

Interactive Analysis of General Beam Configurations using Finite Element Methods and
JavaScript

By

CHRISTOPHER HERNANDEZ

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN AEROSPACE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2016

Copyright © by Christopher Hernandez 2016

All Rights Reserved



Acknowledgements

I would like to thank my family for always being there for me and supporting me through all the years I've been in school. Without them I would not have been able to achieve all that I have up to this point. I would also like to express my gratitude to Dr. Lawrence, who gave me the opportunity to not only learn new skills but was also there to guide and help me make my work the best that it could be, and was instrumental in shaping this work into what it is today. Finally, I would like to thank my friends who I have been with me since my undergraduate studies.

November 21, 2016

Abstract

DEVELOPMENT OF FINITE ELEMENT BEAM ANALYSIS APPLICATION USING JAVASCRIPT

Christopher Hernandez, MS

The University of Texas at Arlington, 2016

Supervising Professor: Kent Lawrence

Advancements in computer technology have contributed to the widespread practice of modelling and solving engineering problems through the use of specialized software. The wide use of engineering software comes with the disadvantage to the user of costs from the required purchase of software licenses. The creation of accurate, trusted, and freely available applications capable of conducting meaningful analysis of engineering problems is a way to mitigate to the costs associated with every-day engineering computations. Writing applications in the JavaScript programming language allows the applications to run within any computer browser, without the need to install specialized software, since all internet browsers are equipped with virtual machines (VM) that allow the browsers to execute JavaScript code.

The objective of this work is the development of an application that performs the analysis of a completely general beam through use of the finite element method. The app is written in JavaScript and embedded in a web page so it can be downloaded and executed by a user with an internet connection. This application allows the user to analyze any uniform or non-uniform beam, with any combination of applied forces, moments, distributed loads, and boundary conditions. Outputs for this application include lists the beam deformations and slopes, as well as lateral and slope deformation graphs, bending stress distributions, and shear and a moment diagrams. To validate the methodology of the GBeam finite element app, its results are verified using the results from obtained from two other established finite element solvers for fifteen separate test cases.

Table of Contents

Acknowledgements.....	iii
Abstract.....	iv
List of Illustrations.....	viii
List of Tables.....	x
1. Introduction.....	1
2. Installed Software Licensing.....	1
2.1 SOFTWARE LICENSING BEFORE THE INTERNET.....	1
2.2 Types of Licenses.....	2
3. JavaScript Programming Language.....	5
3.1 JavaScript: A Brief Overview.....	5
3.2 Developing Computational Applications with JavaScript.....	6
4. The Finite Element Method.....	10
4.1 Overview of FEM: History, Advantages, and Concepts.....	10
4.2 Beam Models: Euler-Bernoulli Beam.....	14
4.3 Formulation of Beam Element Stiffness Matrix: Euler-Bernoulli.....	16
4.4 Formulation of Beam Global Stiffness Matrix: Euler-Bernoulli.....	24
5. JavaScript Finite Element Beam Analysis.....	32
5.1 An Application in Three Parts.....	32
5.1.1 Input Spreadsheet: The Main Menu.....	33
5.1.2 Solution Output Window.....	41
5.1.3 Plot Output Window.....	42
5.2 Sorting Data Into Arrays.....	43
5.3 Creation of Element & Global Stiffness Matrices.....	47
5.3.1 Stiffness Matrix Creation.....	47
5.3.2 Effects of Boundary Conditions On Global Stiffness Matrix.....	50
5.4 Spring Support.....	63
5.5 Applied Forces and Moments.....	64
5.5.1 Application of Distributed Forces.....	65
5.6 Beam Refinement.....	70
6. Solution and Outputs.....	71
6.1 Displacements.....	72

6.2 Displacement Output Window	75
6.3 Plot Output Window	77
6.3.1 Lateral Displacement Diagram	78
6.3.2 Slope Diagram	79
6.3.3 Shear and Moment Diagram	81
6.3.4 Bending Stress Diagram	88
7 Output Comparison	89
8 Conclusions.....	105
9 Future Work.....	107
Appendix A- Additional Application Images	111
Appendix B- Beam Problem Outputs and Comparisons.....	113

List of Illustrations

Figure 1 Circle Area Application Example 1.....	6
Figure 2 Circle Area Application Example 1 Script.....	7
Figure 3 Circle Area Application Example 3.....	7
Figure 4 Circle Area Application Example 2 Script.....	8
Figure 5 Client-Side Application [7]	10
Figure 6 Element Deformation Behavior.....	13
Figure 7 FE Modeling of Tapered Support [9]	14
Figure 8 Euler-Bernoulli Beam Deformation	15
Figure 9 Beam Section Equivalent Force and Moment FBD	16
Figure 10 Beam Element Loading and Displacement Free Body Diagram	18
Figure 11 Beam Stiffness Derivation Displacement Scenarios	19
Figure 12 Beam Element Loading Free Body Diagram.....	22
Figure 13 Beam Node Force Compatibility Diagram.....	25
Figure 14 Beam Node Equivalent Force and Moment Diagram.....	26
Figure 15 Beam Node Displacement Compatibility Diagram	27
Figure 16 Application Flowchart	33
Figure 17 Finite Element Beam Analysis Main Menu Input Spreadsheet.....	34
Figure 18 ETBX Provided Beam Problem Visual Display.....	34
Figure 19 Beam Node Input Placement	35
Figure 20 Beam Moment Sign Convention	38
Figure 21 Possible User Input Distributed Loadings	39
Figure 22 JavaScript Application Output Display Table Example.....	42
Figure 23 GBeam Calculation Flowchart	45
Figure 24 Element Stiffness Matrix Formation Code.....	48
Figure 25 Global Stiffness Matrix Row and Column Index Values	49
Figure 26 Global Stiffness Matrix Formation Code	50
Figure 27 Arbitrary Finite Element Analysis Problem Set Up.....	51
Figure 28 Global Beam Nodal Loading and Deformation Diagram.....	53
Figure 29 Beam Deformation with no Hinge Present.....	57
Figure 30 Beam Deformation with Hinge Present [16]	58
Figure 31 Beam Element w/Hinge Located Left Hand Side.....	59
Figure 32 Distributed Loading Equivalent Form.....	65
Figure 33 Possible User Input Distributed Loading.....	66
Figure 34 Distributed Loading Equivalent Forces and Moments	68
Figure 35 Sloped Distributed Load Split Between Individual Beam Elements.....	68
Figure 36 Hinged Node Rotational Displacement Calculation Code	74
Figure 37 Variable Assignment to Local Storage Example Format	75
Figure 38 Beam Problem Displacement Output Example	76
Figure 39 Slope Displacement Diagram Array Data Assignment Code.....	80
Figure 40 Simply Supported Beam Free Body Diagram	85
Figure 41 Beam Section Moment Diagram Creation.....	86
Figure 42 Beam Moment Diagram Creation Continuation.....	86
Figure 43 Node Geometric Property Assignment Example.....	88

Figure 44 Beam Problem 1 Set Up	91
Figure 45 Beam Problem 1 ETBX Diagram Outputs	94
Figure 46 Beam Problem 1 JavaScript Application Diagram Outputs	94
Figure 47 Beam Problem 2 Set Up	96
Figure 48 Beam Problem 2 ETBX & JavaScript Application Output Diagrams	99
Figure 49 Beam Problem 4 Set Up	101
Figure 50 Beam Problem 4 ETBX & JavaScript Application Diagram Outputs	104
Figure 51 JavaScript Application User Declare Nodes Outputs	111
Figure 52 JavaScript Application User Declared Nodes and Undeclared Nodes Outputs	112
Figure 53 Beam Problem 3 Set Up	113
Figure 54 Beam Problem 3 ETBX & JavaScript Application Output Diagrams	116
Figure 55 Beam Problem 5 Set Up	117
Figure 56 Problem 5 ETBX & JavaScript Application Output Diagrams	120
Figure 57 Beam Problem 6 Set Up	121
Figure 58 Beam Problem 6 ETBX & JavaScript Application Output Diagrams	124
Figure 59 Beam Problem 7 Set Up	125
Figure 60 Beam Problem 7 ETBX & JavaScript Application Output Diagrams	128
Figure 61 Beam Problem 8 Set Up	129
Figure 62 Beam Problem 8 ETBX & JavaScript Application Output Diagrams	132
Figure 63 Beam Problem 9 Set Up	133
Figure 64 Beam Problem 9 ETBX & JavaScript Application Output Diagrams	136
Figure 65 Beam Problem 10 Set Up	137
Figure 66 Beam Problem 10 ETBX & JavaScript Application Output Diagrams	140
Figure 67 Beam Problem 11 Set Up	141
Figure 68 Beam Problem 11 ETBX & JavaScript Application Output Diagrams	144
Figure 69 Beam Problem 12 Set Up	145
Figure 70 Beam Problem 12 ETBX & JavaScript Application Output Diagrams	148
Figure 71 Beam Problem 13 Set Up	149
Figure 72 Beam Problem 13 ETBX & JavaScript Application Output Diagrams	152
Figure 73 Beam Problem 14 Set Up	153
Figure 74 Beam Problem 14 ETBX & JavaScript Application Output Diagrams	156
Figure 75 Beam Problem 15 Set Up	157
76 Beam Problem 15 ETBX & JavaScript Application Output Diagrams	160

List of Tables

Table 1 Stiffness Matrix Formulation Scenario 1 End A Values.....	20
Table 2 Stiffness Matrix Formulation Scenario 1 End B Values.....	21
Table 3 Application Available Boundary Conditions.....	36
Table 4 Application Distributed Load Terminology.....	39
Table 5 Distributed Load Example 1 User Input Methods	40
Table 6 Distributed Load Example 2 User Input Methods	40
Table 7 Distributed Load Example 3 User Input Methods	40
Table 8 Moment of Inertia/Young's Modulus Array Data Sorting.....	46
Table 9 Output Diagram Data Source Array Name and Functions	78
Table 10 Shear Diagram Data Arrays Names and Functions	82
Table 11 Moment Diagram Data Arrays Names and Functions	87
Table 12 Beam Problem Elements Physical Properties and Geometries.....	90
Table 13 Beam Problem 1 Input Format.....	92
Table 14 Beam Problem 1 Node Displacement Values	93
Table 15 Beam Problem 1 Node Displacement Value Comparison	93
Table 16 Beam Problem 2 Input Format.....	97
Table 17 Beam Problem 2 Node Displacement Values	98
Table 18 Beam Problem 2 Node Displacement Value Comparisons.....	98
Table 19 Beam Problem 4 Input Format.....	102
Table 20 Beam Problem 4 Node Displacement Values	103
Table 21 Beam Problem 4 Node Displacement Value Comparisons.....	103
Table 22 Beam Problem 3 Input Format.....	114
Table 23 Beam Problem 3 Node Displacement Values	115
Table 24 Beam Problem 3 Node Displacement Value Comparisons.....	115
Table 25 Beam Problem 5 Input Format.....	118
Table 26 Beam Problem 5 Node Displacement Values	119
Table 27 Beam Problem 5 Node Displacement Value Comparisons.....	119
Table 28 Beam Problem 6 Input Format.....	122
Table 29 Beam Problem 6 Node Displacement Values	123
Table 30 Beam Problem 6 Node Displacement Value Comparisons.....	123
Table 31 Beam Problem 7 Input Format.....	126
Table 32 Beam Problem 7 Node Displacement Values	127
Table 33 Beam Problem 7 Node Displacement Value Comparisons.....	127
Table 34 Beam Problem 8 Input Format.....	130
Table 35 Beam Problem 8 Node Displacement Values	131
Table 36 Beam Problem 8 Node Displacement Value Comparisons.....	131
Table 37 Beam Problem 9 Input Format.....	134
Table 38 Beam Problem 9 Node Displacement Values	135
Table 39 Beam Problem 9 Node Displacement Value Comparisons.....	135
Table 40 Beam Problem 10 Input Format.....	138
Table 41 Beam Problem 10 Node Displacement Values	139

Table 42 Beam Problem 10 Node Displacement Value Comparisons.....	139
Table 43 Beam Problem 11 Input Format.....	142
Table 44 Beam Problem 11 Node Displacement Values	143
Table 45 Beam Problem 11 Node Displacement Value Comparisons.....	143
Table 46 Beam Problem 12 Input Format.....	146
Table 47 Beam Problem 12 Node Displacement Values	147
Table 48 Beam Problem 12 Node Displacement Value Comparisons.....	147
Table 49 Beam Problem 13 Input Format.....	150
Table 50 Beam Problem 13 Node Displacement Values	151
Table 51 Beam Problem 13 Node Displacement Value Comparisons.....	151
Table 52 Beam Problem 14 Input Format.....	154
Table 53 Beam Problem 14 Node Displacement Values	155
Table 54 Beam Problem 14 Node Displacement Value Comparisons.....	155
Table 55 Beam Problem 15 Input Format.....	158
Table 56 Beam Problem 15 Node Displacement Values	159
Table 57 Beam Problem 15 Node Displacement Value Comparisons.....	159

1. Introduction

Moore's Law, created by Intel founder Gordon Moore in 1965, states that computing power would double while simultaneously decreasing in relative cost every two years [1]. For fifty years this trend has been followed by chip companies, including current giants Intel and Samsung. This trend of increasing computing power at decreasing cost has been beneficial in the engineering profession. In this era, where a smart phone has more computing power than the average computer of the 1990's, it is possible to download and run a multitude of engineering software that can aid in the design process. OpenVSP, Abaqus, Ansys, SolidWorks, PTC Creo; these are only a few of the popular programs that can allow an engineer to model and analyze a project they are working on and get results that can be used to judge whether or not the current design iteration meets the current project requirements. The main issue with most of these programs, as well as many other unnamed programs, is that a license can be required in order to run the software on a computer and even then, the software may work only on the single computer in which it was installed. These licenses can be sold by the company that own the software that the user can be interested in and can come in a variety of access levels that can limit the extent of features that the user can access in the software or the amount of computing power that the user is allowed to use the software to perform.

2. Installed Software Licensing

2.1 SOFTWARE LICENSING BEFORE THE INTERNET

The internet makes the acquisition and installation of software relatively easy. A user simply has to visit the website that contains a link to the software that they want, pay the license fee and download the program onto their computer and run it. This makes things easier for

companies in regards to how they keep track of who buys their product. Before the internet became wide spread it was the norm for a user to contact a company, purchase a physical copy of software, and install the software and the software license on a computer. The big difference between then and now is that it was easier to use the physical copy to install the software and the license on multiple computers. With companies now selling their products online, it is easier to keep track of who installs the product on any given computer and it becomes easier to control exactly how many times the product is installed. This new idea of being able to more accurately control the installation of software has led to the creation of different kinds of licenses in order to try and fit the different needs of consumers while still retaining control over who is allowed to use the desired product. [2]

2.2 Types of Licenses

The CAD industry provides a good example over the different types of licenses available along with their possible downsides and benefits. The different types of CAD licenses usually include but are not limited to: Perpetual, Standalone, Network, Subscription, Pay-As-You-Go.

- **Perpetual Licenses**
 - Most common type of license and has been an industry standard for decades.
 - User is allowed to install the license on one computer only.
 - Allows user to keep and use the software indefinitely.
 - Software will be stuck in the version the user installed unless license is bought for possible future releases.

- Lack of automatic upgrades might cause users to miss out on possible tool upgrades, bug fixes, performance improvements, or hardware compatibilities.
- **Standalone Licensing**
 - Standalone licenses are usually perpetual licenses.
 - Allows user to install software one time on one computer.
 - Some companies allow license installation on two separate machines.
 - If license installed on two separate machines, the software can only run from one machine at a time.
- **Network Licensing**
 - Allows users to install the chosen software on multiple computers simultaneously.
 - Softwares using network licensing are often controlled by network server.
 - Software using network servers must be managed and maintained.
 - These types of licenses are usually an additional cost to a standalone license.
 - While this type of license can also count as a perpetual license, there is an additional cost of server maintenance.
 - This license type allows multiple users to access the software for a short period of time.
 - This type of license can be cheap for multiple users if not every users require access to the software simultaneously, or for extended periods in a day.

- **Subscription Licensing**

- Subscription based licensing is seen as a solution to the problem of having to pay for software that needs updates.
- This type of licensing is usually an add-on to perpetual licensing that lasts for a specific amount of time (commonly a year)
- User pays subscription fee when purchasing perpetual license which grants the user rights to new versions of software during subscription period.
- User can opt not to renew subscription and continue using the last software update before subscription ended.
- User risks paying a subscription fee for software that might not update during the subscription period.

- **Pay-as-you-go (Rental Licensing)**

- Pay-as-you-go is a newer type of software licensing.
- With this type of licensing, the user has the right to paid software for a limited time (usually a period of 30 days)
- This type of license is desirable for users that do not wish to buy perpetual license and is often less expensive.
- With this type of license users will always have access to the latest software update.

All of this leads to the conclusion that using engineering software to perform analysis for engineering problems can be costly in a time where significant computing power is widely available. It is this conclusion that demonstrates the need for freely available computation

methods for a variety of engineering problems. This paper focuses on the development of an application that can perform beam analysis using finite element methods using the programming languages JavaScript and HTML. The application is meant to be freely available for any user to operate and modify if they wish to do so. The application, from here on referred to as GBeam, is meant to take in any beam data and any forces and moments that the user wishes to apply to the beam, and output displacement solutions and shear/moment diagrams.

3. JavaScript Programming Language

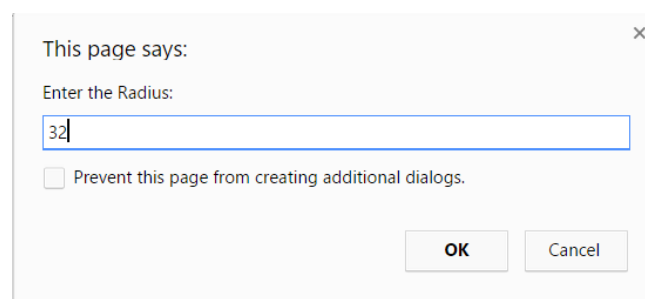
3.1 JavaScript: A Brief Overview

The language used here to program the finite element beam analysis application is JavaScript [3]. Prior to the creation of JavaScript by Netscape in the 1990's, web pages were static, meaning that there was no way for a user to interact with a web page without having to reload the web page entirely. JavaScript allowed web developers to create web pages that were interactive and capable of performing basic operations. The syntax used in JavaScript is inspired by the syntax used in popular programming languages C++ and Java, but most of the similarities between JavaScript and the other two languages end there. JavaScript can best be described as a light-weight, object oriented, interpreted programming language [4]. What this means is that JavaScript is a programming language that is meant to have a minimal memory footprint while being easy to implement. Object-oriented means that in JavaScript the programmer can define the data type of a data structure as well as the types of functions that can be applied to the data structure. In addition, the programmer can set the relationship between objects and also allow objects to inherit characteristics from other objects [5]. Finally, interpreted means that code executed is not directly run by the target machine. Instead another program, known as the

interpreter, will take the code and execute the original source code. An example of interpretation would be if the interpreter sees an operator such as the addition operator or the multiplication operator. When the interpreter reads the program and runs upon an operator such as the ones mentioned above, it will call a function representing that operation. If the interpreter runs across the addition operator it will call a function that might be called “addition(a,b)” which takes as its inputs variables a and b and execute the machine code equivalent of the addition operation.

3.2 Developing Computational Applications with JavaScript

The appeal of using JavaScript to create the finite beam analysis script is its widespread use in the Internet. Since its introduction in the 1990’s JavaScript has gained enough popularity to the point that all browsers are equipped with their own personal JavaScript engines that allow the browser to run JavaScript using just-in-time compiling and act as the interpreter for the JavaScript language [6]. The ability to combine HTML and JavaScript allows many types of analysis application to be written, ranging from extremely simple to something more complicated. An example of one such simple application is shown in this section in the form of an application that allows the user to find the area of a circle when the radius is input by the user. Figure 1 shows, both the input and the output format for this simple application.



$\text{Area} = 3216.990877275948$

Figure 1 Circle Area Application Example 1

```

<!DOCTYPE html>
<html>
  <script>
    var radius = prompt("Enter the Radius: ");
    radius = parseFloat(radius);
    var area = Math.PI*radius*radius;
    document.write("Area = ", area);
  </script>
</html>

```

Figure 2 Circle Area Application Example 1 Script

This kind of application can be very simple to write, as shown by Figure 3, and are not limited to one specific user interface. This same application that calculates a circle area can take a separate appearance, while still conducting the exact same analysis. The only difference between these two applications is the manner in which their respective scripts are written.

Enter radius; press tab or return.

Radius	<input type="text" value="2"/>
Area	<input type="text" value="12.5663706"/>

Figure 3 Circle Area Application Example 3

```

<!DOCTYPE html>
<HTML>
<TITLE>Circle</TITLE>

<FORM>
  <p>Enter radius; press tab or return.</p>
  <div align="left">
    <table border="0" cellpadding="0" width="33%" height="135">
    <tr>
      <td width="100%" height="133">
        <div align="center">
          <table border="0" cellpadding="0"
            width="100%">
            <tr>
              <td width="33%"><font face="Calibri">
                Radius</font></td>
              <td width="33%"><INPUT
                onchange=radius(form);
                value=" " maxLength=9 size=6
                name=radius></td>
            </tr>
            <tr>
              <td width="33%"><font face="Calibri">
                Area</font></td>
              <td width="33%"><INPUT value=" "
                maxLength=7
                size=6 name=area></td>
            </tr>
          </table>
        </div>
      </td>
    </tr>
    </table>
  </div>
</FORM>
<SCRIPT language=javascript type=text/javascript>
  function circle(f) {
    var r = parseFloat(f.radius.value);
    f.area.value = Math.PI*r*r;
  }
</SCRIPT>
</HTML>

```

Figure 4 Circle Area Application Example 2 Script

The ability to create analysis application, both simple and complex, which is easily accessible from any computer web browser, allows the creation of a widely available free program that solves engineering problems feasible. One of the main issues with mainstream engineering software is the fact that more often than not they are not free and once installed on a specific machine it is permanently tied to that machine unless the software and a new license is installed on a separate machine, which might incur more cost on the user side in order to install the software license. With JavaScript, the interpreter used to execute the program is already pre-installed on the machine internet browser. This means that any potential user with an internet

connection would be able to download the script that runs the finite element analysis, conduct beam analysis and also modify the program to give outputs specific to the needs of the user. All of this is possible due to two main points. The first point is that the JavaScript code is embedded into the webpage. The code that makes up a web page can be seen by choosing to inspect the current webpage. The code that makes up the web page can even be edited and manipulated if the user chooses to save the contents of the web page they are currently on and open the contents of the webpage on a text editor such as notepad.

This leads to the second point, that the finite element beam analysis script has the possibility to be extremely customizable, considering that the beam analysis program utilized client-side scripting and not server-side scripting. When writing a script for a webpage the programmer can choose to make their webpage function through client-side or server-side scripting. Client-side scripting gives everything needed to run the program to the user. An example of client-side scripting would be giving the user a form sheet that the user can input the data onto and also giving them the tools to perform whatever operation the form is meant to complete. In the case of the finite element beam analysis program the user is given a spreadsheet to add the beam data and applied beam forces to. The program also gives them access to the JavaScript code that analyses the data and delivers the desired solution. Thus the entirety of the finite element analysis is performed on the users' computer. Server-side scripting is different in that the user will still be given something like a form sheet to fill in with their data, but in order to perform the operation that the form sheet is meant to perform, the data in the form sheet is sent off to a server (hence the name server-side) which will perform the necessary analysis and then return the desired outputs to the user. The main difference between the two scripting methods is that client-side scripting gives the user all of the tools they need to run the main operation of the

script, while server-side scripting has the main operation done by an outside source and the results returned to the user. A client-side based script allows the user to have full access to the tools that conduct the analysis for the beam. This is what allows the user full customization of the analysis tools.

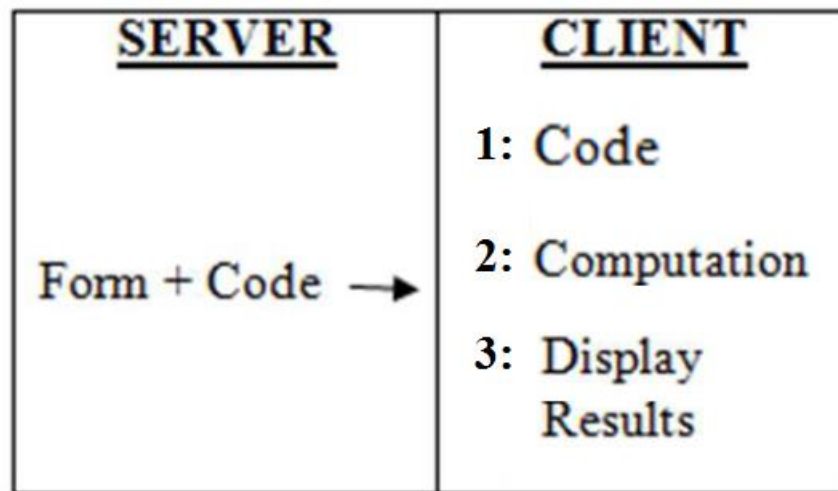


Figure 5 Client-Side Application [7]

4. The Finite Element Method

4.1 Overview of FEM: History, Advantages, and Concepts

Finite element methods can be described as a widely distributed method used to obtain approximate solutions for various boundary value or initial value problems. These types of problems can be described as steady state or unsteady state problems. There is no one direct creator of the finite element method, but instead a series of contributions by different engineers. Recognized major contributors to the finite element method include engineers John Argyris, Ray Clough, M.J. Turner, and O.C. Zienkeiwics who each contributed to different aspects of the method that makes up finite element analysis [8]. These contributions include the use of energy theorems in finite element analysis, the use of finite elements in plane stress problems, and the

use of finite elements to find the stiffness and deflection of complex structures. One of the earliest forms of the finite element method can be traced back to the 1950's at the Boeing Company with Professor Ray Clough who was tasked by M.J. Turner with finding the vibrational properties of a large scale delta wing model. After some disappointing initial results, Turner suggested the formulation of a mathematical model as an assembly of 2D plates connected at their corners. This idea formed the concept for the finite element method. Mathematical models for rectangular and triangular plates were formed, with more emphasis on the formulation of the triangular plate since the triangular plate fit the shape of the delta wing better and it was simpler to derive the in-plane stiffness of the triangle plate than it was to derive the in-plane stiffness of rectangular plate. As work on the formation of the triangle elements continued it became apparent that the triangle assembly yielded good vibrational results and that the approximations of the assembly converged to the results of the physical model as the mesh of triangular elements on the model was refined. It was eventually noted that the use of this direct stiffness method could be used in stress/displacement applications. As time went on, papers on the direct stiffness method were published and more engineers took note. This eventually led to more in depth study and contributions to form what we now know as the finite element method.

There are many advantages to using the finite element method to conduct the beam analysis in with JavaScript [9]. One of the biggest advantages to FEM is that it can be applied to any number of field problems. Examples of field problems can be heat transfer problems, stress analysis problems, magnetic field problems, etc. Finite element analysis also has no geometric restrictions. The elements used to conduct the analysis of the field problem will form into the shape of the model that the problem is based on. This is very important when put in conjunction with the fact that there are no restrictions with the boundary conditions and loadings when using

FEM. This means that in any given body any portion can be supported which also having portions that have distributed or direct loading applied. Additionally the body or region being analyzed does not have to be one single isotropic material. Since the body is recreated using elements, each element has the option of having different material properties from its neighbor. Finally using FEM does not restrict the user to using only beams or only plates etc. Using FEM allows the user to analyze a problem that consists of a combination beams, plates, cables and trusses. Overall this shows that the finite element method is a very flexible tool that allows analysts to tackle a wide variety of problems in order to arrive at an approximate solution that closely resembles the real world response of the problem that the finite element model is based off. These approximations are limited by the analysts' understanding of the problem being modeled and the refinement of the grading mesh that is applied to the finite element model.

Basic finite element method can be said to function on two principal concepts. The first is the equilibrium condition that states the sum of the forces and moments on the body will equal to zero for static loading. The second is the compatibility condition which states that elements connected at a common node along a common edge or common surface before loading remains connected at that node after deformation. A visual example of the compatibility condition is shown below, which shows the possible and not possible deformation after loading of two elements connected by common nodes.

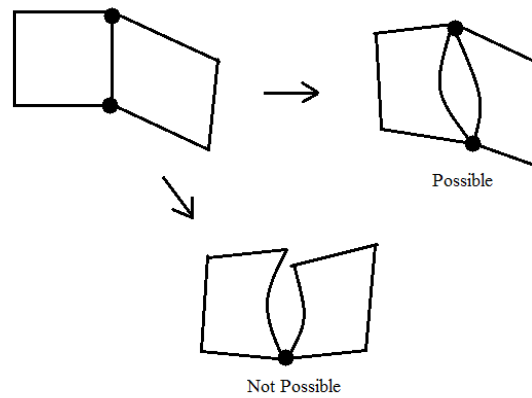


Figure 6 Element Deformation Behavior

The main method used in the finite element beam analysis program is the stiffness method. The stiffness method assumes that the displacements of the nodes are unknown and uses both compatibility and equilibrium conditions to express the governing equation of the problem. The fundamental equation is written as:

$$\{R\} = [K]\{D\} \quad (1)$$

Where $\{R\}$ represents the applied forces on the body, $[K]$ represents the stiffness matrix of the body which is a function of the size, shape and material of the body and $\{D\}$ represents the displacement of the nodes of each element applied to the body to be analyzed. Each problem that applies the finite element method can be broken down into a few key steps. For each problem the engineer must identify what results are being sought and what accuracy is required of the approximate solution. It must be understood that the method is applied to a model rather than the physical problem. Models allow the engineer to ignore unnecessary details that would complicate the analysis and focus only on the required details. These details can then be described mathematically by differential equations and applied boundary conditions. Finally the model itself is discretized into a finite number of elements that will be used to find the

approximate solution. Refining the model to contain more elements may affect the overall accuracy of the solution found. Nodes that connect each element together will be used in conjunction with the compatibility condition in order to ensure that the stiffness matrix that represents the body being analyzed is correct.

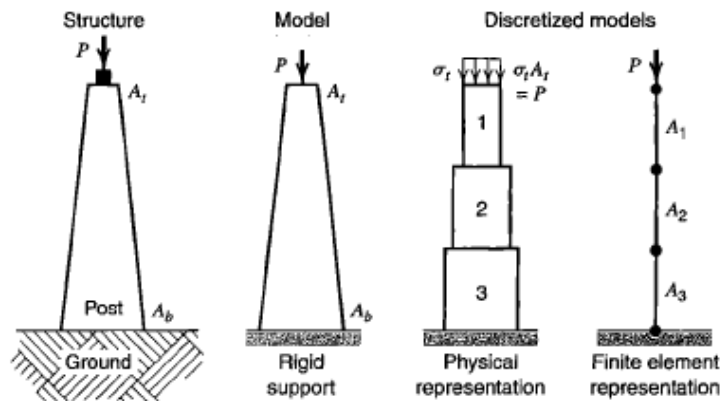


Figure 7 FE Modeling of Tapered Support [9]

The Figure 7 above shows that each individual element of the discretized model shares one common node. This concept of each element of a discretized model sharing at least one node is applicable to all cables, trusses, bars and plates. Each structural component can have different displacement unknowns. With a beam, which is the main focus of this finite element analysis program, the displacement values for each node are lateral displacements and rotation (slope) displacements of the neutral axis. Likewise the forces that can be applied on the beam are applied loads and moments. Knowing what forces can be applied at each node and what displacements can occur at each node allows the engineer to formulate a stiffness matrix that represents each individual element from the discretized model.

4.2 Beam Models: Euler-Bernoulli Beam

There are two types of beam models in classical beam analysis, the Timoshenko beam theory and the Euler-Bernoulli beam theory. Ultimately both theories can be used in order to

find the displacements of a beam. The differences between both theories can be said to stem from a difference in the assumption of shear deformation. There are three rules that are applied to an Euler-Bernoulli model [10]:

- i. The beam cross section is rigid on its plane
- ii. The beam cross section rotates around a neutral surface, remaining plane
- iii. The cross section remains perpendicular to the neutral surface during deformation

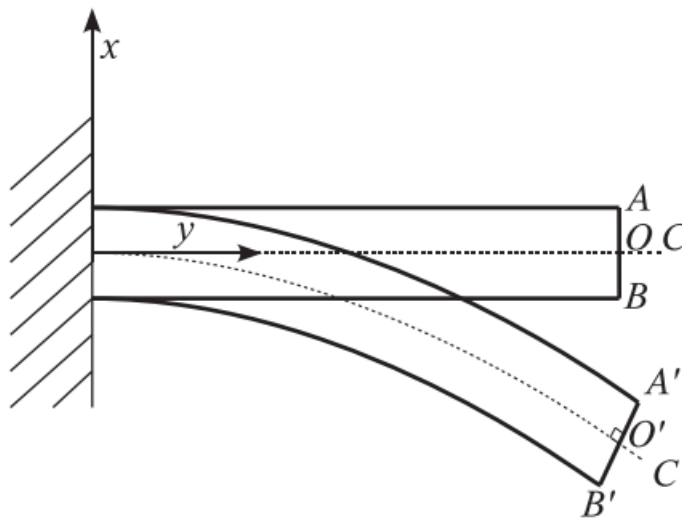


Figure 8 Euler-Bernoulli Beam Deformation

Figure 8 shows the deformation of an Euler-Bernoulli beam. Figure 8 shows the shape of the beam after deformation, which shows the end cross section of the beam as perpendicular to the neutral axis, following the third rule of the Euler-Bernoulli model. Because the cross section remains perpendicular to the neutral axis, it can be concluded that there is no shear deformation in an Euler-Bernoulli model.

4.3 Formulation of Beam Element Stiffness Matrix: Euler-Bernoulli

Formulating the stiffness matrix for a beam element involves an integration of the bending stiffness equation in order to solve possible deflections for a beam element. Integrating the bending stiffness equation once gives us the equation of the rotation/slope equation of the beam. Integrating twice will give us the equation for vertical displacement of the beam. Each of these integrations will cause a series of unknown constants to appear. These unknown constants must be determined in order to find the stiffness matrix for the beam element.

The bending stiffness equation is dependent on the moment value in the x -direction. This moment equation is easily derived using a free body diagram of a section of the beam and the equilibrium condition applied to the summation of the moments. This allows the moment equation to be written as a function of x , which allows the bending stiffness to be found [11].

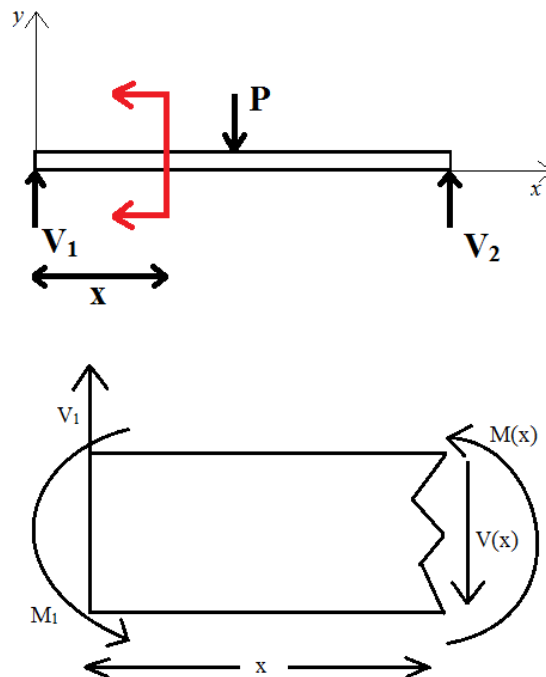


Figure 9 Beam Section Equivalent Force and Moment FBD

$$\sum M_x = 0 \quad (2)$$

$$M_1 + M(x) - V_1x = 0 \quad (3)$$

$$M(x) = V_1x - M_1 \quad (4)$$

Equation 4, which calculates the moment across the length of the beam, can be substituted into Equation 5 [12], turning it into Equation 6, in order to begin to formulate the Euler-Bernoulli beam stiffness matrix.

$$\frac{\partial^2 v(x)}{\partial x^2} = \frac{M(x)}{EI} \quad (5)$$

$$\frac{\partial^2 v(x)}{\partial x^2} = \frac{(V_1x - M_1)}{EI} \quad (6)$$

$$\theta = \int \frac{\partial^2 v(x)}{\partial x^2} = \frac{1}{EI} \left(\frac{V_1x^2}{2} - M_1x \right) + C_1 \quad (7)$$

$$v(x) = \iint \frac{\partial^2 v(x)}{\partial x^2} = \frac{1}{EI} \left(\frac{V_1x^3}{6} - \frac{M_1x^2}{2} \right) + C_1x + C_2 \quad (8)$$

The beam element stiffness matrix is formed by using a free body diagram to list out the possible deformations of a single beam. Figure 10 below shows that the beam has a total of four discrete values: $v_1, \theta_1, v_2, \theta_2$.

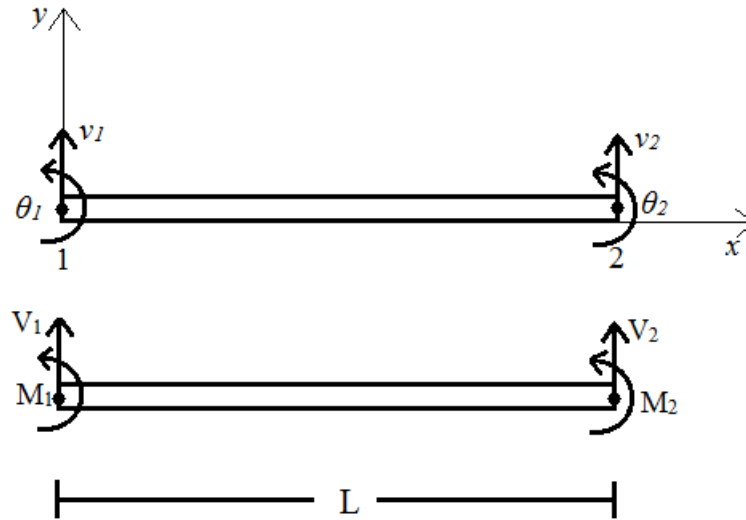


Figure 10 Beam Element Loading and Displacement Free Body Diagram

These values fill the displacement matrix $\{D\}$ while the shear and moment values fill the force matrix $\{R\}$. The beam element will have two types of deformations. One deformation will be the rotation angle, θ , and the other displacement will be lateral displacement, v . Additionally both of these kinds of deformations are not mutually exclusive, meaning that it is possible to have a rotation occur in the beam without having a vertical displacement. In total there will be four different displacement scenarios that need to be solved in order to find the complete element stiffness matrix for a beam element.

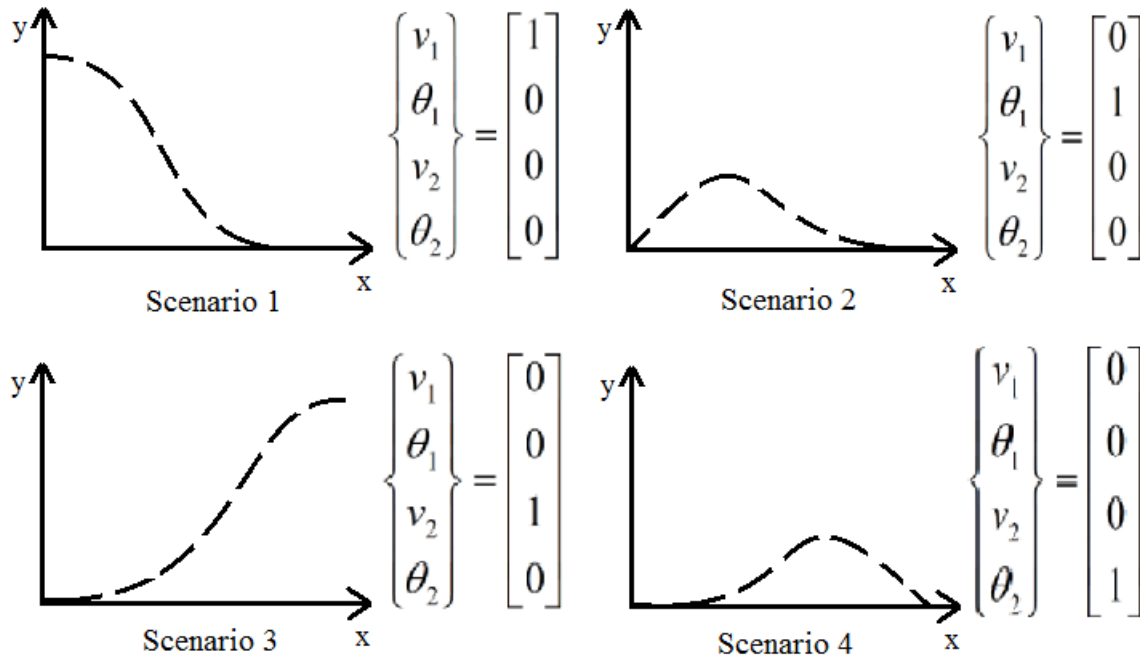


Figure 11 Beam Stiffness Derivation Displacement Scenarios

Each of these scenarios refer to either a vertical displacement or a rotation/slope displacement in the beam that can occur at the beginning of the beam, where $x=0$, or at the end of the beam, where $x=L$. The procedure to find the element stiffness matrix, is shown in detail below, choosing a displacement scenario and apply the known values of that scenario to the rotation/slope and vertical displacement equations. After applying the known values, the unknown values will be solved for in order to find the values of shear force, V , and moment, M , at both ends of the beam. Solving each displacement scenario will give one column of the 4×4 stiffness matrix.

The first column is solve is solved for using scenario one. In scenario one, we start at the left hand side of the beam where we know the following:

Table 1 Stiffness Matrix Formulation Scenario 1 End A Values

Variable	Value
x	0
v_I	1
θ_1	0

These values can then be plugged in to the known rotation/slope and lateral displacement equations to solve for unknowns C_1 and C_2 . This will allow the unknown variables of V_1 , M_1 , V_2 , and M_2 to be solved for in terms of the physical properties of the beam element.

$$\theta(x) = \frac{1}{EI} \left(\frac{V_1 x^2}{2} - M_1 x \right) + C_1 \quad (9)$$

$$\theta(0) = 0 = \frac{1}{EI} \left(\frac{V_1(0)^2}{2} - M_1(0) \right) + C_1 \quad (10)$$

$$C_1 = 0$$

$$v(x) = \frac{1}{EI} \left(\frac{V_1 x^3}{6} - \frac{M_1 x^2}{2} \right) + C_1 x + C_2 \quad (11)$$

$$v(0) = 1 = \frac{1}{EI} \left(\frac{V_1(0)^3}{6} - \frac{M_1(0)^2}{2} \right) + C_1(0) + C_2 \quad (12)$$

$$C_2 = 1$$

Now the values of the constants C_1 and C_2 are known. The next step is to look at the right hand side of the beam where the application of a non-zero value for variable x will allow the unknown variables of V_I and M_I to be solved.

Table 2 Stiffness Matrix Formulation Scenario 1 End B Values

Variable	Value
x	L
v_2	0
θ_2	0

$$\theta(L)=0=\frac{1}{EI}\left(\frac{V_1L^2}{2}-M_1L\right) \quad (13)$$

$$0=\frac{V_1L^2}{2EI}-\frac{M_1L}{EI} \quad (14)$$

$$\frac{V_1L^2}{2EI}=\frac{M_1L}{EI} \quad (15)$$

$$V_1=\frac{2M_1}{L} \quad (16)$$

$$v(L)=0=\frac{1}{EI}\left(\frac{V_1L^3}{6}-\frac{M_1L^2}{2}\right)+1 \quad (17)$$

$$0=\frac{V_1L^3}{6EI}-\frac{M_1L^2}{2EI}+1 \quad (18)$$

$$\frac{V_1L^3}{6EI}=\frac{M_1L^2}{2EI}-1 \quad (19)$$

$$V_1=\frac{3M_1}{L}-\frac{6EI}{L^3} \quad (20)$$

Now the lateral displacement v_l is found in terms of moment M_l and the other physical properties of the beam element, such as the material and the area moment of inertia. Taking these two different equations for vertical displacement, it is possible to solve for the moment and displacement values at the left hand side of the beam in terms of only the physical values of

Young's modulus, length, and area moment of inertia since there are two equations and two unknowns.

$$V_1 = \frac{2M_1}{L} = \frac{3M_1}{L} - \frac{6EI}{L^3} \quad (21)$$

$$\frac{3M_1}{L} - \frac{2M_1}{L} = \frac{6EI}{L^3} \quad (22)$$

$$M_1 = \frac{6EI}{L^2} \quad (23)$$

$$V_1 = \frac{2M_1}{L} = \frac{2\left(\frac{6EI}{L^2}\right)}{L} \quad (24)$$

$$V_1 = \frac{12EI}{L^3} \quad (25)$$

Now the first two rows of the first column of the beam element stiffness matrix are known. In order to find the values of the last two rows the equilibrium principle must be applied to the beam element. This allows the final two force values to be found in terms of the physical properties of the beam element.

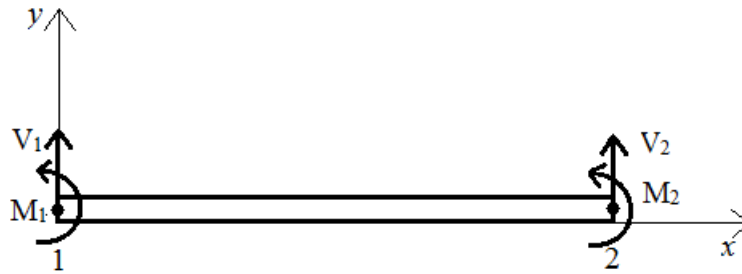


Figure 12 Beam Element Loading Free Body Diagram

$$\sum F_y = 0 = V_1 + V_2 \quad (26)$$

$$V_2 = -V_1 = -\frac{12EI}{L^3} \quad (27)$$

$$V_2 = -\frac{12EI}{L^3} \quad (28)$$

$$\sum M_0 = M_1 + M_2 + V_2 L \quad (29)$$

$$M_2 = -(M_1 + V_2 L) \quad (30)$$

$$M_2 = -\left(\left(\frac{6EI}{L^2}\right) + \left(-\frac{12EI}{L^3}\right)L\right) \quad (31)$$

$$M_2 = -\frac{6EI}{L^2} + \frac{12EI}{L^2} \quad (32)$$

$$M_2 = \frac{6EI}{L^2} \quad (33)$$

The values for V_1 , M_1 , V_2 , and M_2 found in terms of the beam length, area moment of inertia, and the Young's modulus make up the first column of the element stiffness matrix of the beam element.

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix} = \begin{bmatrix} 12EI/L^3 & x & x & x \\ 6EI/L^2 & x & x & x \\ -12EI/L^3 & x & x & x \\ 6EI/L^2 & x & x & x \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (34)$$

Each displacement scenario corresponds to a different column in the element stiffness matrix and each column is found using a method similar to the method used to find the first column of the beam element stiffness matrix. If the remaining three displacement scenarios are applied to the integrations of the beam stiffness equation, the result will be a beam stiffness matrix with the following values:

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix} = \begin{bmatrix} 12EI/L^3 & 6EI/L^2 & -12EI/L^3 & 6EI/L^2 \\ 6EI/L^2 & 4EI/L & -6EI/L^2 & 2EI/L \\ -12EI/L^3 & -6EI/L^2 & 12EI/L^3 & -6EI/L^2 \\ 6EI/L^2 & 2EI/L & -6EI/L^2 & 4EI/L \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (35)$$

This matrix is known as a beam element stiffness matrix, meaning that this matrix represents the stiffness value for one beam element out of an entire field problem that can contain multiple beam elements. As stated by Cook in the first chapter of his book, *Concepts and Applications of Finite Element Analysis*, although an finite element analysis solution may not be exact, the solution can be improved by using more elements to represent the structure. This means that if an engineer is trying to find good approximations for the deformation of a beam, using more than one element may be required. This would mean there would be multiple elements that will each have their own element stiffness matrix. [9]

4.4 Formulation of Beam Global Stiffness Matrix: Euler-Bernoulli

In order to conduct the analysis of a beam problem with more than one element, all the element stiffness matrices would need to be combined into one matrix that is known as the global stiffness matrix. This global stiffness matrix is formed using the compatibility condition that ensures that each shared node will displace in the same direction, which will avoid a discontinuity. This compatibility condition also means that certain values of the element stiffness matrices will superimpose with values from other element stiffness matrices. Forming the global stiffness matrix involves knowing which elements share nodes, what externally applied loads run through those nodes, and what displacements might occur at those nodes. A visual example shown in the Figure 13 below will help to understand compatibility and how it relates to the creation of the global stiffness matrix. This figure shows a hypothetical problem that contains

two beam elements. Each beam element has the option of having different properties, which may or may not have different element stiffness matrices. Thus it is simpler to label the each element stiffness matrix as element a and element b . Each beam node can accept two externally applied loads. These loads are represented by R_1 to R_6 .

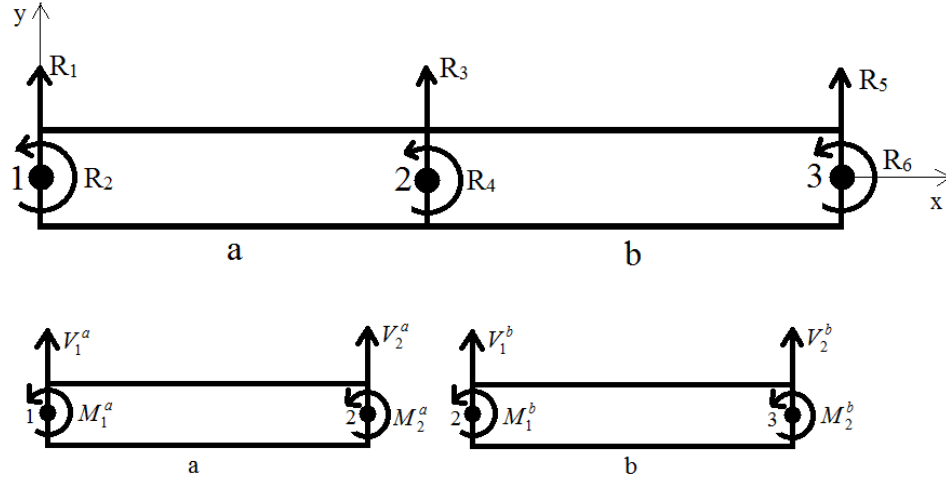


Figure 13 Beam Node Force Compatibility Diagram

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = [K] \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (36)$$

Figure 13 shows that element a and element b both share a common node, node 2. For the complete beam, a series of six possible loadings are assigned to the nodes. These forces, two to a node, represent the possible moment and point loads that can be applied to a single node.

Breaking the beam into its individual components, a series of point loads and moments can be assigned to the individual nodes. The names of these forces V_1 , M_1 , V_2 , and M_2 represent the forces on the left and right hand side of the elements while the lettered super script represents the

element in which these forces are applied. A relationship between the element forces V and M and the global force R can be found using a free body diagram and the equilibrium condition.

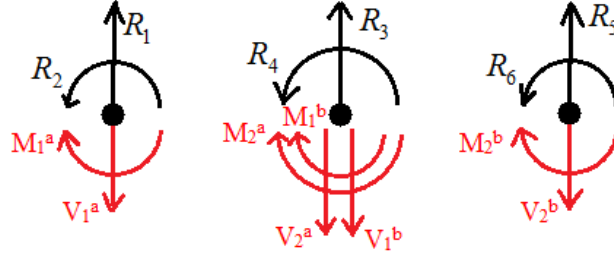


Figure 14 Beam Node Equivalent Force and Moment Diagram

$$\sum F = 0 \quad (37)$$

$$R_1 = V_1^a \quad (38)$$

$$R_3 = V_2^a + V_1^b \quad (39)$$

$$R_5 = V_2^b \quad (40)$$

$$\sum M = 0 \quad (41)$$

$$R_2 = M_1^a \quad (42)$$

$$R_4 = M_2^a + M_1^b \quad (43)$$

$$R_6 = M_2^b \quad (44)$$

Each node can displace in one of two ways, rotationally or laterally. As stated previously, basic FEM relies on the concepts of equilibrium and compatibility. Equilibrium allows the global forces, R , to be written in terms of the forces and moments applied to the nodes of the individual elements. Compatibility will allow the global displacements, D , to be written in their equivalent element displacement values. As before, it is simpler to see the compatibility principle visually. Compatibility ensures that nodes shared between elements in a given problem will displace in the

same direction. This would mean, going from Figure 15 below, global beam displacement D_1 would correspond with element a displacement value v_1^a .

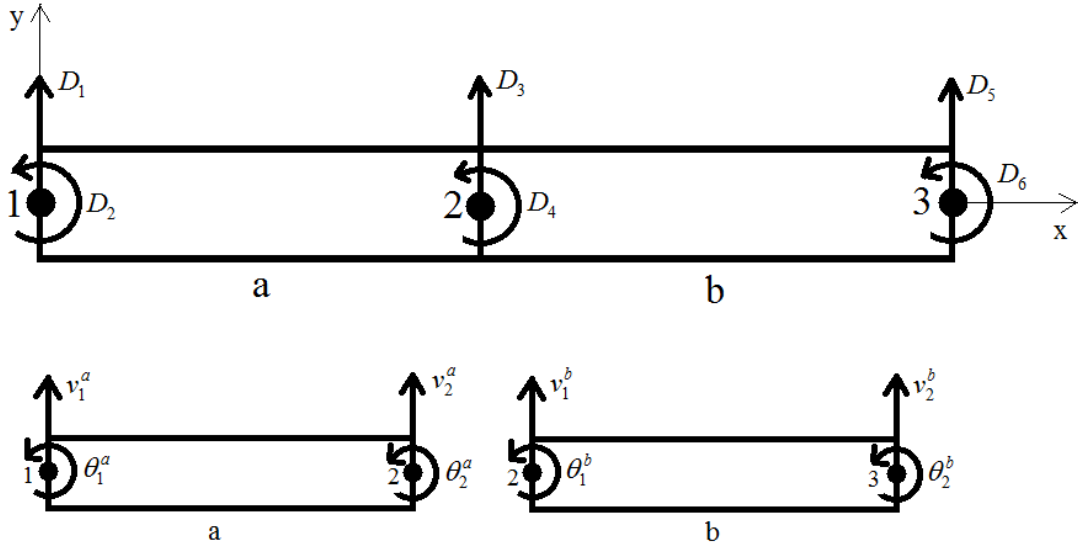


Figure 15 Beam Node Displacement Compatibility Diagram

$$D_1 = v_1^a \quad (45)$$

$$D_2 = \theta_1^a \quad (46)$$

$$D_3 = v_2^a = v_1^b \quad (47)$$

$$D_4 = \theta_2^a = \theta_1^b \quad (48)$$

$$D_5 = v_2^b \quad (49)$$

$$D_6 = \theta_2^b \quad (50)$$

At this point in the derivation, both global variables for forces and displacements are written in terms of the forces and displacements of the beam elements. Finally, the element stiffness matrices are used in order to find the global stiffness matrix of the beam. The governing equation for each beam element can be written as the following.

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix}^a = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix}^a \quad (51)$$

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix}^b = \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix}^b \quad (52)$$

The free body diagram of Figure 13 and Figure 15 show that the governing equation using the global stiffness matrix will have a total of six loadings and six displacements. Using the governing equations of the beam elements and the equilibrium and compatibility conditions it is possible to find the global stiffness matrix values in terms of the element stiffness values. A simple example using equilibrium, compatibility, and the governing equations would allow the first row to the global stiffness matrix to be found.

$$R_1 = V_1^a \quad (53)$$

$$D_1 = v_1^a \quad (54)$$

$$D_2 = \theta_1^a \quad (55)$$

$$D_3 = v_2^a = v_1^b \quad (56)$$

$$D_4 = \theta_2^a = \theta_1^b \quad (57)$$

$$V_1^a = A_{11}v_1^a + A_{12}\theta_1^a + A_{13}v_2^a + A_{14}\theta_2^a \quad (58)$$

$$R_1 = A_{11}D_1 + A_{12}D_2 + A_{13}D_3 + A_{14}D_4 \quad (59)$$

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & 0 & 0 \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (60)$$

The stiffness matrix of element b will not begin to affect the global stiffness matrix until row three of the global stiffness matrix. This is due to the fact that row three and four of the global stiffness matrix represents the shared node of between element a and element b , while rows five and six represent the right hand side node of element b . To see how the shared node affects the global stiffness matrix, a similar approach to the method shown above for the first node can be used.

$$R_3 = V_2^a + V_1^b \quad (61)$$

$$V_2^a = A_{31}v_1^a + A_{32}\theta_1^a + A_{33}v_2^a + A_{34}\theta_2^a \quad (62)$$

$$V_1^b = B_{11}v_1^b + B_{12}\theta_1^b + B_{13}v_2^b + B_{14}\theta_2^b \quad (63)$$

$$R_3 = A_{31}v_1^a + A_{32}\theta_1^a + A_{33}v_2^a + A_{34}\theta_2^a + B_{11}v_1^b + B_{12}\theta_1^b + B_{13}v_2^b + B_{14}\theta_2^b \quad (64)$$

$$R_3 = A_{31}D_1 + A_{32}D_2 + A_{33}D_3 + A_{34}D_4 + B_{11}D_3 + B_{12}D_4 + B_{13}D_5 + B_{14}D_6 \quad (65)$$

$$R_3 = A_{31}D_1 + A_{32}D_2 + (A_{33} + B_{11})D_3 + (A_{34} + B_{12})D_4 + B_{13}D_5 + B_{14}D_6 \quad (66)$$

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & 0 & 0 \\ A_{21} & A_{22} & A_{23} & A_{24} & 0 & 0 \\ A_{31} & A_{32} & (A_{33} + B_{11}) & (A_{34} + B_{12}) & B_{13} & B_{14} \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (67)$$

Following the same method for the remaining rows of the global stiffness matrix the complete global stiffness matrix for a beam with two elements can be found. The completed global stiffness matrix below shows which matrix values from the elements stiffness matrices combine when forming the global stiffness matrix. No matter how many elements the beam field problem contains, the global stiffness matrix will always be of the same form as the global stiffness matrix formed for a two element beam. That is, starting from left to right, the lower right quadrant of the left beam will combine with the upper left quadrant of the right beam. At the same time the element matrix values will be placed diagonally, any unknown global stiffness matrix values that are not automatically filled in by an element stiffness matrix value will be filled in with a zero. This zero is a result of the fact that some nodes in the elements are completely separate from other nodes. As an example, the beam with three nodes and two elements contained displacements D_1 to D_6 and forces R_1 to R_6 . Splitting the beam problem into its elements it is clear that the first element and the second element shared two common global displacements and forces but do not share the remaining four. In other words, some global stiffness matrix values will be zero due to the fact that some force values, such as the force values represented by the first node, will not be affected by displacement values on the third node due to the node representing those possible displacements being too disconnected from the element that contains the first node.

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & 0 & 0 \\ A_{21} & A_{22} & A_{23} & A_{24} & 0 & 0 \\ A_{31} & A_{32} & (A_{33} + B_{11}) & (A_{34} + B_{12}) & B_{13} & B_{14} \\ A_{41} & A_{42} & (A_{34} + B_{21}) & (A_{44} + B_{22}) & B_{23} & B_{24} \\ 0 & 0 & B_{31} & B_{32} & B_{33} & B_{34} \\ 0 & 0 & B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (68)$$

The total size of the global stiffness matrix can always be found by knowing how many nodes are in the system or how many elements are present in the problem. The type of components in the problem also dictates the total size of the global stiffness matrix. In this case, the main concentration will be on the beam element. Each node on the beam problem can carry two possible loadings, an applied force and a moment, and can be displaced two different ways, rotationally or laterally. Because the governing equation is a matrix multiplication problem, the size of each matrix that makes up the equation can be seen as:

$$\{n \times 1\} = [n \times n] \{n \times 1\} \quad (69)$$

$$n = TotalNodes \times 2 \quad (70)$$

$$TotalNodes = NumberOfElements + 1 \quad (71)$$

These stiffness matrices will not necessarily contain the same values from one element to another. This is because it is possible for each element that makes up the beam problem to be different lengths, be made of different materials, or to have a different flexural inertia, I . This can allow the user to analyze beams of most kind, uniform or non-uniform, as long as the material properties and shape measurements are known. At the same any number of elements can be combined into the global stiffness matrix. The only real limit in the refinement of the approximation of the beam displacements comes from the method of computation. If the computation is done by hand, then it is clear that there is only certain number of elements that can be incorporated into the global stiffness matrix before the problem becomes too big. At that point the next step is to move to computation using a computer. Here the analysis becomes easier and more streamlined as it is possible to create a program that takes the data from the user and calculates each element stiffness matrix and then computes the global stiffness matrix. Then it is only a matter of applying the given forces to find the desired displacements. At this point the

biggest limit would be the computing power of the machine that the beam problem is being analyzed on.

While the beam element type used by GBeam is an Euler-Bernoulli beam, which assumes negligible shear deformation, that does not mean that GBeam is only limited to using Euler-Bernoulli beams. With a small amount of modification GBeam can be made to analyze a beam problem that makes use of the Timoshenko beam model. While the derivation of the Timoshenko beam model will not be presented in this paper, the derivation of the stiffness matrix for this beam is presented in Przemieniecki's book *Theory of Matrix Structural Analysis* [13].

5. JavaScript Finite Element Beam Analysis

5.1 An Application in Three Parts

The finite element beam analysis application comprises of three separate windows that each displays the input spreadsheet, a results table, and solution plots respectively. Choosing to display the input data sheets and the results in three separate windows came from a decision to present GBeam in a neat and straight forward manner. Putting the input spreadsheet in the same window as the results spreadsheet and the plots would have resulted in disorderly presentation. At the same time, one of the major benefits of programming GBeam in JavaScript is its ability to allow the user to make changes to the program as they see fit. This includes allowing the user to expand the overall size of the input spreadsheet, which allows the user to input as many nodes as needed or desired. The basic premise of GBeam is shown in the flowchart below, where all data is input by the user in the first window, the input spreadsheet. All relevant data is then put into the browser local storage in order to allow multiple windows to share the beam data. The input spreadsheet window then leads into the solution output window, which gets its data from the

local storage. Finally the solution output window leads to the plot output window, which also gets its data from the browser local storage.

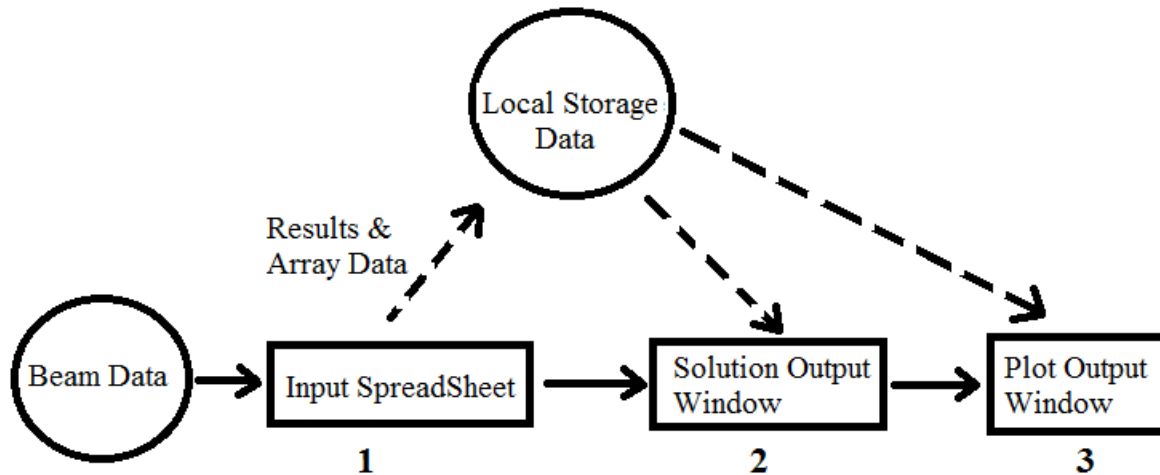


Figure 16 Application Flowchart

5.1.1 Input Spreadsheet: The Main Menu

The first page that the user sees when using GBeam is the input spreadsheet, which is where all the beam data is applied (see Figure 17 below). The main menu is comprised of a total of ten columns. Each user input column is labeled on the top of the spreadsheet and values filled into the cells of each column will not necessarily be read and processed into arrays the same way. What follows is a brief explanation of the function of each column with a more detailed explanation to follow later on in the paper.

Input Data Beam Element										
	Node Location	Boundary	Translational Spring	Torsional Spring	V (Force)	M (Moment)	Distributed Load	Inertia	Youngs Modulus	c (Distance from Neutral Axis)
1	0	c					0/..	1.8E-6	68E9	.5
2	1					200	/J..			
3	2						/J200			
4							/J..			
5							/J..			
6							/J..			
7							/J..			
8							/J..			
9							/J..			
10							/J..			
11							/J..			
12							/J..			
13							/J..			
14							/J..			
15							/J..			
16							/J..			
17							/J..			
18							/J..			
19							/J..			
20							/J..			

Figure 17 Finite Element Beam Analysis Main Menu Input Spreadsheet

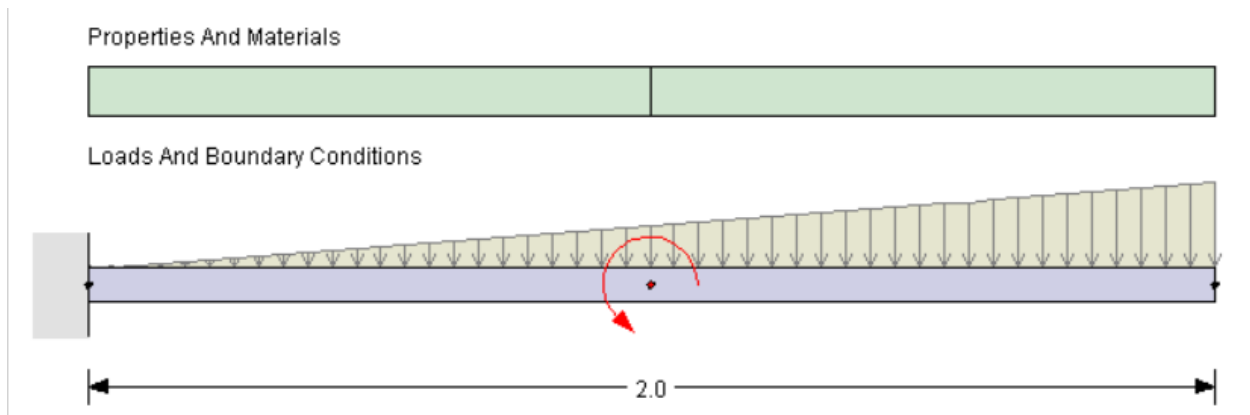


Figure 18 ETBX Provided Beam Problem Visual Display

Column 1: Node (x-location)

This column takes its input as the location of each node on the beam. The nodes that make up the beam can be any distance away from each other as long as the nodes follow a numerical order of smallest to largest, as shown in Figure 19. This column is used to find the

length of each beam element as well as find the total number of beam elements in a given problem. This allows GBeam to know how many stiffness matrices to tabulate and allows GBeam to determine what size to make the global stiffness matrix. The way the script is written, the entire column does not have to be filled in order to proceed with the analysis. Cell values can be left blank as long as there are no blank cells in between the first node cell and the last node cell.

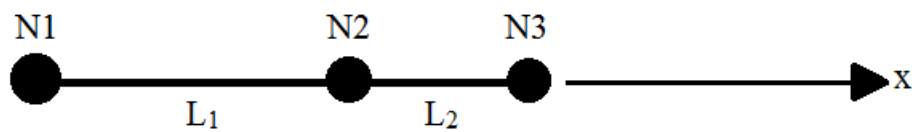


Figure 19 Beam Node Input Placement

Column 2: Boundary

This column gives the user the choice of applying one of four available boundary conditions. These boundary conditions will later be used when creating a boundary condition array that in turn will affect the final form of the beam global stiffness matrix. It is important to note that each boundary condition has an equivalent number assigned to it that is assigned to the boundary condition array. The script then reads this array and decides which rows and column of the global stiffness matrix to change based on the array location and array value of the boundary condition array. The available boundary conditions the user is allowed to apply, along with their corresponding equivalent number values, are shown in the table below.

Table 3 Application Available Boundary Conditions

Boundary Condition
fr (Free) or blank
c (Clamped)
s (Simple Support)
r (Rotation)
h (Hinge)

The free boundary condition is the default node boundary condition. In order to declare a free boundary condition, the user can leave the boundary condition cells blank. This type of boundary condition has no overall effect on the global stiffness matrix. The clamped boundary condition represents a node that is fixed laterally and rotationally. Thus this boundary condition will affect two rows and columns that correspond to the node where the boundary is applied. The support boundary condition fixes the node in the lateral direction. This boundary condition affects the row and column associated with the chosen nodes lateral movement. The rotation boundary condition fixes the node rotationally. This boundary condition affects the global stiffness matrix row and column associated with the nodes rotation.

Column 3: Translational Spring

This column allows the user to apply a translational spring support to the beam. Each spring can be applied to any user declared nodes. The column is set up so that all blank values are read as zeros, meaning no translational springs exist where a blank column cell exists. The addition of a translational spring to a declared node adds the value of the

translational spring to the global stiffness matrix. A more detailed explanation of this process is given later on.

Column 4: Torsional Spring

This column acts much like the column for adding a translational spring, except the value that represents the rotational spring is added to the diagonal term of the global stiffness matrix that corresponds to the rotation of the node it is applied to. As with the application of a translational spring, the effect of a rotational spring on the global stiffness matrix is expanded on later in this section.

Column 5 & 6: V (Force)/M (Moment)

These two columns allow the user to apply direct forces and moments to the declared nodes. As with the other previous columns, blank values are meant to signify a value of zero. Forces and moments can be positive or negative. Positive force values mean that the force is pointing upwards and negative means the force is pointing downwards. As the z-direction is set to be pointing perpendicular from the page towards the reader, a positive moment means that the moment is pointing counter-clockwise, while negative means the moment is pointing clockwise.

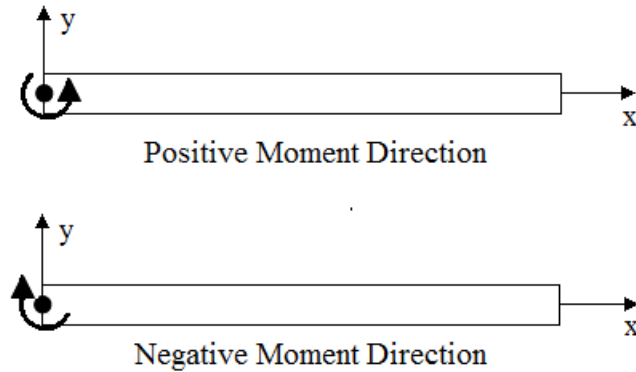


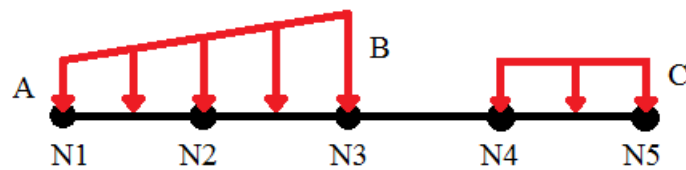
Figure 20 Beam Moment Sign Convention

Column 7: Distributed Load

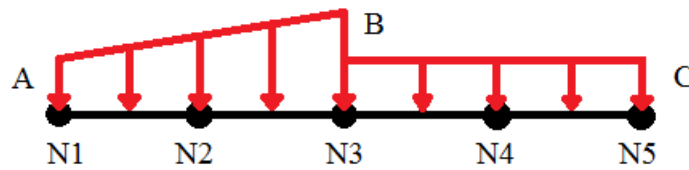
This column represents the user applied distributed load. The distributed load can be uniform or positive/negative linear. There are total of six possible entries for the distributed load column. These inputs, shown below, consist of four basic inputs and two different types of combinations of the basic inputs. The first input is the default entry of the distributed load column. The main point of this input is to serve as a blank value. The script is written to only check for the values of the beginning and end of a distributed load. The second entry represents the declaration of the start of a distributed load. This value is placed on the same row that the user wants the distributed load to begin at and can be ended anywhere from the next node to the end of the beam. The third entry represents the ending of a distributed load. This entry is only valid if there is a previous cell with an entry that represents the beginning of a distributed load. The entry represents a distributed load that runs from the node that it is declared at until the very end of the beam. This entry can be written at any declared node as long as this entry is not put between already declared distributed loads. Doing so will result in an error.

Table 4 Application Distributed Load Terminology

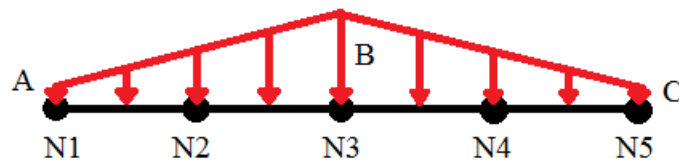
	Distributed Load Input	Meaning
1	../..	Continuation of loads or no loads existing if not placed between two cells with numerical values.
2	#/..	Declaration of the beginning of a distributed load.
3	../#	Declaration of the end of a distributed load.
4	#/#	Declaration of a distributed load that begins at the applied node and continues to the last node of the beam.
5	../#,#/..	Declaration of two separate distributed loads at a single node. First half ends the previous distributed load, while the second half is the beginning of a new distributed load immediately as the previous load ends.
6	../#,#/#	Declaration of two separate distributed loads at a single node. First half ends the previous distributed load, while the second half is the beginning of a new distributed load that continues to the last node of the beam immediately as the previous load ends.



Distributed Load Example 1



Distributed Load Example 2



Distributed Load Example 3

Figure 21 Possible User Input Distributed Loadings

The fifth and sixth entries represent the end of a distributed load followed by the immediate beginning of a new distributed load on the same node. The fifth entry allows the

user to declare a distributed load that can be ended anywhere between the node of declaration and the end of the beam. The sixth entry allows the user to create a distributed load that reaches from the node of declaration to the end of the beam.

Table 5 Distributed Load Example 1 User Input Methods

Node	Distributed Load Input 1	Distributed Load Input 2
1	A/..	A/..
2	../..	../..
3	../B	../B
4	C/C	C/..
5	../..	../C

Table 6 Distributed Load Example 2 User Input Methods

Node	Distributed Load Input 1	Distributed Load Input 2
1	A/..	A/..
2	../..	../..
3	../B,C/C	../B,C/..
4	../..	../..
5	../..	../C

Table 7 Distributed Load Example 3 User Input Methods

Node	Distributed Load Input 1	Distributed Load Input 2
1	A/..	A/..
2	../..	../..
3	../B,B/..	../B,B/C
4	../..	../..
5	../C	../..

Column 8,9, & 10: Inertia, Young's Modulus, Distance from Neutral Axis

These columns represent the moment of inertia, Young's modulus, and distance from the neutral axis, c , of each beam element created by two adjacent nodes. Because these columns represent the material and geometrical values of a beam element and not a node, the

maximum number of cells the user will ever need to fill in is one less than the number of declared nodes. Additionally there is no need for the user to fill in each node of the column. For these columns a minimum of one numerical value is needed to prevent an error output. Starting from the first row, the script creates an array that holds the values of the moment of inertia, Young's modulus, or distance c from the neutral axis. When reading the input values, the script will fill in any blank values with the last numerical value in the column. This allows the user to save time when creating a beam that is uniform throughout or only partially uniform.

5.1.2 Solution Output Window

This window appears immediately after the user inputs all of their data and clicks the run button on the main menu window (Figure 22). This window only outputs the lateral displacement and rotation at each declared node. To ensure accuracy, the script adds an additional eight nodes between each user defined nodes to each element, thus placing a total of ten nodes per segment. This is done in order to refine the displacement outputs and the plots given by the script. While the solution window only displays the displacement/rotation values of the declared nodes, it is possible to view the overall solution for each declared and undeclared node, since those values are stored in the local storage of the web browser. Additionally this page also presents the user with the value and node location of the largest lateral and slope displacement.

Plot

Max Lateral Displacement	-1.356e-1
Max Lateral Displacement Location	3.000e+0
Max Slope Displacement	6.790e-2
Max Slope Displacement Location	4.000e+0

Input Data Beam Solution

Node	Node Location	Lateral Displacement	Slope (Radians)
1	0.000e+0	0.000e+0	-4.408e-2
2	1.000e+0	-4.429e-2	-4.465e-2
3	2.000e+0	-8.958e-2	-4.588e-2
4	3.000e+0	-1.356e-1	L:-4.604e-2/R:6.776e-2
5	4.000e+0	-6.775e-2	6.790e-2
6	5.000e+0	0.000e+0	6.735e-2
7	6.000e+0	6.698e-2	6.685e-2
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

Figure 22 JavaScript Application Output Display Table Example

5.1.3 Plot Output Window

This is the final window in the finite element beam analysis application (Figure 17 and various similar Figures in the Appendix). This window graphically displays a total of four separate plots: a beam lateral displacement plot, a beam slope plot, a shear diagram, and a moment diagram. The first two plots are very simple. The plots are simply the displacement of the beam with respect to the location of each node. The first two plots make use of the solution data for every node, declared and undeclared. The last two plots receive their data from the reaction forces of the beam that are found from the displacement solutions. These reactions forces are then used to find the change in shear and moment throughout the beam.

5.2 Sorting Data Into Arrays

All user input data is sorted into arrays in order to form the governing equation

$\{R\} = [K]\{D\}$. The size of the seven data arrays created by the main menu input spreadsheet of the JavaScript application are $1 \times n$ array where n is equal to the total number of declared nodes. The last two arrays that represent the physical properties of the beam will each be a $1 \times (n-1)$ data array. Additionally a third $1 \times (n-1)$ array will be created that represents the length of each beam element. In JavaScript every array index is zero based, meaning that the index of each array starts with zero. This simply means that if the user input three nodes into the main menu spreadsheet, the values of the first, second, and third element will be written as:

$$node = [node1, node2, node3] \quad (87)$$

$$node[0] = node1$$

$$node[1] = node2 \quad (88)$$

$$node[2] = node3$$

The zero based index, while simple to understand, is very important to remember when working with multiple nodes and elements. This user input data will be used to create the multidimensional array which represents the beam stiffness matrix. It is important to state that creating a multidimensional array in JavaScript involves declaring an array inside an array. All arrays in JavaScript are one dimensional. While it is possible for to create a JavaScript equivalent of a multidimensional array, it is important that the programmer to pay attention when creating the multidimensional array in order to keep all rows consistent. Regular one dimensional arrays can be read as:

$$x = [col1, col2, col3] \quad (89)$$

In the example above each individual element represents an individual column of a one dimensional array. Creating a multidimensional array is simply a matter of applying a series of arrays inside of an array.

$$x = \left[\underbrace{[col1, col2, col3]}_{row1}, \underbrace{[col1, col2, col3]}_{row2}, \underbrace{[col1, col2, col3]}_{row3} \right] \quad (90)$$

When creating a multidimensional array, each inner array represents a row of the new multidimensional array. Each value of the inner array represents a column in the multidimensional array. If the JavaScript programmer must be careful when manually declaring a multidimensional array, since it is possible to create an inner array whose size is not equal to the size of the other arrays. If this is the case, no error will be reported unless the user tries to call an array element that is out of bounds of the array index. Calling the individual value of a multidimensional array is similar to the calling an element from a one dimensional array, with all array indexes starting with zero.

$$x = [[r1c1, r1c2, r1c3], [r2c1, r2c2, r2c3], [r3c1, r3c2, r3c3]] \quad (91)$$

$$x[rowNumber][columnNumber] = ArrayElement \quad (92)$$

All calculations done by the script is done by utilizing the information gathered from the nine columns in the main menu. Each cell in the main menu has a unique individual id. All information entered into each cell is associated with the cell that information is input into. The total size of the each array created from the individual column is dependent on the total number of declared nodes. For the creation of arrays that represent the information from columns 2 to 6 a simple **for loop** is used to read the values from each cell in the columns by calling an individual

id and assigning the value tied to that id to an appropriate array. In the case of the second, fifth and sixth columns (Boundary, Force, and Moment respectively), additional instructions to the script must be given in order to allow and compensate for unfilled cells.

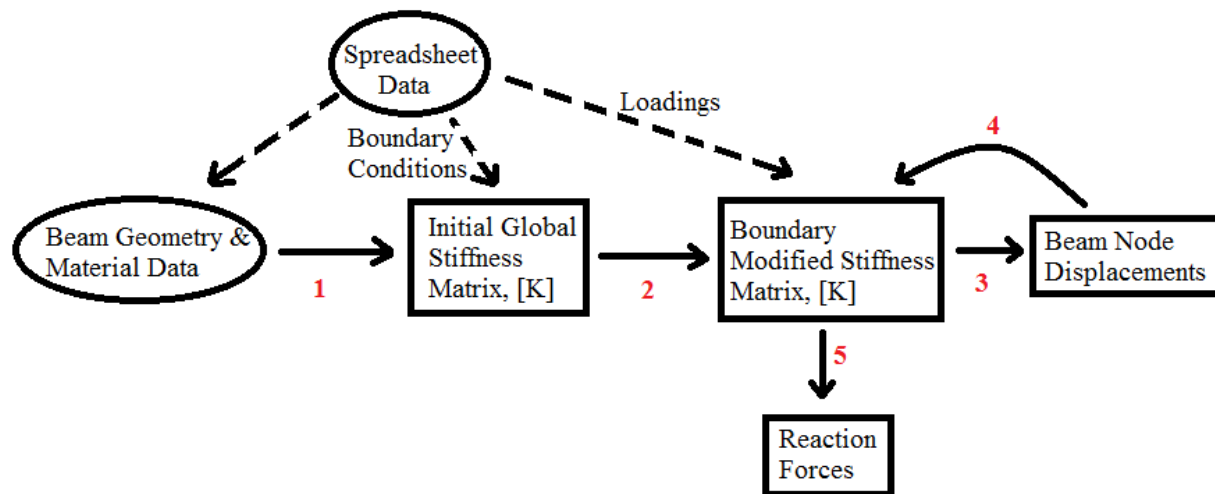


Figure 23 GBeam Calculation Flowchart

For the Boundary column an array (named Boundaries) is created which is assigned the values found in the cells of the Boundary column. Any blank values that are found in the cells that are being cycles through by the **for loop** are assigned to the array Boundaries as a **NaN**, read as Not A Number. As defined by Flanagan's , *JavaScript the Definitive Guide* [14], The **NaN** output is a special JavaScript value that is returned when a mathematical operation yields an undefined or error result. It is important to note that the **NaN** value behaves unusually in that it does not compare equal to any other number, including itself. Later on in the script the array Boundaries is cycles through and the string characters are read and numerical values are assigned to array BC. A new boundary condition array is created in order to account for the fact that eight undeclared nodes are added in between each declared node. This has the effect of increasing the overall length of each array, which is why secondary arrays must be created in order not to

interfere with the data collection from the main menu. The numerical values associated with each boundary condition are mentioned in section 5.1.1. These values are assigned to the array BC, where any blank cell values are automatically assigned a value of zero in order to represent a free node. All undeclared nodes added to the beam also have a value of zero assigned to the array BC to signify that those undeclared nodes are free nodes.

Creating arrays that represent the values input for area moment of inertia and Young's modulus means compensating for the option of letting cell values remain blank in order to make creating uniform beams easier. As stated earlier, the total number of beam elements present in a problem will be equal to one less than the total number of nodes, which would mean that when entering the beam data into the main menu, the total number of cells the user would only have enter is one minus the total number of nodes. The script creates the arrays that represent the inertia and the Young's modulus by cycling through the columns from the first row to the row before where final node has been placed. Starting from the second row if that row happens to be blank/undefined the script assigns it the value of the previous row, which it knows to contain a numerical value. A visual example is shown below, which demonstrates how the script would create the arrays for either column 8 or 9 depending on user input.

Table 8 Moment of Inertia/Young's Modulus Array Data Sorting

Beam Element	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Input	Array	Input	Array	Input	Array	Input	Array
1	A	A	A	A	A	A	A	A
2		A		A	B	B	B	B
3		A	B	B		B	C	C

5.3 Creation of Element & Global Stiffness Matrices

5.3.1 Stiffness Matrix Creation

As explained in section 4.3 creating a stiffness matrix for a beam element requires knowledge of the beam length, beam area moment of inertia, and the beam Young's modulus. All information needed to create the element stiffness matrix is acquired from the main menu of GBeam, with the length of each beam found by using the distance between two nodes as the length of each beam element. In this application a function is created which takes the necessary data and creates a multidimensional array that contains element stiffness matrices for the entire beam. Because the formation of the element stiffness matrix involves using a function, that function can be called at any point in the script as long as the function is fed the correct information. A sample of the code used to create the element stiffness matrix is shown below. This image details the calling of the function needed to create the element stiffness array and the contents of the function. The final form of the element stiffness matrix is also shown below. This form is similar to the form of a multidimensional array shown in the previous section.

$$[K_e] = \left[[K_{e_1}], [K_{e_2}] \dots [K_{e_n}] \right] \quad (93)$$

$$[K_{e_n}] = \left[[K_{11}, K_{12}, K_{13}, K_{14}], [K_{21}, K_{22}, K_{23}, K_{24}], [K_{31}, K_{32}, K_{33}, K_{34}], [K_{41}, K_{42}, K_{43}, K_{44}] \right] \quad (94)$$

```

//This for loop calls the element stiffness function for every element created b
for(i=0;i<InterElemNum;i++)
{
    Ke[i]=Element_Stiffness(InterLength[i],InterYoungs[i],InterInertia[i])
}
//Element Stiffness Function
//This function creates element stiffness matrices for each respective element.
function Element_Stiffness(L,E,I) //Element_Stiffness takes element length,young
{
    var Ka,Kb,Kc,Kd //Ka,Kb,Kc,Kd form the element stiffness matrix for a beam
    Ka=(12*E*I)/(Math.pow(L,3))
    Kb=(6*E*I)/(Math.pow(L,2))
    Kc=(4*E*I)/L
    Kd=(2*E*I)/L
    //Function returns element stiffness matrix.
    return [ [Ka,Kb,-Ka,Kb],
             [Kb,Kc,-Kb,Kd],
             [-Ka,-Kb,Ka,-Kb],
             [Kb,Kd,-Kb,Kc] ]
}

```

Figure 24 Element Stiffness Matrix Formation Code

The process for creating the global stiffness matrix is detailed in section 4.4, with the final product showing that elements one element stiffness matrix will superimpose with elements from the stiffness matrix in the adjacent beam element. Creating the global stiffness matrix in the script is simply a matter of using multiple **for loops** in order to cycle through the element stiffness array and add those values to a new multidimensional array that represents the global stiffness matrix. Because it is possible to create global stiffness matrices that contain multiple zero values in both the lower and upper triangular section, it is best to initially create a multidimensional array containing all zeros that will represent the global stiffness matrix. It is possible to create a global stiffness matrix with initial elements of zero since it is known that the total size of the global stiffness matrix is a function of the total number of nodes present in the beam problem. From there, the process involves replacing the appropriate zero elements with corresponding element stiffness values.

This process involves a total of three **for loops** and three **if statements**. The three **for loops** are used to cycle through each individual stiffness matrix, the matrix row, and the matrix

column while the **if statements** ensure that the indexes of the element stiffness matrices are not exceeded when applying the values of the element stiffness matrices to the global stiffness matrix. For any global stiffness matrix, the superimposed element stiffness matrixes that make up the global stiffness matrix will look like Figure 25. The general pattern found is that the element stiffness matrices will always superimpose starting at the following global stiffness matrix index values $K_G[2][2]$, $K_G[2][3]$, $K_G[3][2]$, and $K_G[3][3]$, with more possible superimpositions occurring every two increments in both the row and column indexes.

Index:	0	1	2	3	4	5
0	A_{11}	A_{12}	A_{13}	A_{14}	0	0
1	A_{21}	A_{22}	A_{23}	A_{24}	0	0
2	A_{31}	A_{32}	$(A_{33} + B_{11})$	$(A_{34} + B_{12})$	B_{13}	B_{14}
3	A_{41}	A_{42}	$(A_{34} + B_{21})$	$(A_{44} + B_{22})$	B_{23}	B_{24}
4	0	0	B_{31}	B_{32}	B_{33}	B_{34}
5	0	0	B_{41}	B_{42}	B_{43}	B_{44}

Figure 25 Global Stiffness Matrix Row and Column Index Values

The code shown below details how the global stiffness matrix, initially filled with zeros, is given the correct element stiffness matrix elements. The code segment takes each individual element stiffness matrix and goes through each row and column in that stiffness matrix. Once the code has gone through the first stiffness matrix, the code moves on to the next element stiffness matrix in the array and the element stiffness matrix row/column counter resets and the row/column counter for the global stiffness matrix increase by an increment of two in order to add the correct elements to the global stiffness matrices again.

```

//The code below replaces appropriate values in Kg_zeros in order to create a complete global stiffness matrix
var e=0;//row value of global stiffness matrix
var f=0;//column value of global stiffness matrix
var m=0;//array location value of element stiffness matrix
for(n=0;n<InterElemNum;n++)
{
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            Kg_zeros[e][f]=Kg_zeros[e][f]+Ke[m][i][j];
            Kg_zeros_Reaction[e][f]=Kg_zeros_Reaction[e][f]+Ke[m][i][j];
            f++;
            if(j==3)
            {
                f=0;
                if(m>0)
                {
                    f=2*m;
                }
            }
            e++;
            if(i==3)
            {
                if(m>0)
                {
                    e=2*m;
                }
            }
        }
        m+=1;
        e=m+2;
        f=m+2;
        if(m>0)
        {
            e=2*m;
            f=2*m;
        }
    }
}

```

Figure 26 Global Stiffness Matrix Formation Code

5.3.2 Effects of Boundary Conditions On Global Stiffness Matrix

In all boundary conditions, all stiffness matrices were modified in such a way so that symmetry would be conserved when applying a known displacement constant (in this case zero) to the element stiffness matrix. Applying this known displacement to the stiffness matrix allows the stiffness matrix to be rewritten to allow the program to conduct an analysis with a possible mixture of known and unknown values at the bounded node(s). Initially each global stiffness matrix will be a complete global stiffness matrix that will need to be modified in accordance with the kinds of boundary conditions applied to the declared nodes. Each modification to the global stiffness matrix will ensure that the displacement of the bounded node follows the behavior that the user wants, while preventing the global stiffness matrix from becoming singular, which

would prevent the displacement of the beam from being found. Each boundary condition will affect the global stiffness in slightly different ways. All changes to the global stiffness matrix will affect the row and columns associated with the displacement that the user is setting to zero with the application of a boundary condition.

As Cook shows in his example [9], for any arbitrary stiffness matrix analysis there will be a mixture of known and unknown force and displacement values. If any one of the displacements were to be set to a known value, the equations could be rewritten to account for the addition of a known displacement. In this example R_1 , R_2 , and D_2 are known with the rest of the displacement/force values unknown.

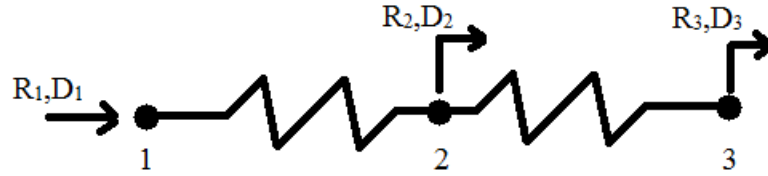


Figure 27 Arbitrary Finite Element Analysis Problem Set Up

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} R_1 \\ R_2 \\ R_3 \end{Bmatrix} \quad (95)$$

$$D_2 = \Delta_2 \quad (96)$$

In its new form the stiffness matrix will lack the symmetry that it originally contained. In order to keep symmetry and remain non-singular, the entirety of the second row of the stiffness matrix can be replaced by the value of $D_2 = \Delta_2$ where Δ_2 can be found in terms of Equation 99 below.

$$\begin{bmatrix} K_{11} & 0 & K_{13} \\ K_{21} & 0 & K_{23} \\ K_{31} & 0 & K_{33} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} R_1 - K_{12}\Delta_2 \\ R_2 - K_{22}\Delta_2 \\ R_3 - K_{32}\Delta_2 \end{Bmatrix} \quad (97)$$

$$K_{21}D_1 + K_{23}D_3 = R_2 - K_{22}\Delta_2 \quad (98)$$

$$\Delta_2 = \frac{-(K_{21}D_1 + K_{23}D_3) + R_2}{K_{22}} \quad (99)$$

Because of Equation 96, the stiffness matrix can reach the final form shown below.

Additionally, due to the fact that the displacement Δ_2 is to be set to zero for this analysis application, then all values of Δ_2 will be crossed out, leaving the form of the governing equation that will be utilized in this application when the boundary conditions are applied.

$$\begin{bmatrix} K_{11} & 0 & K_{13} \\ 0 & 1 & 0 \\ K_{31} & 0 & K_{33} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} R_1 - K_{12}\Delta_2 \\ \Delta_2 \\ R_3 - K_{32}\Delta_2 \end{Bmatrix} \quad (100)$$

$$\begin{bmatrix} K_{11} & 0 & K_{13} \\ 0 & 1 & 0 \\ K_{31} & 0 & K_{33} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} R_1 \\ 0 \\ R_3 \end{Bmatrix} \quad (101)$$

The formation of the boundary conditions detailed below will use the method above applied to the free body diagram shown below in order to detail the general changes that occur when applying a boundary condition to a two element beam.

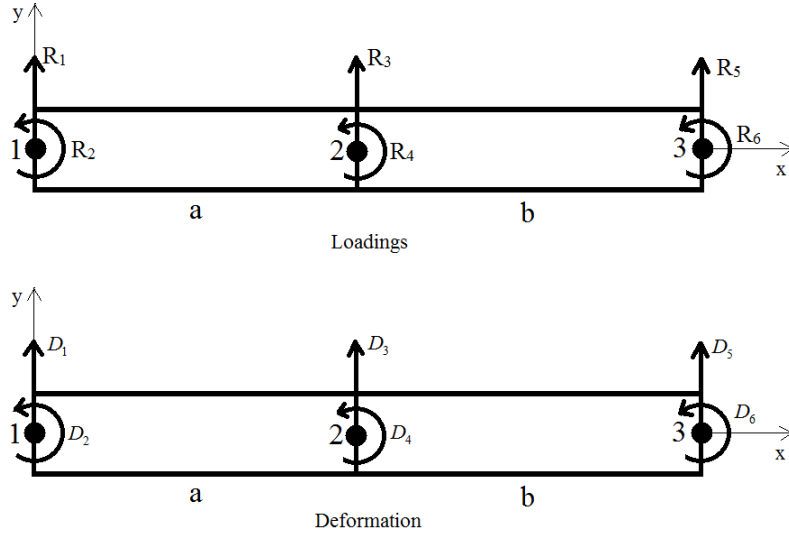


Figure 28 Global Beam Nodal Loading and Deformation Diagram

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & 0 & 0 \\ K_{21} & K_{22} & K_{23} & K_{24} & 0 & 0 \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (102)$$

Adjusting the rows and column associated with the nodal boundary conditions to be set by the user is ultimately easier to implement in GBeam than it is to reduce the size of the stiffness matrix in accordance to the user applied boundary conditions. As detailed below, each of the boundary conditions will end up affecting the stiffness matrix in a specific way. Additionally, these changes to the stiffness matrix are patterned after the specific node that the boundary conditions are applied to. Overall, no matter what specific shape the stiffness matrix takes after the boundary conditions are applied, the size of the matrix will always be the same. It is easier to implement the logic that will adjust certain rows and columns in the stiffness matrix than it is to reduce the size of the stiffness matrix, especially as the stiffness matrix could possibly be a different size each time the analysis is run.

Clamped Boundary Condition

The clamped boundary condition keeps the node from displacing laterally or radially. Any time a node is assigned a clamped boundary condition the global stiffness matrix will be subject to a change originating from the diagonal term directly associated with the displacement value being bounded. The major changes to the global stiffness matrix are the transformation of bounded diagonal elements to value of one, and a change of the rows and columns connected to the diagonals to zero. As an example, if in Figure 28 the first node is assigned the clamped boundary condition, then it is apparent that values D_1 and D_2 will always be zero. This means that in order to ensure that the matrix is invertible and will give the correct values, the first two rows and columns must be changed to reflect the boundary condition applied. Diagonal values K_{11} and K_{12} will change to one in order to keep the matrix invertible.

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & 0 & 0 \\ K_{21} & K_{22} & K_{23} & K_{24} & 0 & 0 \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (103)$$

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & K_{33} & K_{34} & K_{35} & K_{36} \\ 0 & 0 & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (104)$$

Support Boundary Condition

The support boundary condition keeps the node where the support is added from moving laterally while allowing the beam to change in rotation. If the support boundary condition is applied to the first node in Figure 28, then only the first row and column of the global stiffness matrix will be affected. This is due to the fact that, from the free body diagram, the force and displacement variable associated with lateral displacement is R_1 and D_1 respectively. The process for rewriting the global stiffness matrix is the same process used for the clamped boundary condition.

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & 0 & 0 \\ K_{21} & K_{22} & K_{23} & K_{24} & 0 & 0 \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (105)$$

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{22} & K_{23} & K_{24} & 0 & 0 \\ 0 & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\ 0 & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (106)$$

Rotation Boundary Condition

The rotation boundary condition keeps the node where the boundary is applied from moving rotationally. Like the support beam, if the rotation boundary condition is applied to the first node of Figure 28, then the row/column associated with the force and displacement, R_2 and D_2 , will need to be rewritten.

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & 0 & 0 \\ K_{21} & K_{22} & K_{23} & K_{24} & 0 & 0 \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} & K_{36} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (107)$$

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{Bmatrix} = \begin{bmatrix} K_{11} & 0 & K_{13} & K_{14} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ K_{31} & 0 & K_{33} & K_{34} & K_{35} & K_{36} \\ K_{41} & 0 & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{53} & K_{54} & K_{55} & K_{56} \\ 0 & 0 & K_{63} & K_{64} & K_{65} & K_{66} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \quad (108)$$

Hinge Boundary Condition

Applying a hinge boundary condition to the beam element and applying the change to the global stiffness matrix is a more involved process than the changes involved with applying a clamped, support, or rotation boundary condition [15]. To start, take a normal fixed-fixed beam with no hinge present. If a force were applied to the center of the beam, a beam displacement would occur. In this case, the beam displacements could be found using the general global beam equation.

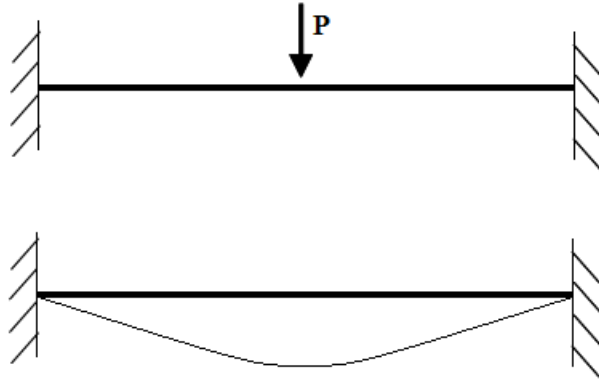


Figure 29 Beam Deformation with no Hinge Present

However, if the beam were to include a hinge, the beam problem would undergo significant changes. Taking the same beam problem shown in Figure 29, a hinge is added to the problem. If a force is applied at the location of the hinge, the beam deformation would look very different from the beam deformation on a beam with no hinge. The big difference between the two types of beams comes in the number of independent variables located at each node present in the beam. In a normal beam, wherever a node is placed, there will only ever be two independent variables; a lateral deflection and a slope. When a node is placed in at a hinge however, there will be three independent variables instead of two. The variables will be one lateral deflection and two slopes.

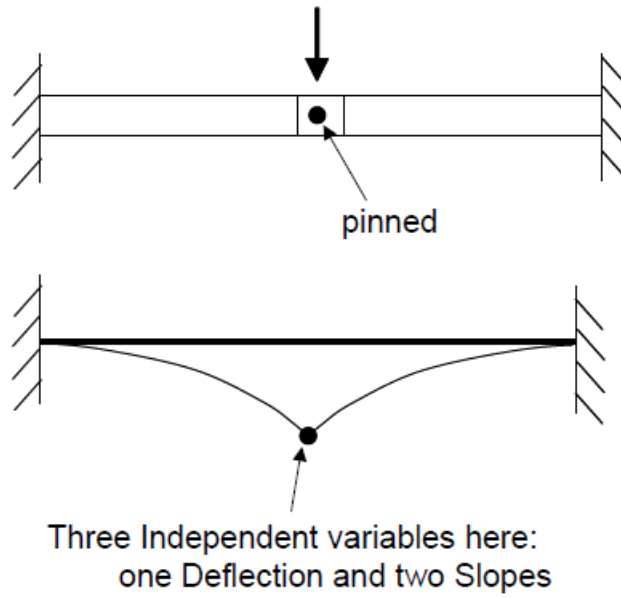


Figure 30 Beam Deformation with Hinge Present [16]

The change in the number of independent variables comes from the fact that although shear is transferred through the hinge on the beam, the moment is not. The ability to transfer moments through a node is what allows the beam to avoid discontinuities. Solving any beam problem with a hinge involves using a method known as condensation in order to account for the fact that moments do not transfer through a hinge. Using condensation to account for the release of the moment from the global stiffness matrix is known as moment end release.

Applying the hinge boundary condition involves condensing the global stiffness matrix. Condensation can simply be described as the combination of algebraic equations. This combination is done in order to reduce the number of unknown variables in a system of equations. A very simple example can be shown below, where it is possible to reduce the two equations into one equation-one unknown.

$$ax + by = e \quad (109)$$

$$fx + gy = h \Rightarrow y = \frac{(h - fx)}{g} \quad (110)$$

$$ax + b \left(\frac{(h - fx)}{g} \right) = e \quad (111)$$

This same process can be used to condense the global stiffness matrix. In a beam element problem, whenever a hinge is introduced to the beam, two individual beam elements share the node where the beam lies. Each element will have their own individual element stiffness matrix that will be combined with the rest of the element stiffness matrixes to form the global stiffness matrix. In order to make sure that the global stiffness matrix accounts for the addition of a hinge boundary condition, one of the two beam elements that share the hinge will need to be subject to a moment end release. End releasing both of the elements that share the hinge causes the moment to be completely released from the beam problem, which causes problems in the analysis. Thus only one element will be subject to condensation.

For this analysis whenever a hinge was introduced to the beam problem, the element with the hinge located at the left hand side became subject to moment end release. Because the left hand side contains the hinge, the moment M_1 will be condensed out of the element stiffness matrix.

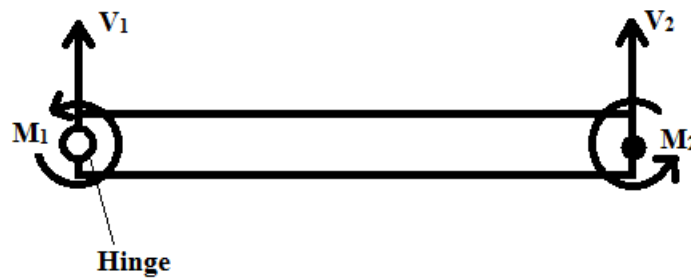


Figure 31 Beam Element w/Hinge Located Left Hand Side

Matrices that will be subject to condensing will be split into two categories, the elements to be retained in the upper part of the matrix and the elements to be condensed in the bottom. The matrix of a beam element with a hinge on the left hand side will be rearranged in order to have the moment M_1 , the value to be condensed, moved to the lower portion of the matrix.

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix} = \begin{bmatrix} K_a & K_b & -K_a & K_b \\ K_b & K_c & -K_b & K_d \\ -K_a & -K_b & K_a & -K_b \\ K_b & K_d & -K_b & K_c \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (112)$$

$$\begin{Bmatrix} V_1 \\ V_2 \\ M_2 \\ M_1 \end{Bmatrix} = \begin{bmatrix} K_a & -K_a & K_b & K_b \\ -K_a & K_a & -K_b & -K_b \\ K_b & -K_b & K_c & K_d \\ \hline K_b & -K_b & K_d & K_c \end{bmatrix} \begin{Bmatrix} v_1 \\ v_2 \\ \theta_2 \\ \theta_1 \end{Bmatrix} \quad (113)$$

Where,

$$K_a = \frac{12EI}{L^3}; K_b = \frac{6EI}{L^2}; K_c = \frac{4EI}{L}; K_d = \frac{2EI}{L} \quad (114)$$

With the element stiffness matrix rearranged, the matrix can be broken into four different sections in order to turn the 4x4 matrix (Equation 113) into an equivalent 2x2 matrix (Equation 115). The four sections of the stiffness matrix shown above will each be renamed going clockwise starting from the upper left quadrant as K_{11} , K_{12} , K_{22} , and K_{21} .

$$\begin{Bmatrix} R_1 \\ R_2 \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \end{Bmatrix} \quad (115)$$

$$[K_{11}] = \begin{bmatrix} K_a & -K_a & K_b \\ -K_a & K_a & -K_b \\ K_b & -K_b & K_c \end{bmatrix}; [K_{12}] = \begin{bmatrix} K_b \\ -K_b \\ K_d \end{bmatrix}; [K_{21}] = [K_b \quad -K_b \quad K_d]; [K_{22}] = [K_c] \quad (116)$$

$$[R_1] = \begin{bmatrix} V_1 \\ V_2 \\ M_2 \end{bmatrix}; [R_2] = [M_1]; [D_1] = \begin{bmatrix} v_1 \\ v_2 \\ \theta_2 \end{bmatrix}; [D_2] = [\theta_1] \quad (117)$$

Using basic matrix algebra, it is apparent from the new $\{R\} = [K]\{D\}$ equation that:

$$\begin{aligned} [R_1] &= [K_{11}][D_1] + [K_{12}][D_2] \\ [R_2] &= [K_{21}][D_1] + [K_{22}][D_2] \end{aligned} \quad (118)$$

Because no moment is transferred at a hinge it can also be assumed that $R_2 = M_1 = 0$

$$[D_2] = [K_{22}]^{-1} \langle [R_2] - [K_{21}][D_1] \rangle \quad (119)$$

$$R_1 = [K_{11}][D_1] + [K_{12}]([K_{22}]^{-1} \langle [R_2] - [K_{21}][D_1] \rangle) \quad (120)$$

$$[R_1] = [K_{11}][D_1] - [K_{12}][K_{22}]^{-1}[K_{21}][D_1] \quad (121)$$

$$\begin{bmatrix} V_1 \\ V_2 \\ M_2 \end{bmatrix} = \langle [K_{11}] - [K_{12}][K_{22}]^{-1}[K_{21}] \rangle \begin{bmatrix} v_1 \\ v_2 \\ \theta_2 \end{bmatrix} \quad (122)$$

$$[R_2] = 0 = [K_{21}][D_1] + [K_{22}][D_2] \quad (123)$$

$$[D_2] = -[K_{22}]^{-1}[K_{21}][D_1] \quad (124)$$

$$[\theta_1] = -[K_{22}]^{-1}[K_{21}] \begin{bmatrix} v_1 \\ v_2 \\ \theta_2 \end{bmatrix} \quad (125)$$

Expanding the R_1 equation out completely gives the new element stiffness matrix with

the slope/moment term for the left hand side of the beam element condensed out. However,

because the element stiffness matrix must be a square 4x4 matrix in order to correctly conduct

the beam analysis, the row and column associated with the slope on the left hand side of the element that contains the hinge will all be replaced with zeros.

$$\begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix} = \begin{bmatrix} K_a - K_c^{-1}K_b^2 & 0 & -K_a + K_c^{-1}K_b^2 & K_b - K_c^{-1}K_dK_b \\ 0 & 0 & 0 & 0 \\ -K_a + K_c^{-1}K_b^2 & 0 & K_a - K_c^{-1}K_b^2 & -K_b + K_c^{-1}K_dK_b \\ K_b - K_c^{-1}K_dK_b & 0 & -K_b + K_c^{-1}K_dK_b & K_c - K_c^{-1}K_d^2 \end{bmatrix} \begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (126)$$

With the new condensed form of the element stiffness matrix found, it is then possible to replace all the appropriate element stiffness matrices and find the displacements of the beam when one or more hinges are present. The beam hinge analysis will not calculate the two different slopes present at the hinge. Instead the slopes will need to be found separately using newly found displacements and the element stiffness matrix elements (Equations 127 and 128).

Both equations to find the slopes at the hinge can be obtained when conducting moment end release on the beam element(s) that contain a hinge. Each of the two slopes are found when releasing the moment on the element that either is assumed to contain the element on the left hand side of the beam or when the element contains a hinge on the right hand side of the beam.

As stated before, when a hinge is present two element beams will contain a hinge, one on the left hand side and one on the right. But it is important to remember that while end releasing

(condensing) the moment out of the element stiffness matrix will allow the hinged beam to be analyzed; conducting the end release on both beam elements will ultimately cause problems in the beam analysis. Thus, condensation will need to be done on only one of the two beams that share a hinge.

$$[\theta_L] = -[K_c]^{-1} \begin{bmatrix} K_b & -K_b & K_d \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \theta_2 \end{bmatrix} \quad (127)$$

$$[\theta_R] = [K_c]^{-1} \begin{bmatrix} K_b & -K_d & K_b \end{bmatrix} \begin{bmatrix} v_1 \\ \theta_1 \\ v_2 \end{bmatrix} \quad (128)$$

5.4 Spring Support

There are two types of springs that can be applied to each declared node of the global beam, a spring that affects the lateral displacement of a node and a torsional spring that affects the rotational displacement of a node. In this application both spring types can be applied to the same node simultaneously. Each spring will affect the lateral or rotational displacement of the beam and the change in displacement will be reflected by the addition of the spring terms to global stiffness matrix of the beam. All spring constants will be applied to the appropriate diagonal term of the global stiffness matrix. This allows GBeam to estimate the displacement beam with spring elements added to the beam. Extension spring constants will be added to the diagonal stiffness matrix elements associated with lateral displacement, while torsion spring constants will be added to the diagonal stiffness matrix elements associated with rotational displacement.

5.5 Applied Forces and Moments

One of the main outputs of the finite element beam analysis application is the displacement of the beam when forces are applied to it. This means that in the $\{R\} = [K]\{D\}$ equation, the displacements are the unknowns and the forces, $\{R\}$, are known values. As explained in earlier sections, each beam element contains two nodes and each node can have a force and moment applied to it. The main menu of GBeam gives provides three different ways to apply forces and moments to the beam. Columns five and six of the main menu allow the user to apply forces and moments directly to the declared nodes in the global beam. The values found in each respective column will be placed into an array that represent applied forces or an array that represents the applied moments. The size of these one dimensional arrays is equal to the total number of nodes present in the beam problem, which included nodes declared by the user and undeclared nodes added in order to refine the outputs of GBeam. Any cells left blank in columns five or six will automatically be assigned a value of zero when forming the force and moment arrays, this also holds true for any array indexes that represent the undeclared nodes added to the global beam. Finally these two arrays will be combined into a single array that will represent the matrix $\{R\}$ in the $\{R\} = [K]\{D\}$ equation. The matrix $\{R\}$, a combination of the force and moment array, will be twice the size of either one since both the force and moment array are the same size. The form of the matrix $\{R\}$ will be the following:

$$\{R\} = \begin{Bmatrix} F_1 \\ M_1 \\ F_2 \\ M_2 \\ F_n \\ M_n \end{Bmatrix} \quad (129)$$

5.5.1 Application of Distributed Forces

Distributed loads applied to the beam must be rewritten into equivalent forces and moments located at the ends of each beam that the distributed load covers. Additionally, analyzing a distributed load in this application requires interpreting the distributed load input and finding the equivalent force/moment on each beam element for each possible input. In total there are a total of three different types of distributed loadings that can be input into GBeam: uniform distributed load, positive slope distributed load, and negative slope distributed load. As shown below, these distributed loadings can be combined when applying a distributed load to a beam. When conducting the analysis, the forces will be split into the possible combination of three basic shapes.

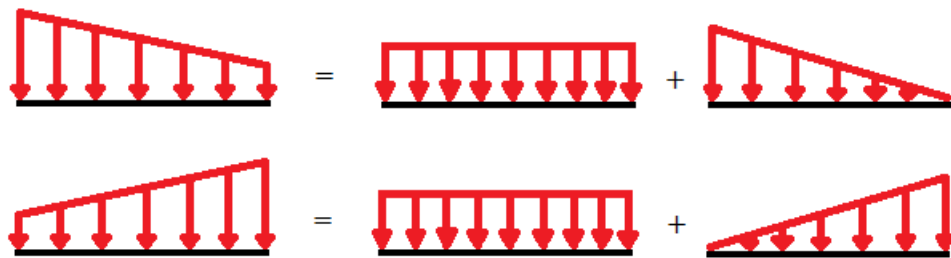


Figure 32 Distributed Loading Equivalent Form

As mentioned in Section 5.1.1 there are six possible acceptable inputs for the distributed load column. When running the analysis, the data from the distributed load column is arranged in a one dimensional array named “Distributed” whose length is equal to the number of declared nodes present in the beam. From here, the string elements located at each array index are further split by using the JavaScript method `split()`. This split method works by taking in a string input and splitting the elements into a one dimensional array with multiple elements based on the presence of a specific user defined character. The main character used to split the elements of the distributed load array is the “/”, while a secondary split is used to with the comma character “,”

in order to allow GBeam to know when a new distributed load is beginning at the same node where a previous distributed load ends.

Taking distributed loading example 1 from Figure 21, assume a user applies a distributed load for a five node beam, which can take the form of Figure 33 below:

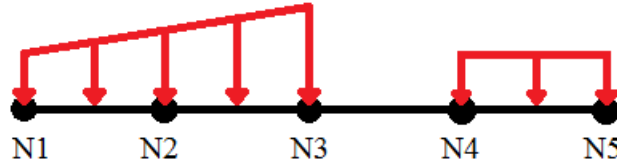


Figure 33 Possible User Input Distributed Loading

If the equivalent distributed load array were to be split using the split method, the new form of the distributed load array would have the form of:

$$Distributed = [\# / .. \quad .. / .. \quad .. / \# \quad \# / \# \quad .. / ..] \quad (130)$$

$$Distributed = [[\# \quad ..] \quad [.. \quad ..] \quad [.. \quad \#] \quad [\# \quad \#] \quad [.. \quad ..]] \quad (131)$$

Next the zero index of each inner array will all be put into one array known as StartNode while the next index of each inner array will be placed in an array known as EndNode. Not coincidentally, the indexes of the StartNode and EndNode distributed load arrays will match up with the indexes of the node location array.

$$StartNode = [\# \quad .. \quad .. \quad \# \quad ..] \quad (132)$$

$$EndNode = [.. \quad .. \quad \# \quad \# \quad ..] \quad (133)$$

$$Nodes = [N1 \quad N2 \quad N3 \quad N4 \quad N5] \quad (134)$$

By splitting the distributed load data into separate arrays which represent the start and end of a distributed load, it becomes possible to check whether each input force has a designated end location and to compute the equivalent force and moments at each beam element the distributed load covers. This is achieved by using arrays *StartNode* and *EndNode* to create arrays made specifically to check that each distributed load has both a beginning and end and to compare the force values at the beginning/end to see whether the distributed load is a uniform load or if it's sloped. These arrays, known as *StartNodeChk* and *EndNodeChk*, are simply the arrays *StartNode*/*EndNode* except replaced with ones and zeros. Zeros represent the element “..” located in both *StartNode* and *EndNode*, while ones represent a numerical element present in both arrays.

$$\begin{aligned} StartNode &= [\# \quad .. \quad .. \quad \# \quad ..] \\ StartNodeChk &= [1 \quad 0 \quad 0 \quad 1 \quad 0] \end{aligned} \tag{135}$$

$$\begin{aligned} EndNode &= [.. \quad .. \quad \# \quad \# \quad ..] \\ EndNodeChk &= [0 \quad 0 \quad 1 \quad 1 \quad 0] \end{aligned} \tag{136}$$

It is very simple to know if each distributed load is complete and has an appropriate beginning and end based on the total sum of *StartNodeChk* and *EndNodeChk*. If the sums of the two arrays were not equal, then GBeam would show an error message. As stated before, each possible distributed load is a combination of one or more basic shapes; uniform rectangular, positive slope triangle, or negative slope triangle. Each of these distributed load shapes have their own equivalent forces and moments [17]. These equivalent forces and moments can be found by taking the distributed load and finding the equivalent point load. This point load can then be used to find the reaction forces and moments at each end of the beam.

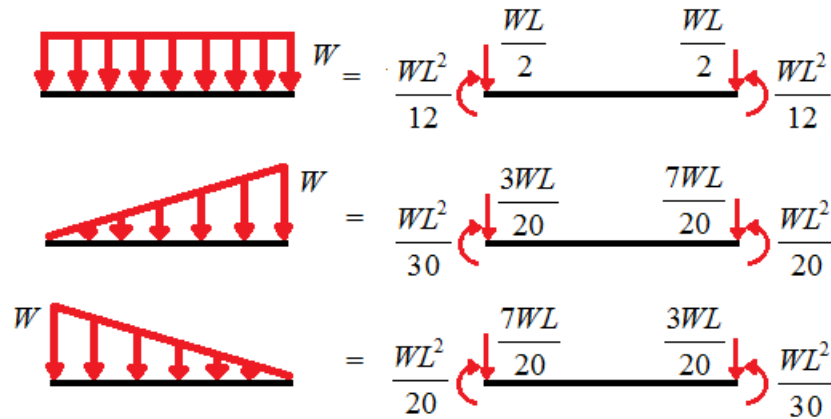


Figure 34 Distributed Loading Equivalent Forces and Moments

Using a **for loop** and variables used to count the index location, GBeam will start at the zero index for both StartNodeChk and EndNodeChk. As soon as GBeam finds the first non-zero values in these arrays, the numerical values located at the indexes of StartNode and EndNode will be compared in order to know which distributed load shapes are present in the beam. Additionally because of the addition of undeclared nodes between each user declared node, equivalent forces and moments at each node (declared or undeclared) will have the possibility of being equal to a combination of calculated forces from different shaped distributed loads. Finding the equivalent loadings at each applicable node (declared and undeclared) involves considering each beam element individually. If for instance a sloped distributed load covered three nodes, then the distributed loading applied to those three nodes could be read as:

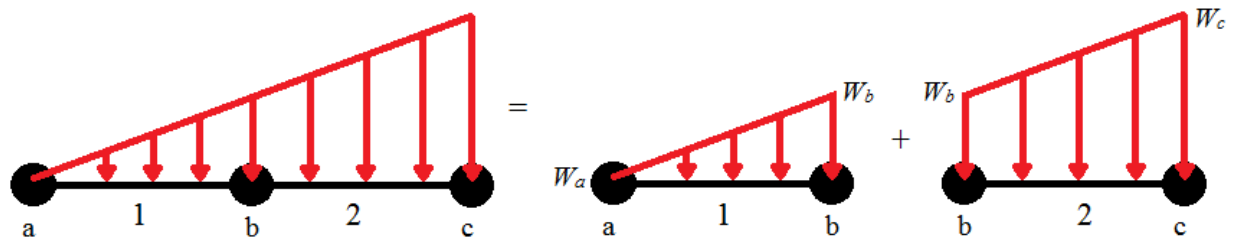


Figure 35 Sloped Distributed Load Split Between Individual Beam Elements

Where element two can be split further into a rectangular uniform distributed load and sloped distributed load similar to element 1. When all the equivalent loads are calculated for each node, the forces and moments for the distributed loading can be superimposed with the user input forces and moments to form a single array that represents the applied forces and moments on the global beam. Before the force array is applied to the $\{R\} = [K]\{D\}$ equation, the effects of the boundary conditions must first be applied to the array.

As explained in sections 5.1.1 and 5.3.2, each boundary condition will inhibit movement laterally, rotationally, or both. In order to keep the beam displacements in line with the expected behavior due to the presence of the boundary conditions, the force and moments values at the appropriate force array indexes must be converted to zero. The force array is ordered sequentially listing point loads and moments for each node. Thus in every force array, the even indexes (zero included) will list the point loads on the nodes while the odd indexes will list the moments applied to the array. In a fashion similar to how the boundary conditions affect the global stiffness matrix, the boundary conditions will make changes to the applied force and moment array.

A clamped boundary condition, which prohibits lateral and radial displacement, will cause the array elements associated with the node(s) the boundary condition is applied to become zero. Similarly, the support boundary condition, which prohibits lateral displacement, applied to any node will cause the point load at the appropriate index to become zero and the rotation boundary condition, which prohibits rotational movement, will cause the applied moment at the appropriate index to become zero. While GBeam will use a **for loop** to look at each node individually to see what boundary conditions are applied to the node in order to rewrite the

indexes of the force and moment arrays, it is possible relate the node number with the array index number to have a better understanding of which indexes are associated with which arrays. If it is assumed that the node numbering, n , starts with one, and that the array indexes start with zero, the point load force indexes and the moment indexes can be related using linear relations.

$$LoadIndex = 2n - 2 \quad (137)$$

$$MomentIndex = 2n - 1 \quad (138)$$

5.6 Beam Refinement

In this application there are two types of nodes that make up the global beam, declared nodes and undeclared nodes. Declared nodes, as the name suggests, are user declared nodes that can have boundary conditions, direct forces, direct moments, and springs applied to them. Additionally beam elements are made up of from sets of two declared nodes. To sum up, all data entered to the main menu of GBeam are classified as user declared values that will be used by GBeam to run the beam analysis. Undeclared nodes are additional nodes added between each set of declared nodes. These additional nodes will in turn make up individual elements that take on the material and shape properties of the elements made by the declared nodes. All declared nodes are designated free nodes with the user unable to change the boundary condition of those nodes. While the user is not able to directly apply point loads or moments to the undeclared nodes or change the boundary conditions of the undeclared nodes, application of distributed loads will still result in equivalent point loads and moments being applied to the undeclared nodes.

The default setting in GBeam in regards to additional nodes is to add a total of eight undeclared nodes in between each two declared nodes. Adding these additional nodes to the beam elements accomplishes two major things: the first is that refining the beam element by adding more nodes will have the effect of making the beam less “rigid” in a sense and will allow

for more accurate estimations when calculating the solution. The second, and main, reason that also related to the first is that the addition of the undeclared nodes keeps the plots from seeming less jagged in appearance. Additional nodes will give GBeam more points to plot, making the outputs smoother in appearance. This would even allow a global beam made of two user declared nodes to maintain a reasonable solution approximation and give a better visual representation of beam displacement. A good visual example that shows how the addition of additional undeclared nodes between the declared nodes affects the plots is presented in the appendix in Figure 50 and Figure 51. The figures show how the inclusion of additional nodes affects the visual display of the solution deformation outputs. Figure 50 shows a graphical display that is based off the deformation of three user declared nodes, while Figure 51 shows the same graphical solution with the addition of undeclared nodes added in between each two adjacent nodes. While applying additional nodes can affect each of both deformation graphs and the shear and moments diagrams, the biggest differences between both figures in this case are the outputs of the lateral and rotational displacement graphs. The addition of more nodes allows for more accurate graphs to be presented.

6. Solution and Outputs

As mentioned in section 5.1, the solution outputs are displayed in two separate windows. The first window displays a three column table which shows the node number, the beam lateral displacement, and the beam rotation (slope) for each user declared node. The second window displays four separate plots which represent the beam lateral displacement, the beam slope, the beam shear diagram and the beam moment diagram. Displaying the solution data in the two separate windows requires the ability to share data between each of the windows, which can be done by assigning data found in individual array indexes to either the browsers local storage or

session storage. Both session storage and local storage are mechanisms of web storage that allow browsers to store values [18] .

Even though session storage and local storage accomplish the same task of storing data values in the browser, there is one key difference between them. Session storage maintains a separate storage area that is available as long as the browser is open, including page reloads and restores. The local storage option accomplishes the same thing as the session storage, except that the values created by the local storage persist even when the browser is closed and reopened. This application makes use of local storage in order to share data between the windows due to its greater permanency compared to the data stored with session storage, which deletes session storage data when the browser or tab holding the data is closed.

The solution to the beam problem comes in two major steps after all the data has been collected. The first step involves taking all the data and finding the displacement of the beam and the reaction forces of the beam. The second step involves taking all the necessary solution data and storing it using the local storage option, from there the solution data is shared between the necessary windows and further processed if needed.

6.1 Displacements

The main function of this application is to take user input beam data and force/moment inputs in order to find the beam displacements, shear diagram, moment diagram, and the displacement plots. In order to find the displacements and reaction forces GBeam makes use of JavaScript code numeric.js [19], which allows GBeam to perform the matrix algebra necessary to solve any beam problem input to GBeams main menu. There will be two types of solution scenarios in GBeam, one where there are no hinges present in the beam problem and one where at least one hinge is present in the beam problem. These two solution scenarios exist because the

addition of hinges to the beam problem will dictate a need to change the appropriate displacement array index values. These index values can be found using the linear equation presented in section 5.5.1 where the moment index number is the same as the node rotation index.

Because the global stiffness matrix will have already accounted for the presence of hinges when approximating the beam displacement, the only change to be made to the displacement array is to include both slopes located at the hinge. As mentioned in section 5.3.2, the presence of a hinge boundary condition on a node means that there will be two different slopes present at the same node. In order to properly display the two different slopes in the displacement output table, both slopes must be calculated and placed into separate arrays that represent the slope when approaching the hinge from the left hand side and the slope when approaching the hinge from the right hand side. This is done by using a **for loop** to cycle through the boundary condition array and calling functions to calculate the slopes when detecting the presence of a hinge.

Detecting the presence of a hinge causes GBeam to call two functions known as ThetaHingeRight and ThetaHingeLeft, which returns the value of the rotation displacement for the beam when approaching the hinge from the left hand side and when approaching the hinge from the right hand side. These functions make use of the slope equations derived from at the end of section 5.3.2 from condensing the stiffness matrix.

```

function ThetaHingeRight(L,E,I,v1,theta1,v2) //The
{
    var Kc,Kb,Kd//Kc,Kb,Kd are the values from the
    Kb=(6*E*I)/(Math.pow(L,2))
    Kc=(4*E*I)/L
    Kd=(2*E*I)/L
    return ((1/Kc)*((-Kb*v1)-(Kd*theta1)+(Kb*v2)))
}
//Function which calculates the rotation angle for
function ThetaHingeLeft(L,E,I,v1,v2,theta2)//Theta
{
    var Kc,Kb,Kd//Kc,Kb,Kd are the values from the
    Kb=(6*E*I)/(Math.pow(L,2))
    Kc=(4*E*I)/L
    Kd=(2*E*I)/L
    return (-(1/Kc)*((Kb*v1)-(Kb*v2)+(Kd*theta2)))
}

```

Figure 36 Hinged Node Rotational Displacement Calculation Code

Once both slopes located at the hinge are known, the array element at the index associated with the hinge is replaced with a string value of the form $L:\theta_1/R:\theta_2$, where θ_1 is the value of the slope when approaching the hinge from the left hand side and θ_2 is the slope when approaching the hinge from the right hand side. These theta values are taken from newly formed slope arrays whose length is equal to the total node number and whose hinge slope index is equal to the index of the hinge location in the boundary condition array.

If no hinge is present in the beam then GBeam will approximate the node displacements by multiplying the inverse of the global stiffness matrix by the force array. Then, after the beam displacement array has been found it becomes possible to find the reaction forces by multiplying the global stiffness matrix by the newly found displacement matrix. The same process occurs when hinges are present in the beam problem, except the reaction forces are found before the displacement array is modified to account for the two separate slopes at a single node. From there, all of the displacement data, the reaction force data, the node data, boundary condition

data, and total hinge number data are placed into the local storage in order to share the data between the two separate output windows.

Because the elements located at each array index must be assigned to the local storage individually, special care must be taken when naming the local storage variables in order to keep the array data in the correct order. This can be done using a **for loop**, which is used to call the index elements from any specific array and assigning that element to a local storage variable whose name is dependent on the current loop iteration value. The code section below provides an example of how a local storage variable is created and named in order to keep the data in the correct sequence when reforming the arrays in a new window.

```
for(i=0;i<Solution.length;i++)
{
    localStorage.setItem("sol_"+i,Solution[i])
}
```

Figure 37 Variable Assignment to Local Storage Example Format

6.2 Displacement Output Window

Once the user chooses to run GBeam, the displacement window automatically opens and displays the calculated displacements of each declared node. The displacement output window consists of a three column table whose row number equals the row number of the main menu input spreadsheet. The first column of the table represents the node number of the user input nodes. The second column represents the lateral displacement of each node. The third column represents the rotational displacement of the nodes. The outputs displayed in each column of the table are the displacements of each user declared node. Additionally this window also displays the location and value of the maximum lateral displacement and slope displacement. This is done through a simple use of the **for loop** that cycles through the node and displacement data and compares lateral and slope displacements to each other.

Plot	
Max Lateral Displacement	5.447e-5
Max Lateral Displacement Node	2.000e+0
Max Slope Displacement	2.383e-4
Max Slope Displacement Node	1.000e+0

Input Data Beam Solution		
Node	Lateral Displacement	Slope (Radians)
1	0.000e+0	0.000e+0
2	-6.808e-6	2.383e-4
3	5.447e-5	-1.899e-16
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Figure 38 Beam Problem Displacement Output Example

Each cell in the table has its own id which will be used to assign the nodal displacements. The names of each cell id incorporate the row number, which starts with zero. The indexes used to assign the lateral and rotational displacement values to the table are a function of the number of undeclared nodes assigned to the beam. Due to the order of the displacement matrix, the lateral displacements will always exist in an even numbered index, which includes index zero, and the rotational displacements will always exist in an odd numbered index. The linear equations to find the indexes of the declared nodes will then be written as:

$$LateralIndex = U \times n \quad (139)$$

$$RotationIndex = (U \times n) + 1 \quad (140)$$

Where U represents the total number of undeclared nodes added in between each declared node. This value remains constant in the linear equation. The value n represents the specific declared node whose lateral or rotational displacement will be assigned to the output table. The indexes used in this lateral equation come from the original node array whose only values are the user declared nodes. Once the user is ready, they may click the plot button which causes GBeam to open the final window. This final window, the plot window, will use all data related to the declared and undeclared nodes to output later and rotational displacement plots as well as the shear and moment diagrams.

6.3 Plot Output Window

The plot output window makes use of the data stored in the local storage in conjunction with the Google charts algorithm to display a visual display of the beam displacements and the shear/moment diagrams. Due to the use of Google charts [20], the plot window is the only portion of the program that requires an internet connection to function. This is due to the fact that the library that allows the user to create graphical displays is very large, even in its condensed form. While it is possible to implement the chart library into the existing application, for the purposes of simplicity the chart library is sourced into the program through a URL link provided by Google developer. Google charts utilizes its own syntax to produce desired plots, which will not be given in detail in this section. This section will focus on the data preparation used to create the desired outputs.

There are six main arrays created which are used to create the output diagrams. All data in the local storage is placed into arrays when the plot window is called. Placing all the data into their own respective arrays makes the data easier to manipulate. As stated in previous sections all

data placed into arrays will be in the correct order due to the naming convention used when assigning data to the local storage.

Table 9 Output Diagram Data Source Array Name and Functions

Array Name	Function
solution	Holds the beam displacement approximations for lateral and rotational displacement
F	Holds the beam shear reaction forces
M	Holds the beam moment reaction forces
nodes	Holds the beam node location data for both declared and undeclared nodes
BC	Holds the beam boundary conditions located at all nodes, declared and undeclared
SecMod	Holds the section modulus at the location of each beam node, declared and undeclared

6.3.1 Lateral Displacement Diagram

All four output plots make use of the data stored in the local storage, where all relevant solution data is kept in order to allow multiple windows to access GBeams data. Out of all four plots, the lateral displacement diagram is the diagram that requires the least amount of data preparation. Just like the with the main menu displacement approximation, the plot window outputs take into account the presence of a hinge boundary condition. This is because the hinge boundary condition causes discontinuity with the rotation displacement diagram, which also affects the moment diagram. The lateral displacement diagram x -axis makes use of all node data stored in the local storage, this includes both user declared nodes and undeclared nodes. The lateral displacement diagram y -axis makes use of all the elements stored in the even numbered indexes of the solution array, starting from index zero. The only preparation that the data needs in order to correctly plot using Google charts is parse the string elements, which will turn the string elements into integers. This is necessary since the browser local storage only accepts string values. The result when implementing the entire node values, declared and undeclared, is a smooth displacement diagram like the one presented in the appendix in Figure 51.

6.3.2 Slope Diagram

The rotational displacement diagram is the plot that requires the most processing in order to correctly account for the presence of a hinge boundary condition. As stated in the previous section all data taken from the local storage comes in the form of strings. Google chart requires that all data plotted using its library should be in the form of integers. Changing the string values to integers is an easy task if no hinges are present. After changing the string values to integers, the Google charts library will be fed all node data as well as the odd numbered index elements located in the solution array, which will result is a smooth rotational displacement plot as presented in the appendix in Figure 51.

In the event that there is at least one hinge present in the beam problem, GBeam will execute a portion of code that will create data arrays that will represent the x-coordinate data and the y-coordinate data. These new arrays must be created in order to account for the discontinuity present at each hinge location, which necessitates plotting two separate y-coordinate points on a single x-coordinate point. When there is a hinge present in the beam, the displacement approximation will contain an element in the array that is of the form $L : \theta_L / R : \theta_R$. All data plotted by the Google charts library must be integers, meaning that the displacement elements will need to be modified in order to be able to convert the elements from a string to an integer.

```

if(hingeCount != 0)//Beam Slope (Rotation) chart data a
{
    for(i=0;i<TotNodes;i++)//loop works by attempting t
    {
        Slopes[i]=(Slopes[i].toString().split("L:"))
        Slopes[i]=(Slopes[i].toString().split("/R:"))
        Slopes[i]=(Slopes[i].toString().split(","))
        if(Slopes[i].length == 3)//Array values at hing
        {
            xCoordSlope.push(nodes[i],nodes[i])
            yCoordSlope.push(Slopes[i][1],Slopes[i][2])
        }
        else//non-hinge values will not have array vale
        {
            xCoordSlope.push(nodes[i])
            yCoordSlope.push(Slopes[i][0])
        }
    }
    for(i=0;i<xCoordSlope.length;i++)
    {
        xCoordSlope[i]=parseFloat(xCoordSlope[i])//arra
        yCoordSlope[i]=parseFloat(yCoordSlope[i])//arra
    }
}

```

Figure 39 Slope Displacement Diagram Array Data Assignment Code

When a hinge value is present in the beam problem it becomes necessary to get rid of all non-numerical characters present in the displacement solution array. This is easily accomplished using the `.split()` command that was used in the main menu to create tokens from an array element. Three different split commands, shown in the code section above, will need to be used in order to correctly split the displacement at the hinge into numerical tokens. It is easier to attempt to tokenize each index element in the array rather than check for specific index values to tokenize using the `.split()` command. Attempting to tokenize an array element that does not contain the character used to break the string will result in the element being replaced by an array that contains the original value of that was subject to tokenization, which ultimately results in the slope displacement array becoming a multidimensional array whose inner array sizes will vary. When an element in the array fits the criteria for tokenization, the end result will be a multidimensional array that replaces the affected element.

$$[\theta_1 \quad L:\theta_L / R:\theta_R \quad \theta_2] \quad (141)$$

$$\left[\begin{array}{c} \left[\begin{array}{c} \theta_1 \\ 0 \\ 0 \end{array} \right] \quad \underbrace{\left[\begin{array}{ccc} NaN & \theta_L & \theta_R \\ 0 & 1 & 2 \end{array} \right]}_1 \quad \left[\begin{array}{c} \theta_2 \\ 0 \\ 2 \end{array} \right] \end{array} \right] \quad (142)$$

Because the final form of the array after tokenization is a multidimensional array it becomes necessary to create new arrays that represent the y-axis points (rotational displacement) and x-axis points (beam node locations). It becomes more evident that this is necessary when taking into consideration that the node arrays will have to include duplicate values in order to correctly plot out the rotation discontinuity. This is what the code section presented above does once the rotational displacement array is tokenized. The code will assign any hinge values to an array representing y-axis values while simultaneously assigning duplicate node location values to a new x-axis array. The final forms of the arrays used to plot the rotation displacement plot are given below.

$$\begin{aligned} xCoordSlope &= [N1 \quad N2 \quad N2 \quad N3] \\ yCoordSlope &= [\theta_1 \quad \theta_{2_L} \quad \theta_{2_R} \quad \theta_3] \end{aligned} \quad (143)$$

6.3.3 Shear and Moment Diagram

Both the shear and the moment diagram make use of the force matrix $\{R\}$ which includes the beam reaction forces and moments as well as the user applied forces and moments. Both the shear and moment diagram will require new node, shear force, and moment arrays to be created due to the process implemented in order to arrange the data used to plot out the shear and moment diagrams. The process to prepare the data for the shear diagram is very straight forward. The process involves creating two new arrays; one that will hold the shear force values going

through the beam and one that will hold the node values that will be applied to the x-axis of the shear diagram. These new arrays are needed due to the same reason that new arrays are created to hold the data when plotting the rotational displacement if hinges are present. Certain sections of the beam, regardless of whether or not a hinge is present on the beam, requires to separate points to be plotted on the same x-coordinate point. This represents a change in the shear force value running through the beam as a result of existing shear forces applied on the beam. All data process comes from two arrays known as nodes and F, whose function is given in the table at the beginning of section 6.3. The new arrays and their functions are given in the table below.

Table 10 Shear Diagram Data Arrays Names and Functions

Array Name	Function
Shear	Holds the shear force value present at each node in the beam
ShearNode	Holds all nodes that will be used to plot the shear diagram

The first step when preparing the data is to assign the very first index of the node array and the shear force to arrays ShearNode and Shear respectively. The reason why the values at the zero indexes are automatically assigned to the new arrays is due to the fact that each new index added to the arrays Shear and ShearNode may be compared to the previous index element in those arrays in order to correctly estimate the shear running through the beam. Using a **for loop**, whose total iterations equals one minus the total number of nodes present, GBeam starts inspecting the elements of the force array, F, starting with index number one and modifying the force array as necessary while simultaneously assigning values to the arrays ShearNode and Node. The example below shows how the node and force arrays change as the **for loop** runs through each iteration. At each iteration, GBeam checks to see if the element in the current index is a zero value. If the element is a zero value, then the current element value and the previous

element value in array F will be added together and replace the current value of array F, while the summation of the current and previous index is assigned to array Shear. Additionally the array ShearNode will be assigned the element found in the current index of array nodes. This trend will continue until a non-zero value is found, as is what happens in the final iteration.

When a non-zero value is found, the current index element value and the previous element value will be summated and assigned to the current index of array F. The big difference between finding a non-zero value instead of a zero value is that instead of only adding one new value to the arrays Shear and ShearNode, two values will be added. For the array Shear the two values added will be the previous index element and the summation of the current non-zero index element and the previous index element value. The two values added to the array ShearNode will be the same value, which is the current index element. This is the reason that new arrays must be created when preparing the data that will be used to plot the shear diagram. Usually the size of each array will be related in some form. Either each array will be equal or double the total amount of nodes present in the beam, but the arrays created to plot the data will need to be a different size in order to hold all the data in order to properly display the output plots.

$$\begin{aligned}
& \left. \begin{aligned} F &= [\#_1 \quad 0 \quad 0 \quad \#_2] \\ node &= [N_1 \quad N_2 \quad N_3 \quad N_4] \\ Shear &= [\#_1] \\ ShearNode &= [N_1] \end{aligned} \right\} \text{Iteration1} \\
& \left. \begin{aligned} F &= [\#_1 \quad \#_1 \quad 0 \quad \#_2] \\ node &= [N_1 \quad N_2 \quad N_3 \quad N_4] \\ Shear &= [\#_1 \quad \#_1] \\ ShearNode &= [N_1 \quad N_2] \end{aligned} \right\} \text{Iteration2} \\
& \left. \begin{aligned} F &= [\#_1 \quad \#_1 \quad \#_1 \quad \#_2] \\ node &= [N_1 \quad N_2 \quad N_3 \quad N_4] \\ Shear &= [\#_1 \quad \#_1 \quad \#_1] \\ ShearNode &= [N_1 \quad N_2 \quad N_3] \end{aligned} \right\} \text{Iteration3} \\
& \left. \begin{aligned} F &= [\#_1 \quad \#_1 \quad \#_1 \quad (\#_1 + \#_2)] \\ node &= [N_1 \quad N_2 \quad N_3 \quad N_4] \\ Shear &= [\#_1 \quad \#_1 \quad \#_1 \quad \#_1 \quad (\#_1 + \#_2)] \\ ShearNode &= [N_1 \quad N_2 \quad N_3 \quad N_4 \quad N_4] \end{aligned} \right\} \text{Iteration4}
\end{aligned} \tag{144}$$

The moment diagram requires the most amount of data processing to correctly display. This is because the moment diagram makes use of all forces and moments in the force matrix $\{R\}$. Preparing the data used in the shear moment diagram comes in two major parts. The first part involves finding the change in moment force throughout the beam as a function of the shear forces and the length of the beam. The second part involves incorporating the moments from the force matrix $\{R\}$ to find appropriate sudden decreases or increases in the moment force throughout the beam.

The change in moment force throughout the beam can be seen as a function of both the shear force applied to the beam and the distance from the force to the end of the beam. The example below [21] shows the basic idea behind the calculations of the moment data points. In

any beam where there are applied forces, there will also be reaction forces. It is also known that a moment is equal to force times distance.

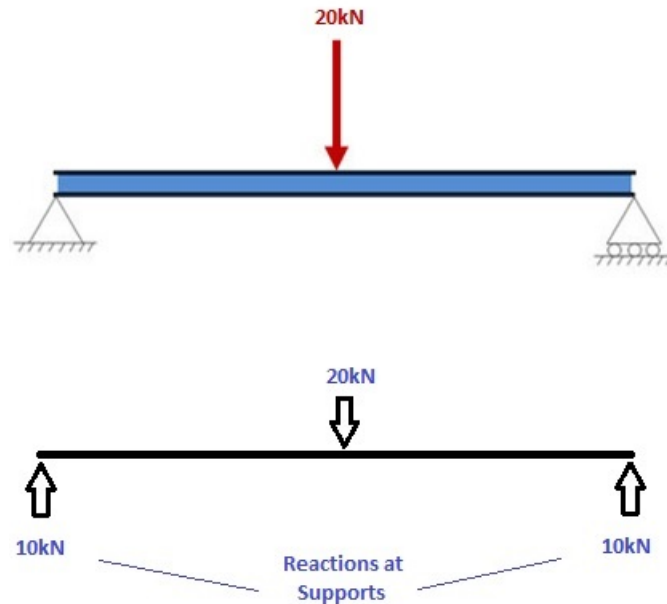


Figure 40 Simply Supported Beam Free Body Diagram

When taking a cut of the beam, so that only the left most force is considered, it can be said that the moment force on the beam increases linearly with increasing distance from the beam, as the force considered is positive. When the middle load is considered, then the moment force will begin to decrease by $(-20kN)x$ while still increasing by $(10kN)x$.

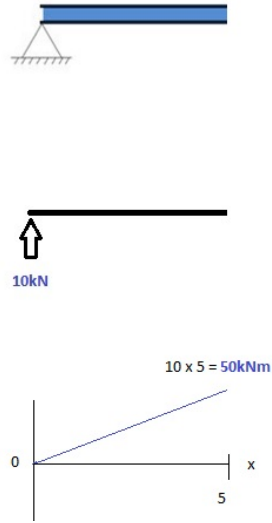


Figure 41 Beam Section Moment Diagram Creation

The inclusion of the middle force would lead to an overall decrease in the slope of the moment force diagram as:

$$\left. \begin{array}{l} 10x \\ 10A + 10(x - A) - 20(x - A) \end{array} \right\} \begin{array}{l} , 0 \leq x \leq A \\ , A \leq x \leq B \end{array} \quad (145)$$

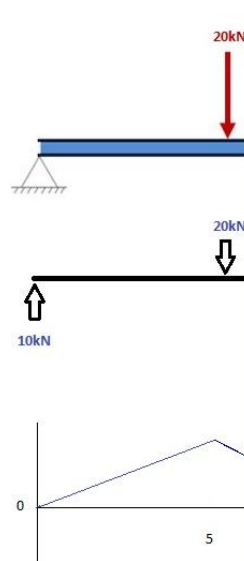


Figure 42 Beam Moment Diagram Creation Continuation

Where A and B are the distances from the beginning of the beam. This basic concept of each shear force creating its own linear change in moment across the beam is transferred to GBeam. Using a **while loop**, GBeam finds the first shear force and finds the next shear force applied to the beam, which allows GBeam to know the total distance between the two forces. The index locations of the two forces are stored in temporary values in order to properly document the moment value at each node between the two forces. Once all moments between the two forces are documented, the process begins again from the second force with GBeam looking for a possible third force. The moment values between forces two and three are found in a manner similar to the second portion of the example above. All of these values will be placed into the first of two new arrays created to hold the data values of the moment diagram. The second of these two arrays will hold the final moment diagram data values when taking the moment reaction forces into consideration.

Table 11 Moment Diagram Data Arrays Names and Functions

Array Name	Function
MmntGrph	Holds moment diagram data points calculated from shear forces
FinalMmntGrph	Incorporates moment reaction forces to values from array MmntGrph
MmntNodes	Holds node location data used in the moment diagram

The incorporation of the moment reaction forces invokes the need for duplicate node values much like with the shear diagram. The presence of a moment on the beam will cause two points to be plotted on the same node location. Additionally adjustments to the moment data will need to be made if a hinge is present on the beam. All moment forces applied to the node where a hinge is located must be set to zero. This is due to the fact that hinges do not transfer moments, as explained in section 5.3.2. Even if a moment is applied to the hinge it will have no overall

effect on the moment diagram. A series of plot outputs are presented in the appendix, which shows various forms the graphs can take depending on the data the user inputs to GBeam.

6.3.4 Bending Stress Diagram

The bending stress diagram output by GBeam utilizes the data created for the moment diagram, the moment of inertia data array and the distance from the neutral axis data array. Equation 146, the bending stress equation [22], can be rewritten into a form that utilized the section modulus of the beam, which is a function of the perpendicular distance of the neutral axis to the top of the beam and the moment of inertia.

$$\sigma_{\max} = M \frac{c}{I} = M \times \frac{1}{S} \quad (146)$$

The method of calculation of the bending stress at each node was done by finding the section modulus values at each node. Thus all node values are assigned a value for the moment of inertia and a value of perpendicular distance from the neutral axis. Assigning geometry values to the nodes will lead to discontinuities in the diagram, as one node can be shared by two different beam elements with different geometric properties. This means it will be possible that one node will be assigned two separate geometric values. The figure below shows a visual example of the explanation given.

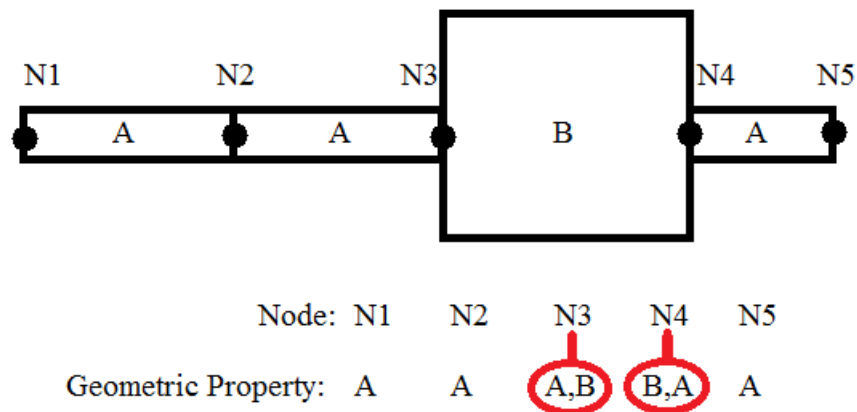


Figure 43 Node Geometric Property Assignment Example

Figure 45 shows why certain beam nodes will be assigned two geometric properties. In Figure 45, both nodes 3 and 4 are shared by beams that are two different sizes. This means that the geometry values will be different depending on whether the node is being approached from the left or from the right. These changes in the geometry of the beam will be seen as sudden drops or rises in the beam stress diagram.

7 Output Comparison

Two separate finite element analysis software are used to validate the solutions given by the JavaScript finite element beam analysis application. The two software used to validate GBeam results are Ansys APDL [23] and Engineering Toolbox general beam analysis program [24]. Both programs allow users to input node, loading, and material data in order to approximate the overall displacement of the beam. The end results are a main menu input spreadsheet that will output the displacement approximations along with five output diagrams.

While all three finite elements analysis programs require the user to input the beam data through different formats, this being especially true for the Ansys APDL software, the main similarity that all three programs share in terms of data input is the fact that there is no option for the user to input specific units. Instead it is up to the user to input all data with consistent units. Any inputs whose units are consistent give understandably consistent units. Both Ansys APDL and ETBX (Engineering Toolbox) program will be used to check the correctness of the displacement approximation of the JavaScript finite element analysis program while only ETBX program will be used to validate the output graphs, mainly the shear and moment diagrams.

Validating GBeam required running a series of beam analysis that make use of all the input options that are available from GBeam. This includes all boundary conditions, loading

options, spring additions, and the option to include non-uniform beam data in the input spreadsheet. A few solution approximation examples will be provided in this section, with the rest of the beam data inputs presented in the Appendix. There will be two types of materials and moments of inertia used in the beam problems. The two different materials will be used to test whether the JavaScript application can correctly approximate the displacement of non-uniform beams with sections made of different materials. The two materials [25] and area moments of inertia used are listed in the table below.

Table 12 Beam Problem Elements Physical Properties and Geometries

Material	Modulus of Elasticity
Aluminum	68GPa
Stainless Steel	77GPa
Area Moment of Inertia	
1.8E-6m ⁴	
3.6E-6m ⁴	

The first listed material and moment of inertia will be used as the default properties when the beam problem consists of only one material and is considered uniform. Fifteen beam scenarios are presented in this paper, with the scenarios gradually becoming more complex. While the bending stress is calculated in ETBX it is not plotted by the application. Bending stress results are listed out in ETBX and validate the results found by GBeam. Some of the cases presented here were run with ETBX, Ansys, and GBeam, while a few were only run with ETBX and GBeam.

Scenario 1

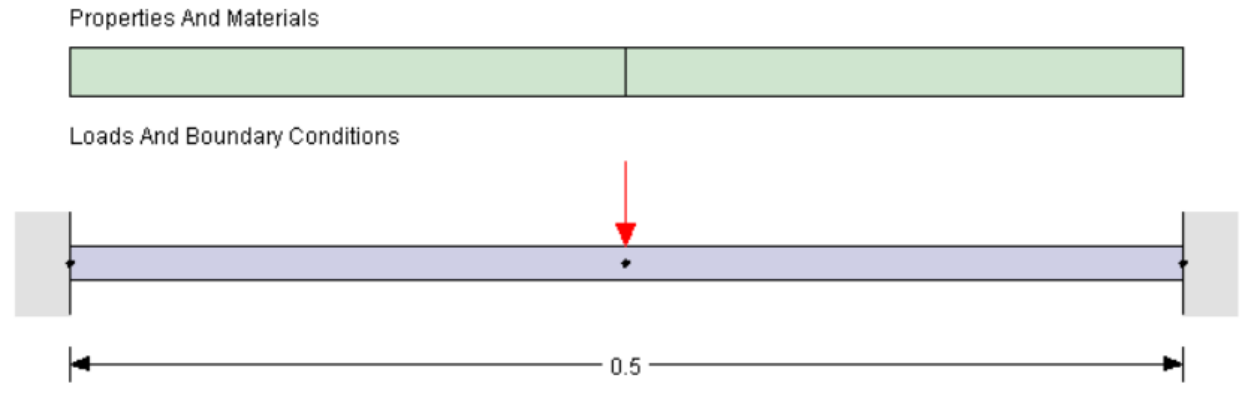


Figure 44 Beam Problem 1 Set Up

The first problem presented is a simple beam case, where the beam is clamped at both ends with a load applied at the middle of the beam. This beam is made up of three nodes, with uniform beam element cross sectional area and material. The input format used to create the beam problem in the JavaScript application is given below. Following the input format is a table detailing the displacement results as given by the Ansys, the JavaScript application and ETBX. Comparisons of the results are given in terms of percent error. Two different types of percent errors are given for both the lateral displacement results and the slope results. In both cases, the JavaScript displacements are compared to either the Ansys displacement results or the ETBX displacement results. In both cases the Ansys and ETBX results are taken to be the true value of beam displacement when compared to the JavaScript displacements.

Table 13 Beam Problem 1 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	C (Dist From Neut. Axis)
1	0	c			0	0	../..	1.80E-06	6.80E+10	.5
2	0.25				-1000	0	../..			
3	0.5	c			0	0	../..			

Table 14 Beam Problem 1 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0	0	0	0	0	0
2	-5.319E-06	-5.319E-6	-5.319E-6	0	0	0
3	0	0	0	0	0	0

Table 15 Beam Problem 1 Node Displacement Value Comparison

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Finally a graphical comparison is shown between the shear diagram, moment diagram, lateral displacement, and slope displacement of the ETBX program and the JavaScript application. Comparison between the two sets of graphs shows high similarity between the values given by both ETBX and the JavaScript application.

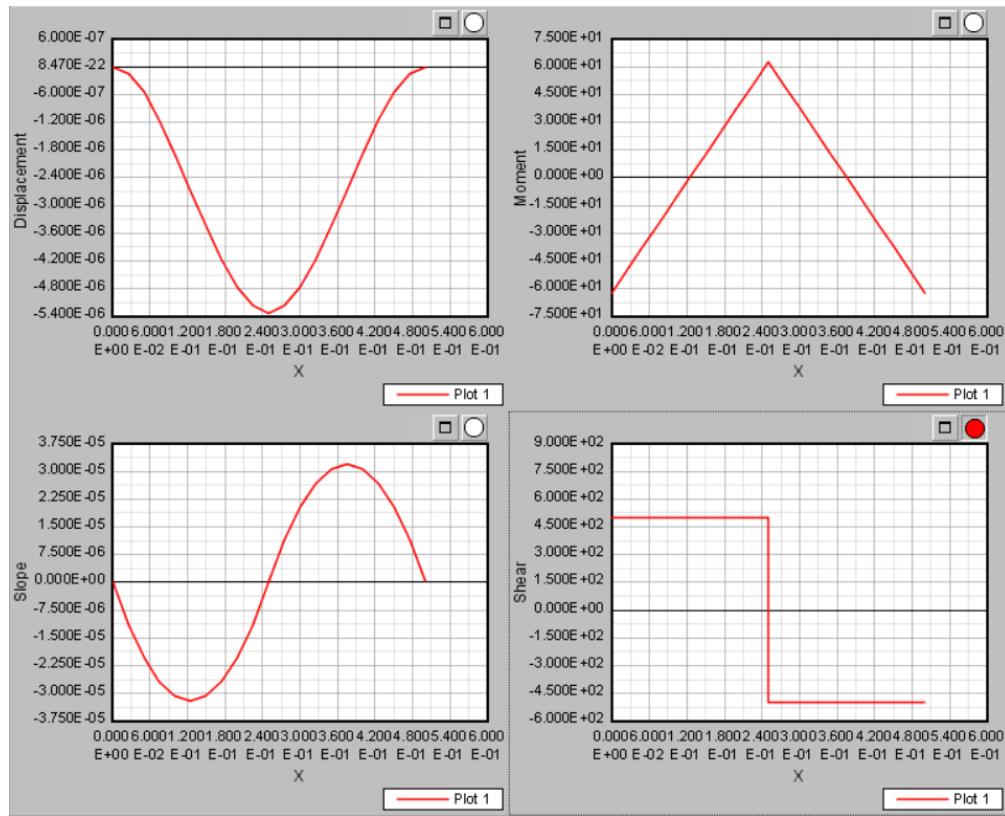


Figure 45 Beam Problem 1 ETBX Diagram Outputs

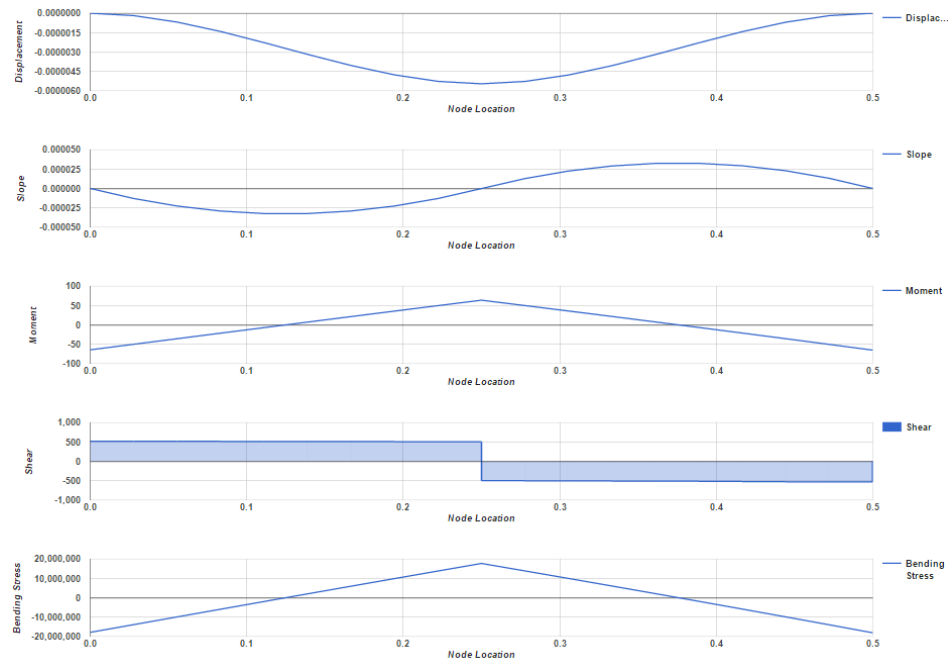


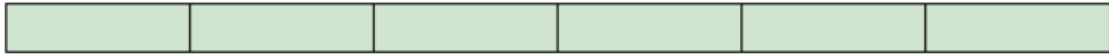
Figure 46 Beam Problem 1 JavaScript Application Diagram Outputs

As seen above, for a simple three declared node case with only one applied force, the overall displacement approximations between the three programs align perfectly; there are no percentage errors between the JavaScript displacements and the Ansys displacements or the JavaScript displacements and the ETBX displacements. Discrepancies are more likely to start occurring in more complex scenarios, such as when more declared nodes are added to the beam problem. As evidenced starting from Scenario 2 in the appendix, the addition of more nodes will create a very slight discrepancy between the JavaScript application displacements and the displacements of the other two programs. In general the results given by the three separate programs are consistent with each other

Scenario 2

Starting with beam scenario 2, the beam problems will contain a total of seven user declared nodes. This problem is a uniform beam with three simple supports placed evenly throughout the beam. Point loads are applied to each end of the beam and moment forces are applied to the left and right of the simple support applied at the center of the beam. The biggest difference between the percentage errors presented in this scenario and the percentage errors presented in scenario 1 is that variations between the displacement results exist, unlike in scenario one where no such variations exist. As explained in scenario 1, the variations can be explained due to the fact that final displacement outputs for the JavaScript application are rounded to the fifth significant digit. However, it is important to note that percentage errors for this scenario do not come close to reaching even one percent, even at the highest percentage error. Comparison between the two sets of graphs from ETBX and the JavaScript application show similar outputs.

Properties And Materials



Loads And Boundary Conditions

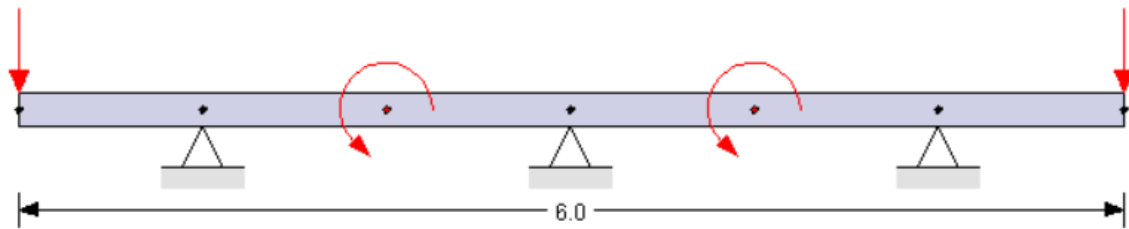


Figure 47 Beam Problem 2 Set Up

Table 16 Beam Problem 2 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	C (Dist From Neut. Axis)
1	0				-500		../..	1.80E-06	6.80E+10	.5
2	1	s					../..			
3	2					100	../..			
4	3	s					../..			
5	4					100	../..			
6	5	s					../..			
7	6				-500		../..			

Table 17 Beam Problem 2 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	-3.34E-03	-3.34E-03	-3.34E-03	4.02E-03	4.02E-03	4.02E-03
2	0.00E+00	0.00E+00	0.00E+00	1.97E-03	1.97E-03	1.97E-03
3	5.11E-04	5.11E-04	5.11E-04	-3.74E-04	-3.75E-04	-3.74E+00
4	0.00E+00	0.00E+00	0.00E+00	-6.81E-05	-6.81E-05	-6.81E-05
5	5.11E-04	5.11E-04	5.11E-04	6.47E-04	6.47E-04	6.47E-04
6	0.00E+00	0.00E+00	0.00E+00	-2.11E-03	-2.11E-03	-2.11E-03
7	-3.47E-03	-3.47E-03	-3.47E-03	-4.15E-03	-4.15E-03	-4.15E-03

Table 18 Beam Problem 2 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0.002997512	0.001708604	0.002489482	0.002862915
2	0	0	0.020259319	0.020309957
3	0.003916807	0.004093056	0.010682049	0.011937339
4	0	0	0.004406386	0.004097952
5	0.003916807	0.004093056	0.001546097	0.002087242
6	0	0	0.018951957	0.020563204
7	0.005760037	0.0063936	0.00240784	0.001203934

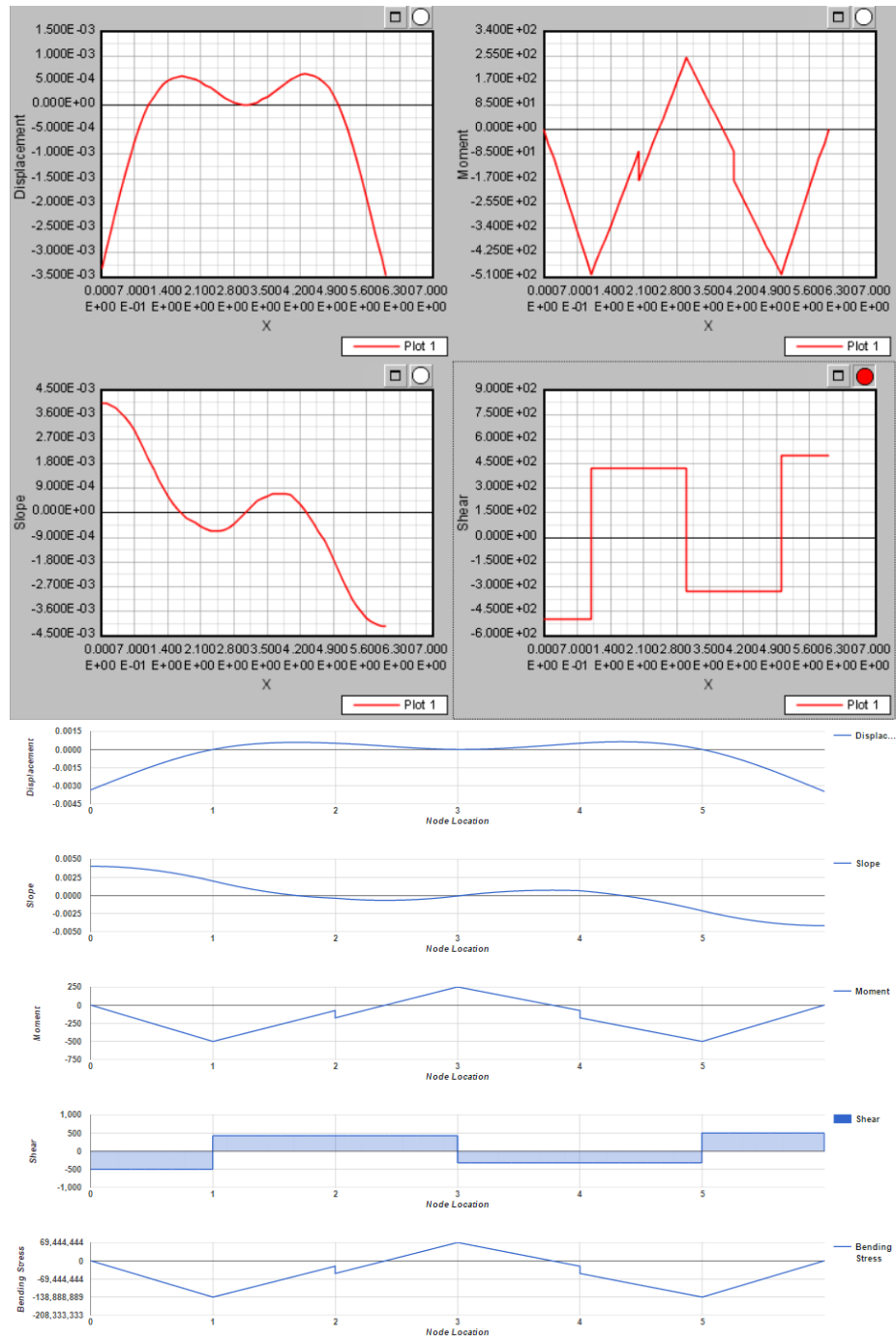


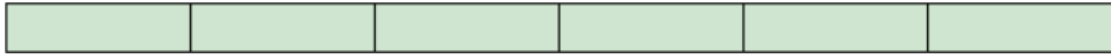
Figure 48 Beam Problem 2 ETBX & JavaScript Application Output Diagrams

Scenario 4

While the previous two presented beam scenario displacement outputs have shown that the variations between the displacements of the three different programs are negligible, this beam case shows that there are certain beam problems that will produce higher percentage errors than have been previously presented. The beam presented in this scenario is the first beam case presented that makes use of a hinge boundary condition. As explained in section 5.3.2 the hinge boundary condition creates a discontinuity in the slope displacement of the beam. This leads to two separate rotation displacements existing at the node where the hinge is applied.

Presented in the displacement result table in the cells that coincide with the node rotation displacement on the hinge are the two separate rotation displacements found at the hinge. While both Ansys and the JavaScript application display both of the rotation displacements found at the hinge, the ETBX program will only display one of the rotation displacements found at the hinge. The ETBX program does take into account a hinge boundary condition, as shown by the graph outputs that shows similar results to the JavaScript application. Variations of the displacement solutions between Ansys and JavaScript are higher than previously presented results, but the differences do not exceed 5%, while the variations between JavaScript and ETBX are in line with previous comparisons, with variations staying at 1% or lower.

Properties And Materials



Loads And Boundary Conditions

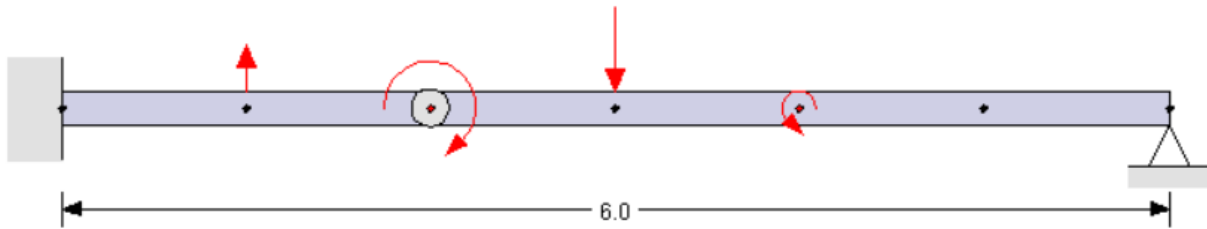


Figure 49 Beam Problem 4 Set Up

Table 19 Beam Problem 4 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m⁴)	Youngs Mod (Pa)	C (Dist From Neut. Axis)
1	0	c					../..	1.80E-06	6.80E+10	.5
2	1				200		../..			
3	2	h				-1000	../..			
4	3				-500		../..			
5	4					150	../..			
6	5						../..			
7	6	s					../..			

Table 20 Beam Problem 4 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	-6.24E-03	-6.35E-03	-6.35E-03	-1.24E-02	-1.24E-02	-1.24E-02
3	-2.36E-02	-2.40E-02	-2.40E-02	L:-0.22263E-01 R:0.21094E-02	L:-2.204e-2 R:2.213e-3	-2.23E-02
4	-2.07E-02	-2.12E-02	-2.12E-02	3.79E-03	3.90E-03	3.90E-03
5	-1.54E-02	-1.57E-02	-1.57E-02	6.81E-03	6.91E-03	6.91E-03
6	-8.07E-03	-8.22E-03	-8.22E-03	7.88E-03	7.98E-03	7.98E-03
7	0.00E+00	0.00E+00	0.00E+00	8.24E-03	8.34E-03	8.34E-03

Table 21 Beam Problem 4 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0	0	0	0
2	1.820222917	0.004410338	0.016118633	0.015393183
3	1.430264049	0.020279456	L:1.00166195032117 R:4.91134919882432	1.001973223
4	2.164794369	0.003633657	2.727632099	0.006670532
5	1.964088935	0.018312317	1.510165707	0.005831787
6	1.857243746	3.64919E-05	1.313535123	0.003670434
7	0	0	1.251684493	0.001702609

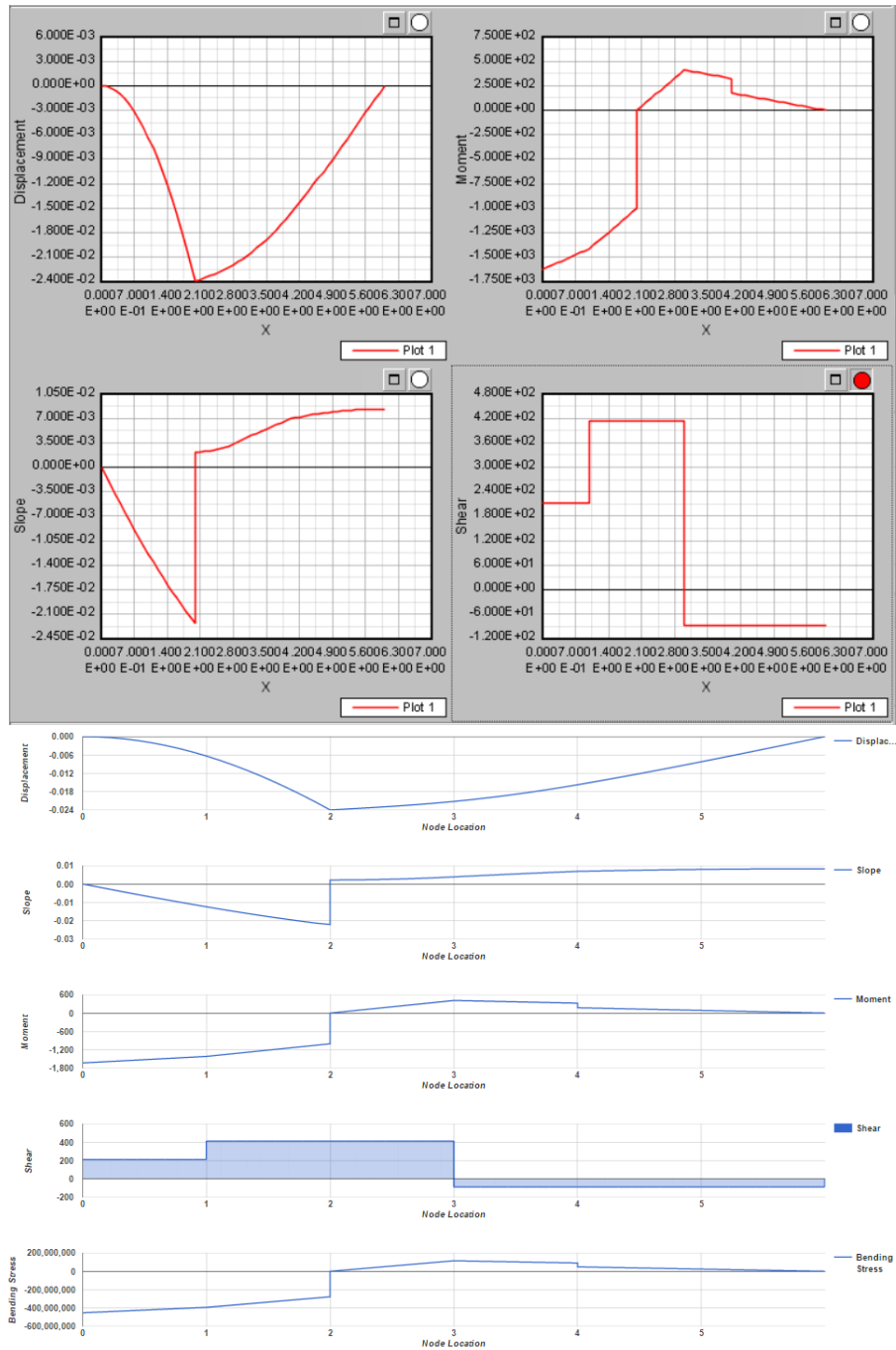


Figure 50 Beam Problem 4 ETBX & JavaScript Application Diagram Outputs

The remaining beam comparisons are presented in the appendix. Overall comparisons between the three different programs show that the deformation results are very consistent with each other.

8 Conclusions

The finite element beam analysis application achieves all the goals that it was meant to achieve. The goals being:

- GBeam is able to analyze multiple types of user input beam problems, as it is able to create and adjust the global stiffness matrix of the beam in accordance to the nodes, material properties, beam inertia values, and boundary conditions input into GBeam.
- GBeam is able to find accurate deformation values for each user input beam problem.
- GBeam is able to output accurate lateral and slope deformation graphs as well as accurate shear diagrams, moment diagrams, and bending stress diagrams.
- GBeam is customizable by the user as it is a client-side application written in JavaScript.
- GBeam spreadsheet data is savable by the user, which allows users to revisit specific problems at various future dates.

GBeam is able to achieve the goals mentioned by implementing a user input spreadsheet that is created using a combination of JavaScript and HTML that allows the user to input the necessary beam data and applied force and moment data that allows GBeam to create a global stiffness matrix that is modified in order to accurately reflect the effect of any possible boundary conditions applied to the beam. This global stiffness is then be used in conjunction with the force matrix created by the combination of applied forces, moments, and distributed loads in order to calculate the beam deformations. This deformation matrix is then used to find the reaction forces

that allow GBeam to calculate the data used to graph the shear and moment diagrams for the beam problem. The values calculated for the moment diagram are then used in conjunction with the section modulus of the beam elements in order to calculate the bending stress at each beam node. In total this process allows GBeam to output a table of beam deformations as well as a series of graphs that detail the beam lateral deformation, the beam slope deformation, the shear diagram, moment diagram, and the bending stress diagram.

The outputs created by the JavaScript application were validated using two separate finite element beam analysis programs. These programs were the ANSYS APDL program the Engineering Toolbox (ETBX) general beam analysis program. The comparison of the output data between the ETBX program and the output data of the JavaScript application show that the overall percent difference between the deformation outputs are very small, almost negligible, meaning that the results of ETBX and the JavaScript application agree. While the percent difference between the deformation data between Ansys APDL and the JavaScript application are greater compared to the percent differences between ETBX and the JavaScript application, the overall results between Ansys APDL and the JavaScript application do not exceed 10%, leaving the deformation results of the JavaScript application within an acceptable range of the Ansys APDL application.

All data input by the user the main menu input spreadsheet can be saved by the user onto their own computer or an external storage device. Saving the data is a simple process that simply involves opening the main menu input spreadsheet in a browser such as Firefox and saving the webpage to the storage folder of the users choosing. The saves file can be given any name in order to facilitate quick reference for future date. At this point all the user would need to do is open the saved menu input spreadsheet to find all their input data intact and run the analysis

again. The user would need to make sure that they have access to the other two files which represent the output window and the plot window.

9 Future Work

This application is fully customizable to potential users of GBeam as this script is written in a client-side format. This means that when the user is presented with GBeam, they are also presented with the tools that make the analysis conducted by GBeam possible. This allows any user with a basic knowledge of the JavaScript programming language to make any changes to GBeam as fits their specific needs. A basic addition to the current version of GBeam which would facilitate easier data inputs would be the addition of a basic materials table that would allow GBeam to know the material value of typical materials used in beams. Each material could be assigned a specific ID, which would allow the user to create a beam without having to look up the specific Young's modulus of certain materials.

While GBeam is currently limited to the analysis of static beam problems whose individual elements are uniform, GBeam itself can be expanded in order to allow the user to solve dynamic problems, problems with elements with variable cross sections, or both. Implementing the option to solve a dynamic beam problem, while requiring more scripting, would not be any more difficult than the scripting that has already been done in order to solve static problems. One of the main additions to GBeam would have to be the ability to form a mass matrix, which would require the ability to input additional data to the main menu spreadsheet, such as beam element material density and beam element cross sectional area. Using this new mass matrix in conjunction with the natural frequency and the stiffness matrix would allow GBeam to solve for the natural frequencies and modes of vibration.

Additionally solving for beam elements with variable cross sections would require the ability to declare the shape of the beam and the measurements at each end of the beam. This would allow GBeam to form the moment of inertia of the variable cross section beam element based on the linear change in the cross section of the beam. Calculating the moment of inertia for a beam element such as this would require additional analysis due to the change of beam element cross sectional area. Although the addition of these analysis types would require a revision of the main menu input spreadsheet, the output windows, and the main analysis portion of the script, these additions would be possible to implement and would not be the only analysis types that could be added to GBeam.

Bibliography

- [1] ITRS, "International Technology Roadmap for Semiconductors 2.0 2015 Edition Executive Report," 2015.
- [2] Brian Benton. (2014, January) AUGI. [Online]. <https://www.augi.com/articles/detail/pricing-trends-in-the-cad-industry>
- [3] David Hall. Stanford Web. [Online]. <http://web.stanford.edu/class/cs98si/slides/overview.html>
- [4] Mozilla Developer Network and Individual Contributors. (2016, July) Mozilla Developer Network. [Online]. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What_is_JavaScript
- [5] s.india. (2008, July) Code Project. [Online]. <http://www.codeproject.com/Articles/27775/Object-Oriented-Programming-Concepts>
- [6] Jen Looper. (2015, September) Telerik Developer Network. [Online]. <http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/>
- [7] B.P. Wang K.L. Lawrence B.H Dennis, "Light-Weight, Client Side Engineering Computational Utilities," *Proceedings of the Twelfth International Conference on Computational Structures Technology*, 2014.
- [8] R.W. Clough, "Early history of the finite element method from the viewpoint of a pioneer," *International Journal for Numerical Methods in Engineering*, vol. 60, no. 1, 2004.
- [9] Robert D Cook, *Concepts and Applications of Finite Element Analysis 4th Edition*. Hoboken: John Wiley & Sons, Inc, 2002.
- [10] Geatano Giunta, Marco Petrolo Erasmo Carrera, *Beam Structures Classical and Advanced Theories*. West Sussex, United Kingdom: John Wiley & Sons, Ltd, 2011.
- [11] Rajeev Kumar. (2014) ME 5310 Lecture Note 2. Lecture Notes.
- [12] R.C. Hibbeler, "Slope and Displacement by Integration," in *Mechanics of Material 8th Edition*. United States: Pearson Prentice Hall, 2011, ch. 12, p. 573.
- [13] J.S.Przemieniecki, *Theory of Matrix Structural Analysis*. United States of America: McGraw-Hill Inc, 1968.
- [14] David Flanagan. (2006) JavaScript: The Definitive Guide, 5th Edition. PDF.

- [15] Rajeev Kumar. (2014) AE5310 Lecture Note 15. Lecture Notes.
- [16] Kent Lawrence. (2010) FEM Modeling. MAE 5310 Lecture Notes.
- [17] W.F. Carroll, *A Primer For Finite Elements in Elastic Structures*. New York: John Wiley & Sons, Inc., 1998, vol. 1.
- [18] Mozilla Developer Network and Individual Contributors. (2016, August) Mozilla Developer Network. [Online]. https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
- [19] Sébastien Loisel. (2012, December) Numericjs. [Online]. <http://numericjs.com/wordpress/?p=79>
- [20] Google Inc., Google Charts API, 2007.
- [21] SkyCiv. (2013) The Free Bending Moment and Shear Force Beam Calculator. [Online]. <http://bendingmomentdiagram.com/tutorials/how-to-find-bending-moment-diagrams/>
- [22] R.C. Hibbeler, "The Flexure Formula," in *Mechanics of Materials 8th Edition*. United States: Pearson Prentice Hall, 2011, ch. 6, p. 287.
- [23] ANSYS, Inc., Ansys Mechanical APDL 16.2.
- [24] P. Parkhi, V. Tandra Sistla, K.L. Lawrence M. Seifert, "Developing and distributing network based engineering solutions," *Advances in Engineering Software*, pp. 453-465, June 1999.
- [25] MatWeb, LLC. MatWeb Material Property Data. [Online]. <http://www.matweb.com/search/DataSheet.aspx?MatGUID=71396e57ff5940b791ece120e4d563e0>

Appendix A- Additional Application Images

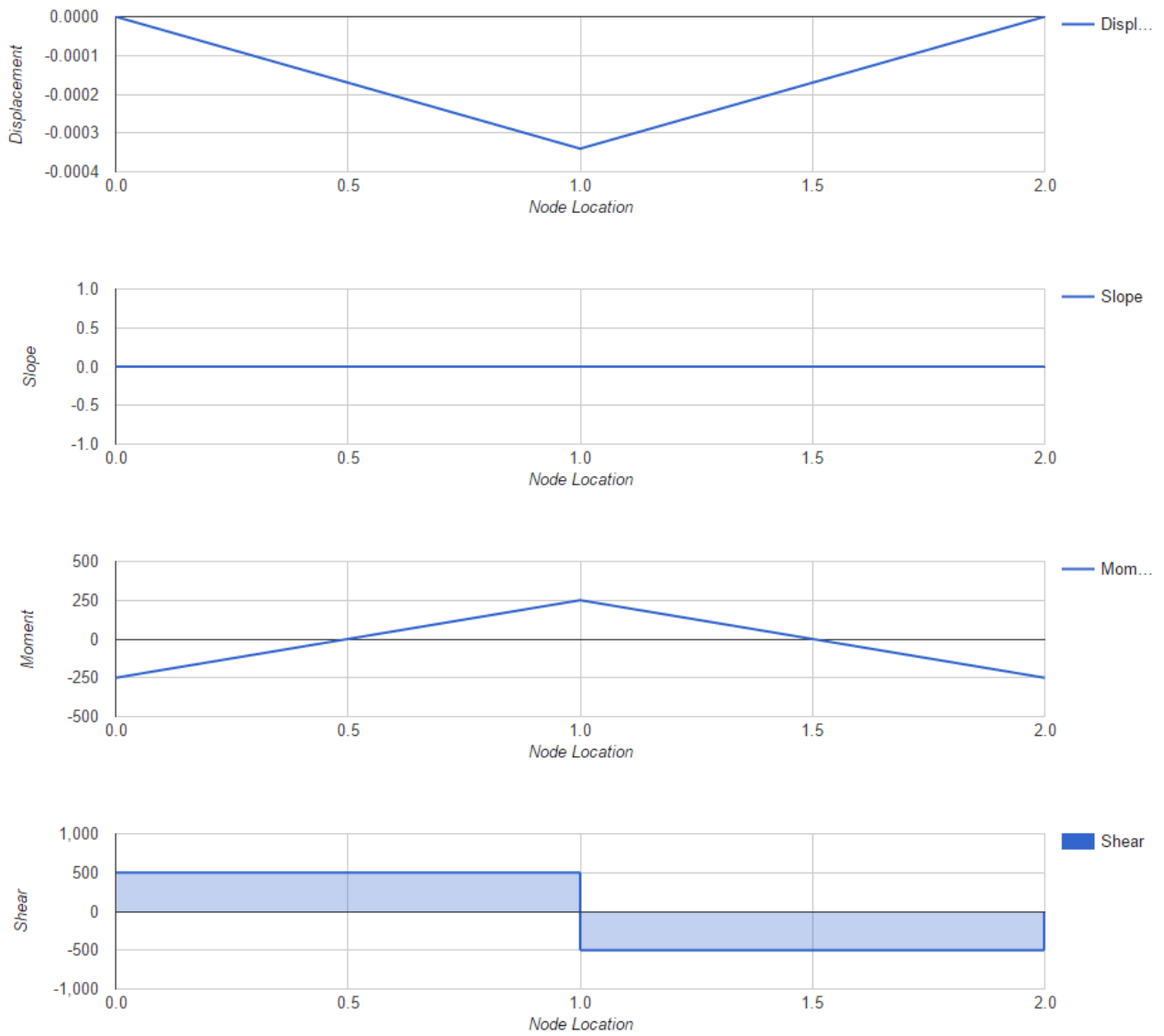


Figure 51 JavaScript Application User Declare Nodes Outputs

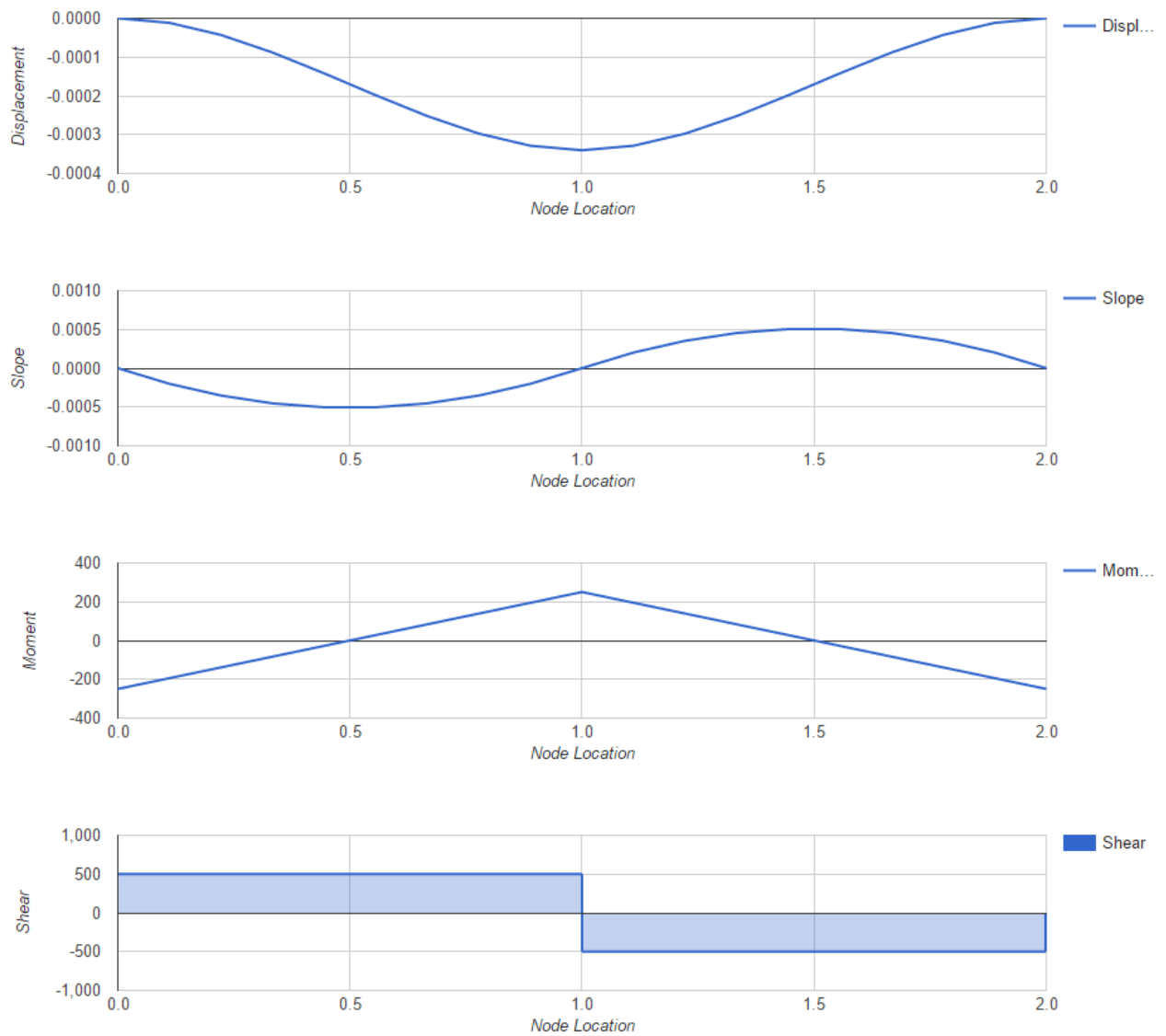


Figure 52 JavaScript Application User Declared Nodes and Undeclared Nodes Outputs

Appendix B- Beam Problem Outputs and Comparisons

Scenario 3

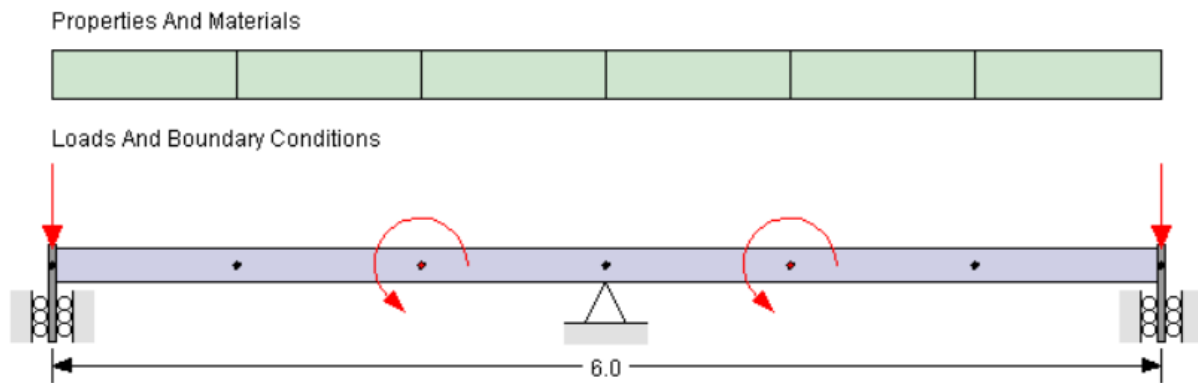


Figure 53 Beam Problem 3 Set Up

Table 22 Beam Problem 3 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist From Neut. Axis)
1	0	r			-500		../..	1.80E-06	6.80E+10	.5
2	1						../..			
3	2					-100	../..			
4	3	s					../..			
5	4					100	../..			
6	5						../..			
7	6	r			-500		../..			

Table 23 Beam Problem 3 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	-8.37E-03	-8.37E-03	-8.37E-03	0.00E+00	0.00E+00	0.00E+00
2	-6.13E-03	-6.13E-03	-6.13E-03	3.81E-03	3.81E-03	3.81E-03
3	-2.11E-03	-2.11E-03	-2.11E-03	3.54E-03	3.54E-03	3.54E-03
4	0.00E+00	0.00E+00	0.00E+00	-1.34E-19	0.00E+00	3.73E-17
5	-2.11E-03	-2.11E-03	-2.11E-03	-3.54E-03	-3.54E-03	-3.54E-03
6	-6.13E-03	-6.13E-03	-6.13E-03	-3.81E-03	-3.81E-03	-3.81E-03
7	-8.37E-03	-8.37E-03	-8.37E-03	0.00E+00	0.00E+00	0.00E+00

Table 24 Beam Problem 3 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0.002388288	0.002185288	0	0
2	0.008159935	0.00736032	0.010491528	0.0095472
3	0.018951957	0.020563204	0.008473858	0.008615077
4	0	0	0	0
5	0.018951957	0.020563204	0.008473858	0.008615077
6	0.008159935	0.00736032	0.010491528	0.0095472
7	0.002388288	0.002185288	0	0

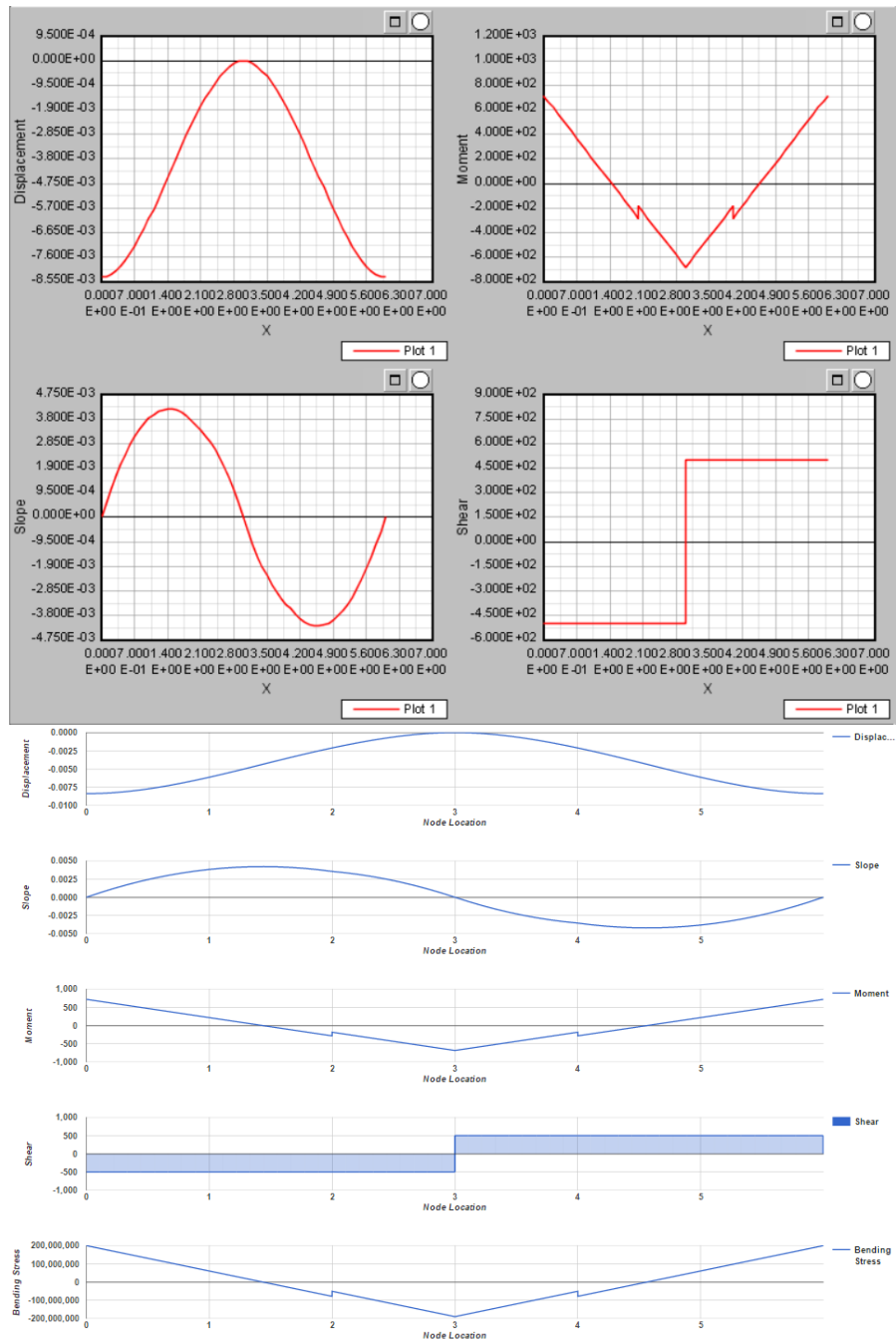
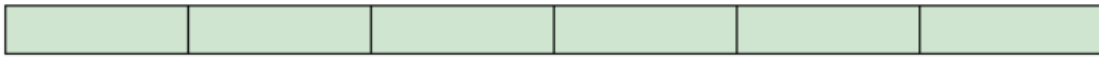


Figure 54 Beam Problem 3 ETBX & JavaScript Application Output Diagrams

Scenario 5

Properties And Materials



Loads And Boundary Conditions

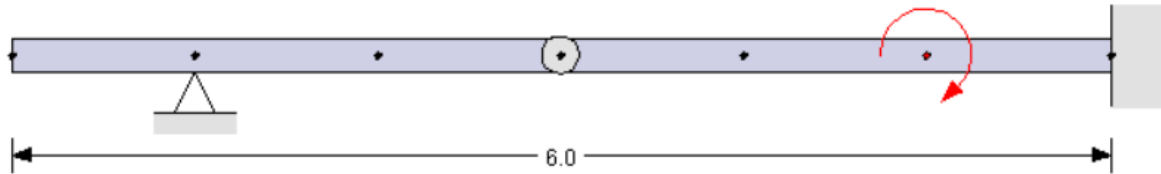


Figure 55 Beam Problem 5 Set Up

Table 25 Beam Problem 5 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0						../..	1.80E-06	6.80E+10	.5
2	1	s					../..			
3	2						../..			
4	3	h					../..			
5	4						../..			
6	5					-500	../..			
7	6	c					../..			

Table 26 Beam Problem 5 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	-5.11E-03	-5.11E-03	-5.11E-03	5.11E-03	5.11E-03	5.11E-03
2	0.00E+00	0.00E+00	0.00E+00	5.11E-03	5.11E-03	5.11E-03
3	5.11E-03	5.11E-03	5.11E-03	5.11E-03	5.11E-03	5.11E-03
4	1.02E-02	1.02E-02	1.02E-02	L:0.51063E-02 R:-0.40850E-02	L:5.106e-3 R:-4.085e-3	5.11E-03
5	6.13E-03	6.13E-03	6.13E-03	-4.09E-03	-4.09E-03	-4.08E-03
6	2.04E-03	2.04E-03	2.04E-03	-4.09E-03	-4.09E-03	-4.08E-03
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Table 27 Beam Problem 5 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0.005875095	0.004093056	0.005875095	0.004093056
2	0	0	0.005875095	0.004093056
3	0.005875095	0.004093056	0.005875095	0.004093056
4	0.029374327	0.023696636	L:0.00587509547030047 R:0	0.004093056
5	0.008159935	0.00736032	0	0.00080784
6	0.024479804	0.023696636	0	0.00080784
7	0	0	0	0

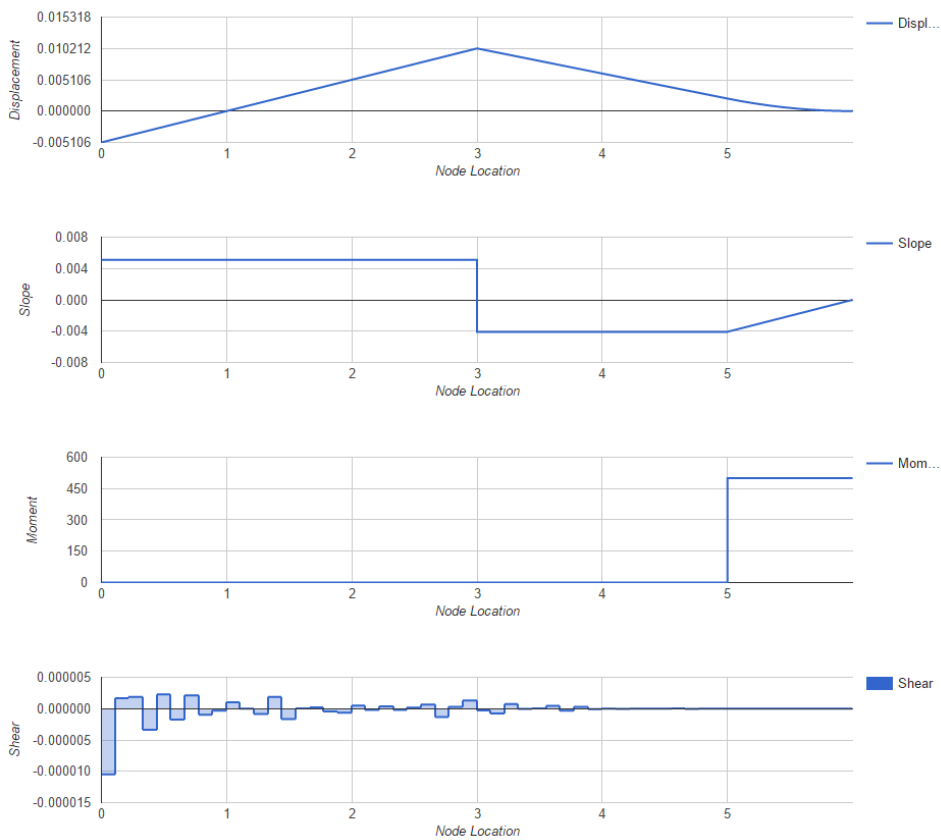
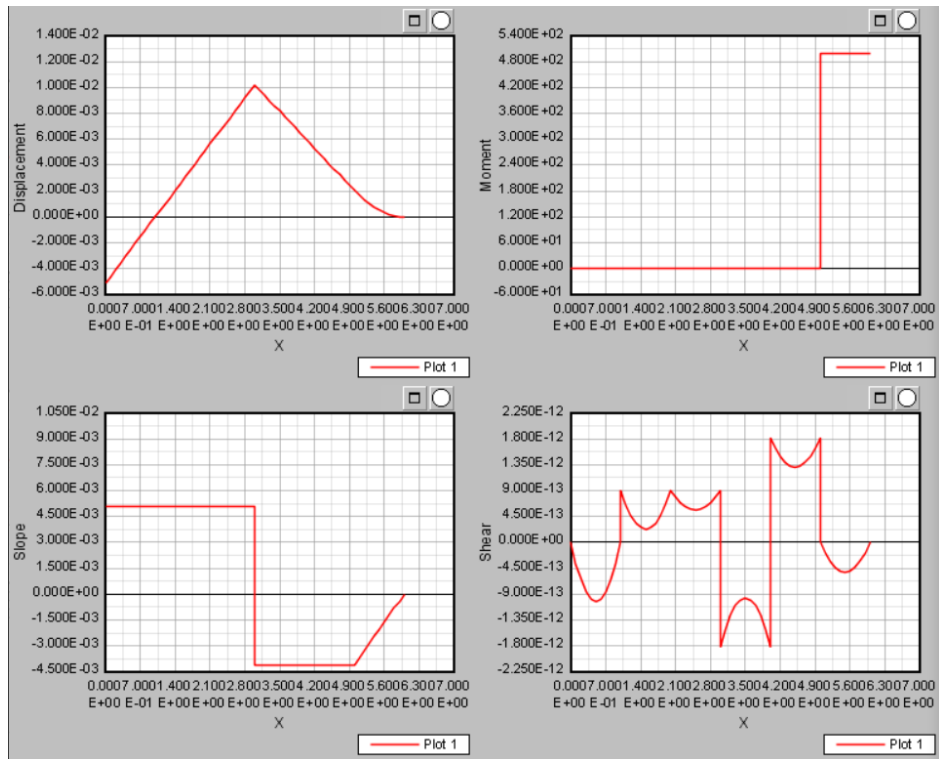
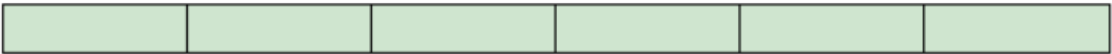


Figure 56 Problem 5 ETBX & JavaScript Application Output Diagrams

Scenario 6

Properties And Materials



Loads And Boundary Conditions

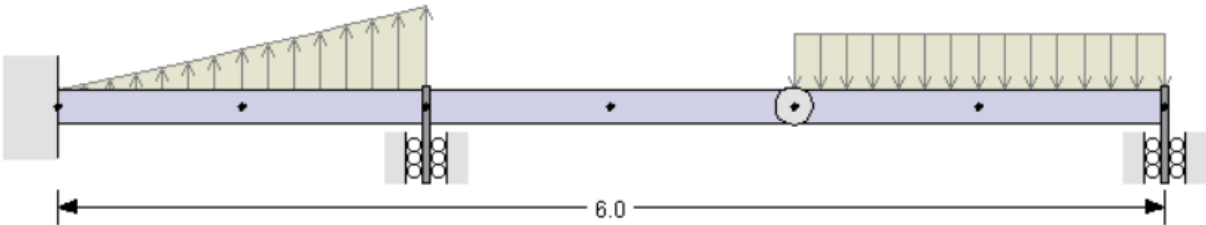


Figure 57 Beam Problem 6 Set Up

Table 28 Beam Problem 6 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0	c					0/..	1.80E-06	6.80E+10	.5
2	1						../..			
3	2	r					../-300			
4	3						../..			
5	4	h					200			
6	5						../..			
7	6	r					200			

Table 29 Beam Problem 6 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	-4.27E-04	-4.66E-04	-4.66E-04	-8.17E-04	-7.66E-04	-7.66E-04
3	-8.76E-04	-1.04E-03	-1.03E-03	0.00E+00	0.00E+00	0.00E+00
4	-3.39E-03	-3.76E-03	-3.76E-03	-4.90E-03	-4.90E-03	-4.90E-03
5	-9.16E-03	-9.75E-03	-9.75E-03	L: -0.65360E-02 R: -0.40850E-02	L: -6.539e-3 R: -4.354e-3	-6.54E-03
6	-1.27E-02	-1.36E-02	-1.36E-02	-2.86E-03	-2.99E-03	-3.00E-03
7	-1.41E-02	-1.52E-02	-1.52E-02	0.00E+00	0.00E+00	0.00E+00

Table 30 Beam Problem 6 Node Displacement Value Comparisons

Node	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/ETBX) (Rotation)
1	0	0
2	0.007054529	0.004099584
3	0.013721689	0
4	0.022085217	0.041595598
5	0.036361007	0.046695598
6	0.001247233	0.054846322
7	0.025796126	0

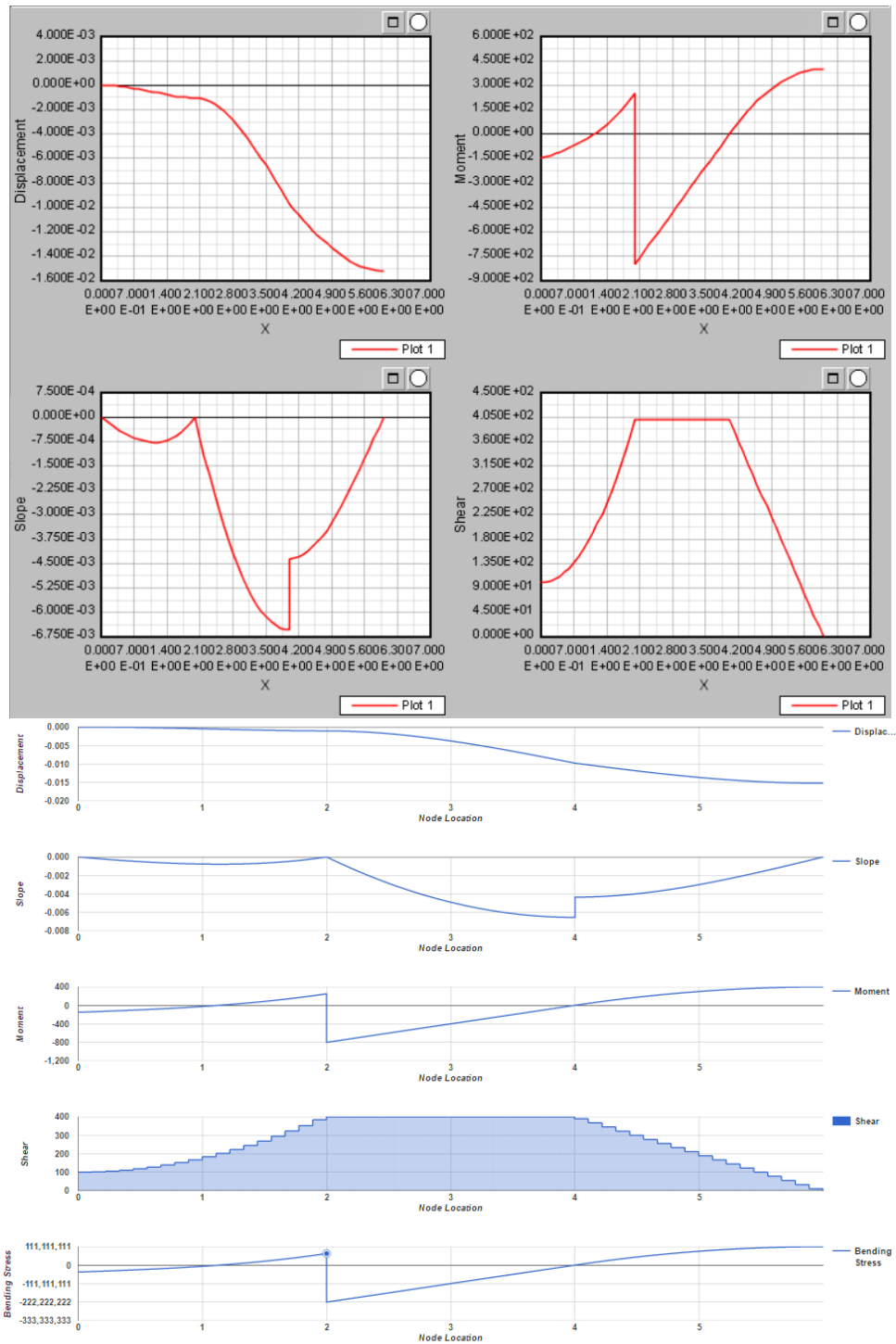


Figure 58 Beam Problem 6 ETBX & JavaScript Application Output Diagrams

Scenario 7

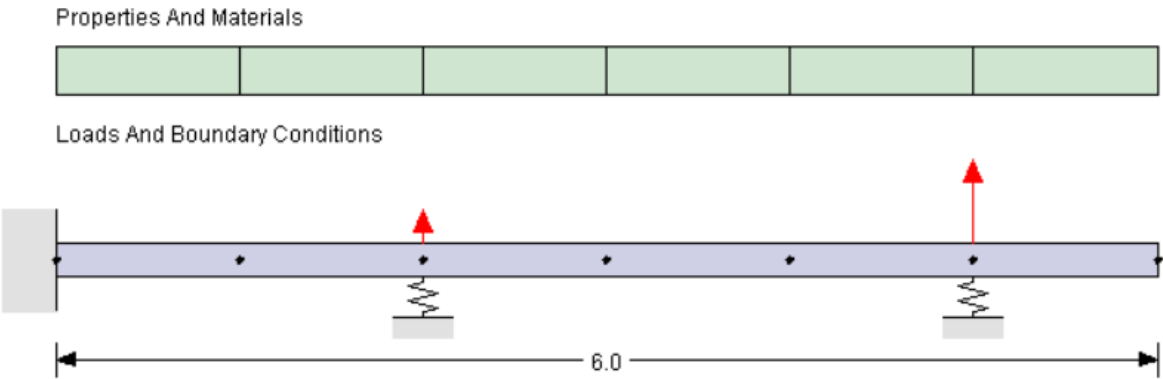


Figure S9 Beam Problem 7 Set Up

Table 31 Beam Problem 7 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	^c (Dist. From Neut. Axis)
1	0	C					../..	1.80E-06	6.80E+10	.5
2	1						../..			
3	2		1000		100		../..			
4	3						../..			
5	4						../..			
6	5		1000		500		../..			
7	6						../..			

Table 32 Beam Problem 7 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	7.31E-03	7.53E-03	7.53E-03	1.45E-02	1.45E-02	1.45E-02
3	2.73E-02	2.77E-02	2.77E-02	2.54E-02	2.53E-02	2.53E-02
4	5.65E-02	5.71E-02	5.71E-02	3.29E-02	3.29E-02	3.29E-02
5	9.18E-02	9.24E-02	9.24E-02	3.75E-02	3.74E-02	3.74E-02
6	1.30E-01	1.31E-01	1.31E-01	3.90E-02	3.89E-02	3.89E-02
7	1.69E-01	1.70E-01	1.70E-01	3.90E-02	3.89E-02	3.89E-02

Table 33 Beam Problem 7 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0	0	0	0
2	2.984217293	0.00355897	0.227696129	0.004218712
3	1.508715395	0.01273611	0.232567307	0.007387838
4	0.955583083	0.000245398	0.230820628	0.008340248
5	0.721462979	0.003787203	0.213561132	0.00733066
6	0.592171037	0.000611625	0.20784686	0.011752474
7	0.414201183	0.009075662	0.20784686	0.011752474

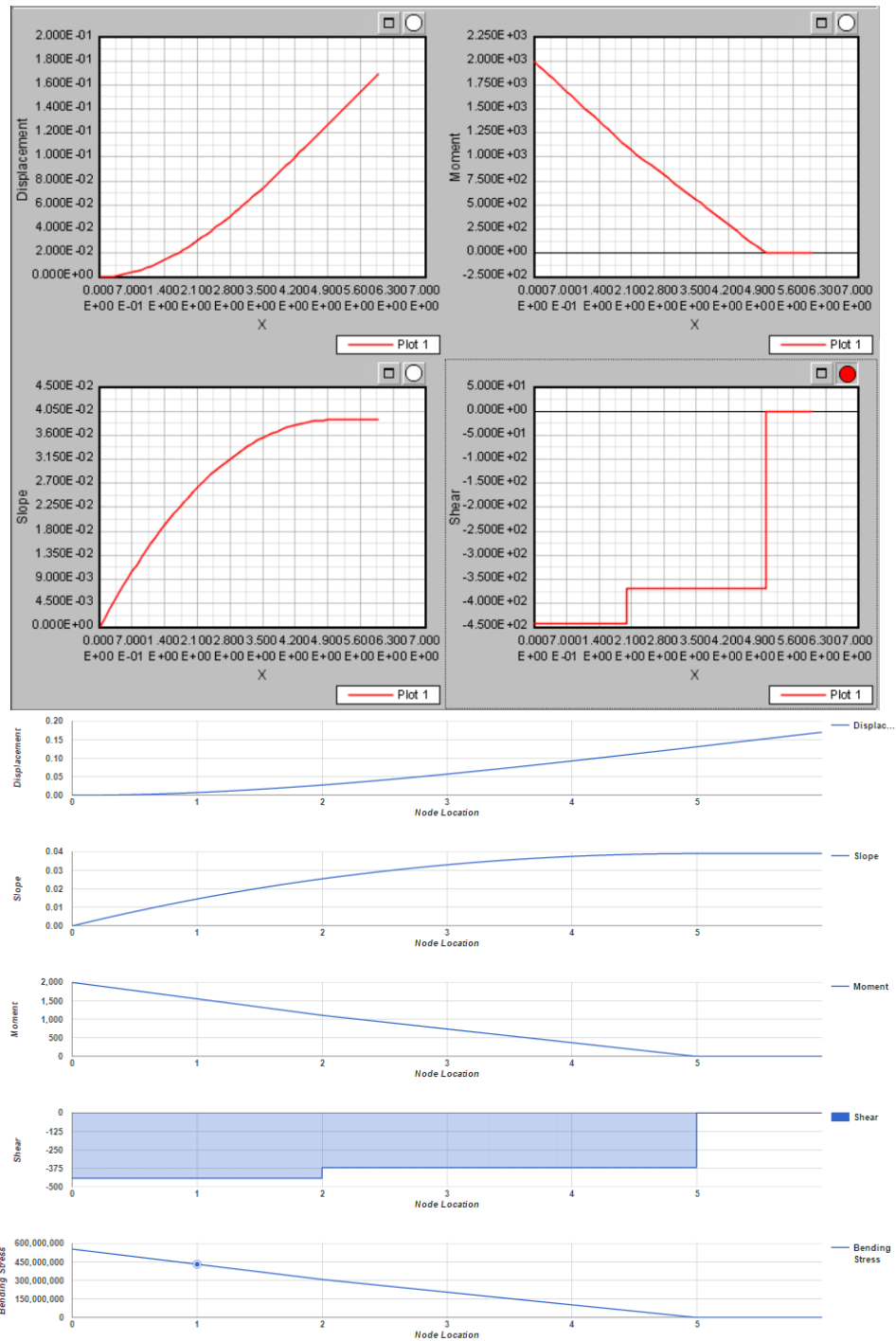


Figure 60 Beam Problem 7 ETBX & JavaScript Application Output Diagrams

Scenario 8

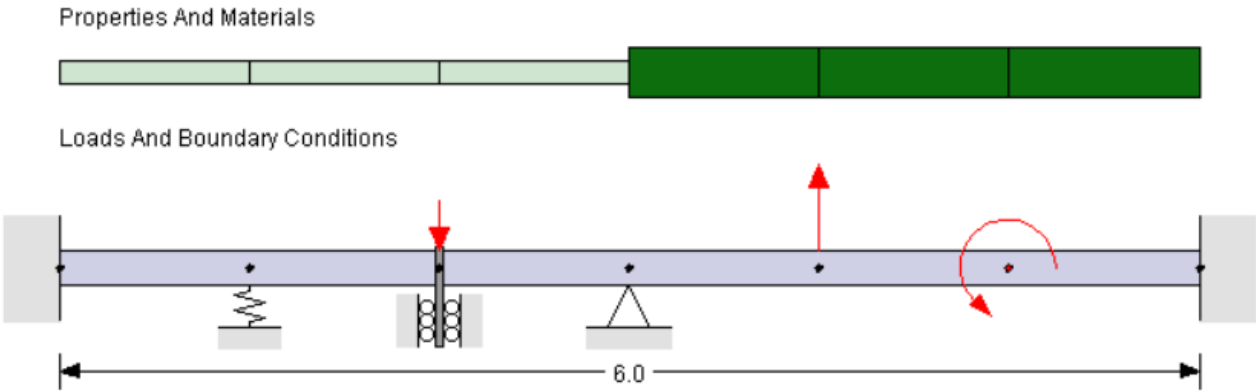


Figure 61 Beam Problem 8 Set Up

Table 34 Beam Problem 8 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0	c					../..	1.80E-06	6.80E+10	.5
2	1		1500				../..			
3	2	r			-300		../..			
4	3	s		1000			../..	3.60E-06	7.70E+10	.25
5	4				700		../..			
6	5					300	../..			
7	6	c					../..			

Table 35 Beam Problem 8 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	-1.66E-04	-2.34E-04	-2.34E-04	-3.11E-04	-3.51E-04	-3.51E-04
3	-3.33E-04	-4.68E-04	-4.68E-04	0.00E+00	0.00E+00	0.00E+00
4	0.00E+00	0.00E+00	0.00E+00	6.00E-04	6.45E-04	6.45E-04
5	3.17E-04	4.36E-04	4.36E-04	1.04E-06	6.68E-05	6.68E-05
6	1.59E-04	2.35E-04	2.35E-04	-2.59E-04	-2.08E-04	-2.08E-04
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Table 36 Beam Problem 8 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0	0	0	0
2	0.001876813	0.014916755	0.000149145	0.00763257
3	0.001450264	0.006600179	6.59974E-05	0
4	0	0	0	0.00350321
5	0.001420324	0.004679753	4.67997E-05	0.006556335
6	0.001106571	0.01636399	0.000163613	0.019053534
7	0	0	0	0

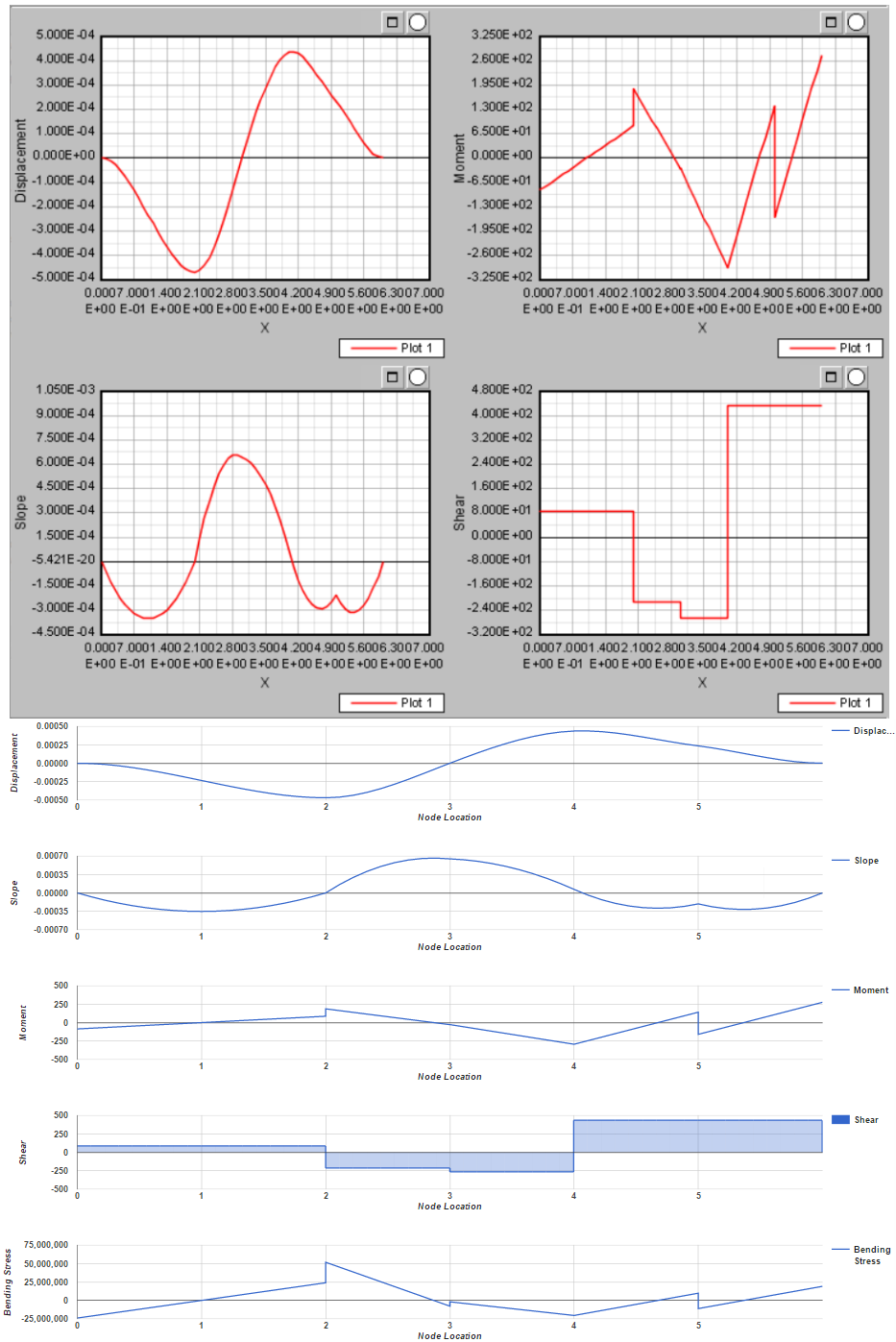


Figure 62 Beam Problem 8 ETBX & JavaScript Application Output Diagrams

Scenario 9

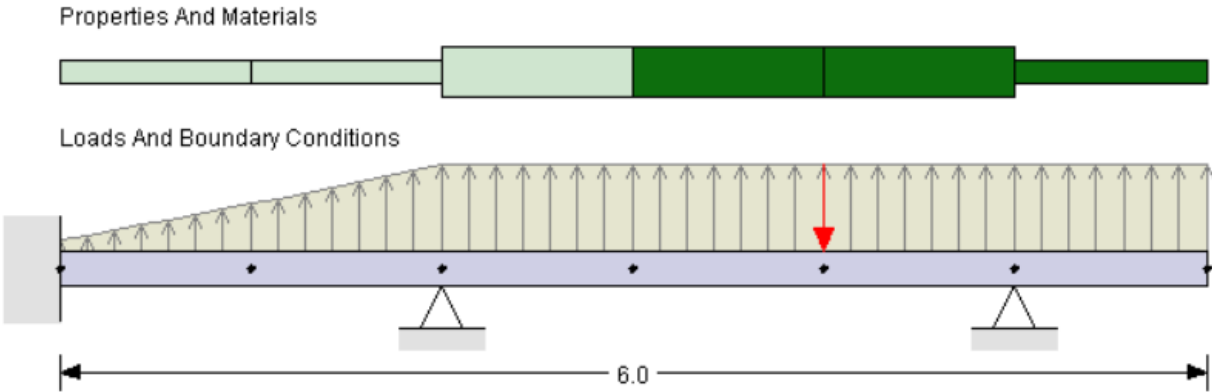


Figure 63 Beam Problem 9 Set Up

Table 37 Beam Problem 9 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut Axis)
1	0	c					-100/..	1.80E-06	6.80E+10	.5
2	1						../..			
3	2	s					../-700, -700/..	3.60E-06		
4	3						../..		7.70E+10	.25
5	4				-500		../..			
6	5	s					../..	1.80E-06		
7	6						../-700			

Table 38 Beam Problem 9 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	2.57E-04	1.91E-04	1.91E-04	4.96E-04	3.06E-04	3.06E-04
3	0.00E+00	0.00E+00	0.00E+00	-1.15E-03	-1.31E-03	-1.31E-03
4	-1.18E-03	-1.54E-03	-1.54E-03	-1.09E-03	-1.15E-03	-1.15E-03
5	-1.27E-03	-1.67E-03	-1.67E-03	9.31E-04	1.00E-03	1.00E-03
6	0.00E+00	0.00E+00	0.00E+00	1.48E-03	1.74E-03	1.74E-03
7	8.07E-04	1.11E-03	1.11E-03	2.21E-04	9.00E-04	9.00E-04

Table 39 Beam Problem 9 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0	0	0	0
2	0.000157373	0.013744176	9.79208E-07	0.010804039
3	0	0	1.52999E-06	0.01277572
4	0.000195224	0.016463348	2.61256E-06	0.025080816
5	0.000179501	0.015257995	1.99164E-06	0.019816789
6	0	0	1.72246E-06	0.015157291
7	0.000360231	0.038093591	1.11112E-07	0.001644471

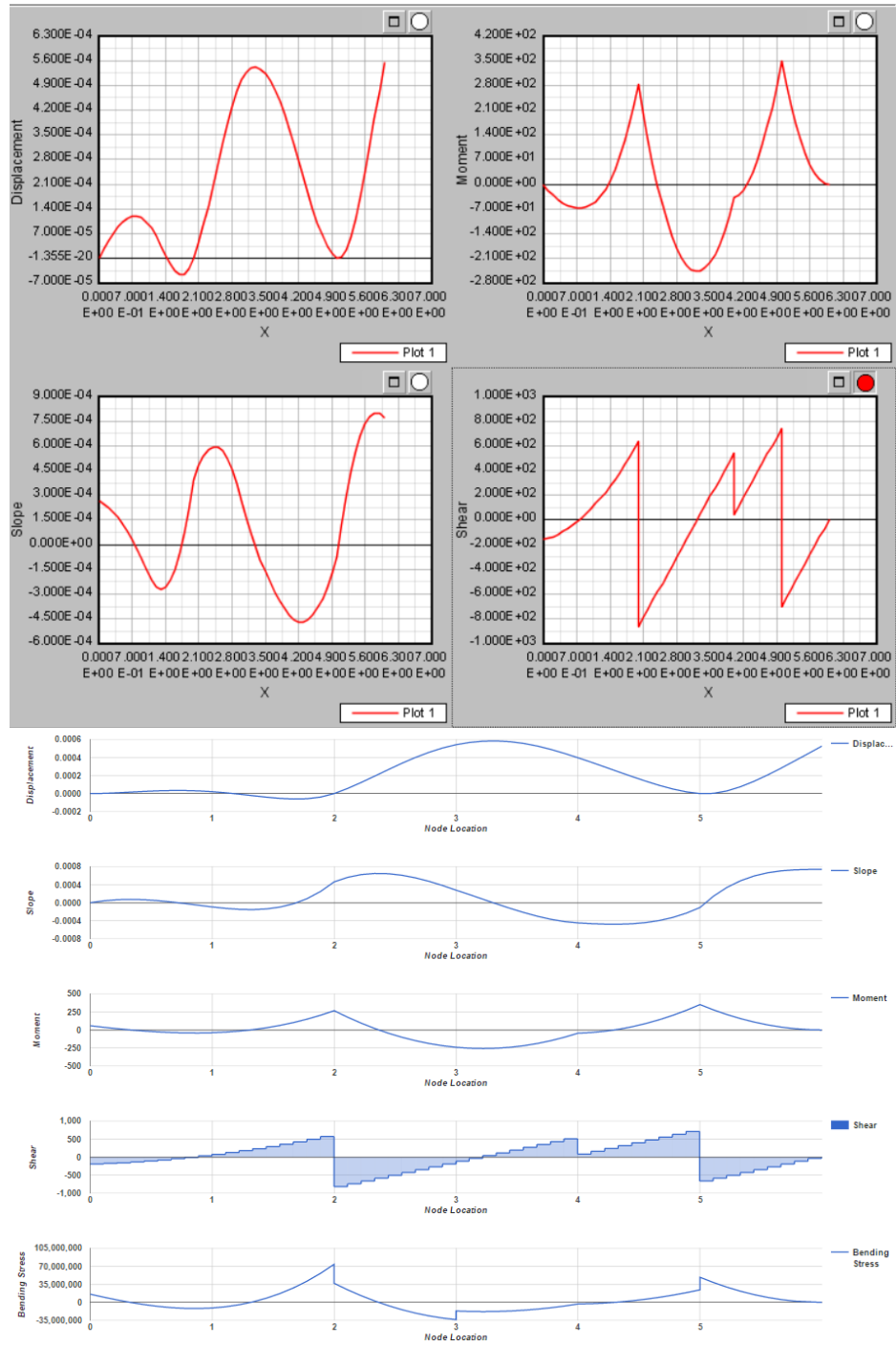


Figure 64 Beam Problem 9 ETBX & JavaScript Application Output Diagrams

Scenario 10

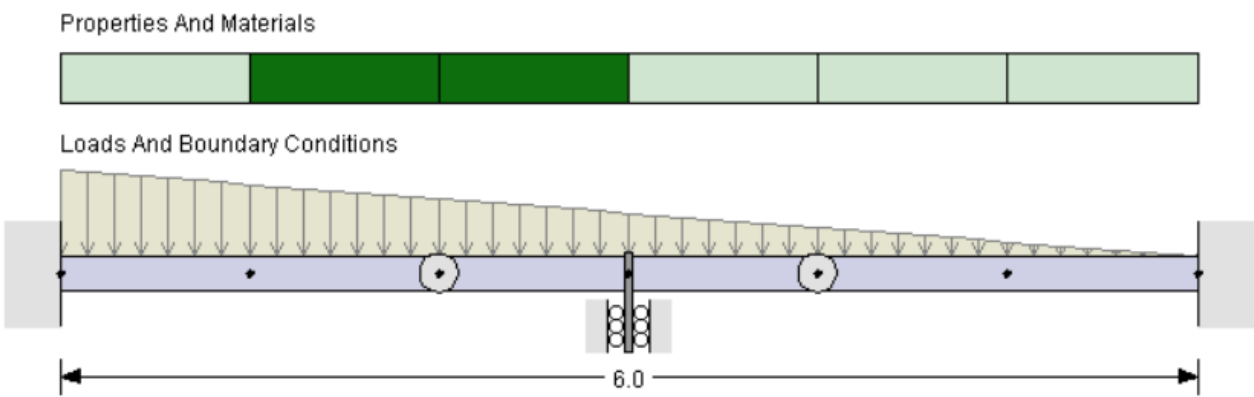


Figure 65 Beam Problem 10 Set Up

Table 40 Beam Problem 10 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0	c					1500	3.60E-06	6.80E+10	.5
2	1						../..		7.70E+10	
3	2	h					../..			
4	3	r					../..		6.80E+10	
5	4	h					../..			
6	5						../..			
7	6	c					../0			

Table 41 Beam Problem 10 Node Displacement Values

Node	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	-5.05E-03	-5.04E-03	-8.53E-03	-8.52E-03
3	-1.47E-02	-1.47E-02	L:-1.009e-2 R:-3.506e-4	-1.01E-02
4	-1.49E-02	-1.49E-02	0.00E+00	0.00E+00
5	-1.38E-02	-1.38E-02	L:1.639e-3 R:1.014e-2	1.64E-03
6	-4.41E-03	-4.40E-03	7.82E-03	7.82E-03
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Table 42 Beam Problem 10 Node Displacement Value Comparisons

Node	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/ETBX) (Rotation)
1	0	0
2	0.033787428	0.034518127
3	0.071967229	0.098908436
4	0.062825448	0
5	0.06779956	0.127841238
6	0.041083841	0.040796688
7	0	0

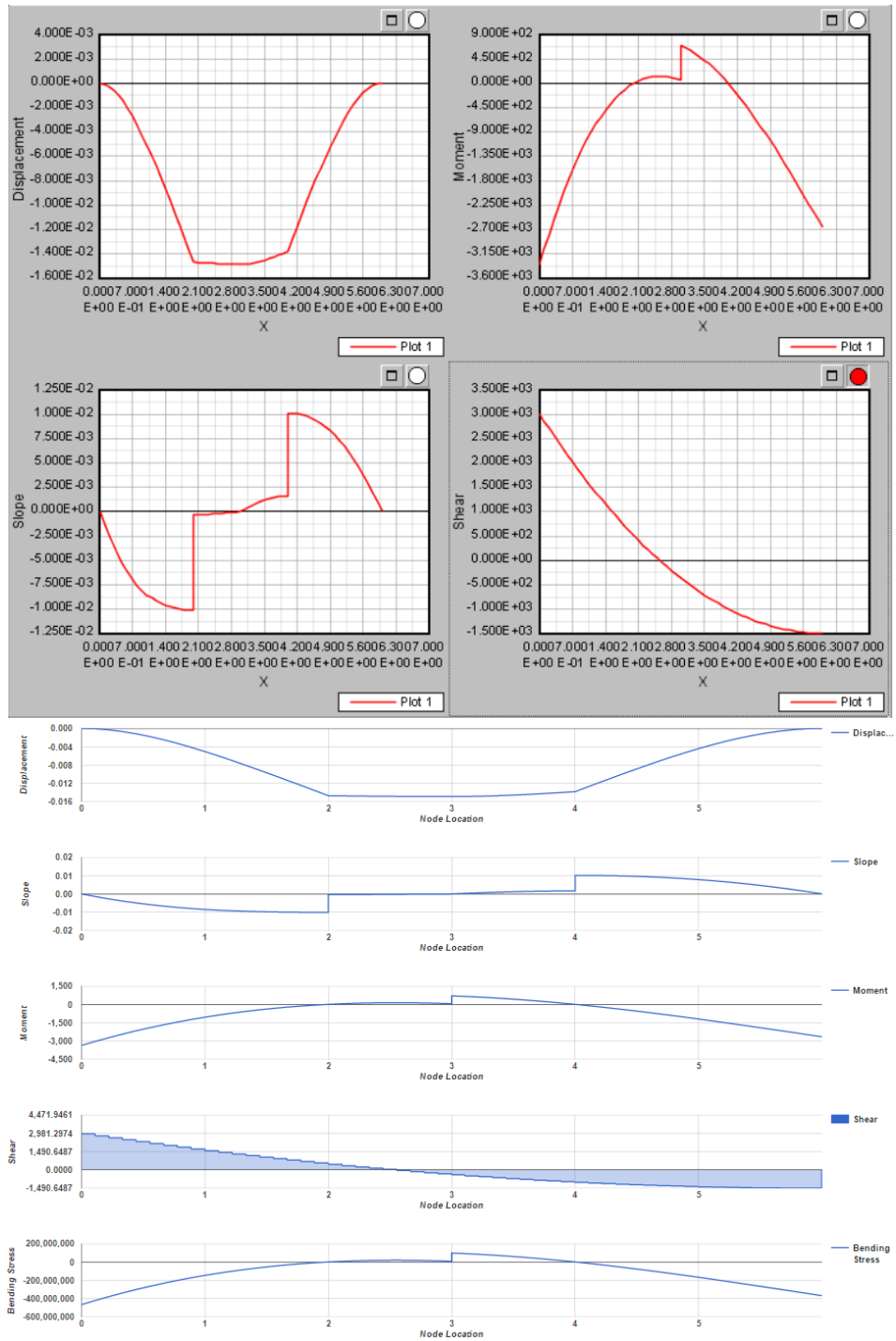


Figure 66 Beam Problem 10 ETBX & JavaScript Application Output Diagrams

Scenario 11

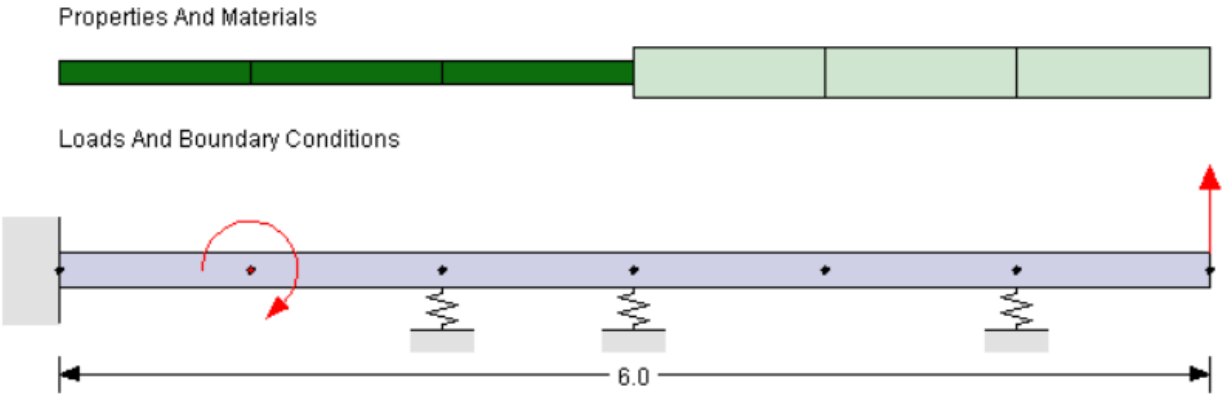


Figure 67 Beam Problem 11 Set Up

Table 43 Beam Problem 11 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist From Neut Axis)
1	0	c					../..	1.80E-06	7.70E+10	1
2	1					-500	../..			
3	2		500				../..			
4	3		750				../..	3.60E-06	6.80E+10	.5
5	4						../..			
6	5		1500				../..			
7	6				700		../..			

Table 44 Beam Problem 11 Node Displacement Values

Node	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	7.72E-03	7.72E-03	1.50E-02	1.50E-02
3	3.08E-02	3.08E-02	3.08E-02	3.08E-02
4	6.83E-02	6.83E-02	4.38E-02	4.38E-02
5	1.15E-01	1.15E-01	4.94E-02	4.94E-02
6	1.67E-01	1.66E-01	5.32E-02	5.32E-02
7	2.21E-01	2.21E-01	5.46E-02	5.46E-02

Table 45 Beam Problem 11 Node Displacement Value Comparisons

Node	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/ETBX) (Rotation)
1	0	0
2	0.001762602	0.003072718
3	0.006878177	0.002566536
4	0.005297325	0.004775188
5	0.028852806	0.007471706
6	0.002462523	0.008595811
7	0.006572551	0.008846936

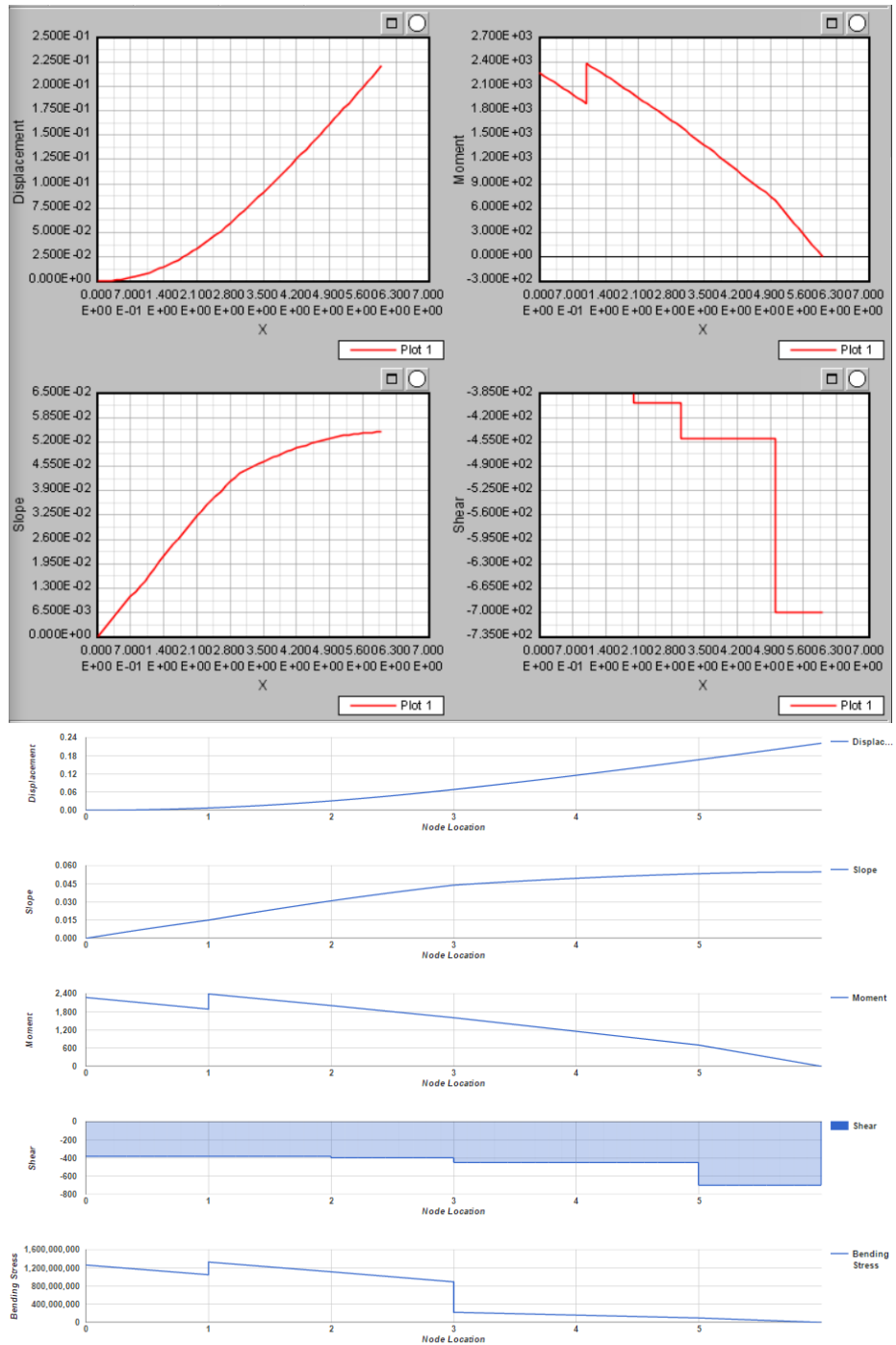


Figure 68 Beam Problem 11 ETBX & JavaScript Application Output Diagrams

Scenario 12

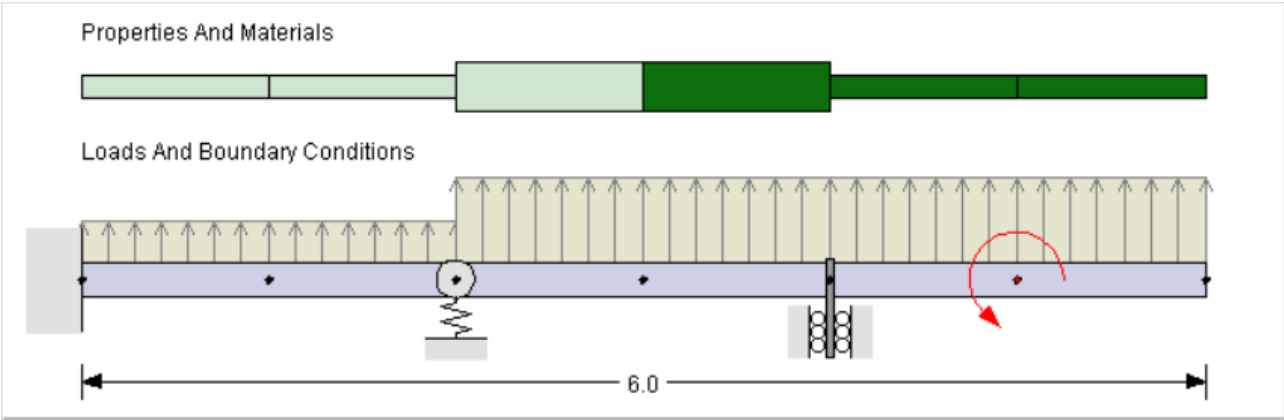


Figure 69 Beam Problem 12 Set Up

Table 46 Beam Problem 12 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0	c					-500/..	1.80E-06	6.80E+10	.1
2	1						../..			
3	2	h	1500				../-500, -1000/-1000	3.60E-06		.5
4	3						../..		7.70E+10	
5	4	r					../..	1.80E-06		
6	5					700	../..			
7	6						../..			

Table 47 Beam Problem 12 Node Displacement Values

Node	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	2.92E-02	2.92E-02	5.21E-02	5.21E-02
3	9.23E-02	9.23E-02	L:6.856e-2 R:2.492e-2	6.85E-02
4	1.15E-01	1.15E-01	1.74E-02	1.74E-02
5	1.24E-01	1.24E-01	0.00E+00	0.00E+00
6	1.32E-01	1.32E-01	1.35E-02	1.35E-02
7	1.46E-01	1.46E-01	1.47E-02	1.47E-02

Table 48 Beam Problem 12 Node Displacement Value Comparisons

Node	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/ETBX) (Rotation)
1	0	0
2	0.020456393	0.02165536
3	0.022123611	0.023546967
4	0.024591922	0.035959526
5	0.047123645	0
6	0.017150133	0.014775754
7	0.03605951	0

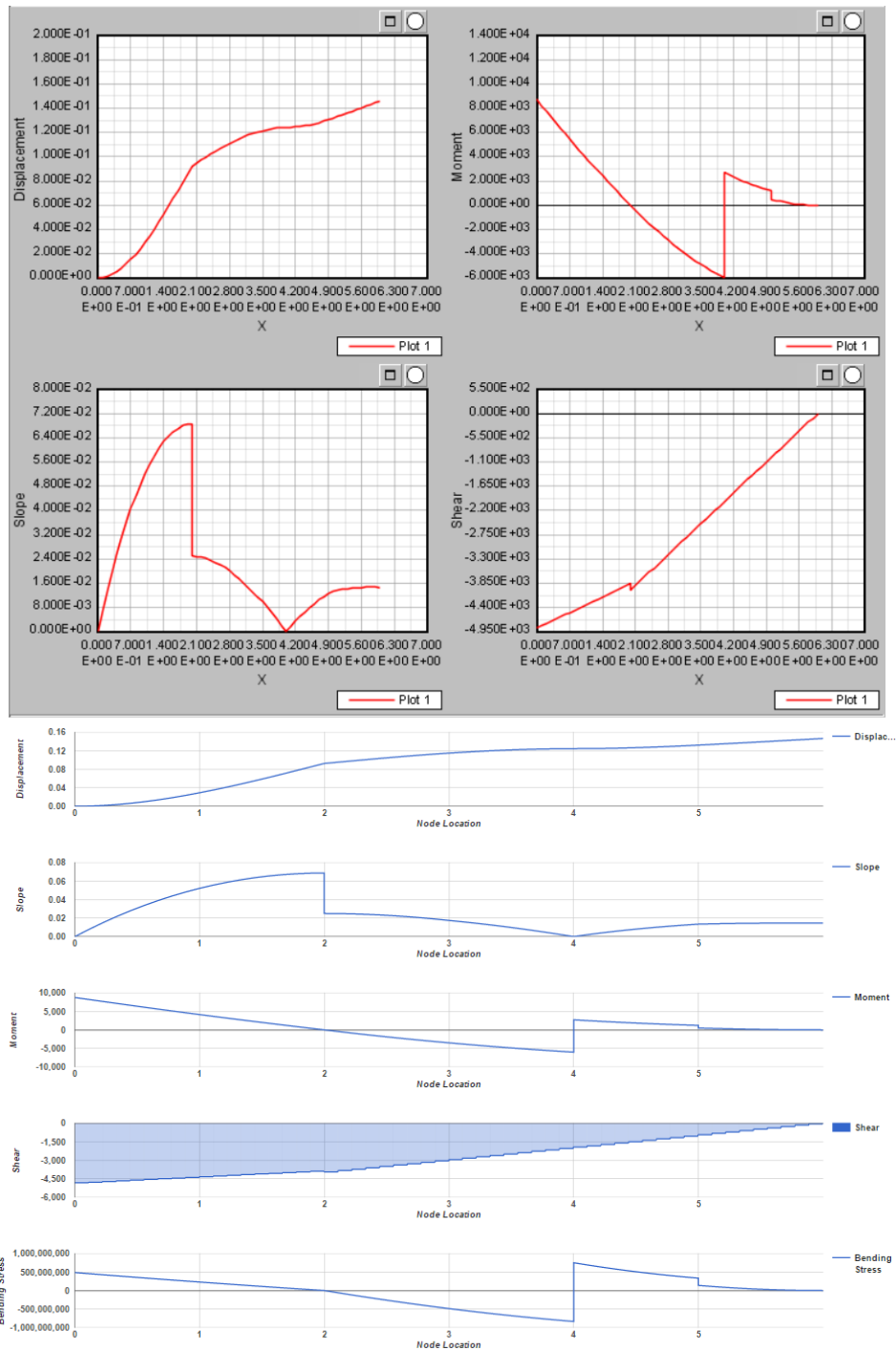


Figure 70 Beam Problem 12ETBX & JavaScript Application Output Diagrams

Scenario 13

Properties And Materials



Loads And Boundary Conditions

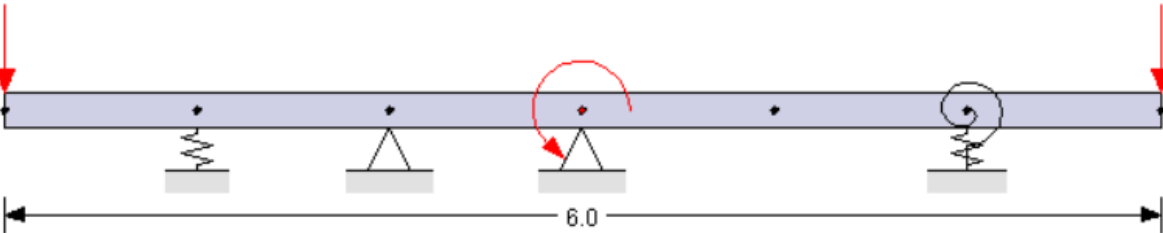


Figure 71 Beam Problem 13 Set Up

Table 49 Beam Problem 13 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	^c (Dist. From Neut. Axis)
1	0				-300		../..	1.80E-06	6.80E+10	.1
2	1		500				../..			
3	2	s		750			../..			
4	3	s		750		500	../..	3.60E-06	7.70E+10	
5	4						../..			
6	5		1000	1000			../..			
7	6				-300		../..			

Table 50 Beam Problem 13 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	-1.09E-02	-1.08E-02	-1.08E-02	7.08E-03	7.03E-03	7.03E-03
2	-4.22E-03	-4.17E-03	-4.17E-03	5.86E-03	5.80E-03	5.80E-03
3	0.00E+00	0.00E+00	0.00E+00	2.18E-03	2.14E-03	2.14E-03
4	0.00E+00	0.00E+00	0.00E+00	-1.91E-03	-1.83E-03	-1.83E-03
5	-3.54E-03	-3.24E-03	-3.24E-03	-4.97E-03	-4.47E-03	-4.47E-03
6	-9.53E-03	-8.59E-03	-8.59E-03	-6.81E-03	-6.06E-03	-6.06E-03
7	-1.67E-02	-1.50E-02	-1.50E-02	-7.42E-03	-6.60E-03	-6.60E-03

Table 51 Beam Problem 13 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0.94556137	0.017791088	0.728751801	0.006758175
2	1.1632039	0.002660665	0.889822548	0.000603133
3	0	0	2.001285229	0.004027941
4	0	0	3.792687405	0.013793081
5	3.26	0.015337154	3.08	0.010488578
6	3.08	0.00413144	2.81	0.007329689
7	2.97	0.006328713	2.60	0.004819144

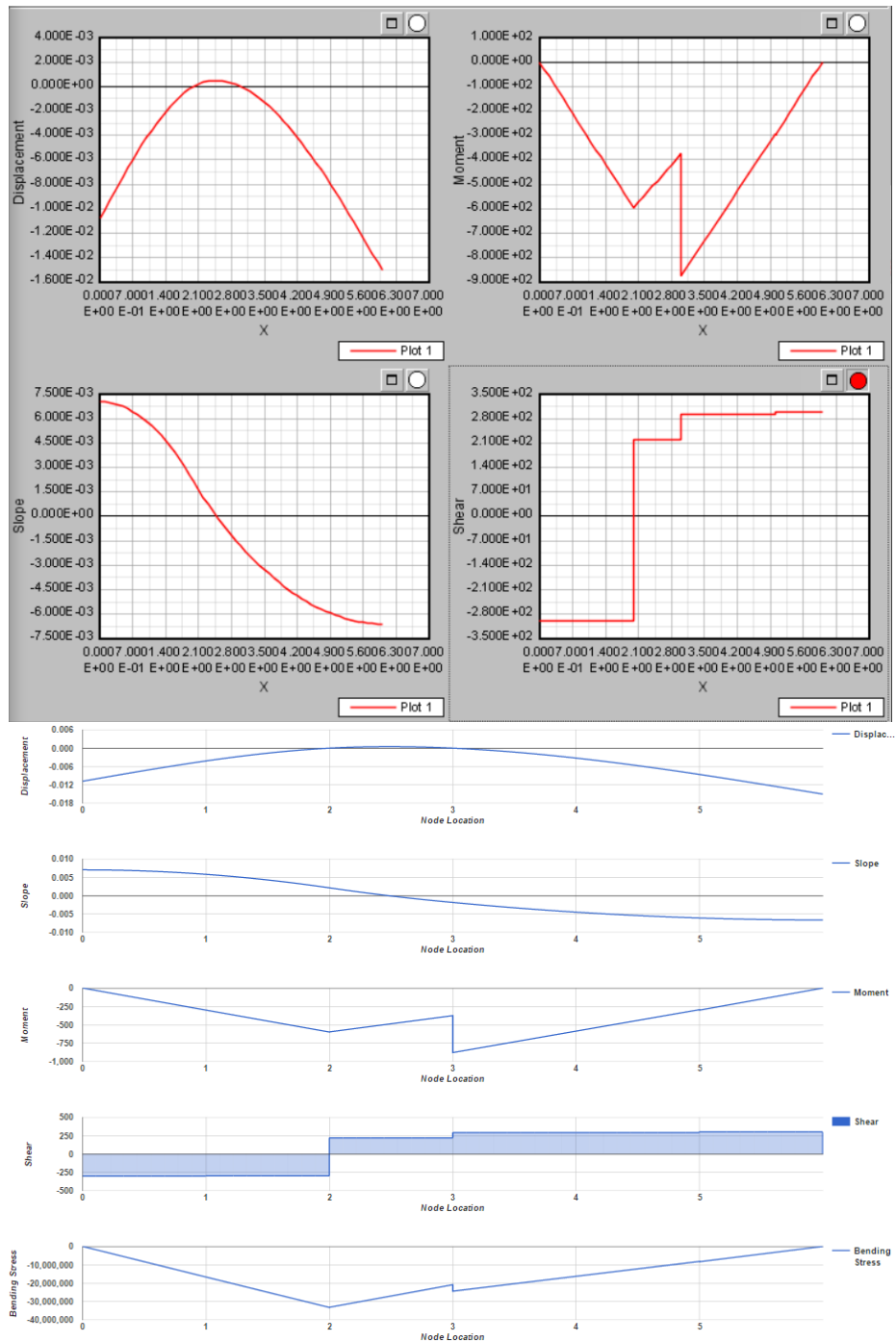


Figure 72 Beam Problem 13 ETBX & JavaScript Application Output Diagrams

Scenario 14

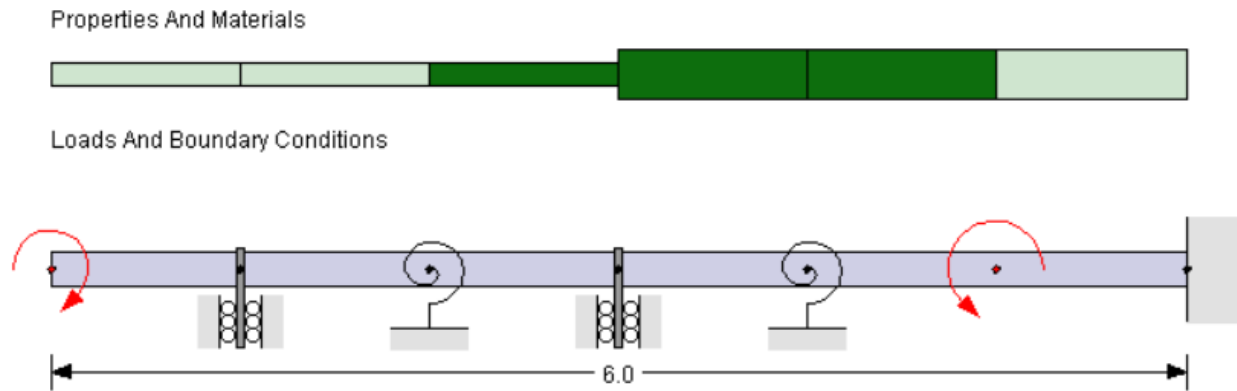


Figure 73 Beam Problem 14 Set Up

Table 52 Beam Problem 14 Input Format

Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m ⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0					-750	../..	1.80E-06	6.80E+10	.1
2	1	r					../..			
3	2			500			../..		7.70E+09	.25
4	3	r					../..	3.60E-06		
5	4			1000			../..			.5
6	5					1000	../..		6.80E+09	
7	6	c					../..			

Table 53 Beam Problem 14 Node Displacement Values

Node	Displacement (m) (ANSYS)	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (ANSYS)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	-8.44E-04	-8.44E-04	-8.44E-04	-6.13E-03	-6.13E-03	-6.13E-03
2	-3.91E-03	-3.91E-03	-3.91E-03	0.00E+00	0.00E+00	0.00E+00
3	-3.91E-03	-3.91E-03	-3.91E-03	1.76E-17	0.00E+00	0.00E+00
4	-3.91E-03	-3.91E-03	-3.91E-03	0.00E+00	0.00E+00	0.00E+00
5	-3.26E-03	-3.26E-03	-3.26E-03	1.30E-03	1.30E-03	1.30E-03
6	-1.30E-03	-1.30E-03	-1.30E-03	2.61E-03	2.61E-03	2.61E-03
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Table 54 Beam Problem 14 Node Displacement Value Comparisons

Node	%Error (JavaScript/Ansys) (Displacement)	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/Ansys) (Rotation)	%Error (JavaScript/ETBX) (Rotation)
1	0.002370455	0.00547558	0.008159935	0.00736032
2	0.012795905	0.012079421	0	0
3	0.012795905	0.012079421	0	0
4	0.012795905	0.012079421	0	0
5	0	0.000276327	0.007686986	0.005688256
6	0.023018492	0.020947261	0.019182812	0.017456058
7	0	0	0	0

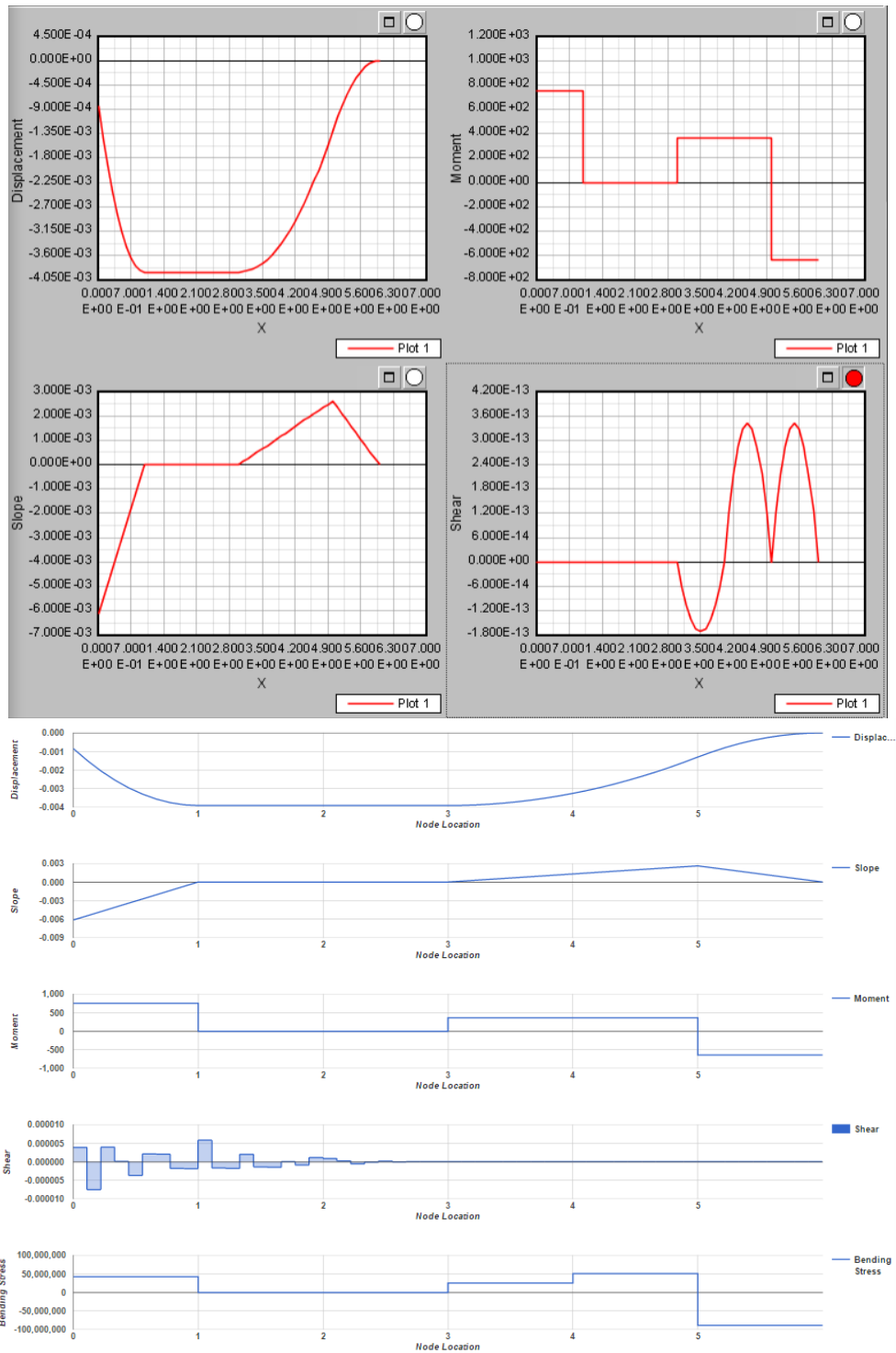


Figure 74 Beam Problem 14 ETBX & JavaScript Application Output Diagrams

Scenario 15

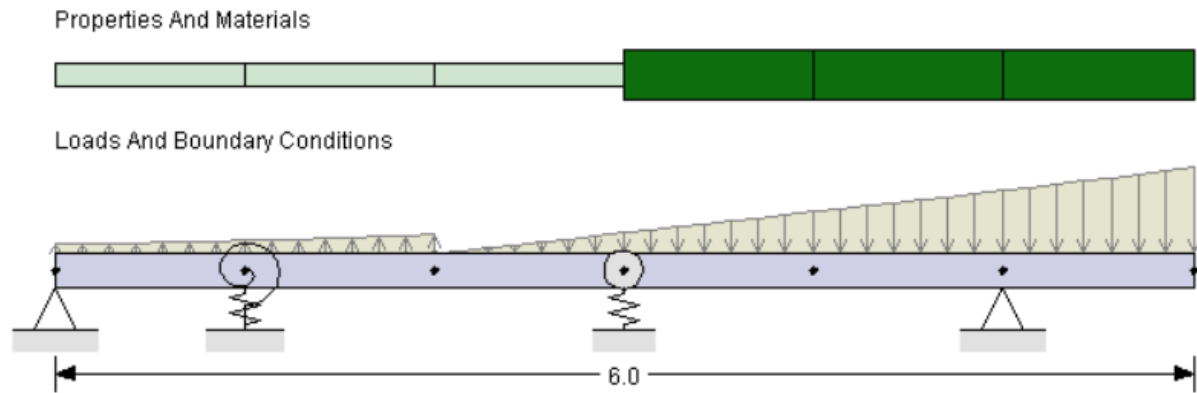


Figure 75 Beam Problem 15 Set Up

Table 55 Beam Problem 15 Input Format

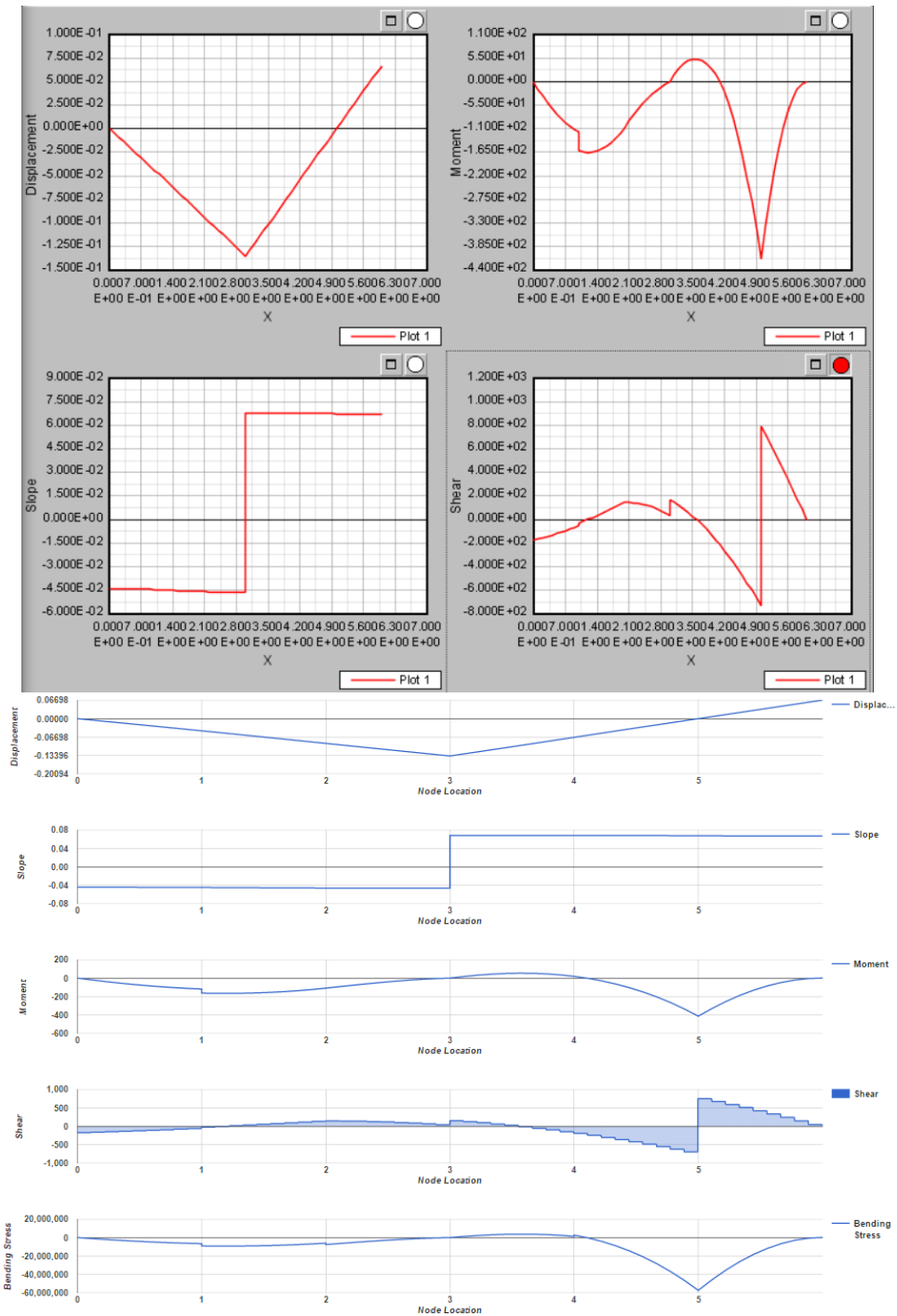
Node	Node Location (m)	Boundary	Translational Spring (N/m)	Rotational Spring (N-m/rad)	Applied Force (N)	Applied Moment (N-m)	Distributed Load (N/m)	Inertia (m⁴)	Youngs Mod (Pa)	c (Dist. From Neut. Axis)
1	0	s					-100/..	1.80E-06	6.80E+10	.1
2	1		500	1000			../..			
3	2						../-200,0/..	3.60E-06	7.70E+10	.25
4	3	h	1000				../..			
5	4						../..			.5
6	5	s					../..			
7	6						../900			

Table 56 Beam Problem 15 Node Displacement Values

Node	Displacement (m) (JavaScript)	Displacement (m) (ETBX)	Rotation (rad) (JavaScript)	Rotation (rad) (ETBX)
1	0.00E+00	0.00E+00	-4.40E-02	-4.40E-02
2	-4.42E-02	-4.42E-02	-4.46E-02	-4.46E-02
3	-8.95E-02	-8.94E-02	-4.58E-02	-4.58E-02
4	-1.36E-01	-1.35E-01	L:-4.620e-2 R:6.777e-2	-4.61E-02
5	-6.78E-02	-6.77E-02	6.79E-02	6.78E-02
6	0.00E+00	0.00E+00	6.74E-02	6.73E-02
7	6.70E-02	6.69E-02	6.69E-02	6.68E-02

Table 57 Beam Problem 15 Node Displacement Value Comparisons

Node	%Error (JavaScript/ETBX) (Displacement)	%Error (JavaScript/ETBX) (Rotation)
1	0	0.125884883
2	0.119491617	0.131331186
3	0.125632498	0.137864324
4	0.130036131	0.123941303
5	0.120999242	0.13248275
6	0	0.135213008
7	0.134169922	0.132211661



76 Beam Problem 15 ETBX & JavaScript Application Output Diagrams