IMPROVEMENT OF COMPUTATIONAL SOFTWARE FOR COMPOSITE CURVED BRIDGE ANALYSIS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET SERHAT KALAYCI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CIVIL ENGINEERING

JANUARY 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Erdal Çokca
Department Head

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Cem Topkaya
Supervisor

Examining Committee Members

Prof. Dr. Polat Gülkan	(METU,CE)	
Asst. Prof. Dr. Cem Topkaya	(METU,CE)	
Prof. Dr. M. Semih Yücemen	(METU,CE)	
Prof. Dr. S. Tanvir Wasti	(METU,CE)	
Volkan Aydoğan(M.S.)	(PROYA)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : AHMET SERHAT KALAYCI

Signature :

ABSTRACT

IMPROVEMENT OF COMPUTATIONAL SOFTWARE FOR COMPOSITE CURVED BRIDGE ANALYSIS

Kalaycı, Ahmet Serhat
M.S., Department of Civil Engineering
Supervisor : Asst. Prof. Dr. Cem Topkaya

January 2005, 157 pages

In highway bridge construction, composite curved girder bridges are becoming more popular recently. Reduced construction time, long span coverage, economics and aesthetics make them more popular than the other structural systems. Although there exist some methods for the analysis of such systems, each have shortcomings. The use of Finite Element Method (FEM) among these methods is limited except in the academic environments. The use of commercial FEM software packages in the analysis of such systems is cumbersome as it takes too much time to form a model. Considering such problems a computational software was developed called UTRAP in 2002 which analyzes bridges for construction loads by taking into account the early age deck concrete. As the topic of this thesis work, this program was restructured and new features were added. In the following thesis work, the program structure, modeling considerations and recommendations are discussed together with the parametric studies.

Keywords: Composite Curved Bridge, Finite Element Method, Software, Parametric Studies

ÖZ

KAVİSLİ KOMPOZİT KÖPRÜ ANALİZİ İÇİN BİR BİLGİSAYAR PROGRAMI GELİŞTİRİLMESİ

Kalaycı, Ahmet Serhat
Yüksek Lisans., İnşaat Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Cem Topkaya

Ocak 2005, 157 sayfa

Günümüzde karayolu köprü inşaatlarında kompozit köprüler daha popüler bir hale gelmiştir. İnşaat süresinin kısalması, uzun açıklıkları geçebilme, ekonomi ve estetik gibi etkenler bunları diğer yapısal sistemlere göre daha yaygın bir hale getirmiştir. Bu tip sistemlerin analizi için çeşitli yöntemler olsa da her birinin kendine özgü kısıtlamaları vardır. Anılan yöntemlerin içinde Sonlu Elemanlar Yönteminin kullanımı ise akademik çevreler dışında yaygın değildir. Bu tip yapısal sistemlerin çözümünde ticari Sonlu Eleman Yöntemi yazılımlarının kullanımı ise modellerin oluşturulması uzun zaman aldığı için zordur. Bu tip sorunların çözümü için 2002 yılında UTRAP adında, inşaat yükleri altında ve yeni dökülen tabliye betonunun çelik kirişler üzerindeki etkilerini de modelleyebilen bir program geliştirilmişir. Bu program yeniden düzenlenmiş ve yeni özellikler eklenmiştir. Bu tez çalışmasında program yapısı, modelleme tekniği ve tavsiyeleri ile beraber yapılan parametrik çalışmalar da açıklanmaktadır.

Anahtar Kelimeler: Kompozit Kavisli Köprü, Sonlu Elemenlar Yöntemi, Yazılım, Parametrik Çalışmalar.

TABLE OF CONTENTS

PLAGIARISM………………………………………………………………. III

ABSTRACT…………………………………………………………………. IV

ÖZ………………………………………………………………………… V

TABLE OF CONTENTS……………………………………………………. VI

CHAPTERS

1. INTRODUCTION……………………………………………………... 1

1.1 General……………………………………………………… 1

2. PROGRAM STRUCTURE…………………………………………….. 7

2.1 General……………………………………………………… 7

2.2 Inputs………………………………………………………... 7

2.2.1 Geometry………………………………………….. 7

2.2.2 Plate Properties…………………………………… 9

2.2.3 Bracing……………………………………………. 9

2.2.4 Support……………………………………………. 9

2.2.5 Stud……………………………………………….. 9

2.2.6 Pour Sequence…………………………………….. 9

2.3 Pre-processor Module………………………………………. 10

2.3.1 Preliminaries……………………………………… 10

2.3.2 Node Generation………………………………….. 12

2.3.3 Shell Element Generation………………………… 27

2.3.4 Internal and External Braces……………………… 30

2.3.5 Top Lateral Braces……………………………….... 33

2.3.6 Shear Studs……………………………………….... 35

2.3.7 Support Elements…………………………………. 37
2.3.8 Pin Nodes…………………………………………. 38
2.3.9 Assigning Shell Element Properties………………. 41
2.3.9.1 Property Shell Library…………………….. 41
2.3.9.2 Property Shell Indexes……………………. 43
2.3.10 Assigning the Stud Properties…………………….. 45
2.3.10.1 Stud Modification Factors…………….. 47
2.3.11 Assigning the Properties of Internal, External and Top Lateral Braces…………………………… 47
2.4 Processor Module…………………………………………… 48
2.4.1 Preliminaries……………………………………… 48
2.4.2 Gaussian Quadrature Data and Shape Function Arrays…………………………………………….... 48
2.4.3 Determination of the Number of Non-zero Entries in the Global Stiffness Matrix…………………….. 49
2.4.4 Formation of the "irowindex" and "icolumn" vectors…………………………………………….. 51
2.4.5 Formation of the Structural Stiffness Matrix……... 53
2.4.5.1 Assembly of Shell Elements……………… 54
2.4.5.2 Assembly of Top Lateral Braces………….. 55
2.4.5.3 Assembly of Internal Braces……………… 56
2.4.5.4 Assembly of External Braces……………... 57
2.4.5.5 Assembly of Support Elements…………… 59
2.4.5.6 Assembly of Studs……………………….... 59
2.4.6 Modification for Support Conditions……………... 60
2.4.7 Formation of the Load Vector…………………….. 60
2.4.8 Solution for Displacements……………………….. 61
2.5 Post-processor Module……………………………………… 62
2.5.1 Preliminaries……………………………………… 62
2.5.2 Cross-sectional Deflections……………………….. 62
2.5.3 Cross-sectional Rotations…………………………. 62

2.5.4 Top Lateral Brace Forces………………………… 63
2.5.5 Internal Brace Forces……………………………... 63
2.5.6 External Brace Forces…………………………….. 64
2.5.7 Cross-sectional Forces……………………………. 65

3. NUMERICAL MODELING DETAILS AND PROGRAM VERIFICATION…………………………………………………….. 69
3.1 Modeling of the Physical System…………………………… 69
3.2 Element Formulations………………………………………. 71
3.2.1 Shell Element Formulation……………………….. 71
3.2.2 Truss Element Formulation……………………….. 80
3.2.3 Spring Element Formulation……………………… 80
3.3 Solver Basics………………………………………………... 81
3.3.1 Sparse Solver Storage Format…………………….. 82
3.4 Program Verification with Existing Solutions……………… 83
3.4.1 General……………………………………………. 83
3.4.2 Hand Calculations………………………………… 84
3.4.3 Published Solutions……………………………….. 85

4. MODELING RECOMMENDATIONS FOR COMPOSITE BRIDGE ANALYSIS……………………………………………….. 88
4.1 Simply Supported Bridge Case……………………………... 88
4.1.1 Deflections……………………………………….... 93
4.1.1.1 Node Normals…………………………….. 96
4.1.2 Top Lateral Brace Forces…………………………. 97
4.1.3 Internal Brace Forces……………………………... 104
4.1.4 Cross-sectional Forces……………………………. 108
4.1.5 Cross-sectional Stresses…………………………... 109
4.2 Continuous Bridge Case…………………………………….. 111
4.2.1 Deflections……………………………………….. 115
4.2.2 Top Lateral Brace Forces………………................ 116

4.2.3 Internal Brace Forces…………………………...... 118
4.2.4 Cross-sectional Forces……………………………. 120
4.2.5 Cross-sectional Stresses…………………………... 120
4.3 Simply Supported Double I-girder Bridge………………….. 120
4.3.1 Deflections……………………………………... 124
4.3.2 Top Lateral Brace Forces…………………………. 126
4.3.3 External Brace Forces…………………………….. 126
4.3.4 Cross-sectional Forces……………………………. 128
4.3.5 Cross-sectional Stresses…………………………... 128
4.4 Solver Type and Performance………………………………. 129
4.4.1 Direct vs. Iterative Solver: A Case Study………… 130

5. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH……………………………………………………… 134
BIBLIOGRAPHY………………………………………………………… 137
APPENDIX-A: SUBROUTINES LIST…………………………………... 139
APPENDIX-B: VARIABLES LIST……………………………………… 147
APPENDIX-C: USER'S MANUAL……………………………………... 153

CHAPTER 1
INTRODUCTION

1.1 General

Composite highway interchange bridges are being built more frequently recently. This is mainly because of the ease of construction, long span coverage, economy and aesthetics. A typical composite bridge consists of a concrete deck at the top and steel girder(s) supporting the deck (Fig.1.1,Fig.1. 2).

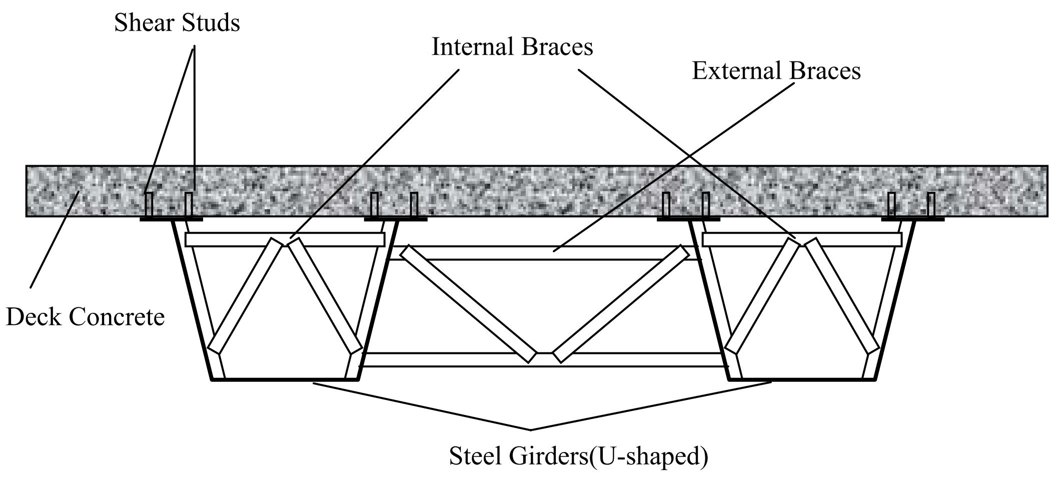


Figure 1.1: Typical Composite Bridge Cross-section

Although trapezoidal box girders are extensively used owing to their high torsional stiffness after the hardening of the deck concrete, steel I-girders are also common in practice. Shear studs are welded to the top flanges of the girders to achieve composite action (Fig. 1.1). Curved interchange bridges are generally long-spanned and the deck concrete should generally be placed not at once but in sequence because of the large volume of the concrete, and also to control shrinkage. This requires consideration of the structural behavior of the steel girders at early concrete ages during construction. Another major problem faced in these structures is the relatively low torsional stiffness observed before the achievement of composite behavior between the deck and the girders through the studs as the concrete hardens. In order to sustain the stability of the girders internal, external and top lateral braces are used (Fig. 1.1).

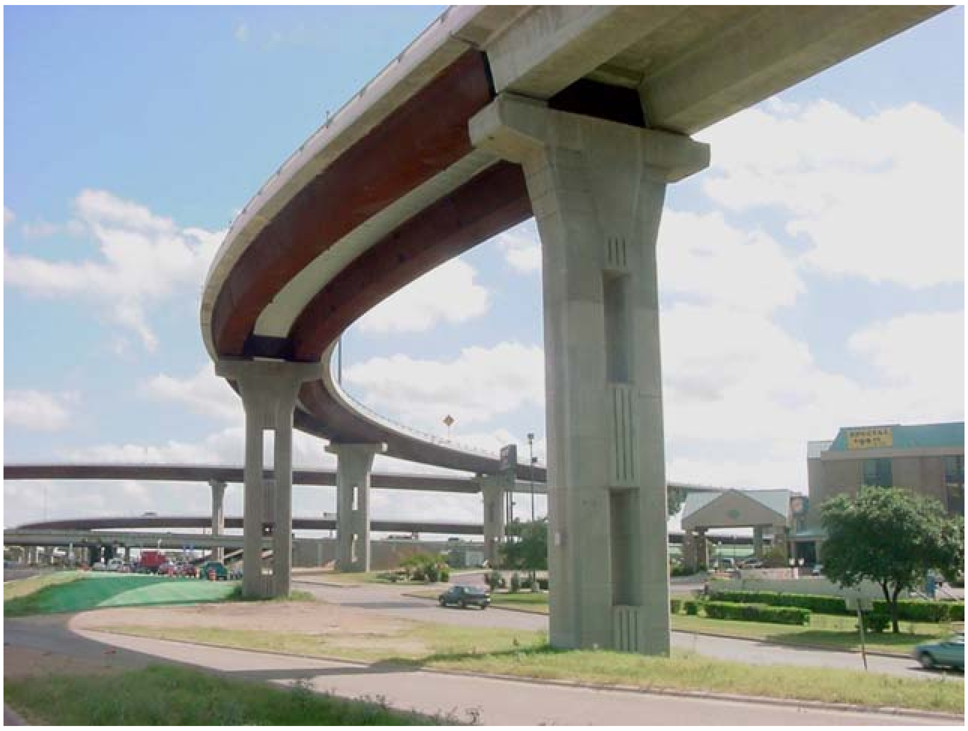


Figure 1.2: Composite Bridge

There are several methods to analyze curved steel girders including the approximate hand methods, finite difference method, finite strip method, grid analysis method and the finite element method (FEM). Among these, the most effective one is the FEM in terms of both modeling and accuracy. In the FEM, the complex structures are divided into large, finite number of elements and the behavior of these elements is well defined. Accumulating the individual element responses, the overall complex structure response can be obtained. The FEM allows one to determine the stresses and strains at any location in a cross-section. The main shortcoming of the FEM is the extensive computer resources it requires. A large number of equations has to be solved in order to get the required outputs. Although the box and I-shaped steel girders can be analyzed using general-purpose finite element packages which require little finite element background, formation of the models can be difficult and time consuming. Also accurate modeling of composite action between the concrete deck and steel girder and behavior of the composite bridge at the early concrete ages needs extra effort. In addition to this, parametric studies cannot be easily carried out as every time a parametric study is performed a new model needs to be created.

Until now FEM was not extensively used as a tool to analyze composite curved bridges. Also the behavior of composite bridges at early concrete ages was not understood clearly. In the field it was observed that (Topkaya, Williamson, Frank 2004) before the concrete gains strength, the steel girders fail to resist the torsional stresses created (Fig. 1.3).

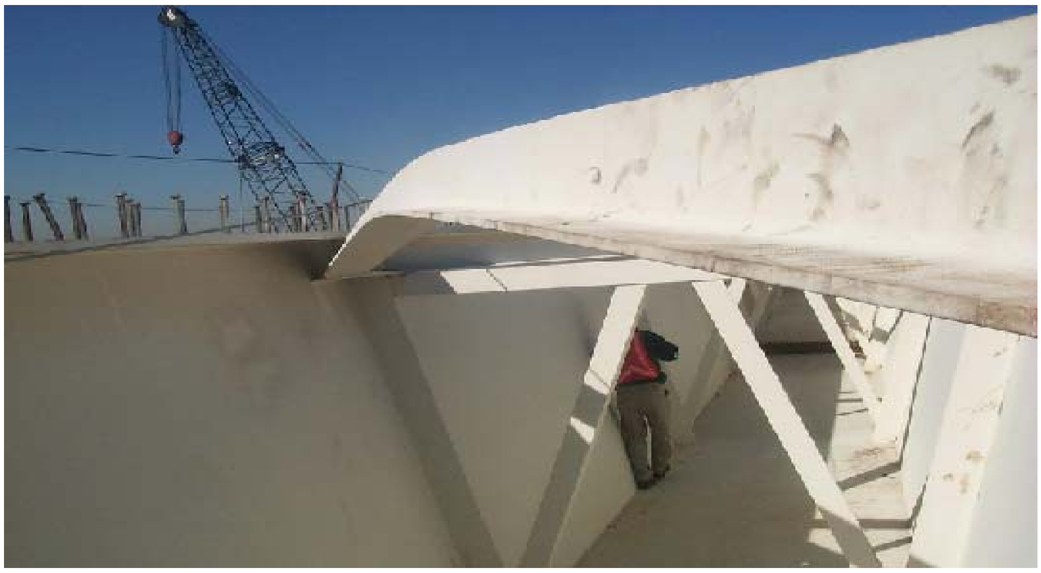


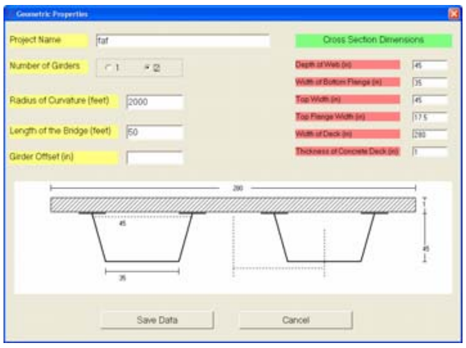
Figure 1.3: Trapezoidal box girder failure during construction

In order to put these issues into the design, to speed up the design process and to allow for easy parametric studies a computer program named UTRAP was developed (Topkaya, Williamson 2003). UTRAP consists of an analysis module and the input module. The analysis module of the program was written in Fortran and the input module is a Graphic User Interphase (GUI) written in Visual Basic. The inputs to the program are entered through the GUI and the GUI converts these inputs into a text file. The main engine of the program reads these inputs from the text, performs the analysis and produces the outputs as a text file. Finally, GUI reads the outputs and displays them according to the display format.

UTRAP is able to analyze trapezoidal box girders under the construction loads. The analysis module of the program is a linear, three dimensional finite element code consisting of pre-processing, processing and the post processing components. The analysis module forms the finite element mesh, element connectivity and the material properties according to information supplied from the GUI. Concrete deck and the steel girders are modeled with nine-node shell elements, internal, external and top lateral braces are modeled with two-node truss elements and the shear studs are modeled with two-node spring elements. For all bridges a constant mesh density is used. The program is able to analyze single and dual girders. At a given cross-section top flanges are modeled with two shell elements, webs and bottom flanges are modeled with four shell elements and the concrete deck is modeled with ten shell elements for single girder systems and twenty elements for dual girders systems. The program utilizes the Imperial System of units and the constant element size along the bridge is 0.6m(2-ft). The types of the internal, external and top lateral braces implemented into the program conform with the practice. For the concrete pour sequence analysis, the concrete deck can be divided into segments along the bridge. For each segment different values for the concrete modulus, stud stiffness and distributed load can be entered. UTRAP is capable of reporting the stresses over the entire cross-section, directly computing the brace forces, taking into account the partial composite behavior.

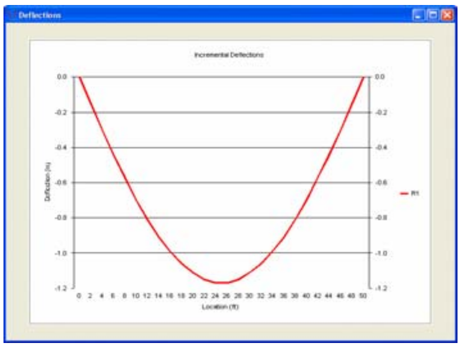
Through the GUI cross-sectional dimensions, plate thickness, locations of supports and braces, properties of the concrete deck and construction loads can be inputted to the program. The GUI has also the ability to display the analysis results both numerically and graphically (Fig. 1.4).

Although UTRAP is a useful program for bridge designers and gives outputs compatible with the available analysis tools, it has some limitations. These limitations are both geometric, such as the constant element size and structural, such as the single element type used for modeling. Also the program has a rigid structure so that new features cannot be easily implemented. In order to overcome this problem, the program needed to be restructured by rewriting from the beginning, keeping certain subroutines, adding new ones and forming a new program structure so that in the future, improvements can be easily made with minimum effort.



Project Name	faf
Number of Girders	1 [✓ 2]
Radius of Curvature (feet)	2000
Length of the Bridge (feet)	50
Girder Offset (in)	

Cross Section Dimensions	
Depth of Web (in)	45
Width of Bottom Flange (in)	35
Top Width (in)	45
Top Flange Width (in)	17.5
Width of Deck (in)	280
Thickness of Concrete Deck (in)	1



(a) (b)

Figure 1.4: (a)Geometric Properties Input Form, (b)Deflection Diagram

One of the limitations of the program was the constant element size which leads to a constant mesh density. The user was not able to change mesh density so the differences between having a fine mesh or a coarse mesh could not be observed. Also a single nine-node shell element was implemented into the program to model the steel girders and the concrete deck but there was no alternative for this element so the effects of the element size on the results were unclear. Another drawback was that the program was only capable of solving single or dual girders with box steel sections but in practice I-sections are also used and the number of girders can be more than two. The program was able to solve only the straight girders and the constant curvature bridges but bridge curvature can be variable. This type of variable curvature bridges are commonly observed in heavily congested freeway to freeway interchanges. A direct sparse solver was implemented in UTRAP but there are also iterative sparse solvers so the performance of these two types need to be compared. Finally, the system of units used in the program is only the Imperial System of Units but SI units are more commonly used in the world.

The thesis work consists of the restructuring of the computer program UTRAP and overcoming the above mentioned limitations. First of all, the constant element size used in the program is converted into variable element size. In addition to the previously implemented nine-node element into the program another four-node element is implemented. Thirdly I-shaped girders are implemented into the program. Also the program is restructured so that it takes into account the variable radius of curvature along the length of the bridge. To compare the performances of the direct sparse solvers and the iterative sparse solvers, a sparse iterative solver is adopted into the program. Finally the present form of the program is also compatible with the SI units.

In the following sections first the overall program algorithm will be explained including the flow chart and all the individual subroutines. In the second part Finite Element Formulation and the Numerical Modeling Details will be discussed. The third part consists of the verification of the developed software with the hand calculations and published solutions. After that parametric studies, mesh convergence and analysis recommendations for design will be presented.

CHAPTER 2

PROGRAM STRUCTURE

2.1 General

The developed program is composed of more than 14 000 lines of Fortran code excluding the solver libraries. The solver libraries used in the program are readily available equation solver packages. The main engine of the developed program consists of the pre-processor, processor and post-processor modules. In the pre-processor module, first, the geometry of the bridge is defined together with the node locations within a cross-section and along the span. Then the elements forming the finite element mesh are formed and element properties are assigned. After establishing the geometry using the loading data the nodal loads are assigned. Upon the formation of the geometry and assignment of the nodal loads the processor module of the program assembles the global stiffness matrix and solves the equilibrium equations to obtain the displacements. The nodal displacements are processed in the post processor module to give the required stresses and forces. The program flowchart is given in Figure 2.1.

2.2 Inputs

As mentioned before the inputs of the program are entered through the GUI and GUI creates a text file readable by the main engine of the program. The inputs to the program can be classified into six main categories namely geometry, plate properties, bracing, support, stud and pour sequence.

2.2.1 Geometry

Geometric inputs of the program are the number of girders which may be single or dual, chosen element size which will affect the mesh density, number of

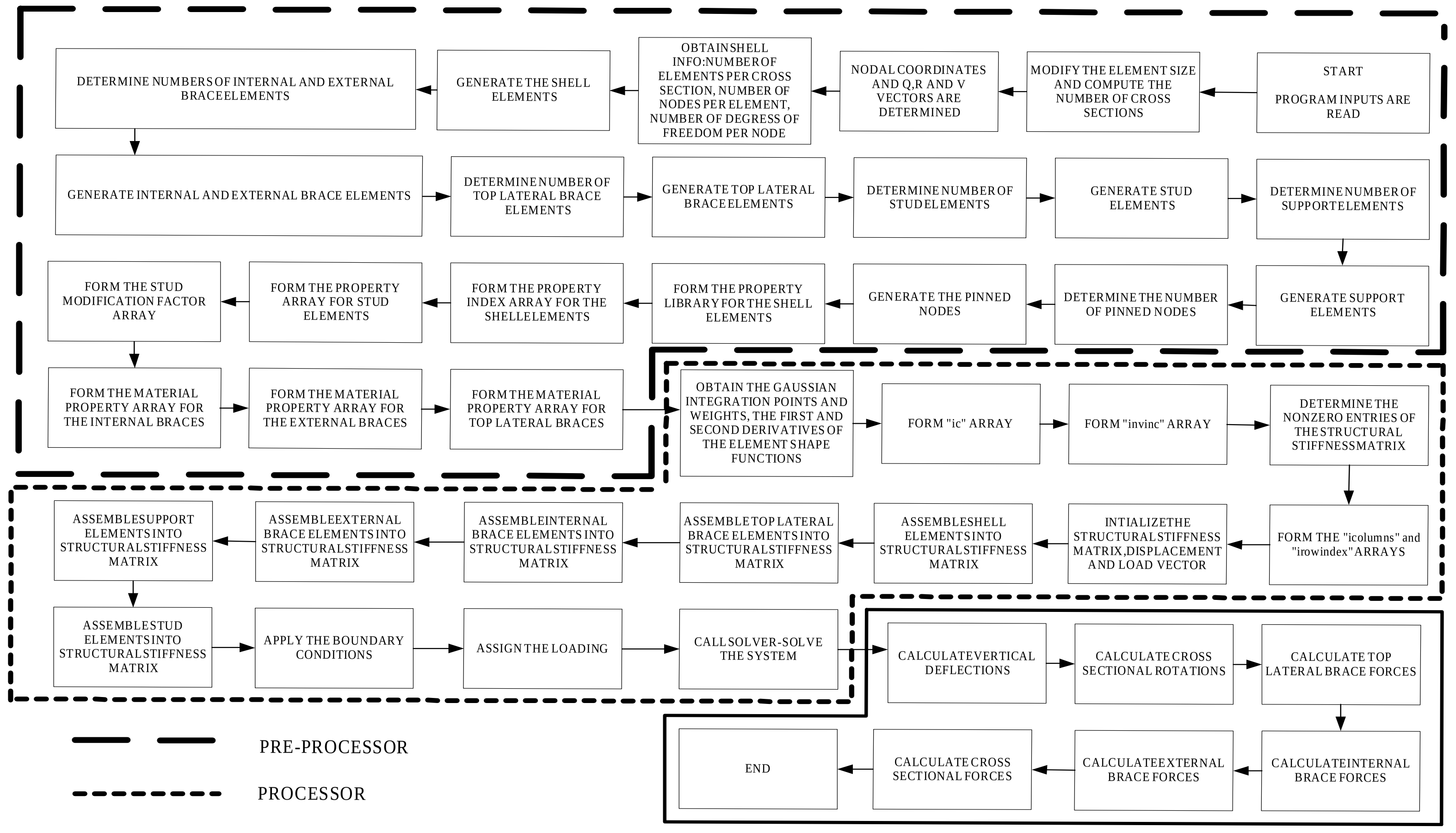

START
PROGRAM INPUTS ARE READ
MODIFY THE ELEMENT SIZE AND COMPUTE THE NUMBER OF CROSS SECTIONS
NODAL COORDINATES AND Q,R AND V VECTORS ARE DETERMINED
OBTAINSHELL INFO:NUMBEROF ELEMENTS PER CROSS SECTION, NUMBER OF NODES PER ELEMENT, NUMBER OF DEGRESS OF FREEDOM PER NODE
GENERATE THE SHELL ELEMENTS
DETERMINE NUMBERS OF INTERNAL AND EXTERNAL BRACE ELEMENTS
GENERATE INTERNAL AND EXTERNAL BRACE ELEMENTS
DETERMINE NUMBER OF TOP LATERAL BRACE ELEMENTS
GENERATE TOP LATERAL BRACE ELEMENTS
DETERMINE NUMBER OF STUD ELEMENTS
GENERATE STUD ELEMENTS
DETERMINE NUMBER OF SUPPORT ELEMENTS
GENERATE SUPPORT ELEMENTS
DETERMINE THE NUMBER OF PINNED NODES
GENERATE THE PINNED NODES
FORM THE PROPERTY LIBRARY FOR THE SHELL ELEMENTS
FORM THE PROPERTY INDEX ARRAY FOR THE SHELLELEMENTS
FORM THE PROPERTY ARRAY FOR STUD ELEMENTS
FORM THE STUD MODIFICATION FACTOR ARRAY
FORM THE MATERIAL PROPERTY ARRAY FOR THE INTERNAL BRACES
FORM THE MATERIAL PROPERTY ARRAY FOR THE EXTERNAL BRACES
FORM THE MATERIAL PROPERTY ARRAY FOR TOP LATERAL BRACES
OBTAIN THE GAUSSIAN INTEGRATION POINTS AND WEIGHTS, THE FIRST AND SECOND DERIVATIVES OF THE ELEMENT SHAPE FUNCTIONS
FORM "ic" ARRAY
FORM "invinc" ARRAY
DETERMINE THE NONZERO ENTRIES OF THE STRUCTURAL STIFFNESSMATRIX
FORM THE "icolumns" and "irowindex" ARRAYS
INTIALIZETHE STRUCTURALSTIFFNESS MATRIX,DISPLACEMENT AND LOAD VECTOR
ASSEMBLESHELL ELEMENTSINTO STRUCTURALSTIFFNESS MATRIX
ASSEMBLETOP LATERAL BRACE ELEMENTS INTO STRUCTURALSTIFFNESS MATRIX
ASSEMBLEINTERNAL BRACE ELEMENTS INTO STRUCTURALSTIFFNESS MATRIX
ASSEMBLEEXTERNAL BRACE ELEMENTS INTO STRUCTURALSTIFFNESS MATRIX
ASSEMBLESUPPORT ELEMENTSINTO STRUCTURALSTIFFNESS MATRIX
ASSEMBLESTUD ELEMENTSINTO STRUCTURALSTIFFNESS MATRIX
APPLY THE BOUNDARY CONDITIONS
ASSIGN THE LOADING
CALL SOLVER-SOLVE THE SYSTEM
CALCULATEVERTICAL DEFLECTIONS
CALCULATE CROSS SECTIONAL ROTATIONS
CALCULATE TOP LATERAL BRACE FORCES
CALCULATEINTERNAL BRACE FORCES
CALCULATEEXTERNAL BRACE FORCES
CALCULATE CROSS SECTIONAL FORCES
END
PRE-PROCESSOR
PROCESSOR

Figure 2.1: Program Flowchart

segments including the length and radius curvature of each segment, girder section type-either trapezoidal or I section, chosen shell element type which may be four-node or nine-node elements and the cross-sectional dimensions. The cross-sectional dimensions are web depth, bottom flange width, top width, top flange width, deck width and concrete deck thickness.

2.2.2 Plate Properties

Inputs for the plate properties are entered to define the thicknesses of the webs, bottom flanges and top flanges. The user can enter variable thicknesses along the length of the bridge.

2.2.3 Bracing

Bracing inputs are internal, external and top lateral brace locations, types and areas which may be again variable along the span.

2.2.4 Support

To input the supports the number and locations should be specified.

2.2.5 Stud

The inputs related to studs include stud locations, spacing and number per flange.

2.2.6 Pour Sequence

The pour sequence inputs are length of the poured concrete, concrete modulus, stud stiffness and the loading which are specified for every pour sequence. When the concrete is placed, before it hardens and gains strength, it has to be carried by the steel girders. The developed program has the ability to analyze girders with semi-cured concrete.

2.3 Pre-processor Module

2.3.1 Preliminaries

After the inputs are read and assigned to different variables, the pre-processing module of the program starts. First of all the total length of the bridge is obtained by adding all the segment lengths. The element size is truncated so that the total bridge length is an integer multiple of the element size. This adjustment was necessary as the total length of the bridge specified by the user may not be same as the total length of the bridge obtained after mesh generation. The truncation is done as in Algorithm 2.1.

```
Algorithm 2.1
1. The integer number of element divisions are assigned to a dummy variable
idum = Length of the bridge(as specified by the user)/element size(as
specified by the user)
2. idum is assigned as the number of divisions.
ndiv(number of divisions) = idum
3.Truncated element size is found by dividing the total length of bridge by ndiv
Element size(truncated) = Length of the bridge(as specified by the user)/ndiv
```

After the truncated element size is obtained, the total number of cross-sections is calculated for the two types of elements, four-node and nine node-shell elements. The geometric difference between these two element types is that nine-node element has middle nodes and four-node element does not. When generating the mesh, the program generates more cross-sections for the nine node shell element. The number of cross-sections for bridges modeled with nine-node elements generated for the nine-node element is twice the number of divisions plus one. The number of the cross-sections for the four-node element is calculated by adding one to the number of divisions.

There are three variables used to define different cross-sections that the program can analyze. These variables are the number of girders (denoted by ngird in the program), steel girder section type (denoted by isec_type in the program) and shell element type (denoted by ielem_type in the program). Number of girders can be either one or two, steel girder section type can be either trapezoidal or I-section and the shell element type can be either nine-node element or four-node element. This makes a total of eight different cases. When defining a specific case a three digit number notation is used like ‘xyz’. Here x stands for the number of girders, y stands for the steel girder section type and z stands for the shell element type. For example 212 defines the cross-section having two I girders with four-node elements. The program UTRAP is structured in such a way that in the restructured program, the number of girders can easily be increased and new steel cross-sections can easily be added. In addition to the nine-node element which the program utilizes a four-node element is added into the program and if new elements need to be utilized they can be added without much effort.

As discussed previously the program utilizes a constant cross-sectional mesh density for each cross-sectional case. For example for case 111, the concrete deck is modeled with ten shell elements, the top flanges are modeled with two shell elements, the web and bottom flanges are modeled with four shell elements. Table 2.1 shows the eight different cases and how many shell elements are used for modeling the different parts.

For each of the eight cross-sections the number of nodes per cross-section are shown in Table 2.2.

Table 2.1: Shell modeling

	Number of shell elements used for modeling			
Case	Concrete deck	Top Flange	Bottom Flange	Web
111	10	2	4	4
112	10	2	4	4
121	6	2	2	4
122	6	2	2	4
211	20	2	4	4
212	20	2	4	4
221	12	2	2	4
222	12	2	2	4

The number of nodes for each cross-section is constant and is calculated internally in the program.

Table 2.2:Number of Nodes Per Cross-section for Each Case

Case	Number of nodes per cross-section
111	54
112	28
121	30
122	16
211	107
212	55
221	59
222	31

2.3.2 Node Generation

The global x,y and z coordinates of the nodes are computed from the cross-sectional dimensions and the radius of curvature of the individual segments. In order to define the shell element geometry, three mutually orthogonal unit vectors are formed for each node denoted by Q, R, and V. Among these unit vectors, V always points in the direction of the thickness of the shell element and R always points in the direction of the tangent to the arc length. Q is the unit vector which is orthogonal to the other two vectors. The unit vectors for the 111 case can be seen on the Fig. 2.2.

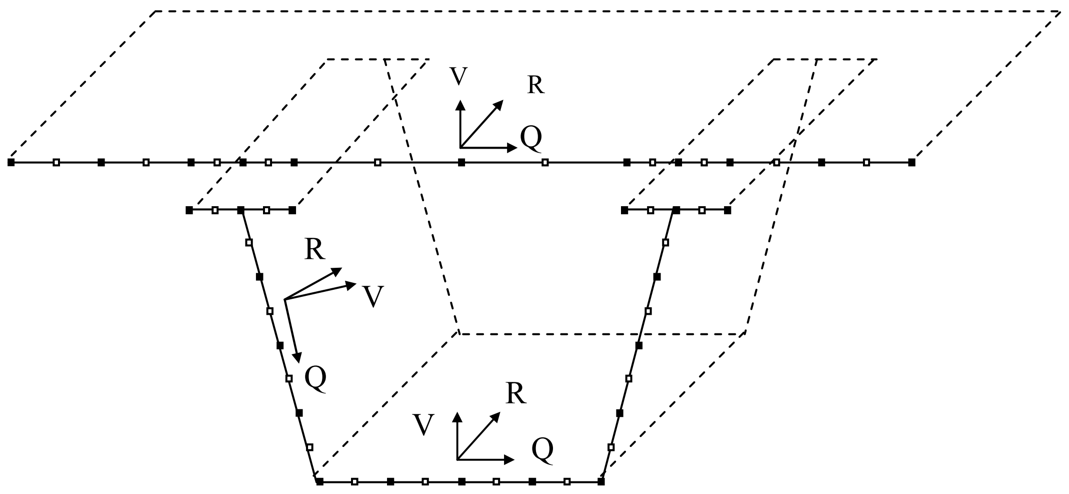


Figure 2.2: Node Locations and Unit Vectors for Case 111

In the main program, the subroutine "get_coordinates" is called and this subroutine itself calls the subroutines to compute coordinates and unit vectors for each node for every cross-section type. For example subroutine "get_coordinatesxyz" computes the coordinates and unit vectors for the cross-section having "x" "y"-type steel sections composed of "z" type shell elements. The notation used for expressing the coordinates of a node is xy(a,b). Here "a" can take the values 1 for the x-coordinate, 2 for the y-coordinate and 3 for the z-coordinate. The second variable in the parenthesis, b, stands for the node number.

In a given cross-section, nodal coordinates are calculated parametrically by the cross-sectional dimensions. These cross-sectional dimensions are web depth, bottom flange length, top length, top flange width, deck width, and the offset distance (Fig. 2.3).

After the determination of the nodal point coordinates(x and y coordinates) for the first cross-section, the program generates the nodal point coordinates for the rest of the bridge(x,y and z coordinates) using the individual segment lengths and radius of curvatures (Algorithm 2.2, Figure 2.4).

The nodes for the all eight cross-sections are shown in Fig. 2.5 through Fig 2.12.

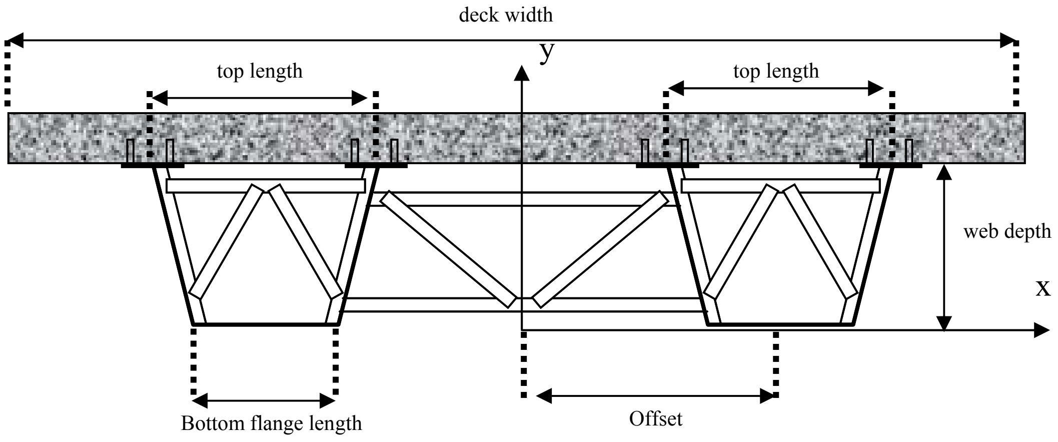


Figure 2.3: Cross-sectional Dimensions

Algorithm 2.2

1. For each bridge segment having a different radius of curvature, find the start and end cross-section numbers. Force the last cross-section number be equal to the previously calculated total number of cross-sections value.

2. Set the global x and z-coordinates of the centers of each curved segment(cx, cz) and **vax**, **vay** ,**vbx** and **vby** to zero.

cx = 0.0

cz = 0.0

3. Loop over the segments and locate the x and z-coordinates of the centers.

nnpcs : number of nodes per cross-section

rnn : reference node number

cx = xy(1,(icsec_start(ij)-1)*nnpcs +rnn)+vax*al_rcurv_segm(ij)

cz = xy(3,(icsec_start(ij)-1)* nnpcs + rnn)+vaz*al_rcurv_segm(ij)

4. Define angle theta and set it to zero.

θ = 0.0

Theta is the angle enclosed by the arc length starting from the first cross-section up the cross-section for which the nodal coordinates are to be determined.

```
5.Update θ.
θ_new = θ_previous + element size/radius of curvature ( 4-node element)
θ_new = θ_previous + (element size/2.0)/radius of curvature ( 9-node element)
6. Form the unit vector va starting from the center of the segment (if it has certain curvature) pointing along the radius to the reference node on the bridge center line.
vax = cos(θ)
vaz = sin(θ)
5. Form the unit vector, vb which is perpendicular to va
6. If the bridge segment is straight x and z-coordinates of the generated segments are calculated as follows(9-node element):
do ik = 1, number of cross-sections per segment
do im = 1, number of nodes per cross-sections
xy(1,(ik* nnpcs)+im)=xy(1,(ik-1)* nnpcs +im)-element size/2.0*vbx
xy(3,(ik* nnpcs)+im)=xy(3,(ik-1)* nnpcs +im)- element size /2.0*vbz
end do
end do
7. If the bridge segment(ij) is curved, x and z-coordinates of the generated segments are calculated as follows(9-node element):
do ik = 1, number of cross-sections per segment
do im = 1, number of nodes per cross-sections
xy(1,(ik* nnpcs)+im)=cx+vax*(xy(1,im)-radius of curvature(ij))
xy(3,(ik* nnpcs)+im)=cz+vaz*(xy(1,im)- radius of curvature (ij))
end do
end do
```

Having obtained the x,y and z-coordinates of all nodes forming the finite element mesh, previously explained Q, R and V unit vectors (Fig 2.2) which are used in defining the shell geometry are calculated. This calculation is explained in Algorithm 2.3.

Algorithm 2.3

1. The **Q**, **R** and **V** vectors for the nodes lying on the horizontal plane(nodes forming the deck, bottom flange and top flange) are calculated first by forming a unit vector on the x-z plane using the bottom flange nodes extending from left to right.

For Case 111;

qx= xy(1,46+(i-1)*54)-xy(1,38+(i-1)*54)

qz= xy(3,46+(i-1)*54)-xy(3,38+(i-1)*54)

2. By diving **qx** and **qz** components by the norm of the vector which is;

Norm = $\sqrt{(\mathrm{qx}^2 + qz^2)}$

3. **Q** unit vector is obtained which always points towards the center.

R is the unit vector which is perpendicular to **Q** so the dot product of the two vectors should be zero, then it should have the components;

$\mathbf{R_X = Q_Z}$

$\mathbf{R_Z = -Q_X}$

V unit vector always points toward the center of the shell so it will have only y-component and the value of it is 1.

$\mathbf{V_Y = 1.0}$

4. For the webs which are rather curved, first the x and z components of the **R** unit vector is found by calculating the x and z components of the unit vector on the horizontal bottom flange. For case 111;

ax=xy(1,46+(i-1)*54)-xy(1,38+(i-1)*54)

az=xy(3,46+(i-1)*54)-xy(3,38+(i-1)*54)

vect1= $\sqrt{(\mathrm{ax}^2 + az^2)}$

ax=ax/vect1

ay=0.0

az=az/vect1

$\mathbf{R_X = az}$

$\mathbf{R_Z = -ax}$

5. The next step is to find the x,y and z components of the **Q** unit vector. So for

111 case;

Q_X=xy(1,38+(i-1)*54)-xy(1,30+(i-1)*54)

Q_Y=xy(2,38+(i-1)*54)-xy(2,30+(i-1)*54)

Q_Z=xy(3,38+(i-1)*54)-xy(3,30+(i-1)*54)

vect2= $\sqrt{(\mathrm{qx}^2 + qy^2 + qz^2)}$

Q_X =qx/vect2

Q_Y =qy/vect2

Q_Z =qz/vect2

6. **V** is the unit vector perpendicular to both **Q** and **R**, so it can be found by the cross product of **Q** and **R**.

V = Q $_X$ R

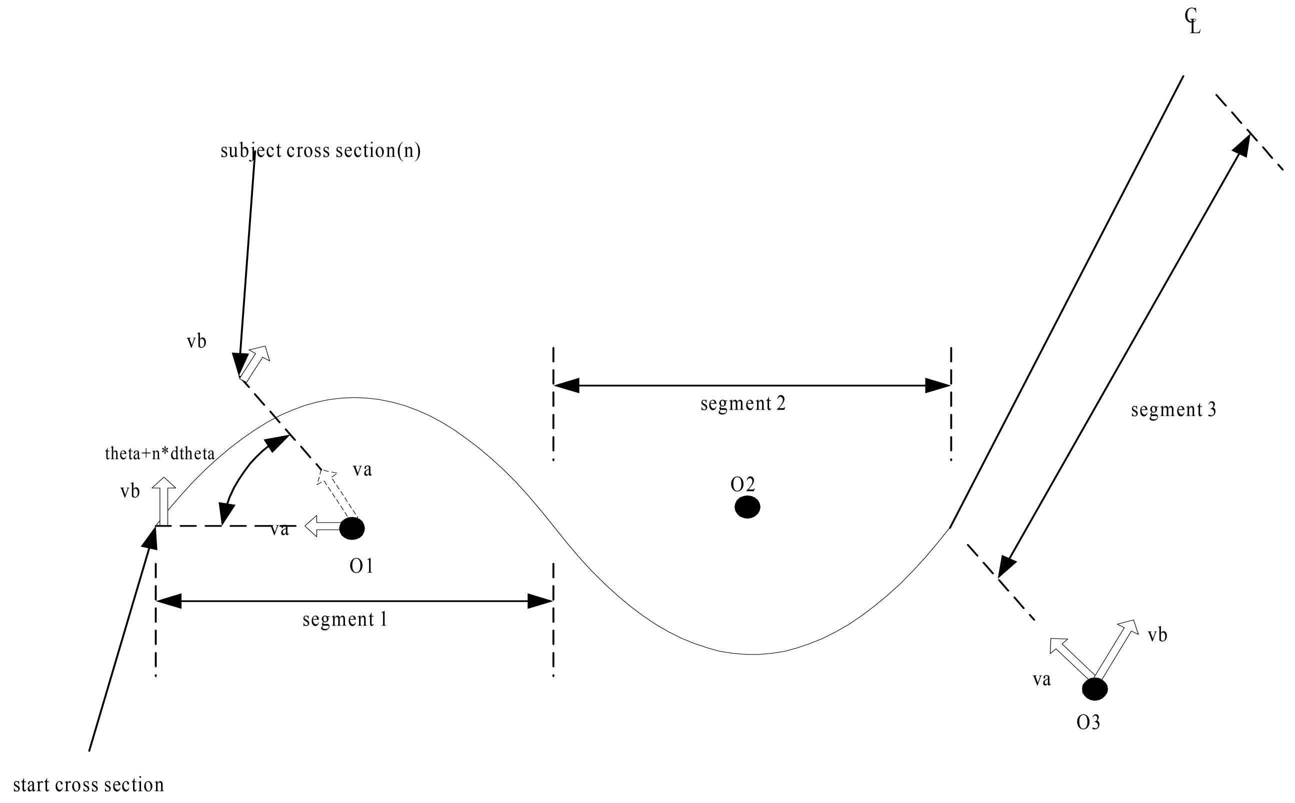


Figure 2.4: Typical variable radius of curvature bridge and the parameters involved in coordinate calculations

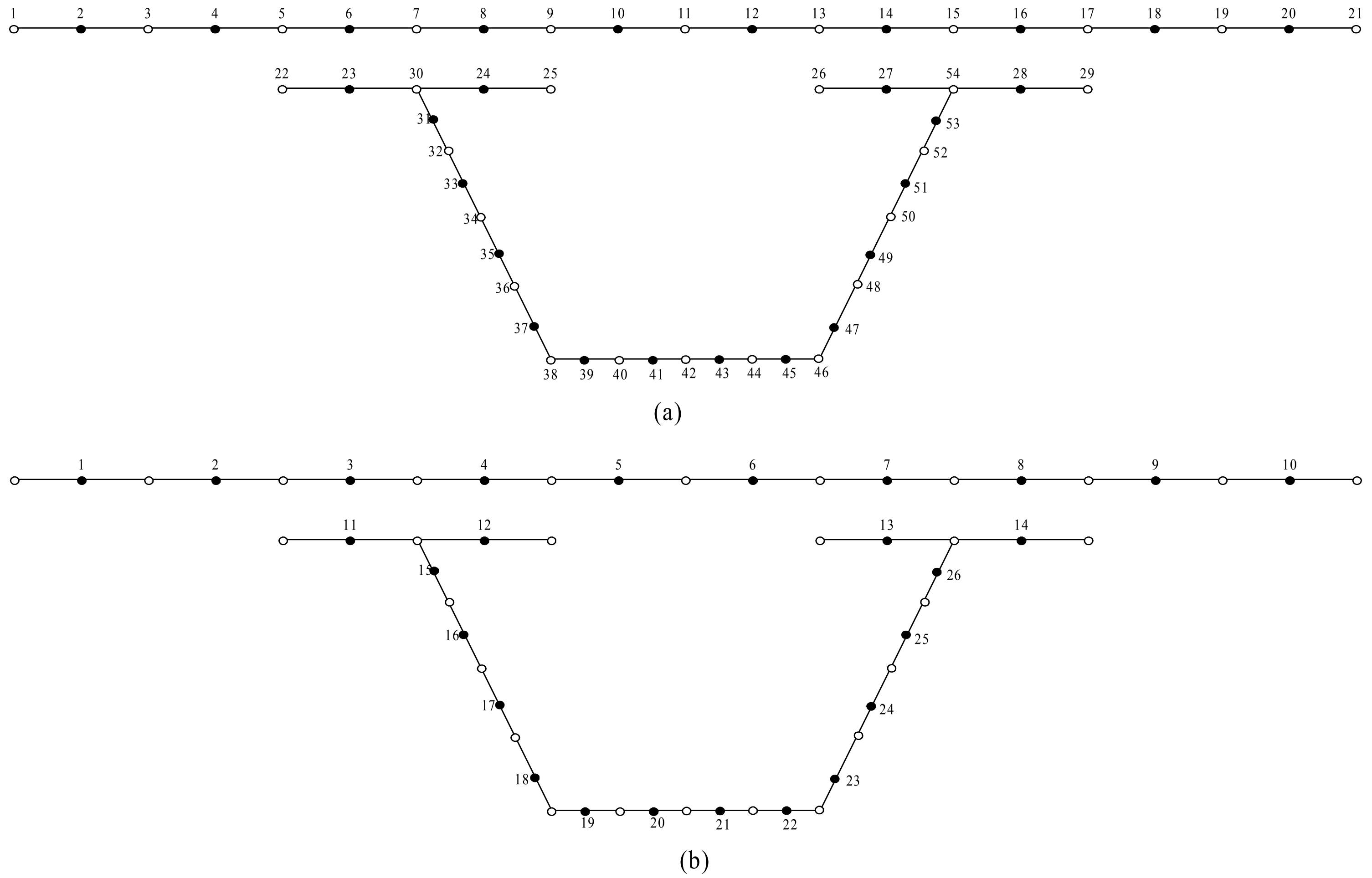


Figure 2.5: (a)Node and (b)Element Numbering for Case 111

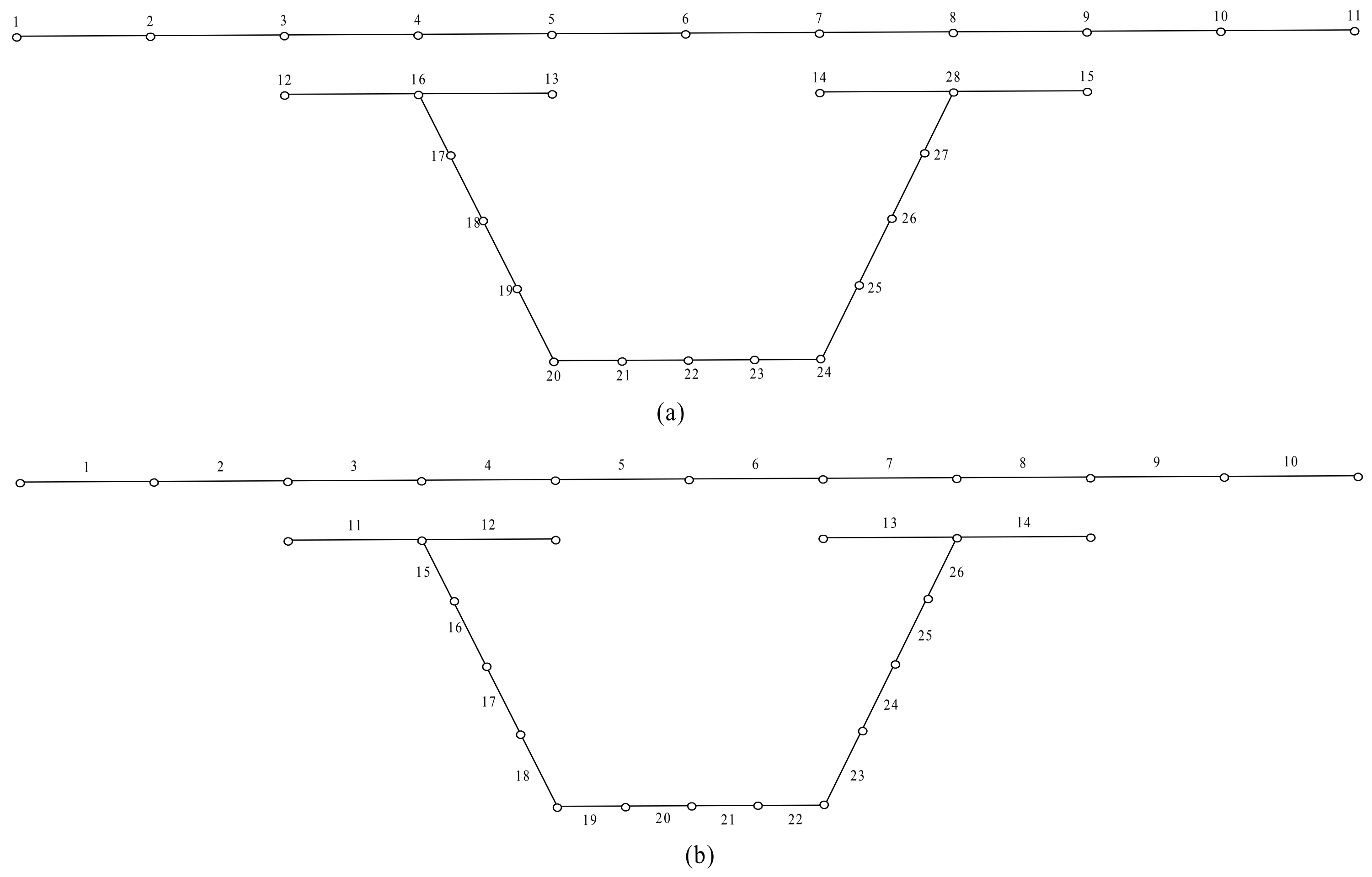
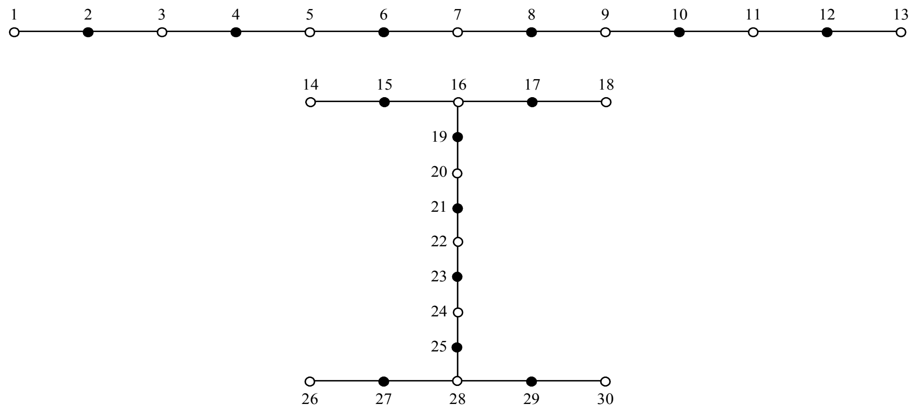
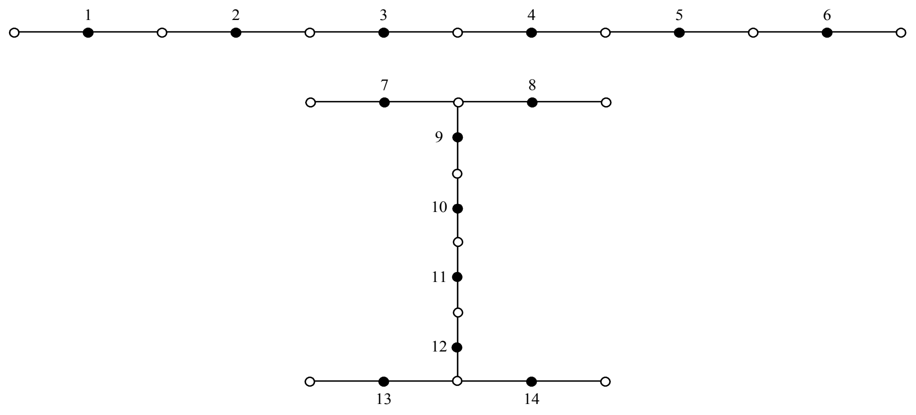


Figure 2.6: (a)Node and (b)Element Numbering for Case 112

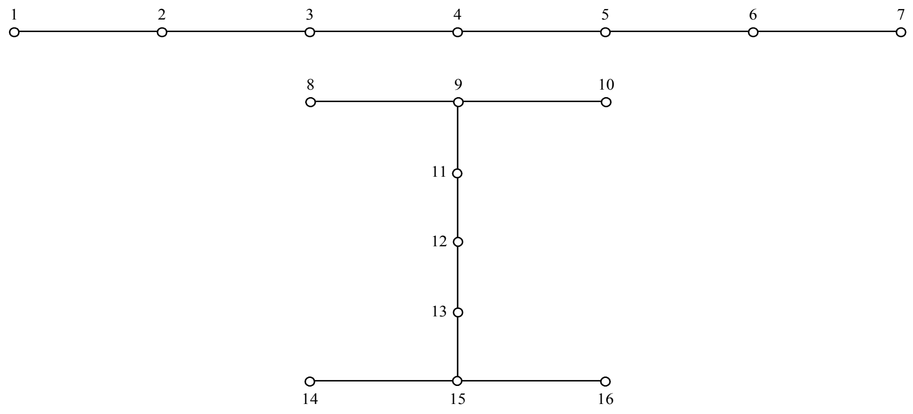


(a)

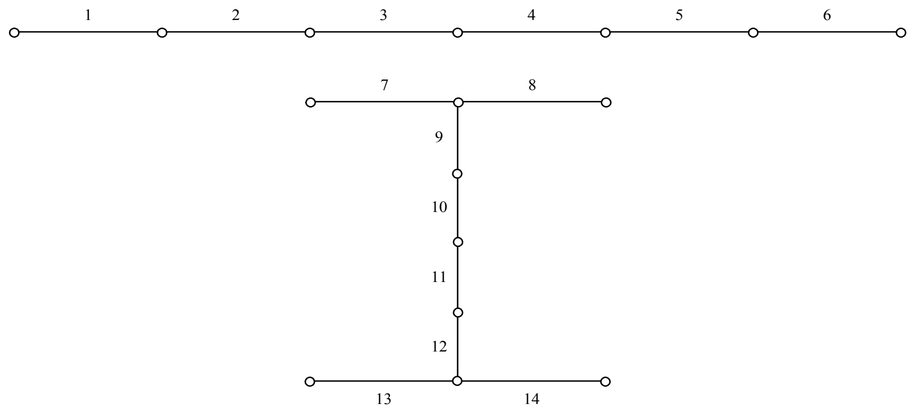


(b)

Figure 2.7: (a)Node and (b)Element Numbering for Case 121



(a)



(b)

Figure 2.8: (a)Node and (b)Element Numbering for Case 122

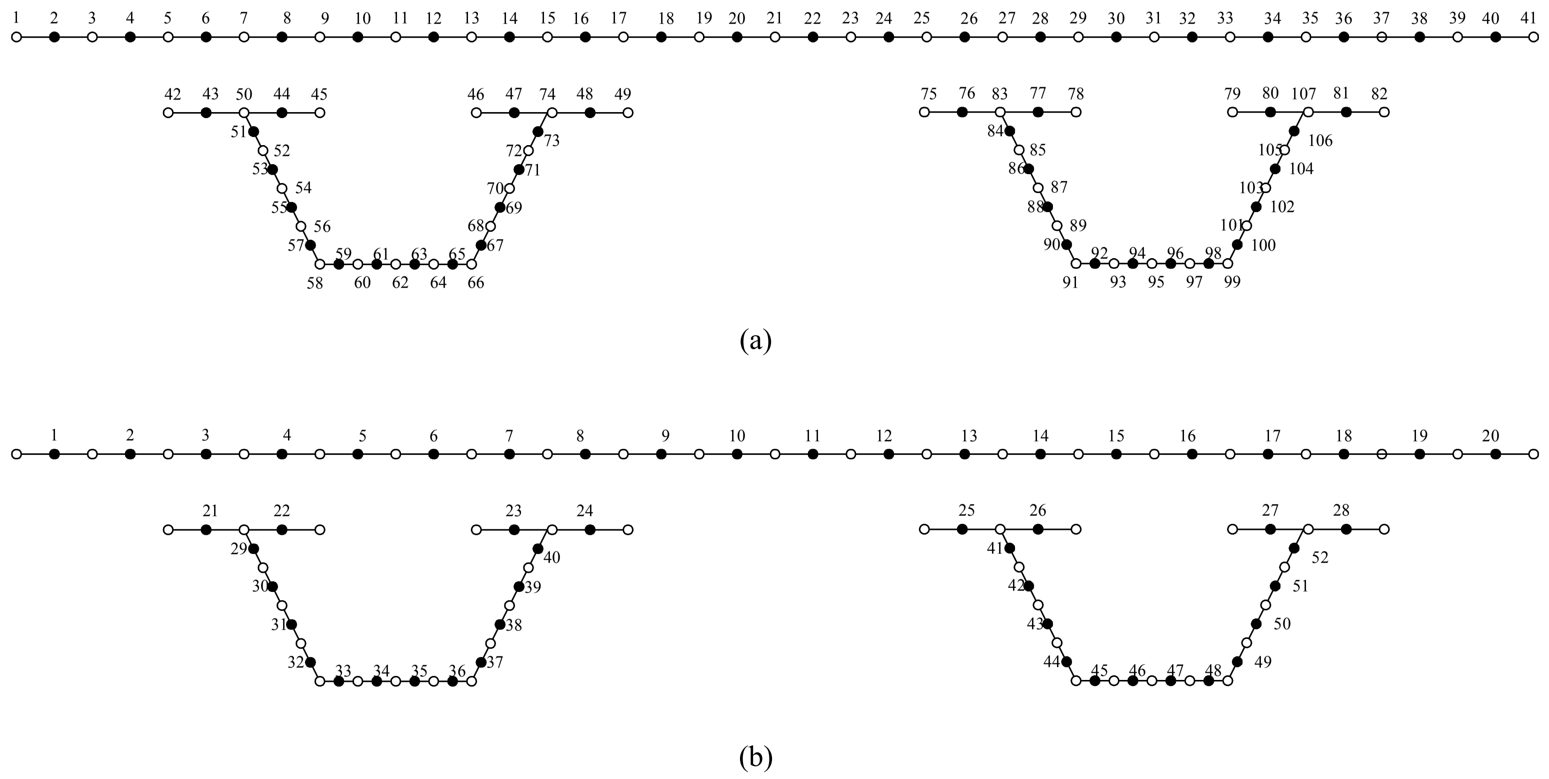


Figure 2.9: (a)Node and (b)Element Numbering for Case 211

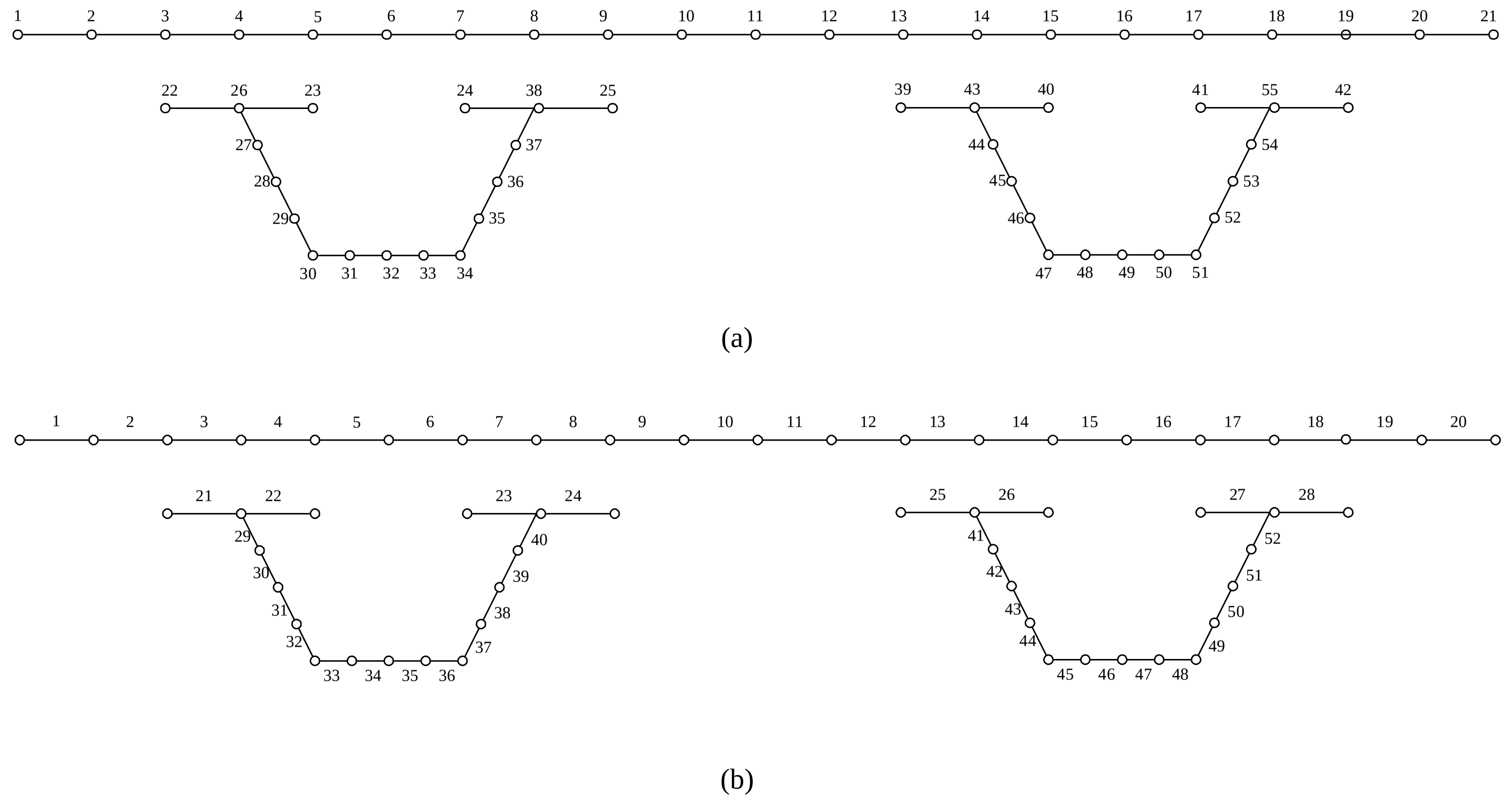
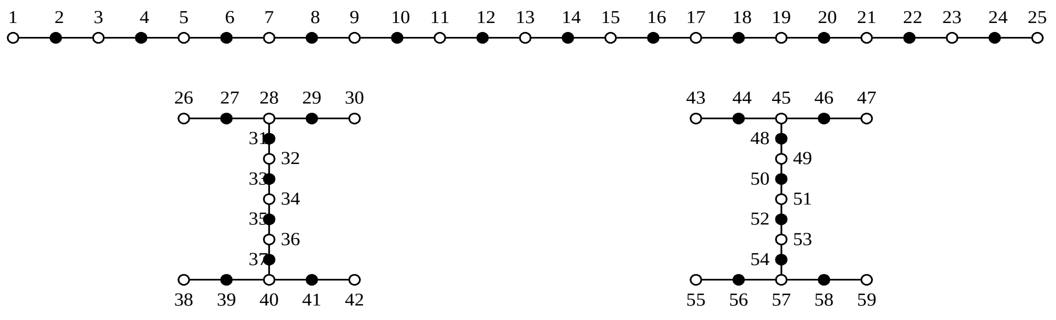
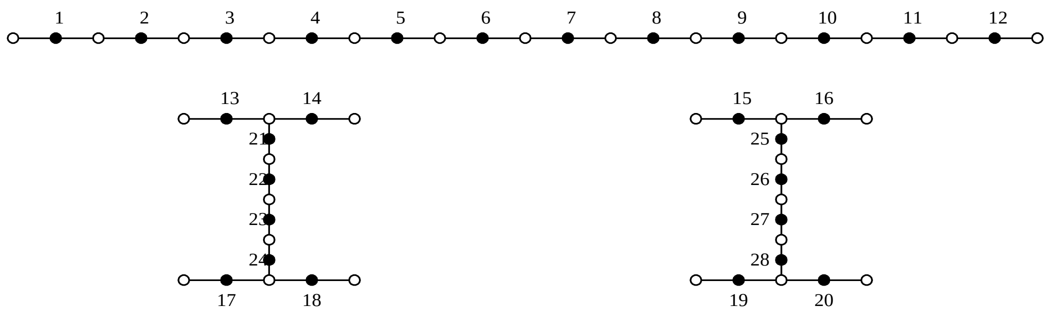


Figure 2.10: (a)Node and (b)Element Numbering for Case 212

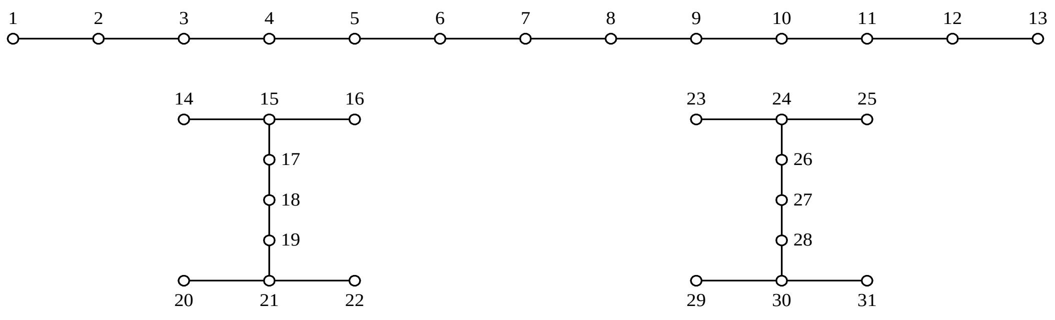


(a)

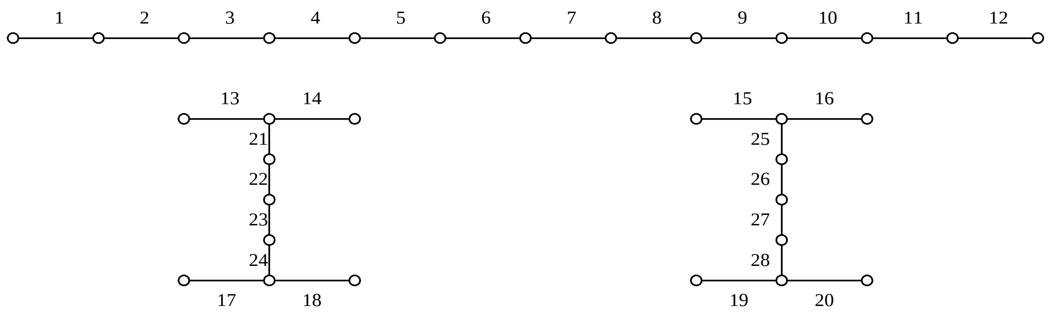


(b)

Figure 2.11: (a)Node and (b)Element Numbering for Case 221



(a)



(b)

Figure 2.12: (a)Node and (b)Element Numbering for Case 222

After the nodal coordinates and unit vectors associated with them are obtained, the next step is to generate the shell elements. Before the shell element generation subroutines are called, in the main program three variables are set to zero. These are the number of elements per cross-section, the number of nodes per element and the number of degrees of freedom per node. These variables are cross-section specific, as they are basically different for all eight cases. To give an example for case 111, the number of elements per cross-section is 26, the number of nodes per element is 9 and the number of degrees of freedom per node is 5.

2.3.3 Shell Element Generation

Before the subroutines to generate the shell elements are called, the subroutine "get_shell_info" is called which gives the number of elements per cross-section, the number of nodes per element and the number of degrees of freedom per node values for each of the eight cross-sections. The information obtained in this subroutine is used in the upcoming subroutines (Table 2.3). Although the number of degrees of freedom per node value is constant as 5 for each case, if another element requiring the sixth degree of freedom is to be used,

Table 2.3:Shell Element Information

Case	Number of elements per cross-section	Number of nodes per element	Number of degrees of freedom per node
111	26	9	5
112	26	4	5
121	14	9	5
122	14	4	5
211	52	9	5
212	52	4	5
221	28	9	5
222	28	4	5

by this variable the properties of that element associated with the number of degrees of freedom per node can easily be accommodated into the program.

After the required information for all eight cross-sections are dictated, the shell element generation starts. In the main program the subroutine “elgen_shell” is called and this subroutine call eight subroutines for all eight cases in the form “elgen_shell_xyz” where x,y and z stand for the number of girders, the steel girder section type and the element type used for modeling, respectively as before. For example “elgen_shell_111” stands for the subroutine to generate the shell elements for the case 111.

In the shell element generation subroutines main point is the way the element nodes are numbered because the elements are defined by the nodes. The numbering is done starting from the lower left corner of the element and in counter-clockwise direction. First the corner nodes are numbered and after that, if they exist the mid-nodes are numbered (Fig 2.13). To give an example consider the element 1 of the cross-section 111 (Fig. 2.14).

In Fig. 2.12 the first shell element forming the deck of Case 111 is shown. For the sake of clarity only the node numbers forming that element are shown. As the element node numbering method suggests the first node is numbered as 1. Nodes 3, 111 and 109 follow that node. As the element is a nine-node element having mid-nodes the fifth node is 2. Nodes 57, 110, 55 follow node 2. The last node is the node 56. So in order to define element 1 for the case 111 the required sequence of nodes is, 1, 3, 111, 109, 2, 57, 110, 55, 56.

In order to define all the shell elements along the length of the bridge, first, the element definitions for the first cross-section are completed. After that, by looping over the total number of element divisions, all the shell elements forming the bridge are generated.

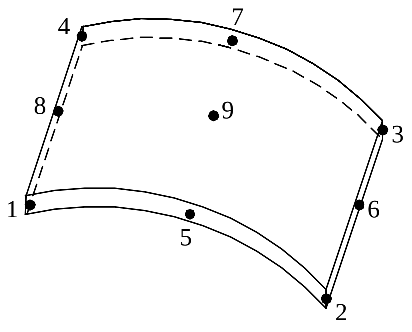


Figure 2.13: 9-node Shell Element Node Numbering

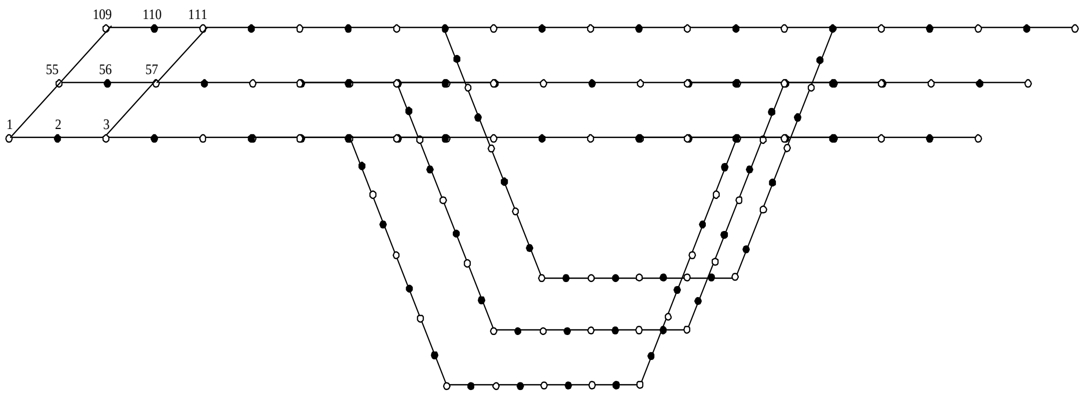


Figure 2.14: Shell Element 1 for Case 111

2.3.4 Internal and External Braces

In order to overcome the stability problems of steel girders of composite bridges internal and external braces are used (Fig 1.1). Internal and external braces increase the lateral rigidity and torsional stiffness of the steel girders especially before the poured deck concrete hardens. In the program, the internal and external braces are modeled as truss systems. As mentioned earlier two types of steel girder sections are implemented into the program: trapezoidal girder and I-girder. Naturally internal braces cannot be used with the I-girders but external braces can be used with both types if dual girder system is utilized. The number of internal braces and number of internal brace elements are different concepts for dual girders as number of internal brace elements is twice the number of internal braces because there exists two girders. Another remainder is that while generating both the internal braces and the external braces, 4 nodes are required.

For both internal and external braces, in the main program, the relationship between the number of internal/external braces and internal/external brace elements are stated. The information supplied in this section is given in Table 2.4.

Table 2.4:Relationships between number of internal/external braces and internal/external brace elements

Case	Number of Internal Brace Elements	Number of External Brace Elements
111	Number of Internal Braces	0
112	Number of Internal Braces	0
121	0	0
122	0	0
211	2 x Number of Internal Braces	Number of External Braces
212	2 x Number of Internal Braces	Number of External Braces
221	0	Number of External Braces
222	0	Number of External Braces

After the information in Table 2.4 is established the subroutines “elgen_int_brace” and “elgen_ext_brace” are called in the main program. These two subroutines again call the secondary subroutines to generate internal and external braces respectively. The general form of these secondary subroutines is “elgen_int_bracexyz/elgen_ext_bracexyz” where x, y and z stand for the number of girders, steel girder section type and the element type used for modeling respectively as before. For example “elgen_int_brace111/ elgen_ext_brace111” stands for the subroutine to generate the internal/external braces for the case 111. Current version of the program utilizes only one type of internal and external braces (Fig. 2.15).

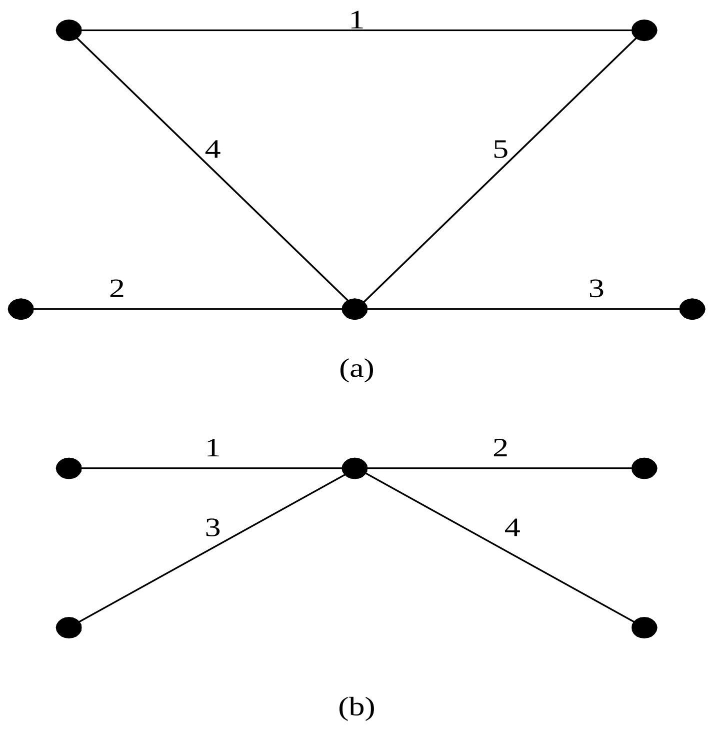


Figure 2.15: (a)External and (b)Internal Brace Types and Nodes

In order to define the internal braces along the length of the bridge, first, the internal brace of the first cross-section is defined and then the rest is generated as there always exists a constant number of nodes between the consecutive cross-sections. Internal brace generation for the first cross-section of the 111 case is shown in Figure 2.16.

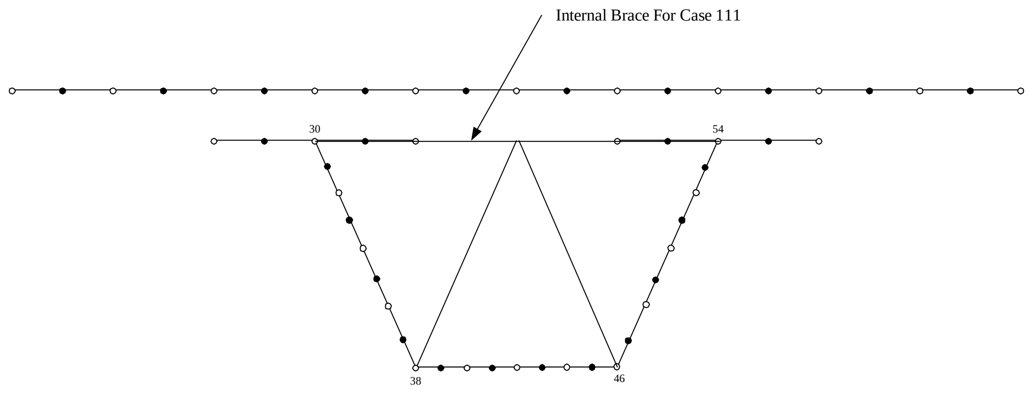


Figure 2.16: Internal Brace and Nodes for Case 111

As shown in Figure 2.16 the number of nodes necessary to define an internal brace for a single girder system is 4. For the sake of simplicity only the nodes for defining the internal braces are shown. The locations of the internal braces were inputted to the program previously. As the total length of the bridge is an integer multiple of the element size, the location of the internal braces are written in terms of the element size. For the four-node element integer division of the location of the internal brace “i” by the element size gives the the location of the internal brace in terms of the internal multiple cross-sections. As for the nine-node element there exists the middle node instead of the “element size”, the user defined internal brace location values are divided by “element size/2”.

For the 111 case, the node locations to define the internal brace for the first cross-section is 30, 54, 38, 46 respectively. By looping over the total number of internal braces and multiplying the above specified node numbers by the internal brace locations and total number of nodes per cross-section value which is 54 in case 111 the internal brace generation for the entire bridge is obtained.

In order to define external braces along the length of the bridge, first, the external brace of the first cross-section is defined and then the rest is generated as there always exists a constant number of nodes between cross-sections. External brace generation for the first cross-section of the 211 case is shown in Figure 2.17.

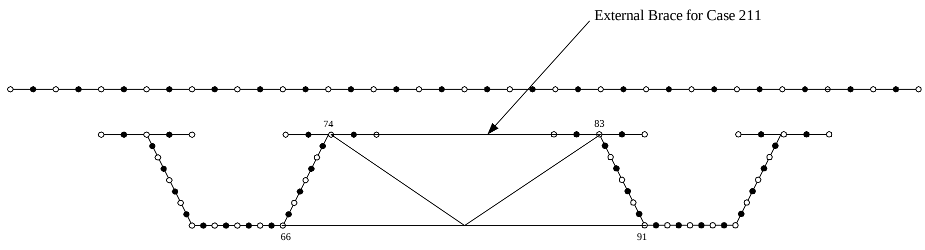


Figure 2.17: External Brace and Nodes for Case 211

2.3.5 Top Lateral Braces

Another precaution to prevent the distortion of the steel section before the deck concrete hardens is to put top lateral braces which connect the two flanges of the trapezoidal girders in single or dual trapezoidal girder systems and the two I-girders in dual I-girder systems. Top lateral braces tend to increase the torsional stiffness of the steel girders by producing pseudo closed sections. In the program the top laterals are modeled as truss elements. There are two types of top laterals that can be used in the program (Fig. 2.18).

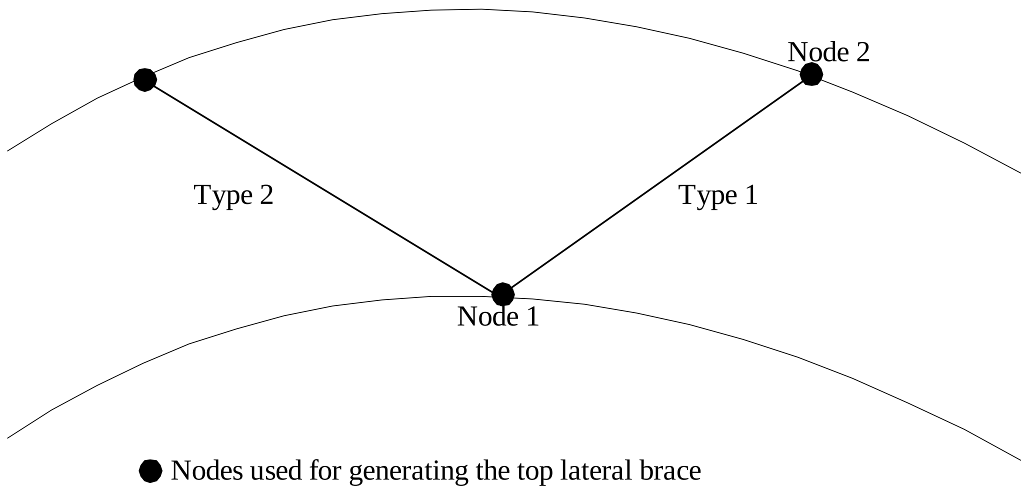


Figure 2.18: Top Lateral Brace Types and Nodes

As the top lateral braces are used for connecting the top flanges of a trapezoidal girder in single or dual girder systems or the two I-girders of a dual I-girder system there are no top lateral braces for single I-girder systems. In dual girder systems the number of the top lateral elements are twice the number of top laterals inputted to the program.

In the main program, the subroutine "elgen_toplt" is called and this subroutine again calls the secondary subroutines in the form "elgen_topltxyz" where x,y and z stand for the number of girders, steel girder section type and finite element type used for modeling, respectively as before. For example "elgen_toplt111" stands for the subroutine to generate the top lateral braces for the case 111.

Top lateral braces for Case 111 is shown in Figure 2.19.

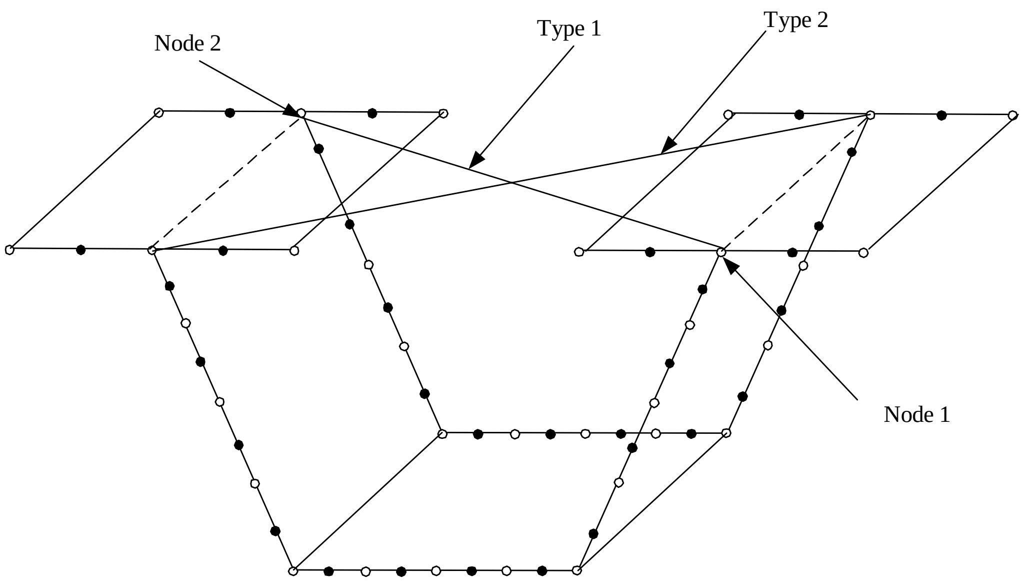


Figure 2.19: Top Lateral Braces for Case 111

2.3.6 Shear Studs

In order to connect the deck concrete to the underlying steel girder(s), stud elements are used. Stud elements play a key role in the development of the composite action between the concrete and steel deck. In the program, the stud elements are modeled as spring elements and it is assumed that three shear studs per flange are used. The number of the shear studs used per cross-section for each of the eight cases are given in Table 2.5.

Table 2.5:The number of Shear Studs Used

Case	Number of Shear Studs Used per Cross-section
111	6
112	6
121	3
122	3

Table 2.5:The number of Shear Studs Used(Continued)

Case	Number of Shear Studs Used per Cross-section
211	12
212	12
221	6
222	6

After the number of studs elements is obtained, the stud element generation starts. In the main program the subroutine "elgen_studs" is called which again calls subroutines in the form "elgen_studsxyz" for all eight cases to generate studs where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling, respectively as before. For example "elgen_studs111" stands for the subroutine to generate the stud elements for the case 111.

The stud elements are generated for every cross-section, so for the same stud element in successive cross-sections, the node numbers used for generation should be increased by the total number of nodes per cross-section value. Stud elements for Case 111 are shown in Figure 2.20.

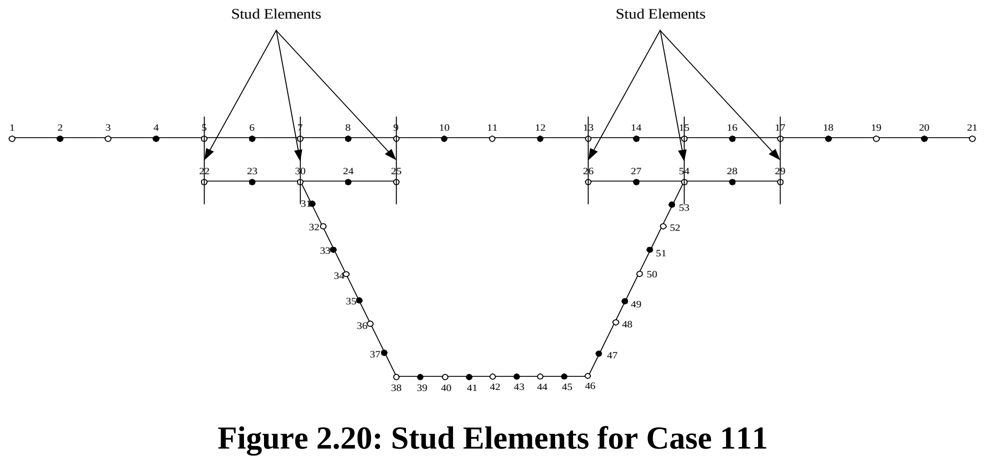


Figure 2.20: Stud Elements for Case 111

Two nodes are necessary to define a stud element. One node is on the concrete deck and the other node is on the steel girder. In fact these two nodes lie on each other. For example to define the stud elements in the first cross-section for the case 111 the necessary nodes on the concrete deck are 22,30,25,26,54,29. Their counterpart on the steel girders are 5,7,9,13,15,17. So the leftmost stud element is defined by the nodes 22 and 5. The rest of the stud elements along the bridge length are generated in a similar manner, simply by adding the multiplication of total number of nodes per cross-section value with the number of cross-sections to the individual node numbers.

2.3.7 Support Elements

In practice, in order to reduce the torsional stresses induced from the loading, diaphragms made of steel plates are used at the support locations as they have very high torsional stiffness. In order to simulate this, very stiff truss systems are used at the support locations. By preventing the relative movements of the edges of the cross-sections rigid diaphragm action is modeled.

Except for the trapezoidal dual girder systems, for all other six sections the number of support elements equal to the number of supports as entered by the user. For trapezoidal dual girder systems the number of support elements equal to three times the number of support elements (Fig 2.21).

To generate the supports, in the main program the subroutine "elgen_support" is called which again calls subroutines in the form "elgen_supportxyz" for all eight cases to generate support elements where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example "elgen_support111" stands for the subroutine to generate the support elements for the case 111.

The support locations are inputted by the user and as it was explained before, these values are divided by the element size for the four-node element and by half the element size for the nine-node element to obtain the support locations in terms of integer number of cross-sections. After the support locations are expressed in terms of the element size, the cross-section associated with this value is determined and the node locations to form the support elements at that particular cross-section are obtained. By this way the support element nodes are obtained.

Four nodes are required to define a support element. For Case 111 the node locations of the support at the first cross-section are 30,54,38 and 46. The other node locations for other support are going to be integer multiples of these node numbers.

2.3.8 Pin Nodes

While modeling the supports, bottom flange nodes of the first support is modeled as pin supports. For the rest of the supports, the bottom flange nodes of the steel girders are modeled as rollers. To give an example for the Case 111, nodes 38,..,46 are pinned if the support is at the first cross-section. The corresponding nodes for the other supports are modeled as rollers. The number of total number of pinned nodes for a bridge is calculated by multiplying the total number of supports by the number of pinned nodes for the first cross-section. Table 2.6 shows the pinned nodes for all eight cases.

To generate the pinned nodes, in the main program the subroutine "gen_pinnodes" is called which again calls subroutines in the form "gen_pinnodesxyz" for all eight cases to generate the pinned nodes where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example "gen_pinnodes111" stands for the subroutine to generate the pinned nodes for the case 111.

Table 2.6:Pinned Nodes

Case	Pinned Nodes	Number of Pinned Nodes
111	38,..46	9
112	20,..24	5
121	26,..30	5
122	14,..16	3
211	58,..66-91,..99	18
212	30,..34-47,..51	10

Table 2.6:Pinned Nodes(Continued)

Case	Pinned Nodes	Number of Pinned Nodes
221	38,..42-55,..59	10
222	20,..22-29,..31	6

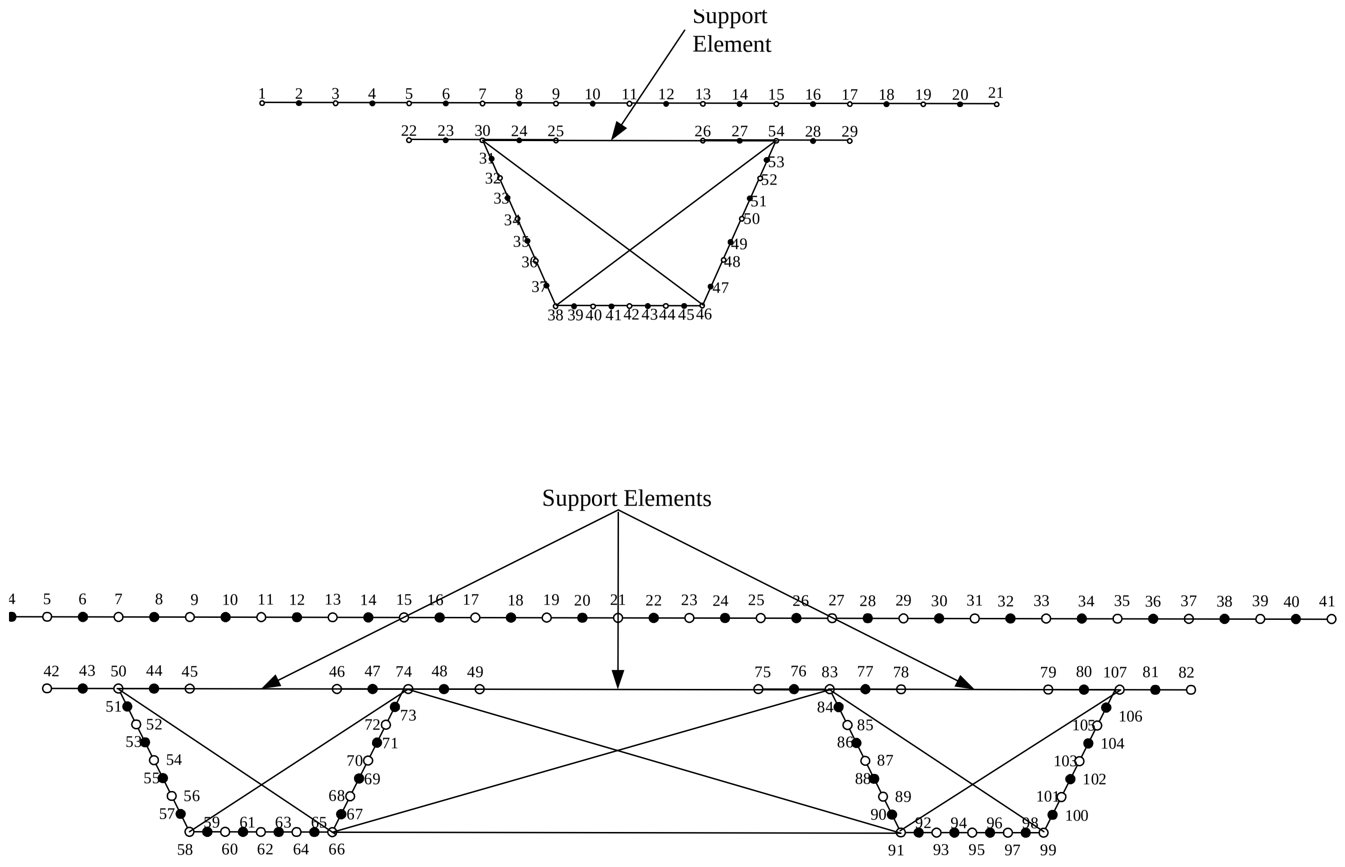


Figure 2.21: Support Elements for Single and Dual Girder Systems

2.3.9 Assigning Shell Element Properties

2.3.9.1 Property Shell Library

As the subject bridges are long and in order to get more accurate results, the program requires finer meshes and the computer memory requirement is high. Also, in the program, there is an option that user can take multiple runs with the program according the pour sequence scheme that he/she inputs. All these tend to increase the physical memory that the programs needs for execution. To give a rough estimate for a Case 111 bridge which is 100 m long and the element size is 0.1 m, the number of the cross-sections formed will be 1 000 and the number of shell elements generated will be 26 000 if we utilize four-node elements. In addition to this, the size of the global stiffness matrix that program assembles will be 130 000 by 130 000.

The required inputs for the shell elements are steel modulus; web, bottom flange and top flange thicknesses of the steel girders, concrete modulus and thickness of the deck. In order to save from memory space required, a property library is formed in the form “prop_sh_lib(n_runs, 4, nwebt+ nbotft +ntft+ n_deck)” where “nwebt” is the different number of web thicknesses, “nbotft” is the different bottom flange thicknesses, “ntft” is the different top flange thicknesses and “n_deck” is the different concrete deck concrete pour sequences. First by looping over number of runs and number of web thicknesses the number of different web thicknesses along the bridge length are entered. This storage is shown in Algorithm 2.4

Algorithm 2.4

```
do ij = 1,number of runs
do ik = 1,number of web thicknesses
prop_sh_lib(ij,1,ik)=steelmodulus
prop_sh_lib(ij,2,ik)=0.3d0
prop_sh_lib(ij,3,ik)=web thickness
prop_sh_lib(ij,4,ik)=1.0d0
end do
end do
```

The second and fourth entries in the above representation are the poisson's ratio and type of the element respectively which are not entered by the user. After the web thicknesses are stored, then the second loop is carried out starting from number of web thicknesses to number of bottom flange thicknesses (Alg. 2.5).

Algorithm 2.5

```
do ij = 1,number of runs
do ik = number of web thicknesses+1,number of web thicknesses+number of bottom flange thicknesses
prop_sh_lib(ij,1,ik)=steelmodulus
prop_sh_lib(ij,2,ik)=0.3d0
prop_sh_lib(ij,3,ik)=botft(ik-nwebt)
prop_sh_lib(ij,4,ik)=1.0d0
end do
end do
```

The property shell library's remaining entries for top flange thicknesses and number of decks are filled in a similar way.

The property shell library is a reference table of properties for the shell elements. Once the shell library is formed, instead of assigning the shell properties for every shell element one by one and repeating the entries for large number of shell elements, the program is structured so that for each element it refers to the associated property by mapping the corresponding shell element property.(Fig. 2.22).

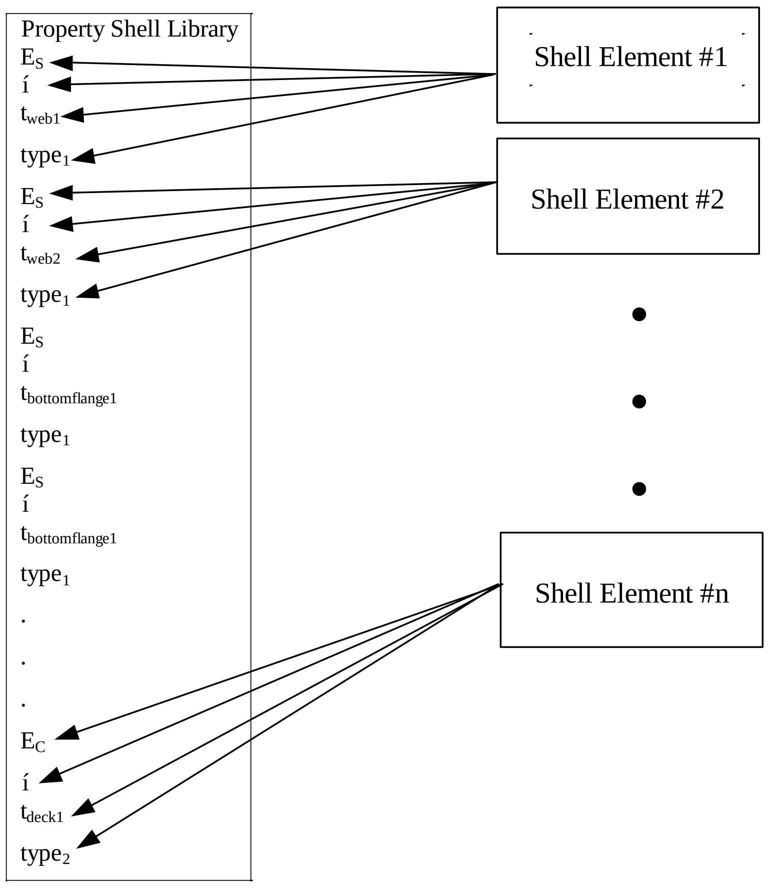


Figure 2.22:Shell Element Property Mapping

2.3.9.2 Property Shell Indexes

After the property shell library is formed, the task is to correctly map the shell element properties to the associated shell elements. The shell elements could be used for modeling the flanges, webs or the concrete deck. They will have different geometric properties and each of these properties should be assigned properly.

The properties are not stored but instead their property shell library indexes are stored. In this way when the global stiffness matrix is assembled the properties of the shell elements are assigned quickly.

In order to assign the property shell index values to the shell elements in the main program a subroutine “form_prop” is called which again calls subroutines of the form “form_propxyz” to form the shell properties for all eight cases where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example “form_prop111” stands for the subroutine to assign the property indexes for the Case 111.

In the “form_propxyz” subroutine first the property shell index array of size “number of shell elements” is formed. After that the internal arrays for the deck, the web, the top flange and the bottom flange thicknesses are obtained. These arrays are used to determine the start and end division numbers for each deck segment, number of web thickness, number of top flange thickness and number of bottom flange thickness values. For example if a bridge is divided into 30 divisions and if we have 3 different top flange thickness values along the bridge and their lengths are same, the start and end divisions numbers for the first, second and third top flange thickness change intervals will be 1 to10,11 to 20 and 21 to 30 respectively. Here for all start and end division numbers, the start division number is forced to be one and end division number is forced to be zero.

After the start and end division numbers are obtained for number of decks, number of web thicknesses, number of top flange thicknesses and number of bottom flange thicknesses the assignment of the property shell indexes to the associated shell elements is started.

Here only the algorithm for the deck thicknesses will be explained and rest proceeds in a similar way. The algorithm to assign the deck thicknesses to shell elements forming the deck for Case 111 is given in Algorithm 2.6.

Algorithm 2.6

```
1. Loop over the user inputted different number of deck thicknesses value.
do im=1,number of decks
    2. Determine ikpro which locates the deck thickness property in the property library
    ikpro=#of web thicknesses+#of bottom flange thicknesses+#number of top flange thicknesses+im
    3. Loop over the previously calculated start and end division numbers for number of decks
    do ij=start division number of deck(im),end division number of deck(im)
        4. The shells forming the deck for the 111 case is numbered as 1 to 10 (Fig.2.4). So loop over the deck shell elements to assign the property shell index values.
        do ik=1,10
            ielnum=ik+(ij-1)*total number of elements per cross-section(26 for Case 111)
            iprop_sh_index(ielnum)=ikpro
        end do
    end do
end do
```

Algorithm 2.6 is repeated for the web, top flange and bottom flange thickness by changing the necessary variables so at the end the properties of all the shells forming the bridge are obtained.

2.3.10 Assigning the Stud Properties

As it will be described in Chapter 3, the studs are modeled as spring elements. The studs are used to connect the steel girders to the concrete deck. To simulate the wet concrete behavior, in addition, the user specifies concrete modulus and the user defines stud stiffness for each concrete pouring sequence. So the stiffness is variable along the bridge length.

As the studs are modeled with spring elements, the only property required for the studs is the stud stiffness. In order to assign the stud properties to the stud elements in the main program, a subroutine “form_stpr” is called which again calls subroutines of the form “form_stprxyz” to assign the stud properties for all eight cases where x,y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example “form_stpr111” stands for the subroutine to assign the stud properties for the Case 111.

In the subroutine “form_stprxyz” first the arrays “prop_stud” of size number of runs by number of stud elements are formed. Then the internal arrays to store the start and end cross-section numbers for different stud properties are generated. The start and end cross-section numbers for different stud property intervals are obtained. The overall start cross-section number is forced to be zero and overall end cross-section number is forced to be the total number of cross-sections. Then looping over the number of runs, the number of deck divisions, start and end cross-section numbers and finally number of studs per cross-section, the stud properties are assigned (Alg. 2.7).

Algorithm 2.7

```
do i = 1,number of runs
do j = 1,number of deck division numbers
do k = start cross-section, end cross-section
do l = 1,number of studs per cross-section
ielnum=ik+(ij-1)* number of studs per cross-section
prop_stud(il,ielnum)=stud_stf(il,1)
end do
end do
end do
end do
```

2.3.10.1 Stud Modification Factors

As it was explained before three studs are assigned to each flange. So for each trapezoidal girder there are 6 studs and for each I-girder there are 3 there studs. However in the program the user is free to enter any number of studs per flange. The number of studs can take values less than three and more than three. In order to simulate this, the individual stud stiffnesses assigned in the previous step are modified by stud modification factors. If the user specified number of studs per flange is less than three, the stud modification factor for the required number of studs is set to 1.0 and the rest is deamplified by a very small number (1E-8 in the program). If the user specified number of studs per flange is greater than three, the stud stiffness are amplified by the total number of stud stiffness.

In order to obtain the stud modification factors, in the main program a subroutine “form_std_mod” is called which again calls subroutines of the form “form_std_modxyz” to assign the stud modification factors for all eight cases where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example “form_std_mod111” stands for the subroutine to assign the stud modification factors for the Case 111.

2.3.11 Assigning the Properties for Internal, External and Top Lateral Braces

As compared with the shell elements, the numbers of the internal, external and top lateral braces are small. So there is no storage problem when the properties of internal, external and top lateral braces are assigned. For all brace types the required inputs are the steel modulus, area of the brace and the type of the brace. By looping over each brace element these properties are assigned to the braces one by one. For internal and top lateral braces in dual girder systems as the internal and top lateral braces in one of the girders is the same as those of the other, first the properties of the braces of one girder is assigned and then it is copied for the one, on the other girder. Property arrays for internal, external and top lateral braces are of size three by the total number of respective brace types.

2.4 Processor Module

2.4.1 Preliminaries

After the geometries are formed and the material properties are assigned in the Pre-Processor Module, the Processor Module starts. In the Processor Module, basically the stiffness matrices for all elements are defined, the global stiffness matrix and the load vector are assembled, equilibrium equations are solved and the nodal displacements are obtained. In this section, the programming technique will be discussed and the theoretical background will be given in Chapter 3.

2.4.2 Gaussian Quadrature Data and Shape Function Arrays

Because of the complex algebraic expressions present in element stiffness matrix computations the exact evaluation of these integrals is not always possible. If this is the case these integrals should be numerically evaluated. This process is called numerical integration. Among the numerical integration techniques, the most popular one is the Gaussian Quadrature. In Gaussian Quadrature the integral expression is approximated as a series of sums of multiplication of quadrature points and quadrature weights.

In the program Gaussian Quadrature parameters for nine-node and four node elements are obtained. In order to obtain the quadrature points and weights for the nine-node element a subroutine called “gauss3x3” is called in the main program. This subroutine stores the Gaussian quadrature points and weights for nine integration points. Likewise a subroutine called “gauss2x2” is called in the main program for the four-node element. This subroutine stores the Gaussian quadrature points and weights for four integration points.

After the Gaussian quadrature points and weights for the nine-node and four-node elements are defined the shape functions and their derivatives are obtained for the two types of elements.

For the nine-node element, in the main program, a subroutine called “shaper9” is called. In this subroutine the shape functions and the first and the second derivatives of the shape functions are stored. Likewise for the four node element, in the main program, a subroutine called “shaper4” is called. In this subroutine the shape functions and the first and the second derivatives of the shape functions for the 4-node element are stored. The shapes functions are stored in the array “shapes”, the first and second derivatives are stored in the “dshapes” array.

2.4.3 Determination of the Number of Nonzero Entries in the Global Stiffness Matrix

The sparse solver package developed by Compaq, which is a part of the Compaq Extended Math Library (CXML) and implemented into the program requires only the nonzero entries of the upper triangular half of the symmetric structural stiffness matrix be stored. Also auxiliary vectors should be formed to define the locations of the nonzero entries should be supplied. Using these matrices, the solver is able to reorder and factor the stiffness matrix and solve for the displacements.

In order to form the auxiliary matrices, first of all, a temporary array called “ic” is formed that contains all the nodes. “ic” is a 10 by total number of elements array. The total number of elements is found by summing the number of shells, internal braces, external braces, top lateral braces, studs and support elements. In a typical representation of the form “ic(a,b)” “a” stands for the number of nodes and “b” is the element number. As the total number of nodes per any type of element is 9(nine-node shell element) the first dimension of the array “ic” is chosen to be between 0 and 9. While forming the “ic” array, first the array is defined and entries are set to zero. Then a subroutine called “form_ic” is called in the main program. In this subroutine starting with shell elements and proceeding with the internal braces, external braces, top lateral braces, studs and support elements the array entries are filled.

The next step is to form the “invinc” array which contains information about which node is connected to which elements. In the invinc(a,b) representation “a” denotes up to how many elements are connected to the subject node and “b” denotes the number of nodes. The algorithm for this process is given in Algorithm 2.8.

Algorithm 2.8

1. Form the "invinc" array and set the array entries to zero.

```
invinc(0:15,total number of nodes) = 0
```

2. Loop over the total number of elements to find which node is connected to which element and to how many elements.

```
do ik= 1, total number of elements
  do ij=1, ic(0,ik)
    invinc(0, ic(ij, ik))=invinc(0, ic(ij, ik))+1
    invinc((invinc(0, ic(ij, ik))), ic(ij, ik))=ik
  end do
end do
```

In order to find the number of nonzeros in the stiffness matrix subroutine "form_nonzero" is called in the main program. This subroutine computes the number of nonzero entries in the structural stiffness matrix. In the subroutine "form_nonzero" first the arrays "invinc" and "ic" array are retrieved. Then nonzero count is set to zero. A loop over all the number of nodes is started. Then another loop is started in order to find which elements have that node. In the program each node has five degrees of freedom (DOF). After the elements connected to subject node is found, the other nodes of those elements are determined. If a node does not belong to any element connected to the subject node, the DOF's associated with that node for the DOF's of the subject node will be zero. The above discussion is given in Algorithm 2.9.

Algorithm 2.9

```
1. Start a loop over all nodes.
2. Define a dummy array of size, say, 200 and set all entries to 0
do i = 1,200
idum1(i) = 0
3. Loop over all elements that contain that node(Guarantee that the node is not written before and greater than the previous node)
do ikl=1,invinc(0,ij)
ie=invinc(ikl,ij)
ldum=ldum+1
idum1(ldum)=ic(ik,ie)
4. Compute the number of nonzero entries in the stiffness matrix
ldum2=ldum*number of DOF's per node
do ikl=1, number of DOF's per node
irclg=ldum2-ikl+1
nonzeros=nonzeros+irclg
5. Go to 1
```

As it was explained before the solver package implemented in the program requires that the structural stiffness matrix be expressed as a vector of size “nonzeros”. By this way the zero elements of the structural stiffness matrix is not stored. The solver reorders the structural stiffness matrix according to the “irowindex” and icolumns” vectors. “irowindex” is a vector of size “number of nodes*number of degrees of freedom per node + 1”. “icolumns” vector is vector of size “nonzeros”. “irowindex” and “icolumns” vectors define the location of every nonzero entry in the structural stiffness matrix.

2.4.4 Formation of the “irowindex” and “icolumns” vectors

After “irowindex” and “icolumns” vectors are allocated in the main program, the subroutine “form_connect” is called. The subroutine “form_connect” fills the entries of the “irowindex” and “icolumns” vectors. The entries in “irowindex” vector indexes the first nonzero entry in an upper triangular matrix in a row. The entries are numbered from top to bottom and from left to right. This requires that row index number of the (i+1)th row minus the row index number of the ith row gives the number of nonzero entries in the ith row. In order to force this relationship hold every time, a dummy row index is added as the last element of “irowindex” array which is one more than the previous. Therefore altogether there are “n” rows but “(n+1)” row indexes. While filling the entries of the “icolumns”, the column numbers of each nonzero entry of the structural stiffness matrix is stored. The algorithm for the above explained procedure is given Algorithm 2.10.

Algorithm 2.10

```
1. Row index number for the first entry is set to 1
irowindex(1)=1
2. Start a loop over all nodes.
3. Define a dummy array of size, say, 200 and set all entries to 0.0
do i = 1,200
idum1(i) = 0
4. Loop over all elements that contain that node(Guarantee that the node is not
written before and greater than the previous node)
do ikl=1,invinc(0,ij)
ie=invinc(ikl,ij)
ldum=ldum+1
idum1(ldum)=ic(ik,ie)
5. Sort the entries of idum1 in ascending order
6. Convert the obtained nodal connectivity to DOF connectivity
ldum2=0
```

```
do ji=1,ldum
istr=(idum1(ji)-1)*number of DOF's per node
do ki=1,5
ldum2=ldum2+1
idum2(ldum2)=istr+ki
end do
end do
7. Obtain the "irowindex" and "icolumns" vectors
do ikl=1, number of DOF's per node
irown=(ij-1)* number of DOF's per node +ikl
irclg=ldum2-ikl+1
idu7=0
do il=ikl,ldum2
idu7=idu7+1
icolumns(irowindex(irown)-1+idu7)=idum2(il)
end do
irowindex(irown+1)=irowindex(irown)+irclg
end do
8. Go to 1.
```

After the required vectors are obtained the "invinc" and "ic" matrices are deallocated in the main program.

2.4.5 Formation of the Structural Stiffness Matrix

As mentioned before, the direct and indirect solvers implemented into the program require the structural stiffness matrix be reordered and expressed as a column vector. The structural stiffness matrix, denoted by "ssm" in the program, is of size "nonzeros". In order to form the structural stiffness matrix for each desired run, the entries are first set to 0.0. Also the entries of the required right hand side of the system of equilibrium and displacement vectors, "rhs" and "uv" respectively are set to 0.0. The size of these two vectors is number of nodes times number of degrees of freedom per node.

After “ssm”, “rhs” and “uv” entries are set to zero, the assembly of the local stiffness matrix entries into the global stiffness matrix is started. The assembly process is carried out for the shell, top lateral, external, internal, support and stud elements individually.

2.4.5.1 Assembly of Shell Elements

The girders and deck of a typical bridge is modeled with shell elements so the most common element of the finite element mesh is the shell element. As mentioned before there are two types of shell elements implemented into the program so this results in some differences in shell assembly process.

In the main program the subroutine “assemb_shell” is called. In the subroutine first a loop is started over all the shell elements. For each shell element the x, y and z coordinates and the corresponding Q, R, and V vectors of the nodes are retrieved. After that the material properties of the individual shell elements are assigned using property shell library and property shell index number. The next step is to initialize the element force “ef” and element stiffness “ek” matrices. In the subroutine another subroutine, “l_r_shell3d02”, is called which computes the element stiffness matrix in local coordinates and then rotation matrix of the associated element and finally the stiffness matrix in global coordinates. The theoretical background will be discussed in Chapter 3.

After the rotated element stiffness matrix is obtained, then by looping over the number of nodes per element and number of degrees of freedom per node the related local degrees of freedom and global degrees of freedom are paired. By looping over the number of degrees of freedom per element the row index and column number of the element stiffness matrix entry is mapped on the global structural stiffness matrix. Comparing the elemental row index and column numbers with the global counterparts and adding if they coincide, the global structural stiffness matrix entries are filled for shell elements. The above discussion is given in Algorithm 2.11.

```
Algorithm 2.11
1. Start a loop over all shell elements.
2. Start a loop over each individual number of nodes per shell element.
3. Retrieve the nodal coordinates, Q, R, and V vectors.
4. Assign the material properties (Modulus of Elasticity(E), Poisons ratio(ν),
thickness(h) and type) to the shell element.
5. Initialize the element stiffness matrix.
ek(i,j) = 0.0
6. Obtain the rotated element stiffness matrix.
7. Transform the local degree of freedom entries of the element stiffness
matrix into the global global degrees of freedom.
8. Fill the global structural stiffness matrix for the shell element.
9. Go to 1.
```

2.4.5.2 Assembly of Top Lateral Braces

Top lateral braces are simple truss elements so their element stiffness matrix computations does not require much effort. Simple truss formulation will do and then the rotated element stiffness matrix entries are transformed into global coordinates.

In order to assemble the top lateral braces, in the main program, the subroutine "assemb_toplt" is called. In the "assemb_toplt" subroutine first a loop is started over all top lateral brace elements and then the top lateral brace nodal coordinates are retrieved. After that the material properties are assigned to individual top lateral brace elements. The next step is to initialize the element stiffness matrix of individual top lateral brace elements. Then a subroutine "get_trussk" is called in the subroutine to compute the rotated element stiffness matrix of the top lateral brace element which itself is a simple truss element. After obtaining the rotated element stiffness matrix the element degrees of freedom in local coordinates are transformed into global degrees of freedom by looping over each degree of freedom. By looping over the number of degrees of freedom per element the row index and column number of the element stiffness matrix entry is mapped on the global structural stiffness matrix. Comparing the elemental row index and column numbers with the global counterparts and adding if they coincide the global structural stiffness matrix entries are filled for top lateral brace elements.

2.4.5.3 Assembly of Internal Braces

As the internal braces and external braces are composed of more than one truss element, their assembly process is not as simple as the top lateral brace elements'. In the assembly of internal and external braces static condensation technique is used. First of all the truss elements are assembled together to form a superelement. Second the degrees of freedom which are not shared with the steel girder are condensed out.

In the assembly of the internal braces, first, in the main program the subroutine "assemb_intbr" is called. In the subroutine a loop is started over all internal brace elements'. The nodal coordinates are retrieved and material properties are assigned to each internal brace element. The twelve by twelve element stiffness matrix "ek" is initialized to zero. Then another subroutine "get_intbrk" is called which computes the assembled stiffness matrix of the components of the internal braces, condenses the assembled internal brace element matrix for the unshared degrees of freedom with the steel girders, and finally gives the internal brace element stiffness matrix. The algorithm for the subroutine "get_intbrk" is given Algortihm 2.12.

After obtaining the element stiffness matrix the element degrees of freedom in local coordinates are transformed into global degrees of freedom by looping over each degree of freedom. By looping over the number of degrees of freedom per element the row index and column number of the element stiffness matrix entry is mapped on the global structural stiffness matrix. Comparing the elemental row index and column numbers with the global counterparts and adding if they coincide the global structural stiffness matrix entries are filled for internal brace elements.

2.4.5.4 Assembly of External Braces

The assembly of process of the external braces are similar to the internal braces. In the assembly of the external braces, first, in the main program the subroutine “assemb_extbr” is called. In the subroutine a loop is started over all external brace elements. The nodal coordinates are retrieved and material properties are assigned to each external brace element. The twelve by twelve element stiffness matrix “ek” is initialized to zero. Then another subroutine “get_extbrk” is called which computes the assembled stiffness matrix of the components of the external braces, condenses the assembled external brace element matrix for the unshared degrees of freedom with the steel girders and finally gives the external braces element stiffness matrix. The algorithm for the subroutine “get_extbrk” is similar to Algorithm 2.12.

Algorithm 2.12

1. The internal braces are actually composed of five nodes so the coordinates of all the four nodes are retrieved and the coordinates of the fifth node is calculated form the other two (Fig. 2.23).

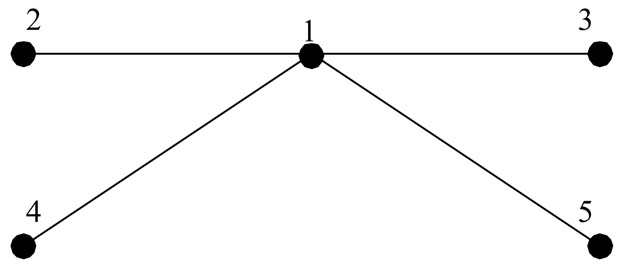


Figure 2.23: Internal Brace Nodes

coordinates of 1 = (coordinates of 2 + coordinates of 3)/2

2. The fifteen by fifteen element stiffness matrix is initialized to zero.
3. A loop is started over all four truss elements forming the internal brace.
4. The material properties are assigned to the truss elements.

5. The element stiffness matrices of all truss elements are initialized to zero.

6. Rotated element stiffness matrices are formed by calling the "get_trussk" subroutine.

ek2(15,15)

7. The positions of the entries of the rotated element matrix in the internal brace element stiffness matrix are obtained.

8. Artificial stiffness are assigned to the nodes where there may be zero stiffness.

ek2(1,1) = ek2(1,1) + 1.e-5

ek2(2,2) = ek2(2,2) + 1.e-5

ek2(3,3) = ek2(3,3) + 1.e-5

9. Correction factor is calculated for all the entries of ek2.

factor = -ek2(j,i)/ek2(i,i)

10. The entries of ek2 are recalculated taking into account the factor.

ek2(j,k)=ek2(j,k)+factor*ek2(i,k)

11. The condensed internal brace element stiffness matrix is obtained by ignoring the first three entries of ek2 from left to right and from top to bottom

ek(12,12) = ek2(15-3,15-3)

The assembly of the element stiffness matrix into the global stiffness matrix for the external braces follows the same procedure explained for the internal braces. The nodes for the external brace elements are given in Figure 2.24.

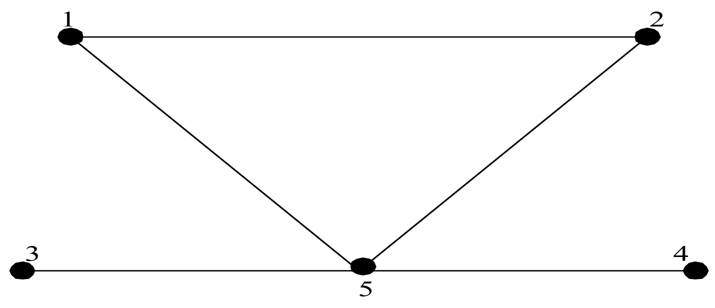


Figure 2.24: External Brace Nodes

2.4.5.5 Assembly of Support Elements

As discussed previously the supports are modeled as rigid truss systems so that the torsional stresses induced from the loading can be handled.

In the assembly of the supports the subroutine “assemb_support” is called in the main program. In this subroutine, first, a loop is started over all the support elements. For each support element the nodal coordinates are retrieved and the material properties are assigned. The twelve by twelve element stiffness matrix is initialized to zero. After that the subroutine “get_supportk” is called to compute the element stiffness matrix for each support element. The algorithm for the subroutine “get_supportk” is similar to that of shell elements.

The assembly of the element stiffness matrix into the global stiffness matrix for the support elements follows the same procedure explained for the internal braces.

2.4.5.6 Assembly of Studs

In the program the stud elements are modeled with spring elements so their stiffness matrix computation is simple and explained in Chapter 3.

In the main program the subroutine “assemb_stud” is called. In this subroutine first a loop is started over all stud elements. For each stud element the stud stiffness is modified with corresponding stud modification factor. The six by six element stiffness matrix is initialized to zero. After that the subroutine “get_studk” is called to compute the element stiffness matrix for each stud element. Having obtained the stiffness matrix for each stud element, now next step is to assemble the entries to the global structural stiffness matrix.

The assembly of the element stiffness matrix into the global stiffness matrix for the studs follows the same procedure explained for the internal braces.

2.4.6 Modification for Support Conditions

As the supports are modeled as rigid truss systems they do not allow displacements for some of their degrees of freedom. So some of the structural displacements should be modified to simulate this behavior. Penalty method is used to prescribe the support conditions. Specifically at the support locations where previously pinned nodes were assigned the structural stiffness matrix entries are modified by adding a very big number to the associated entry. By this way the structural displacement at the subject degree of freedom of the subject node is zero.

In order to modify for the support conditions, in the main program, a subroutine “apply_support” is called. In this subroutine a loop is started over all pinned nodes and for the proposed degrees of freedom, to the corresponding entries in the global structural stiffness matrix a very big number, say 1.0 E20, is added.

2.4.7 Formation of the Load Vector

After the structural stiffness matrix is obtained, the next step is to compute the load vector. The load vector comprises the right hand side of the system of equilibrium equations so this vector is denoted by “rhs” in the program.

To compute the load vector, in the main program, the subroutine “assign_dist_load” is called which again calls subroutines in the form “assign_dist_loadxyz” for all eight cases to form the load vectors where x,y and z stand for the number of girders, the steel girder section type and the element type used for modeling respectively as before. For example “assign_dist_load111” stands for the subroutine to generate the load vector for the case 111.

In a typical subroutine of the form “assign_dist_loadxyz”, first the start and end cross-section numbers of concrete pour intervals of the deck are determined. The start and end cross-section numbers are forced to be zero and "ncsec" respectively. On each cross-section point loads are acted on the specified nodes. These nodes are repeating nodes for each cross-section. Table 2.7 lists the nodes on which the equivalent nodal loads are acted for each cross-section case.

On each specified node the load coming from the tributary area of each cross-section are distributed.

Table 2.7:Load Nodes

Case	Load Nodes
111	22,30,25,26,54,29
112	12,16,13,14,28,15
121	14,16,18
122	8,9,10
211	42,50,45,46,74,49,75,83,78,79,107,82
212	22,26,23,24,38,25,39,43,40,41,55,42
221	26,28,30,43,45,47
222	14,15,16,23,24,25

2.4.8 Solution for the Displacements

After the global stiffness matrix and the load vector are obtained nodal displacements are obtained by solving the system of equations. The global stiffness matrix and the load vector are given as input to the solver.

In the main program the subroutine "solver" is called. This subroutine contains the previously explained direct sparse solver of the CXML. Another solver called "ITPACK" which is an iterative solver was adopted into the program to compare it with the direct solver. The results of this comparison will be given in Chapter 4.

The subroutine "solver" conveys the calculated structural stiffness matrix and load vector to the solver. The solver solves the system of equations and a solution vector "uv" is obtained. "uv" contains the structural displacements for each degree of freedom.

2.5 Post-Processor Module

2.5.1 Preliminaries

After the structural displacements are obtained they are post-processed to obtain the average vertical cross-sectional deflections, cross-sectional rotations, top lateral, internal, external brace forces, cross-sectional forces and stresses.

2.5.2 Cross-sectional Deflections

After solving for the displacements the vertical deflections of each node are obtained but a representative value should be specified to give the vertical deflection of the bridge. This is essential for the evaluation of the deflection of the bridge along the span.

In order to post-process the vertical deflections, in the main program, the subroutine "post_defl" is called which again calls subroutines in the form "post_deflxyz" for all eight cases to post-process the deflections where x, y and z stand for the number of girders, the steel girder section type and the element type used for modeling respectively as before. For example "post_defl111" stands for the subroutine to post-process the deflections for the case 111.

If the post-processed cross-section is a single girder system, the average deflection of the nodes on the bottom flange corners are calculated. This value is a good approximation for the vertical deflection of the subject bridge at its specified location. To be clear, for the Case 111 the average vertical deflection of the nodes 38 and 46 can be used as the bridge deflection at the specified location.

For dual girder systems the deflections under each girder are given separately.

2.5.3 Cross-sectional Rotations

The cross-sectional rotations are necessary in the design of steel girders which are susceptible to torsional stresses.

In order to calculate cross-sectional rotations, in the main program, the subroutine "post_rot" is called which again calls subroutines in the form "post_rotxyz" for all eight cases to compute the cross-sectional rotations where x, y and z stand for the number of girders, steel girder section type and finite element type used for modeling respectively as before. For example “post_rotl111” stands for the subroutine to compute the cross-sectional rotations for the case 111.

If the cross-sectional rotations for a single girder system is to be calculated, the difference of the vertical deflections of the nodes on the bottom flange corners of the steel girder are divided by the bottom flange length. For the sake of clarity, for Case 111, the cross-sectional rotation is found by dividing the difference between the vertical deflections of the nodes 38 and 46 by the bottom flange length.

For dual girder systems the cross-sectional rotations of each girder are calculated and reported separately.

2.5.4 Top Lateral Brace Forces

The forces induced in the top lateral braces are also computed in the program.

In order to calculate top lateral brace forces, in the main program, the subroutine “post_toplt2” is called. In this subroutine first a loop started over all top lateral brace elements. Then for each top lateral brace element the coordinates of the element nodes are retrieved. After that material properties are assigned to each element. In the next step element load vector and element stiffness matrix are initialized to zero. The subroutine “get_trussk” is called and the element stiffness matrix entries are formed again. The positions of the displacements corresponding to the element nodal degrees of freedom in the global displacement vector are located and they are stored in an array called “displ”. Element force vector is obtained by multiplying the element stiffness matrix with the “displ” vector. Unit vectors are obtained in local x, y and z directions and they are multiplied with the corresponding load vector entries to obtain the resultant loads. The axial force in the top lateral element is calculated.

2.5.5 Internal Brace Forces

In order to calculate internal brace forces, in the main program, the subroutine “post_intbr” is called. In this subroutine first a loop started over all internal brace elements. Note that each internal brace element is composed of four truss elements. For each of these truss elements, the coordinates of the element nodes are retrieved. In this stage only nodal coordinates shared with the steel girder are retrieved. After that, material properties are assigned to each element. In the next step element load vector and element stiffness matrix for the super element are initialized to zero. The subroutine “get_intbrk” is called and the element stiffness matrix entries are formed again. The positions of the displacements corresponding to the element nodal degrees of freedom in the global displacement vector are located and they are stored in an array called “displ”. Element force vector is obtained by multiplying the element stiffness matrix with the “displ” vector. The coordinates of all five nodes of the internal brace elements are retrieved. For each truss member forming the internal brace superelement unit vectors are obtained in local x, y and z directions and they are multiplied with the corresponding load vector entries to obtain the resultant loads. The axial force in each of the truss elements forming the internal brace are obtained.

2.5.6 External Brace Forces

In order to calculate external brace forces, in the main program, the subroutine “post_extbr” is called. In this subroutine first a loop started over all external brace elements. Note that each external brace element is composed of four truss elements. Then for each truss element the coordinates of the element nodes are retrieved. In this stage only nodal coordinates shared with the steel girder are retrieved. After that material properties are assigned to each element. In the next step element load vector and element stiffness matrix for the super element are initialized to zero. The subroutine “get_extbrk0” is called and the element stiffness matrix entries are formed again. The positions of the displacements corresponding to the element nodal degrees of freedom in the global displacement vector are located and they are stored in an array called “displ”. Element force vector is obtained by multiplying the element stiffness matrix with the “displ” vector. The coordinates of all five nodes of the external brace element are retrieved again. For each truss member forming the external brace superelement unit vectors are obtained in local x, y and z directions and they are multiplied with the corresponding load vector entries to obtain the resultant loads. The axial force in each of the truss elements forming the external brace are obtained.

2.5.7 Cross-sectional Forces

The last step in the post-processing part is to compute the internal forces. In a cross-section along the bridge 3 force components, in the direction of previously defined Q, R, and V vectors, and two couples in the direction of previously defines Q and R vectors are defined. These forces and moments are denoted by “$\mathbf{f_q}$”, ”$\mathbf{f_r}$”, ”$\mathbf{f_r}$”, ”$\mathbf{am_q}$”, ”$\mathbf{am_r}$” in the program, respectively (Fig. 2.25).

In order to compute the cross-sectional forces, first in the main program, the subroutine “post_csec_for” is called. The subroutine “post_csec_for” itself calls eight subroutines in the form “post_csec_forxyz” to compute the cross-sectional forces for eight different cases. Here x stand for the number of girders, y stands for the steel girder section type and z stands for the finite element type used for shell modeling. The cross-sectional forces are computed in any cross-section where the plane is an interface between two sets of shell elements. This means that the cross-sectional forces are computed at distances of integer multiple of element size. The desired forces can be outputted at either every element boundary or at every user defined integer multiple of divisions. For each element, the cross-sectional forces are computed for the rear nodes (nodes 3,4 and 7 in shell numbering) and then accumulated for the cross-section (Fig. 2.26).

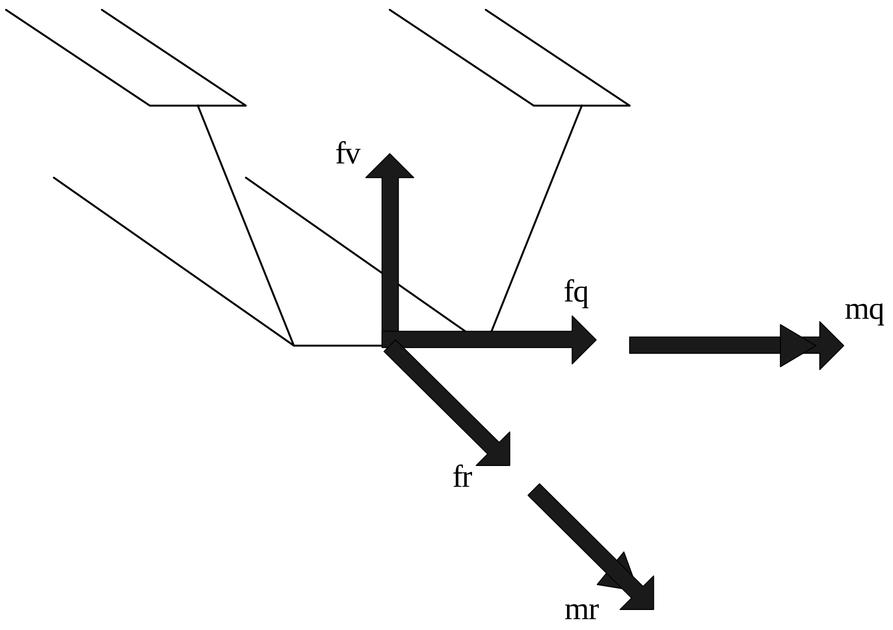


Figure 2.25: Cross-sectional Forces

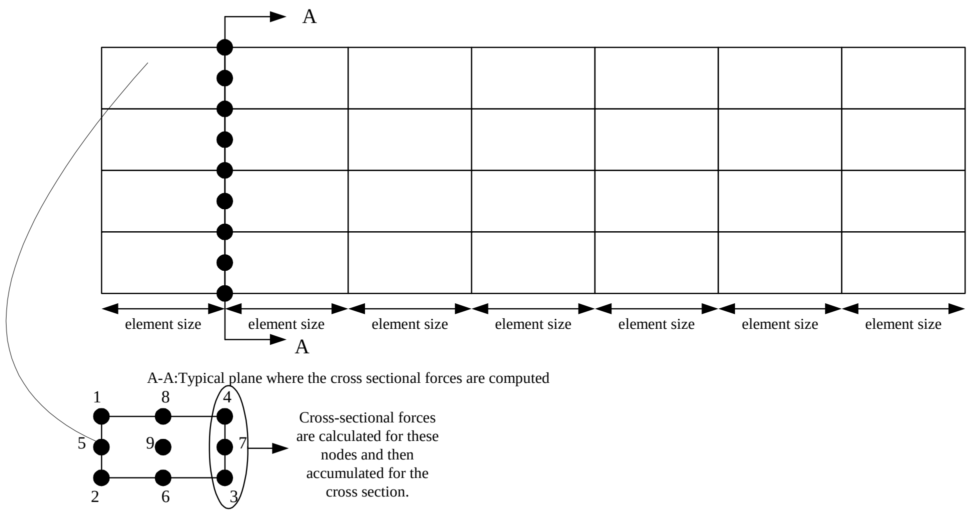

A
element size
A
A-A:Typical plane where the cross sectional forces are computed
1
8
4
5
9
7
2
6
3
Cross-sectional forces are calculated for these nodes and then accumulated for the cross section.

Figure 2.26: Computation of Cross-sectional Forces

When the internal moments are calculated at the nodes the nodal moments are calculated first. After that the moments created by the nodal forces about a reference node within the cross-section are determined. These two values are summed and the cross-sectional forces are determined for the desired direction. The reference node is within the symmetry plane of the cross-section.

In a typical subroutine of the form “post_csec_forxyz”, first the location where the cross-sectional forces are to be computed is determined. At that cross-section the nodes where the nodal forces are going to be calculated are determined together with the reference node. For each node Q, R, and V vectors are retrieved and the x, y and z coordinates of the reference node is assigned to three unique variables, namely “xref”, “yref” and “zref”. The variables to store the cross-sectional forces are set to zero. After that a loop is started over all the nodes of the subject cross-section. For each node the $\mathbf{f_q}$, $\mathbf{f_r}$ and $\mathbf{f_v}$ forces and **am** and **aq** moments are computed by calling the subroutine “post_sh_nfor9” for the nine-node element. For the four-node element the corresponding subroutine is “post_sh_nfor4”. Within the loop, the computed nodal forces are summed to get the cross-sectional $\mathbf{f_q}$, $\mathbf{f_r}$ and $\mathbf{f_v}$ forces. The vertical distances of the nodes to the reference node are determined and by multiplying these with the corresponding nodal forces the moments created by the nodal forces are determined. Also the normal stress and shear stresses within each cross-section are determined in the subroutine. The algorithm for the subroutine “post_sh_nfor9” is given in Algorithm 2.13. The algorithm for the subroutine “post_sh_nfor4” is similar.

Algorithm 2.13

```
1. Loop over the nodes within the shell element to retrieve the coordinates, Q, R, and V vectors.
2. Assign the material properties.
3. Initialize the shell element force vector and stiffness matrix.
4. Retrieve the stiffness matrix of the element.
5. Locate the element degrees of freedom in the global stiffness matrix, and displacement vector.
6. Using the corresponding displacements at the rear nodes compute the nodal forces.
7. Transform the calculated forces and moments to global coordinates.
8.  Call the related subroutine to compute the nodal stresses.
9. Go to 1
```

CHAPTER 3

NUMERICAL MODELING DETAILS AND PROGRAM VERIFICATION

3.1 Modeling the Physical System

In the literature there are several methods to analyze the steel girder – concrete deck interaction. In one of these methods the concrete deck is modeled with brick elements and the steel girder is modeled with shell elements (Tarhini and Frederick, 1992). The studs are modeled with spring elements. The shortcoming with this method is the large number of brick elements, thus large number of degrees of freedom required to capture the flexural response with sufficient accuracy (Fig. 3.1).

In another approach, both the steel girder and the concrete deck are modeled with shell elements (Brockenrough, 1986, and Tabsh and Sahajwani, 1997). The concrete deck and the steel girders are connected together with connector (beam) elements. The length of the connector elements has to be chosen by the analyst to properly model the offset between the neutral axis of the top flange of the girders and that of the deck. This approach is the most popular technique presented in the literature but there is no common consensus on how to choose the connector length.

In the developed software another approach was employed which addresses both of the above-mentioned problems. In a given cross-section two types of shell elements are used for modeling. In the shell element formulation the three-dimensional domain is represented by a surface. For steel sections, the reference surface is the middle surface, whereas, for the concrete deck, the bottom surface is the reference surface (Fig. 3.1). Steel sections and the concrete deck are connected to each other by spring elements, which represent the stud connectors. With this modeling the number of degrees of freedom is reduced as compared with the brick modeling. In addition to this, it properly models the interface behavior by eliminating the beam elements and including the girder offset by using the bottom shell surface as the reference surface.

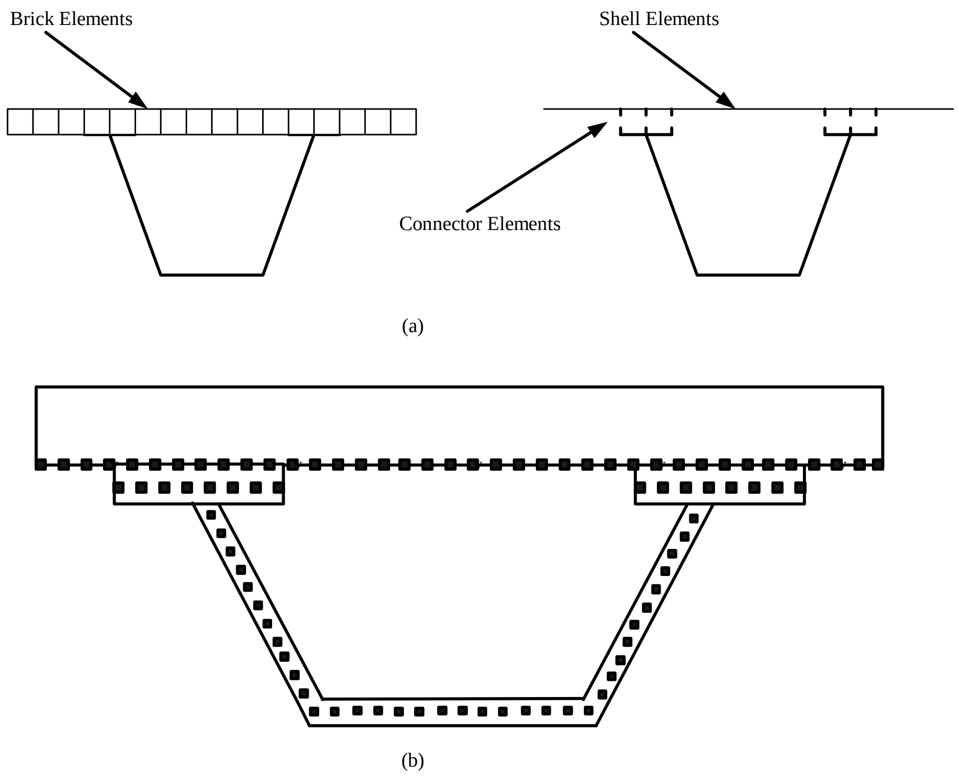


Figure 3.1 (a) Different Modeling Techniques for Deck-Flange Interface, (b) Reference Surfaces for Shell Elements

Three dimensional models of a double I-girder straight, and a single box-girder variable degree of curvature bridge are shown in Figure 3.2. The nodal coordinates and elements are generated using the developed program and then imported to the ANSYS program which is a general purpose finite element modeling and analysis tool.

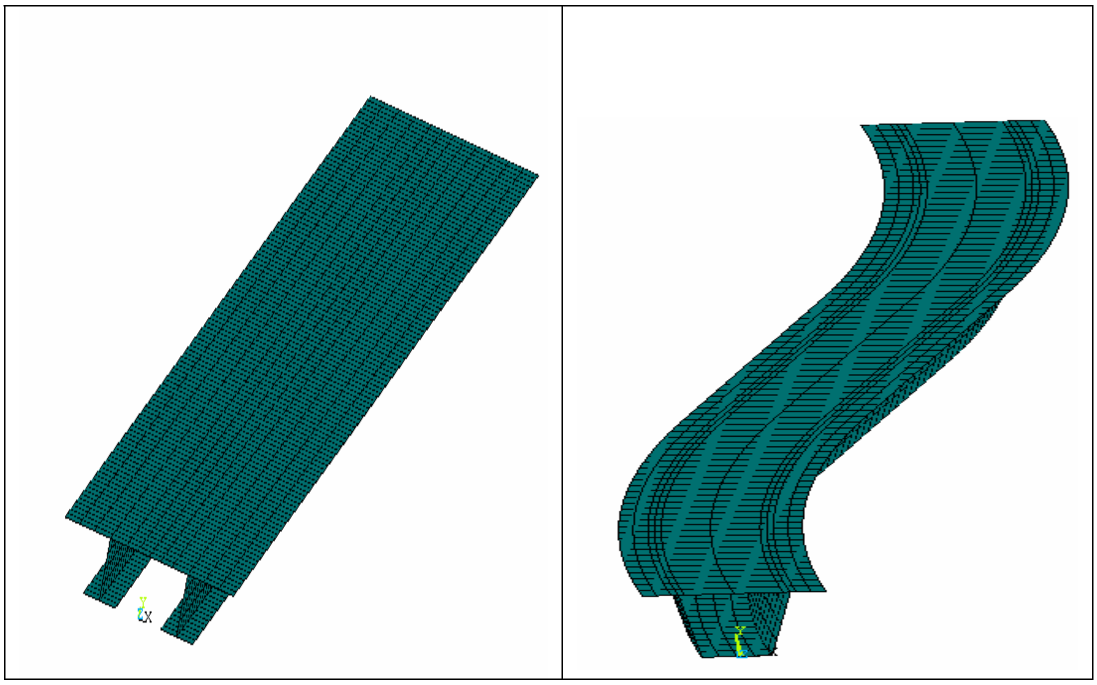

(a) (b)

Figure 3.2: Three dimensional views of (a) double I-girder, straight bridge and (b) variable radius of curvature, single box-girder bridge

3.2 Element Formulations

3.2.1 Shell Element Formulation

The program contains two types of shell elements: nine-node isoparametric shell element and a four-node isoparametric shell element (Fig.3.3). The formulation of these two elements are similar, the only difference is the shape functions.

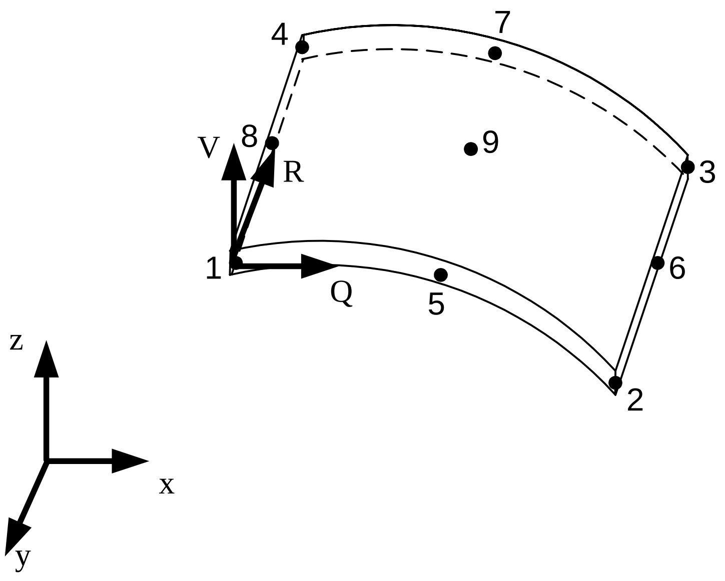


Figure 3.3: Nine-Node Shell Element

The interpolation functions for the 9-node (Fig. 3.4) element implemented into the program are as follows;

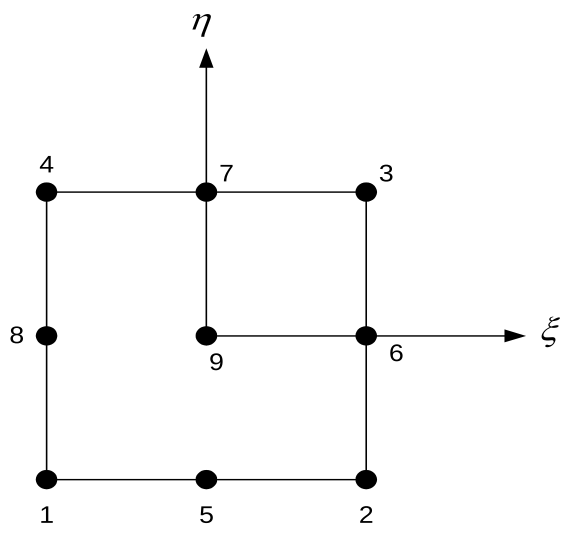


Figure 3.4: 9-node element

$$N_1 = \frac{1}{4}(\xi^2 - \xi)(\eta^2 - \eta)$$
$$N_2 = \frac{1}{4}(\xi^2 + \xi)(\eta^2 - \eta)$$
$$N_3 = \frac{1}{4}(\xi^2 + \xi)(\eta^2 + \eta)$$
$$N_4 = \frac{1}{4}(\xi^2 - \xi)(\eta^2 + \eta)$$
$$N_5 = \frac{1}{2}(1 - \xi^2)(\eta^2 - \eta)$$
$$N_6 = \frac{1}{2}(\xi^2 + \xi)(1 - \eta^2)$$
$$N_7 = \frac{1}{2}(1 - \xi^2)(\eta^2 + \eta)$$
$$N_8 = \frac{1}{2}(\xi^2 - \xi)(1 - \eta^2)$$
$$N_9 = (1 - \xi^2)(1 - \eta^2)$$

The interpolation functions for the 4-node element (Fig. 3.5) implemented into the program are as follows;

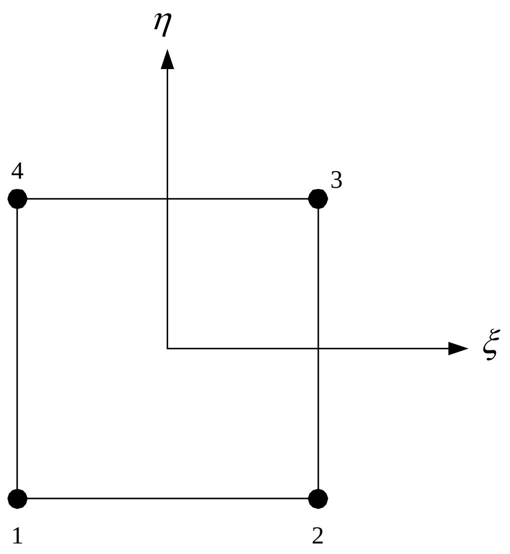


Figure 3.5: 4-node element

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$
$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$
$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$
$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$

The geometry of the element is defined as:

$$x(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(x_i + \zeta\frac{h}{2}V_{xi}\right)N_i\left(\xi,\eta\right)\right]$$
$$y(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(y_i + \zeta\frac{h}{2}V_{yi}\right)N_i\left(\xi,\eta\right)\right]$$
$$z(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(z_i + \zeta\frac{h}{2}V_{zi}\right)N_i\left(\xi,\eta\right)\right]$$

where ξ,η,ζ are the coordinate axes for the mapped element and ζ is in the thickness direction. **V**$\mathbf{V_i}$ ($\mathbf{V_{xi}}$, $\mathbf{V_{yi}}$, $\mathbf{V_{zi}}$) is the unit vector at each node, which is in the direction of the nodal fiber. “h” is the thickness of the shell. $N_i\left(\xi,\eta\right)$ is the shape function for node number i.

At each node there are three displacement (u, v, w) and two rotational (α,β) degrees of freedom. In order to define the rotation axes for α and β a right-handed triplet of mutually orthogonal unit vectors (**V**, **R**, **Q**) have to be chosen at each node.

The displacement components (u, v, w) are interpolated as follows:

$$u(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(u_i + \zeta\frac{h}{2}(-\alpha_i R_{xi} + \beta_i Q_{xi}\right)N_i\left(\xi,\eta\right)\right]$$
$$v(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(v_i + \zeta\frac{h}{2}(-\alpha_i R_{yi} + \beta_i Q_{yi}\right)N_i\left(\xi,\eta\right)\right]$$
$$w(\xi,\eta,\zeta) = \sum_{i=1}^{nne}\left[\left(w_i + \zeta\frac{h}{2}(-\alpha_i R_{zi} + \beta_i Q_{zi}\right)N_i\left(\xi,\eta\right)\right]$$

Derivatives of displacements are calculated as follows:

$$\frac{\partial}{\partial \xi} x(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(x_i + \zeta \frac{h}{2} V_{xi} \right) \frac{\partial}{\partial \xi} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \eta} x(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(x_i + \zeta \frac{h}{2} V_{xi} \right) \frac{\partial}{\partial \eta} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \zeta} x(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(\frac{h}{2} V_{xi} \right) N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \xi} y(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(y_i + \zeta \frac{h}{2} V_{yi} \right) \frac{\partial}{\partial \xi} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \eta} y(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(y_i + \zeta \frac{h}{2} V_{yi} \right) \frac{\partial}{\partial \eta} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \zeta} y(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(\frac{h}{2} V_{yi} \right) N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \xi} z(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(z_i + \zeta \frac{h}{2} V_{zi} \right) \frac{\partial}{\partial \xi} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \eta} z(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(z_i + \zeta \frac{h}{2} V_{zi} \right) \frac{\partial}{\partial \eta} N_i(\xi,\eta) \right]$$

$$\frac{\partial}{\partial \zeta} z(\xi,\eta,\zeta) = \sum_{i=1}^{nne} \left[\left(\frac{h}{2} V_{zi} \right) N_i(\xi,\eta) \right]$$

The Jacobian and its inverse are defined as follows:

$$J = \begin{pmatrix} \frac{\partial}{\partial \xi} x & \frac{\partial}{\partial \xi} y & \frac{\partial}{\partial \xi} z \\ \frac{\partial}{\partial \eta} x & \frac{\partial}{\partial \eta} y & \frac{\partial}{\partial \eta} z \\ \frac{\partial}{\partial \zeta} x & \frac{\partial}{\partial \zeta} y & \frac{\partial}{\partial \zeta} z \end{pmatrix} \qquad J^{-1} = \begin{pmatrix} \frac{\partial}{\partial x} \xi & \frac{\partial}{\partial x} \eta & \frac{\partial}{\partial x} \zeta \\ \frac{\partial}{\partial y} \xi & \frac{\partial}{\partial y} \eta & \frac{\partial}{\partial y} \zeta \\ \frac{\partial}{\partial z} \xi & \frac{\partial}{\partial z} \eta & \frac{\partial}{\partial z} \zeta \end{pmatrix}$$

Derivatives used in the B matrix formation are defined as:

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial x}$$

$$\frac{\partial N_i}{\partial y} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial y}$$

$$\frac{\partial N_i}{\partial z} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial z} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial z}$$

Derivatives of displacements functions with respect to coordinates (x,y,z) are as follows:

$$\frac{\partial u}{\partial x}=\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial x}\right]$$

$$\frac{\partial u}{\partial y}=\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial y}\right]$$

$$\frac{\partial u}{\partial z}=\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(u_i+\zeta\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{xi}+\beta_i Q_{xi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial z}\right]$$

$$\frac{\partial v}{\partial x}=\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial x}\right]$$

$$\frac{\partial v}{\partial y}=\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial y}\right]$$

$$\frac{\partial v}{\partial z}=\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(v_i+\zeta\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{yi}+\beta_i Q_{yi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial z}\right]$$

$$\frac{\partial w}{\partial x}=\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial x}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial x}\right]$$

$$\frac{\partial w}{\partial y}=\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial y}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial y}\right]$$

$$\frac{\partial w}{\partial z}=\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\xi}N_i(\xi,\eta)\frac{\partial\xi}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(w_i+\zeta\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)\frac{\partial}{\partial\eta}N_i(\xi,\eta)\frac{\partial\eta}{\partial z}\right]+\sum_{i=1}^{nne}\left[\left(\frac{h}{2}(-\alpha_i R_{zi}+\beta_i Q_{zi})\right)N_i(\xi,\eta)\frac{\partial\zeta}{\partial z}\right]$$

The B matrix is formed as follows:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \\ \frac{\partial w}{\partial x}+\frac{\partial u}{\partial z} \\ \frac{\partial w}{\partial y}+\frac{\partial v}{\partial z} \end{bmatrix} = \begin{pmatrix} \frac{\partial N}{\partial x} & 0 & 0 & -\frac{h}{2}R_{xi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right) & -\frac{h}{2}Q_{xi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right) & \cdots \\ 0 & \frac{\partial N}{\partial y} & 0 & -\frac{h}{2}R_{yi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right) & -\frac{h}{2}Q_{yi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right) & \cdots \\ 0 & 0 & \frac{\partial N}{\partial z} & -\frac{h}{2}R_{zi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & -\frac{h}{2}Q_{zi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & \cdots \\ \frac{\partial N}{\partial y} & \frac{\partial N}{\partial x} & 0 & -\frac{h}{2}R_{xi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right)-\frac{h}{2}R_{yi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right) & \frac{h}{2}Q_{xi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right)+\frac{h}{2}Q_{yi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right) & \cdots \\ \frac{\partial N}{\partial z} & 0 & \frac{\partial N}{\partial x} & -\frac{h}{2}R_{zi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right)-\frac{h}{2}R_{xi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & \frac{h}{2}Q_{zi}\left(\zeta\frac{\partial N}{\partial x}+N\frac{\partial \zeta}{\partial x}\right)+\frac{h}{2}Q_{xi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & \cdots \\ 0 & \frac{\partial N}{\partial z} & \frac{\partial N}{\partial y} & -\frac{h}{2}R_{zi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right)-\frac{h}{2}R_{yi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & \frac{h}{2}Q_{zi}\left(\zeta\frac{\partial N}{\partial y}+N\frac{\partial \zeta}{\partial y}\right)+\frac{h}{2}Q_{yi}\left(\zeta\frac{\partial N}{\partial z}+N\frac{\partial \zeta}{\partial z}\right) & \cdots \end{pmatrix} \begin{bmatrix} u \\ v \\ w \\ \alpha \\ \beta \\ \cdots \\ \cdots \end{bmatrix}$$

The rigidity matrix, D, must contain the shell assumption that, the stress component along the thickness direction is zero. As a result of this a rigidity matrix similar to the one used in two-dimensional plane stress analysis is obtained.

$$D^{local} = \frac{E}{1-\upsilon^2}\begin{pmatrix} 1 & \upsilon & 0 & 0 & 0 & 0 \\ \upsilon & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-\upsilon^2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-\upsilon^2}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-\upsilon^2}{2} \end{pmatrix}$$

As the stiffness terms are calculated in the global coordinates, the rigidity matrix should be transformed from local to global coordinates at each integration point. This is done by multiplying the rigidity matrix with the rotation matrix, $\underline{ROT}$. In order to from the rotation matrix, a local orthogonal coordinate axes consisting of unit vectors t_1, t_2, t_3 should be formed where t_3 is the vector normal to the shell surface at the point of consideration. The orthogonal local axes are formed as follows:

At the point of consideration:

$$\vec{t_1} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \end{pmatrix}^T \qquad \vec{t_2} = \begin{pmatrix} \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{pmatrix}^T$$

Form the unit vectors:

$$\vec{t_1} = \frac{\vec{t_1}}{\left|\vec{t_1}\right|} \qquad \vec{t_2} = \frac{\vec{t_2}}{\left|\vec{t_2}\right|}$$

Calculate the normal vector t_3:

$$\vec{t_3} = \vec{t_1} \times \vec{t_2}$$

Re-orient the t$_2$ vector:

$$\vec{t_2} = \vec{t_3} \times \vec{t_1}$$

The rotation matrix that transforms stress-strain laws is:

$$\underline{ROT} = \begin{pmatrix} l_1^2 & m_1^2 & n_1^2 & l_1m_1 & m_1n_1 & n_1l_1 \\ l_2^2 & m_2^2 & n_2^2 & l_2m_2 & m_2n_2 & n_2l_2 \\ l_3^2 & m_3^2 & n_3^2 & l_3m_3 & m_3n_3 & n_3l_3 \\ 2l_1l_2 & 2m_1m_2 & 2n_1n_2 & l_1m_2 + l_2m_1 & m_1n_2 + m_2n_1 & n_1l_2 + n_2l_1 \\ 2l_2l_3 & 2m_2m_3 & 2n_2n_3 & l_2m_3 + l_3m_2 & m_2n_3 + m_3n_2 & n_2l_3 + n_3l_2 \\ 2l_3l_1 & 2m_3m_1 & 2n_3n_1 & l_3m_1 + l_1m_3 & m_3n_1 + m_1n_3 & n_3l_1 + n_1l_3 \end{pmatrix}$$

Where the entries are the directional cosines of vectors t$_1$, t$_2$, t$_3$.

$l_1 = t_{1x}$

$l_2 = t_{2x}$

$l_3 = t_{3x}$

$m_1 = t_{1y}$

$m_2 = t_{2y}$

$m_3 = t_{3y}$

$n_1 = t_{1z}$

$n_2 = t_{2z}$

$n_3 = t_{3z}$

The global rigidity matrix, D, is calculated as follows:

$$D = \underline{ROT}^T D^{local} \underline{ROT}$$

The stiffness matrix is:

$$K = \sum_{ip=1}^{nip} \left\{ B^T D B \det(J) w(ip) \right\}$$

The implementation uses regular integration that is 3 integration points in ξ, η directions and 2 integration points in ζ direction for the 9-node element and 2 integration points in ξ, η directions and 2 integration points in ζ for the 4-node element.

3.2.2 Truss Element Formulation

A standard 3 dimensional, 2-node linear truss element is implemented into the program. (Fig 3.6) The stiffness formulation is as follows:

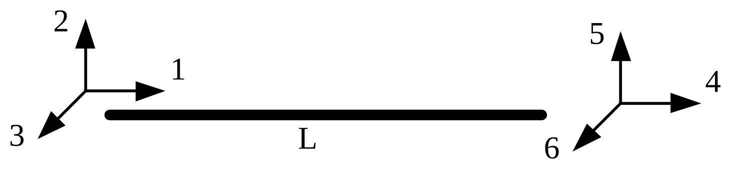


Figure 3.6: 3 dimensional, 2 node truss element

$$\begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Where;

E: Modulus of Elasticity, A: Area of Truss Member, and L: Length of Truss Member

3.2.3 Spring Element Formulation

A standard 2-node, three dimensional spring element is implemented into the program (Fig 3.7). The stiffness matrix formulation is as follows:

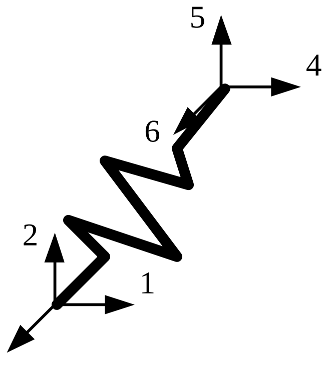


Figure 3.7: 3 dimensional, 2-node spring element

$$\begin{bmatrix} K_1 & 0 & 0 & -K_1 & 0 & 0 \\ 0 & K_2 & 0 & 0 & -K_2 & 0 \\ 0 & 0 & K_3 & 0 & 0 & -K_3 \\ -K_1 & 0 & 0 & K_1 & 0 & 0 \\ 0 & -K_2 & 0 & 0 & K_2 & 0 \\ 0 & 0 & -K_3 & 0 & 0 & K_3 \end{bmatrix}$$

Where;

K_1, K_2 and K_3 are stiffness values in three global directions.

3.3 Solver Basics

The main solver of the developed program is the direct solver which is a part of the Compaq Extended Math Library (CXML). An iterative solver called ITPACK is also adopted into the program to compare the direct and iterative solvers. As discussed previously the direct solver is supplied as a library file and can be compiled with the program.

Suppose we have a system of equations of the form:

$$Ax = B$$

where A is an n by n matrix and x and B are n element column vectors.

In this representation for our case A is the stiffness matrix and vectors x and B are displacement vectors and load vectors, respectively. We know that most of the elements of the A matrix are zero. Such matrices are named as sparse matrices. Generally speaking computer softwares that find solutions to linear equation systems are called solvers. A solver specifically designed to solve sparse systems is called a sparse solver. The sparse solvers can be direct or iterative. Iterative solvers start with an initial approximation of the solution vector and try to converge to the actual result as close as possible. On the other hand, in a direct solver, the matrix A is factored into upper and lower triangular matrices and a forward and backward triangular solution process is employed. The solution time required for direct solvers is more predictable compared with the iterative solvers but for some well-conditioned systems the iterative solvers may be more efficient.

3.3.1 Sparse Solver Storage Format

The direct solver adopted into the program requires only the nonzero entries of the structural stiffness matrix be stored. In addition to this, the solver requires two vectors that define the locations of the nonzero entries. Based on the information of the nonzero entries and their locations the solver is capable of reordering and factoring the stiffness matrix and solving for the displacements. Only the upper or lower triangular half of the matrix is stored.

The non-zero entries are stored in a vector called “nonzeros”, the column and row numbers of these non-zero entries are also stored in two matrices called “icolumns” and “irowindex”, respectively. When the elements of the matrix “nonzeros” are filled, the upper triangular half of the structural stiffness matrix is considered and the nonzero elements are stored starting with the upper first row excluding the zero elements. The “icolumns” vector is of the same size as the vector “nonzeros”. The entries of the “icolumns” vector give the column numbers of the corresponding entries of the vector “nonzeros”. The “irowindex” vector gives the location of the first non-zero entry within each row.

Since the “irowindex” vector gives the location of the first non-zero within a row, and the non-zeros are stored consecutively, then we are able to compute the number of non-zeros in the i-th row as the difference of the row indexes of the consecutive rows give the desired value.

In order have this relationship hold for the last row of A, we need to add an entry (dummy entry) to the end of “irowindex” array whose value is equal to the number of non-zeros in A, plus one. This makes the total length of the “irowindex” array one larger than the number of rows of A.

The above discussion can be more clarified with the following example. Suppose we have symmetrical matrix A;

$$A = \begin{bmatrix} 1 & 4 & 9 & 8 & 7 \\ * & 2 & 0 & 0 & 0 \\ * & * & 5 & 0 & 0 \\ * & * & * & 8 & 0 \\ * & * & * & * & 3 \end{bmatrix}$$

The “nonzeros” vector will be;

$$nonzeros = \begin{bmatrix} 1 & 4 & 9 & 8 & 7 & 2 & 5 & 8 & 3 \end{bmatrix}$$

The “icolumns” vector will be;

$$icolumns = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 3 & 4 & 5 \end{bmatrix}$$

and the “irowindex” vector will be;

$$irowindex = \begin{bmatrix} 1 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

The storage schemes for both direct and sparse solvers adopted into the program are same.

The displacement vector “uv” and the right hand side vector “rhs” in the program are vectors of size, the number of nodes times the number of degrees of freedom per node.

3.4 Program Verification with Existing Solutions

3.4.1 General

The validity of the computational software was tested both with approximate hand methods and with published solutions in the literature.

3.4.2 Hand Calculations

While the computational software is compared with the hand methods, a horizontal, single I-girder is modeled as a simply supported beam and the construction loading is applied. The deck concrete is assumed to be just poured and the effects of stud stiffnesses are ignored. The single I-girder bridge is solved both with program developed and the simple beam bending formula. The analyzed bridge cross section is shown in Figure 3.8.

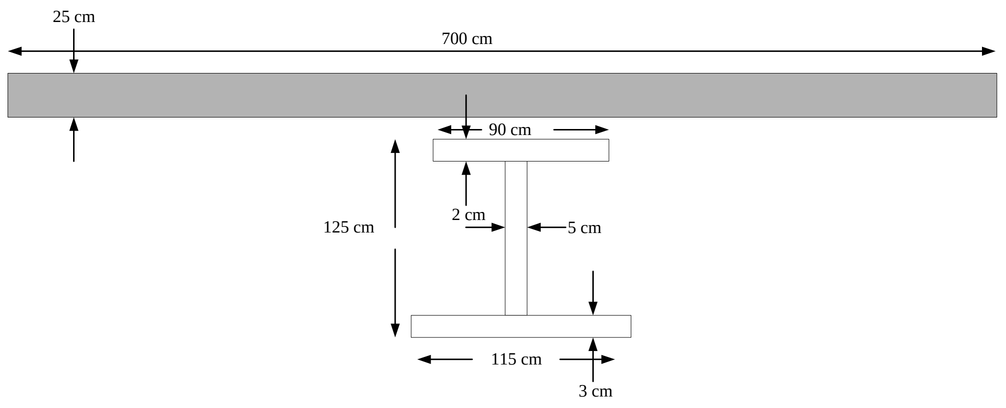


Figure 3.8: The Cross-section of the Analyzed Bridge

The element size is chosen as 100 cm and the modulus of elasticity of steel is 20 000 000 N/cm^2. The bridge is 6000 cm long and assumed to be straight. There are no internal, external or top lateral brace members. The concrete modulus and stud stiffnesses are taken to be very close to zero and the distributed loading on the bridge is 10 N/cm. The bridge is simply supported with the supports at 0 and 6000 cm.

Maximum bending moment, maximum shear force, maximum normal stress, maximum shear stress and mid-span deflection values obtained by using both the program and simple beam bending formula are given in Table 3.1

Table 3.1: Comparison of the Program Outputs and Hand Methods

Parameter	UTRAP	Hand Calculation
Maximum Bending Moment(M_{max}) (kN.m)	450	450
Maximum Shear Force(Vmax) (kN)	30	30
Maximum Normal Stress(σ_{max}) (N/cm^2)	1245.87	1229.72
Maximum Shear Stress(τ_{max}) (N/cm^2)	55.12	56.63
Maximum Deflection(δ_{max}) (cm)	3.29	3.24

As it can be inferred from Table 3.1, the program gives sufficiently close outputs as the simple bending formula. The slight difference between the program outputs and the hand calculations can be attributed to the element size selected and the node normals.

3.4.3 Published Solutions

The computational software results were compared with the hand method developed by the researchers Fan and Helwig (1999) in order to predict the top lateral brace member forces in curved box girders. The method proposed by Fan and Helwig was compared with the commercially available finite element analysis package, ANSYS. The predictions of the hand method were in excellent agreement with the finite element analysis. In this section, the published finite element analysis results are compared with the results of the developed software. The bridge analyzed by Fan and Helwig (1999) was a three-span single girder system having a radius of 291 m and .a length of 195.06 m. The details of the bridge are given in Fig. 3.9.

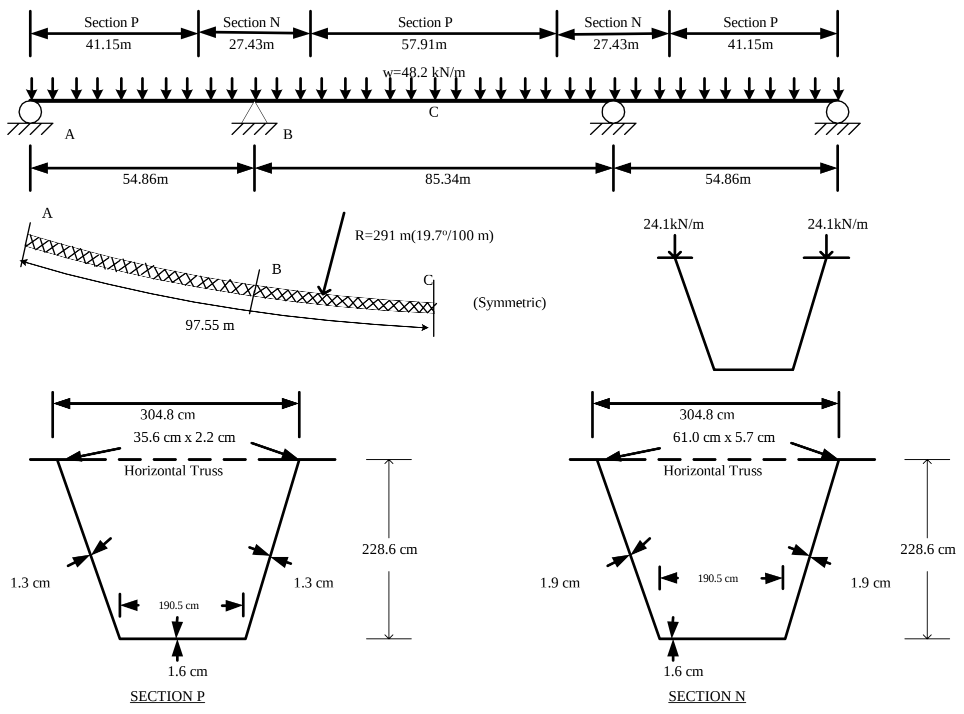


Figure 3.9: Layout and Cross-sectional Dimensions of the Bridge (Fan(1999))

Internal braces were located at every 3 m, and an X-type top lateral system between internal brace points was utilized. The top lateral brace members were WT 6 × 13 sections, while the internal brace elements were L 4×4× 5/16 sections. The distributed loading on the bridge was 48.2 kN/m. A constant top flange width of 35.6 cm was assumed. The thickness of top flange plates in Section N was modified to 9.7 cm to give the same plate area.

The top lateral members were grouped into (X1 and X2) according to their orientation. Force levels for these top lateral members obtained from finite element analysis were presented by Fan and Helwig (1999). These force levels are compared with the predictions from the developed software in Figures 3.10 and 3.11. It could be concluded that the developed software is capable of producing results similar to the published solutions.

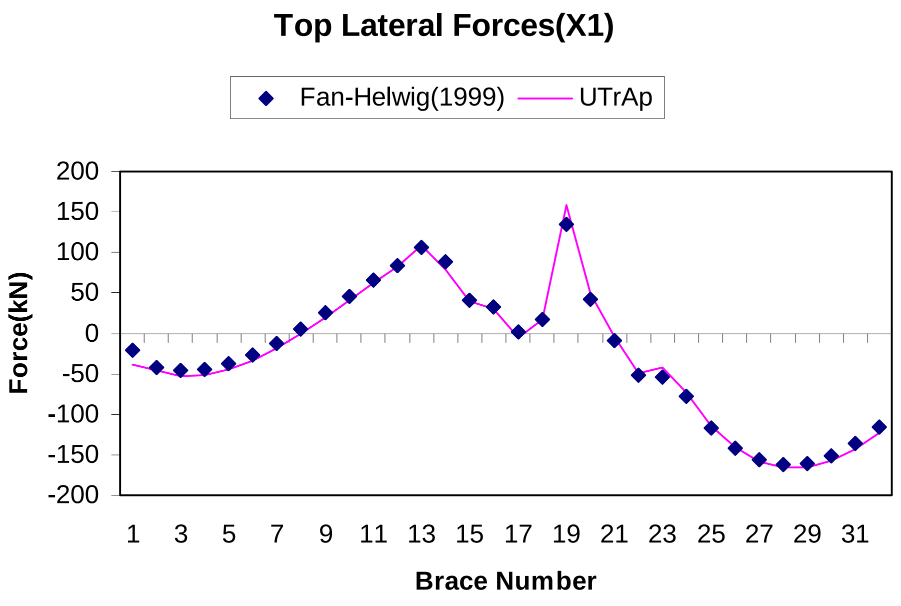


Figure 3.10: Comparison of Published and UTRAP Results for X1 Diagonals

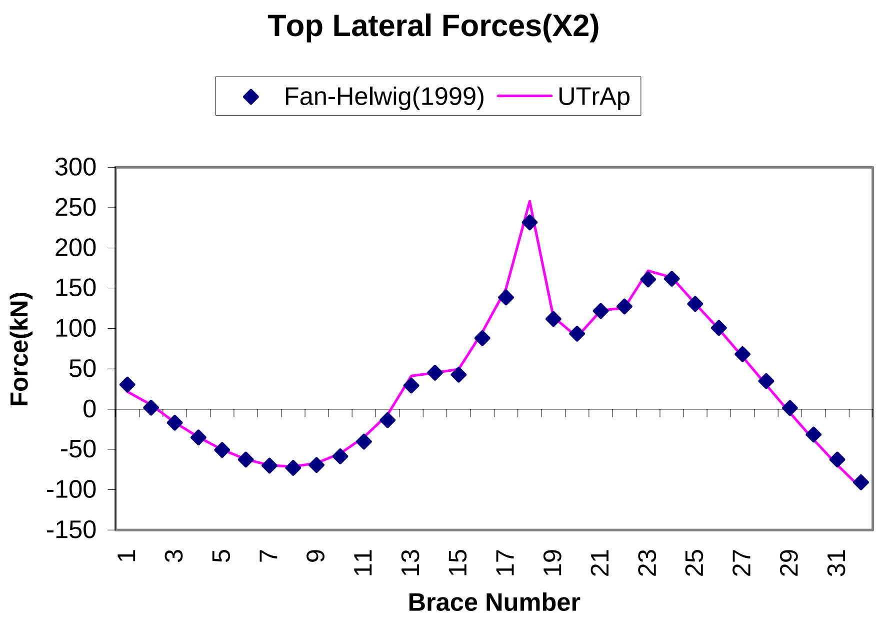


Figure 3.11: Comparison of Published and UTRAP Results for X2 Diagonals

CHAPTER 4

MODELING RECOMMENDATIONS FOR COMPOSITE BRIDGE ANALYSIS

4.1 Simply Supported Bridge Case

In order to report some design and modeling recommendations, mesh convergence studies are conducted by analyzing three bridges, first of which is a simply supported bridge. The configuration of the personal computer used for the analysis includes an Intel Centrino Mobile 1.4 GHz Processor and a 512 MB DDR RAM. The simply supported bridge analyzed is a 60.0 m long box-girder bridge consisting of 19 internal braces each 3.0 m apart. There is one top lateral brace for each panel, which make a total of 20 top lateral braces. The uniform loading on the bridge is 19kN/m and the elastic modulus of the steel is 200 000 MPa. The cross-sectional dimensions are given in Figure 4.1.

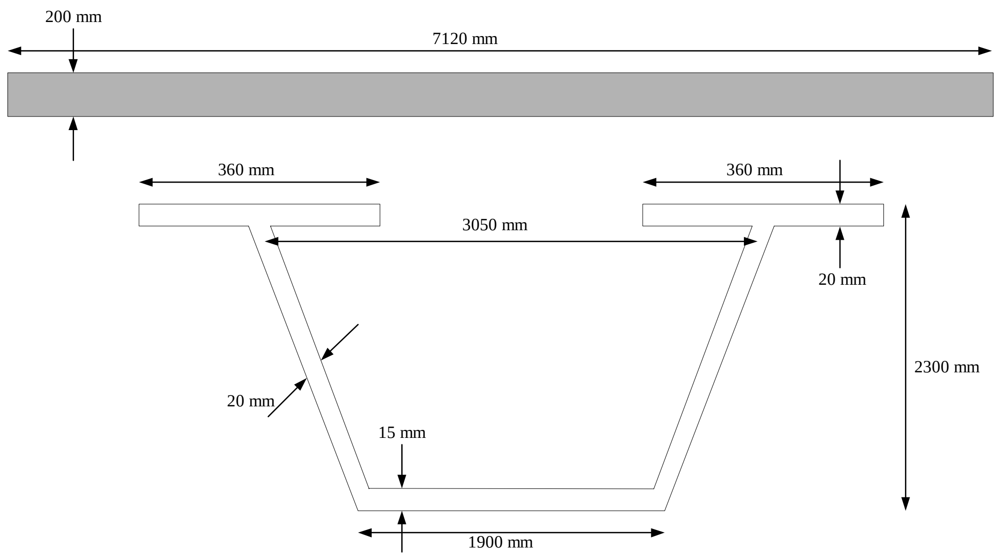


Figure 4.1: Cross-sectional Dimensions

The concrete modulus and stud stiffnesses are taken as zero in order to simulate the girder behavior at early ages of concrete.

The simply supported bridge is analyzed for five different radius of curvatures. The radius of curvatures for these bridges are defined as the subtended angle the bridge covers. The subtended angle is the angle in degrees that is enclosed by 100 m of representative bridge length. The bridges are analyzed for straight, 5^o/100 m, 10^o/100 m, 20^o/100 m, 30^o/100 subtended angles (Fig. 4.2).

The element sizes are taken as 60 cm, 75 cm, 100 cm, 150 cm and 300 cm, which are panel length/5, panel length/4, panel length/3, panel length/2 and panel length respectively. Both of the element types implemented into the program, 4-node and 9-node elements, are used in the analysis (Table 4.1).

Table 4.1 Analysis Parameters

Subtended Angle(degrees/100 m)	0,5,10,20,30
Element Size(cm)	60,75,100,150,300
Element Type	4-node, 9-node

As it can be inferred from Table 4.1 a total of 50 runs are performed for the simply supported bridge.

While studying the performance of the element types and element sizes, the required time for solution and required memory are also recorded using CXML direct solver and are given in Table 4.2.

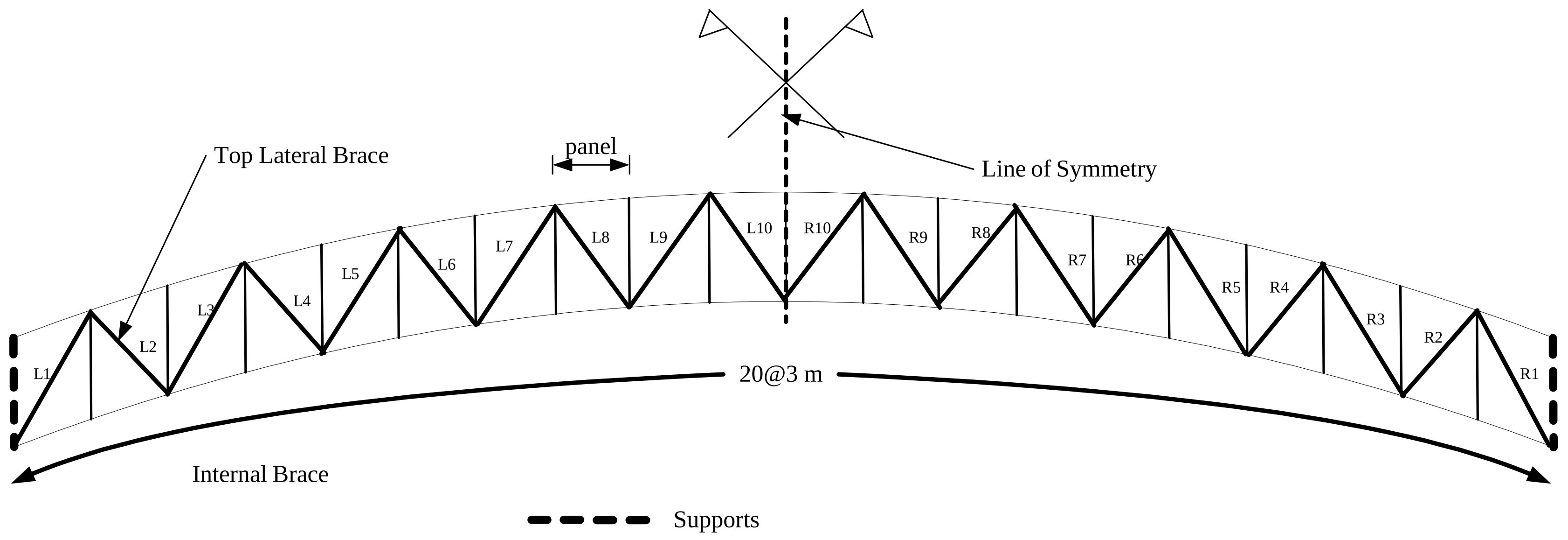


Figure 4.2: Analyzed Bridge

Table 4.2: Solution Time and Required Physical Memory

Subtended angle/100 m	Element type	Element Size(cm)	Solution Time(sec)	Required Physical Memory(MB)
0^o	4-node	60	5.9	24
		75	4.8	19
		100	3.6	14
		150	2.3	10
		300	1.2	2
	9-node	60	41.0	120
		75	32.4	95
		100	23.9	69
		150	15.7	45
		300	7.8	22

In Table 4.2 the solution time and the required physical memory for only 0^o/100m subtended angle are shown. As the number of elements and nodes do not depend on the degree of curvature these values are same for different subtended angles as long as the element sizes are the same.

As it can be seen on Table 4.2, when the 4-node element is used, as expected, the solution time and required memory are less when compared with the 9-node element. In Figures 4.3 and 4.4 the variation of Solution Time and Required Physical Memory with respect to the element size for 0^o subtended angle and 9-node element are shown respectively.

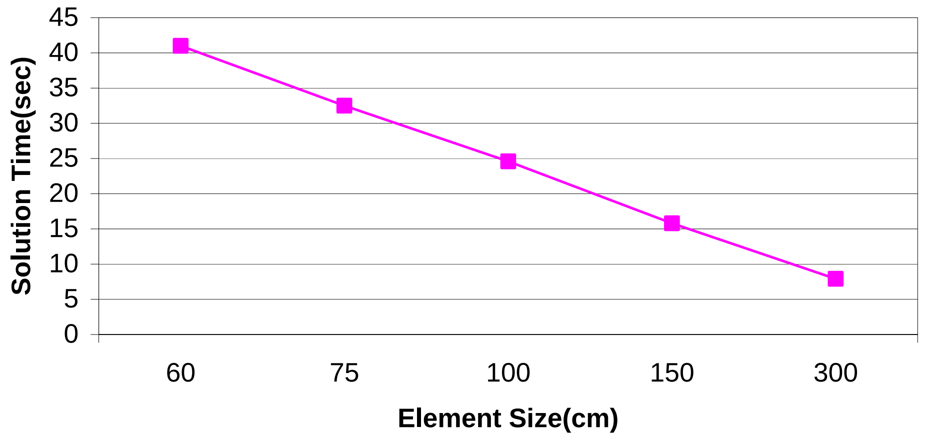


Figure 4.3: Variation of Solution Time with respect to the Element Size

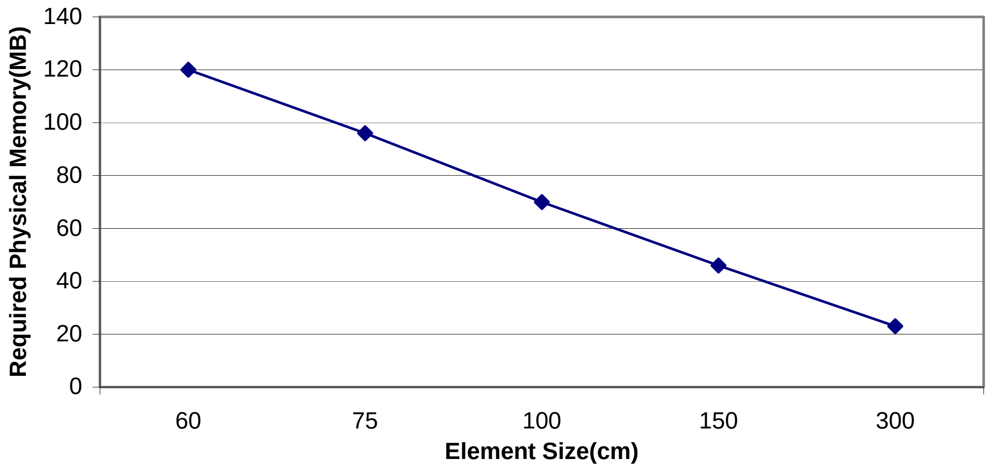


Figure 4.4: Variation of Required Physical Memory with respect to the Element Size

As it can be seen from Figure 4.3 and Figure 4.4, both the solution time and required physical memory for program execution follow a similar trend when plotted against the element size. Furthermore it can be concluded that when the element size is doubled both the memory requirement and solution time are halved.

4.1.1 Deflections

The maximum deflection for the 0^o/100 m subtended angle for 4-node and 9-node elements with respect to the element size used is given in Fig. 4.5 and Fig. 4.6, respectively. A representative deflection profile along the bridge is also given in Figure 4.7, for 30^o/100 m subtended angle, 9-node element and 60 cm element size.

As it can be inferred from the two figures, the degree of accuracy of the results obtained with the 9-node element is far more realistic than the 4-node element. For the 9-node element as the element size increases, the obtained results for the maximum deflection does not differ much, even for the 150 cm and 300 cm element sizes the obtained results are 18.12 cm and 18.05 cm respectively, even though the element size doubles. On the other hand for the 4-node element as the element size increases the maximum deflection value decreases and the difference between the maximum deflection values obtained with the 150 cm element size and 300 cm element size differs about 23%. Therefore it can be concluded that 9-node element is a more reliable element than the 4-node element. However considering advantages of solution time and required memory, 4 node element is still feasible when the element size is taken as 150 cm. Smaller elements sizes for the 4-node elements may also be considered but there is a modeling constraint associated with them which will be discussed in Section 4.1.1.1.

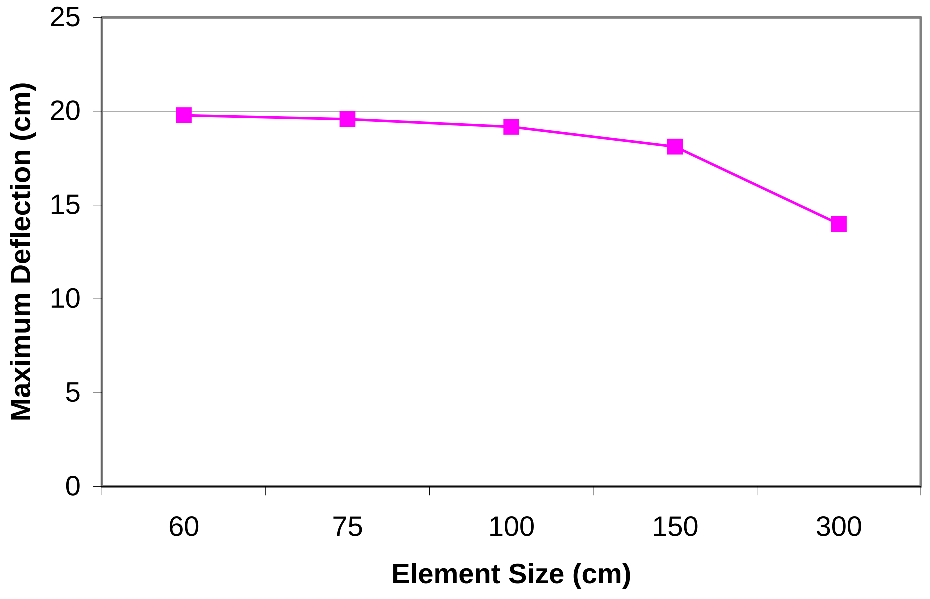


Figure 4.5: Maximum Deflections for Various Element Sizes for 4-node element

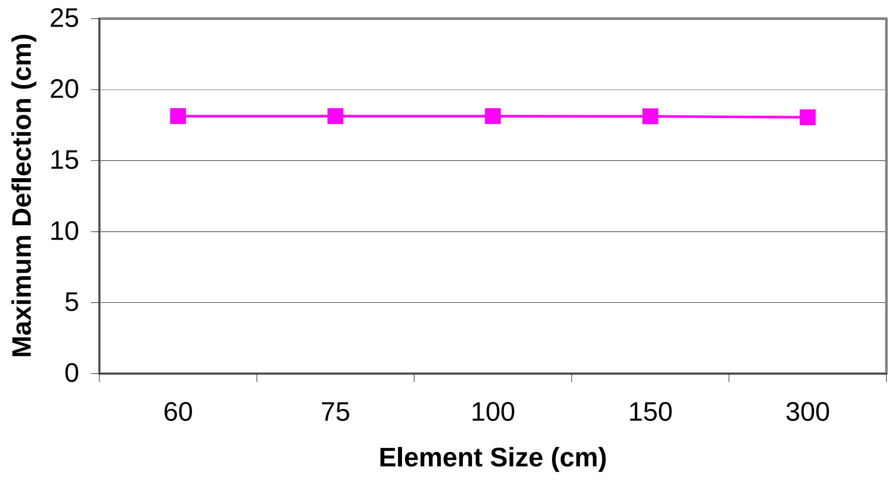


Figure 4.6: Maximum Deflections for Various Element Sizes for 9-node element

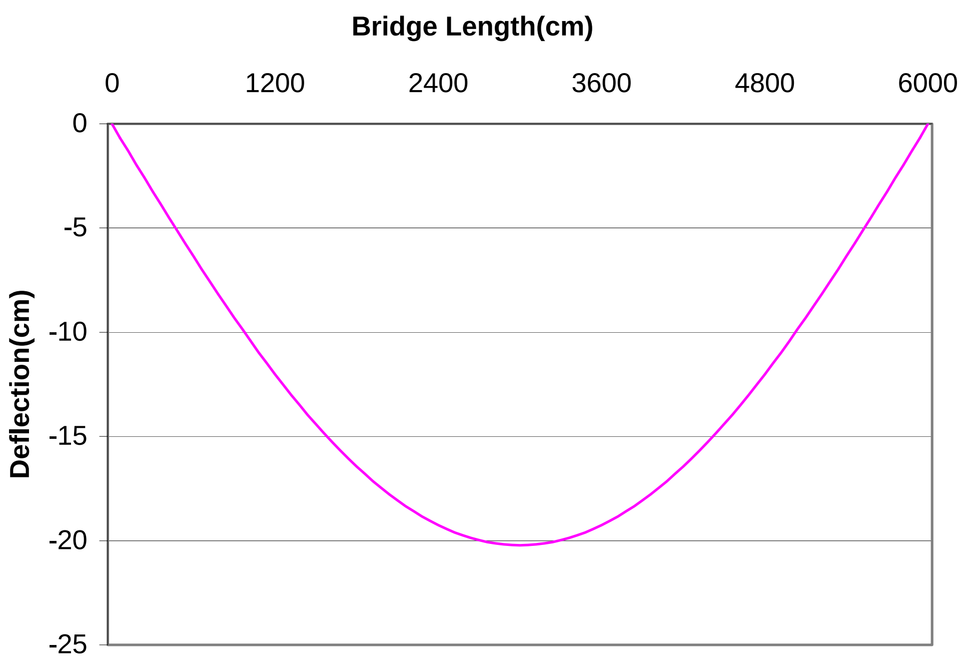


Figure 4.7: Deflection Profile Along the Bridge

In Figure 4.7 the symmetrical deflection profile agrees with the inputs.

The variation of maximum deflection with respect to the radius of curvature and element size for the 9-node and 4-node elements are given in Table 4.5 and Table 4.6, respectively.

Table 4.5: Variation of Maximum Deflection with Subtended Angle(9-node element)

Maximum Deflection(cm)					
Element Size (cm)	**Subtended Angle: 0^o/100m**	**Subtended Angle: 5^o/100m**	**Subtended Angle: 10^o/100m**	**Subtended Angle: 20^o/100m**	**Subtended Angle: 30^o/100m**
60	18.13	18.19	18.36	19.05	20.21
75	18.13	18.19	18.36	19.05	20.21
100	18.13	18.18	18.35	19.04	20.2

Table 4.5: Variation of Maximum Deflection with Subtended Angle (9-node element)(Continued)

Maximum Deflection(cm)					
Element Size (cm)	Element Size (cm)	Element Size (cm)	Element Size (cm)	Element Size (cm)	Element Size (cm)
150	18.12	18.17	18.34	19.03	20.18
300	18.05	18.1	18.27	18.95	20.09

Table 4.6: Variation of Maximum Deflection with Subtended Angle (4-node element)

Maximum Deflection(cm)					
Element Size (cm)	Subtended Angle: 0^o/100m	Subtended Angle: 5^o/100m	Subtended Angle: 10^o/100m	Subtended Angle: 20^o/100m	Subtended Angle: 30^o/100m
60	19.78	19.83	20.00	20.65	21.76
75	19.58	19.63	19.79	20.45	21.55
100	19.17	19.22	19.38	20.03	21.13
150	18.11	18.16	18.32	18.95	20.03
300	13.99	14.04	14.18	14.77	15.76

From Tables 4.5 and 4.6 it can be inferred that even the 300 cm element size when 9-node elements are used gives good results in terms of maximum deflection. Also when 4-node element is used, quite useful results are obtained considering the time and memory advantages the element offers.

4.1.1.1 Node Normals

When the maximum deflection values for 4-node and 9-node elements are compared it is seen that the 4-node element gives higher values than the 9-node element (Figs 4.5, 4.6). This unexpected flexible behavior can be attributed to node normals.

As discussed in Chapter 2, the shell geometry is defined by the node normals Q, R and V which are defined at each node. Among these unit vectors V always points in the direction of the thickness of the shell element and R always points in the direction of the tangent to the arc length. Q is the unit vector which is orthogonal to the other two vectors.

Consider a trapezoidal girder in Figure 4.8. The node normals at the top flange-web interface are oriented as given in the figure.

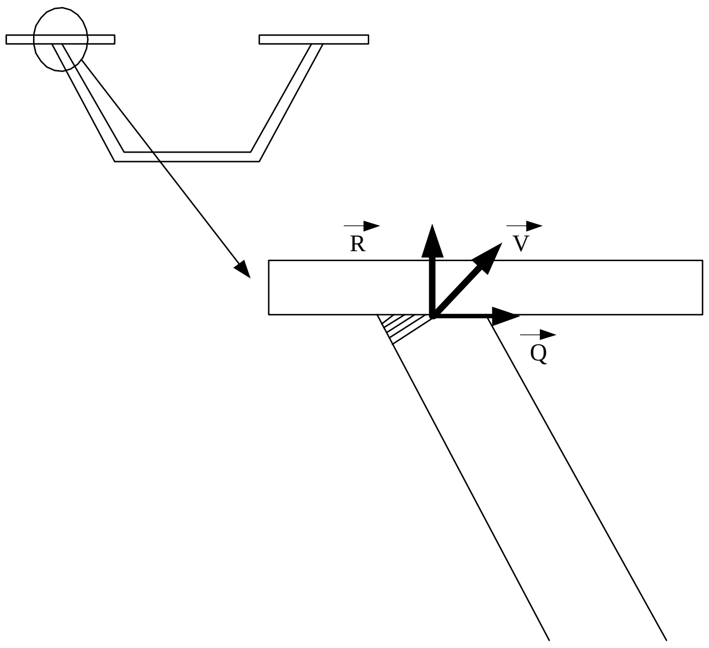


Figure 4.8:Node Normals at the Interphase

As it can be seen in the figure, when the node normals are used to define the geometry, at the connections, as in the case of Figure 4.8, the shaded area is discarded. This results in getting higher deflection values as the member sizes decrease. The cumulative effect of this shortcoming is more dominant when 4-node elements are used because their meshes are coarser.

4.1.2 Top Lateral Brace Forces

In Table 4.6, maximum top lateral brace forces for different subtended angles, element types and element sizes are shown. The given values are the absolute values, they can be either tensile or compressive.

When the maximum top lateral brace forces for 0^o/100 m subtended angle are examined it is seen that the obtained results for the 4-node element are even two fold of the 9-node counterparts for most of the cases, however when the subtended angle increases this difference become negligible and the 4-node element becomes feasible. In Figures 4.9 and 4.10 this behavior is shown.

Table 4.6: Maximum Top Lateral Forces

Subtended angle/100 m	Element type	Element Size(cm)	Maximum Top Lateral Force(kN)
0^o	4-node	60	60
		75	65
		100	71
		150	75
		300	65
	9-node	60	32
		75	32
		100	33
		150	34
		300	65
5^o	4-node	60	82
		75	86
		100	91
		150	94
		300	85
	9-node	60	61
		75	61
		100	61
		150	63
		300	86
10^o	4-node	60	127
		75	130
		100	133
		150	135
		300	127
	9-node	60	112
		75	112
		100	112
		150	113
		300	130

Table 4.6:Maximum Top Lateral Forces(Continued)

Subtended angle/100 m	Element type	Element Size(cm)	Maximum Top Lateral Force(kN)
20^o	4-node	60	232
		75	233
		100	234
		150	234
		300	224
	9-node	60	226
		75	226
		100	226
		150	227
		300	234
30^o	4-node	60	343
		75	344
		100	344
		150	343
		300	330
	9-node	60	337
		75	337
		100	337
		150	338
		300	345

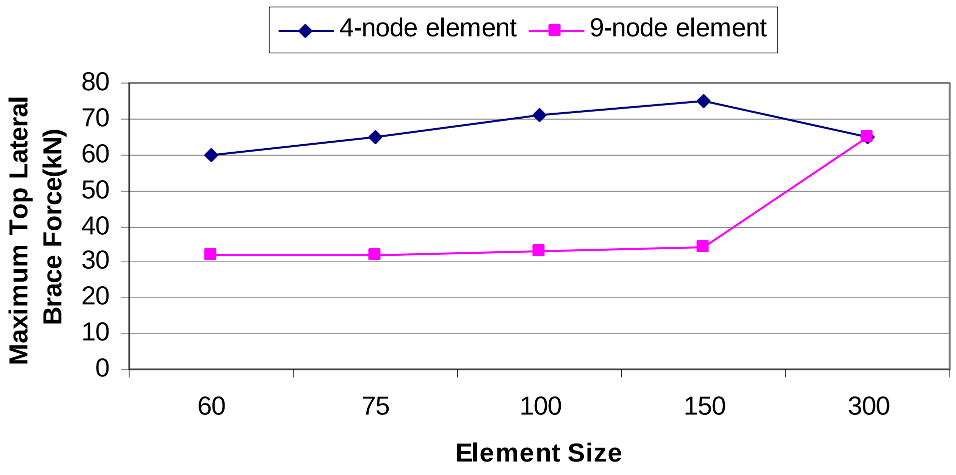


Figure 4.9: Maximum Top Lateral Brace Forces for 0^o/100 m subtended Angle

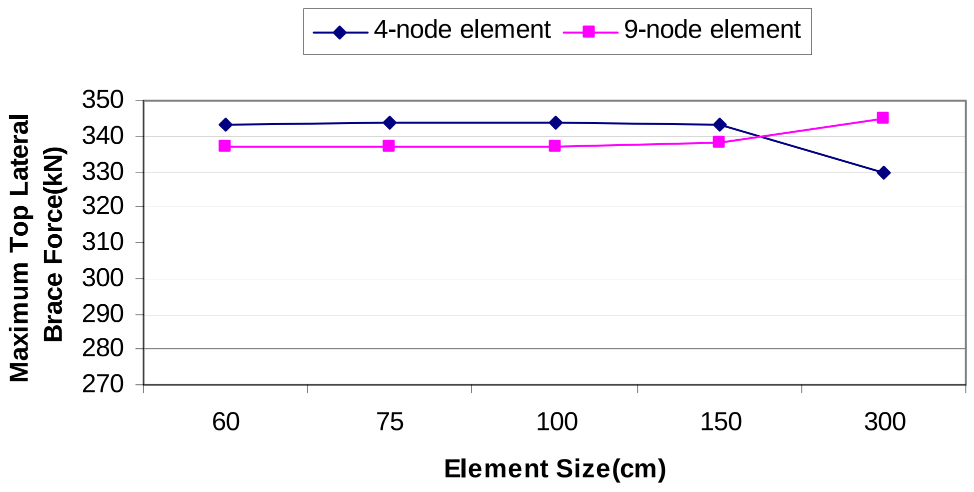


Figure 4.10: Maximum Top Lateral Brace Forces for 30°/100 m subtended Angle

As it can be inferred from Figures 4.9 and 4.10 which show the maximum top lateral brace forces for 0°/100 m and 30°/100 m subtended angles with respect to the element size, even the 4-node elements are reliable when the degree of curvature is high.

In Tables 4.7 and 4.8 the top lateral brace forces for 0° subtended angle and 30° subtended angle are given.

Table 4:7 Top Lateral Brace Forces for 0^o/100 m Subtended Angle

TOP LATERAL BRACE	SUBTENDED ANGLE = 0^o/100 m									
	ELEMENT TYPE = 4 NODE ELEMENT					ELEMENT TYPE = 9 NODE ELEMENT				
	ES=60 cm	ES=75 cm	ES=100 cm	ES=150 cm	ES=300 cm	ES=60 cm	ES=75 cm	ES=100 cm	ES=150 cm	ES=300 cm
L1	-5	-5	-6	-6	-5	-3	-3	-3	-3	-6
L2	-15	-16	-18	-19	-15	-8	-8	-8	-9	-16
L3	-25	-27	-30	-32	-27	-14	-14	-14	-15	-27
L4	-34	-37	-40	-42	-36	-18	-18	-19	-19	-36
L5	-41	-45	-49	-52	-44	-22	-22	-23	-24	-44
L6	-47	-51	-56	-59	-51	-26	-26	-26	-27	-51
L7	-52	-57	-62	-66	-57	-28	-28	-29	-30	-56
L8	-56	-61	-66	-70	-61	-30	-30	-31	-32	-61
L9	-58	-63	-69	-73	-63	-32	-32	-32	-34	-63
L10	-60	-65	-71	-75	-65	-32	-32	-33	-34	-65
R10	-60	-65	-71	-75	-65	-32	-32	-33	-34	-65
R9	-58	-63	-69	-73	-63	-32	-32	-32	-34	-63
R8	-56	-61	-66	-70	-61	-30	-30	-31	-32	-61
R7	-52	-57	-62	-66	-57	-28	-28	-29	-30	-56
R6	-47	-51	-56	-59	-51	-26	-26	-26	-27	-51
R5	-41	-45	-49	-52	-44	-22	-22	-23	-24	-44
R4	-34	-37	-40	-42	-36	-18	-18	-19	-19	-36
R3	-25	-27	-30	-32	-27	-14	-14	-14	-15	-27
R2	-15	-16	-18	-19	-15	-8	-8	-8	-9	-16
R1	-5	-5	-6	-6	-4	-3	-3	-3	-3	-6

ES = Element Size

Table 4:8 Top Lateral Brace Forces for 30^o/100 m Subtended Angle

TOP LATERAL BRACE	SUBTENDED ANGLE = 0^o/100 m									
	ELEMENT TYPE = 4 NODE ELEMENT					ELEMENT TYPE = 9 NODE ELEMENT				
	ES=60 cm	ES=75 cm	ES=100 cm	ES=150 cm	ES=300 cm	ES=60 cm	ES=75 cm	ES=100 cm	ES=150 cm	ES=300 cm
L1	329	328	327	324	315	334	333	333	333	329
L2	-343	-344	-344	-343	-330	-337	-337	-337	-338	-345
L3	284	282	279	275	271	296	296	296	295	283
L4	-317	-320	-323	-324	-310	-303	-303	-303	-304	-321
L5	209	205	201	197	198	229	229	228	227	206
L6	-260	-264	-269	-271	-257	-240	-240	-240	-241	-265
L7	117	113	107	103	108	142	142	141	140	113
L8	-181	-185	-191	-194	-181	-156	-156	-156	-158	-186
L9	16	11	5	1	10	43	43	43	41	11
L10	-86	-91	-97	-101	-89	-59	-59	-59	-61	-91
R10	-86	-91	-97	-101	-89	-58	-58	-59	-61	-91
R9	16	11	5	1	10	43	43	43	41	11
R8	-181	-185	-191	-194	-181	-155	-155	-156	-157	-186
R7	117	113	107	103	108	142	141	141	140	113
R6	-260	-264	-268	-271	-257	-239	-239	-240	-241	-265
R5	209	205	201	196	198	228	228	228	227	206
R4	-317	-320	-323	-324	-310	-303	-303	-303	-304	-321
R3	284	282	279	275	271	296	296	296	295	283
R2	-343	-344	-344	-343	-330	-337	-337	-337	-338	-345
R1	329	328	327	324	315	333	333	332	332	329

ES = Element Size

In Tables 4.7 and 4.8, it is seen that when the radius of curvature is small, as in the case of Figure 4.7 in which the analyzed bridge is straight, the 300 cm element gives much higher values than the 150 cm element for the 9-node element so in terms of top lateral brace forces the 300 cm element is not efficient. Smaller element sizes should be selected, however, the 150 cm element size seems efficient as the difference between the top lateral brace forces obtained by using smaller element sizes is far less than 2% for most of the members. In Figure 4.11 the top lateral brace forces for 0^o subtended angle obtained by using 9-node element and with 150 and 300 cm element sizes are plotted. As a result of these, it may be recommended that element size should be at most half the panel size.

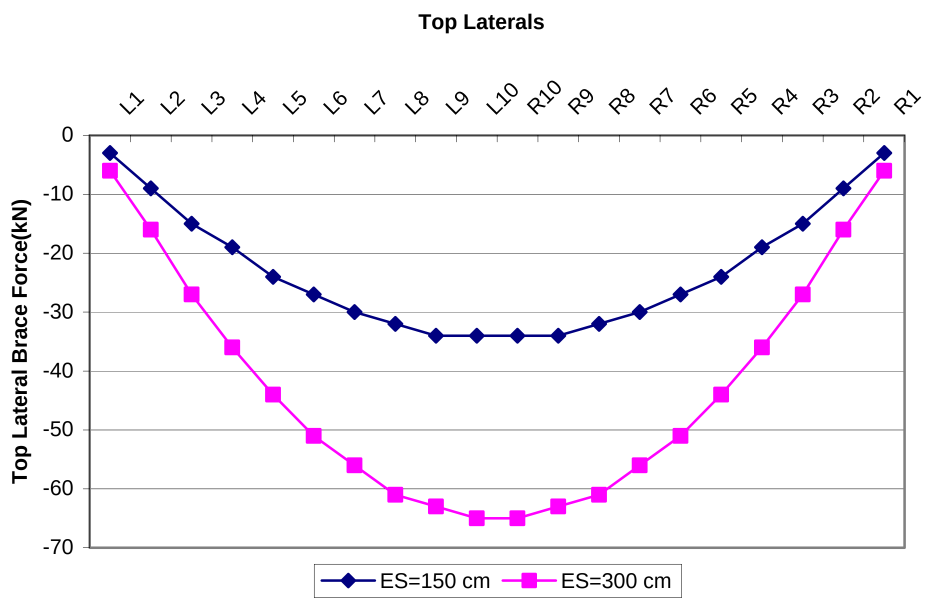


Figure 4.11:Top Lateral Brace Forces for 150 and 300 cm Element Sizes (ES = Element Size)

When the subtended angle increases, as torsional effects come into the scene, the 300 cm element size performs better as shown in Figure 4.12.

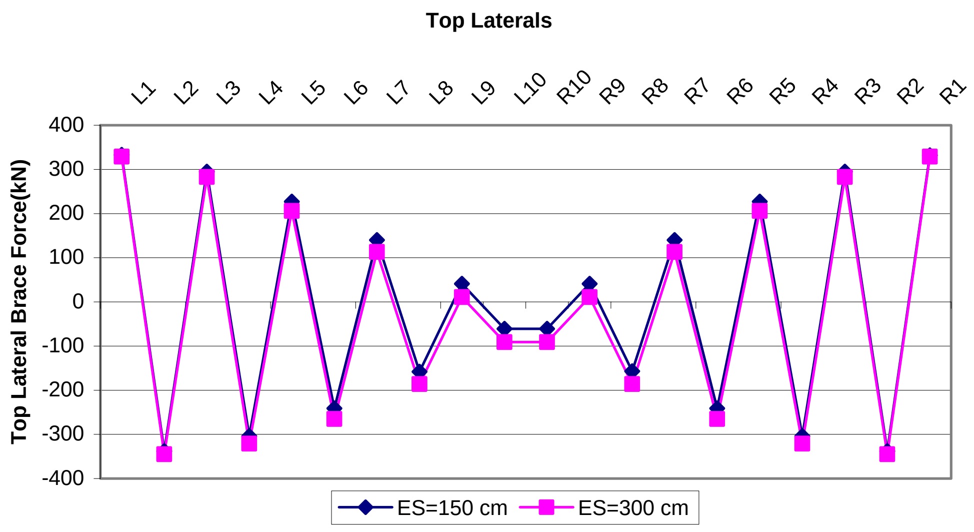


Figure 4.12: Top Lateral Brace Forces for 150 and 300 cm Element Sizes (ES = Element Size)

In Figure 4.12 it is seen that except from some mid-elements the results obtained with 150 cm and 300 cm element sizes agree.

4.1.3 Internal Brace Forces

In Chapter 2 the internal braces were defined as in Figure 4.13. The internal brace elements are composed of 4 truss elements and the numbering system is shown in the figure. Due to the symmetry of the cross-section and loading, the forces in members 3 and 4 should have opposite signs but same magnitude.

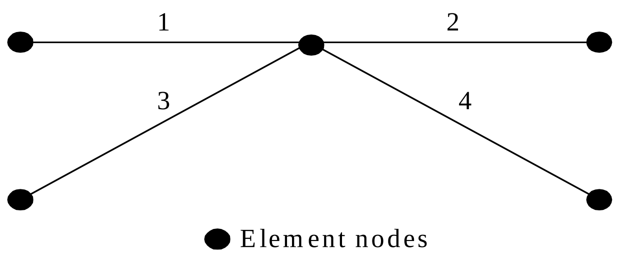


Figure 4.13: Typical Internal Brace Element

In Figures 4.14 and 4.15 the maximum force in the members forming the internal brace are shown for 0^{o}/100 m and 30^{o}/100 m subtended angle and for 4-node and 9-node elements. The maximum force member is the element number 1 for the specified cases.

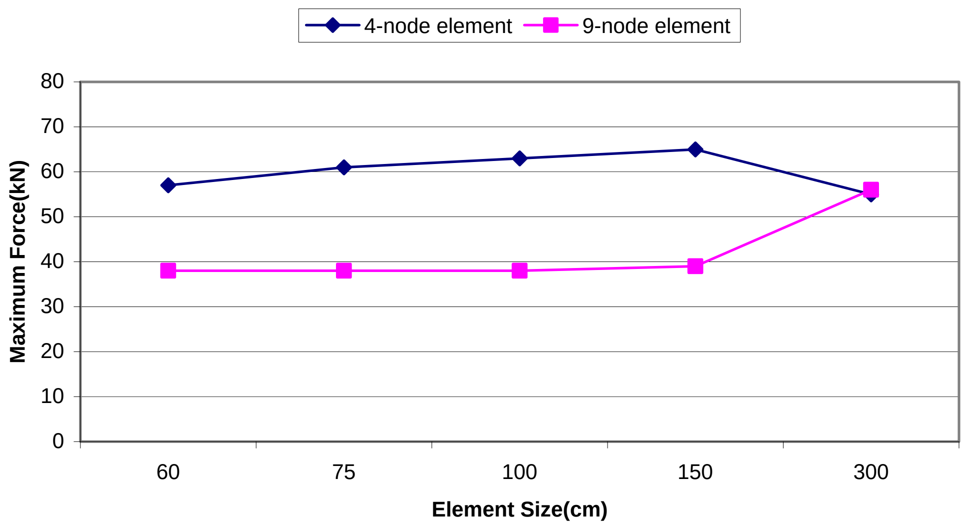


Figure 4.14: Maximum Internal Brace Element Force for 0^{o}/100 m Subtended Angle

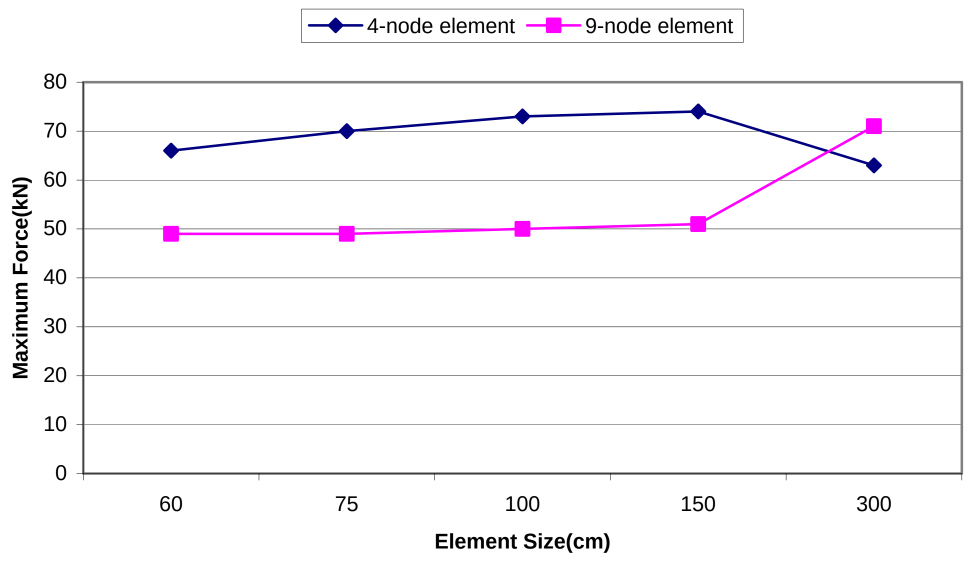


Figure 4.15: Maximum Internal Brace Element Force for 30^{o}/100 m Subtended Angle

As it can be seen in the above figures there is a significant difference in obtained axial force levels when 4-node and 9-node elements are used. Both figures show a big diversion from the trend when the 300 cm element is used which is because of combined the effect of node normals and mesh fineness.

In Table 4.9 the forces in the 4 members forming an internal brace are shown for all 19 internal braces along the bridge for 30^{o}/100 m subtended angle using 9-node elements. From the table it may be concluded that 150 cm element performs very close to 60 cm element, thus it is reliable and suitable for design but the 300 cm element size deviates from the actual values for about %50 or even more for most of the cases. Therefore, 300 cm element size is not feasible.

In Figure 4.16 the force in internal brace number 1 for the first 9 internal braces are plotted for 60 cm, 150 cm and 300 cm element sizes. The rest of the members are not plotted due to the symmetry. In the figure it is clear that although similar results are obtained for 60 cm and 150 cm element sizes 300 cm element size gives far higher results.

Table 4.9: Internal Brace Element Forces for 30^o/100 m subtended angle, 9-node element and 60 cm, 150 cm and 300 cm element size

Internal Brace Number	R=30^o/100 m subtended angle											
	Internal Brace Element 1			Internal Brace Element 2			Internal Brace Element 3			Internal Brace Element 4		
	ES=60 cm	ES=150 cm	ES=300 cm	ES=60 cm	ES=150 cm	ES=300 cm	ES=60 cm	ES=150 cm	ES=300 cm	ES=60 cm	ES=150 cm	ES=300 cm
1	13	14	19	5	6	11	-10	-11	-9	10	11	9
2	22	22	28	19	19	23	-4	-4	-7	4	4	7
3	20	21	32	1	2	17	-25	-25	-20	25	25	20
4	34	35	47	30	30	37	-6	-6	-12	6	6	12
5	27	28	46	-1	0	23	-38	-37	-29	38	37	29
6	42	44	60	37	38	48	-7	-7	-16	7	7	16
7	32	33	55	-3	-1	28	-46	-45	-36	46	45	36
8	47	49	68	41	42	54	-8	-8	-19	8	8	19
9	32	36	59	-4	-2	30	-50	-49	-39	50	49	39
10	49	51	71	43	44	56	-8	-9	-20	8	9	20
11	34	36	59	-4	-2	30	-50	-49	-39	50	49	39
12	47	49	68	41	42	54	-8	-8	-19	8	8	19
13	32	33	55	-3	-1	28	-46	-45	-36	46	45	36
14	42	44	60	37	38	48	-7	-7	-16	7	7	16
15	27	28	46	-1	0	23	-38	-37	-29	38	37	29
16	34	35	47	30	30	37	-5	-6	-12	5	6	12
17	20	21	32	1	2	17	-25	-25	-19	25	25	19
18	22	22	28	19	19	23	-4	-4	-7	4	4	7
19	14	14	19	5	6	11	-10	-11	-10	10	11	10

ES = Element Size

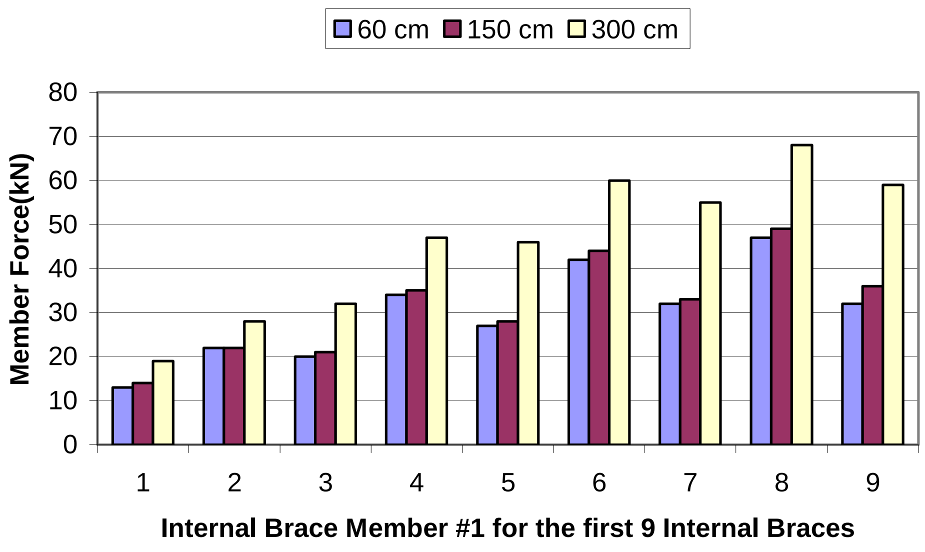


Figure 4.16:Internal Brace Member #1 Forces for the first 9 Internal Braces

As the 4-node element gives more conservative values compared with the 9-node element when equal element sizes are used except the 300 cm element size, the 4-node element can be used at least in the preliminary design phase considering the time and memory advantages it offers.

4.1.4 Cross-sectional Forces

In Chapter 2, it was explained that the program is able to calculate the shear force, bending moment and torsional force within a cross-section. When the bridge is straight no torsional stresses develop so the torsional moment is zero. When degree of curvature is increased gradually, the torsional moments tend to increase.

It is pleasing to state that, as expected, all the element types and element sizes implemented into the program perform successfully and similar results are obtained for all cases that fully agree with the statics.

4.1.5 Cross-sectional Stresses

The program is able to calculate the cross-sectional stresses at specified points within the cross-section. In Figure 4.17 the element integration points within the cross-section for which the cross-sectional stresses are computed are shown.

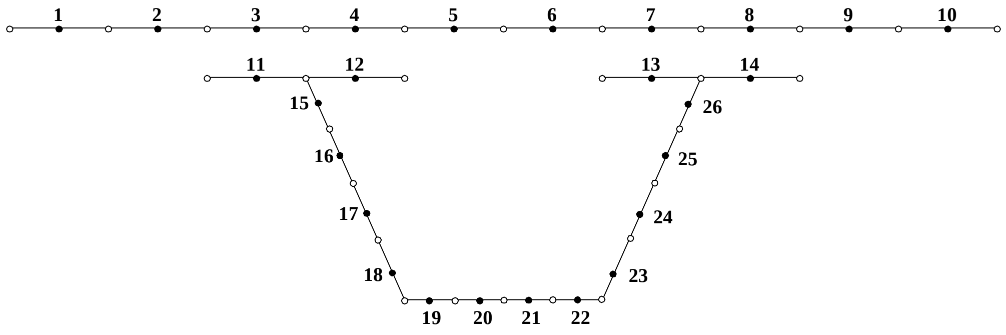


Figure 4.17: Locations Where the Cross-sectional Stresses are Computed

The program is able to calculate both normal and shear stresses at a specific location. In Figure 4.18, the variation of normal stress at 300 cm right of the left support for 0^o/100 m subtended angle is plotted against the element size for both 4-node and 9-node element types at stress location 12.

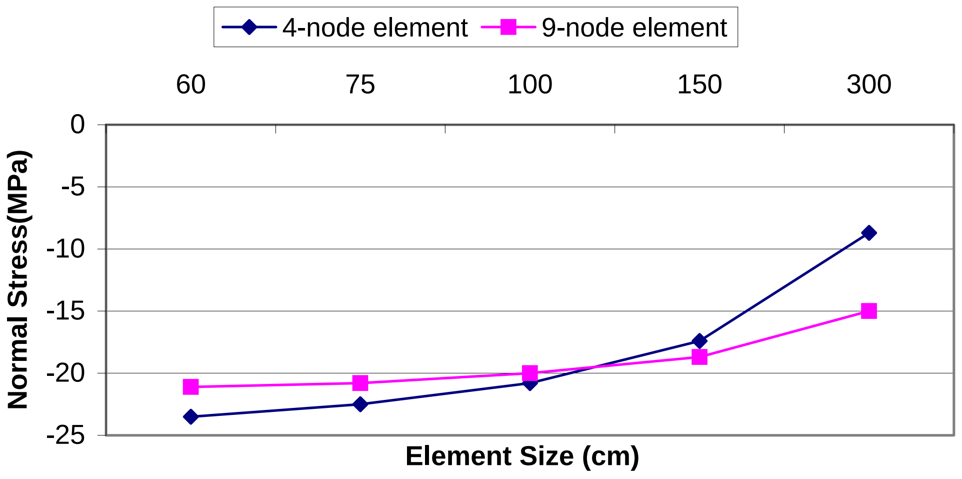


Figure 4.18: Normal Stress vs. Element Size for 0^{o}/100 m Subtended Angle for both 4-node and 9-node Element Types

As it can be seen on the above figure the 4-node element gives conservative values up to 100 cm element size, however after that the reliability of the element size declines. Also the 300 cm element size for the 9-node element deviates from the actual value which is around 20 MPa, therefore while using the 9-node element for detailed analysis the element size should be at least half the panel length.

Similar arguments stated for the 0^{o}/100 m subtended angle can be restated for the 30^{o}/100 m subtended angle. As it can be seen in Figure 4.19 due to the torsional effects involved because of the increase in curvature, the cross-sectional normal stresses are almost doubled.

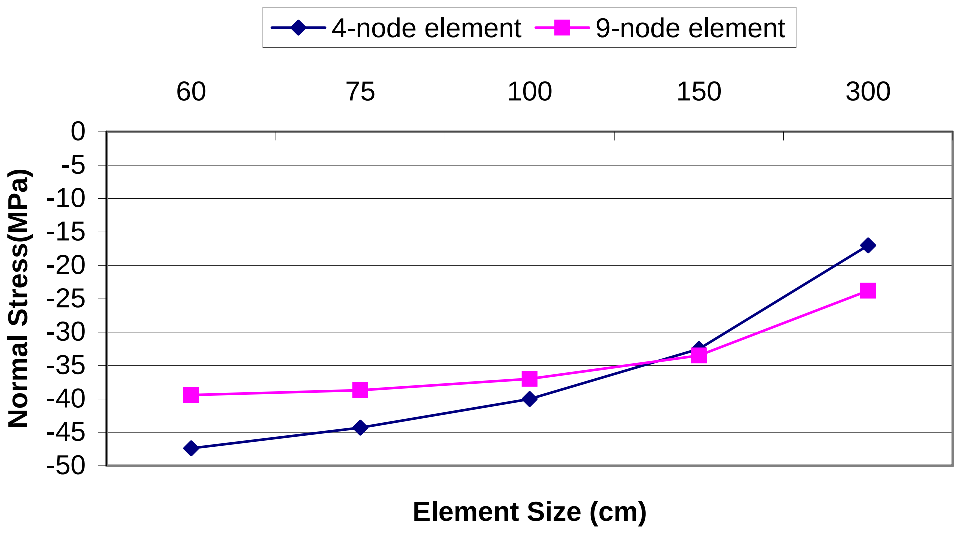


Figure 4.19: Normal Stress vs Element Size for 30^o/100 m Subtended Angle for both 4-node and 9-node Element Types

The above discussion for the normal stresses can be extended for the shear stresses and similar conclusions can be drawn.

4.2 Continuous Bridge Case

The second bridge analyzed is a three span continuous bridge of total length 150 m. The middle span is 60 m long and the side two spans are 45 m each. The cross-sectional dimensions are the same as the simply supported case but at the proximity of the negative moment region the plate thicknesses are doubled as shown in Figure 4.20.

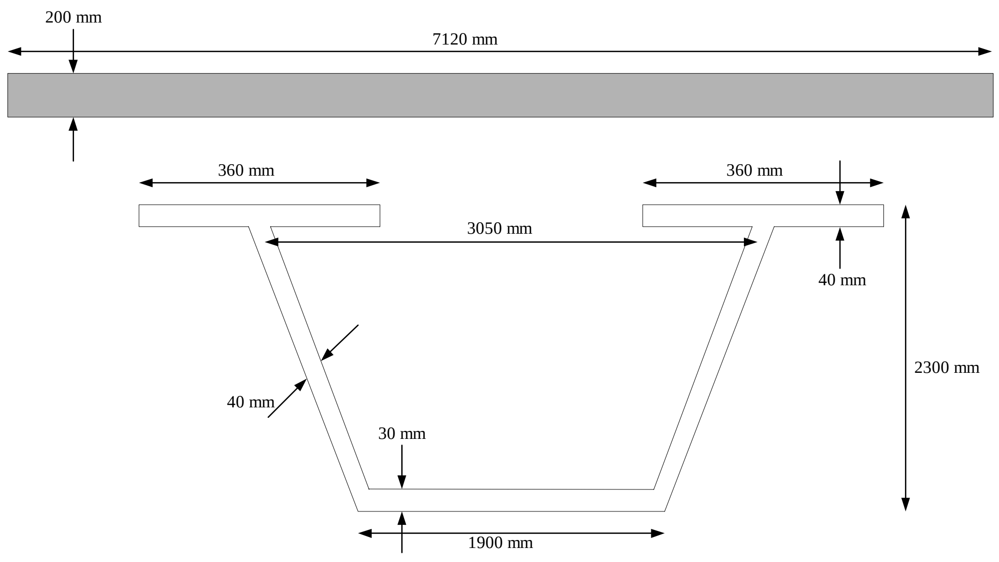


Figure 4.20: Cross-sectional Dimensions at Negative Moment Regions

Negative moment regions are identified as 12 m to the right and to the left of the middle supports, so a total of 48 m bridge length is modeled using the above cross-section. At every 3.0 m except from the supports there is an internal brace, which make a total of 47 internal braces. The panel length is 3.0 m and for each panel there is a top lateral brace as in the simply supported bridge case. The total number of top lateral braces is 50 (Fig 4.21).

The concrete modulus and stud stiffnesses are taken as zero in order to simulate the girder behavior at early ages of concrete.

The continuous bridge is analyzed for 4 subtended angles, which are namely straight, $10^o/100$ m, $20^o/100$ m and $30^o/100$.

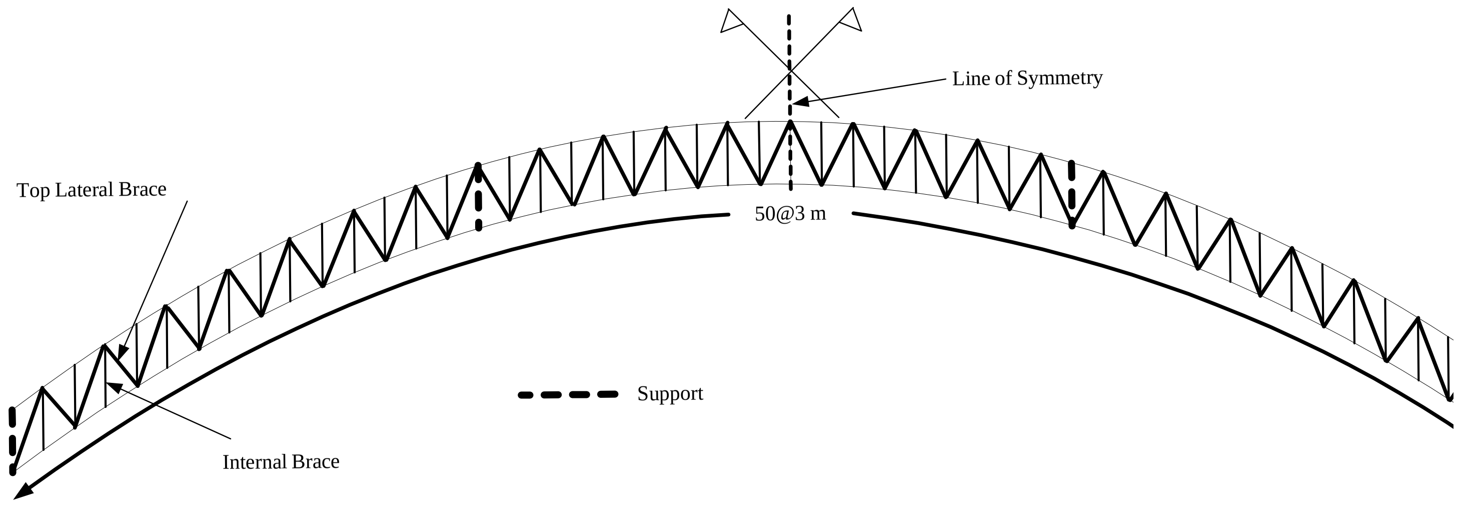


Figure 4.21: Analyzed Bridge

The element size is taken as 60 cm, 100 cm, 150 cm and 300 cm. The analysis parameters are given in Table 4.10.

Table 4.10: Analysis Parameters

Subtended Angle(degrees/100 m)	0, 10, 20, 30
Element Size(cm)	60, 100, 150, 300
Element Type	4-node, 9-node

As it can be inferred from Table 4.10 a total of 32 runs are taken for the continuous bridge case. The supports are at 0 m, 45 m, 105 m and 150m. The uniformly distributed load on the bridge is 19 kN/m and the elastic modulus of the steel is 200 000 MPa.

As in the case of the simply supported bridge, the solution time and required physical memory are recorded and given in Table 4.11.

Table 4.11: Solution Time and Required Memory

Subtended angle/100 m	Element type	Element Size(cm)	Solution Time(sec)	Required Physical Memory(MB)
0^o	4-node	60	14.6	58
		100	8.7	36
		150	5.8	24
		300	2.9	3
	9-node	60	110.6	300
		100	61.2	180
		150	40.7	121
		300	19.9	58

As the number of elements and nodes are same for all subtended angles, in Table 4.11 only the values for 0^o/100 m subtended angle are given. The rest is similar.

In Table 4.11 it is seen that, when the 4-node element is used, the solution time decreases five fold from 60 cm element size to 300 cm element size but the required memory decreases about 20 fold. This is not the case for the 9-node element as shifting from 60 cm element size to 300 cm element size both the solution time and required physical memory drop by five fold. In terms of solution time and required physical memory 4-node element performs better as expected.

4.2.1 Deflections

A representative deflection profile along the bridge for 30^o/100 m subtended angle, 9-node element and 300 cm element size is given in Figure 4.22.

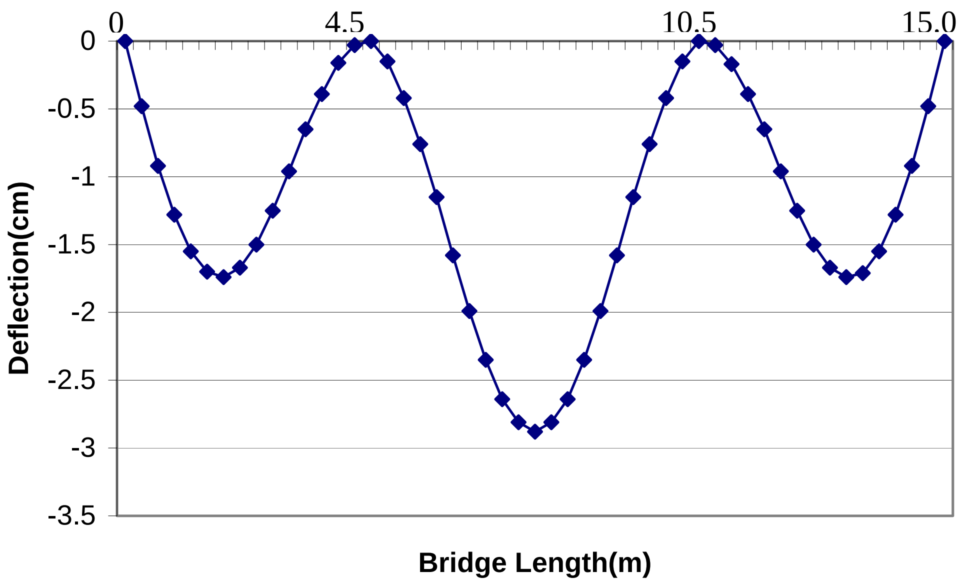


Figure 4.22: Deflection Profile along the Bridge for 30^o/100 m subtended angle, 9-node element and 300 cm element size

In Figure 4.23 maximum deflection is plotted against the element size for 4-node and 9-node elements and 0^o/100 m subtended angle.

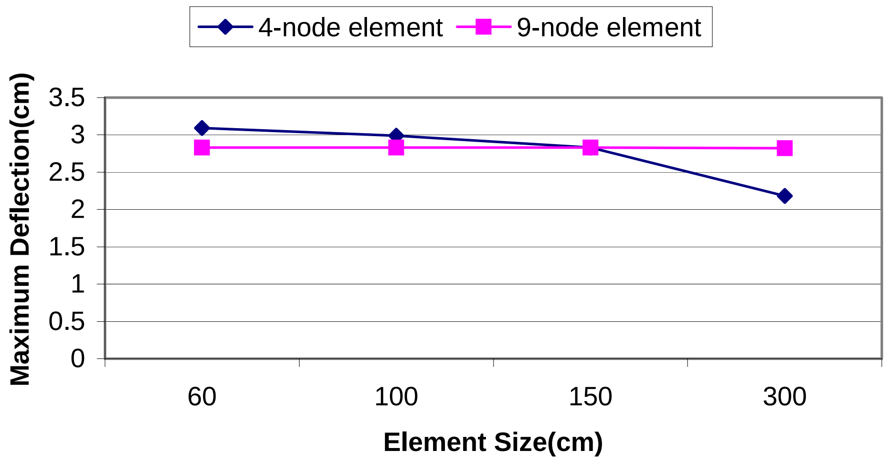


Figure 4.23: Maximum Deflection vs. Element Size

From Figure 4.23 it is understood that in terms of deflections, when using the 9-node element, even the 300 cm element size perform very accurately. However when the 4-node element is used 300 cm element size fails but element sizes up to 150 cm are feasible as they give a little conservative but close values as the 9-node element.

The above discussion is valid for all subtended angles. The reason why the 4-node element gives more conservative results was discussed in Section 4.1.1.1

4.2.2 Top Lateral Brace Forces

The maximum top lateral brace forces for 0^o/100 m subtended angle are drawn for both 4-node and 9-node elements and with respect to the element size are plotted in Figure 4.24.

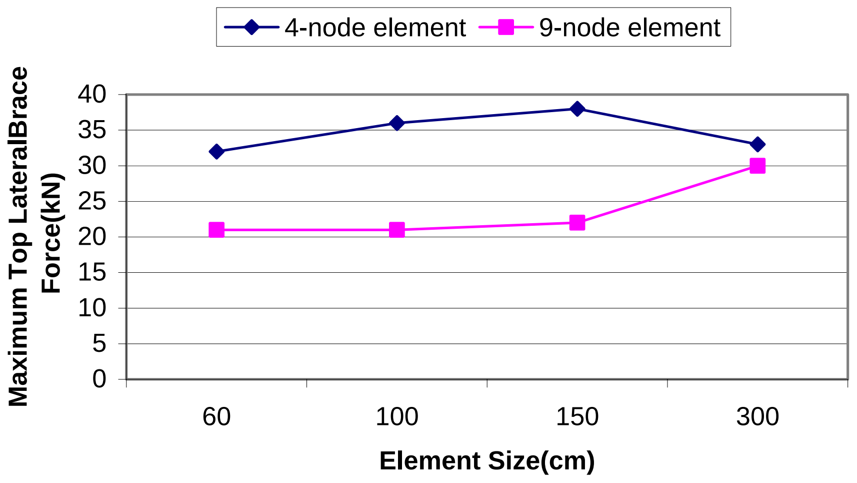


Figure 4.24: Maximum Top Lateral Brace Force vs. Element Size

As it can be seen in the figure above, the 4-node element gives conservative values for all element sizes as compared with the 9-node element. For the 9-node element the 300 cm element size deviates from the correct value. Considering these, it can be stated that, while designing for the top lateral brace members and for the straight bridge case 4-node element can be used keeping in mind that it gives conservative values and while using the 9-node element for detailed design the element size should be at most half the panel length.

The same graph for the straight bridge is repeated for 30^o/100 m subtended angle in Figure 4.25.

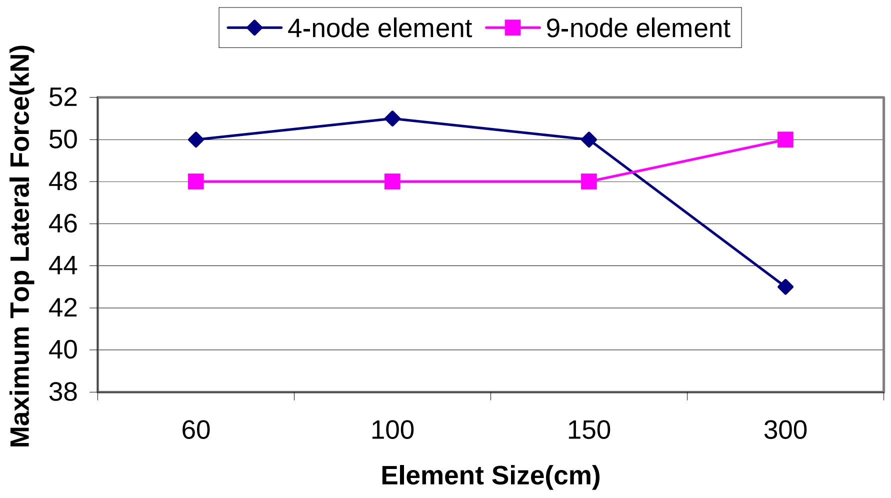


Figure 4.25: Maximum Top Lateral Force vs. Element Size

From the above figure it can be inferred that when the subtended angle increases, the 300 cm element size for both 4-node and 9-node elements become infeasible so it is recommended that while a curved bridge is analyzed, the element length should be at most half the panel length for both element types.

4.2.3 Internal Brace Forces

Similar to the simply supported bridge case the maximum internal brace element force is observed in member 1 (Fig 4.13). The variation of this with respect to the element type and element size for 0^o/100 m subtended angle is given in Figure 4.26.

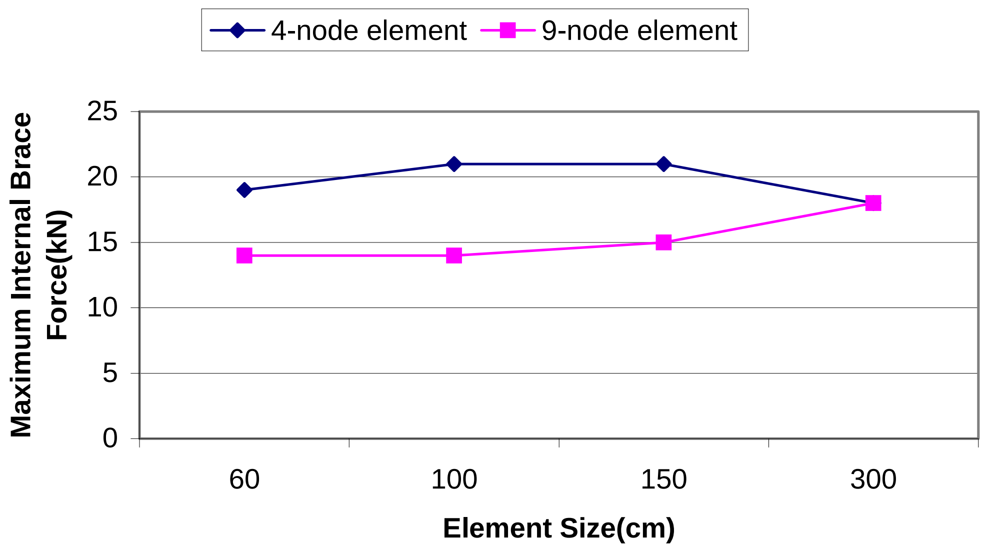


Figure 4.26: Maximum Internal Brace Force vs. Element Size

Similar conclusions stated for the top lateral brace elements can also be drawn for internal braces, such as, when the bridge is straight both 4-node and 9-node elements can be used if conservative results of the 4-node element is desirable. While performing detailed analysis with the 9-node element, 300 cm element size may give far higher results than the actual.

In Figure 4.27 the maximum internal brace element force is plotted for different element sizes and types for 30^{o}/100 m subtended angle. From the figure it can be inferred that, as similar to the top lateral brace forces case, the 300 cm element performs poorly for both element types when the degree of curvature of the bridge increases. Therefore while analyzing the internal braces, the element size should be selected such that it is at most half the panel length.

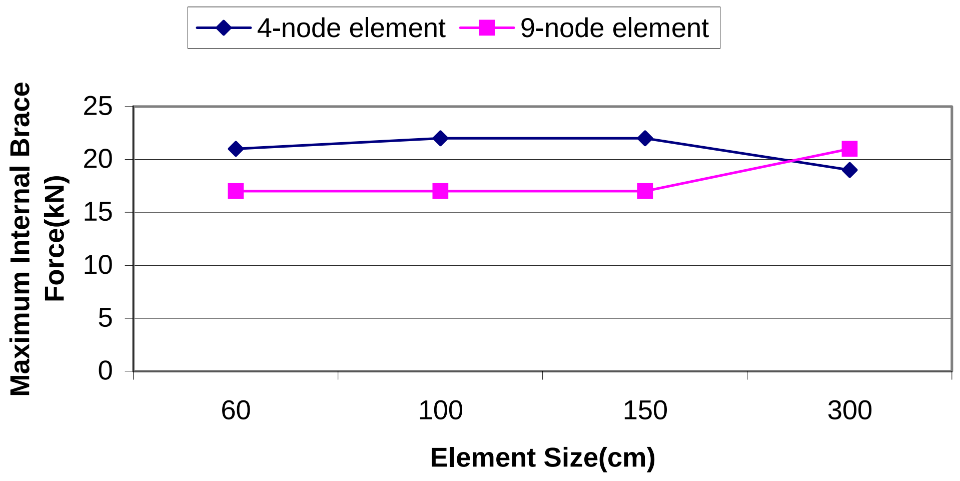


Figure 4.27: Maximum Internal Brace Force vs. Element Size

4.2.4 Cross-sectional Forces

As expected, similar to the simply supported bridge case, the program performs successfully while computing the internal forces, the vertical shear, bending moment and the torsion for all element types, element sizes and subtended angles.

4.2.5 Cross-sectional Stresses

The discussion on cross-sectional stresses for the continuous bridge case is similar to the simply supported bridge case. The 4-node element gives conservative values up to 100 cm element size and after that the reliability of the element declines. Although the 9-node element gives more accurate results, for detailed analysis and design purposes it is recommended that element size should be selected such that it is at most 150 cm.

4.3 Simply Supported Double-I Girder Bridge

The next bridge analyzed is a 60.0 m long double I-girder bridge consisting of 19 external braces each 3.0 m apart. There are 2 top lateral braces(cross brace) for each panel, which make a total of 40 top lateral braces. The uniform loading on the bridge is 30 kN/m and the elastic modulus of the steel is 200 000 MPa. The cross-sectional dimensions are given in Figure 4.28 and plan view of the bridge is shown in Figure 4.29.

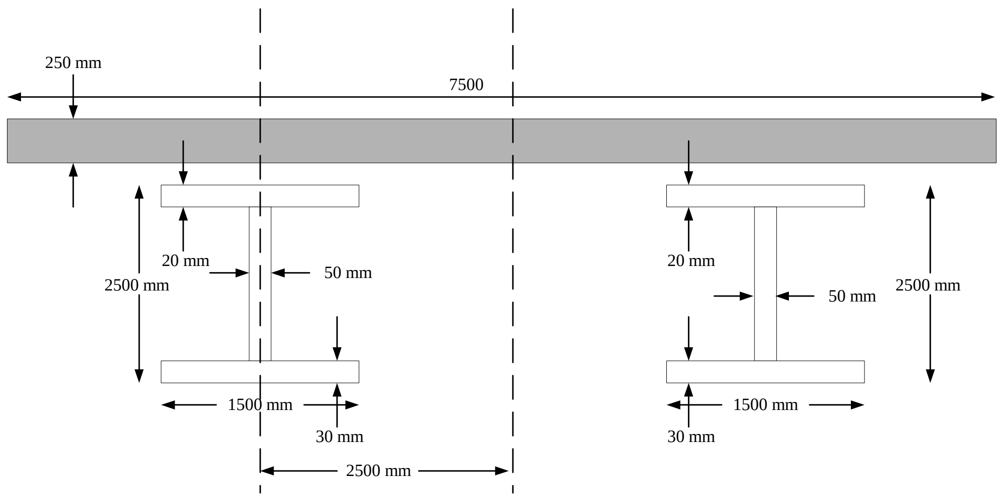


Figure 4.28: Cross-sectional Dimensions

The cross-sectional areas of the external braces are 20 cm^2, and the cross-sectional areas of the top lateral braces are 25 cm^2. The studs are placed at every 60 cm. The concrete modulus and stud stiffnesses are selected very close to zero in order to simulate the behavior of the girder under construction loading at early ages of deck concrete.

The bridge is analyzed for 4 subtended angles, which are namely straight, 10^o/100 m, 20^o/100 m and 30^o/100 m.

The element size is taken as 60 cm, 100 cm, 150 cm and 300 cm. The analysis parameters are given in Table 4.12.

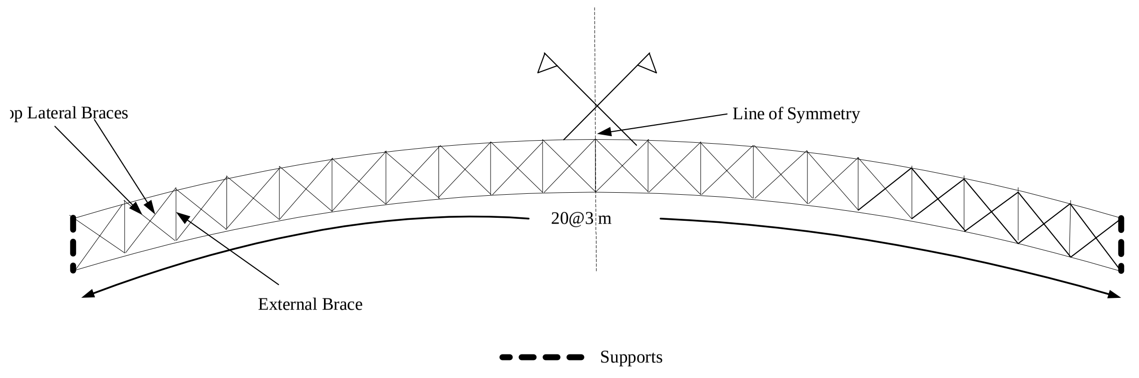


Figure 4.29: Analyzed Bridge

Table 4.12: Analysis Parameters

Subtended Angle (degrees/100 m)	0, 10, 20, 30
Element Size(cm)	60, 100, 150, 300
Element Type	4-node, 9-node

As it can be inferred from Table 4.12 a total of 32 runs are taken for the continuous bridge case. The supports are at 0 m and 60 m.

As in the cases of the simply supported bridge and continuous bridge, the solution time and required physical memory are recorded as given in Table 4.13 for only 0^o/100 m subtended angle. As the number of nodes and elements are same for different subtended angles, as long as the element size does not change the required time and memory will be the same.

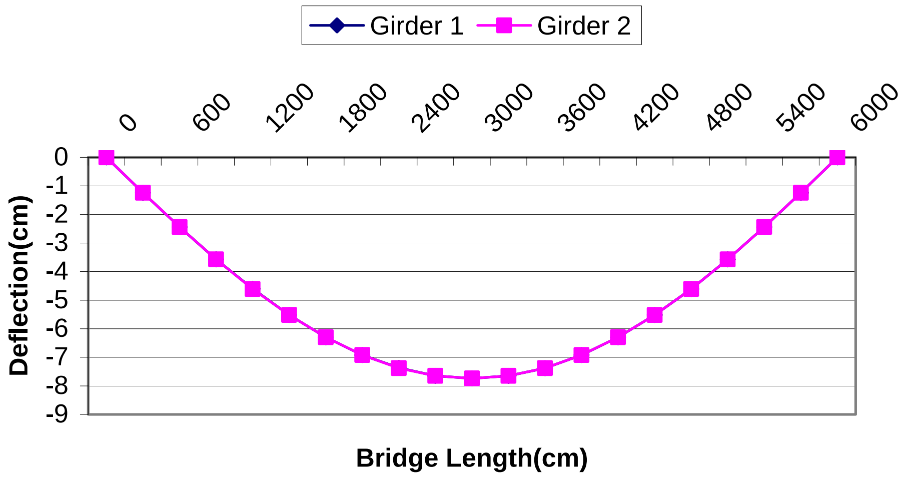
Table 4.13: Solution Time and Required Memory

Subtended angle/100 m	**Element type**	**Element Size(cm)**	**Solution Time(sec)**	**Required Physical Memory(MB)**
0^o	4-node	60	6.4	25
		100	3.8	15
		150	2.6	3
		300	1.4	2
	9-node	60	42.5	118
		100	25.2	69
		150	16.7	46
		300	8.3	23

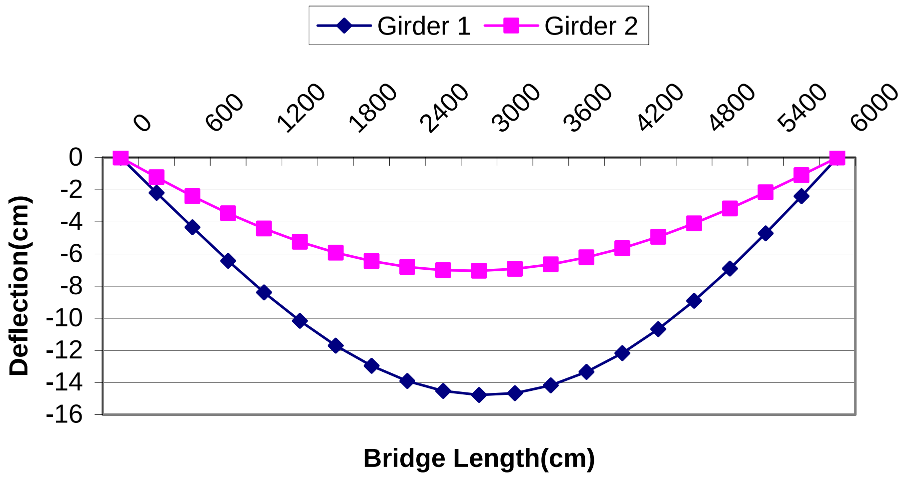
Similar conclusions as those of the simply supported and continuous bridges can be drawn also here, as the 4-node element is utilized faster than the 9-node element and the required physical memory is less. Also there is almost a linear dependence between the element sizes and required time and physical memory.

4.3.1 Deflections

The program calculates displacements for each girder. Representative deflection profiles along the bridge for 0^o/100 m and 30^o/100 m subtended angle, 9-node element and 300 cm element size are given in Figure 4.30 and 4.31.



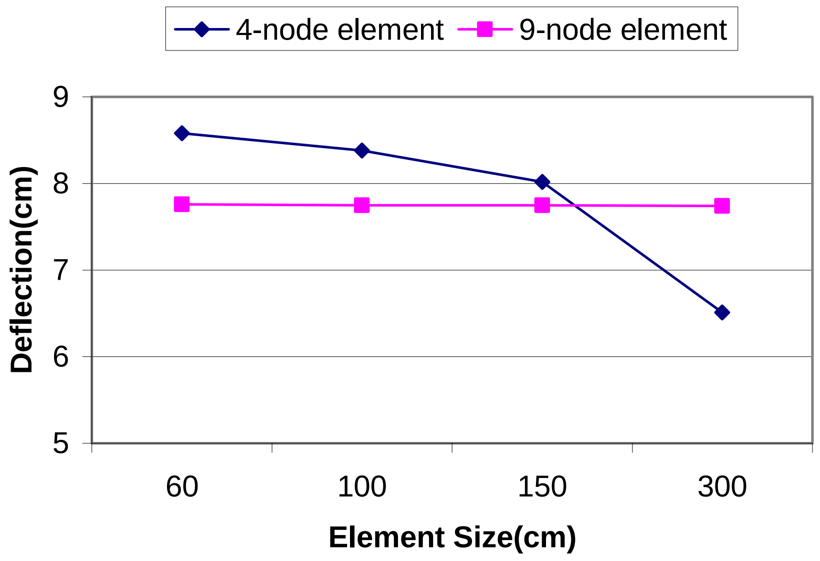
4.30: Deflection Profile Along the Bridge for 0^o/100 m subtended angle, 9-node element and 300 cm element size



4.31: Deflection Profile along the Bridge for 10^o/100 m subtended angle, 9-node element and 300 cm element size

In Figure 4.30 the deflection profiles for girder 1 and girder 2 coincide. As it can be seen in Fig. 4.30 and Fig. 4.31 when the subtended angle thus the torsional moment increases different deflection profiles are observed.

In Figure 4.32 maximum deflection is plotted against the element size for 4-node and 9-node elements and 0^o/100 m subtended angle.



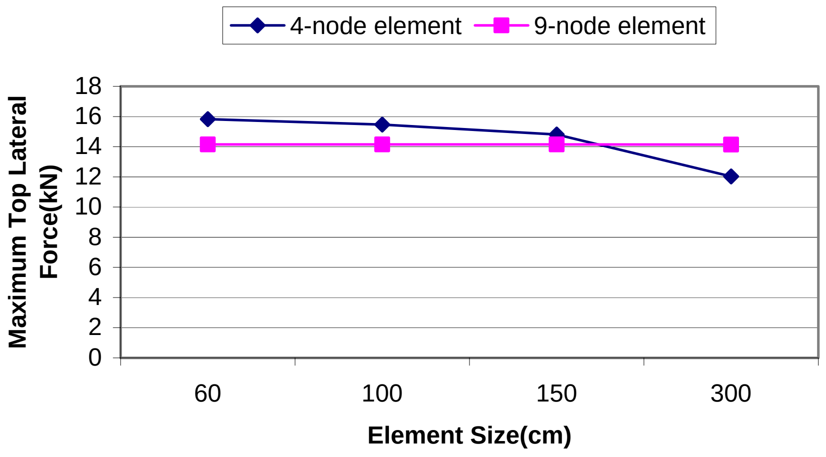
4.32: Maximum Deflection vs Element Size

As in the cases of the other bridges analyzed, 9-node element performs better than the 4-node and for all element sizes and the 4-node element is nor recommended for detailed design. However up to 150 cm element size the 4-node element gives conservative values, thus it may be used in preliminary analysis or may be used when there is time or memory limitation.

The above discussion is valid for all subtended angles. The reason why the 4-node element gives more conservative results for some element sizes was discussed in Section 4.1.1.

4.3.2 Top Lateral Brace Forces

The maximum top lateral brace forces for 0^o/100 m subtended angle are drawn for both 4-node and 9-node elements and with respect to the element size are plotted in Figure 4.33.



4.33: Maximum Top Lateral Force vs. Element Size

As it can be seen in the figure above, the 4-node element gives conservative values up to 150 cm element size as compared with the 9-node element. Similar conclusions that were stated for the simply supported and continuous bridges are valid.

4.3.3 External Brace Forces

Because of the geometry of the cross-section, there are no internal braces in double I-girder systems. Instead, external braces are used. In Chapter 2 the external braces were defined as in Figure 4.34. The external brace elements are composed of 5 truss elements and the numbering system is shown in the figure.

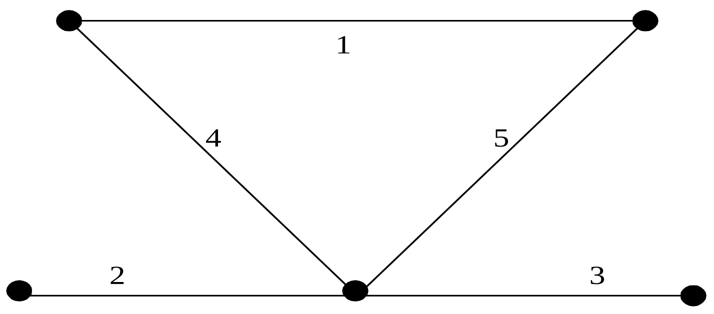


Figure 4.34: Typical External Brace Element

In Figures 4.35 and 4.36 the maximum force in the members forming the external brace are shown for 0^o/100 m and 30^o/100 m subtended angle and for 4-node and 9-node elements. The maximum force element is the element number 1 for the specified cases.

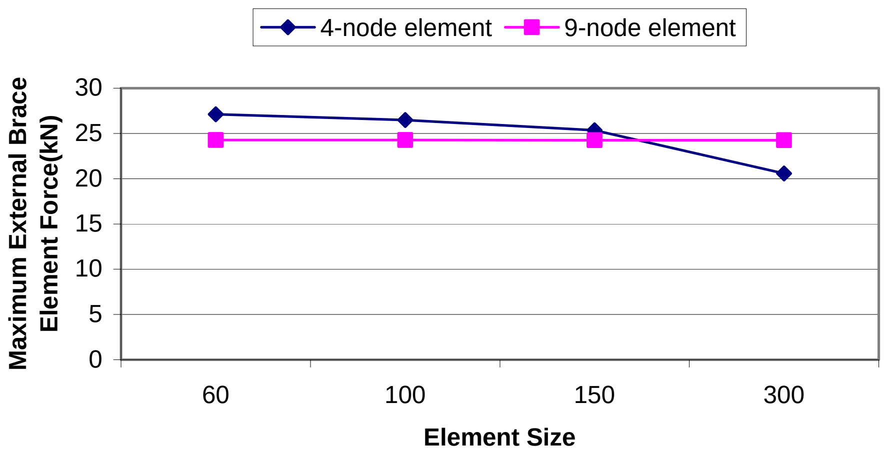


Figure 4.35: Maximum External Brace Element Force for 0^o/100 m subtended angle

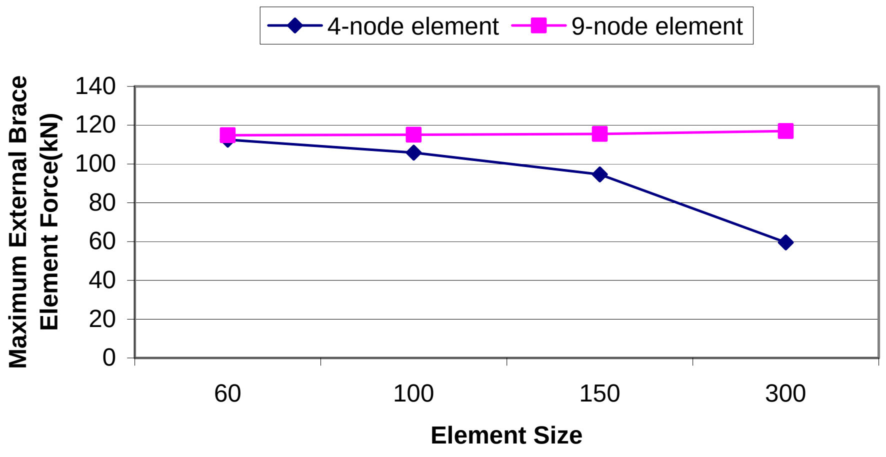


Figure 4.36: Maximum External Brace Element Force for 30^o/100 m subtended angle

The maximum force member for the 0^o/100 m subtended angle is the member 1 of Figure 4.34 and for the 30^o/100 m subtended angle is the member 4 or 5 (these are equal force members in terms of magnitude). When the bridge is straight, the 4-node element behaves conservatively up to 150 cm element size. However if there is torsion the 4-node element is not reliable for all element sizes.

4.3.4 Cross-sectional Forces

As it is expected, similar to the simply supported and continuous bridge cases, the program performs successfully while computing the internal forces, the vertical shear, bending moment and the torsion for all element types, element sizes and subtended angles.

4.3.5 Cross-sectional Stresses

The discussion on cross-sectional stresses for the continuous bridge case is similar to the simply supported bridge case. The 4-node element gives conservative values up to 100 cm element size and after that the reliability of the element decreases. Although the 9-node element gives more accurate results, for detailed analysis and design purposes it is recommended that element size should be selected such that it is at most 150 cm.

4.4 Solver Type and Performance

As previously explained a direct sparse solver of Compaq Extended Math Library (CXML) and an iterative sparse solver called ITPACK was implemented into the program. The direct solver, as the name implies, does not use any iterative technique but directly gives the desired displacement vector. On the other hand ITPACK uses iterative techniques to converge to the desired solution.

ITPACK is a collection of seven FORTRAN subroutines for solving large sparse linear systems employing adaptive accelerated iterative algorithms. The algorithms in ITPACK have been tested most extensively for linear systems arising from elliptic partial differential equations. ITPACK uses four major iterative solution techniques, which are the Jacobi method, the Successive Overrelaxation method, the Symmetric Overrelaxation method and the Reduced System method, and two acceleration procedures which are Chebyshev (Semi-Iteration) and Conjugate Gradient for rapid performance. All the four methods listed cannot be combined with the mentioned two acceleration techniques so ITPACK includes seven subroutines which are Jacobi Conjugate Gradient (JCG), Jacobi Semi-Iteration (JSI), Successive Overrelaxation (SOR), Symmetric Successive Overrelaxation Conjugate Gradient (SSORCG), Symmetric Successive Overrelaxation Semi-Iteration (SSORSI), Reduced System Conjugate Gradient (RSCG) and Reduced System Semi-Iteration (RSSI). The user is able to select the method of choice and the solution is carried out accordingly.

In the main program the required parameters for the method of solution are inputted and then the corresponding subroutine is called from the library of subroutines which is compiled as an independent FORTRAN file. The critical parameters of the iterative solvers which are common for all iterative techniques implemented are the Convergence Tolerance, which is denoted by rparm (1), and the Maximum Number of Iterations, which is denoted by ITMAX, in the program. The convergence tolerance determines the desired accuracy of the result. If the convergence tolerance is selected relatively small the convergence time is higher but the accuracy of the result increases. On the other hand if the maximum number of iterations is selected as a relatively small number the solver stops execution at that specified maximum number of iteration and the accuracy of the obtained result decreases. In order to obtain good performance from the solver given a specific method of iteration an optimum balance between time limit and degree of accuracy should be satisfied.

The matrix storage format of ITPACK is same as the direct sparse solver of CXML, which played an important role in selecting as the iterative solver. A detailed discussion on matrix storage format was discussed in Chapter 2.

4.4.1 Direct vs. Iterative Solver: A Case Study

In order to compare the direct and iterative solvers adopted into the program a generic bridge is selected and it is analyzed with both the direct and the iterative solvers. The bridge considered is a simply supported, 30 m long box-girder bridge with 3.0 m panel length. There are 9 internal braces and 10 top lateral braces which is 1 for each panel. The stud and deck concrete stiffnesses are taken to be zero. The cross-sectional dimensions are same as those of the bridge of Figure 4.1. The elastic modulus of the steel is taken to be 200 000 MPa.

The iterative subroutines that are used for solution are JCG, JSI, SOR, SSORCG and SSORSI subroutines. The RSCG and RSSI subroutines are not considered because they require a different matrix storage format for solution. Therefore the generic bridge is analyzed with CXML direct solver and five iterative methods of ITPACK. The maximum number of iterations is taken as 10 000 and the following solution times are obtained (Table 4.14).

Table 4.14: Solution Times of Solvers

SOLUTION TIME(sec)						
Converge Tolerance	Method of Solution					
	Direct (CXML)	Iterative(ITPACK)				
		JCG	JSI	SOR	SSORCG	SSORSI
1.00E-06	0.7	1.98	10.90	14.6	30.2	28.1
1.00E-05		1.81	4.80	14.6	3.3	28.1
1.00E-04		1.13	4.10	14.6	2.5	23.7
1.00E-03		0.91	3.30	11.9	1.1	3.4
1.00E-02		0.66	2.60	2.8	0.7	0.8
1.00E-01		0.55	1.60	2.6	0.6	0.6

Although the above solution times are reported not all the iterative solution subroutines converge to the true solution. In Table 4.15 the maximum deflection values obtained are given.

Table 4.15: Maximum Deflections obtained using the Solvers

MAXIMUM DEFLECTION(cm)						
Converge Tolerance	Method of Solution					
	Direct (CXML)	Iterative(ITPACK)				
		JCG	JSI	SOR	SSORCG	SSORSI
1.00E-06	0.89	0.89	0.89	0.89	0.89	0.29
1.00E-05		0.89	0.89	0.89	0.89	0.29
1.00E-04		0.89	0.89	0.89	0.89	0.25
1.00E-03		0.89	0.89	0.89	0.89	0.03
1.00E-02		0.89	0.89	1.01	0.87	0
1.00E-01		0.03	0.89	0.99	0	0

As it can be seen on above two tables, the direct solver is faster than the iterative solver. Of the iterative solution techniques considered JCG is the fastest converging technique of all. However, as the convergence tolerance is increased for obtaining the results faster the JCG subroutine fails. JSI gives accurate results for all the convergence tolerance levels but the solution time is 3 to 5.5 times longer than JCG. SOR gives quite accurate results as compared with the SSORCG and SSORSI subroutines but the solution time is higher than the JSI subroutine. Considering the SOR subroutine the solution times are same for 1e-6, 1e-5 and 1e-4 convergence tolerance levels because actually the maximum number of iterations, 10 000, is insufficient for the subroutine to converge for the specified convergence tolerance levels. The obtained maximum deflection results are far below the correct value because the convergence tolerance exceeds the number of significant digits reported. Except for the 1e-6 convergence tolerance level the SSORCG subroutine seems fast but for 1e-2 and 1e-1 convergence levels it cannot converge to the true value. Among all the iterative subroutines, the SSORSI seems to be the poorest. This subroutine cannot converge to the true result for all convergence tolerance levels for 10 000 maximum number of iterations. In Table 4.16 the maximum deflection values using SSORSI subroutine for 1e-6 convergence tolerance level are shown for different maximum number of iteration (ITMAX) values.

Table 4.16: Solution Times and Maximum Deflection for SSORSI Subroutine

ITMAX	Solution Time (sec)	Maximum Deflection (cm)
10 000	28.1	0.29
20 000	55.7	0.49
30 000	83.5	0.62
40 000	111.3	0.71
50 000	138.6	0.77
60 000	166.1	0.81
70 000	194.7	0.83

As ITMAX value is increased the SSORSI subroutine converges to the true result but the solution time increases unfeasibly.

Considering the above results it may be concluded that for the solution of symmetric sparse linear systems the direct solvers are more useful compared with the iterative solvers. The matrices produced by the assembly of shell elements are ill-conditioned which is an unfavorable condition for the iterative solvers. Also the bridge length is selected to be 30, which is fixed for all solver types, is a relatively short length. In practice such bridges are much longer and in the analysis of such bridges using the iterative solvers the error involved will be higher. Among the iterative solution techniques Jacobi Method with Conjugate Gradient acceleration technique is the most successful technique considering both the solution time and quality of convergence. The solution times and convergence qualities of the iterative solvers may be improved by changing the some parameters like convergence tolerance and maximum number of iterations but this requires extensive research and familiarity of the methods which is beyond the scope of this work.

CHAPTER 5
SUMMARY, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

As they are manufactured and erected rapidly, composite bridge construction in highways have become more popular in the last decades. Consisting of single or multi box or I-girders at the bottom and concrete deck at the top, connected to each other with shear studs, internal, external and top lateral braces, these structural systems also offer strength, economy and aesthetics. After the composite action is achieved between the concrete deck and the steel girders, especially the trapezoidal box-girders have very high torsional stiffness and provide adequate strength. However, the problems arise before the deck concrete hardens and composite behavior is achieved through the shear studs. The stresses, originating from the wet concrete, cause girder failures.

In order to simulate the behavior of the bridge before the composite action is achieved a computer program employing Finite Element Method (FEM), called UTRAP was developed (Topkaya, Williamson 2003). This program is able to analyze the bridge for concrete pour sequence and the developed stresses and forces within the steel girders, internal and external braces and top laterals can be monitored at every stage of the construction. This program is able to analyze only the trapezoidal girders. The bridge is modeled with 9-node shell elements and the element size used for creating the finite element mesh is constant. Also the program is not able to analyze variable radius of curvature bridges. The solver adopted into the program was a direct solver of Compaq Extended Math Library (CXML). Finally the system of units that the program can support is only the Imperial System of units.

Keeping the limitations explained in mind, comprising the subject of this thesis work, the program UTRAP was improved. First of all, the program had a rigid structure that improvements cannot be easily be done. In order to overcome this, the program is restructured and converted into a state that improvements can easily be done whenever needed. I-girders are added to the program so that in addition to the trapezoidal girders, single and double I-girders can also be analyzed with the program. A 4-node element was implemented in order to compare the performance of the 9-node and 4-node elements. The element size is stored as a variable in order to adjust the density of the finite element mesh can be adjusted. The program is improved so that it can analyze variable radius of curvature bridges which are commonly observed in practice. An iterative solver package, ITPACK, was adopted into the program in order to test the performance of iterative solvers and compare them with the direct solver. The system of units is arranged so that the user is able to work with any system of units he/she prefers. The improvements to the program are summarized in Table 5.1.

Table 5.1: The Improvements of the Program

UTRAP (2003)	UTRAP (2005)
Has a rigid structures changes cannot be easily implemented	Has a flexible structure
Element size is constant	Element size is variable
Utilizes 9-node shell elements	Utilizes both 9-node and 4-node shell elements
Box Girders can be analyzed	Box Girders and I-girders can be analyzed
Constant curvature and straight bridges are analyzed	Variable curvature and straight bridges are analyzed
The adopted solver is a direct sparse solver	In addition to the direct solver an iterative solver is adopted into the program
Uses Imperial system of units	Uses imperial and metric system of units

In the first part of this thesis, the program structure is explained under pre-processor, processor and post-processor headings. In the preprocessor module, the nodes and elements are generated and their material and geometric properties are assigned. In the processor module, the structural stiffness matrix and the load vector is formed by assembling the individual elements to the global system. The obtained structural stiffness matrix and the load vector are arranged to be compatible with the matrix storage format of the solvers. After this the solution is carried out and the structural displacements are obtained. In the post-processor module, the structural displacements obtained in the processor module are post-processed and the cross-sectional deflections, cross-sectional rotations, top lateral brace forces, internal brace forces, external brace forces, cross-sectional forces and cross-sectional stresses are obtained.

In the second part of the thesis, numerical modeling details are discussed and program verification with existing solutions are carried out. The details of the shell, truss and spring element formulations are explained. Also the required storage format of the matrices for the solvers are discussed. At the end of this part, the program is verified with published solutions.

The third part of this thesis comprises the modeling recommendations for composite bridge analysis. For this part three bridges are taken with different geometric features, girder types, internal, external and top lateral brace configurations, variable curvatures along the length and some finite element modeling advices are given using the developed program. At the end of this part, a comparison is made between direct and iterative solvers and some performance measures are presented.

In conclusion, the analysis capabilities of the program UTRAP were improved. With the improved capabilities the behavior of 4-node and 9-node elements, the convergence rates and accuracies of direct and iterative solvers were compared. Some parametric studies are conducted to provide modeling recommendations for the bridge designers using the developed software. In the future, live load analysis could be implemented into the program. New shell elements and steel girder types could be added. The maximum number of girders, which is limited to two in the current state, could be increased. It is assumed that there is no elevation difference between the supports of the bridges, however in the future elevation difference parameter can be added into the analysis. The cross-sectional dimensions could be variable along the bridge length which is now constant. The stiffener elements could be added and modeled with finite elements. The bridge ends could be dapped which is the common form in practice and finally superelevation could be added into the analysis.

BIBLIOGRAPHY

Topkaya, C. (2002). “Behavior of Curved Steel Trapezoidal Box Girders During Construction.” Doctoral Dissertation, University of Texas at Austin.

Topkaya, C., Williamson, E. B. (2003). “Development of Computational Software for Analysis of Curved Girders under Construction Loads.” Computers and Structures, Vol. 81, pp.2087-2098.

Fan, Z., and Helwig, T. A. (1999). “Behavior of Steel Box Girders with Top Flange Bracing.” ASCE Journal of Structural Engineering, Vol. 125, No. 8, pp.829-937

Cheplak, B. A., Memberg, M., Frank, K. H., Yura, J.A., Williamson, E.B., Chen, B. S., and Topkaya, C.(2002) “Field Studies of Steel Trapezoidal Box Bridges.” Research Report, University of Texas at Austin.

Ahmad, S., Irons, B. M., and Zienkiewicz, O. C. (1970). “Analysis of Thick and Thin Shell Structures by Curved Finite Elements.” International Journal for Numerical Methods in Engineering, Vol. 2, pp.419-451.

Benzi, M., Kouhia, R., and Tuma, M. (1998). “An Assessment of Some Preconditioning Techniques in Shell Problems.” Communications in Numerical Methods in Engineering, Vol. 14, pp.897-906

Kincaid, D. R., Respess, J. R., Young, D. M., and Grimes, R. G. (2003). “ITPACK 2C: A Fortran Package for Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Methods.” Research Report, University of Texas at Austin, Boeing Computer Services Company.

Cook, R. D., Malkus, D. S., Plesha, M. E., Witt, R. J., (2002). "Concepts and Applications of Finite Element Analysis." 4th Ed. John Wiley and Sons.

Reddy, J. N., (1993). "An Introduction to the Finite Element Method." 2nd Ed. McGraw – Hill International Editions

Compaq Extended Math Library (2001). "Reference Guide." Compaq Computer Corporation, Houston, Texas.

ANSYS (1997). "User Manuals and Guides." ANSYS Inc., USA.

Razaqpur, A. G., Nofal, M. (1989). "A Finite Element for Modeling the Nonlinear Behavior of Shear Connectors in Composite Structures." Computers and Structures, Vol. 32, No. 1, pp.169-174.

Tabsh, S. W., Sahajwani, K. (1997). "Approximate Analysis of Irregular Slab-on-Girder Bridges." ASCE Journal of Bridge Engineering, Vol. 2, No. 1, pp. 11-17.

APPENDIX – A SUBROUTINES LIST

In Table A1, the names of the subroutines in the program are listed together with the arguments and functions of the subroutines

Table A-1: Subroutines List

Subroutine name	Arguments	Function
get_nnpcsec	nnpcsec, ngird, isec_type, ielem_type	Computes the number of nodes per cross-section
get_coordinates	ngird, isec_type, ielem_type, numnodes, xy, vai, qai, rai, al_segm, al_rcurv_segm, elemsize	Calls the appropriate subroutines to compute the nodal coordinates and Q, R and V vectors
get_xy111 get_xy112 get_xy121 get_xy122 get_xy211 get_xy212 get_xy221 get_xy222	xy, vai, qai, rai, numnodes, al_segm, al_rcurv_segm, elemsize	Obtain the nodal coordinates of all nodes and obtain Q, R and V vectors for the specified cross-section
get_shell_info	n_elem_pcsec, n_nodes_pelem, n_dof_pnode, ngird, isec_type, ielem_type	Returns the number of elements per cross-section, the number of elements per node and number of degrees of freedom per node for each of the 8 cross-section combinations
elgen_shell	nodes_shell, n_shell_elm, ndiv, ngird, isec_type, ielem_type	Calls subroutines which form the shell elements by assigning the appropriate nodes for each of the 8 cross-section combinations
elgen_shell_111 elgen_shell_112 elgen_shell_121 elgen_shell_122 elgen_shell_211 elgen_shell_212 elgen_shell_221 elgen_shell_222	nodes_shell, n_shell_elm, ndiv	Forms the shell elements by assigning the appropriate nodes for each of the 8 cross-section combinations
elgen_int_brace	nodes_intbr, n_intbr_elm, aloc_int_brc, n_int_brc, ngird, isec_type, ielem_type, elemsize	Calls subroutines that generates the internal braces for the 4 cross-sections having internal braces

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
elgen_int_brace111 elgen_int_brace112 elgen_int_brace211 elgen_int_brace212	nodes_intbr, n_intbr_elm, aloc_int_brc, n_int_brc, elemsize	Generates the internal braces for the specified cross-sections
elgen_ext_brace	nodes_extbr, n_extbr_elm, aloc_ext_brc, n_ext_brc, ngird, isec_type, ielem_type, elemsize	Calls subroutines which form external braces for the 4 cross-sections having external braces
elgen_ext_brace211 elgen_ext_brace212 elgen_ext_brace221 elgen_ext_brace222	nodes_extbr, n_extbr_elm, aloc_ext_brc, n_ext_brc, elemsize	Generates the external braces for the specified cross-sections
elgen_toplt	nodes_toplt, n_toplt_elm, ktype_top_ltr, ielem_type, aloc1_top_ltr, aloc2_top_ltr, n_top_ltr, ngird, isec_type, elemsize	Calls subroutines which form the top laterals for the 6 cross-sections having top lateral braces
elgen_toplt111 elgen_toplt112 elgen_toplt211 elgen_toplt212 elgen_toplt221 elgen_toplt222	nodes_toplt, n_toplt_elm, ktype_top_ltr, aloc1_top_ltr, aloc2_top_ltr, n_top_ltr, elemsize	Generates the top lateral braces for the specified top lateral brace members
elgen_studs	nodes_stud, n_stud_elm, ielem_type, isec_type, ngird	Calls subroutines which form the studs for all the 8 cross-sections
elgen_studs111 elgen_studs112 elgen_studs121 elgen_studs122 elgen_studs211 elgen_studs212 elgen_studs221 elgen_studs222	nodes_stud, n_stud_elm, ncsec	Forms the studs for the specified cross-section
elgen_support	nodes_support, n_support_elm, aloc_support, n_support, elemsize, ngird, isec_type, ielem_type	Calls subroutines which form the supports for all the 8 cross-sections
get_ncsec	ndiv, ielem_type, ncsec	Computes the number of cross-sections

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
elgen_support111 elgen_support112 elgen_support121 elgen_support122 elgen_support211 elgen_support212 elgen_support221 elgen_support222	nodes_support, n_support_elm, aloc_support, n_support, elemsize	Forms the supports for the specified cross-section
gen_pinnodes	aloc_support, n_support, nodes_pin, n_pin_nodes, elemsize, ngird, isec_type, ielem_type	Calls subroutines which form the pinned nodes for all the 8 cross-sections
gen_pinnodes111 gen_pinnodes112 gen_pinnodes121 gen_pinnodes122 gen_pinnodes211 gen_pinnodes212 gen_pinnodes221 gen_pinnodes222	aloc_support, n_support, nodes_pin, n_pin_nodes, elemsize	Forms the pinned nodes for the specified cross-section
form_plib	prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, webt, botft, tft, conc_mod, deckt, steelmodulus	Forms the property library for the shell elements
form_prop	iprop_sh_index, n_shell_elm, al_pour, n_deck, alwebt, nwebt, albotft, nbotft, altft, ntft, ngird, isec_type, ielem_type, elemsize, ndiv	Calls the subroutines which assigns the shell properties for all the 8 cross-sections
form_prop111 form_prop112 form_prop121 form_prop122 form_prop211 form_prop212 form_prop221 form_prop222	iprop_sh_index, n_shell_elm, al_pour, n_deck, alwebt, nwebt, albotft, nbotft, altft, ntft, elemsize, ndiv	Assigns the shell properties by mapping the property library
form_stpr	prop_stud, n_runs, n_stud_elm, n_deck, al_pour, stud_stf, elemsize, ngird, isec_type, ielem_type, ncsec	Calls subroutines which form the stud properties for all the 8 cross-sections

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
form_stpr111 form_stpr112 form_stpr121 form_stpr122 form_stpr211 form_stpr212 form_stpr221 form_stpr222	prop_stud, n_runs, n_stud_elm, n_deck, al_pour, stud_stf, elemsize, ncsec	Assigns the stud properties for the specified cross-section
form_std_mod	stud_mod_fac, n_stud_elm, al_studsp, stud_sp, n_s_flange, n_studsp, elemsize, ngird, isec_type, elem_type, ncsec	Calls the subroutines which modify stud stiffnesses for all the 8 cross-sections
form_std_mod111 form_std_mod112 form_std_mod121 form_std_mod122 form_std_mod211 form_std_mod212 form_std_mod221 form_std_mod222	stud_mod_fac, n_stud_elm, al_studsp, stud_sp, n_s_flange, n_studsp, elemsize, ncsec	Modifies the stud stiffnesses for the specified cross-section
gauss3x3 gauss2x2	weights,xi	Calculates the Gaussian Quadrature data for numerical integration for 9-node and 4-node elements
shaper9 shaper4	xi,psi	Computes the shape functions, first and second derivatives of the shape functions For the 9-node and the 4-node elements
form_ic	ic, ne, n_shell_elm, n_intbr_elm, n_extbr_elm, n_toplt_elm, n_stud_elm, n_support_elm, nodes_shell, nodes_intbr, nodes_extbr, nodes_toplt, nodes_stud, nodes_support, ielem_type, n_nodes_pelem	Stores the node data of each element
form_nonzero	invinc, numnodes, ic, ne, nonzeros, n_dof_pnode	Locates and stores the nonzero entries of the structural stiffness matrix

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
form_connect	invinc, numnodes, ic, ne, icolumns, irowindex, nonzeros, n_dof_pnode	Forms the element connectivity data by filling “irowindex” and “icolumns” vectors
assemb_shell	xy, vai, qai, rai, nodes_shell, n_shell_elm, numnodes, prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, iprop_sh_index, ssm, icolumns, irowindex, nonzeros, irun, n_nodes_pelem, n_dof_pnode	Assembles the shell elements into the structural stiffness matrix
assemb_toplt	xy, nodes_toplt, n_toplt_elm, numnodes, prop_toplt, ssm, icolumns, irowindex, nonzeros, n_dof_pnode	Assembles the top lateral brace elements into the structural stiffness matrix
assemb_intbr	xy, nodes_intbr, n_intbr_elm, numnodes, prop_intbr, ssm, icolumns, irowindex, nonzeros, n_dof_pnode	Assembles the internal brace elements into the structural stiffness matrix
assemb_extbr	xy, nodes_extbr, n_extbr_elm, numnodes, prop_extbr, ssm, icolumns, irowindex, nonzeros, n_dof_pnode	Assembles the external brace elements into the structural stiffness matrix
assemb_support	xy, nodes_support, n_support_elm, numnodes, ssm, icolumns, irowindex, nonzeros, n_dof_pnode, steelmodulus	Assembles the support elements into the structural stiffness matrix
assemb_stud	nodes_stud, n_stud_elm, n_runs, prop_stud, numnodes, ssm, icolumns, irowindex, nonzeros, irun, stud_mod_fac, n_dof_pnode	Assembles the studs into the structural stiffness matrix
apply_support	nodes_pin, n_pin_nodes, ssm, nonzeros, irowindex, numnodes, ngird, n_dof_pnode, n_pin_nodespcsec	Applies the boundary conditions
assign_dist_load	rhs, numnodes, al_pour, n_deck, dist_load, n_runs, irun, n_dof_pnode, elemsize, ngird, isec_type, ielem_type, ncsec	Call subroutines that assign the distributed loading to the nodes for all the 8 cross-sections

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
assign_dist_load111 assign_dist_load112 assign_dist_load121 assign_dist_load122 assign_dist_load211 assign_dist_load212 assign_dist_load221 assign_dist_load222	rhs, numnodes, al_pour, n_deck, dist_load, n_runs, irun, n_dof_pnode, elemsize, ncsec	Assigns the distributed loading to the nodes for all the 8 cross-sections
l_r_shell3d02	-	Forms the element stiffness matrix for shell elements
get_trussk	ek, coords, amatprop	Forms the element stiffness matrix for truss elements
get_intbrk	ek, coords, amatprop	Forms the element stiffness matrix for internal brace elements
get_extbrk	ek, coords, amatprop	Forms the element stiffness matrix for external brace elements
get_supportk	ek, coords, amatprop	Forms the element stiffness matrix for support elements
get_studk	ek, studstf	Forms the element stiffness matrix for studs
invert	a, ainv, det	Returns the inverse of the square matrix “a”
solver	ssm, irowindex, icolumns, rhs, uv, numnodes, nonzeros	Calls the direct solver(CXML)
solver_itpack	ssm, irowindex, icolumns, rhs, uv, numnodes, nonzeros, n_dof_pnode, itype	Calls the iterative solver(ITPACK)
post_defl	uv, numnodes, irun, ndiv, n_dof_pnode, elemsize, ngird, isec_type, ielem_type	Calls the subroutines which post-process the solution vector to obtain the vertical deflections for all the 8 cases

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
post_defl111 post_defl112 post_defl121 post_defl122 post_defl211 post_defl212 post_defl221 post_defl222	uv, numnodes, irun, ndiv, n_dof_pnode, elemsize	Post-processes the solution vector to obtain the vertical deflections for the specified case
post_rot	uv, numnodes, ndiv, irun, botfl, n_dof_pnode, elemsize, ngird, isec_type, ielem_type	Calls the subroutines to post-process the solution vector to obtain the cross-sectional rotations for all the 8 cases
post_rot111 post_rot112 post_rot121 post_rot122 post_rot211 post_rot212 post_rot221 post_rot222	uv, numnodes, ndiv, irun, botfl, n_dof_pnode, elemsize	Post-processes the solution vector to obtain the cross-sectional rotations for the specified case
post_toplt2	xy, numnodes, nodes_toplt, n_toplt_elm, prop_toplt, uv, irun, n_dof_pnode	Post-processes the solution vector to obtain the top lateral brace forces for the 6 cases having top lateral braces
post_intbr	xy, numnodes, nodes_intbr, n_intbr_elm, prop_intbr, uv, irun, n_dof_pnode	Post-processes the solution vector to obtain the internal brace forces for the 4 cases having internal braces
post_extbr	xy, numnodes, nodes_extbr, n_extbr_elm, prop_extbr, uv, irun, n_dof_pnode	Post-processes the solution vector to obtain the external brace forces for the 4 cases having external braces
tr_axforce	coords, displ, amatprop, axforce	Calculates the top lateral forces by multiplying the member stiffness matrix with the structural displacements

Table A-1: Subroutines List (Continued)

Subroutine name	Arguments	Function
post_csec_for	xy, vai, qai, rai, nodes_shell, n_shell_elm, numnodes, prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, iprop_sh_index, uv, nonzeros, irun, totallength, elemsize, ioutelem, n_dof_pnode, ngird, isec_type, ielem_type	Calls the subroutines to calculate the cross-sectional forces for all the 8 cases
post_csec_for111 post_csec_for112 post_csec_for121 post_csec_for122 post_csec_for211 post_csec_for212 post_csec_for221 post_csec_for222	xy, vai, qai, rai, nodes_shell, n_shell_elm, numnodes, prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, iprop_sh_index, uv, nonzeros, irun, totallength, elemsize, ioutelem, n_dof_pnode	Calculates the cross-sectional forces for all the 8 cases
post_sh_nfor9	xy, vai, qai, rai, nodes_shell, n_shell_elm, numnodes, prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, iprop_sh_index, uv, nonzeros, irun, iel, ql, rl, vl, fl3, fl4, fl7, aml3, aml4, aml7, str22, str12	Calculates the contribution of each shell element to the cross-sectional forces for 9-node shell elements
post_sh_nfor4	xy, vai, qai, rai, nodes_shell, n_shell_elm, numnodes, prop_sh_lib, n_runs, nwebt, nbotft, ntft, n_deck, iprop_sh_index, uv, nonzeros, irun, iel, ql, rl, vl, fl3, fl4, aml3, aml4, str22, str12	Calculates the contribution of each shell element to the cross-sectional forces for 4-node shell elements
l_r_shell3d02	-	Forms the element stiffness matrix for shell elements
l_r_shell3d03	-	Forms the element stiffness matrix for shell elements
l_r_shell3d04	-	Forms the element stiffness matrix for shell elements

APPENDIX – B VARIABLES LIST

In Table A2, the names of the variables, descriptions and types of them are given.

Table A-2: Variables List

Variable Name	Description	Type
al_pour	length of the concrete poured	Real array of size(n_deck)
al_rcurv_segm	radius of curvature of a segment	Real array of size(nsegm)
al_segm	length of the bridge segment	Real array of size(nsegm)
al_studsp	length of the stud spacing intervals	Real array of size(n_studsp)
albotft	length of bottom flange thickness change intervals	Real array of size(nbotft)
aloc_ext_brc	location of the external braces	Real array of size(n_ext_brc)
aloc_int_brc	location of the internal braces	Real array of size(n_int_brc)
aloc_support	location of the supports	Real array of size(n_support)
aloc1_top_ltr	starting coordinate of the top laterals	Real array of size(n_top_ltr)
aloc2_top_ltr	ending coordinate of the top laterals	Real array of size(n_top_ltr)
altft	length of top flange thickness change intervals	Real array of size(ntft)
alwebt	length of web thickness change intervals	Real array of size(nwebt)
amatprop	material properties of the shell elements	Real array of size(12)
area_ext_brc	cross-sectional areas of the external braces	Real array of size(n_ext_brc)
area_int_brc	cross-sectional areas of the internal braces	Real array of size(n_int_brc)
area_top_ltr	cross-sectional areas of the top laterals	Real array of size(n_top_ltr)
botfl	bottom flange length	Real
botft	bottom flange thickness	Real array of size(nbotft)
cmd1	input heading	Character
conc_mod	modulus of elasticity of the concrete	Real array of size(n_runs,n_deck)
coords	x,y,z coordinates of the shell elements	Real array of size(3,9)

Table A-2: Variables List (Continued)

Variable Name	Description	Type
deckt	deck thickness	Real
deckw	deck width	Real
displ(5,9)	nodal displacements of the shell elements	Real array of size(5,9)
dist_load	distributed loading	Real array of size(n_runs,n_deck)
drb	the multiplication the matrices to obtain the stresses	Real array of size(9,45)
dshapes	derivatives of the shape functions	Real array of size(2,9,16)
dtheta	internal variable used in the subroutine to form the curvature of the bridge along the length	Real
dummy	variable used for calculating the program execution time	Real
ef	force vector for the shell elements	Real array of size(81)
ek	element stiffness matrix for the shell elements	Real array of size(81,81)
elemsize	element size of the FEM Model	Real
ianalysis_type	analysis type under different loading conditions	Integer
ic	nodes of all elements	Integer array of size(0:9,ne)
icolumns	column numbers of the nonzero entries of the structural stiffness matrix	Integer array of size(nonzeros)
icsec_end	internal variable used in the subroutine to form the curvature of the bridge along the length	Integer Array of size(nsegm)
icsec_segm	internal variable used in the subroutine to form the curvature of the bridge along the length	Integer Array of size(nsegm)
icsec_start	internal variable used in the subroutine to form the curvature of the bridge along the length	Integer Array of size(nsegm)
ielem_type	type of the elements to be used in FEM analysis	Integer
inpcmd	input heading	Logical array of size(20)
invinc	contains information about which node is connected to which element	Integer Array of size(0:15,numnodes)

Table A-2: Variables List (Continued)

Variable Name	Description	Type
ioutelem	an integer number for which the user specifies the results to be displayed for	Integer
iprop_sh_index	property indexes for shell elements	Integer array of size(n_shell_elm)
irowindex	row index numbers of the nonzero entries of the structural stiffness matrix	Integer array of size(numnodes*n_dof_pnode+1)
irun	Internal variable used as a counter for the number of runs	Integer
isec_type	section type of the beams	Integer
isolver_type	equation solver type	Integer
k_type_int_brc	type of the internal braces	Integer array of size(n_int_brc)
ktype_ext_brc	type of the external braces	Integer array of size(n_ext_brc)
ktype_top_ltr	type of top laterals	Integer array of size(n_top_ltr)
n_deck	number of different deck length when pouring concrete	Integer
n_dof_pnode	number of degrees of freedom per node	Integer
n_elem_pcsec	number of elements per cross-section	Integer
n_ext_brc	number of external braces	Integer
n_extbr_elem	number of external braces	Integer
n_int_brc	number of internal braces	Integer
n_intbr_elem	number of internal braces	Integer
n_nodes_pelem	number of elements per node	Integer
n_pin_nodes	number of pinned nodes	Integer
n_runs	number of program analysis runs	Integer
n_runs	number of runs	Integer
n_s_flange	number of studs per flange	Integer array of size(n_studsp)
n_shell_elem	number of shell elements	Integer
n_stud_elm	number of stud elements	Integer
n_studsp	number of stud spacings	Integer
n_support	number of supports	Integer

Table A-2: Variables List (Continued)

Variable Name	Description	Type
n_support_elm	number of support elements	Integer
n_top_ltr	number of top laterals	Integer
n_toplt_elm	number of top laterals	Integer
nbotft	number of bottom flange thickness change intervals	Integer
ncsec	total number of cross-sections	Integer
ndiv	number of divisions(2 divisions per element for 9 node element and 1 division per element for 4 node element)	Integer
ngird	number of girders	Integer
nint	number of integration points	Integer
nnode	number of nodes	Integer
nnpcsec	number of nodes per cross-section	Integer
nodes_extbr	nodes of external braces	Integer array of size(4,n_extbr_elm)
nodes_intbr	nodes of internal braces	Integer array of size(4,n_intbr_elm)
nodes_pin	node numbers of the pinned nodes	Integer array of size(n_pin_nodes)
nodes_shell	nodes of shell elements	Integer array of size(9,n_shell_elm or 4,n_shell_elm)
nodes_stud	nodes of the studs	Integer array of size(2,n_stud_elm)
nodes_support	nodes of the supports	Integer array of size(4,n_support_elm)
nodes_toplt	nodes of top laterals	Integer array of size(2,n_toplt_elm)
nsegm	number of segments having different radius of curvatures along the length	Integer
ntft	number of top flange thickness change intervals	Integer
numnodes	total number of nodes	Integer
nwebt	number of web thickness change intervals	Integer
offset	girder offset length	Real
prop_extbr	properties of the external braces	Real array of size(3,n_extbr_elm)

Table A-2: Variables List (Continued)

Variable Name	Description	Type
prop_intbr	properties of the internal braces	Real array of size(3,n_intbr_elm)
prop_sh_lib	property library for the shell elements	Real array of size(n_runs,4,nwebt+nbotft +ntft+n_deck)
prop_stud	properties of the studs	Real array of size(n_runs,n_stud_elm)
prop_toplt	properties of the top laterals	Real array of size(3,n_toplt_elm)
prpject	project name	Character
psi	shape functions, first and second derivatives of the shape functions	Real array of size (3,9)
qai	unit vectors defining shell geometry	Real arrray of size(3,numnodes)
qi	vectors defining the shell geometry	Real array of size(3,9)
rai	unit vectors defining shell geometry	Real arrray of size(3,numnodes)
rhs	right hand side of the system of linear equations	Real array of size(nonzeros)
ri	vectors defining the shell geometry	Real array of size(3,9)
segm_legth	length of each segment having a different radius of curvature	Real array of size(n_segm)
shapes	shape Functions	Real array of size(9,16)
ssm	structural Stiffness Matrix	Real array of size(numnodes*n_dof_pno de)
start	variable used for calculating the program execution time	Real
steelmodulus	modulus of Elasticity of Steel	Real
stud_mod_fac	stud modification factor	Real array of size (n_stud_elm)
stud_sp	stud spacing	Real array of size(n_studsp)
stud_stf	stud stiffnesses	Real array of size(n_runs,n_deck)
tft	top flange thickness	Real array of size(ntft)
tfw	top flange width	Real

Table A-2: Variables List (Continued)

Variable Name	Description	Type
theta	internal variable used in the subroutine to form the curvature of the bridge along the length	Real
topl	top length	Real
tot_length	internal variable used in the subroutine to form the curvature of the bridge along the length	Real
totallength	total length of the bridge after adjusting the element size	Real
totime	variable used for calculating the program execution time	Real
uv	structural displacements	Real Array of size(numnodes*n_dof_pnode)
vai	unit vectors defining shell geometry	Real arrray of size(3,numnodes)
vi	vectors defining the shell geometry	Real array of size(3,9)
webd	depth of web	Real
webt	web thickness	Real array of size(nwebt)
weights	weights of the Gaussian Quadrature	Real array of size(9)
xi	integration points of the Gaussian Quadrature	Real array of size(2,9)
xy	x,y,z coordinates of the nodes	Real array of size(3,numnodes)

APPENDIX – C USER’S MANUAL

As discussed previously the inputs to the program are entered through a text file named “utinp” which is in the same folder as the executable file of the program. After the execution of the program, the outputs are stored as individual text files. The output files are also stored in the same folder as the input file. Table A-3 lists the names of the outputs files and the description of information stored in them.

Table A-3: The Outputs Files and Their Descriptions

Output File Name	**Description of the Contents of the Output File**
axf0987	The number of the top lateral braces and the top lateral brace forces.
ebr0987	The number of the external braces and the external brace forces at each element.
ibr0987	The number of the internal braces and the internal brace forces at each element.
def0987	The location along the bridge and the vertical deflection value at that specific location.
rot0987	The location along the bridge and the angular rotation value at that specific location.
sec0987	The location along the bridge and the shear force, bending moment and torsional moment at that specific location.
str0987	The location along the bridge and the normal and shear stresses at that specific location.

In Table A-4 a sample input file and the descriptions of the input fields are listed.

Table A-4: Sample Input File and Descriptions of the Input Fields (*)

Input Field	**Description**
prname “input file “	• name of the project.
ngird 2	• number of girders, 1 or 2.
elemsize 150	• element size, 150 cm in this case.
steelmodulus 20000000	• elastic modulus of steel, 20 000 000 N/cm^2 in this case.
no_of_segments 3 6000 , 1200 6000, 0 6000, -1200	• number of segments with different radius of curvature, • segment length, radius (3, 6000 cm long segments, (-) sign indicates negative curvature
section_type 1	• section type, 1 for box girder and 2 for I-girder.
secdim 230 , 190 , 305 , 36 , 712 , 20 , 230	• section dimensions in cm (web depth, bottom flange width, top length, top flange width, deck width, deck thick, offset, respectively).
webthick 3 6000 , 1 9000 , 2 3000 , 4	• number of different web thicknesses along length, • length of segment, thickness.
botfthick 2 7500 , 1 7500 , 2	• number of different bottom flange thicknesses along length, • length of segment, thickness.

Table A-4: Sample Input File and Descriptions of the Input Fields (Continued)

tfthick 4 4500 , 1 4500 , 2 4500 , 4 4500 , 5	• number of different top flange thicknesses along length. • length of segment, thickness(in cm)
internal_brace 8 2000, 1 , 15 4000, 1 , 15 6000, 1 , 15 8000, 1 , 15 10000, 1 , 15 12000, 1 , 15 14000, 1 , 15 16000, 1 , 15	• number of internal braces, • location, type, cross-sectional area.
external_brace 10 2000, 1 , 20 4000, 1 , 20 6000, 1 , 20 8000, 1 , 20 10000, 1 , 20 12000, 1 , 20 14000, 1 , 20 16000, 1 , 20	• number of external braces, • location, type, cross-sectional area.

Table A-4: Sample Input File and Descriptions of the Input Fields (Continued)

top_lateral 6 1 , 0 , 3000 , 25 2 , 3000 , 6000 , 25 1 , 6000 , 9000 , 25 2 , 6000 , 9000 , 25 1 , 9000 , 12000 , 25 2 , 12000 , 15000 , 25	• number of top lateral braces, • type, start location, end location, cross-sectional area.
support 3 0 9000 18000	• the number of supports. • support locations.
element_type 1	• element type, 1 for 9-node element and 2 for 4-node element.
Studs 2 9000 , 200 , 2 9000 , 150 , 3	• number of different stud roperty es, • length, stud spacing, number of studs per flange.
analysis_type 1	• analysis type, 1 for pour sequence analysis, the program supports only “1” in its present situation.
solver_type 2	• solver type, 1 for direct solver and 2 for iterative solver.
pour_seq 1 1 600 0.00001 , 0.00001 ,190	• number of runs, • number of deck segments, • length of the deck segment, • concrete modulus, stud stiffness and distributed load value.

Table A-4: Sample Input File and Descriptions of the Input Fields (Continued)

outelmnum 2	• stress display per element.

(*) Any consistent system of units can be entered.