not as useful, in the end it still accomplishes the stated goals. For example the 1D Beam problem has a choice of 64 different finite element problems to choose from by simply varying the load, boundary conditions, and the number of elements. This is more than sufficient to demonstrate how the FE solutions accuracy vary with changes in geometric, material, and element properties.

2.4 Format to Display Results

The best way to display the results of the interactive selections and computations with *Maple* was through the use of pictures. By providing the equations and beam description as pictures, such as in Figure 2.5, allows anybody with an internet connection and a recent web browser access to use this site. Currently, there is no set standard for providing equations on the internet. While MathML is getting better, there are just too many requirements that must be followed which would restrict the minimum system requirements and rule out a great portion of students who may not be experts with computers.

With that aside, using text, equations and pictures through each step of the selection process shows the user what types of features for their problem they have selected without having to remember or go back and forth between the webpages. Providing the selections for the user to choose from with descriptions and pictures helps the user understand the choices they have and eliminate most, if not all, confusion or mistakes that may arise.

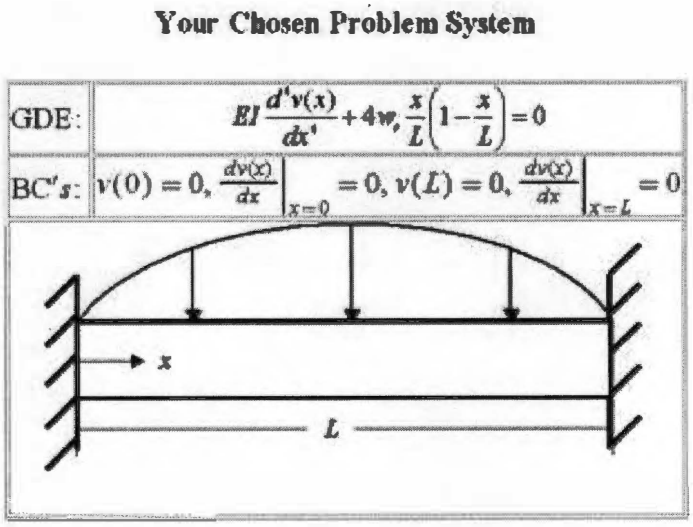

Figure 2.5: Example of Displayed Selected Information

Chapter 3

Program Overview

To better understand the workings of the program, it is necessary to first solve the problem as one would do on paper. This solution will be done for the analytical and finite element solution methods. From this point, the same solutions will be demonstrated in the program from the tutorial page and the interactive testbed page. Providing the problem in these three formats will allow for a better understanding of the usefulness of the program to the user and also allow for comparisons between these formats. The information provided in this chapter directly correlates to material found on the tutorial section of this website.

3.1 Problem Statement

The problem that is going to be solved in this chapter is the one dimensional Euler-Bernoulli Beam Problem. The beam has a constant load, w_o, length, L, moment of inertia, I, and modulus of elasticity, E. The beam is also cantilevered with the left end fixed and the right end free, see Figure 3.1. The Governing Differential Equation (GDE) is represented by

$$EI\frac{d^4v(x)}{dx^4} + w_o = 0 \tag{3.1}$$

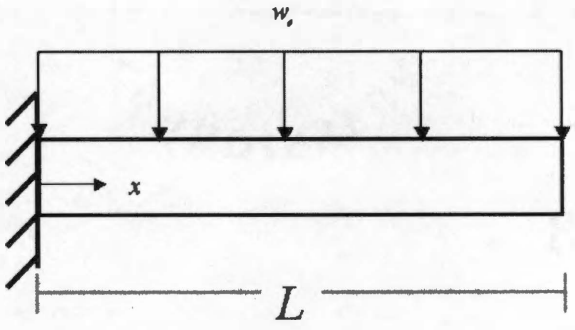

Figure 3.1: One Dimensional Cantilevered Euler-Bernoulli Beam

where $v(x)$ is the transverse displacement of the beam. The boundary conditions for the left end of the beam are given by

$$v(0) = 0, \phi(x) = \frac{dv(x)}{dx}\Big|_{x=0} = 0 \tag{3.2}$$

where ϕ is the rotation and the boundary conditions for the right end of the beam are given by

$$M(x) = EI\frac{d^2v(x)}{dx^2}\Big|_{x=L} = 0, V(x) = EI\frac{d^3v(x)}{dx^3}\Big|_{x=L} = 0 \tag{3.3}$$

where $M(x)$ is the moment and $V(x)$ is the shear.

From this information alone, is now possible to solve this problem either analytically or with the finite element method. The first method tackled is the Analytical Solution.

3.2 Analytical Solution

Starting with the GDE, one can integrate this equation four times.

$$EI\frac{d^3v(x)}{dx^3} = EI\int \frac{d^4v(x)}{dx^4}dx = -\int w_o dx = -w_o x + C_1 \tag{3.4}$$

$$EI\frac{d^2v(x)}{dx^2} = EI\int \frac{d^3v(x)}{dx^3}dx = -\frac{w_o}{2}x^2 + C_1 x + C_2 \tag{3.5}$$

$$EI\frac{dv(x)}{dx} = EI\int \frac{d^2v(x)}{dx^2}dx = -\frac{w_o}{6}x^3 + \frac{C_1}{2}x^2 + C_2 x + C_3 \tag{3.6}$$

$$EIv(x) = EI\int \frac{dv(x)}{dx}dx = -\frac{w_o}{24}x^4 + \frac{C_1}{6}x^3 + \frac{C_2}{2}x^2 + C_3x + C_4 \tag{3.7}$$

Now that we have a function for $v(x)$, we can apply the four boundary conditions to solve for the unknown constants of integration.

$$v(0) = 0 \Rightarrow C_4 = 0 \tag{3.8}$$

$$\phi(0) = \frac{dv(x)}{dx}\Big|_{x=0} = 0 \Rightarrow C_3 = 0 \tag{3.9}$$

$$V(L) = \frac{d^3v(x)}{dx^3}\Big|_{x=L} \Rightarrow C_1 = w_oL \tag{3.10}$$

$$M(L) = \frac{d^2v(x)}{dx^2}\Big|_{x=L} \rightarrow C_2 = -w_oL^2 + \frac{w_oL^2}{2} \Rightarrow C_2 = -\frac{w_o}{2} \tag{3.11}$$

Plugging these values back into $v(x)$ yields

$$v(x) = \frac{1}{EI}\left(-\frac{w_o}{24}x^4 + \frac{w_oL}{6}x^3 - \frac{w_oL^2}{4}x^2\right) \tag{3.12}$$

From this equation one can now find the rotation, moment, and shear as follows

$$\phi(x) = \frac{1}{EI}\left(-\frac{w_o}{6}x^3 + \frac{w_oL}{2}x^2 - \frac{w_oL^2}{2}x\right) \tag{3.13}$$

$$M(x) = -\frac{w_o}{2}x^2 + w_oLx - \frac{w_oL^2}{2} \tag{3.14}$$

$$V(x) = -w_ox + w_oL \tag{3.15}$$

Plotting these functions, Figures 3.2 through 3.5, yields a good visualization to the respective functions. The functions are normalized by dividing through by any constants to develop a uniformity throughout the results.

3.3 Finite Element Solution

3.3.1 Element Formulation

The same example used in Section 3.2 will now be analysed using the Finite Element method. As stated before, the beam has a length L, material properties, E and I, and a

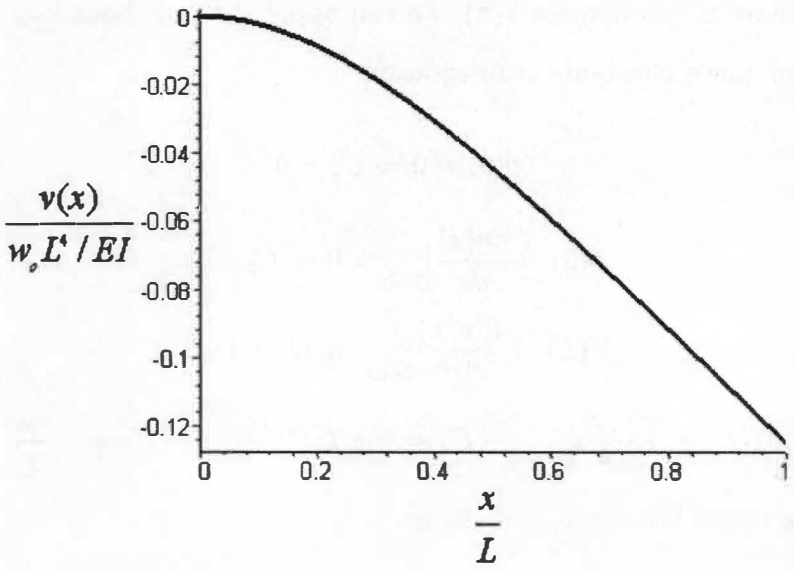

Figure 3.2: Normalized Displacement

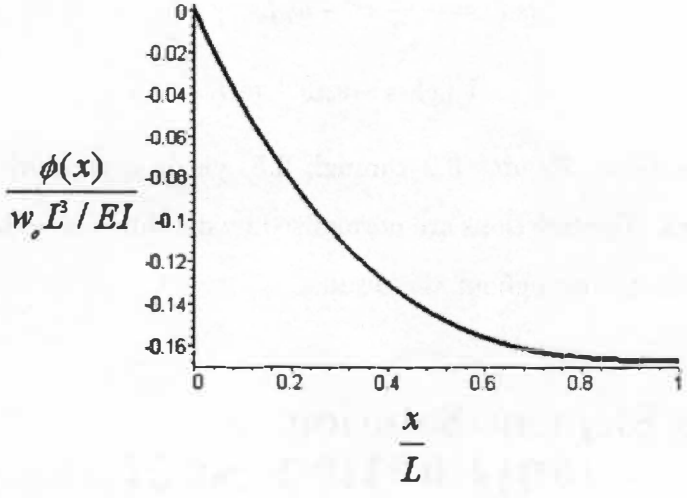

Figure 3.3: Normalized Rotation

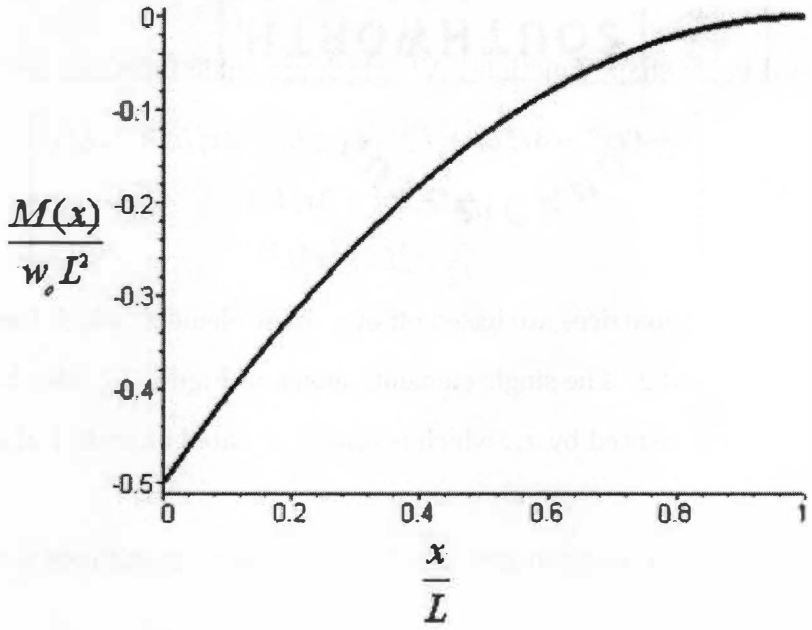

Figure 3.4: Moment

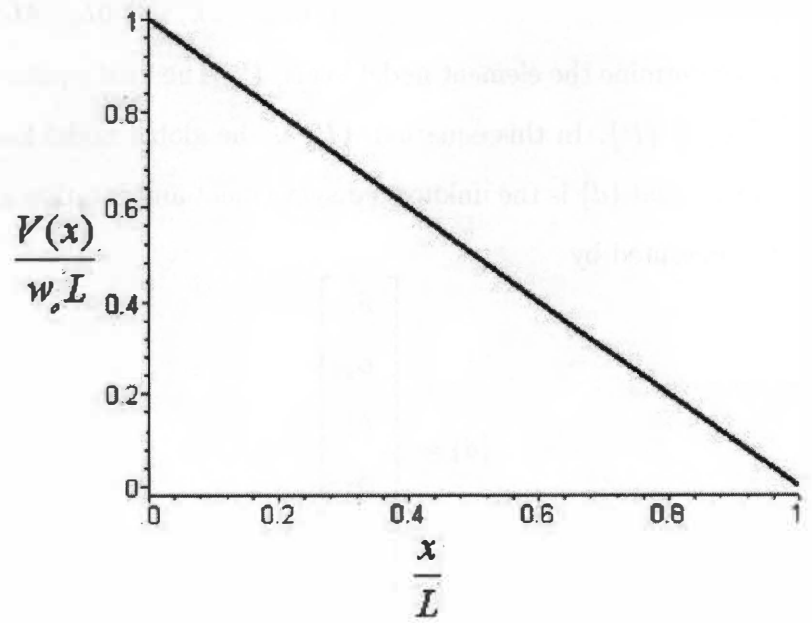

Figure 3.5: Shear

constant distributed load, w_o. To start, the problem will be discretized into two elements as shown in Figure 3.6.

The standard beam shape functions, N^T which are cubic functions are chosen.

$$N^T = \begin{bmatrix} \frac{1}{L_e^3}(2x_e^3 - 3x_e^2L_e + L_e^3) & \frac{1}{L_e^3}(x_e^3L_e - 2x_e^2L_e^2 + x_eL_e^3) \\ \frac{1}{L_e^3}(-2x_e^3 + 3x_e^2L_e) & \\ \frac{1}{L_e^3}(x_e^3L_e - x_e^2L_e^2) & \end{bmatrix} \tag{3.16}$$

This and other element matrices are based off of a single element, which has a length L_e, and has two nodes 1 and 2. The single element, shown in Figure 3.7, also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element.

With this information, one can now determine the element stiffness matrix,k_e given by the equation below.

$$k_e = EI\int_0^{L_e} \frac{d^2\{N\}^T}{dx^2}\frac{d^2\{N\}}{dx^2}dx_e \Rightarrow k_e = \frac{EI}{L_e^3}\begin{bmatrix} 12 & 6L_e & -12 & 6L_e \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 \\ -12 & -6L_e & 12 & 6L_e \\ 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix} \tag{3.17}$$

The next step is to determine the element nodal loads, P_e. The final equation that needs to be solved is $[k]\{d\} = \{P\}$. In this equation, $\{P\}$ is the global nodal loads, $[k]$ is the global stiffness matrix, and $\{d\}$ is the unknown displacement and rotation global matrix. This equation is represented by

$$\{d\} = \begin{bmatrix} \delta_1 \\ \phi_1 \\ \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} \tag{3.18}$$

To determine the element nodal loads the equation

$$P_e = \int_0^{L_e} \{N\}^T w(x)dx_e = \int_0^{L_e} \{N\}^T w_o dx_e \tag{3.19}$$

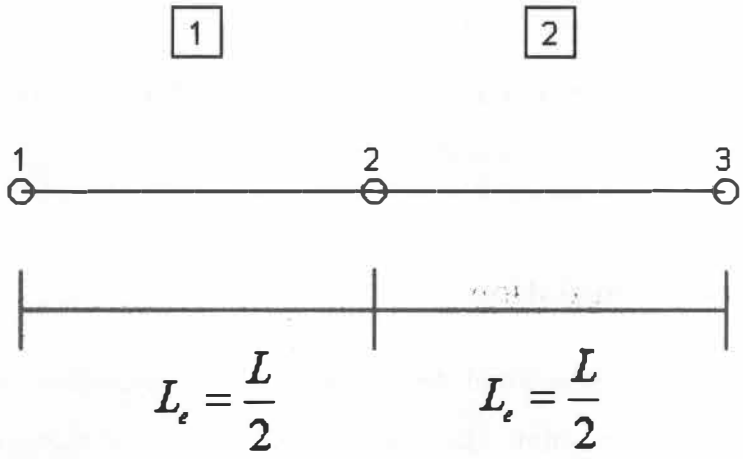

Figure 3.6: 1D Beam Discretization

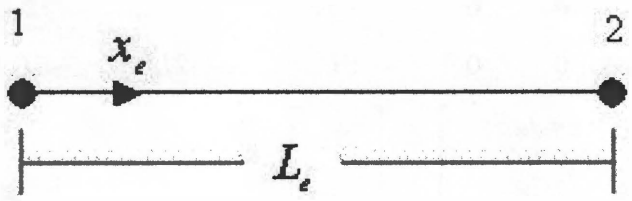

Figure 3.7: Uniform Element

is used, which upon integrating yields the element nodal load matrix

$$P_e = \begin{bmatrix} f_e^1 \\ m_e^1 \\ f_e^2 \\ m_e^2 \end{bmatrix} = \begin{bmatrix} \frac{w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \\ \frac{w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \end{bmatrix} \tag{3.20}$$

In (3.20), the terms f_e^1 and m_e^1 represent the nodal force and moment for a single element at node 1, hence the superscript 1 and subscript e [5].

3.3.2 Global Formulation

Now that the element stiffness and element nodal load matrices have been determined, they can be assembled to obtain the global stiffness and load matrices. The global stiffness matrix, k, is found using superposition, $k = k_e^1 + k_e^2$. The global load matrix, P, is found in a similar manner using superposition but also includes the reactions R at the nodes which have an external support.

$$k = \frac{EI}{L_e^3} \begin{bmatrix} 12 & 6L_e & -12 & 6L_e & 0 & 0 \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 & 0 & 0 \\ -12 & -6L_e & 12+12 & 6L_e-6L_e & -12 & 6L_e \\ 6L_e & 2L_e^2 & -6L_e+6L_e & 4L_e^2+4L_e^2 & -6L_e & 2L_e^2 \\ 0 & 0 & -12 & -6L_e & 12 & 6L_e \\ 0 & 0 & 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix},$$

$$P = \begin{bmatrix} \frac{w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \\ -\frac{w_o L_e}{2} - \frac{w_o L_e}{2} \\ \frac{w_o L_e^2}{12} - \frac{w_o L_e^2}{12} \\ \frac{w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \end{bmatrix} + \begin{bmatrix} R_L \\ M_L \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.21}$$

where R_L and M_L are the reactions at the left or fixed end of the beam. The final global equation $[k]\{d\} = \{P\}$ becomes

$$\frac{EI}{L_e^3}\begin{bmatrix} 12 & 6L_e & -12 & 6L_e & 0 & 0 \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 & 0 & 0 \\ -12 & -6L_e & 24 & 0 & -12 & 6L_e \\ 6L_e & 2L_e^2 & 0 & 8L_e^2 & -6L_e & 2L_e^2 \\ 0 & 0 & -12 & -6L_e & 12 & 6L_e \\ 0 & 0 & 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix}\begin{bmatrix} \delta_1 \\ \phi_1 \\ \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} -\frac{w_o L_e}{2} + R_L \\ \frac{-w_o L_e^2}{12} + M_L \\ -w_o L_e \\ 0 \\ \frac{-w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \end{bmatrix} \tag{3.22}$$

3.3.3 Boundary Conditions

Now that equation $[k]\{d\} = \{P\}$ has been formed, it is necessary to apply the displacement and rotation boundary conditions only. The moment and shear boundary conditions are not applied to this problem because the finite element method calculates these values. The approximate values for shear and moment are compared with the known boundary conditions to assess solution accuracy. Remembering that the two boundary conditions used are $v(0) = 0$ and $\phi(0) = 0$, they can be applied to the stiffness and load matrices. Since $x = 0$ corresponds to node 1, in the displacement matrix, d, δ_1 and ϕ_1 are now known and become $\delta_1 = 0$ and $\phi_1 = 0$. Upon applying these conditions, the matrix equation can be reduced to two different equation systems. The first being

$$\frac{EI}{L_e^3}\begin{bmatrix} 24 & 0 & -12 & 6L_e \\ 0 & 8L_e^2 & -6L_e & 2L_e^2 \\ -12 & -6L_e & 12 & -6L_e \\ 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix}\begin{bmatrix} \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} -w_o L_e \\ 0 \\ \frac{-w_o L_e}{2} \\ \frac{w_o L_e^2}{12} \end{bmatrix} \tag{3.23}$$

and the second represented as

$$\frac{EI}{L_e^3}\begin{bmatrix} -12 & 6L_e & 0 & 0 \\ -6L_e & 2L_e^2 & 0 & 0 \end{bmatrix}\begin{bmatrix} \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} -\frac{w_o L_e}{2} + R_L \\ \frac{w_o L_e^2}{12} + M_L \\ -w_o L_e \\ 0 \end{bmatrix} \tag{3.24}$$

In (3.23) and (3.24), the displacement matrix, d, has values of δ_2 and ϕ_2 which represent the nodal displacement and rotation at node 2 respectively. The same idea applies for δ_3 and ϕ_3.

3.3.4 Solution Process

Equation (3.23) is used to solve for the nodal displacements and rotations. Once these values are known, equation (3.24) is used to solve for the unknown reactions. Upon solving equation (3.23), and substituting the length of the element in terms of the beam length, $L_e = \frac{L}{2}$, the nodal displacements and rotations become

$$\begin{bmatrix} \delta_1 \\ \phi_1 \\ \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{17}{384}\frac{w_o L^4}{EI} \\ -\frac{7}{48}\frac{w_o L^3}{EI} \\ -\frac{1}{8}\frac{w_o L^4}{EI} \\ -\frac{1}{6}\frac{w_o L^3}{EI} \end{bmatrix} \tag{3.25}$$

Taking the values of the displacements found in (3.25) and substituting them back in (3.24) and solving yields the values of the reactions to be

$$\begin{aligned} R_L &= w_o L \\ M_L &= \tfrac{w_o L^2}{2} \end{aligned} \tag{3.26}$$

Remembering statics, the reactions at the fixed support are determined to be

$$\begin{aligned} R_L &= w_o L \\ M_L &= \tfrac{w_o L^2}{2} \end{aligned} \tag{3.27}$$

The reactions found analytically match exactly to the finite element method reactions.

3.4 Comparison with Exact Solution

Having solved for the nodal displacements and rotations, it is now possible to calculate the approximate displacement, rotation, moment, and shear to obtain a better

feeling of how our FE approximation compares to the exact solution. The approximate displacement function, v_{fe} is found on an element-by-element basis. So for the displacement, there are two functions used to approximate the exact displacement found above. The first approximate function, v_{fe}^1, is valid over the domain of element 1 and the second approximate function, v_{fe}^2 is valid over the domain of element 2. The approximate functions use the shape functions again.

$$v_{fe}^1(x_e) = N^T \begin{bmatrix} \delta_1 \\ \phi_1 \\ \delta_2 \\ \phi_2 \end{bmatrix}, \quad v_{fe}^2(x_e) = N^T \begin{bmatrix} \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} \tag{3.28}$$

After solving for (3.28) and substituting $L_e = \frac{L}{2}$, the approximate displacement function, $v_{fe}(x)$ becomes

$$v_{fe}(x) = \begin{cases} \frac{1}{EI}\left(\frac{w_o L}{8}x^3 - \frac{23 w_o L^2}{96}x^2\right) & \text{for} \quad \frac{x}{L} < \frac{1}{2} \\ \frac{1}{EI}\left(a_1 x^3 - a_2 x^2 + a_3 x - a_4\right) & \text{for} \quad \frac{1}{2} \leq \frac{x}{L} \leq 1 \end{cases} \tag{3.29}$$

where the constants in (3.29) are defined as

$$\begin{aligned} a_1 &= \frac{w_o L}{24} \\ a_2 &= \frac{5 w_o L^2}{96} + \frac{w_o L}{16} \\ a_3 &= \frac{w_o L}{32} + \frac{5 w_o L^2}{96} - \frac{7 w_o L^3}{48} \\ a_4 &= \frac{17 w_o L^4}{384} - \frac{7 w_o L^3}{96} + \frac{5 w_o L^2}{384} + \frac{w_o L}{192} \end{aligned} \tag{3.30}$$

Comparing the approximate solution to the exact solution found in (3.12) shows that the order of the FE solution is one less than the order of the exact solution. Upon graphing both functions, Figure 3.8, a good idea of how close the approximate function is to the exact function can be determined. Also note that the FE solution at the nodes is exactly equal to the theoretical solution for the displacement.

Now that the approximate displacement function is found, the approximate rotation, moment, and shear can be calculated by taking successive derivatives of the displacement function, remembering that the rotation is the derivative of the displacement, the

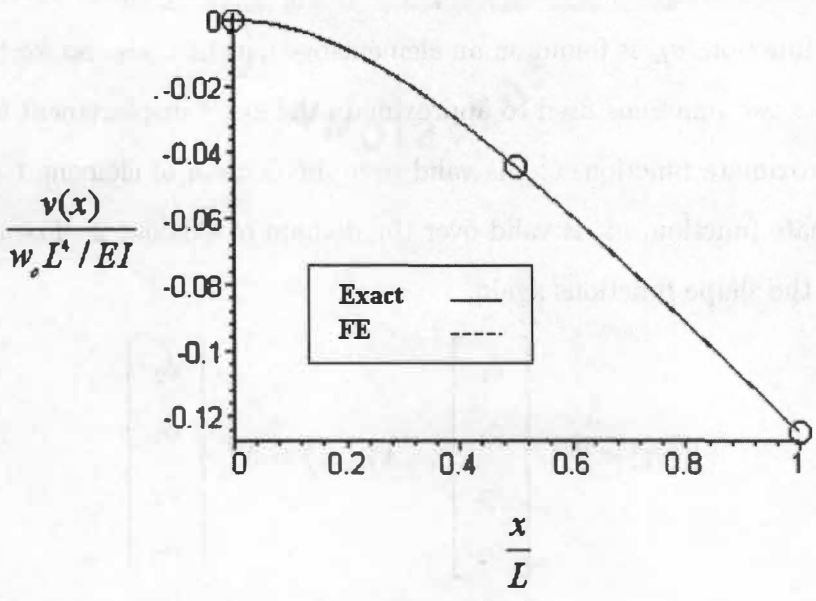

Figure 3.8: Comparison of Exact vs Approximate Displacement

moment is the derivative of the rotation times EI, and the shear is the derivative of the moment. Therefore, the rotation approximate function, $\phi_{fe}(x)$, is

$$\phi_{fe}(x) = \begin{cases} \frac{1}{EI}\left(\frac{3w_oL}{8}x^2 - \frac{23w_oL^2}{48}x\right) & \text{for} \quad \frac{x}{L} < \frac{1}{2} \\ \frac{1}{EI}\left(b_1x^2 - b_2x + b_3\right) & \text{for} \quad \frac{1}{2} \leq \frac{x}{L} \leq 1 \end{cases} \tag{3.31}$$

where

$$\begin{aligned} b_1 &= \frac{w_oL}{8} \\ b_2 &= \frac{5w_oL^2}{48} + \frac{w_oL}{8} \\ b_3 &= \frac{w_oL}{32} + \frac{5w_oL^2}{96} - \frac{7w_oL^3}{48} \end{aligned} \tag{3.32}$$

The approximate moment, $M_{fe}(x)$, becomes

$$M_{fe}(x) = \begin{cases} \frac{3w_oL}{4}x - \frac{23w_oL^2}{48} & \text{for} \quad \frac{x}{L} < \frac{1}{2} \\ \frac{w_oL}{4}x - \frac{5w_oL^2}{96} & \text{for} \quad \frac{1}{2} \leq \frac{x}{L} \leq 1 \end{cases} \tag{3.33}$$

Finally, the approximate shear, $V_{fe}(x)$, becomes

$$M_{fe}(x) = \begin{cases} \frac{3w_oL}{4} & \text{for} \quad \frac{x}{L} < \frac{1}{2} \\ \frac{w_oL}{4} & \text{for} \quad \frac{1}{2} \leq \frac{x}{L} \leq 1 \end{cases} \tag{3.34}$$

Now, when comparing these approximate functions to their exact counterparts in equations (3.12)-(3.15), it can be seen that they are also lower by one polynomial order. This difference stems from the order of the GDE and the order of the shape functions chosen. Upon plotting the rotation, moment, and shear, a better idea of accuracy can be observed in spite of this difference in polynomial order. These plots are represented in Figures 3.9-3.11.

As can be seen, the rotation, moment, and shear get increasingly worse when compared to the exact solution. This of course stems from taking derivatives of an approximate function which results in an increase in error. By selecting a higher discretization, i.e. a larger number of elements, the approximate solution will come closer to the exact solution along the beam for the displacement, rotation, moment, and shear.

For further evaluation, Figures 3.12-3.15 represented below show the results of the FE displacement, rotation, moment, and shear for 2,4,8, and 16 elements evaluated at $x = L$. For our normalized system, this means, that it is evaluated at $\frac{x}{L} = 1$.

As can be seen, the 2 element FE displacement and rotation match the exact solution exactly at the nodes including the end of the beam. So a 2 or 4 element mesh would be enough to generate good results. But when observing the moment and shear comparisons, it takes upwards of 16 elements and higher to get good results.

Therefore, it is important to know what results are needed before the FE process is started. If only the displacement or rotation is needed, 2 elements would suffice and time would not be spent on a finer mesh. If looking at the moment or shear though, 2 elements wouldn't suffice and a higher order mesh would be needed.

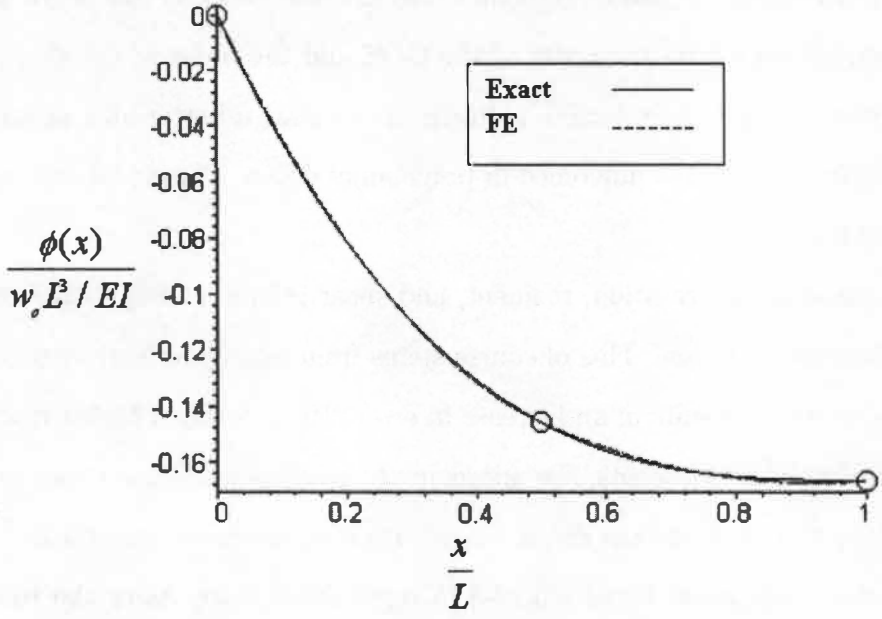

Figure 3.9: Comparison of Exact vs Approximate Rotation

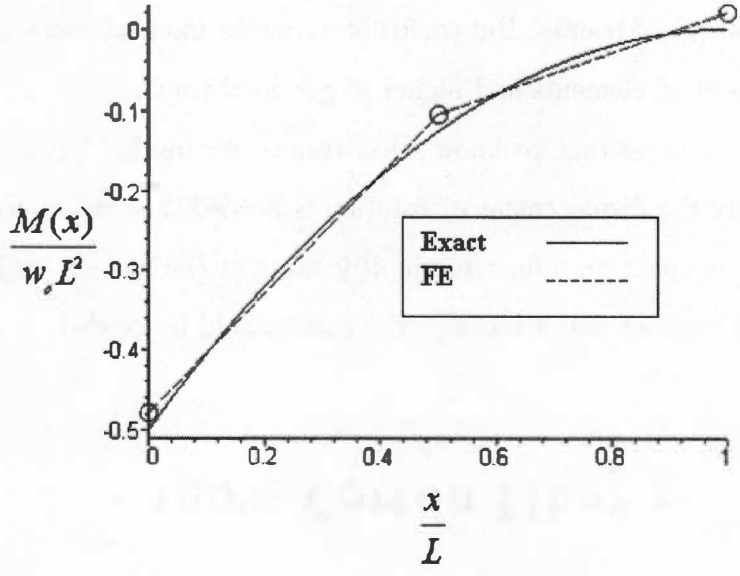

Figure 3.10: Comparison of Exact vs Approximate Moment

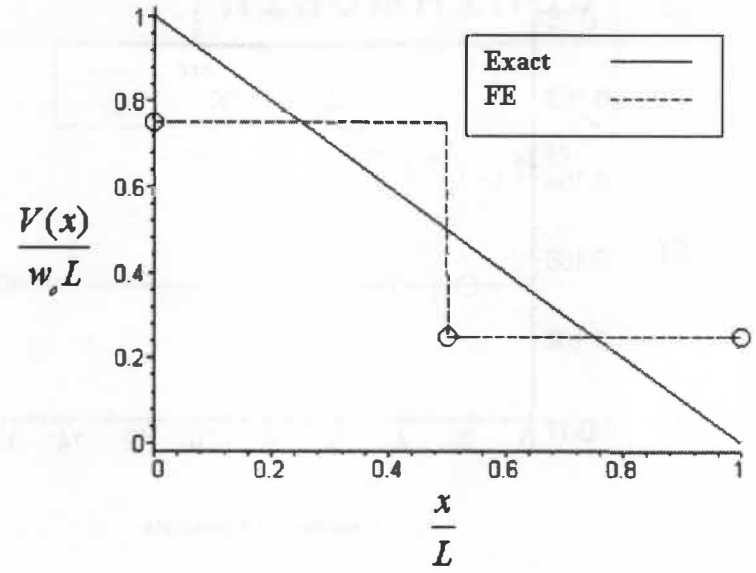

Figure 3.11: Comparison of Exact vs Approximate Shear

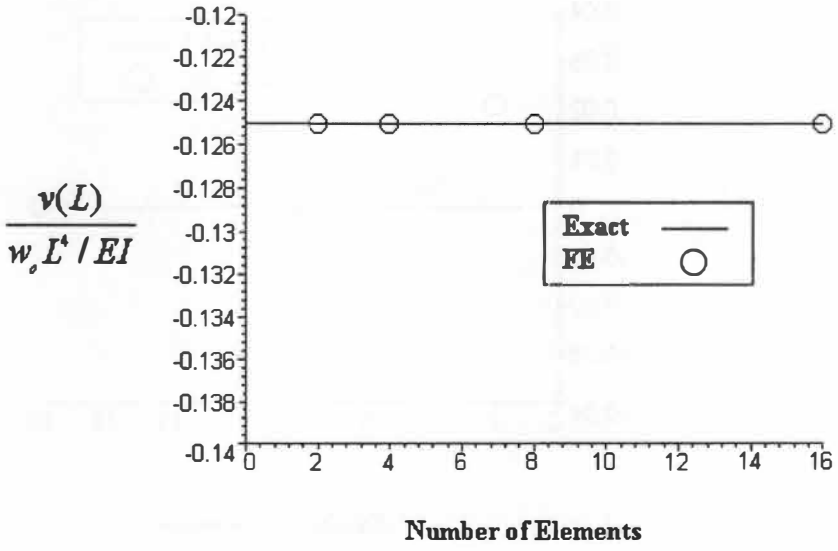

Figure 3.12: Normalized Displacement at the End of the Beam

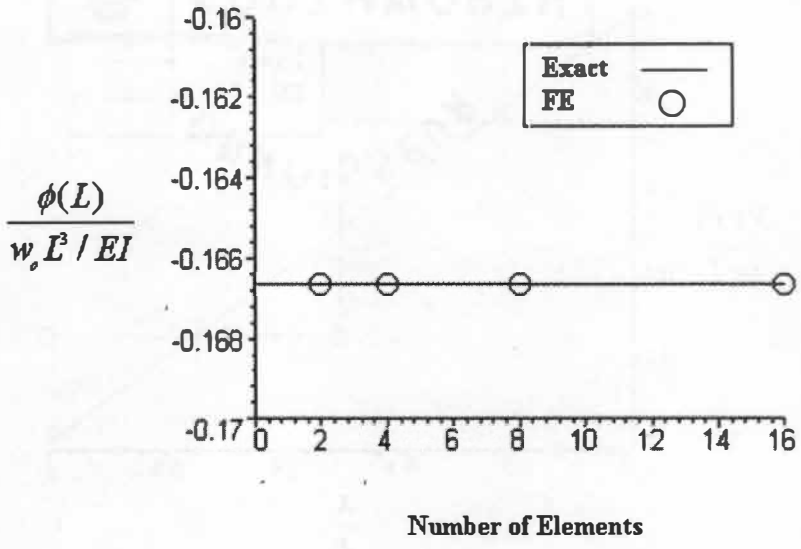

Figure 3.13: Normalized Rotation at the End of the Beam

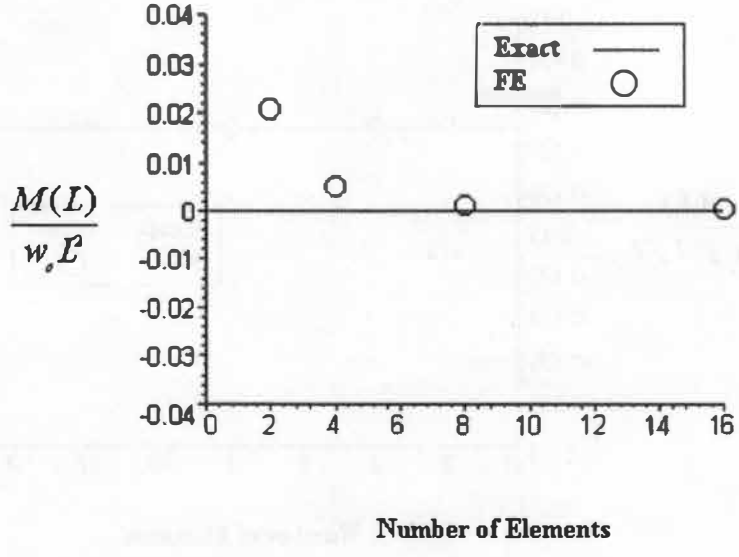

Figure 3.14: Normalized Moment at the End of the Beam

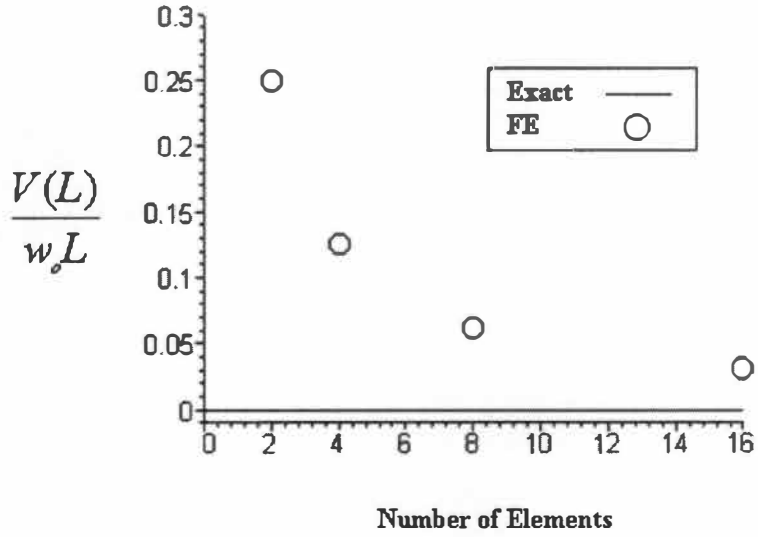

Figure 3.15: Normalized Shear at the End of the Beam

Chapter 4

TestBed Tutorial

The TestBed Tutorial demonstrates how to use the Interactive TestBed. The help page is designed exactly like the Interactive section to walk the user through the process of developing a problem, viewing its solution, and comparing its results.

The help page is designed so that only one selection will work on each page, forcing the user to follow the help file and design one specific problem, while making sure that the user can experiment with other choices and not get lost amid the myriad of problems to design. Once on the page for the interactive testbed help page, the student is presented with the following selections as shown in Figure 4.1.

The Testbed Tutorial will instruct the user to select the **One Dimensional Euler Bernoulli Beam**. Once this has been selected, the user is transported to the next step, the Load selection page. The first thing the user sees is the GDE for their selected problem, and a figure showing the current coordinate system and any necessary dimensions, Figure 4.2.

Below their current problem, the allowable loading cases are listed with an accompanying picture and equation as shown in Figure 4.3. For this example the user is instructed to select the **constant load** case.

Upon selecting the proper load, the user is transported to the next step, the Boundary Condition selection page. Again, the user is first presented with a recap of the problem

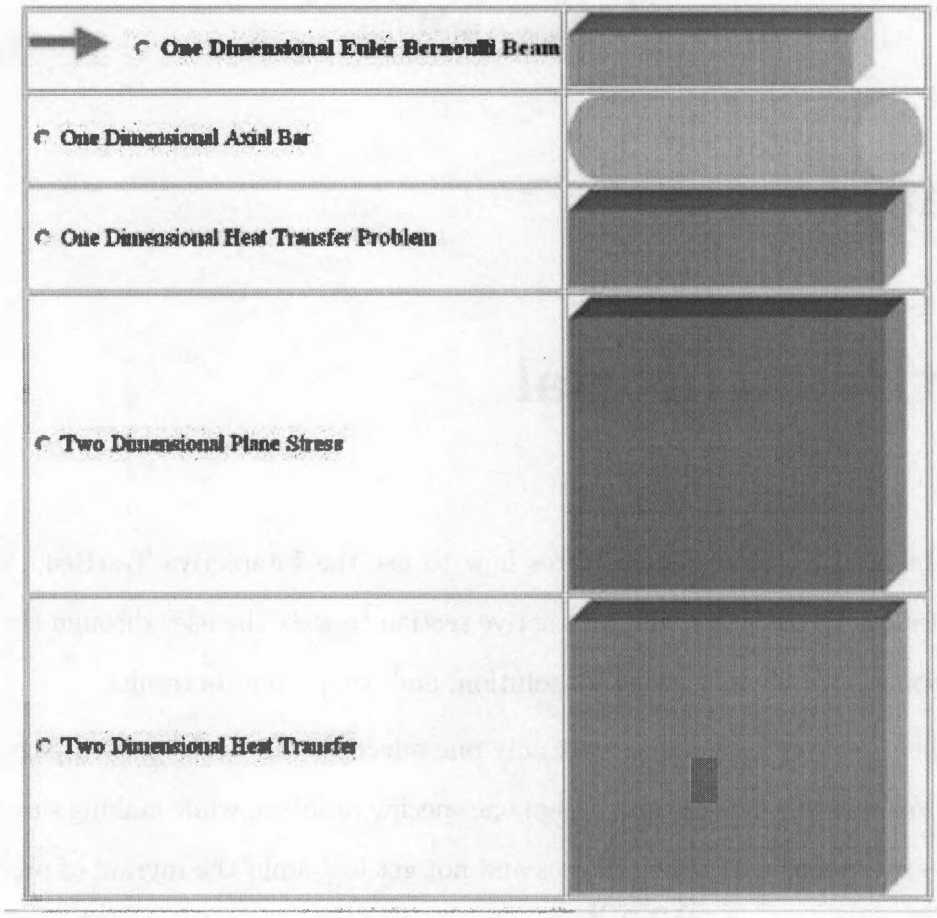

Figure 4.1: Problem Class Selection Page - Select 1D Beam

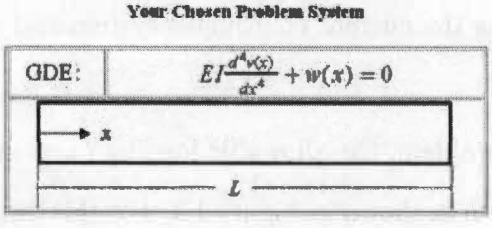

Figure 4.2: Current Selected Problem - GDE

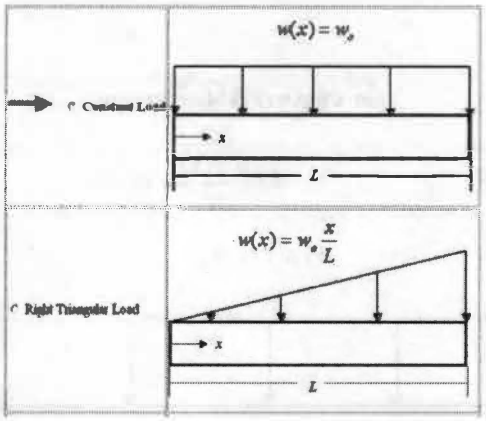

Figure 4.3: Load Selection Page - Select Constant Load

they are developing, Figure 4.4. Here the user is presented with the GDE updated to include the selected load case, $w(x) = w_o$ and the picture has also been updated.

Below the recap, the user now has a choice of boundary conditions. The boundary conditions are shown with an accompanying picture, Figure 4.5, and for this example the user is instructed to select **Fixed at Left End, Free at Right End**.

Upon selecting the proper boundary conditions, the user is then taken to the Element information page. On this page, as expected, the user is given a recap of what he has developed so far. This includes the GDE, Loading, and BC information including relative equations and pictures, Figure 4.6. Below the information regarding the problem being developed, the user is presented with his choice of element (if applicable) and the number of elements to use. For this example, the user does not have a choice of element, only the number of elements to use, Figure 4.7. Also provided is a popup window with information from the tutorial describing the shape functions and element stiffness matrix development.

The user is instructed to select the **2 Element** case. This will transport the user to the solution to their particularly developed problem. Here the user is presented with a recap for their entire problem including the GDE, Loading condition, boundary conditions, and element selection with accompanying equations and pictures as shown

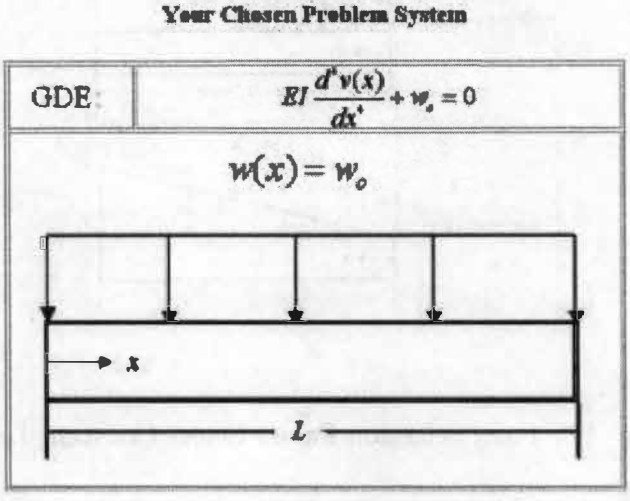

Figure 4.4: Current Selected Problem - GDE + Loading

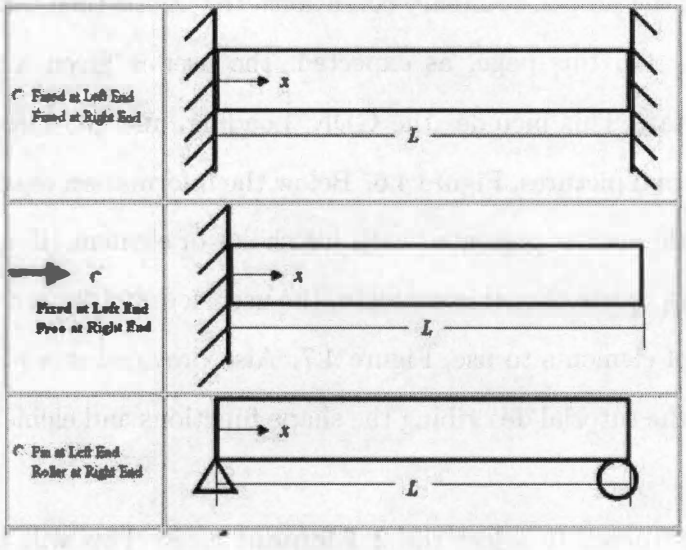

Figure 4.5: BC Selection Page - Select Fixed Left, Free Right

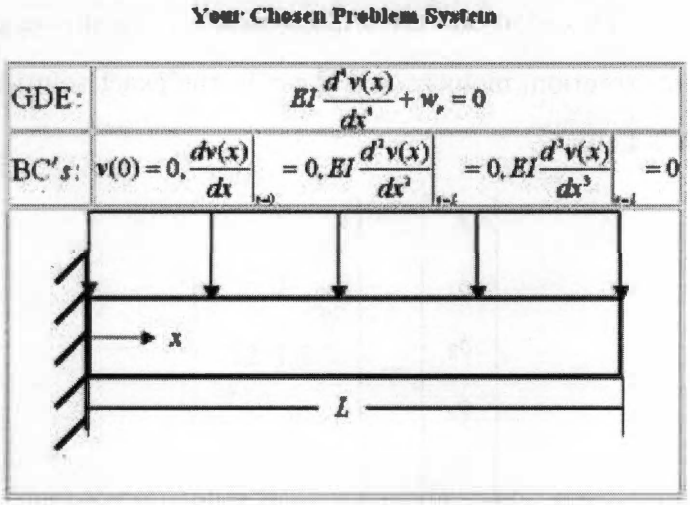

Figure 4.6: Current Selected Problem - GDE + Loading + BC's

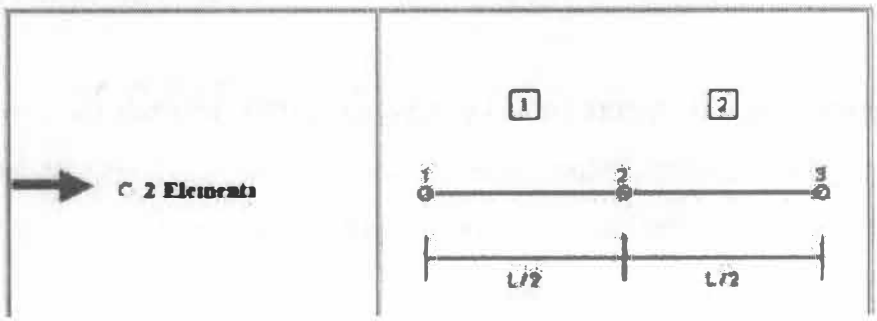

Figure 4.7: Element Selection Page - Select 2 Elements

in Figure 4.8.

Below the problem statement is the solution to their problem. For this example the nodal displacements and rotations are given along with accompanying graphs comparing the FE displacement, rotation, moment and shear to the exact solution, represented in (4.1) and Figures 4.9-4.12.

$$\begin{bmatrix} \delta_1 \\ \phi_1 \\ \delta_2 \\ \phi_2 \\ \delta_3 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{17}{384}\frac{w_o L^4}{EI} \\ -\frac{7}{48}\frac{w_o L^3}{EI} \\ -\frac{1}{8}\frac{w_o L^4}{EI} \\ -\frac{1}{6}\frac{w_o L^3}{EI} \end{bmatrix} \tag{4.1}$$

Following these graphs is a brief discussion on the accuracy of the solution for this example. Again, it can be seen that the rotation, moment, and shear get increasingly worse when compared to the exact solution. Again, this of course stems from taking derivatives of an approximate function which results in an increase in error. By selecting a higher discretization, i.e. a larger number of elements, the approximate solution will come closer to the exact solution along the beam for the displacement, rotation, moment, and shear.

After the solution is presented, the user has the option to choose whether or not to view the 4, 8, 16, or compare all choices for the same problem. This option is shown in Figure 4.13.

The addition of this feature provides easy navigation between the same problem allowing the user to quickly compare different discretization solutions without having to go back and forth using the back button and regular menu selections.

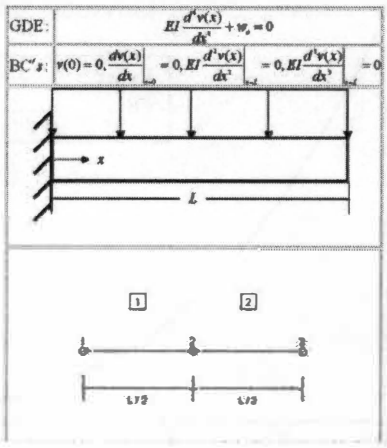

Figure 4.8: Current Selected Problem - GDE + Loading + BC's + Elements

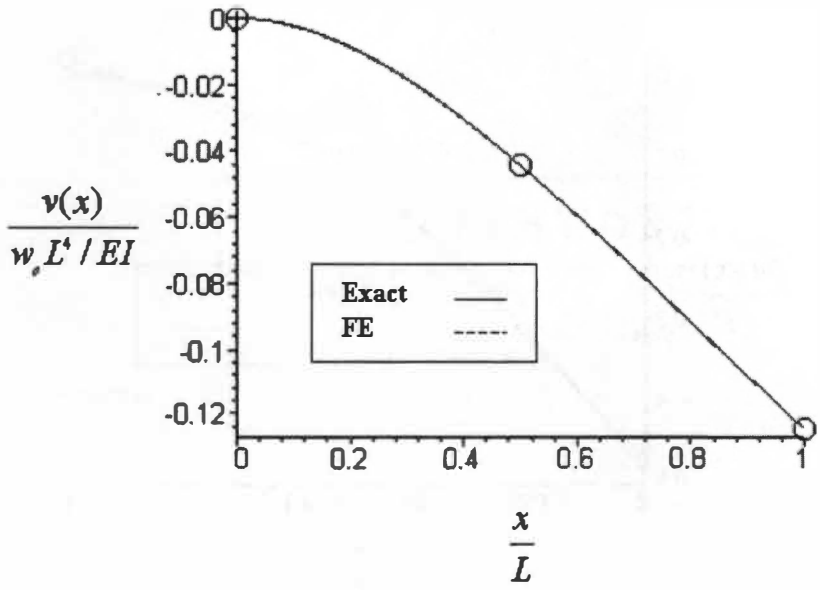

Figure 4.9: Analytical vs FE Normalized Displacement

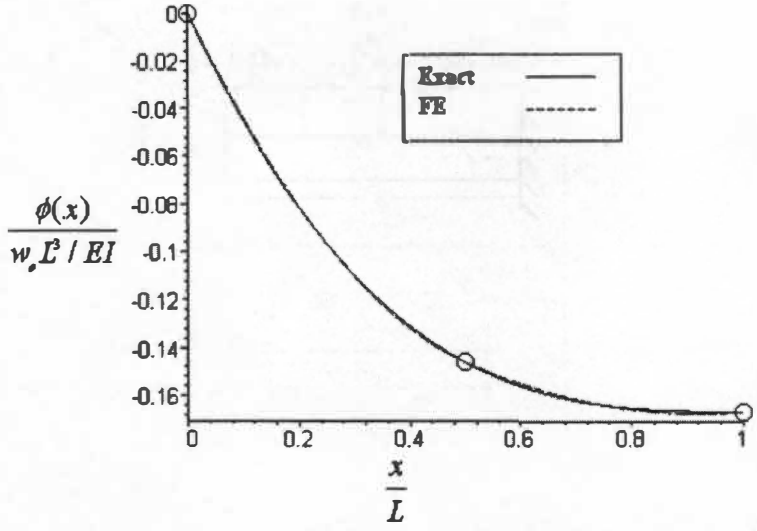

Figure 4.10: Analytical vs FE Normalized Rotation

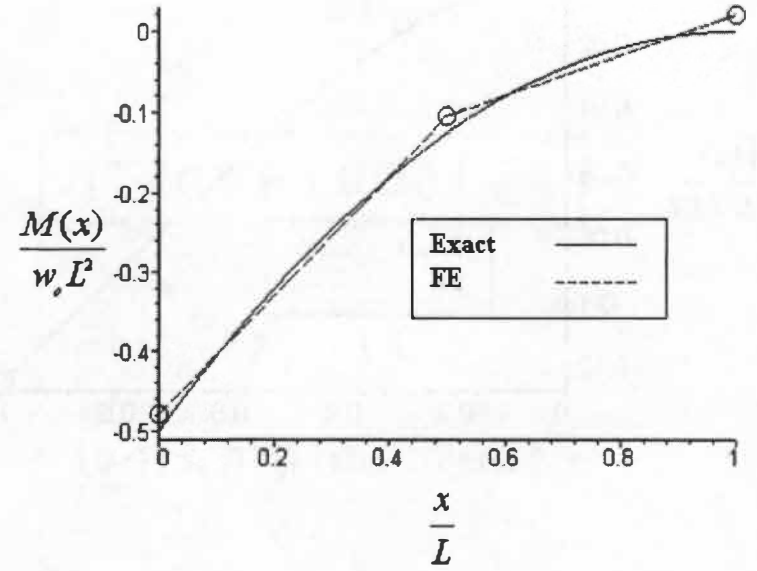

Figure 4.11: Analytical vs FE Normalized Moment

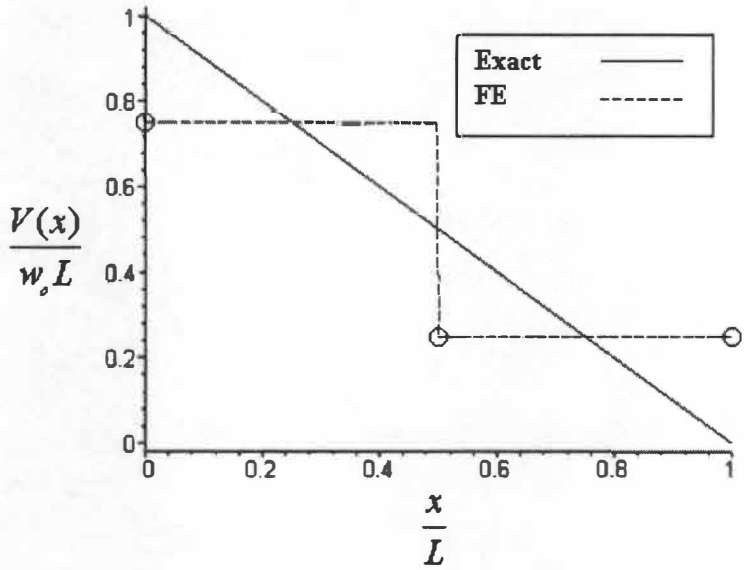

Figure 4.12: Analytical vs FE Normalized Shear

Figure 4.13: Choice to View Other Element Discretizations for Same Problem

Chapter 5

Summary of Program

The two main sections of the program are the tutorial section and the Interactive TestBed section. The tutorial section walks the user through a full finite element problem. It starts with the problem description and follows with an analytical solution. The finite element solution is found next and the final section compares the two solutions for accuracy. The testbed section works through the many different finite element problems that the user can develop and then displays the solution to the specific problem developed. This chapter is devoted to describing all the types of problems available in the website.

5.1 Tutorial Section

The following classes of problems are available in the tutorial section of the website:

- One Dimensional Axial Bar
  - Two Linear Elements
  - One Quadratic Element
- One Dimensional Beam
  - Two Standard Euler Bernoulli Elements

- One Dimensional Heat Conduction
  - Two Linear Elements
  - One Quadratic Element
- Two Dimensional Plane Stress
  - Two Linear Elements
- Two Dimensional Heat Conduction
  - Two Linear Elements
  - One Quadratic Element

5.1.1 One Dimensional Axial Bar

The one dimensional bar tutorial section takes the user through the example of a 1D bar with a constant load P_o applied along the sheath. The bar has a length L, modulus of elasticity E, and area A. The bar is fixed at the left end and free at the right end. The problem is solved using two linear elements and one quadratic element.

The GDE for the tutorial is

$$EA\frac{du(x)}{dx} + P_o = 0, \tag{5.1}$$

where $u(x)$ is displacement along the x-direction. The boundary conditions for this problem are

$$u(0) = 0, \quad \sigma(0) = E\frac{du(x)}{dx}\Big|_{x=L} = 0 \tag{5.2}$$

The problem is shown in Figure 5.1.

From here the analytical solution is solved and the results are displayed. Then the finite element method is applied using the two different types of elements. The importance of choosing a shape function and the results obtained using different types of elements is described.

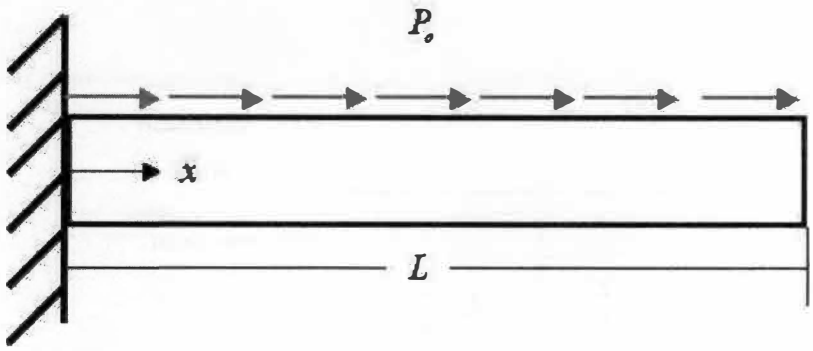

Figure 5.1: 1D Bar - Problem Description

Two Linear Elements

The bar is discretized into two linear elements of uniform length shown in figure 5.2.

The linear shape function is chosen and has the form

$$N = \left[\begin{array}{cc} 1 - \frac{x_e}{L_e} & \frac{x_e}{L_e} \end{array} \right] \tag{5.3}$$

The linear element has a length L_e, and has two nodes, 1 and 2. The single element also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element as shown in Figure 5.3.

One Quadratic Element

The bar is discretized into one quadratic element shown in Figure 5.4.

The quadratic shape function is chosen and has the form

$$N = \left[\begin{array}{ccc} 1 - 3\frac{x_e}{L_e} + 2\frac{x_e{}^2}{L_e} & 4\frac{x_e}{L_e} - 4\frac{x_e{}^2}{L_e} & 2\frac{x_e{}^2}{L_e} - \frac{x_e}{L_e} \end{array} \right] \tag{5.4}$$

The quadratic element has a length L_e, and has three nodes, 1,2, and 3. The single element also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element. This element is shown in Figure 5.5.

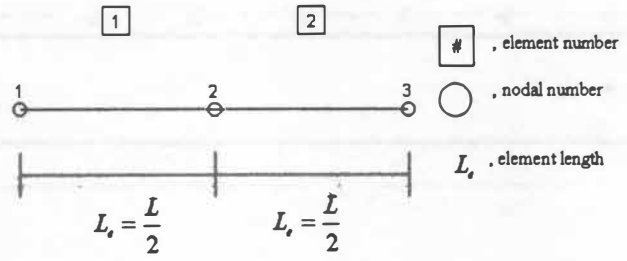

Figure 5.2: 1D Bar Discretization - Two Linear Elements

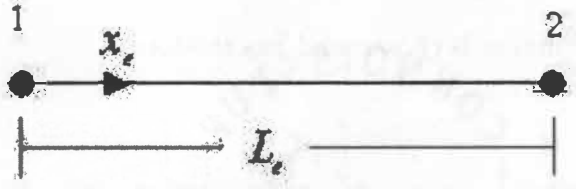

Figure 5.3: Uniform Linear Element

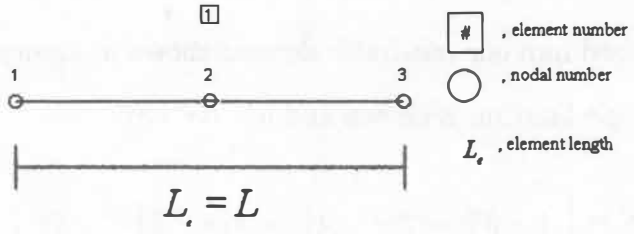

Figure 5.4: 1D Bar Discretization - One Quadratic Element

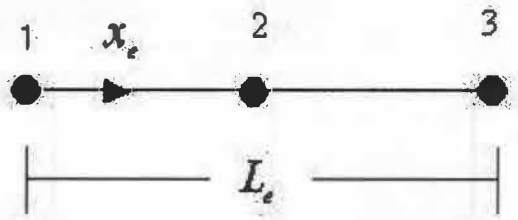

Figure 5.5: Quadratic Element

5.1.2 One Dimensional Beam

The one dimensional beam tutorial takes the user through a 1D Euler Bernoulli beam with a constant load w_o, applied along the top. The beam has a length L, and material properties E and I. The beam is fixed at the left end and free at the right end. The problem is solved using two uniform elements derived from the standard beam shape functions. The GDE is

$$EI\frac{d^4v(x)}{dx^4} + w_o = 0, \tag{5.5}$$

where $v(x)$ is the transverse displacement of the beam. The boundary conditions are

$$v(0) = 0, \quad \left.\frac{dv}{dx}\right|_{x=L} = 0, \quad \left.\frac{d^2v}{dx^2}\right|_{x=L} = 0, \quad \left.\frac{d^3v}{dx^3}\right|_{x=L} = 0 \tag{5.6}$$

The Figure is shown in Figure 5.6.

From here the analytical solution is solved and the results are displayed. Then the finite element method is applied. The importance of choosing the number of elements and the accuracy of these results obtained using different elements is described.

Two Standard Beam Elements

The beam is discretized into two standard Euler-Bernoulli elements shown in Figure 5.7.

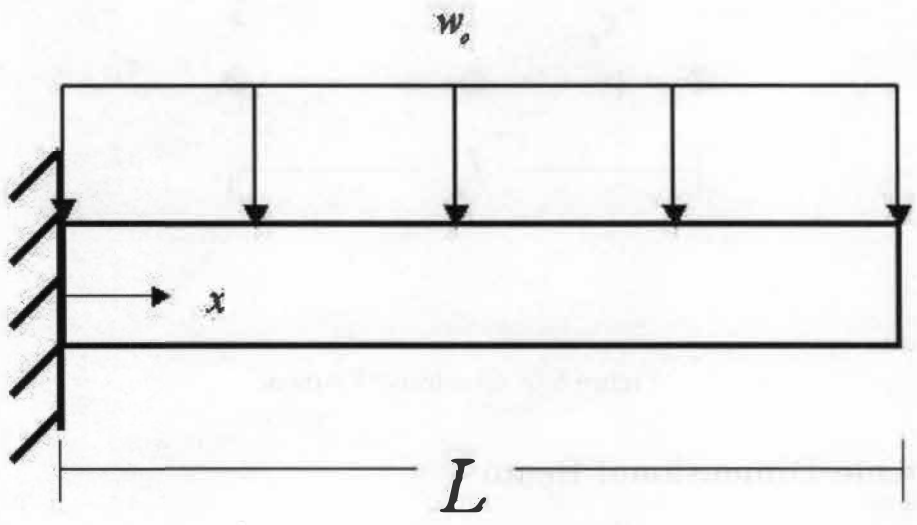

Figure 5.6: 1D Cantilevered Beam

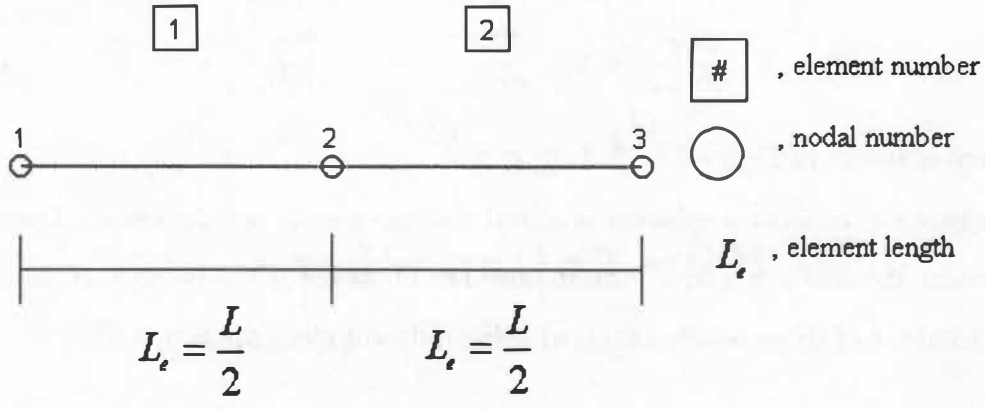

Figure 5.7: 1D Beam Discretization

The beam element is taken from the standard beam shape functions, N,

$$N^T = \begin{bmatrix} \frac{1}{L_e^3}(2x_e^3 - 3x_e^2 L_e + L_e^3) \\ \frac{1}{L_e^3}(x_e^3 L_e - 2x_e^2 L_e^2 + x_e L_e^3) \\ \frac{1}{L_e^3}(-2x_e^3 + 3x_e^2 L_e) \\ \frac{1}{L_e^3}(x_e^3 L_e - x_e^2 L_e^2) \end{bmatrix} \tag{5.7}$$

The linear element has a length L_e, and has two nodes, 1 and 2. The single element also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element as in Figure 5.8.

5.1.3 One Dimensional Heat Conduction

The one dimensional heat conduction without convection tutorial takes the user through a 1D bar with a constant source term Q, length L, and constant thermal conductivity k. The bar has a constant temperature T_o on the right end and a constant flux q_o along the left end. The problem is solved using two uniform linear elements and one quadratic element. The picture is shown in Figure 5.9. The GDE is

$$k\frac{d^2T(x)}{dx^2} + Q = 0, \tag{5.8}$$

where $T(x)$ is the temperature function along the bar. The boundary conditions are

$$-k\frac{dT}{dx}\Big|_{x=0} = q_o, \quad T(L) = T_o \tag{5.9}$$

From here the analytical solution is solved and the results are displayed. Then the finite element method is shown using two different types of elements, the linear and quadratic elements. The importance of choosing a shape function and the results obtained using different types of elements is described.

Two Linear Elements

The bar is discretized into two uniform linear elements shown in Figure 5.10.

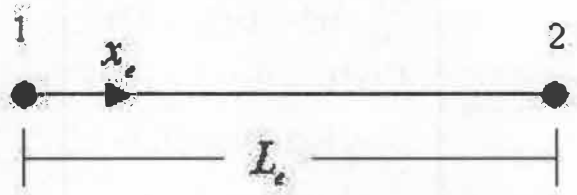

Figure 5.8: Standard Beam Element

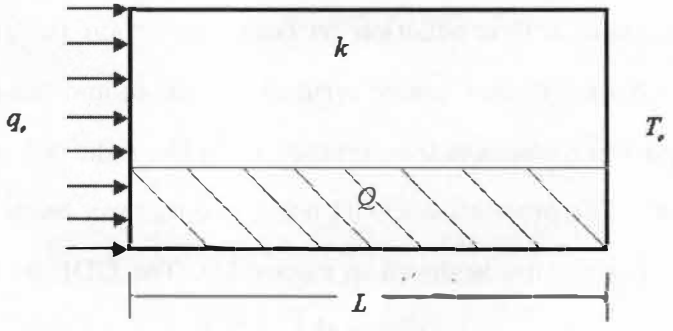

Figure 5.9: 1D Heat Conduction

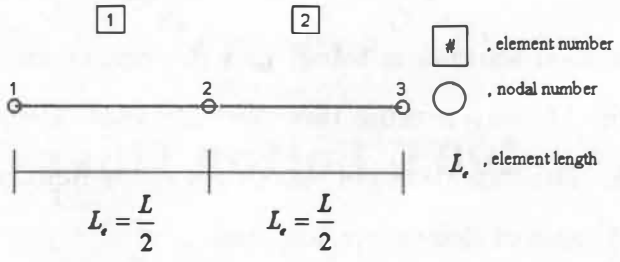

Figure 5.10: 1D Heat Discretization - Two Linear Elements

The linear shape function is chosen and has the form

$$N = \left[\begin{array}{cc} 1 - \frac{x_e}{L_e} & \frac{x_e}{L_e} \end{array} \right] \tag{5.10}$$

The linear element has a length L_e, and has two nodes, 1 and 2. The single element also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element as in Figure 5.11.

One Quadratic Element

The bar is discretized into one quadratic element as in Figure 5.12.

The quadratic shape function is chosen and has the form

$$N = \left[\begin{array}{ccc} 1 - 3\frac{x_e}{L_e} + 2\frac{x_e}{L_e}^2 & 4\frac{x_e}{L_e} - 4\frac{x_e}{L_e}^2 & 2\frac{x_e}{L_e}^2 - \frac{x_e}{L_e} \end{array} \right] \tag{5.11}$$

The quadratic element has a length L_e, and has three nodes, 1,2, and 3. The single element also has a localized coordinate system represented by x_e, which is always situated at node 1 of each element shown in Figure 5.13.

5.1.4 Two Dimensional Plane Stress

The two dimensional plane stress tutorial takes the user through a 2D beam with a constant normal load $-w_o$ applied across the top of the beam, Modulus of Elasticity E, ν is Poisson's ratio, $L = 2h$, x-directional length $4h$, y-directional length $2h$, and a roller applied along the left side of the structure. The coordinate system is located at the center of the beam. The problem is solved using two linear elements. The GDE's are

$$\frac{E}{1-\nu^2}\left(\frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^2 v}{\partial x \partial y}\right) + \frac{E}{2(1+\nu)}\left(\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 v}{\partial x \partial y}\right) = 0 \tag{5.12}$$

and the second one is

$$\frac{E}{1-\nu^2}\left(\frac{\partial^2 v}{\partial y^2} + \nu\frac{\partial^2 u}{\partial x \partial y}\right) + \frac{E}{2(1+\nu)}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y}\right) = 0, \tag{5.13}$$

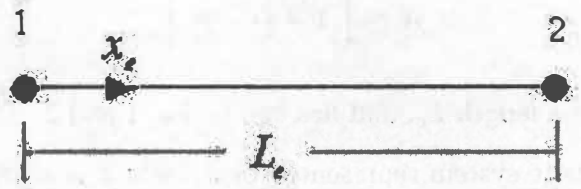

Figure 5.11: Uniform Linear Element

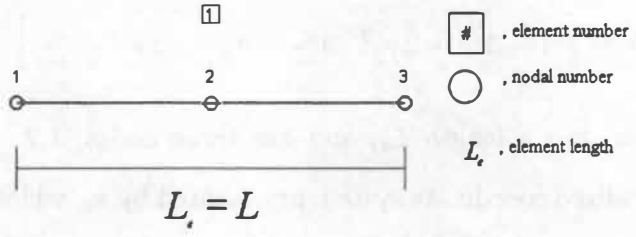

Figure 5.12: 1D Heat Discretization - One Quadratic Element

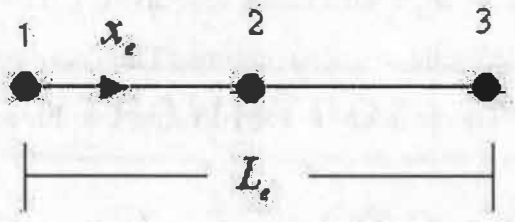

Figure 5.13: Quadratic Element

where $u = u(x, y)$ is the x-directional displacement and $v = v(x, y)$ is the y-directional displacement. The boundary conditions are

$$u(-2h, y) = 0, \quad \text{for all} \quad -h \leq y \leq h, \quad \left.\frac{\partial u}{\partial y}\right|_{x=-2h, y=-h} = 0, \quad \left.\frac{\partial v}{\partial x}\right|_{x=-2h, y=-h} = 0, \tag{5.14}$$

for the roller and for the traction

$$\sigma_{yy}|_{x,y=h} = -w_o \tag{5.15}$$

The picture is shown in Figure 5.14.

From here the analytical solution is solved and the results are displayed. Then the finite element method is shown using two different types of solution methods, Full and Under Integration. The importance of choosing different ways to solve a 2D problem and the results obtained are described.

Two Linear Elements

The beam is discretized into two uniform linear elements shown in Figure 5.15.

The linear shape function is chosen and has the form

$$N^T = \frac{1}{4ab}\begin{bmatrix} (2a - x_e)(b - y_e) & 0 \\ 0 & (2a - x_e)(b - y_e) \\ (2a + x_e)(b - y_e) & 0 \\ 0 & (2a + x_e)(b - y_e) \\ (2a + x_e)(b + y_e) & 0 \\ 0 & (2a + x_e)(b + y_e) \\ (2a - x_e)(b + y_e) & 0 \\ 0 & (2a - x_e)(b + y_e) \end{bmatrix}, \tag{5.16}$$

where a is the element length in the x-direction, b is the element length in the y-direction, and the local coordinate system represented by (x_e, y_e) is always situated at the center of each element as shown in Figure 5.16.

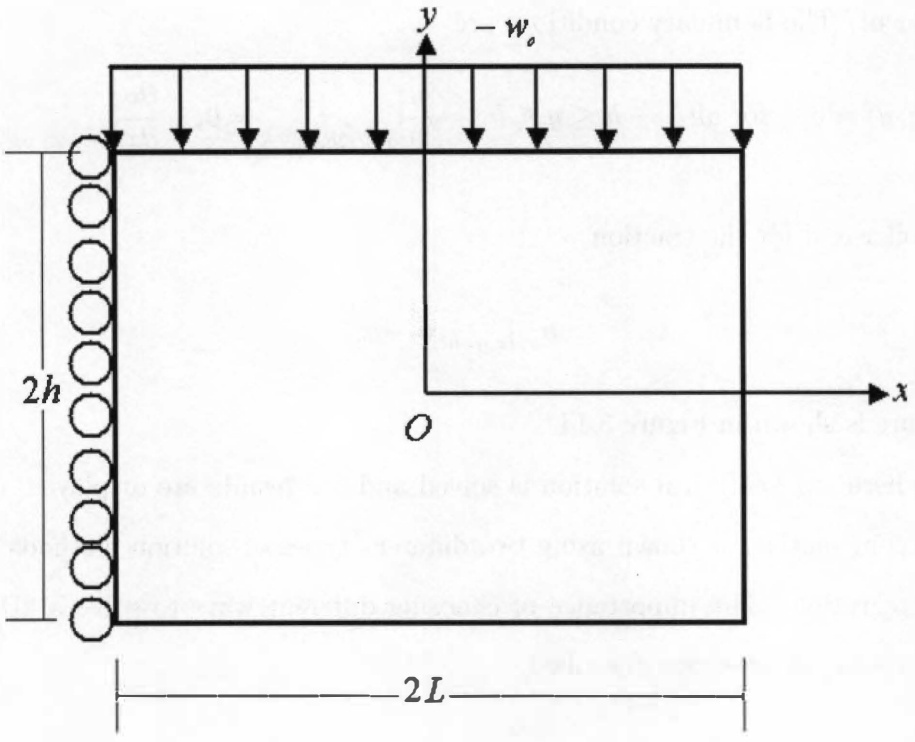

Figure 5.14: 2D Plane Stress

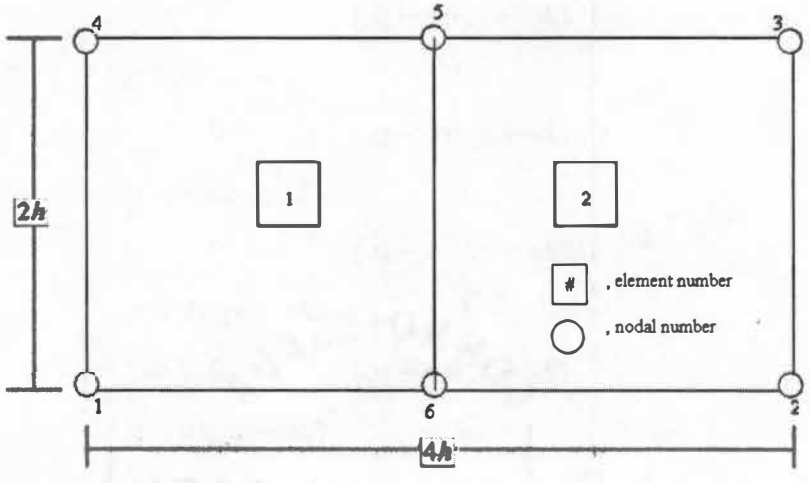

Figure 5.15: 2D Beam Discretization

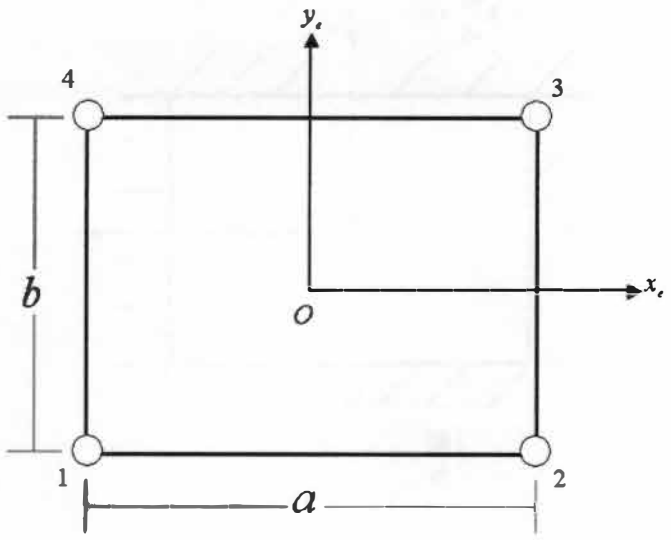

Figure 5.16: Linear Element

5.1.5 Two Dimensional Heat Conduction

The two dimensional heat conduction tutorial takes the user through a 2d bar with a constant thermal conductivity k, zero source Q, x-directional length $2h$, y-directional length $2h$, a constant flux q_o along the right edge and a constant temperature T_o along the left edge. The problem is solved using two linear elements and one quadratic element. The GDE is

$$k\nabla^2 T = 0, \quad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{5.17}$$

The temperature $T = T(x, y)$ is a function of x and y. The boundary conditions are

$$k\nabla T|_{x,y=-h} = 0, \quad k\nabla T|_{x=h,y} = q_o, \quad k\nabla T|_{x,y=h} = 0, \quad T(-h, y) = 0 \tag{5.18}$$

The picture is shown in Figure 5.17.

From here the analytical solution is solved and the results are displayed. Then the finite element method is shown using two different types of elements, the linear and quadratic elements. The importance of choosing a shape function and the results obtained using different types of elements is described.

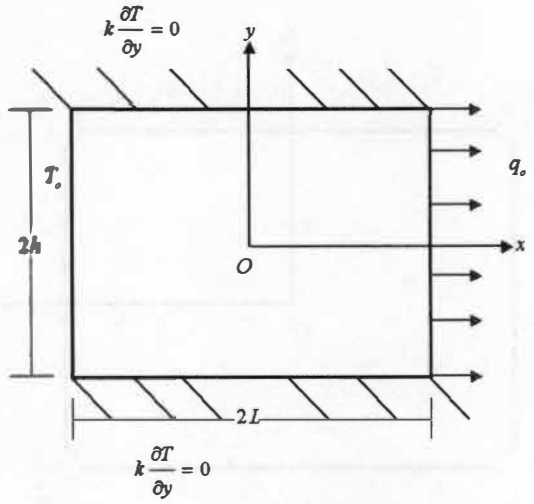

Figure 5.17: 2D Heat Conduction

Four Linear Elements

The bar is discretized into four uniform linear elements shown in Figure 5.18. The linear shape function is chosen and has the form

$$N^T = \frac{1}{4ab}\begin{bmatrix} (2a-x_e)(b-y_e) & 0 \\ 0 & (2a-x_e)(b-y_e) \\ (2a+x_e)(b-y_e) & 0 \\ 0 & (2a+x_e)(b-y_e) \\ (2a+x_e)(b+y_e) & 0 \\ 0 & (2a+x_e)(b+y_e) \\ (2a-x_e)(b+y_e) & 0 \\ 0 & (2a-x_e)(b+y_e) \end{bmatrix}, \tag{5.19}$$

where a is the element length in the x-direction, b is the element length in the y-direction, has four nodes, and the local coordinate system represented by (x_e, y_e) is always situated at the center of each element as shown in Figure 5.19.

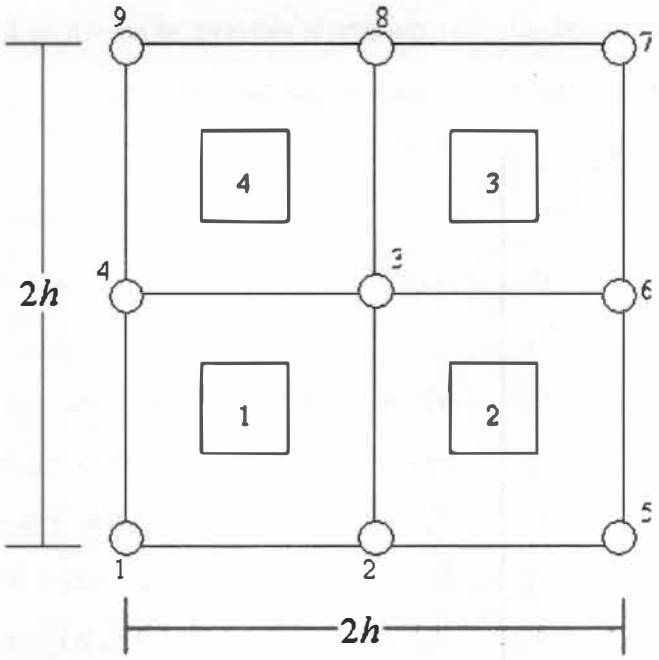

Figure 5.18: 2D Heat Discretization - Four Linear Elements

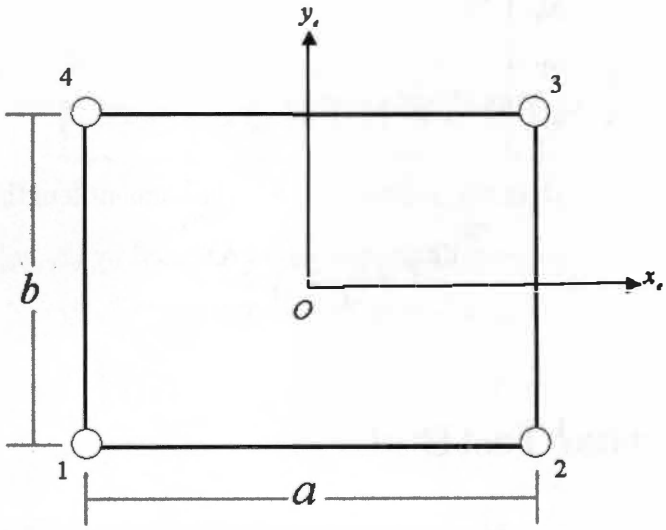

Figure 5.19: Linear Element

One Quadratic Element

The beam is discretized into one quadratic element as shown in Figure 5.20.

The quadratic shape function is chosen and has the form

$$N^T = \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \\ N_4 & 0 \\ 0 & N_4 \\ N_5 & 0 \\ 0 & N_5 \\ N_6 & 0 \\ 0 & N_6 \\ N_7 & 0 \\ 0 & N_7 \\ N_8 & 0 \\ 0 & N_8 \end{bmatrix}, \quad \begin{aligned} N_1 &= \tfrac{1}{4}(1-x_e)(1-y_e)(-x_e-y_e+1) \\ N_2 &= \tfrac{1}{4}(1+x_e)(1-y_e)(x_e-y_e-1) \\ N_3 &= \tfrac{1}{4}(1+x_e)(1+y_e)(x_e+y_e-1) \\ N_4 &= \tfrac{1}{4}(1-x_e)(1+y_e)(-x_e+y_e-1) \\ N_5 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-x_e) \\ N_6 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-y_e) \\ N_7 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-x_e) \\ N_8 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-y_e) \end{aligned}, \tag{5.20}$$

where a is the element length in the x-direction, b is the element length in the y-direction, has eight nodes, and the local coordinate system represented by (x_e, y_e) is always situated at the center of each element as shown in Figure 5.21.

5.2 Interactive TestBed

The following classes of problems are available in the computational testbed section of the website:

- One Dimensional Axial Bar

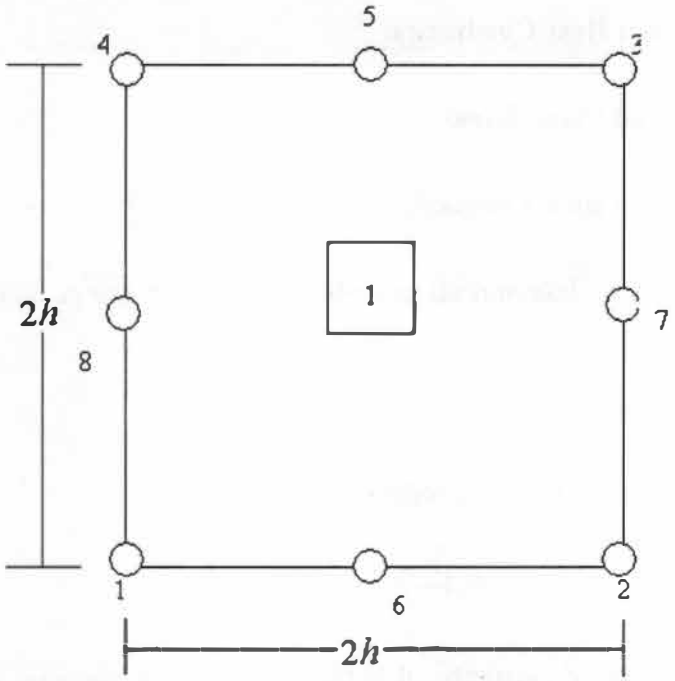

Figure 5.20: 2D Heat Discretization - One Quadratic Element

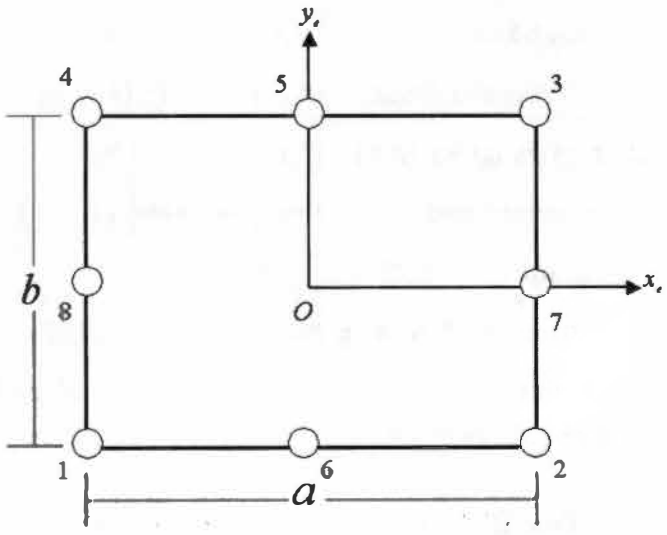

Figure 5.21: Quadratic Element

- One Dimensional Euler-Bernoulli Beam
- One Dimensional Heat Conduction
- Two Dimensional Plane Stress
- Two Dimensional Heat Conduction

Details on each problem class and all available user choices are as follows.

5.2.1 Axial Bar

For the axial bar, the GDE is as follows:

$$EA\frac{du(x)}{dx} + P(x) = 0, \tag{5.21}$$

where E is the modulus of elasticity, A is the area, $P(x)$ is the load applied along the sheath, has a length L, and $u(x)$ is the displacement in the x-direction.

The user can choose from the following four loading functions: constant load, left triangular load, right triangular load, and a polynomial load. The four loads take the form

$$\begin{array}{lrcl} \text{Constant Load} & P(x) & = & P_o \\ \text{Left Triangular Load} & P(x) & = & P_o\left(1-\frac{x}{L}\right) \\ \text{Right Triangular Load} & P(x) & = & P_o\frac{x}{L} \\ \text{Polynomial Load} & P(x) & = & 4P_o\frac{x}{L}\left(1-\frac{x}{L}\right) \end{array} \tag{5.22}$$

The four loads are shown in Figure 5.22.

The user can choose from the following two boundary conditions: fixed at the left end/free at the right end, and fixed at both ends. The two boundary conditions take the form

$$\begin{array}{lll} \text{Fixed Left End, Free Right End} & u(0)=0 & \sigma(L) = E\frac{du}{dx}\Big|_{x=L} = 0 \\ \text{Fixed at Both Ends} & u(0)=0 & u(L)=0 \end{array} \tag{5.23}$$

The two boundary conditions are shown in Figure 5.23.

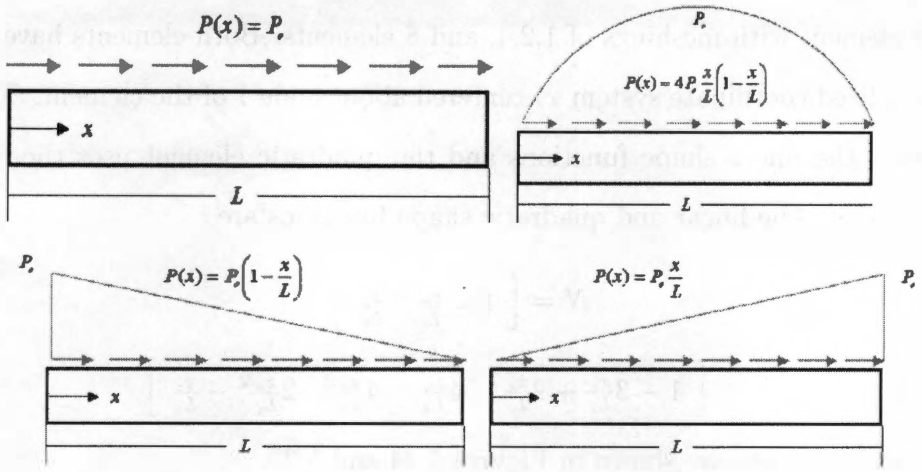

Figure 5.22: 1D Bar Loading Conditions

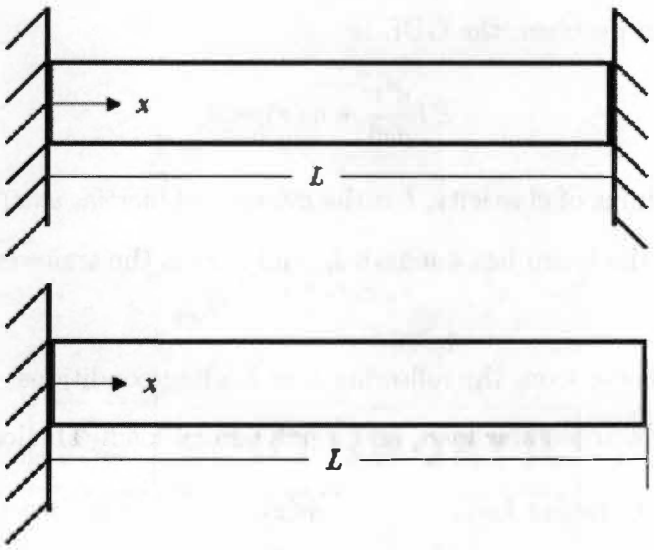

Figure 5.23: 1D Bar Boundary Conditions

The user can choose both the element type and element number. The choices are the two noded linear element with meshings of 2,4,8, and 16 elements, and the three noded quadratic element with meshings of 1,2,4, and 8 elements. Both elements have a length L_e, and localized coordinate system x_e centered about node 1 of the element. The linear element uses the linear shape functions and the quadratic element uses the quadratic shape functions. The linear and quadratic shape functions are

$$N = \left[\begin{array}{cc} 1 - \frac{x_e}{L_e} & \frac{x_e}{L_e} \end{array} \right] \tag{5.24}$$

$$N = \left[\begin{array}{ccc} 1 - 3\frac{x_e}{L_e} + 2\frac{x_e}{L_e}^2 & 4\frac{x_e}{L_e} - 4\frac{x_e}{L_e}^2 & 2\frac{x_e}{L_e}^2 - \frac{x_e}{L_e} \end{array} \right] \tag{5.25}$$

The two element types are shown in Figures 5.24 and 5.25.

The element meshing choices are shown in Figure 5.26 and 5.27. All combinations of load, boundary conditions, element type, and element meshings are possible. This results in 64 possible problems that the user can investigate.

5.2.2 Euler-Bernoulli Beam

For the 1D Beam problem, the GDE is

$$EI\frac{d^4v}{dx^4} + w(x) = 0, \tag{5.26}$$

where E is the modulus of elasticity, I is the moment of inertia, $w(x)$ is the load applied along the top, and the beam has a length L, and $v(x)$ is the transverse displacement of the beam.

The user can choose from the following four loading conditions: constant load, left triangular load, right triangular load, and a polynomial load. The loads take the form

$$\begin{array}{lrcl} \text{Constant Load} & w(x) & = & w_o \\ \text{Left Triangular Load} & w(x) & = & w_o\left(1 - \frac{x}{L}\right) \\ \text{Right Triangular Load} & w(x) & = & w_o\frac{x}{L} \\ \text{Polynomial Load} & w(x) & = & 4w_o\frac{x}{L}\left(1 - \frac{x}{L}\right) \end{array} \tag{5.27}$$

The four loads are shown in Figure 5.28.

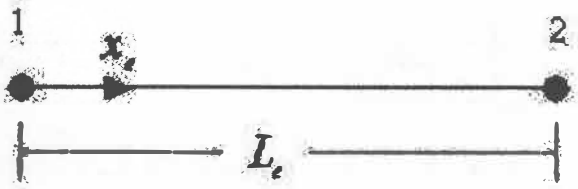

Figure 5.24: Uniform Linear Element

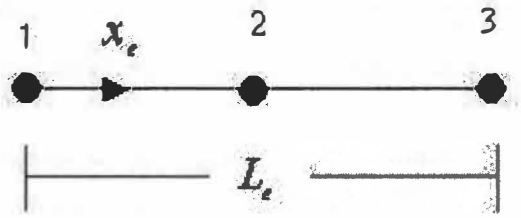

Figure 5.25: Quadratic Element

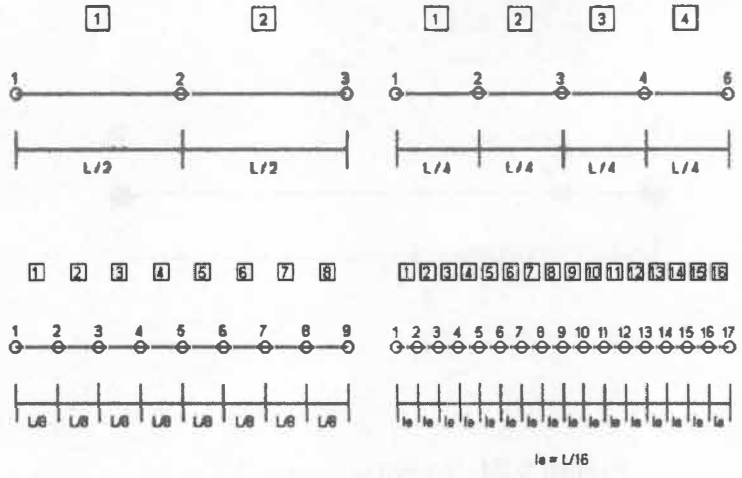

Figure 5.26: 1D Bar - Linear Element Meshings

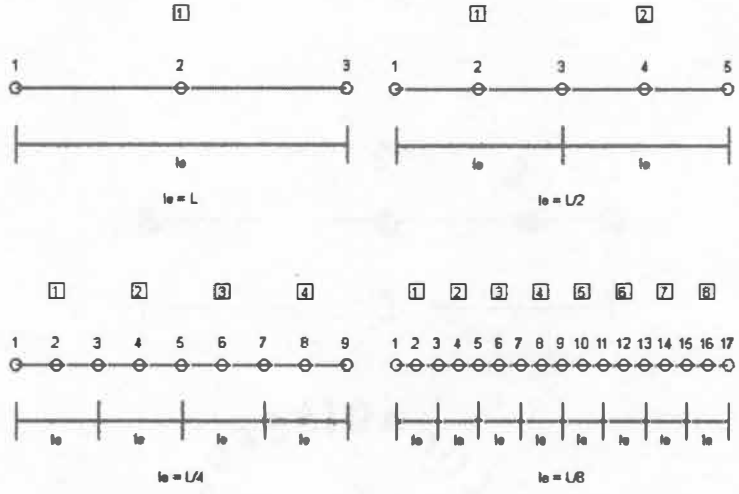

Figure 5.27: 1D Bar - Quadratic Element Meshings

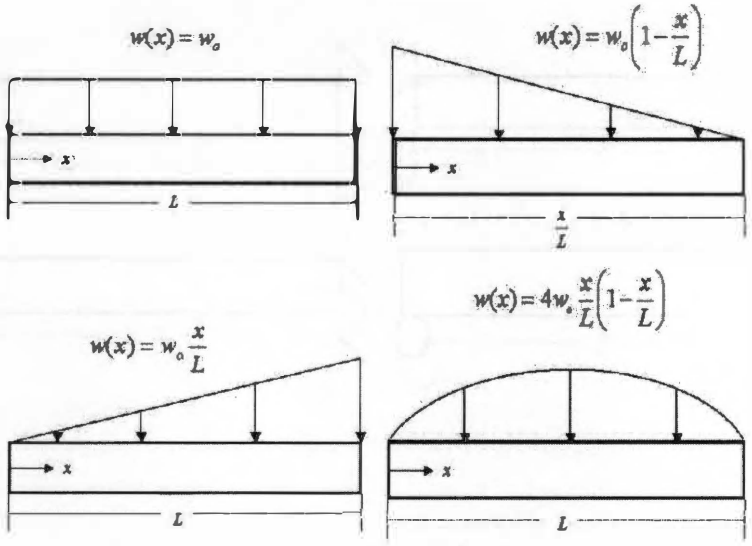

Figure 5.28: Beam Loading Conditions

The user can choose from the following four boundary conditions: fixed at the left end/free at the right end, fixed at both ends, fixed at the left end/a roller at the right end, pin at the left end/roller at the right end. The four boundary conditions take the form

$$\begin{array}{lllll}
\text{Fixed Both} & v(0) = 0 & \left.\frac{dv}{dx}\right|_{x=0} = 0 & v(L) = 0 & \left.\frac{dv}{dx}\right|_{x=L} = 0 \\
\text{Fix Left Free Right} & v(0) = 0 & \left.\frac{dv}{dx}\right|_{x=0} = 0 & \left.\frac{d^2v}{dx^2}\right|_{x=L} = 0 & \left.\frac{d^3v}{dx^3}\right|_{x=L} = 0 \\
\text{Fix Left Roller Right} & v(0) = 0 & \left.\frac{dv}{dx}\right|_{x=0} = 0 & v(L) = 0 & \left.\frac{d^2v}{dx^2}\right|_{x=L} = 0 \\
\text{Pin Left Roller Right} & v(0) = 0 & v(L) = 0 & \left.\frac{d^2v}{dx^2}\right|_{x=0} = 0 & \left.\frac{d^2v}{dx^2}\right|_{x=L} = 0
\end{array} \tag{5.28}$$

The four boundary conditions are shown in Figure 5.29.

The user does not have a choice of element types, but does have a choice of element meshings with 2,4,8 and 16 elements. The standard beam shape functions are used and

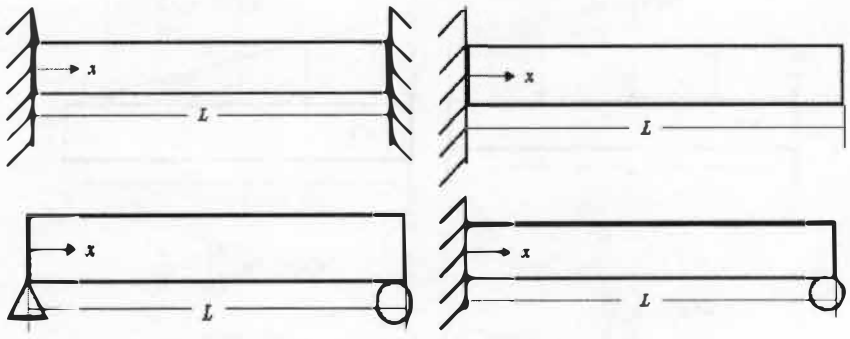

Figure 5.29: Beam Boundary Conditions

have the form

$$N^T = \begin{bmatrix} \frac{1}{L_e^3}(2x_e^3 - 3x_e^2 L_e + L_e^3) \\ \frac{1}{L_e^3}(x_e^3 L_e - 2x_e^2 L_e^2 + x_e L_e^3) \\ \frac{1}{L_e^3}(-2x_e^3 + 3x_e^2 L_e) \\ \frac{1}{L_e^3}(x_e^3 L_e - x_e^2 L_e^2) \end{bmatrix} \tag{5.29}$$

The element meshings are shown in Figure 5.30. All of these combinations of load, boundary condtions, and element mesh size present the user with 64 possible problems to investigate.

5.2.3 Heat Conduction

For heat conduction, the GDE is as follows:

$$k\frac{d^2T(x)}{dx^2} + Q(x) = 0, \tag{5.30}$$

where k is the thermal conductivity, Q is the source, and $T(x)$ is the temperature. The bar has a length L.

The user can choose from the following four source functions: constant source, left triangular source, right triangular source, and a polynomial source. The four sources

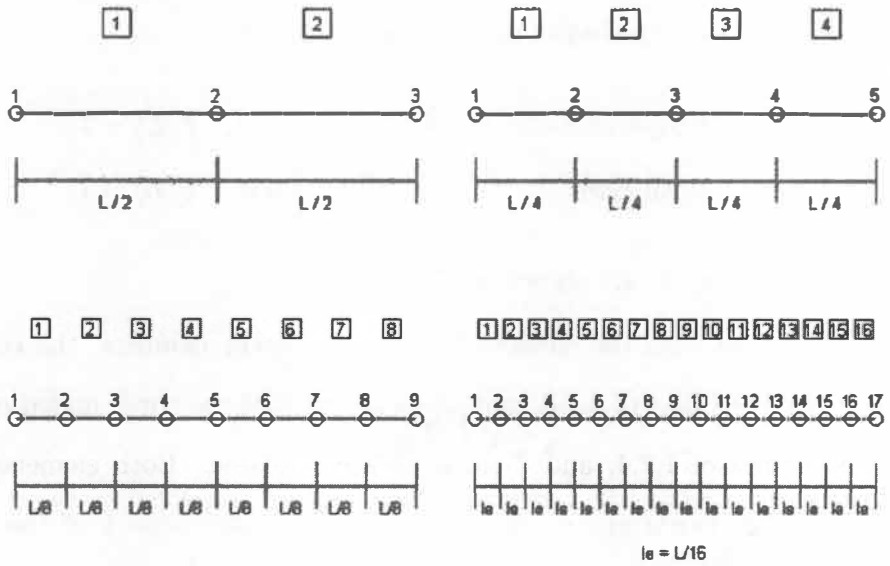

Figure 5.30: 1D Beam - Element Mesh Choices

take the form

$$\begin{array}{rrcl} \text{Constant Source} & Q(x) & = & Q_o \\ \text{Left Triangular Source} & Q(x) & = & Q_o\left(1-\frac{x}{L}\right) \\ \text{Right Triangular Source} & Q(x) & = & Q_o\frac{x}{L} \\ \text{Polynomial Source} & Q(x) & = & 4Q_o\frac{x}{L}\left(1-\frac{x}{L}\right) \end{array} \tag{5.31}$$

The four sources are shown in Figure 5.31.

The user can choose from the following two boundary conditions: flux at the left end/fixed temperature at the right end and insulated at the left end/ fixed temperature at the right end. The two boundary conditions take the form

$$\begin{array}{rrr} \text{Flux Left Fixed Temperature Right} & -k\frac{dT}{dx}\Big|_{x=0} = -q_o & T(L) = T_o \\ \text{Fixed at Both Ends} & -k\frac{dT}{dx}\Big|_{x=0} = 0 & T(L) = T_o \end{array} \tag{5.32}$$

The two boundary conditions are shown in Figure 5.32.

The user can choose from the element types and element numbers, the two noded linear element with meshings of 2,4,8, and 16 elements, and the three noded quadratic element with meshings of 1,2,4, and 8 elements are available. Both elements have a length L_e, and localized coordinate system x_e centered about node 1 of the element. The linear element uses the linear shape functions and the quadratic element uses the quadratic shape functions. The linear and quadratic shape functions are

$$N = \left[\begin{array}{cc} 1-\frac{x_e}{L_e} & \frac{x_e}{L_e} \end{array}\right] \tag{5.33}$$

$$N = \left[\begin{array}{ccc} 1-3\frac{x_e}{L_e}+2\frac{{x_e}^2}{{L_e}^2} & 4\frac{x_e}{L_e}-4\frac{{x_e}^2}{{L_e}^2} & 2\frac{{x_e}^2}{{L_e}^2}-\frac{x_e}{L_e} \end{array}\right] \tag{5.34}$$

The two element types are shown in Figures 5.33 and 5.34.

The element meshing choices are shown in Figure 5.35 and 5.36. All of these combinations of load, bc's, element type, and element meshings are possible choices. This results in 64 possible problems that the user can investigate.

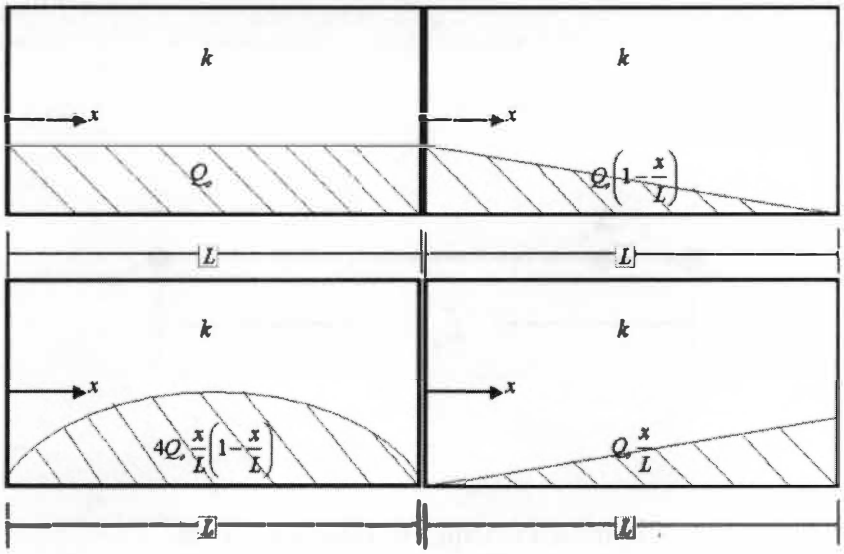

Figure 5.31: 1D Heat Source Conditions

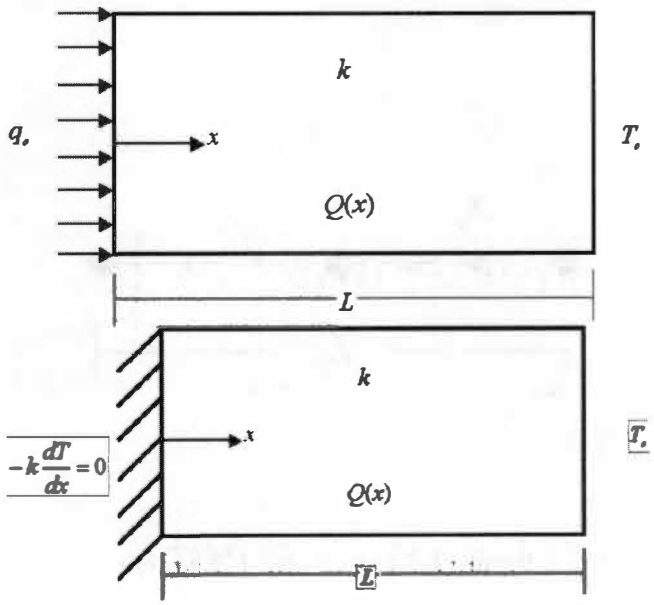

Figure 5.32: 1D Heat Boundary Conditions

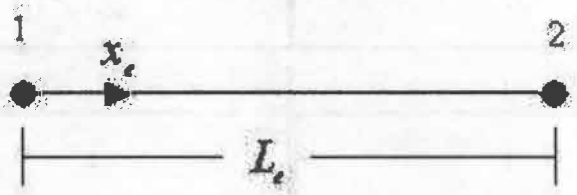

Figure 5.33: Uniform Linear Element

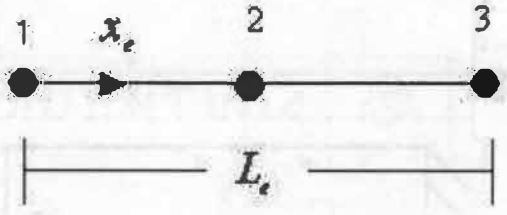

Figure 5.34: Quadratic Element

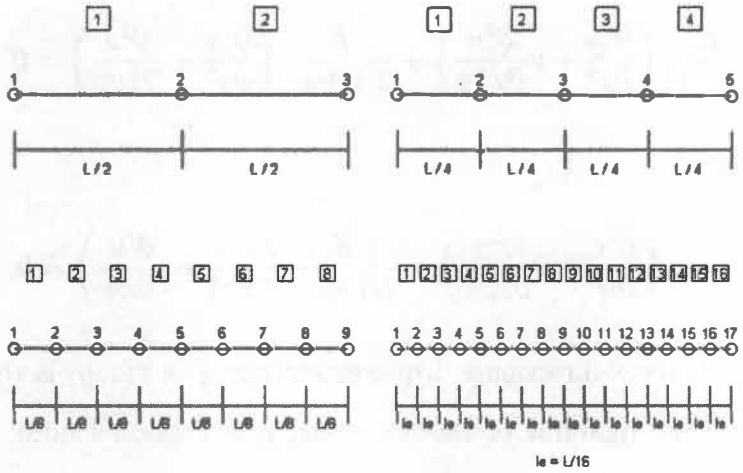

Figure 5.35: 1D Heat - Linear Element Meshings

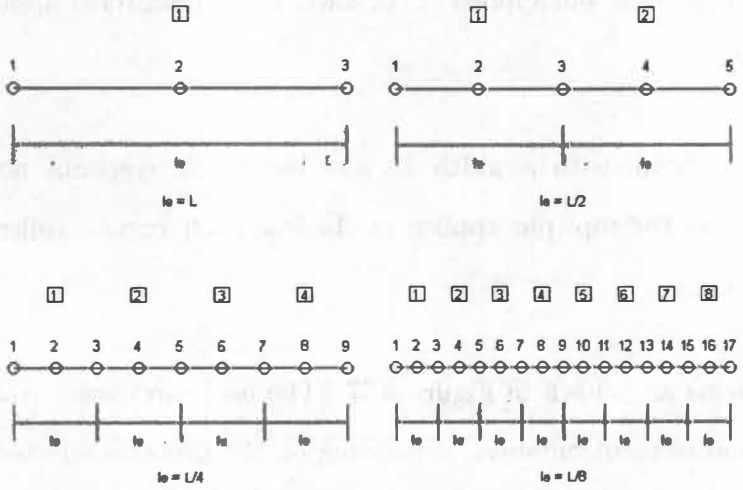

Figure 5.36: 1D Heat - Quadratic Element Meshings

5.2.4 Two Dimensional Plane Stress

For 2D Plane Stress, the GDE's are:

$$\frac{E}{1-\nu^2}\left(\frac{\partial^2 u}{\partial x^2}+\nu\frac{\partial^2 v}{\partial x \partial y}\right)+\frac{E}{2(1+\nu)}\left(\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 v}{\partial x \partial y}\right)=0 \tag{5.35}$$

and the second one is

$$\frac{E}{1-\nu^2}\left(\frac{\partial^2 v}{\partial y^2}+\nu\frac{\partial^2 u}{\partial x \partial y}\right)+\frac{E}{2(1+\nu)}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 u}{\partial x \partial y}\right)=0, \tag{5.36}$$

where $u = u(x, y)$ is the x-directional displacement and $v = v(x, y)$ is the y-directional displacement, E is the modulus of elasticity, and ν is Poisson's ratio. The user can choose from the following three problems:

1. Square beam with a length h, constant normal load w_o applied across the top, pin applied at the lower left corner, roller applied at the upper left corner.

2. Rectangular beam with a width $4h$ and height $2h$, constant shear load P_o applied across the right face, pin applied at the lower left corner, roller applied at the upper left corner.

3. Rectangular beam with a width $4h$ and height $2h$, constant normal load $-w_o$ applied across the top, pin applied at the lower left corner, roller applied at the upper left corner.

The three problems are shown in Figure 5.37. The user can choose from the following element types and element numbers, depending on the problem selected. The user has the option of the four noded linear element with meshings of 1,2,4 and 8 elements, and the 8 noded quadratic element with meshings of 1 and 2 elements are available. Both elements have a localized coordinate system, (x_e, y_e), centered about the origin. The linear element uses the linear shape functions, and the quadratic element uses the

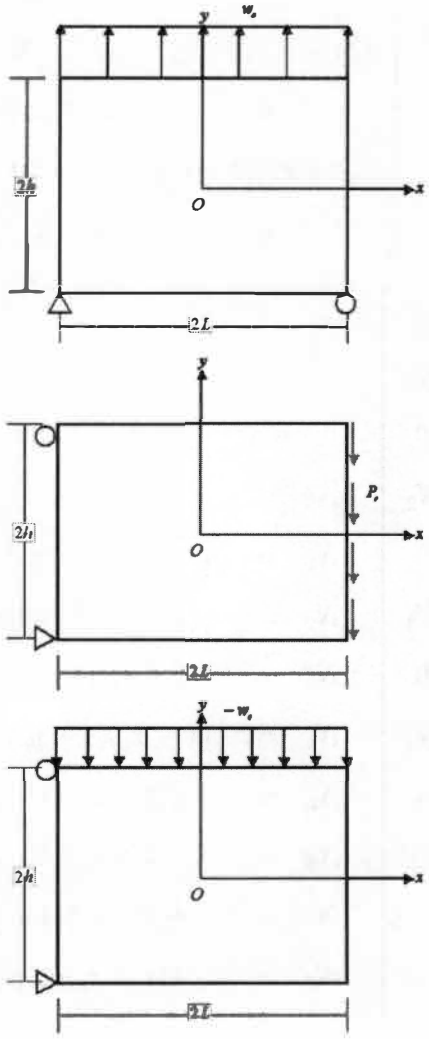

Figure 5.37: 2D Plane Stress Problems

quadratic shape functions. The linear and quadratic shape functions are

$$N^T = \frac{1}{4ab}\begin{bmatrix} (2a-x_e)(b-y_e) & 0 \\ 0 & (2a-x_e)(b-y_e) \\ (2a+x_e)(b-y_e) & 0 \\ 0 & (2a+x_e)(b-y_e) \\ (2a+x_e)(b+y_e) & 0 \\ 0 & (2a+x_e)(b+y_e) \\ (2a-x_e)(b+y_e) & 0 \\ 0 & (2a-x_e)(b+y_e) \end{bmatrix} \tag{5.37}$$

$$N^T = \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \\ N_4 & 0 \\ 0 & N_4 \\ N_5 & 0 \\ 0 & N_5 \\ N_6 & 0 \\ 0 & N_6 \\ N_7 & 0 \\ 0 & N_7 \\ N_8 & 0 \\ 0 & N_8 \end{bmatrix}, \quad \begin{aligned} N_1 &= \tfrac{1}{4}(1-x_e)(1-y_e)(-x_e-y_e+1) \\ N_2 &= \tfrac{1}{4}(1+x_e)(1-y_e)(x_e-y_e-1) \\ N_3 &= \tfrac{1}{4}(1+x_e)(1+y_e)(x_e+y_e-1) \\ N_4 &= \tfrac{1}{4}(1-x_e)(1+y_e)(-x_e+y_e-1) \\ N_5 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-x_e) \\ N_6 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-y_e) \\ N_7 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-x_e) \\ N_8 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-y_e) \end{aligned} \tag{5.38}$$

The two element types are shown in Figures 5.38 and 5.39.

The user can choose between solving the problem using full or under integration. Any given problem can be numerically integrated to obtain an approximate solution. For these

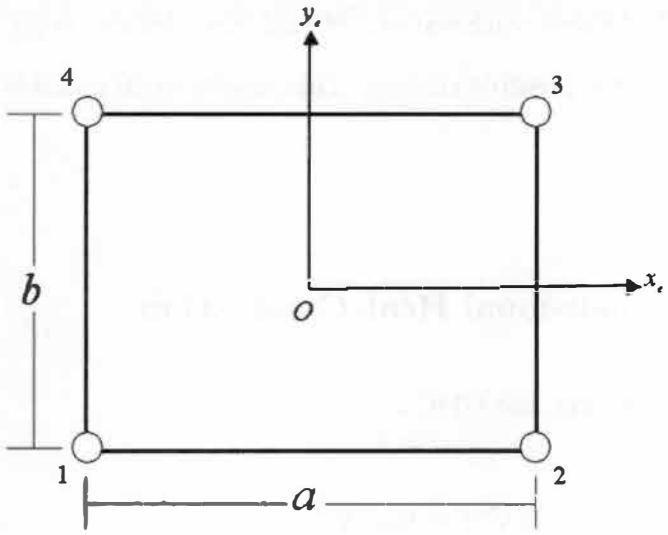

Figure 5.38: Linear Element

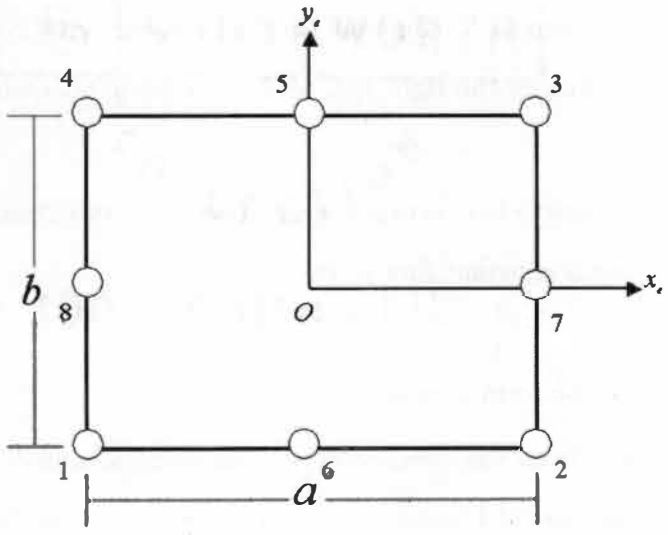

Figure 5.39: Quadratic Element

problems, Gaussian Quadrature is chosen to perform the numerical integration. For an in-depth explanation of Gaussian Quadrature and Under Integration, see the section in the Appendix. All of these combinations of problem, element type, element meshings, and integration type are possible choices. This results in 26 possible problems that the user can investigate.

5.2.5 Two Dimensional Heat Conduction

For 2D heat conduction, the GDE is:

$$k\nabla^2 T = 0, \quad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \tag{5.39}$$

where $T = T(x, y)$ is the temperature function of x and y and k is constant thermal conductivity.

The user can choose from the following two problems:

1. Square bar with a length h, constant thermal conductivity k, insulated top and bottom, constant flux at the right end, and constant temperature at the left end.

2. Square bar with a length h, constant thermal conductivity k, insulated top, bottom, and left sides, and a constant flux at the right end.

The two problems take the form shown in Figure 5.40.

The user can choose from the element types and element numbers, the four noded linear element with meshings of 1 and 4, and the 8 noded quadratic element with a meshing of 1 element. Both elements have a localized coordinate system, (x_e, y_e), centered about the origin. The linear element uses the linear shape functions, and the quadratic element uses the quadratic shape functions. The linear and quadratic shape functions

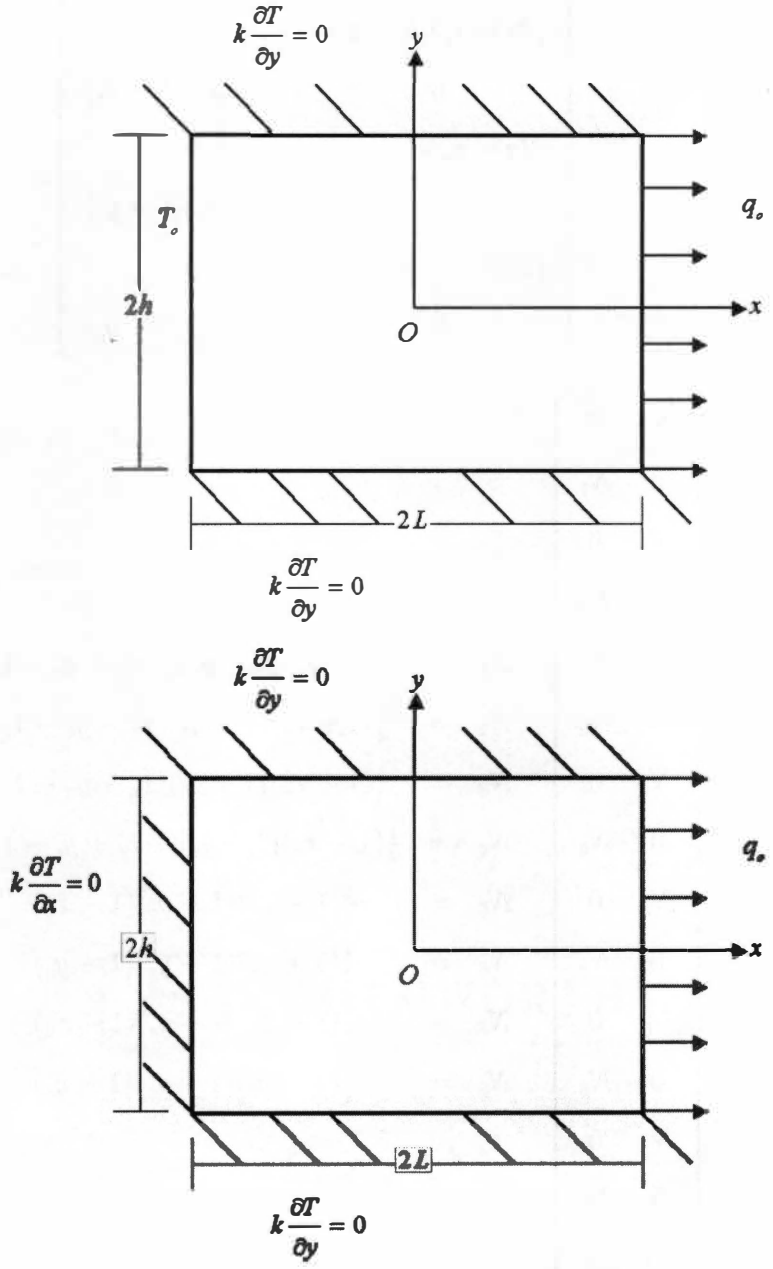

Figure 5.40: 2D Heat - All Problems

are

$$N^T = \frac{1}{4ab}\begin{bmatrix} (2a-x_e)(b-y_e) & 0 \\ 0 & (2a-x_e)(b-y_e) \\ (2a+x_e)(b-y_e) & 0 \\ 0 & (2a+x_e)(b-y_e) \\ (2a+x_e)(b+y_e) & 0 \\ 0 & (2a+x_e)(b+y_e) \\ (2a-x_e)(b+y_e) & 0 \\ 0 & (2a-x_e)(b+y_e) \end{bmatrix} \tag{5.40}$$

$$N^T = \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \\ N_4 & 0 \\ 0 & N_4 \\ N_5 & 0 \\ 0 & N_5 \\ N_6 & 0 \\ 0 & N_6 \\ N_7 & 0 \\ 0 & N_7 \\ N_8 & 0 \\ 0 & N_8 \end{bmatrix}, \quad \begin{aligned} N_1 &= \tfrac{1}{4}(1-x_e)(1-y_e)(-x_e-y_e+1) \\ N_2 &= \tfrac{1}{4}(1+x_e)(1-y_e)(x_e-y_e-1) \\ N_3 &= \tfrac{1}{4}(1+x_e)(1+y_e)(x_e+y_e-1) \\ N_4 &= \tfrac{1}{4}(1-x_e)(1+y_e)(-x_e+y_e-1) \\ N_5 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-x_e) \\ N_6 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-y_e) \\ N_7 &= \tfrac{1}{2}(1+x_e)(1+y_e)(1-x_e) \\ N_8 &= \tfrac{1}{2}(1-x_e)(1+y_e)(1-y_e) \end{aligned} \tag{5.41}$$

The two elements are shown in Figures 5.41 and 5.42.

The user can choose between solving the problem using full or under integration. Any given problem can be numerically integrated to obtain an approximate solution. For these

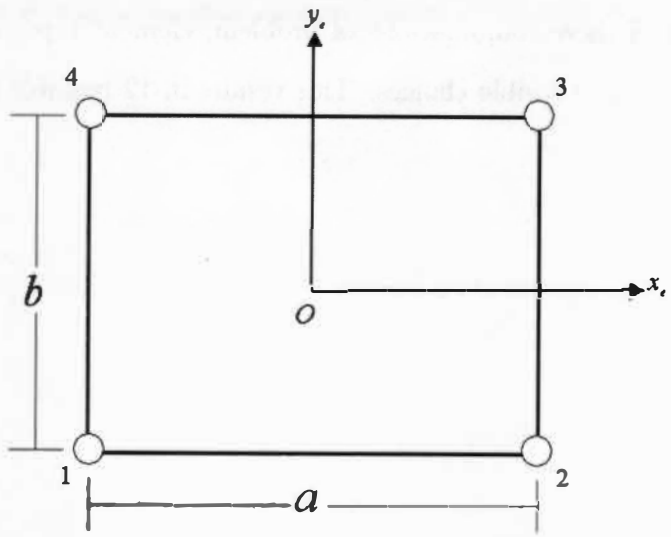

Figure 5.41: Linear Element

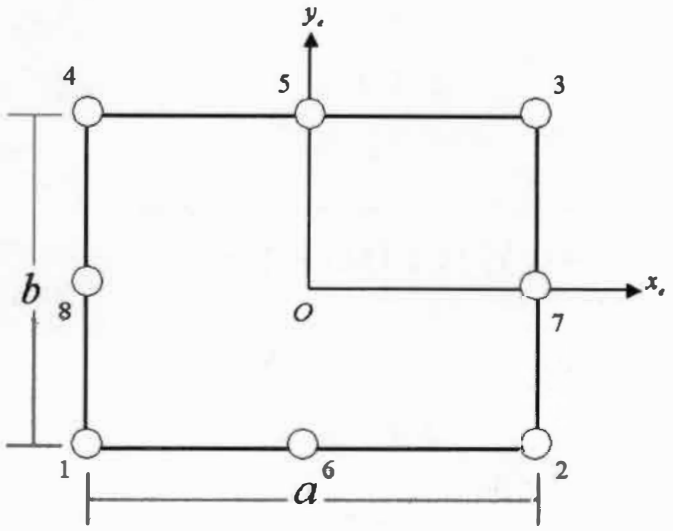

Figure 5.42: Quadratic Element

problems, Gaussian Quadrature is chosen to perform the numerical integration. For an in-depth explanation of Gaussian Quadrature and Under Integration, see the section in the Appendix. All of these combinations of problem, element type, element meshings, and integration type are possible choices. This results in 12 possible problems that the user can investigate.

Chapter 6

Conclusions

While working on this thesis, the author noticed a number of things related to the teaching/learning of finite elements (FE). First, it is very hard to find some basic FE concepts online. It is just as hard to find specific concepts or examples in books or at the library. During coursework, the student may have a good textbook or not, and that determines a lot of what they will learn. Whether the student has a good professor or not determines the rest. Having somthing to fall back on for simple, yet detailed instructions on introductory FE concepts is important. Even the best books cannot cover everything. Most material presents the simplest cases of solvable problems and doesn't provide a good assortment of basic solvable problems. So with this information in hand, the author set about to write this thesis and website.

Upon completing this project, the author has noticed a few more things. Once available, this site provides information regarding introductory finite elements in a analytical and numerical way. It is a resource which provides a good reference to any course discussing introductory finite elements or as a handy primer to look back at certain concepts that an advanced user may have forgotten. Regardless of the textbook or professor, this site helps to fill in gaps that may have been overlooked and to reinforce concepts that have been covered. Lastly, the interactive aspect of this website provides the user the ability to quickly solve different problems with little difficulty. The author is happy with

the final project, although there were many problems encountered along the way

The biggest problem was trying to determine the level of interactivity. Originally, the interactive section would allow students to enter their GDE System in a equation editor environment, select their element type and number, and solve their developed system. The biggest problem with this, was it may take a person many tries to enter the system correctly, or even enter a problem that is not feasable or realistic. So this amount of user freedom was not viable.

From here, the interactive section allowed the user to choose their problem class, say 1D Beam, and then any load function they wanted in an equation editor environment. From here, they would enter the number of dirichlet and neumann boundary conditions along with element information. Then their problem was solved and the results generated back to the user. This solution worked except if the user didn't enter their load function in correctly, the results would not be what they expected. Also, if the user enters a unrealistic function, the results generated won't tell the user anything useful and they won't learn anything from this page. The whole point of this site is to allow students to learn while they play around with different problems.

Therefore, more restriction was required. Along with restricting the problem, immediate computation of a problem had to be removed as well. The reason for this is cpu usage. A 16 element 1D Beam solution takes approximately an hour and a half to compute. This is on a stand alone computer, not across a network on a Celeron 1.7GHz, 256MB RAM, and Windows XP or RedHat Linux 8.0 environment. Expecting a user to sit around for an hour while their problem is solved is not feasable. Also, expecting a server to be dedicated to computing solutions for an hour per user for a unlimited number of users would not be realistic. So, with great respect for the network administrator, all the solutions to the interactive problems had to be solved ahead of time and presented to the user accordingly. All the results are presented as pictures or text.

Equations also had to be displayed as pictures. The original idea was to use MathML, the emerging web standard set forth by the w3c [11]. The problem with this idea, is

that MathML is still in its early stages, and current web browsers do not offer complete compatability. MathML has not been enabled in html yet, and would require xml for success. XML has not been totally synched with javascript when used with internet explorer, so users would have to use netscape or mozilla [9].

To further restrict the user to achieve total MathML compatability, the user would have to edit security settings in IE, install fonts for IE and netscape to use to display the MathML characters, and install a plugin for certain versions of netscape and IE so that the MathML enabled XML namespaced XHTML Class of HTML can properly display the equations. This can be very confusing and frustrating.

So, this is why pictures were used for the equations. Just about everyone can use any browser and a good internet connection to view this website, and see all of its features without any configuration changes needed on their part. The only requirements is that the user has a little FE background or equivalent background and the minimum knowledge required to use a web browser to browse the internet.

In the future, as technology improves, internet standards get futher developed, and all browsers are forced to comply with these standards, a future grad student may embark on the task of completing the vision of a completely interactive testbed capable of solving any equation system input into it and quickly calculate and display the results in the simplest manner possible. In the meantime the current program and format could be expanded to include such items as 3D elements, dynamic or transient problems, and other classs of GDE's such as electricity and magnetism, etc.

Bibliography

[1] *ABAQUS* - Finite Element Software, online webpages at `http://www.hks.com`

[2] *ANSYS* - Finite Element Software, online webpages at `http://www.ansys.com`

[3] *COSMOS* - Finite Element Software, online webpages at `http://www.cosmosm.com`

[4] *Engineering Science 551w* - Online Finite Element Course, online webpages at `http://cfdlab.engr.utk.edu/~551w/`

[5] Logan, Daryl L. **A First Course in the Finite Element Method**. Brooks/Cole Publishing Company, 2001

[6] *Maple* - Symbolic & Numerical Mathematical Software Package, online webpages at `http://www.maplesoft.com`

[7] *Matlab* - Numerical Mathematical Software Package specializes in matrices, online webpages at `http://www.mathworks.com`

[8] *Mathematica* - Numerical Mathematical Software Package with excellent web integration, online webpages at `http://www.mathematica.com`

[9] Mozilla - Open Source Web Browser that Netscape is based off of which adheres to the w3c standards, online webpages at `http://www.mozilla.org`

[10] *PDELab* - Interactive Anayltical Partial Differential Equation Solver WebSite, online webpages at `http://www.webpdelab.org`

[11] World Wide Web Consortium (W3C) - Organization working to create uniform web standards and force web browsers to adopt this standard, online webpages at `http://www.w3c.org`

Appendices

Appendix A

Gaussian Quadrature Theory

Gaussian Quadrature is used to implement under integration. It is also possible to use Gaussian Quadrature to fully integrate a solution. This appendix goes through the theory and application of Full and Under Integration as it is used in this theory.

A.1 Full Integration

For the finite element methods used in this thesis, full integration is equivalent to analytically solving the stiffness matrix.

$$k_e = t \int_{-h}^{h} \int_{-h}^{h} B^T D B dx_e dy_e, \tag{A.1}$$

where

$$B^T = \frac{1}{4h^2} \begin{bmatrix} -(h-y) & 0 & -(h-x) \\ 0 & -(h-x) & -(h-y) \\ h-y & 0 & -(h+x) \\ 0 & -(h+x) & h-y \\ h+y & 0 & h+x \\ 0 & h+x & h+y \\ -(h+y) & 0 & h-x \\ 0 & h-x & -(h+y) \end{bmatrix}, D = E \begin{bmatrix} \frac{1}{1-\nu^2} & \frac{\nu}{1-\nu^2} & 0 \\ \frac{\nu}{1-\nu^2} & \frac{1}{1-\nu^2} & 0 \\ 0 & 0 & \frac{1}{2}\frac{1-\nu}{1-\nu^2} \end{bmatrix} \tag{A.2}$$

This equation is simply solved by performing the integrations and the element stiffness matrix is found accordingly. Along with this method, Gaussian Quadrature could be performed to fully solve the integration. Quadrature theory predicts that to fully integrate any (2n-1)-polynomial function it is necessary to implement a n-pt. Quadrature rule.

For example, given the linear polynomial $z(x,y) = 1+2x+2y+4xy$, a 1-pt. Quadrature rule can be applied to solve this integral exactly. The integral of this function is

$$I = \int_{-h}^{h}\int_{-h}^{h} 1+2x+2y+4xydxdy \tag{A.3}$$

Solving this integral leads to $I = 4h^2$. To solve this linear polynomial integral, it is only necessary to apply a 1-pt quadrature rule, i.e. $I = w_x w_y z(x,y)$. From this point, an approximate polynomial is selected that has enough constants to solve for the three unknowns w_x, w_y, and $z(x,y)$. The polynomial selected is

$$z_a = c_o + c_1 x + c_2 y \tag{A.4}$$

Next, solving this approximate integral is performed.

$$A = \int_{-h}^{h}\int_{-h}^{h} c_o + c_1 x + c_2 y dxdy \rightarrow A = 4c_o h^2 \tag{A.5}$$

Now the Guass approximation $A_g = w_x w_y z_a(0,0)$ since our 2D problem has 1-pt. Gaussian which has its point in the center of our beam. Plugging in the value for $z_a(0,0) = c_o$ leads to $A_g = w_x w_y c_o$. The error for any approximation when compared to the exact solution is $e = A - A_g$ and to minimize this error would occur when the derivative of this error is zero. Therefore

$$\frac{\partial e}{\partial c_o} = 0 = 4h^2 - w_x w_y \tag{A.6}$$

For this problem, $w_x = w_y$ since our problem is uniform and as such $w = \pm 2h$. Plugging in this value of w and $z(0,0)$ into the I_g equation leads to $I_g = 4h^2$ which exactly solves the integral using Gauss Quadrature compared to simply integrating the problem.

This is how full integration works. For problems that cannot easily be solved by integrating, Gaussian Quadrature is an effective method to use.

A.2 Under Integration

Under Integration follows the same method as full integration except you choose a fewer number of Guass pts. For example, for a cubic polynomial, a 2-pt. quadrature rule is needed to get an exact solution, but if you choose a 1-pt. quadrature rule, you will still get an approximate solution, and this solution is said to be an under-integrated solution. For example, given the integral of the quadratic polynomial

$$I = \int_{-h}^{h}\int_{-h}^{h} \frac{3}{2}h^2 - \frac{1}{2}\nu h^2 - 2hy - hx + y^2 + \frac{1}{2}x^2 - \frac{1}{2}\nu x^2 dxdy \tag{A.7}$$

which is taken from a term in the element stiffness matrix and setting $E = 1$. Performing a 1-pt quadrature rule on this integral will underintegrate the polynomial. Integrating the polynomial exactly leads to the solution

$$I = 6h^4 - 2\nu h^4 + \frac{4}{6}h^4 - \frac{4}{6}\nu h^4 + \frac{4}{3}h^4 \tag{A.8}$$

Under integrating requires that $I = w_x w_y z(x_o, y_o)$ which is a 1-pt quadrature solution. The approximate polynomial is chosen to be $z_a = c_o + c_1 x + c_2 y$. Integrating this approximate solution yields

$$A = \int_{-h}^{h}\int_{-h}^{h} c_o + c_1 x + c_2 y dxdy = 4c_o h^2 \tag{A.9}$$

As was the case above, the error is equal to A minus A_g, where $A_g = w^2 c_o$. The derivative of the error is equal to zero, which leads to

$$\frac{\partial e}{\partial c_o} = 0 = 4h^2 - w^2 \rightarrow w = \pm 2h \tag{A.10}$$

Again, the gaussian quadrature solution to the integral becomes

$$I_g = w^2 z(0,0) = 4h^2\left(\frac{3}{2}h^2 - \frac{1}{2}\nu h^2\right) = 6h^4 - 2\nu h^4 \tag{A.11}$$

When comparing the 1-pt. quadrature solution to the fully integrated solution, it can be seen that by underintegrating the function, the quadratic polynomial was simply reduced to the constant and the linear terms. But when you integrate a linaer term with

symmetric bounds, i.e. $-h \ldots h$, the linear term becomes quadratic and will drop out of the integral, as shown below.

$$\int_{-h}^{h} x dx = \left[\frac{x^2}{2}\right]_{-h}^{h} = \frac{1}{2}\left[(h)^2 - (-h)^2\right] = 0 \tag{A.12}$$

Therefore the linear terms drop out and we are only left with fully integrating the constant terms. Therefore underintegrating the solution exactly determines the constant terms in the full integrated solution, and therfore we can determine the underintegrated stiffness matrix and use it when necessary.

The method of underintegrating through Gaussian Quadrature is useful for problems such as shear locking. The problem of shear locking occurs when a solution has excessive stiffness, so fully integrating the solution will lead to discontinuities in the solution. By under integrating, we can remove some of that stiffness allowing the problem room to move and remove the shear locking.

Appendix B

Sample Maple Code

B.1 One Dimensional Code

```
> restart:
> with(linalg):
Warning, the protected names norm and trace have been redefined and
unprotected

> printlevel := 3: # this allows for viewing nested loop solutions
>
> nelem := 2:
> nnode := nelem + 1:
>
> # Tell maple to choose which element matrix from library (later)
> eb1dke := matrix([[12*EI/L^3,6*EI/L^2,-12*EI/L^3,6*EI/L^2],
> [6*EI/L^2,4*EI/L,-6*EI/L^2,2*EI/L],
> [-12*EI/L^3,-6*EI/L^2,12*EI/L^3,-6*EI/L^2],
> [6*EI/L^2,2*EI/L,-6*EI/L^2,4*EI/L]]);
```

```
                [    EI         EI          EI         EI  ]
                [12 ----     6 ----     -12 ----    6 ---- ]
                [    3          2           3          2  ]
                [   L          L           L          L   ]
                [                                         ]
                [    EI         EI          EI         EI  ]
                [ 6 ----     4 ----     -6 ----     2 ---- ]
                [    2          L           2          L   ]
                [   L                      L              ]
      eb1dke := [                                         ]
                [    EI         EI          EI         EI  ]
                [-12 ----    -6 ----    12 ----    -6 ----]
                [    3          2           3          2  ]
                [    L          L           L          L  ]
                [                                         ]
                [    EI         EI          EI         EI  ]
                [ 6 ----     2 ----     -6 ----     4 ---- ]
                [    2          L           2          L   ]
                [   L                      L              ]
```

```
#
> # Initialize the element matrices as zeros size of global matrix
> for aa from 1 by 1 to nnode do kelemnt(aa) :=
> matrix(2*nnode,2*nnode,0) end do:
>
> # Copy the original element matrix into its respective positions in
> each element matrix for specified problem
> for bb from 1 by 1 to nelem do kelement2(bb) :=
```

```
> copyinto(eb1dke,kelemnt(bb),2*bb-1, 2*bb-1) end do:
#
> kelement2(nnode) := kelemnt(nnode):
> # Initialize the global stiffness matrix for the specified size
> master := matrix(2*nnode,2*nnode,0):

> # If loop and for loop to add the element matrices into the master
> global stiffness matrix
> if (nelem <= 2) then master := matadd(kelement2(1),kelement2(2)) else
> for cc from 1 by 1 to nelem do kelement2(cc+1) :=
> matadd(kelement2(cc),kelement2(cc+1)): master := kelement2(nnode): end
> do end if:
> # Fin.  From here it is just importing information and modifying the
> initial matrix to use, along with exporting correct information,
> Export(MapleExpression)
> evalm(master);
```

$$\begin{bmatrix} \frac{12\,EI}{L^3} & \frac{6\,EI}{L^2} & -\frac{12\,EI}{L^3} & \frac{6\,EI}{L^2} & 0 & 0 \\ \frac{6\,EI}{L^2} & \frac{4\,EI}{L} & -\frac{6\,EI}{L^2} & \frac{2\,EI}{L} & 0 & 0 \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$

```
[-12 ---- , -6 ---- , 24 ---- , 0 , -12 ---- , 6 ----]
[     3         2         3              3         2 ]
[    L         L         L              L         L  ]
[                                                    ]
[  EI         EI              EI         EI        EI ]
[6 ---- ,  2 ---- ,  0 ,  8 ---- ,  -6 ---- ,  2 ----]
[  2       L               L          2         L  ]
[ L                                   L             ]
[                                                    ]
[              EI         EI         EI        EI  ]
[0 ,  0 , -12 ---- , -6 ---- , 12 ---- , -6 ----]
[               3          2          3         2  ]
[              L          L          L         L   ]
[                                                   ]
[              EI         EI         EI        EI  ]
[0 ,  0 ,  6 ---- ,  2 ---- ,  -6 ---- ,  4 ----]
[              2          L          2         L   ]
[             L                     L              ]
```

```
#
> n := matrix([[(2*x^3 - 3*x^2*L + L^3)/L^3], [(x^3*L - 2*x^2*L^2 +
> x*L^3)/L^3], [(-2*x^3 + 3*x^2*L)/L^3], [(x^3*L - x^2*L^2)/L^3]]);
```

$$\begin{bmatrix} \frac{2x^3 - 3x^2L + L^3}{L^3} \\ \frac{x^3L - 2x^2L^2 + xL^3}{L^3} \\ \frac{-2x^3 + 3x^2L}{L^3} \\ \frac{x^3L - x^2L^2}{L^3} \end{bmatrix}$$

```
                    [                       ]
                    [ 3       2  2        3]
                    [x  L - 2 x  L  + x L ]
                    [---------------------]
                    [          3          ]
                    [         L           ]
              n := [                      ]
                    [       3      2      ]
                    [   -2 x  + 3 x  L    ]
                    [   --------------    ]
                    [          3          ]
                    [         L           ]
                    [                      ]
                    [    3      2  2       ]
                    [   x  L - x  L       ]
                    [   ------------      ]
                    [          3          ]
                    [         L           ]
```

```
> dern := map(diff,n,x):
> der2n := map(diff,dern,x):
> der3n := map(diff,der2n,x):
>
> for zzz from 0 by 1 to (nelem-1) do w(zzz+1) :=
> -4*w_o*((x+zzz*L)/(nelem*L))*(1 - ((x+zzz*L)/(nelem*L))): end do:
> for zzzz from 1 by 1 to nelem do loady(zzzz) :=
> map(int,scalarmul(n,w(zzzz)),x=0..L) end do:
> for aa from 1 by 1 to nnode do pelem(aa) := matrix(2*nnode,1,0) end
```

```
> do:
> for bb from 1 by 1 to nelem do pelem2(bb) :=
> copyinto(loady(bb),pelem(bb),2*bb-1,1) end do:
>
> pelem2(nnode) := pelem(nnode):
> P := matrix(2*nnode,1,0):
> if (nelem <= 2) then Pfinal := matadd(pelem2(1),pelem2(2)) else for cc
> from 1 by 1 to nelem do pelem2(cc+1) :=
> matadd(pelem2(cc),pelem2(cc+1)): Pfinal := pelem2(nnode): end do end
> if:
> dispmatrix := matrix(2*nnode,1,1):
> for dd from 1 by 2 to 2*nnode do dispmatrix :=
> mulrow(dispmatrix,dd,delta[(dd+1)/2]) end do:
> for ee from 2 by 2 to 2*nnode do dispmatrix :=
> mulrow(dispmatrix,ee,phi[ee/2]) end do:
> evalm(dispmatrix):
> dbc_length := 1:
> nbc_length := 1:
> rbc_length := 0:
>
> dbc_node[1] := 1:
> dbc_value[1] := 0:
> nbc_node[1] := 1:
> nbc_value[1] := 0:
>
> #  Recursive Steps!!!
> if(dbc_length >= 1) then for aa from 1 by 1 to dbc_length do master1
> := mulrow(master,dbc_node[1],0): master1 :=
```

```
> mulcol(master1,dbc_node[1],0): master1[dbc_node[1],dbc_node[1]] := 1:
> Pfinal1 := mulrow(Pfinal,dbc_node[1],0): kdbcadd :=
> submatrix(master,1..2*nnode,dbc_node[1]..dbc_node[1]): kdbcadd1 :=
> scalarmul(kdbcadd,dbc_value[1]): end do end if:
> if(nbc_length >= 1) then for bb from 1 by 1 to nbc_length do master2
> := mulrow(master1,nbc_node[1]+1,0): master2 :=
> mulcol(master2,nbc_node[1]+1,0): master2[nbc_node[1]+1,nbc_node[1]+1]
> := 1: Pfinal2 := mulrow(Pfinal1,nbc_node[1]+1,0): knbcadd :=
> submatrix(master,1..2*nnode,nbc_node[1]+1..nbc_node[1]+1): knbcadd1 :=
> scalarmul(knbcadd,nbc_value[1]): end do end if:
> # if(rbc_length >= 1) then for cc from 1 by 1 to rbc_length do
> rbcnode[cc] := rbc_value[cc] end do end if:
> if(dbc_length >= 1) then Pfinal3 := matadd(Pfinal2,kdbcadd1): end if:
> if(nbc_length >= 1) then Pfinal4 := matadd(Pfinal3,knbcadd1): end if:
> solution := multiply(inverse(master2),Pfinal4):
> print(solution);
```

$$
\begin{bmatrix} 0 \\ \\ 0 \\ \\ 4 \\ 11\ L\ \ w\_o \\ \text{-- ------} \\ 24 \quad EI \\ \\ 3 \\ 11\ L\ \ w\_o \end{bmatrix}
$$

```
                    [- -- ------ ]
                    [  15   EI   ]
                    [            ]
                    [     4      ]
                    [  56 L  w_o ]
                    [- -- ------ ]
                    [  45   EI   ]
                    [            ]
                    [     3      ]
                    [    L  w_o ]
                    [- 4/5 ------]
                    [       EI   ]
```

```
> for zz from 1 by 1 to nelem do vfe(zz) :=
> multiply(transpose(n),submatrix(solution,2*zz-1..2*zz+2,1..1)):
> vfe1(zz) := simplify(vfe(zz)[1,1]): vfe2(zz) := subs(x=xe,vfe1(zz)):
> vnode(zz) := eval(vfe2(zz),xe=0): rfe(zz) :=
> multiply(transpose(dern),submatrix(solution,2*zz-1..2*zz+2,1..1)):
> rfe1(zz) := simplify(rfe(zz)[1,1]): rfe2(zz) := subs(x=xe,rfe1(zz)):
> rnode(zz) := eval(rfe2(zz),xe=0): mfe(zz) :=
> multiply(transpose(der2n),submatrix(solution,2*zz-1..2*zz+2,1..1)):
> mfe1(zz) := simplify(mfe(zz)[1,1]): mfe2(zz) := subs(x=xe,mfe1(zz)):
> mnode(zz) := eval(mfe2(zz),xe=0): sfe(zz) :=
> multiply(transpose(der3n),submatrix(solution,2*zz-1..2*zz+2,1..1)):
> sfe1(zz) := simplify(sfe(zz)[1,1]): sfe2(zz) := subs(x=xe,sfe1(zz)):
> snode(zz) := eval(sfe2(zz),xe=0): end do:
> vnode(nelem+1) := eval(vfe2(nelem),xe=1/nelem):
> rnode(nelem+1) := eval(rfe2(nelem),xe=1/nelem):
```

```
> mnode(nelem+1) := eval(mfe2(nelem),xe=1/nelem):
> snode(nelem+1) := eval(sfe2(nelem),xe=1/nelem):
> vpreplotinfo := seq([x/nelem,vnode(x+1)],x=0..nelem):
> rpreplotinfo := seq([x/nelem,rnode(x+1)],x=0..nelem):
> mpreplotinfo := seq([x/nelem,mnode(x+1)],x=0..nelem):
> spreplotinfo := seq([x/nelem,snode(x+1)],x=0..nelem):
> L := LL/nelem:
> LL := 1:
> w_o := 1:
> EI := 1:
> dispfe := proc(x) if(x<1/nelem) then subs(xe=x,vfe2(1)) elif
> (x>=1/nelem and x<2/nelem) then subs(xe=x-1/nelem,vfe2(2)) elif
> (x>=2/nelem and x<3/nelem) then subs(xe=x-2/nelem,vfe2(3))
> elif(x>=3/nelem and x<4/nelem) then subs(xe=x-3/nelem,vfe2(4))
> elif(x>=4/nelem and x<5/nelem) then subs(xe=x-4/nelem,vfe2(5))
> elif(x>=5/nelem and x<6/nelem) then subs(xe=x-5/nelem,vfe2(6))
> elif(x>=6/nelem and x<7/nelem) then subs(xe=x-6/nelem,vfe2(7))
> elif(x>=7/nelem and x<8/nelem) then subs(xe=x-7/nelem,vfe2(8))
> elif(x>=8/nelem and x<9/nelem) then subs(xe=x-8/nelem,vfe2(9))
> elif(x>=9/nelem and x<10/nelem) then subs(xe=x-9/nelem,vfe2(10))
> elif(x>=10/nelem and x<11/nelem) then subs(xe=x-10/nelem,vfe2(11))
> elif(x>=11/nelem and x<12/nelem) then subs(xe=x-11/nelem,vfe2(12))
> elif(x>=12/nelem and x<13/nelem) then subs(xe=x-12/nelem,vfe2(13))
> elif(x>=13/nelem and x<14/nelem) then subs(xe=x-13/nelem,vfe2(14))
> elif(x>=14/nelem and x<15/nelem) then subs(xe=x-14/nelem,vfe2(15))
> else subs(xe=x-15/nelem,vfe2(16)) end if end:
>
> rotfe := proc(x) if(x<1/nelem) then subs(xe=x,rfe2(1)) elif
```

```
> (x>=1/nelem and x<2/nelem) then subs(xe=x-1/nelem,rfe2(2)) elif
> (x>=2/nelem and x<3/nelem) then subs(xe=x-2/nelem,rfe2(3))
> elif(x>=3/nelem and x<4/nelem) then subs(xe=x-3/nelem,rfe2(4))
> elif(x>=4/nelem and x<5/nelem) then subs(xe=x-4/nelem,rfe2(5))
> elif(x>=5/nelem and x<6/nelem) then subs(xe=x-5/nelem,rfe2(6))
> elif(x>=6/nelem and x<7/nelem) then subs(xe=x-6/nelem,rfe2(7))
> elif(x>=7/nelem and x<8/nelem) then subs(xe=x-7/nelem,rfe2(8))
> elif(x>=8/nelem and x<9/nelem) then subs(xe=x-8/nelem,rfe2(9))
> elif(x>=9/nelem and x<10/nelem) then subs(xe=x-9/nelem,rfe2(10))
> elif(x>=10/nelem and x<11/nelem) then subs(xe=x-10/nelem,rfe2(11))
> elif(x>=11/nelem and x<12/nelem) then subs(xe=x-11/nelem,rfe2(12))
> elif(x>=12/nelem and x<13/nelem) then subs(xe=x-12/nelem,rfe2(13))
> elif(x>=13/nelem and x<14/nelem) then subs(xe=x-13/nelem,rfe2(14))
> elif(x>=14/nelem and x<15/nelem) then subs(xe=x-14/nelem,rfe2(15))
> else subs(xe=x-15/nelem,rfe2(16)) end if end:
>
> momfe := proc(x) if(x<1/nelem) then subs(xe=x,mfe2(1)) elif
> (x>=1/nelem and x<2/nelem) then subs(xe=x-1/nelem,mfe2(2)) elif
> (x>=2/nelem and x<3/nelem) then subs(xe=x-2/nelem,mfe2(3))
> elif(x>=3/nelem and x<4/nelem) then subs(xe=x-3/nelem,mfe2(4))
> elif(x>=4/nelem and x<5/nelem) then subs(xe=x-4/nelem,mfe2(5))
> elif(x>=5/nelem and x<6/nelem) then subs(xe=x-5/nelem,mfe2(6))
> elif(x>=6/nelem and x<7/nelem) then subs(xe=x-6/nelem,mfe2(7))
> elif(x>=7/nelem and x<8/nelem) then subs(xe=x-7/nelem,mfe2(8))
> elif(x>=8/nelem and x<9/nelem) then subs(xe=x-8/nelem,mfe2(9))
> elif(x>=9/nelem and x<10/nelem) then subs(xe=x-9/nelem,mfe2(10))
> elif(x>=10/nelem and x<11/nelem) then subs(xe=x-10/nelem,mfe2(11))
> elif(x>=11/nelem and x<12/nelem) then subs(xe=x-11/nelem,mfe2(12))
```

```
> elif(x>=12/nelem and x<13/nelem) then subs(xe=x-12/nelem,mfe2(13))
> elif(x>=13/nelem and x<14/nelem) then subs(xe=x-13/nelem,mfe2(14))
> elif(x>=14/nelem and x<15/nelem) then subs(xe=x-14/nelem,mfe2(15))
> else subs(xe=x-15/nelem,mfe2(16)) end if end:
>
> shrfe := proc(x) if(x<1/nelem) then subs(xe=x,sfe2(1)) elif
> (x>=1/nelem and x<2/nelem) then subs(xe=x-1/nelem,sfe2(2)) elif
> (x>=2/nelem and x<3/nelem) then subs(xe=x-2/nelem,sfe2(3))
> elif(x>=3/nelem and x<4/nelem) then subs(xe=x-3/nelem,sfe2(4))
> elif(x>=4/nelem and x<5/nelem) then subs(xe=x-4/nelem,sfe2(5))
> elif(x>=5/nelem and x<6/nelem) then subs(xe=x-5/nelem,sfe2(6))
> elif(x>=6/nelem and x<7/nelem) then subs(xe=x-6/nelem,sfe2(7))
> elif(x>=7/nelem and x<8/nelem) then subs(xe=x-7/nelem,sfe2(8))
> elif(x>=8/nelem and x<9/nelem) then subs(xe=x-8/nelem,sfe2(9))
> elif(x>=9/nelem and x<10/nelem) then subs(xe=x-9/nelem,sfe2(10))
> elif(x>=10/nelem and x<11/nelem) then subs(xe=x-10/nelem,sfe2(11))
> elif(x>=11/nelem and x<12/nelem) then subs(xe=x-11/nelem,sfe2(12))
> elif(x>=12/nelem and x<13/nelem) then subs(xe=x-12/nelem,sfe2(13))
> elif(x>=13/nelem and x<14/nelem) then subs(xe=x-13/nelem,sfe2(14))
> elif(x>=14/nelem and x<15/nelem) then subs(xe=x-14/nelem,sfe2(15))
> else subs(xe=x-15/nelem,sfe2(16)) end if end:
> dispex := proc(x) 4*(-(1/24)*x^2 + (1/36)*x^3 -(1/120)*x^5 +
> (1/360)*x^6) end:
> rotex := proc(x) 4*(-(2/24)*x + (3/36)*x^2 - (5/120)*x^4 +
> (6/360)*x^5) end:
> momex := proc(x) 4*(-(2/24) + (6/36)*x - (20/120)*x^3 + (30/360)*x^4)
> end:
> shrex := proc(x) 4*((6/36) - (60/120)*x^2 + (120/360)*x^3) end:
```

```
> p2 := plots[pointplot]({vpreplotinfo},symbol=circle,symbolsize=20);

  p2 := PLOT(POINTS([0., 0.], [.5000000000, -.02864583333],

        [1., -.07777777778]), SYMBOL(CIRCLE, 20))

> p1 :=
> plot([dispfe,dispex],0..1,color=[blue,black],thickness=[1,1],linestyle
> =[3,1],axes=framed):
> plots[display]({p1,p2});


> p3 := plots[pointplot]({rpreplotinfo},symbol=circle,symbolsize=20);

  p3 := PLOT(POINTS([0., 0.], [.5000000000, -.09166666667],

        [1., -.1000000000]), SYMBOL(CIRCLE, 20))

> p4 :=
> plot([rotfe,rotex],0..1,color=[blue,black],thickness=[1,1],linestyle=[
> 3,1],axes=framed):
> plots[display]({p3,p4});

> p5 := plots[pointplot]({mpreplotinfo},symbol=circle,symbolsize=20);

  p5 := PLOT(POINTS([0., -.3208333333], [.5000000000, -.04583333333],

        [1., .01250000000]), SYMBOL(CIRCLE, 20))
```

```
> p6 :=
> plot([momfe,momex],0..1,color=[blue,black],thickness=[1,1],linestyle=[
> 3,1],axes=framed):
> plots[display]({p5,p6});

> p7 := plots[pointplot]({spreplotinfo},symbol=circle,symbolsize=20);

  p7 := PLOT(POINTS([.5000000000, .1166666667], [1., .1166666667],

        [0., .5500000000]), SYMBOL(CIRCLE, 20))

> p8 :=
> plot([shrfe,shrex],0..1,color=[blue,black],thickness=[1,1],linestyle=[
> 3,1],axes=framed):
> plots[display]({p7,p8});
```

B.2 Two Dimensional Code

```
> ### 2D Square Example Solution for a rectangle
> ## Element Formulation
> restart:
> with(linalg):
Warning, the protected names norm and trace have been redefined and
unprotected

> with(plots):
Warning, the name changecoords has been redefined
```

```
> printlevel := 2:
>
>
> # Stiffness Matrix Setup
>
> n :=
> matrix([[((h-x)*(h-y))/(4*h^2)],[((h+x)*(h-y))/(4*h^2)],[((h+x)*(h+y))
> /(4*h^2)],[((h-x)*(h+y))/(4*h^2)]]):
> N :=
> matrix([[n[1,1],0,n[2,1],0,n[3,1],0,n[4,1],0],[0,n[1,1],0,n[2,1],0,n[3
> ,1],0,n[4,1]]]):
> NT12 := transpose(N):
> NT23 := transpose(N):
> NT34 := transpose(N):
> NT41 := transpose(N):
> NT := transpose(N):
> B :=
> matrix([[-(h-y)/(4*h^2),0,(h-y)/(4*h^2),0,(h+y)/(4*h^2),0,-(h+y)/(4*h^
> 2),0],[0,-(h-x)/(4*h^2),0,-(h+x)/(4*h^2),0,(h+x)/(4*h^2),0,(h-x)/(4*h^
> 2)],[-(h-x)/(4*h^2),-(h-y)/(4*h^2),-(h+x)/(4*h^2),(h-y)/(4*h^2),(h+x)/
> (4*h^2),(h+y)/(4*h^2),(h-x)/(4*h^2),-(h+y)/(4*h^2)]]):
> DD :=
> scalarmul(matrix([[1,nu,0],[nu,1,0],[0,0,(1-nu)/2]]),E/(1-nu^2)):
> k1 := map(int,multiply(transpose(B),DD,B),x=-h..h):
> ke := simplify(scalarmul(map(int,k1,y=-h..h),t)):
>
> # Under Integration Setup
```

```
> ### For under integration of the given problem, we need a 1 point
> gauss method, i.e. W_{1}*W_{2}*z(x_{o},y_{o}) will underintegrate the
> function.  For this problem, the function is evaluated at x=y=0 since
> there is only a 1-pt. gaussian quadrature in effect.  Include the
> numerical under-integration method used in the thesis.  This method
> yields only the constant terms which get integrated since the linear
> terms go to zero in the integration from -h..h conveniently.  To apply
> this in maple, i will break up the integration of the stiffness matrix
> and remove the unwanted terms before i integrate, so i only integrate
> what would be found using a 1pt gauss quadrature rule.
>
> kunder1 := map(expand,multiply(transpose(B),DD,B)):
> kunder2 := map2(remove, has, kunder1, x):
> kunder3 := map2(remove, has, kunder2, y):
> kunder4 := map(int, kunder3, x=-h..h):
> kundere := simplify(scalarmul(map(int, kunder4, y=-h..h),t)):
>
> nelem := 1:
> nnode := 2*(nelem + 1)^2:
> for aa from 1 by 1 to nnode do kelement(aa) := matrix(nnode,nnode,0):
> kelementu(aa) := matrix(nnode,nnode,0): end do:
> for bb from 1 by 1 to nelem do kelement2(bb) :=
> copyinto(ke,kelement(bb),2*bb-1, 2*bb-1): kelement2u(bb) :=
> copyinto(kundere,kelementu(bb),2*bb-1,2*bb-1): end do:
> kelement2(nnode) := kelement(nnode): kelement2u(nnode) :=
> kelementu(nnode):
> k := matrix(nnode,nnode,0): kunder := matrix(nnode,nnode,0):
> if nelem <= 2 then k := matadd(kelement2(1),kelement2(nnode)): kunder
```

```
> := matadd(kelement2u(1),kelement2u(nnode)): else for cc from 1 by 1 to
> nelem do kelement2(cc+1) := matadd(kelement2(cc),kelement2(cc+1)): k
> := kelement2(nnode): kelement2u(cc+1) :=
> matadd(kelement2(cc),kelement2(cc+1)): kunder := kelement2u(nnode):
> end do end if:
> evalm(k);
```

$$
\begin{bmatrix} & E\,t & & E\,t & \\ \%1 & -\frac{1/8}{-1+nu} & \%5 & \%3 & \%4 & \frac{1/8}{-1+nu} & \%2 & \%6 \end{bmatrix}
$$

$$
\begin{bmatrix} & E\,t & & E\,t & \\ -\frac{1/8}{-1+nu} & \%1 & \%6 & \%2 & \frac{1/8}{-1+nu} & \%4 & \%3 & \%5 \end{bmatrix}
$$

$$
\begin{bmatrix} & E\,t & & E\,t \\ \%5 & \%6 & \%1 & \frac{1/8}{-1+nu} & \%2 & \%3 & \%4 & -\frac{1/8}{-1+nu} \end{bmatrix}
$$

$$
\begin{bmatrix} & E\,t & & E\,t & \\ \%3 & \%2 & \frac{1/8}{-1+nu} & \%1 & \%6 & \%5 & -\frac{1/8}{-1+nu} & \%4 \end{bmatrix}
$$

$$
\begin{bmatrix} & E\,t & & E\,t & \\ \%4 & \frac{1/8}{-1+nu} & \%2 & \%6 & \%1 & -\frac{1/8}{-1+nu} & \%5 & \%3 \end{bmatrix}
$$

$$
\begin{bmatrix} E\,t & & E\,t & \end{bmatrix}
$$

```
[1/8 ------- , %4 , %3 , %5 , - 1/8 ------- , %1 , %6 , %2]
[    -1 + nu                          -1 + nu                ]

[                        E t                            E t  ]
[%2 , %3 , %4 , - 1/8 ------- , %5 , %6 , %1 , 1/8 -------]
[                      -1 + nu                        -1 + nu]

[                    E t                              E t      ]
[%6 , %5 , - 1/8 ------- , %4 , %3 , %2 , 1/8 ------- , %1]
[                 -1 + nu                          -1 + nu    ]
```

$$\%1 := \frac{1}{6}\,\frac{t\,E\,(-3+nu)}{(-1+nu)^2}$$

$$\%2 := -\frac{1}{6}\,\frac{t\,E\,nu}{(-1+nu)^2}$$

$$\%3 := -\frac{1}{8}\,\frac{t\,E\,(3\,nu-1)}{(-1+nu)^2}$$

$$\%4 := -\frac{1}{12}\,\frac{t\,E\,(-3+nu)}{}$$

```
                  2
          -1 + nu


        t E (3 + nu)
  %5 := 1/12 ------------
                     2
             -1 + nu


        t E (3 nu - 1)
  %6 := 1/8 --------------
                     2
             -1 + nu


> evalm(kunder);

      [              E t           E t                     E t
      [%1 , - 1/8 ------- , 1/8 ------- , %2 , %3 , 1/8 ------- ,
      [          -1 + nu       -1 + nu                 -1 + nu

             E t       ]
      - 1/8 ------- , %4]
           -1 + nu     ]

      [       E t                      E t          E t
      [- 1/8 ------- , %1 , %4 , - 1/8 ------- , 1/8 ------- , %3 ,
      [     -1 + nu                  -1 + nu      -1 + nu

              E t  ]
```

```
%2 , 1/8 -------]
      -1 + nu]

[      E t                    E t          E t
[1/8 ------- , %4 , %1 , 1/8 ------- , - 1/8 ------- , %2 ,
[    -1 + nu                -1 + nu      -1 + nu

          E t  ]
%3 , - 1/8 -------]
       -1 + nu]

[             E t          E t                    E t
[%2 , - 1/8 ------- , 1/8 ------- , %1 , %4 , 1/8 ------- ,
[           -1 + nu       -1 + nu                -1 + nu

        E t       ]
- 1/8 ------- , %3]
      -1 + nu     ]

[           E t            E t                      E t
[%3 , 1/8 ------- , - 1/8 ------- , %4 , %1 , - 1/8 ------- ,
[         -1 + nu         -1 + nu                  -1 + nu

      E t       ]
1/8 ------- , %2]
    -1 + nu     ]

[      E t                    E t          E t
```

```
[1/8 ------- , %3 , %2 , 1/8 ------- , - 1/8 ------- , %1 ,
[    -1 + nu                 -1 + nu        -1 + nu

                E t  ]
%4 , - 1/8 -------]
              -1 + nu]
```

```
[        E t                      E t          E t
[- 1/8 ------- , %2 , %3 , - 1/8 ------- , 1/8 ------- , %4 ,
[      -1 + nu                    -1 + nu      -1 + nu

             E t  ]
%1 , 1/8 -------]
           -1 + nu]
```

```
[           E t            E t                    E t
[%4 , 1/8 ------- , - 1/8 ------- , %3 , %2 , - 1/8 ------- ,
[         -1 + nu        -1 + nu                  -1 + nu

       E t       ]
1/8 ------- , %1]
    -1 + nu      ]
```

```
          t E (-3 + nu)
%1 := 1/8 -------------
                 2
          -1 + nu
```

```
                t E (3 nu - 1)
  %2 := - 1/8 --------------
                        2
                  -1 + nu

                t E (-3 + nu)
  %3 := - 1/8 -------------
                      2
                -1 + nu

              t E (3 nu - 1)
  %4 := 1/8 --------------
                      2
                -1 + nu
```

```
> kdiff := simplify(k - kunder):
> evalm(kdiff);
```

$$\begin{bmatrix} \%1 & 0 & \%3 & 0 & \%1 & 0 & \%2 & 0 \\ 0 & \%1 & 0 & \%2 & 0 & \%1 & 0 & \%3 \\ \%3 & 0 & \%1 & 0 & \%2 & 0 & \%1 & 0 \\ 0 & \%2 & 0 & \%1 & 0 & \%3 & 0 & \%1 \\ \%1 & 0 & \%2 & 0 & \%1 & 0 & \%3 & 0 \end{bmatrix}$$

```
[0     %1    0     %3    0     %1    0     %2]
[                                              ]
[%2    0     %1    0     %3    0     %1    0 ]
[                                              ]
[0     %3    0     %1    0     %2    0     %1]
```

$$\%1 := \frac{1}{24}\,\frac{t\,E\,(-3+\nu)}{(-1+\nu)^2}$$

$$\%2 := -\frac{1}{6}\,\frac{t\,E\,\nu}{(-1+\nu)^2} + \frac{1/8\,E\,t}{-1+\nu}$$

$$\%3 := \frac{1}{12}\,\frac{t\,E\,(3+\nu)}{(-1+\nu)^2} - \frac{1}{8}\,\frac{E\,t}{-1+\nu}$$

```
> kunder2 := kunder + scalarmul(kdiff,1/2):
> # Force setup has to be fixed due to placement and nodal stuff i.e.
> one force on side applies to two nodes - which two is the key !!!! -
> How to Loop Force Setup !!!
>
> # Given surface traction w_o/P_o matrices:
> # The surface traction(s) on surface is a maximum of 16 total, +/-4
```

```
> shear, +/-4 normal
> ## Elemental Force Matrix Formulation ##
> ### 1-2 Plane ###
> T12 := matrix([[0],[w_o]]): # Normal traction to the x=x, y=-h surface
> S12 := matrix([[P_o],[0]]): # Shear traction to the x=x, y=-h surface
> NT_F12 := NT12:
> TSendnode12 := 2:
> for kk from (TSendnode12*2)+1 to nnode do NT_F12[kk,1] := 0:
> NT_F12[kk,2] := 0: end do:
> NT_F12 := map(rcurry(eval, 'y' = -h),NT_F12):
> T12e := scalarmul(map(int,multiply(NT_F12,T12),x=-h..h),t):
> S12e := scalarmul(map(int,multiply(NT_F12,S12),x=-h..h),t):
> ### 2-3 Plane ###
> T23 := matrix([[w_o],[0]]): # Normal traction to the x=h, y=y surface
> S23 := matrix([[0],[P_o]]): # Shear traction to the x=h, y=y surface
> NT_F23 := NT23:
> NT_F23[1,1] := 0: NT_F23[2,2] := 0: NT_F23[7,1] := 0: NT_F23[8,2] :=
> 0:
> NT_F23 := map(rcurry(eval, 'x' = h),NT_F23):
> T23e := scalarmul(map(int,multiply(NT_F23,T23),y=-h..h),t);

                              [   0   ]
                              [       ]
                              [   0   ]
                              [       ]
                              [t w_o h]
                              [       ]
                              [   0   ]
```

```
T23e := [      ]
        [t w_o h]
        [      ]
        [  0   ]
        [      ]
        [  0   ]
        [      ]
        [  0   ]
```

```
> S23e := scalarmul(map(int,multiply(NT_F23,S23),y=-h..h),t):
> ### 3-4 Plane ###
> T34 := matrix([[0],[w_o]]): # Normal traction to the x=x, y=h surface
> S34 := matrix([[P_o],[0]]): # Normal traction to the x=x, y=h surface
> NT_F34 := NT34:
> NT_F34[1,1] := 0: NT_F34[2,2] := 0: NT_F34[3,1] := 0: NT_F34[4,2] :=
> 0:
> NT_F34 := map(rcurry(eval, 'y' = h),NT_F34):
> T34e := scalarmul(map(int,multiply(NT_F34,T34),x=-h..h),t):
> S34e := scalarmul(map(int,multiply(NT_F34,S34),x=-h..h),t):
> ### 4-1 Plane ###
> T41 := matrix([[w_o],[0]]): # Normal traction to the x=-h, y=y surface
> S41 := matrix([[0],[P_o]]): # Normal traction to the x=-h, y=y surface
> NT_F41 := NT41:
> NT_F41[5,1] := 0: NT_F41[6,2] := 0: NT_F41[3,1] := 0: NT_F41[4,2] :=
> 0:
> NT_F41 := map(rcurry(eval, 'x' = -h),NT_F41):
> T41e := scalarmul(map(int,multiply(NT_F41,T41),y=-h..h),t):
> S41e := scalarmul(map(int,multiply(NT_F41,S41),y=-h..h),t):
```

```
> ## Global Force Matrix Formulation ##
> for ff from 1 by 1 to nnode do pelem(ff) := matrix(nnode,1,0) end do:
> for gg from 1 by 1 to nelem do pelem2(gg) :=
> copyinto(T23e,pelem(gg),2*gg-1,1) end do;
```

$$
\text{pelem2}(1) := \begin{bmatrix} 0 \\ \\ 0 \\ \\ t\ w\_o\ h \\ \\ 0 \\ \\ t\ w\_o\ h \\ \\ 0 \\ \\ 0 \\ \\ 0 \end{bmatrix}
$$

```
> pelem2(nnode) := pelem(nnode):
> Pfinal := matrix(nnode,2,0):
> if (nelem <= 2) then P := matadd(pelem2(1),pelem2(nnode)) else for hh
> from 1 by 1 to nelem do pelem2(hh+1) :=
> matadd(pelem2(hh),pelem2(hh+1)): P := pelem2(nnode): end do end if:
> evalm(P);
```

```
[   0   ]
[       ]
[   0   ]
[       ]
[t w_o h]
[       ]
[   0   ]
[       ]
[t w_o h]
[       ]
[   0   ]
[       ]
[   0   ]
[       ]
[   0   ]
```

```
> # Boundary Condition Setup
> ubc_length := 1: # # of Fixed x-direction bc's
> vbc_length := 2: # # of Fixed y-direction bc's
> ubc_node[1] := 1: # node in which bc applies
> ubc_value[1] := 0: # value in which bc imposes
> vbc_node[1] := 2: # node in which bc applies
> vbc_value[1] := 0: # value in which bc imposes
> ubc_node[2] := 3:
> ubc_value[2] := 0:
> vbc_node[2] := 4:
> vbc_value[2] := 0:
> k1 := k:
```

```
> kun1 := kunder2:
> P1 := P:
> if(ubc_length >= 1) then for ii from 1 by 1 to ubc_length do k1 :=
> mulrow(k1,ubc_node[ii],0): k1 := mulcol(k1,ubc_node[ii],0):
> k1[ubc_node[ii],ubc_node[ii]] := 1: P1 := mulrow(P1,ubc_node[ii],0):
> kun1 := mulrow(kun1,ubc_node[ii],0): kun1 :=
> mulcol(kun1,ubc_node[ii],0): kun1[ubc_node[ii],ubc_node[ii]] := 1: end
> do end if:
> k2 := k1:
> kun2 := kun1:
> P2 := P1:
> if(vbc_length >= 1) then for jj from 1 by 1 to vbc_length do k2 :=
> mulrow(k2,vbc_node[jj],0): k2 := mulcol(k2,vbc_node[jj],0):
> k2[vbc_node[jj],vbc_node[jj]] := 1: P2 := mulrow(P2,vbc_node[jj],0):
> kun2 := mulrow(kun2,vbc_node[jj],0): kun2 :=
> mulcol(kun2,vbc_node[jj],0): kun2[vbc_node[jj],vbc_node[jj]] := 1: end
> do end if:
> evalm(k2);

       [1 , 0 , 0 , 0 , 0 , 0 , 0 , 0]

       [0 , 1 , 0 , 0 , 0 , 0 , 0 , 0]

       [                                    t E (-3 + nu)
       [0 , 0 , %1 , 0 , %2 , %3 , - 1/12 ------------- ,
       [                                         2
       [                                   -1 + nu
```

```
       E t  ]
- 1/8 -------]
     -1 + nu]
             ]

[0 , 0 , 0 , 1 , 0 , 0 , 0 , 0]

[                              E t          t E (3 + nu)
[0 , 0 , %2 , 0 , %1 , - 1/8 ------- , 1/12 ------------ , %3
[                            -1 + nu              2
[                                          -1 + nu

]
]
]
]

[                          E t              t E (3 nu - 1)
[0 , 0 , %3 , 0 , - 1/8 ------- , %1 , 1/8 -------------- ,
[                        -1 + nu                  2
[                                          -1 + nu

  ]
%2]
  ]
  ]

[              t E (-3 + nu)            t E (3 + nu)
```

```
[0 , 0 , - 1/12 ------------ , 0 , 1/12 ----------- ,
[                        2                       2
[                 -1 + nu                 -1 + nu

    t E (3 nu - 1)              E t  ]
1/8 -------------- , %1 , 1/8 -------]
          2                   -1 + nu]
     -1 + nu                         ]

[                E t                          E t       ]
[0 , 0 , - 1/8 ------- , 0 , %3 , %2 , 1/8 ------- , %1]
[             -1 + nu                      -1 + nu     ]

     t E (-3 + nu)
%1 := 1/6 ------------
           2
      -1 + nu

      t E nu
%2 := - 1/6 --------
           2
      -1 + nu

      t E (3 nu - 1)
%3 := - 1/8 --------------
            2
       -1 + nu
```

```
> evalm(kun2);

        [1 , 0 , 0 , 0 , 0 , 0 , 0 , 0]

        [0 , 1 , 0 , 0 , 0 , 0 , 0 , 0]

        [                                    t E (-3 + nu)
        [0 , 0 , %1 , 0 , %2 , %3 , - 5/48 ------------- ,
        [                                         2
        [                                   -1 + nu

               E t  ]
        - 1/8 -------]
              -1 + nu]
                    ]

        [0 , 0 , 0 , 1 , 0 , 0 , 0 , 0]

        [                                E t
        [0 , 0 , %2 , 0 , %1 , - 1/8 ------- ,
        [                              -1 + nu
        [

               E t     1/24 t E (3 + nu)     ]
        1/16 ------- + ----------------- , %3]
             -1 + nu              2          ]
                           -1 + nu           ]
```

```
[                          E t              t E (3 nu - 1)
[0 , 0 , %3 , 0 , - 1/8 ------- , %1 , 1/8 -------------- ,
[                       -1 + nu                      2
[                                             -1 + nu

  ]
%2]
  ]
  ]
```

```
[               t E (-3 + nu)
[0 , 0 , - 5/48 ------------- , 0 ,
[                        2
[                 -1 + nu

      E t     1/24 t E (3 + nu)       t E (3 nu - 1)
1/16 ------- + ----------------- , 1/8 -------------- , %1 ,
    -1 + nu              2                    2
                      -1 + nu              -1 + nu

     E t  ]
1/8 -------]
    -1 + nu]
           ]
```

```
[                E t                          E t       ]
[0 , 0 , - 1/8 ------- , 0 , %3 , %2 , 1/8 ------- , %1]
[              -1 + nu                        -1 + nu     ]
```

```
                    t E (-3 + nu)
%1 := 7/48 -------------
                           2
                     -1 + nu

                       E t              t E nu
%2 := - 1/16 ------- - 1/12 --------
                     -1 + nu                 2
                                      -1 + nu

                    t E (3 nu - 1)
%3 := - 1/8 --------------
                           2
                     -1 + nu
```

```
> h := L/nelem:
> solution := multiply(inverse(k2),T34e): # Just Substitute load
> condition here
> solunder := multiply(inverse(kun2),P2): # Just Substitute load
> condition here
> simplify(solution);
```

$$\begin{bmatrix} 0 \\ \\ 0 \\ \\ nu\ w\_o\ L \end{bmatrix}$$

```
                    [-2 --------]
                    [      E    ]
                    [           ]
                    [     0     ]
                    [           ]
                    [   nu w_o L]
                    [-2 --------]
                    [      E    ]
                    [           ]
                    [    w_o L  ]
                    [  2 -----  ]
                    [      E    ]
                    [           ]
                    [     0     ]
                    [           ]
                    [    w_o L  ]
                    [  2 -----  ]
                    [      E    ]
```

```
> simplify(solunder);
```

```
                  [            0            ]
                  [                         ]
                  [            0            ]
                  [                         ]
                  [         2               ]
                  [    (6 nu  + nu - 9) w_o L ]
                  [  2 ---------------------- ]
```

```
[         (-3 + nu) E          ]
[                              ]
[              0               ]
[                              ]
[                          2   ]
[   w_o L (-12 - nu + 7 nu )   ]
[ 2 ------------------------   ]
[         (-3 + nu) E          ]
[                              ]
[        2                     ]
[   (7 nu  - 3 nu - 6) w_o L   ]
[-2 ------------------------   ]
[         (-3 + nu) E          ]
[                              ]
[              2               ]
[  w_o L (13 nu  - 2 nu - 15)  ]
[2 --------------------------  ]
[         (-3 + nu) E          ]
[                              ]
[       2                      ]
[   (5 nu  + 3 nu - 6) w_o L   ]
[ 2 ------------------------   ]
[         (-3 + nu) E          ]
```

```
>
> # Post Processing Setup
> sigma[e] := simplify(multiply(DD,B,solution));
```

```
                                  [ 0 ]
                                  [   ]
                        sigma[e] := [w_o]
                                  [   ]
                                  [ 0 ]
```

```
> sigmau[e] := simplify(multiply(DD,B,solunder));
```

```
                [          (6 x nu + L nu + 6 y - 3 L) w_o          ]
                [          ------------------------------           ]
                [                   L (-3 + nu)                     ]
                [                                                   ]
                [                   (y nu + x) w_o                  ]
 sigmau[e] := [                 6 --------------                    ]
                [                    L (-3 + nu)                    ]
                [                                                   ]
                [      (6 y nu + 6 x nu - L nu - 6 x - 6 y + 3 L) w_o]
                [- 1/2 -------------------------------------------- ]
                [                     L (-3 + nu)                   ]
```

```
> sigmau2 := sigmau[e]:
> sigma2 := sigma[e]:
> # sigma2 := map(rcurry(eval, 'L' = 1),sigma2): sigma2 :=
> map(rcurry(eval, 't' = 1),sigma2): sigma2 := map(rcurry(eval, 'E' =
> 1),sigma2):
> # sigma2 := map(rcurry(eval, 'x' = 0),sigma2): sigma2 :=
> map(rcurry(eval, 'L' = 1),sigma2): sigma2 := map(rcurry(eval, 'nu' =
> 1),sigma2): sigma2 := map(rcurry(eval, 't' = 1),sigma2): sigma2 :=
```

```
> map(rcurry(eval, 'y' = 0),sigma2): sigma2 := map(rcurry(eval, 'w_o' =
> 1),sigma2): sigma2 := map(rcurry(eval, 'E' = 1),sigma2):
> # sigmau2 := map(rcurry(eval, 'x' = 0),sigmau2): sigmau2 :=
> map(rcurry(eval, 'L' = 1),sigmau2): sigmau2 := map(rcurry(eval, 'nu' =
> 1/2),sigmau2): sigmau2 := map(rcurry(eval, 't' = 1),sigmau2): sigmau2
> := map(rcurry(eval, 'y' = 0),sigmau2): sigmau2 := map(rcurry(eval,
> 'w_o' = 1),sigmau2): sigmau2 := map(rcurry(eval, 'E' = 1),sigmau2):
> evalm(sigma2);
```

$$\begin{bmatrix} 0 \\ \\ w_o \\ \\ 0 \end{bmatrix}$$

```
> evalm(sigmau2);
```

$$\begin{bmatrix} \dfrac{(6\,x\,nu + L\,nu + 6\,y - 3\,L)\,w_o}{L\,(-3 + nu)} \\ \\ 6\,\dfrac{(y\,nu + x)\,w_o}{L\,(-3 + nu)} \\ \\ -\dfrac{1}{2}\,\dfrac{(6\,y\,nu + 6\,x\,nu - L\,nu - 6\,x - 6\,y + 3\,L)\,w_o}{L\,(-3 + nu)} \end{bmatrix}$$

Vita

I was born in CT 24 years ago on August 15^{th}, 1978. That is 8843 days from my birthdate to today, October 31^{st}, 2002. Had an average childhood, other than being oblivious to everything. Went to private parochial grade school and high school. By the end of that time, I had acquired a dislike for CT and New England, so I went to college at the University of TN, Knoxville.

Along the way, I hung out with people, met Jess, my future wife to be in a few months on December 13^{th}, 2002. Things got better after meeting her, not that things were all that bad before. So school went by quickly hanging out with her when not in classes or doing work.

I graduated at the age of 21 with a B.S. in Physics, and spent one year working towards a Ph.D. in physics when I decided to switch to Engineering so I would be able to get a job and support my future family. After college, I began work on my thesis, boy did that take long time. I have now been working on finishing my master's for two years, or five semesters, with many problems met along the way. Spending my time with classes, teaching, doing my thesis project, hoping my advisor makes a good recovery, trying to clear up my credit and military status since January when I found out my brother stole my identity and ruined my credit and enlisted in the military under my identity.

But back to happier things. At the time of writing this, I am a few days from completing my Master's Thesis and getting my M.S. in Engineering Science with a Specialization in Solid Mechanics. From here, my job and the rest of my married life begins. :-)

I would like to thank **Jess**, Dr. Pionke, Dr. Yu, Dr. Sawhney, Will, Ralphie, Maple Support, W3C support, Open Source Software, parents, friends, work, and if some of this seems crazy, its due to listening to the *BareNaked Ladies* while writing this paper.

None of this would be Possible $^{w}/_{o}$ Jess' Love, Patience, and Tolerance