Cost Optimization of Fully Continuous Prestressed Concrete I-Girder of Bridge

by

Munshi Galib Muktadir

MASTER OF SCIENCE IN CIVIL ENGINEERING (STRUCTURAL)

Department of Civil Engineering
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

March, 2018

Cost Optimization of Fully Continuous Prestressed Concrete I-Girder of Bridge

by

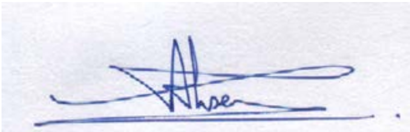
Munshi Galib Muktadir

A thesis submitted to the Department of Civil Engineering of Bangladesh University of Engineering and Technology, Dhaka, in partial fulfilment of the requirements for the degree of

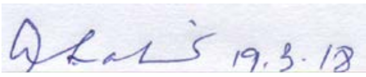
MASTER OF SCIENCE IN CIVIL ENGINEERING (STRUCTURAL)

The thesis titled **"Cost Optimization of Fully Continuous Prestressed Concrete I-Girder of Bridge"** submitted by **Munshi Galib Muktadir, Roll No.: 0412042330, Session: April 2012** has been accepted as satisfactory in partial fulfilment of the requirement for the degree of M.Sc. Engineering (Civil and Structural) on 19th March, 2018.

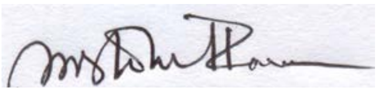
BOARD OF EXAMINERS



Dr. Raquib Ahsan — **Chairman** (Supervisor)
Professor
Department of Civil Engineering
BUET, Dhaka.



Dr. Ahsanul Kabir — **Member** (Ex-officio)
Professor and Head
Department of Civil Engineering
BUET, Dhaka.



Dr. Shohel Rana — **Member**
Assistant Professor
Department of Civil Engineering
BUET, Dhaka.

Dr. Md. Mahmudur Rahman — **Member** (External)
Professor
Department of Civil Engineering
AUST, Dhaka

DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Munshi Galib Muktadir

ACKNOWLEDGEMENT

At first I would like to express my whole hearted gratitude to Almighty Allah for each and every achievement of my life. May Allah lead every person to the way of for-ever successful.

I would like to express my great respect and gratitude to my thesis supervisor, Dr. Raquib Ahsan, Professor, Department of Civil Engineering, BUET for providing me continuous support and guideline to perform this research work and to prepare this concerted dissertation. His contribution to me can only be acknowledged but never be compensated. His consistent inspiration helped me to work diligently throughout the completion of this research work and also contributed to my ability to approach and solve a problem. Despite many difficulties and limitations he tried his best to support the author in every field related to this study. I would also like to express my respect and gratitude to my teacher Dr. Shohel Rana, Assistant Professor, Department of Civil Engineering, BUET. It was not easy to complete this work successfully without his invaluable suggestions and continuous help and encouragement.

I would like to express my deepest gratitude to the Department of Civil Engineering, BUET, The Head of the Department of Civil Engineering and all the members of BPGS committee to give me such a great opportunity of doing my M.Sc. and this contemporary research work on structural optimization.

I would like to render sincere gratitude to Dr. S. N. Ghani for providing useful knowledge on optimization.

I would like to convey my gratefulness and thanks to my parents and my family, their undying love, encouragement and support throughout my life and education. Without their blessings, achieving this goal would have been impossible.

At last I would like to thank my department and my respected supervisor Dr. Raquib Ahsan once again for giving me such an opportunity, which has obviously enhanced my knowledge and skills as a structural engineer to a great extent.

ABSTRACT

Optimum design of a two-span continuous post-tensioned prestressed concrete I-girder of a bridge super-structure is presented in the thesis. The objective is to minimize the total cost of the girders of the bridge considering the cost of materials, fabrication and installation. The design variables considered for the cost minimization of the girders of the bridge, are girder spacing, various cross sectional dimensions of the girder, number of strands per tendon, number of tendons, tendon configuration, slab thickness and ordinary reinforcement for deck slab and girder. Explicit constraints on the design variables are considered on the basis of geometric requirements, practical dimension for construction and code restrictions. Implicit constraints for design are considered according to AASHTO LRFD 2007.

The present optimization problem is characterized by having mixed continuous, discrete and integer design variables and having multiple local minima. Hence a global optimization algorithm called EVOP, is adopted which is capable of locating directly with high probability the global minimum without any requirement for information on gradient or sub-gradient. A computer program is developed to formulate optimization problem which consists of mathematical expression required for the design and analysis of the bridge system, three functions: an objective function, an implicit constraint function and an explicit constraint function and input control parameters required by the optimization algorithm. To determine the design moment and shear for the two-span continuous girder at various positions of the span, the computer program was incorporated with computer application of stiffness method to solve the indeterminate girder. No generalized equation for influence line of indeterminate girders was used, rather coordinates of the non-linear influence line were determined using basic stiffness method concept and were used to determine design live load moment and shear. Finally, to solve the problem, the program is linked to the optimization algorithm.

As constant design parameters have influence on the optimum design, the optimization approach is performed for various such parameters resulting in considerable cost savings. Parametric studies are performed for various girder spans (40 m, 60 m and 80m), girder concrete strengths (40 MPa and 50 MPa) and three different unit costs of the materials including fabrication and installation. From the parametric study, it is found that, optimum girder depth increases with increase in cost of steels. On an average, girder depth increases 22% with increase in cost of steel for 40 MPa concrete. On the other hand, for 50 MPa concrete, the average increase in girder depth comes out to be 19%. Optimum number of strand is higher in higher span girder. Number of strand decreases 17% with increase in cost of steel for 40 MPa concrete. In case of 50 MPa concrete, the average decrease in number of strand is 16%. Girder spacing is found to be higher in smaller span than larger span girder and optimum deck slab thickness comes out to be higher in shorter span as the girder spacing is higher in shorter span.

CONTENTS

List of Figures

List of Tables

List of Abbreviation

AASHTO	American Association of State Highway and Transportation Officials
BRTC	Bureau of Research Testing & Consultation
EVOP	Evolutionary Operation
PC	Prestressed Concrete
PCI	Precast/ Prestressed Concrete Institute
RHD	Roads and Highway Department, Bangladesh.

NOTATION

A_{net}	= Net area of the precast girder;
A_{tf}	= Transformed area of precast girder;
A_s	= Area of prestressing steel;
b	= Average flange width;
d	= Effective depth for flexure;
d_s	= Effective depth for shear;
d_{req}, d_{prov}, d_{min}	= Required, provided and minimum effective depth of deck slab respectively;
e_i, e	= Eccentricity of tendons at initial stage and at final stage of precast section respectively;
EFW	= Effective flange width;
f_y^*	= Yield stress of pre-stressing steel = 0.9 f_{su};
f_r	= Modulus of rupture of girder concrete = $0.625\sqrt{f'_c}$ (MPa);
f_{pu}	=Ultimate strength of prestressing steel;
f_y	= Yield strength of ordinary steel;
f_c	= Compressive strength of girder concrete;
f_{cdeck}	= Compressive strength of deck slab concrete;
f_{ci}	= Compressive strength of girder concrete at initial stage;
F_{1i}, F_{2i}, F_{3i}, F_{4i}	= Prestressing force after instantaneous losses at various sections;
I_{net}	= Moment of inertia of net girder section;
I_{tf},	= Transformed moment of inertia of precast section;
K	= Wobble coefficient;
S_t, S_{tc}	= Section Modulus of top fiber of transformed precast & composite section respectively;
S_{deck}	= Section Modulus of deck slab;
Y_1, Y_2, Y_3, Y_{end}	= Centroidal distance of tendons from bottom at section1, section2, section3 and end section respectively;
μ	= Friction coefficient;
δ	= Anchorage Slip;

CHAPTER 1

INTRODUCTION

1.1 General

In structural design process, the main objective of the designer is to design a structure that will perform its desired performance in terms of strength (so that safety is assured) and serviceability (so that the desired use of the structure is assured). These main two objectives of the designer should be achieved at as low a cost as possible (so that economy is assured). But in the traditional process of structural design, the designer focuses mainly on attaining the first two objectives i.e. on attaining required strength and serviceability and often sacrifices the economy issue. It is not that the designer willingly avoids to reach the design which will result in the lowest cost, rather he avoids this attempt because it is a very slow, difficult, tedious and therefore costly process. The reason of not making an attempt to reach most cost effective design is tried to be focused on in the following writings. The discussion is done from the perspective of bridge design.

1.2 Difficulties in Attaining the Most Cost Optimum Design

Suppose it is desired to construct a structure (bridge) across a river (Fig. 1.1, Fig. 1.2, and Fig. 1.3). The width of the bridge (no. of lanes) is fixed. There are many types of options (i.e. type of bridges) to be chosen for this purpose. For the present case, a girder type bridge has been chosen. A girder bridge transfers load through girder/beam, deck/slab, pier and footing system. For the above case, as the width of the river is small, no intermediate pier and footing is considered in between two abutments at two banks. The target is now to design the girders and the slab/deck.

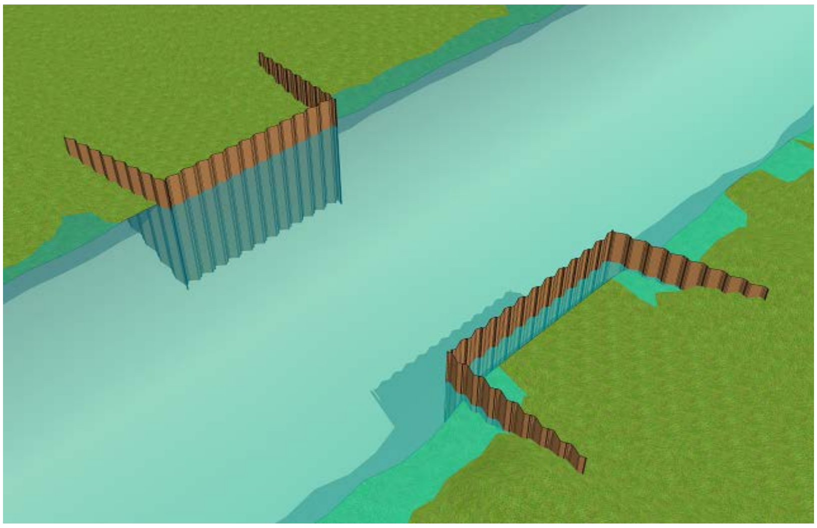


Fig. 1.1 River cross section

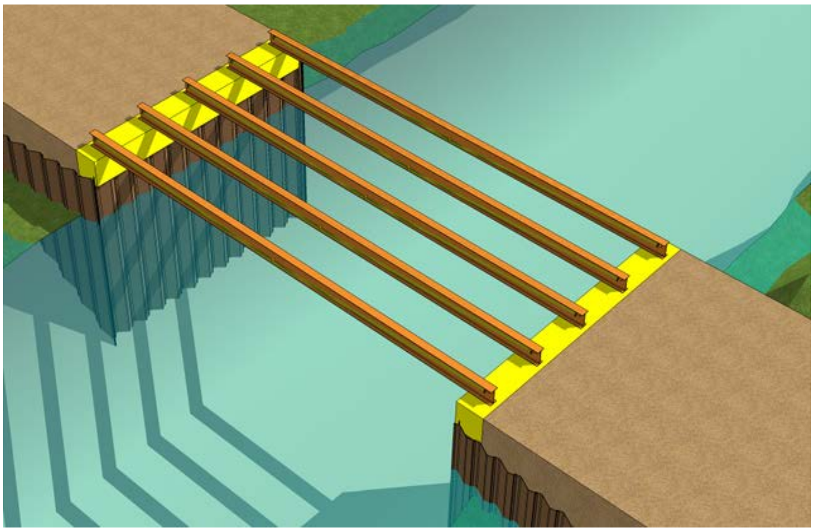


Fig. 1.2 Girders / Beams

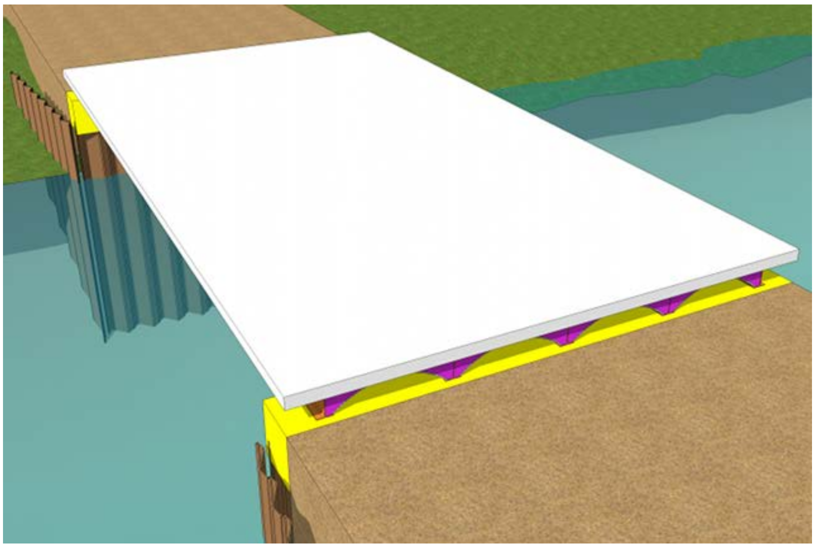


Fig. 1.3 Deck on girders

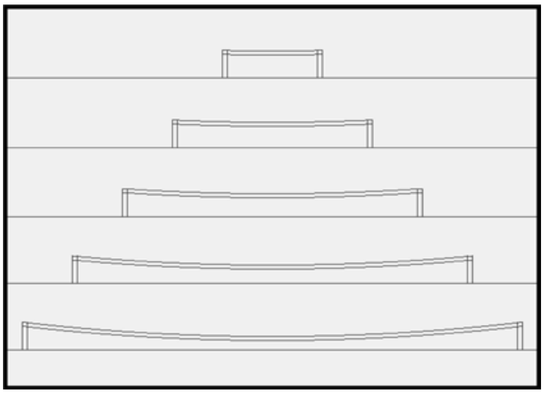


Fig. 1.4 Increasing span length

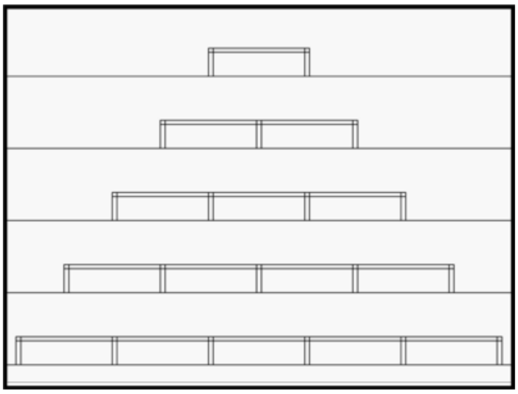


Fig. 1.5 Increasing no. of supports

Now, it is seen from Fig 1.4 and Fig. 1.5 that there can be many combinations of girder and deck type to satisfy the required design strength and serviceability. It is evident that if the spacing between the girders is increased, the thickness of the deck has to be increased which will increase the cost of material. Again, if spacing between the girders is reduced, the thickness of the deck could be reduced which will reduce the cost of material. But reducing girder spacing means increased number of girders which will again increase the cost simultaneously. Now, determining which option is the best one in terms of cost-efficiency is a process of trial and error.

For dealing with only two design variables i.e. thickness of deck and number of girders makes the above problem easy to solve. But if it is considered that increasing number of girder actually reduces load on each girder, then it is evident that increasing number of girders will not add to the cost in a linear-proportionality as the cross section of each girder is decreasing simultaneously.

Now, a larger bridge is considered in the following section and some other possible design variables are being introduced.

Fig. 1.6 shows the end part of a large bridge showing only two supports of the bridge. One of the supports is the abutment at the bank and the other is the pier and footing in the river.

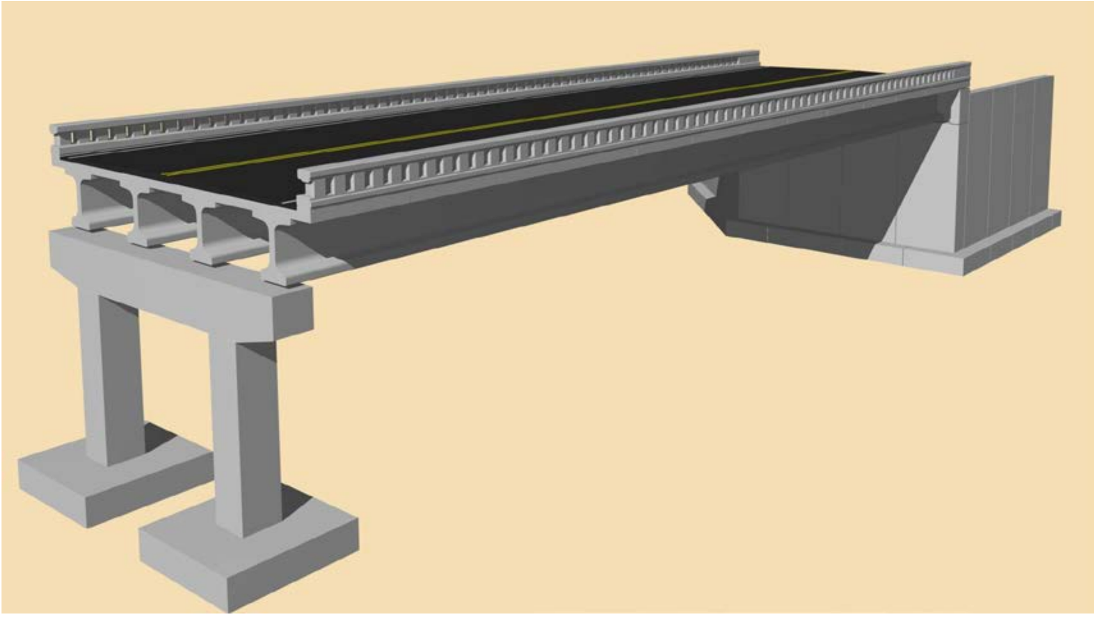


Fig. 1.6 End portion of a deck girder bridge system

It can be said that, in the way girder spacing, number of girder, cross section of girder, and deck thickness were related to each other in the previous problem, in the same way, in this problem, pier to pier spacing (span length), number of pier, pier dimension, pier depth (according to river bottom profile), girder cross section are inter-related.

Again, there is footing below every pier to transfer the load safely to underneath soil. There are various types of footing and each type has design variables of its own. The design of footing depends on load coming to it and the soil below the footing across the river bed.

So, here are the already stated inter-related design variables of beam/deck-girder bridge system:

- Deck/slab thickness
- Beam/Girder dimension(cross-section)
- Beam/Girder Spacing (i.e. no. of girder)
- Pier to pier Spacing (i.e. span length)
- Foundation depth
- Foundation type/dimension

There can be thousands of combination of these design variables which will perform the desired strength and serviceability criteria pretty satisfactorily. But which option or combination is the most efficient in terms of cost is the point of interest.

Traditional manual iterative approaches based on experience / heuristics of the designers were very poor to handle such complex problems. Again, traditional computer aided optimum design (using optimization technique) that have been used so far were not efficient enough to deal with this very highly complex problem and could not find the global minimum point for cost.

1.3 Background and Present State of Problem

Prestressed Concrete (PC) I-girder Bridge systems are widely used bridge system for short to medium span (20m to 60m) highway bridges due to its moderate self weight, structural efficiency, ease of fabrication, fast construction, low initial cost, long life expectancy, low maintenance and simple deck removal & replacement etc. (PCI, 2003). In order to compete with steel bridge systems, the design of PC I-girder Bridge must lead to the most economical use of the materials (PCI, 1999). Large numbers of design variables are involved in the design process of the present bridge system. All the variables are related to each other leading to numerous alternative feasible designs. In conventional design, bridge engineers follow an iterative procedure to design the prestressed I-girder bridge structure. Most design offices do not advocate realistic estimate of material costs, just only satisfy all the specifications set forth by design codes. So there is no attempt to reach the best design that will yield minimum cost, weight or volume.

A global optimization algorithm named EVOP (Evolutionary Operation) (Ghani, 1989) was used in determining the most cost-efficient design of 30m, 40m and 50m long post-tensioned pre-stressed concrete I-girder bridge system (Rana, 2010). EVOP is capable of locating directly with high probability the global minimum. To formulate the optimization problem a computer program was developed in C++. The optimization approach was applied on a real life project (Teesta Bridge, Bangladesh) and a 30% cost efficient design was found.

In the thesis titled "Cost Optimization of Post-Tensioned Pre-stressed concrete I-Girder Bridge System" (Rana, 2010), the author optimized a simply supported post-tensioned pre-stressed concrete I-girder of bridge. Next step to even more cost-effectiveness constitutes considering the girders to be continuous. That is, taking the number of span into consideration as a design variable and also taking into account the difference in design of piers and footings in case of corresponding number of spans.

1.4 Objectives of the Present Study

Ahsan et al. (2012) has recently demonstrated successful use of a global optimization algorithm – EVOP developed by Ghani (1995) in cost optimum design of simply supported post-tensioned I-girder of bridges. Simple span systems, however, can lead to leakage through the deck and deterioration of beam-ends, bearings and the substructure. When beams are made continuous, structural efficiency and long-term performance can be significantly improved (PCI, 2003). Rana et al. (2013) adopted EVOP to optimize two-span post-tensioned bridge which was made continuous for superimposed dead loads and live loads using deck reinforcement. The focus of the present research is using EVOP effectively in handling optimization problems of prestressed bridge structures made continuous by post-tensioning which by resisting the deck weight can significantly improve the structural performance of longer span bridges.

Hence, objective of the present study is cost minimization of a two-span continuous post-tensioned prestressed concrete I-girder of bridge by adopting optimization approach to obtain the optimum value of the following design variables:

(i) Cross-sectional dimensions of the components of the bridge superstructure (girder & deck slab) and
(ii) Prestressing tendon size (number of strands per tendon), number of tendons, tendon arrangement & layout, ordinary reinforcement in deck slab and girder.

1.5 Scope and Methodology of the Study

In the present study a cost optimum design approach of a two-span continuous post-tensioned PC I-girder of bridge is presented considering the cost of materials, labor, fabrication and installation. The bridge system consists of precast girders with cast-in situ reinforced concrete deck. A large number of design variables and constraints are considered for cost optimization of the bridge system. A global optimization algorithm named EVOP (Evolutionary Operation) (Ghani, 1995) is used which is capable of locating directly with high probability the global minimum. The optimization method solves the optimization problem and gives the optimum solutions.

In their study, Ahsan et al. (2012) and Rana et al. (2013) developed a computer program written in C++ language to formulate the optimization problem which has the following components:

(i) Mathematical expression required for the design and analysis of the bridge system,
(ii) An objective function (it describes the cost function of the bridge to be minimized)
(iii) Implicit constraints or design constraints (it describes the design or performance requirements of the bridge system)
(iv) Explicit constrains (it describes the upper and lower limit of design variables or parameters) and
(v) Input control parameters for the optimization method.

In the present study, the subroutines developed by Ahsan et al. (2012) and Rana et al. (2013) (written in C++ language) is updated for girder made continuous by post-tensioning. Considering the girders to be continuous makes the analysis part of the problem complicated. The design moment and shear force at different sections for a specific span length and number of spans cannot be found now using simpler formula, rather it can be found using different methods for solving indeterminate structure. In this study, stiffness method is used to determine the design bending moment and shear force at any section of a continuous bridge having any number of spans with equal span length according to AASHTO specification. To make the basic stiffness method applicable to computer aided analysis, 'computer application of stiffness method' was used with required sequence of logic. To find out the live load and impact shear force and moment, it was necessary to construct influence lines for different sections. No general equation of influence line has been used; rather the coordinate values of different points of the influence line are determined using the basic stiffness concept. It was necessary to consider both primary and secondary moment due to prestress for the post-tensioned continuous beam. Two design codes, namely American Association of State Highway and Transportation Officials (AASHTO)-2007 for highway bridges and Precast/ Prestressed Concrete Institute (PCI)- PCI bridge design manual were followed in design part. In the last step, the updated C++ subroutines was linked with the EVOP algorithm written in FORTRAN language to figure out the most cost optimum solution for the bridge I-girders under consideration and to study relations among various design variables and their effects on overall cost of the I-girders. Repeating the same process, optimum girder configurations of continuous post-tensioned bridge I-girders for different spans will be determined.

1.6 Organization of the Thesis

Apart from this chapter, the remainder of the thesis has been divided into six chapters.

Chapter 2 presents literature review concerning past research on the field of cost optimization of simply supported PC bridge structures and continuous PC bridge structures.

Chapter 3 presents the various design criteria that should be satisfied for the design of continuous PC I-Girder bridge structures.

Chapter 4 presents the information about the various features of an optimization method and brief description about the procedure of global optimization method, EVOP, which is adopted in this study.

Chapter 5 presents the formulation of optimization problem of the bridge and linking process of optimization problem to the optimization method to obtain the optimum solution.

Chapter 6 presents the optimized results and discussions of the bridge system.

Finally, **Chapter 7** presents major conclusions and recommendation for future scopes of study.

Chapter 2

Literature Review

2.1 Introduction

In the last three decades, much work has been done in structural optimization, in addition to considerable developments in mathematical optimization. Despite this, there has always been a gap between the progress of optimization theory and its application to the practice of bridge engineering. In 1994, Cohn and Dinovitzer (1994) estimated that the published record on structural optimization since 1960 can conservatively be placed at some 150 books and 2500 papers, the vast majority of which deal with theoretical aspects of optimization. Documentation in their comprehensive catalogue of published examples shows that very little work has been done in the area of optimizing concrete highway bridges.

2.2 Past Research on Optimization of Prestressed Concrete Beams

The objective of most of the papers published on optimization of PC structures is minimization of cost of the structures. For the optimization of PC structure the general cost function for prestressed concrete structures considered in the past research can be expressed in the following form:

$$C_m = C_{cb} + C_{sb} + C_{pb} + C_{fb} + C_{sbv} + C_{fib} \tag{2.1}$$

where, C_m is the total material cost, C_{cb} is the cost of concrete in the girder, C_{sb} is the cost of reinforcing steel, C_{pb} is cost of prestressing steel, C_{fb} is the cost of the formwork and C_{sbv} is the cost of shear steel;

Goble and Lapay (1971) minimized the cost of post-tensioned prestressed concrete T-section beams based on the ACI code (ACI, 1963) by using the gradient projection method (Arora, 1989). The cost function included the first four terms of Eq. (2.1). They stated that the optimum design seemed to be unaffected by the changes in the cost coefficients. However, subsequent researchers to be discussed later rebut this conclusion.

Naaman (1976) compared minimum cost designs with minimum weight designs for simply supported prestressed rectangular beams and one-way slabs based on the ACI code. The cost function included the first, third, and fourth terms of Eq. (2.1) and was optimized by a direct search technique. He concluded that the minimum weight and minimum cost solutions give approximately similar results only when the ratio of cost of concrete per cubic yard to the cost of prestressing steel per pound is more than 60. Otherwise, the minimum cost approach yields a more economical solution, and for ratios much smaller than 60 the cost optimization approach yields substantially more economical solutions. He also pointed out that for most projects in the US the aforementioned ratio is less than 60.

Cohn and MacRae (1984a) considered the minimum cost design of simply supported RC and partially or fully pre-tensioned and post-tensioned concrete beams of fixed cross-sectional geometry subjected to serviceability and ultimate limit state constraints including constraints on flexural strength, deflection, ductility, fatigue, cracking, and minimum reinforcement, based on the ACI code or the Canadian building code using the feasible conjugate-direction method (Kirsch, 1993). The beam can be of any cross-sectional shape subjected to distributed and concentrated loads. Their cost function is similar to Eq. (2.1). For the examples considered they concluded that for post-tensioned members partial prestressing appears to be more economical than complete prestressing for a prestressing-to-reinforcing steel cost ratio greater than 4. For pretensioned beams, on the other hand, complete pre-stressing seems to be the best solution. For partially prestressed concrete they also concluded that for a prestressing-to-reinforcing steel cost ratio in the range of 0.5 to 6, the optimal solutions vary a little. Cohn and MacRae (1984b) performed parametric studies on 240 simply supported, reinforced, partially, or completely pre- and post-tensioned prestressed concrete beams with different dimensions, depth-to-span ratios, and live load intensities. They concluded that, in general, RC beams are the most cost-effective at high depth-to-span ratios and low live load intensities. On the other hand, completely prestressed beams are the most cost-effective at low depth-to-span ratios and high live load intensities. For intermediate values, partial prestressing is the most cost-effective option.

Saouma and Murad (1984) presented the minimum cost design of simply supported, uniformly loaded, partially prestressed, I-shaped beams with unequal flanges subjected to the constraints of the 1977 ACI code. The optimization problem was formulated in terms of nine design variables: six geometrical variables plus areas of tensile, compressive, and prestressing steel. The constrained optimization problem was transformed to an unconstrained optimization problem using the interior penalty function method (Kirsch, 1993) and was solved by the quasi-Newton method. They found the optimum solutions for several beams with spans ranging from 6m to 42 m, assuming both cracked and un-cracked sections, and reported cost reductions in the range of 5% to 52 %. They also concluded that allowing cracking to occur does not reduce the cost by any significant measure.

Linear programming methods were used by Kirsch (1985, 1993, and 1997) to optimize indeterminate prestressed concrete beams with prismatic cross sections through a "bounding procedure". To simplify the problem, a two-level formulation was used to reduce the problem size and eliminate potential numerical difficulties encountered because of the fundamentally different nature of the design variables. The concrete dimensions were optimized in one level, and the tendon variables (prestressing force and layout coordinates) were determined in another level. As a first step, a lower bound on the concrete volume was established without evaluating the tendon variables. The corresponding minimum prestressing force was an upper bound. Similarly, a lower bound on the prestressing force was determined by assuming the maximum concrete dimensions. Based on the two bounding solutions, a lower bound on the objective function was evaluated. The best of the bounding solutions was first checked for optimality. If necessary, the search for the optimum was then continued in the reduced space of the concrete variables using a feasible directions technique. For any assumed concrete dimensions, a reduced linear programming problem was solved. The process was repeated until the optimum was reached.

Lounis and Cohn (1993a) presented a prestressed I-beam cost optimization method for individual bridge components using continuous design variables. They first found the maximum feasible girder spacing for each of the available precast girder shapes and then minimized the prestressed and nonprestressed reinforcement in the I-beams and deck.

Lounis and Cohn (1993b) presented a multi-objective optimization formulation for minimizing the cost and maximizing the initial camber of post-tensioned floor slabs with serviceability and ultimate limit state constraints of the ACI code. The cost objective function was chosen as the primary objective and the camber objective function is transformed into a constraint with specified lower and upper bounds. The resulting single optimization problem was then solved by the projected Lagrangian method. The cost function for the slab included only the first and third terms of Eq. (2.1).

Khaleel and Itani (1993) presented the minimum cost design of simply supported partially prestressed concrete unsymmetrical I-shaped girders as per ACI Building code. The objective function was similar to Eq. (2.1). The sequential quadratic programming method was used to solve the nonlinear optimization problem assuming both cracked and uncracked sections. They concluded that an increase in the concrete strength does not reduce the optimum cost significantly, and higher strength in prestressing steel reduces the optimum cost to a certain extent. They claimed that some amount of reinforcing steel facilitates the development of cracking in the concrete, which reduces the cost of materials and improves ductility.

2.3 Past Research on Cost Optimization of Simply Supported Prestressed Concrete Bridge Structures

Torres et al. (1966) presented the minimum cost design of prestressed concrete highway bridges subjected to AASHTO loading by using a piecewise LP method. The independent design variables were the number and depth of girders, prestressing force, and tendon eccentricity. They further defined dependent design variables as the spacing of girders, tendon cross sectional area, initial prestress, and the slab thickness and reinforcement. They claim their cost function includes the costs of transportation, erection, and bearings in addition to the material costs of concrete and steel, but do not give any detail. They presented results for bridges with spans ranging from 20 ft to 110 ft (6.1m to 33.5 m) and with widths of 25 ft (7.6 m) and 50 ft (15.2 m).

Using integer programming, Jones (1985) formulated the minimum cost design of precast, prestressed concrete simply supported box girders used in a multi-beam highway bridge and subjected to the AASHTO (1977) loading assuming that the cross-sectional geometry and the grid work of strands are given and fixed. The design variables are the concrete strength, and the number, location and draping of strands (moving the strands up at the end of the beam). The constraints used were release and service load stresses, ultimate moment capacity, cracking moment capacity, and release camber. The cost function included only the first and third terms of Eq. (2.1).

Yu et al. (1986) presented the minimum cost design of a prestressed concrete box bridge girder used in a balanced cantilever bridge (consisting of two end cantilever and overhang spans and one middle simple span) based on the British code and using general geometric programming (Beightler and Phillips, 1976). The cost function included the material costs of concrete, prestressing steel, and the metal formwork. They included the labor cost of the metal formwork, roughly as 1.5 times the cost of the material for the formwork. The design variables were the prestressing forces, the eccentricities, and the girder depths for all spans.

Cohn and Lounis (1994) applied the above three-level cost optimization approach to multi-objective optimization of partially and fully prestressed concrete highway bridges with span lengths of 10m to 15m and widths of 8m to 16 m. Their objective functions included the minimum superstructure cost, minimum weight of prestressing steel, minimum volume of concrete, maximum girder spacing, minimum superstructure depth, maximum span-to-depth ratio, maximum feasible span length, and minimum superstructure camber. For a four-lane 20m length single-span bridge, they concluded that the voided slab and the precast I-girder systems were more economical than the solid slab and one- and two-cell box girders. Lounis and Cohn (1995a) also concluded that voided slab decks are more economical than box girders for short spans (less that 20 m) and wide decks (greater than 12 m), and single-cell box girders were more economical for medium spans (more than 20 m) and narrow decks (less than 12 m). The single-cell box girder, however, resulted in the deepest superstructure, which might be a drawback when there was restriction on the depth of the deck. Multi-criteria cost optimization of bridge structures was further discussed by Lounis and Cohn (1995b, 1996).

Fereig (1985) linearized the problem of prestressed concrete design optimization to determine the adequacy of a given concrete section and the minimum necessary prestressing force. He developed preliminary design charts that could be used to determine the required prestressing force for a given pretensioned, simply supported Canadian Precast–Prestressed Concrete Institute (CPCI) bridge girder, for any given span length and girder spacing.

Fereig (1996) presented the minimum cost preliminary design of single span bridge structures consisting of cast-in-place RC deck and girders based on the AASHTO code (AASHTO, 1992). The author linearized the problem by approximating the nonlinear constraints by straight lines and solves the resulting linear problem by the Simplex method. The author concluded that 'it is always more economical to space the girder at the maximum practical spacing'. Fereig (1999) compared the required prestressing forces obtained in his latter study with those that would be obtained using concrete with cylinder strength of 69 MPa. It was found that using the higher concrete strength allowed a reduction in the prestressing force from 4 to 12% depending on the girder spacing and span for the example considered.

Ahsan et al. (2012)has recently demonstrated successful use of a global optimization algorithm – EVOP developed by Ghani in cost optimum design of simply supported post-tensioned I-girder of bridges. Rana et al. (2013) adopted EVOP to optimize two-span post-tensioned bridge which was made continuous for superimposed dead loads and live loads using deck reinforcement.

2.4 Past Research on Cost Optimization of Continuous Prestressed Concrete Bridge Structures

Kirsch (1972) presented the minimum cost design of continuous two-span prestressed concrete beams subjected to constraints on the stresses, pre- stressing force, and the vertical coordinates of the tendon by linearizing the nonlinear optimization problem approximately and solving the reduced linear problem by the linear programming (LP) method. His cost function included only the first and third terms of Eq. (2.1). Kirsch (1973) extended this work to prestressed concrete slabs.

Cohn and Lounis (1993a) presented the minimum cost design of partially and fully prestressed concrete continuous beams and one-way slabs. The optimization was based on the limit state design and an optimization method named projected Lagrangian algorithm. They simultaneously satisfied both collapse and serviceability limit state criteria based on the ACI code. The material nonlinearity was idealized by an elastoplastic constitutive relationship. A constant prestressing force and prestressing losses were assumed. Their cost function included the first three terms of Eq. (2.1). They reported that the total cost decreases with the increase in the allowable tensile stress.

Lounis and Cohn (1993a) presented the minimum cost design of short and medium span highway bridges consisting of RC slabs on precast, post-tensioned, prestressed concrete I-girders satisfying the serviceability and ultimate limit state constraints of the Ontario Highway Bridge Design Code (OHBDC, 1983). They used a three-level optimization approach. In the first level they dealt with the optimization of the bridge components including dimensions of the girder cross-sections, slab thickness, amounts of reinforcing and prestressing steel, and tendon eccentricities by the projected Lagrangian method. In the second level, they considered the optimization of the longitudinal layout such as the number of spans, restraint type and span length ratios and transverse layout such as the number of girders and slab overhang length. In the third level, they considered various structural systems such as solid or voided slabs on precast I- or box girders. They used a sieve-search technique (Kirsch, 1993) for the second and third levels of optimization. Their cost function included the material costs of concrete, reinforcement, and connections at piers. They also included the costs of fabrication, transportation, and erection of girders assuming a constant value per length of the girder. They concluded by optimizing a complete set of bridge system resulting in a more economical structure than optimizing the individual components of the bridge. Based on their optimization studies they recommended simply supported girders for prestressed concrete bridges up to 27m (89 ft) long, two-span continuous girders for span lengths from 28m (92 ft) to 44m (144 ft), three-span continuous girders for span lengths of 55m (180 ft) to 100m (328 ft), and two- span or three-span continuous girders for an intermediate range of 44m (144 ft) to 55m (180 ft).

Han et al. (1995) discussed the minimum cost design of partially prestressed concrete rectangular, and T-shape beams based on the Australian code using the discretized continuum-type optimality criteria (DCOC) method. The cost function included the first four terms of Eq. (2.1). They concluded that for a simply supported beam, a T-shape is more economical than a rectangular section. Han et al. (1996) used the DCOC method to minimize the cost of continuous, partially prestressed and singly reinforced T-beams with constant cross-sections within each span. A three-span and a four-span continuous beam example were also presented.

Rana et al. (2013) adopted EVOP to optimize two-span post-tensioned bridge which was made continuous for superimposed dead loads and live loads using deck reinforcement.

2.5 Concluding Remarks

The great majority of papers on cost optimization of prestressed concrete structures include the material costs of concrete, steel, and formwork. Some researchers ignore the cost of the formwork. However, this cost is significant and should not be ignored. Other costs such as the cost of labor, fabrication and installation are often ignored. Most of the studies on prestressed concrete bridge structures, except the work of Ahsan et al. (2012) and Rana et al. (2013), either minimized the cost of individual components only or used standard AASHTO sections instead of considering cross-sectional dimensions as design variables or considered the cost of materials only. Most the studies considered prestressing strands to be located in a fixed position to obtain eccentricity which is not practical and the lump sum value (a fixed percentage of initial presress) of prestress losses. Only in their works, Ahsan et al. (2012) and Rana et al. (2013) considered variable location of prestressing strands and actual value of prestress loss. Most of the studies deal with optimization of pre-tensioned I-girder bridge systems. None of these studies, except the work of Ahsan et al. (2012) and Rana et al. (2013), deals with total cost optimization of the post-tensioned I-girder bridge systems considering all cross-sectional dimensions, prestressing tendons layout as design variables and also cost of materials including fabrication and installation. Rana et al. (2013) adopted EVOP to optimize two-span post-tensioned bridge which was made continuous for superimposed dead loads and live loads using deck reinforcement. The focus of the present research is using EVOP in handling optimization problems of prestressed bridge structures made continuous by post-tensioning which by resisting the deck weight can significantly improve the structural performance of longer span bridges.

CHAPTER 3

CONTINUOUS PRESTRESSED CONCRETE BRIDGE DESIGN

3.1 Introduction

Prestressing can be defined as the application of pre-determined force or moment to a structural member in such a manner that the combined internal stresses in the member resulting from this force or moment and any anticipated condition of external loading will be confined within specific limits. Thus prestressing refers to the permanent internal stress in a structure to improve performance by reducing the effect of external forces. The compression performance of concrete is strong but its tension performance is weak. The main idea of prestressing concrete is to counteract the tension stresses that are induced by external forces. For instance, prestressing wire placed eccentrically, the force in tendon produces an axial compression and hogging moment in the beam. While under service loads the same beam will develop sagging moments. Thus, it is possible to have the entire section in compression when service loads are imposed on the beam. This is the main advantage of prestressed concrete. It is well known that reinforced concrete cracks in tension. But there is no cracking in fully prestressed concrete since the entire section is in compression. Thus, it can be said that prestress provides a means for efficient usage of the concrete cross-section in resisting the external loads.

3.2 Reinforced Concrete versus Prestressed Concrete

Both reinforced concrete (RC) and prestressed concrete (PC) consist of two materials, concrete and steel. But high strength concrete and steel are used in prestressed concrete. Although they employ the same material, their structural behavior is quite different. In reinforced concrete structures, steel is an integral part and resists force of tension which concrete cannot resist. The tension force develops in the steel when the concrete begin to crack and the strains of concrete are transferred to steel through bond. The stress in steel varies with the bending moment. The stress in steel should be limited in order to prevent excessive crack of concrete. In fact the steel acts as a tension flange of a beam. In prestressed concrete, on the other hand the steel is used primarily for inducing a prestress in concrete. If this prestress could be induced by other means, there is little need of steel. The stress in steel does not depend on the strain in concrete. There is no need to limit the stress in steel in order to control cracking of concrete. The steel does not act as a tension flange of a beam. The behaviors of RC and PC flexural members are illustrated using Figure 3.1 and Figure 3.2 respectively.

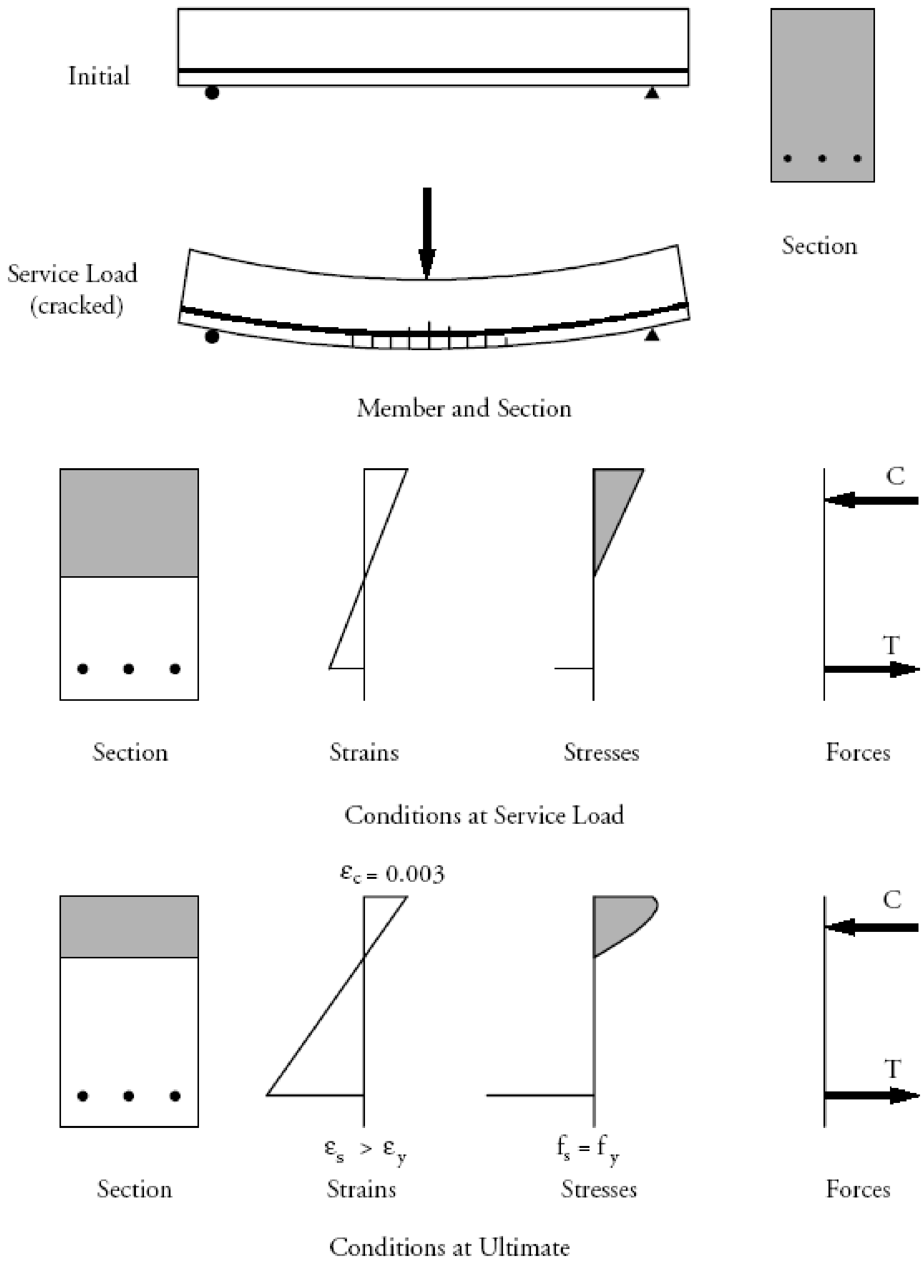


Figure 3.1 Behaviors of reinforced concrete members (PCI 2003)

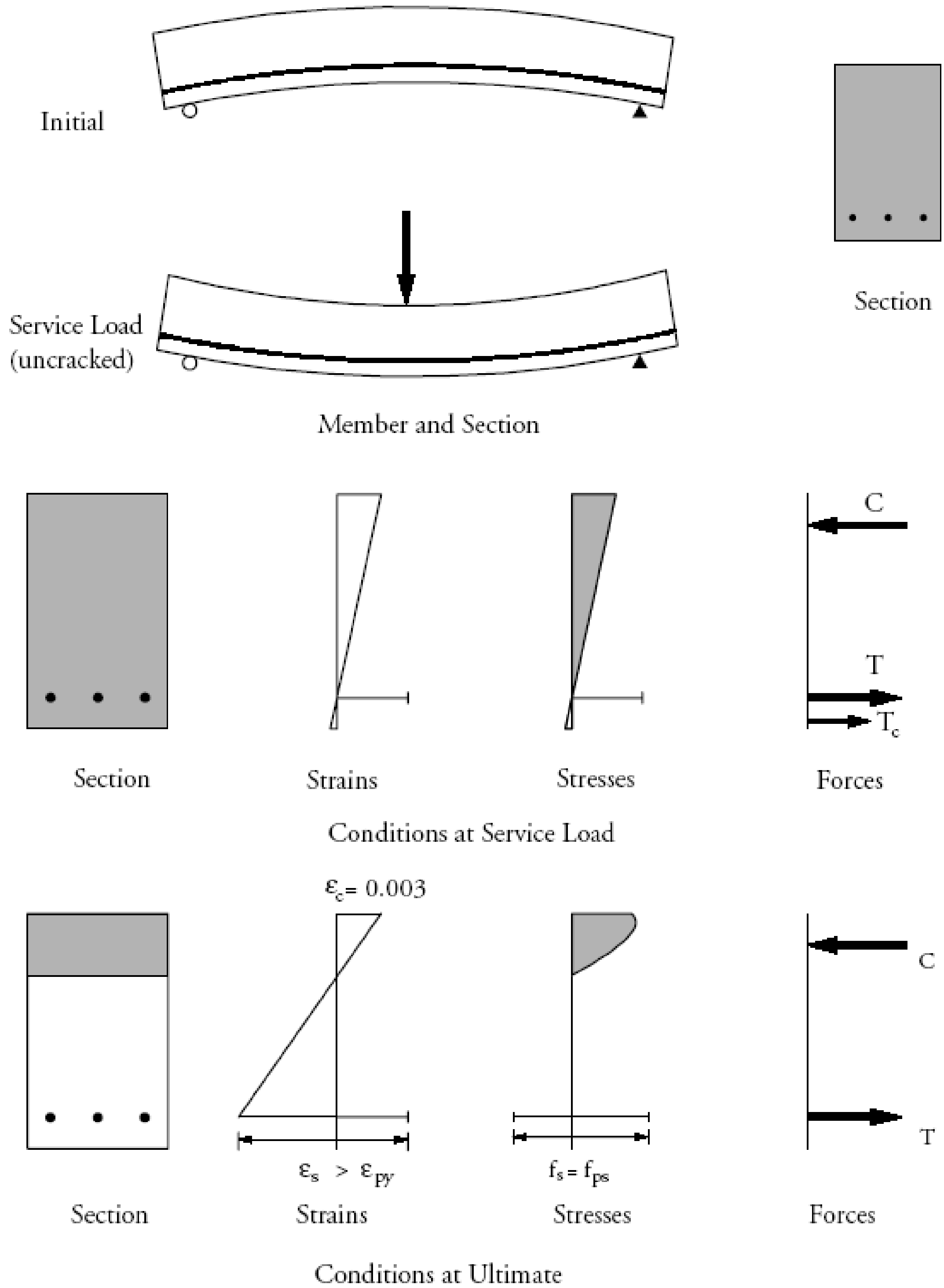


Figure 3.2 Behaviors of prestressed concrete members (PCI 2003)

Figure 3.1 shows the conditions in a reinforced concrete member that has mild steel reinforcement and no prestressing. Under service load conditions, concrete on the tension side of the neutral axis is assumed to be cracked. Only concrete on the compression side is effective in resisting loads. In comparison, a prestressed concrete member is normally designed to remain uncracked under service loads (Figure 3.2). Since the full cross-section is effective, the prestressed member is much stiffer than a conventionally reinforced concrete member resulting in reduced deflection. No unsightly cracks are expected to be seen. Reinforcement is better protected against corrosion. Fatigue of strand due to repeated truck loading is generally not a design issue when the concrete surrounding the strands is not allowed to crack. At ultimate load conditions, conventionally reinforced concrete and prestressed concrete behave similarly. However, due to the lower strength of mild bars, a larger steel quantity is needed to achieve the same strength as a prestressed member. This increases the member material costs for a conventionally reinforced member.

3.3 Advantages and Disadvantages of Prestressed Concrete

The most important feature of prestressed concrete is that it is free of cracks under working loads and it enables the entire concrete section to take part in resisting moments. Due to no-crack condition in the member, corrosion of steel is avoided when the structure is exposed to weather condition. The behavior of prestressed concrete is more predictable than ordinary reinforced concrete in several aspects. Once concrete cracks, the behavior of reinforced concrete becomes quite complex. Since there is no cracking in prestressed concrete, its behavior can be explained on a more rational basis. In prestressed concrete structures, sections are much smaller than that of the corresponding reinforced concrete structure. This is due to the fact that dead load moments are counterbalanced by the prestressing moment resulting from prestressing forces and shear resisting capacity of such section is also increased under prestressing. The reduced self-weight of the structure contribute to further reduction of material for foundation elements. Other feature of prestressed concrete is its increased quality to resist impact, high fatigue resistance and increased live load carrying capacity. Prestressed concrete is most useful in constructing liquid containing structures and nuclear plant where no leakage is acceptable and also used in long span bridges and roof systems due to its reduced dead load. On the other hand, prestressed concrete also exhibit certain disadvantages. Some of the disadvantages of prestressed concrete construction are:

(i) It requires high strength concrete that may not be easy to produce
(ii) It uses high strength steel, which might not be locally available
(iii) It requires end anchorage, end plates, complicated formwork
(iv) Labor cost may be greater, as it requires trained labor and
(v) It calls for better quality control

Generally, prestressed concrete construction is economical, as for example a decrease in member sections results in decreased design loads to obtain an economical substructure.

3.4 Prestressing Systems

The prestress in concrete structure is induced by either of the two processes. Pre tensioning and post tensioning. *Pre-tensioning* is accomplished by stressing wires, or strands called tendon to a pre-determined amount by stretching them between anchoring posts before placing the concrete. The concrete is then placed and the tendons become bonded to the concrete throughout their length. After the concrete has hardened, the tendon will be released from the anchoring posts. The tendon will tend to regain their original length by shortening and in this process they transfer a compressive stress to the concrete through bond. The tendons are usually stressed by hydraulic jacks. The other alternative is *post- tensioning.* In post-tensioning, the tendons are stressed after the concrete is cast and hardened to certain strength to withstand the prestressing force. The tendon are stressed and anchored at the end of the concrete section. Here, the tendons are either coated with grease or bituminous material or encased with flexible metal hose before placing in forms to prevent the tendons from bonding to the concrete during placing and curing of concrete. In the latter case, the metal hose is referred to as a sheath or duct and remains in the structure. After the tendons are stressed, the void between tendon and the sheath is filled with grout. Thus the tendons are bonded with concrete and corrosion is prevented. Bonded systems are more commonly used in bridges, both in the superstructure (the roadway) and in cable-stayed bridges, the cable-stays. There are post-tensioning applications in almost all facets of construction. Post-tensioning allows bridges to be built to very demanding geometry requirements, including complex curves, variable super-elevation and significant grade changes. In many cases, post-tensioning allows construction that would otherwise be impossible due to either site constraints or architectural requirements. The main difference between the two pre stressing system is:-

(i) Pre tensioning is mostly used for small member, whereas post- tensioning is used for larger spans.
(ii) Post- tensioned tendon can be placed in the structure with little difficulties in smooth curved profile. Pre-tensioned tendon can be used for curved profile but needs extensive plant facilities.
(iii) Pre-tensioning system has the disadvantage that the abutment used in anchoring the tendon has to be very strong and cannot be reused until the concrete in the member has sufficiently hardened and removed from bed.
(iv) Loss of prestress in pre-tensioning is more pronounced than that of post tensioning.

3.5 Anchorage Zone and Anchorage System

In post-tensioned girders, the prestressing force is transferred to girders in their end portions known as *anchorage zones*. The anchorage zone is geometrically defined as the volume of concrete through which the concentrated prestressing force at the anchorage device spreads transversely to a linear stress distribution across the entire cross section. The prestresssing force is transferred directly on the ends of the girder through bearing plate and anchors. As a result the ends are subjected to high bursting stresses. So it becomes necessary to increase the area of the girder's cross section in the end portion in order to accommodate the raised tendons, their anchorages, and the support bearing. This is accomplished by gradually increasing the web width to that of the flange; the resulting enlarged section is called *end block*. Design of the anchorage zone may be done by independently verified manufacturer's recommendations for minimum cover, spacing and edge distances for a particular anchorage device.

An anchorage system consists of a cast iron guide incorporated in the structures which distributes the tendon force into the concrete end block. On the guide sits the anchorage block, into which the strands are anchored by means of three-piece jaws, each locked into a tapered hole. The anchorage guide is provided with an accurate and robust method of fixing and the tendon, as it is provided with substantial shutter fixing holes and, at its opposite end, a firm screw type fixing for the sheath in addition it incorporates a large front access grout injection point which, by its careful transition design, is blockage free. All anchorage systems are designed to the same principles, varying only in size and numbers of strands. Freyssinet C range anchorage system for 15.2 mm diameter strands is shown in Figure 3.3(a), 3.3(b) and 3.3(c) and metal sheath is shown in Figure 3.4.

Figure 3.3(a) 13C15 Post-tensioning anchorage system (C range, Freyssinet Inc.)

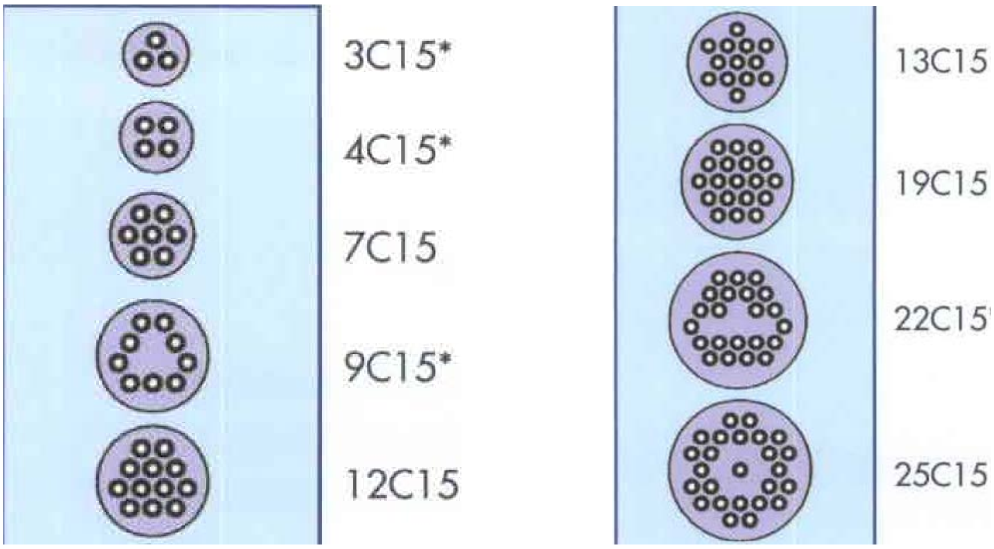
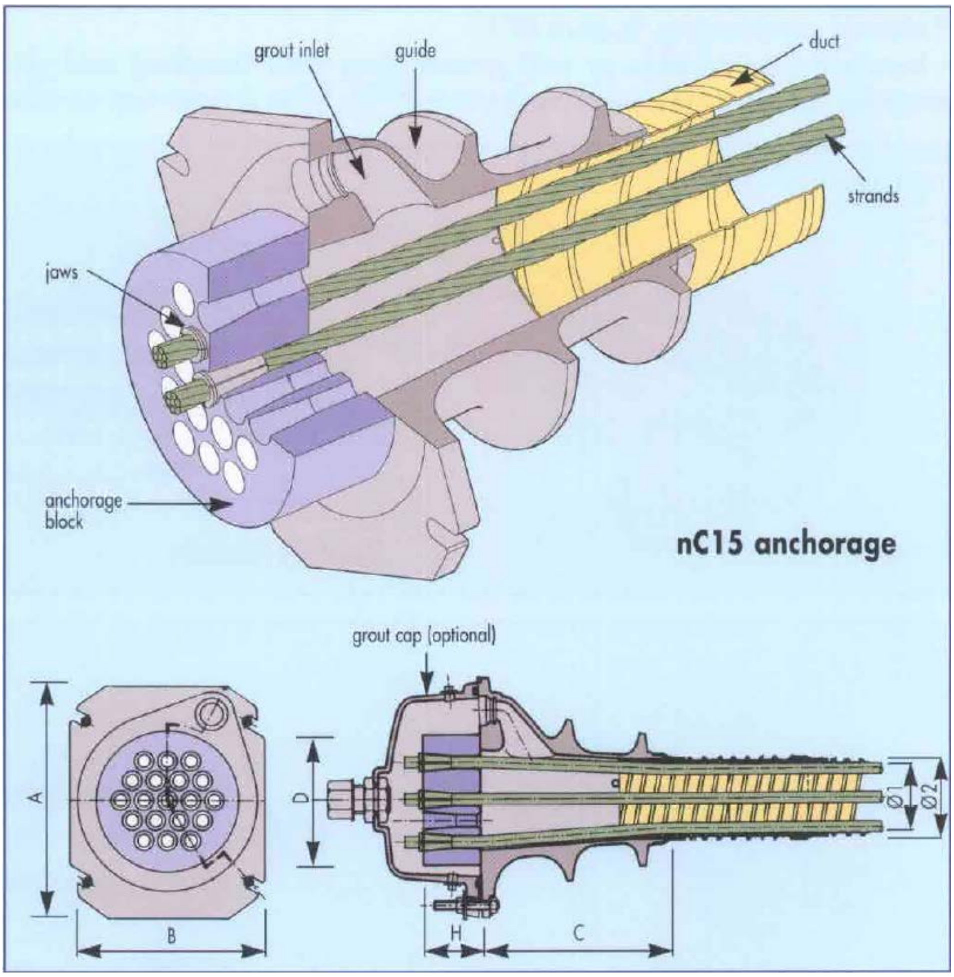


Figure 3.3(b) Range of anchorages (C range, Freyssinet Inc.)



Size	A	B	C	D	H	Ø1*	Ø2**
3C15	150	110	120	85	50	40	45
4C15	150	120	125	95	50	45	50
7C15	180	150	186	110	55	60	65
9C15	225	185	260	150	55	65	70
12C15	240	200	165	150	65	80	85
13C15	250	210	246	160	70	80	85
19C15	300	250	256	185	80	95	100
22C15	330	275	430	220	90	105	110
25C15	360	300	400	230	95	110	115

All dimensions are in mm

Figure 3.3(c) Freyssinet C range anchorage system (C range, Freyssinet Inc.)

Figure 3.4 Metal sheaths for providing duct in the girder

3.6 Prestress Losses

Loss of prestress is defined as the difference between the initial stresses in the strands (jacking stress), and the effective prestress in the member (at a time when concrete stresses are to be calculated). There are three types of losses in post-tensioned prestressed concrete, friction losses, short term (instantaneous) losses and long term (time dependent) losses. The short term losses are anchorage slip loss and the loss due to elastic shortening of concrete. The long term losses are loss due to creep of concrete, loss due to shrinkage of concrete and loss due to steel relaxation. These losses need to be accounted for before checking the adequacy of a girder section under the residual prestress. A number of methods are available to predict loss of prestress. They fall into three main categories, listed in order of increasing complexity:

(i) Lump sum estimate methods
(ii) Rational approximate methods and
(iii) Detailed time-dependent analyses

3.6.1 Frictional losses, L_{WC}

When a tendon is jacked from one or both ends the stress along the tendon decreases away from the jack due to the effects of friction. Frictional loss occurs only in post-tensioned member. The friction between tendons and the surrounding material is not small enough to be ignored. This loss may be considered partly to be due to length effect (wobble effect) and party to curvature effect. In straight elements, it occurs due to wobble effect and in curved ones; it occurs due to curvature and wobble effects.

If angle of curve is θ and F_1 is force on pulling end of the curve, then force F_2 on the other end of the curve is given as

$$F_2 = F_1 e^{-\mu\theta} \qquad (3.1)$$

Similarly, the relation between F_1 and F_2 due to length effect (wobble effect) is given as

$$F_2 = F_1 e^{-Kx} \tag{3.2}$$

The combined effect is

$$F_2 = F_1 e^{-(\mu\theta + Kx)} \tag{3.3}$$

So the loss in the force is,

$$F_2 - F_1 = F_1\left(1 - e^{-(\mu\theta + Kx)}\right) \tag{3.4}$$

Where,

x = length of a prestressing tendon from the jacking end to any point under consideration;

μ = coefficient of friction;

K = wobble friction coefficient per unit length of tendon;

θ = sum of the absolute values of angular change of post-tensioning tendon from jacking end to the point under investigation;

F_1 = Jacking force;

The value of μ and K for different type of cables can be read from Codes. The total loss for all tendons (number of tendons = N_T) may be expressed by the following equation:

$$L_{WC} = \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + KL)}\right) \tag{3.5}$$

3.6.2 Instantaneous losses

3.6.2.1 Anchorage loss, L_{ANC}

Anchor set loss of prestress occurs in the vicinity of the jacking end of post-tensioned members as the post-tensioning force is transferred from the jack to the anchorage block. During this process, the wedges move inward as they seat and grip the strand. This results in a loss of elongation and therefore force in the tendon. The value of the strand shortening generally referred to as anchor set, ΔL, varies from about 3 mm to 9mm with an average value of 6 mm. It depends on the anchorage hardware and jacking equipment. The anchor set loss is highest at the anchorage. It diminishes gradually due to friction effects as the distance from the anchorage increases. Anchor set loss can be calculated by Eq. (3.6).

$$L_{ANC} = \frac{A_s E_s}{L} \Delta \tag{3.6}$$

where

A_s = Area of prestressing steel

E_s = Modulus of elasticity of prestressing steel

Δ = Anchorage slip

3.6.2.2 Elastic shortening loss, L_{ES}

As the prestressed is transferred to the concrete the member itself shortens and the prestressing steel shorten with it. Therefore, there is a loss of prestress in the steel. In post tensioning, the tendons are not usually stretched simultaneously. Moreover, the first tendon that is stretched is shortening by subsequent stretching of all other tendon. Only the last tendon is not shortening by any subsequent stretching. An average value of strain change can be computed and equally applied to all tendons. The prestress loss due to elastic shortening in post-tensioned members is taken as the concrete stress at the centroid of the prestressing steel at transfer, f_{cgp}, multiplied by the ratio of the modulus of elasticities of the prestressing steel and the concrete at transfer.

$$L_{ES} = K_{es}\frac{E_s}{E_{ci}} f_{cgp\,1} A_s + K_{es}\frac{E_s}{E_{ci}} f_{cgp\,2} A_s =$$

$$K_{es}\frac{E_s}{E_{ci}}\left(\frac{F_o}{A_g} + \frac{F_o e1^2}{I} - \frac{M_{G1}\, e1}{I}\right) A_s + K_{es}\frac{E_s}{E_{ci}}\left(\frac{F_o}{A_g} + \frac{F_o e2^2}{I} - \frac{M_{G2}\, e2}{I}\right) A_s \tag{3.7}$$

where,

f_{cgp1} = sum of concrete stresses at the center of gravity of prestressing tendons due to the prestressing force at transfer and the self weight of the member at the sections of maximum positive moment.

f_{cgp2} = sum of concrete stresses at the center of gravity of prestressing tendons due to the prestressing force at transfer and the self weight of the member at the sections of maximum negative moment.

M_{G1}= Maximum positive moment due to self weight of the girder.

M_{G2}= Maximum negative moment due to self weight of the girder.

e_1= eccentricity at position of maximum positive moment.

e_2= eccentricity at position of maximum negative moment.

E_{ci} = modulus of elasticity of the concrete at transfer

Applying this equation requires estimating the force in the strands after transfer. Initial Estimate of F_o is unknown as yet loss for elastic shortening is undetermined. Let

$$F_o = F - L_{WC} - L_{ANC} \tag{3.8}$$

With F_o from Eq. (3.8), L_{ES} is calculated from Eq. (3.7) and F_o is continuously updated using Eq. (3.9) until the updated value becomes equal to previous value.

$$F_o = F - L_{ES} - L_{WC} - L_{ANC} \tag{3.9}$$

3.6.3 Time-dependent losses

3.6.3.1 Loss due to shrinkage of concrete, L_{SH}

Shrinkage in concrete is a contraction due to drying and chemical changes. It is dependent on time and moisture condition but not on stress. The amount of shrinkage varies widely, depending on the individual conditions. For the propose of design, an average value of shrinkage strain would be about 0.0002 to 0.0006 for usual concrete mixtures employed in prestressed construction. Shrinkage of concrete is influenced by many factors but in this work the most important factors volumes to surface ratio, relative humidity and time from end of curing to application of prestress are considered in calculation of shrinkage losses. The equation for estimating loss due to shrinkage of concrete, L_{SH}, is roughly based on an ultimate concrete shrinkage strain of approximately –0.00042 and a modulus of elasticity of approximately 193 GPa for prestressing strands. According to AASHTO 2007 the expression for prestress loss due to shrinkage is a function of the average annual ambient relative humidity, R_H, and is given as for post-tensioned members.

$$L_{SH} = 0.8(117.3 - 1.03\ R_H)\ (MPa) \tag{3.10}$$

Where, R_H = the average annual ambient relative humidity (%). The average annual ambient relative humidity may be obtained from local weather statistics.

3.6.3.2 Loss due to creep of concrete, L_{CR}

Creep is the property of concrete by which it continues to deform with time under sustained load at unit stresses within the accepted elastic range. This inelastic deformation increase at decreasing rate during the time of loading and its magnitude may be several times larger than that of the short term elastic deformation. The strain due to creep varies with the magnitude of stress. It is a time dependent phenomena. Creep of concrete result in loss in steel stress. The expression for prestress losses due to creep is,

$$L_{CR} = 0.083f_{cgp} - 0.048f_{cds} \text{ (MPa)} \tag{3.11}$$

where,

f_{cgp} = concrete stress at the center of gravity of the prestressing steel at transfer

f_{cds} = change in concrete stress at center of gravity of prestressing steel due to permanent loads, except the load acting at the time the prestressing force is applied. Values of f_{cds} should be calculated at the same section or at sections for which f_{cgp} is calculated. The value of f_{cds} includes the effect of the weight of the diaphragm, slab and haunch, parapets, future wearing surface, utilities and any other permanent loads, other than the loads existing at transfer at the section under consideration, applied to the bridge.

$$f_{cgp} = \left(\frac{F_o}{A_{tf}} + \frac{F_o e1^2}{I} - \frac{M_{G1}\, e1}{I}\right) + \left(\frac{F_o}{A_{tf}} + \frac{F_o e2^2}{I} - \frac{M_{G2}\, e2}{I}\right) \tag{3.12}$$

$$f_{cds} = \frac{(M_{P1} - M_{G1})\, e1}{I} + \frac{M_{C1} e_{c1}}{I_c} + \frac{(M_{P2} - M_{G2})\, e2}{I} + \frac{M_{C2} e_{c2}}{I_c} \tag{3.13}$$

where, M_{G1} = Moment due to girder self weight at position of maximum positive moment; M_{G2} = Moment due to girder self weight at position of maximum negative moment; e_1 = eccentricity at position of maximum positive moment; e_2 = eccentricity at position of maximum negative moment; M_{P1} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum positive moment; M_{P2} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum negative moment; M_{C1} = Composite dead load moment due to future wearing coarse and curb self weight at position of maximum positive moment; M_{C2} = Composite dead load moment due to future wearing coarse and curb self weight at position of maximum negative moment; I_c = moment of inertia of composite section; e_{c1} = Eccentricity of tendons in composite section at position of maximum positive moment; e_{c2} = Eccentricity of tendons in composite section at position of maximum negative moment;

3.6.3.3 Loss due to relaxation of steel, L_{SR}

Relaxation is assumed to find the loss of stress in steel under nearly constant strain at constant temperature. It is similar to creep of concrete. Loss due to relaxation varies widely for different steels and its magnitude may be supplied by the steel manufactures based on test data. This loss is generally of the order of 2% to 8% of the initial steel stress. Losses due to relaxation should be based on approved test data. If test data is not available, the loss may be assumed to be 2.5 % of initial stress. AASHTO 2007 provides Eq.3.14 to estimate relaxation after transfer for post-tensioned members with stress-relieved or low relaxation strands. The expression for prestress losses due to steel relaxation is,

$$L_{SR} = 34.48-0.00069L_{ES}-0.000345\,(L_{SH}+ L_{CR})\ (\text{MPa}) \tag{3.14}$$

3.7 Designs for Flexure

The design of a prestressed concrete member in flexure normally involve selection and proportioning of a concrete section, determination of the amount of prestressing force and eccentricity for given section. The design is done based on strength (load factor design) and on behavior at service condition (allowable stress design) at all load stages that may be critical during the life of the structure from the time the prestressing force is applied. In the next section the allowable stress design (Elastic design) and the load factor design (ultimate design) of a section for flexure is briefly described.

3.7.1 Allowable stress design (ASD)

This design method ensures stress in concrete not to exceed the allowable stress value both at transfer and under service loads. The member is normally designed to remain un-cracked under service loads. Consequently, it is assumed that the member can be analyzed as homogeneous and elastic under service loads that all assumptions of simple bending theory can be applied.

Stresses in concrete due to prestress and load:

Stresses in concrete due to prestress are usually computed by an elastic theory. For a prestressing force *F* applied to a concrete section with an eccentricity *e*, the prestress is resolved into two components a concentric force *F* through the centroid and a moment $F \times e$. Using elastic formula the stress at top extreme fiber or bottom extreme fiber due to axial compression, *F* and moment, $M = Fe$ is given by

$$f = -\frac{F}{A} \pm \frac{F \times e}{S} \tag{3.15}$$

where, S = section modulus for top or bottom fiber.

For post-tensioned member before being bonded, for the values of *A* and *S,* the net concrete section has to be used. After the steel is bonded, these should be based on the transformed section. The stresses produced by any external load as well as own weight of the member is given by elastic theory as:

$$f = \frac{M}{S} \tag{3.16}$$

The resulting stresses due to prestress and loads are as follows:

$$f = -\frac{F}{A} \pm \frac{F \times e}{S} \pm \frac{M}{S} \tag{3.17}$$

3.7.2 Ultimate strength design

This method of design ensures that the section must have sufficient strength (resisting moment) under factored loads. The nominal strength of the member is calculated, based on the knowledge of member and material behavior. The nominal strength is modified by a strength reduction factor Φ, less than unity, to obtain the design strength. The required strength is an overload stage which is found by applying load factors γ, greater than unity, to the loads actually expected.

Stress in the prestressed steel at flexural failure:

For bonded members with prestressing only, the average stress in prestressing reinforcement at ultimate load, f_{su}, is:

$$f_{su} = f_{pu}\left(1 - \frac{\gamma^*}{\beta_1}\rho\frac{f_s}{f_c'}\right) \tag{3.18}$$

where,

γ^* = factor for type of pre stressing steel = 0.28 (for low relaxation steel)

β_1 = ratio of depth of equivalent compression zone to depth of prestressing steel

f'_c = compressive strength of concrete

for $f_c' \leq 27.6$ MPa, $\beta_1 = 0.85$;

for 27.6 MPa $\leq f_c' \leq 55.2$ MPa, $\beta_1 = 0.85 - 0.00725(f_c'[\text{MPa}] - 27.6)$;

for $f_c' \geq 55.2$ MPa, $\beta_1 = 0.65$

$\rho = \frac{A_s}{bd}$

b = effective flange width

d = effective depth for flexure

Design flexural strength:

With the stress in the prestressed tensile steel when the member fails in flexure, the design flexural strength is calculated as follows:

The depth of equivalent rectangular stress block, assuming a rectangular section, is computed by:

$$a = \frac{A_s f_{su}}{0.85 f_c' b} \tag{3.19}$$

For sections with prestressing strand only and the depth of the equivalent rectangular stress block less than the flange thickness (t_f), the design flexural strength should be taken as:

$$\Phi M_n = \Phi\left\{A_s f_{su} d\left(1 - 0.6\frac{\rho f_{su}}{f_c'}\right)\right\} \tag{3.20}$$

For sections with prestressing strand only and the depth of the equivalent rectangular stress block greater than the flange thickness, the design flexural strength should be taken as:

$$\Phi M_n = \Phi\left\{A_{sw} f_{su} d\left(1 - 0.6\frac{A_{sw} f_{su}}{b_w d f_c'}\right)\right\} + 0.85 f_c (b - b_w) t_f\left(d - \frac{t_f}{2}\right) \tag{3.21}$$

where,

$A_{sw} = A_s - A_{sf}$

$A_{sf} = 0.85 f_c (b - b_w) \frac{t_f}{f_{su}}$

b_w = Web width

For negative moment region, the design flexural strength is calculated as follows:

The depth of equivalent rectangular stress block is computed by:

$$a = \frac{A_s\, f_{su}}{0.85 f_c^{'}\, bw} \tag{3.22}$$

The design flexural strength should be taken as:

$$\Phi M_n = \Phi\left\{A_s f_{su}\, d\left(1 - 0.6\frac{\rho f_{su}}{f_c^{'}}\right)\right\} \tag{3.23}$$

3.8 Ductility Limit

3.8.1 Maximum prestressing steel

According to the AASHTO Standard Specifications (AASHTO 2007), the prestressed concrete members must be designed so that the steel yields when the ultimate capacity is reached. Therefore, the maximum prestressing steel constraints for the composite section are given below:

$$\omega \leq \omega_u \tag{3.24}$$

where, ω = Reinforcement index; ω_u = Upper bound to Reinforcement index = $0.36\beta_1$. This constraint ensures the steel will yield as the ultimate capacity is approached. AASHTO gives the following formula for calculating reinforcement index (ω)

$$\omega = \frac{\rho f_{su}}{f_c^{'}} \tag{3.25}$$

3.8.2 Minimum prestressing steel

AASHTO limit the minimum value of prestressing steel to be used in prestressing concrete section. The prestressing steel in a section should be adequate to develop ultimate moment at critical section at least 1.2 times the cracking moment M_{cr}^*.

$\Phi M_n \geq 1.2\ M_{cr}^*$ (3.26)

At position of maximum positive moment,

$$M_{cr}^* = (f_r + f_{pe})S_{bc} - M_{P1}\left(\frac{S_{bc}}{S_b} - 1\right) \quad (3.27)$$

$$f_{pe} = \frac{F_e}{A_{tf}} + \frac{F_e e_{c1}}{S_b} \quad (3.28)$$

where, f_r = modulus of rupture; f_{pe}= compressive stress in concrete due to effective prestress forces only (after allowance for all prestress losses) at extreme fiber of section where tensile stress is caused by externally applied load; S_b, S_{bc} = Section Modulus of bottom fiber of transformed precast & composite section respectively; e_{c1} = eccentricity of composite section at position of maximum positive moment; M_{P1} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum positive moment;

At position of maximum negative moment,

$$M_{cr}^* = (f_r + f_{pe})S_{tc} - M_{P2}\left(\frac{S_{tc}}{S_t} - 1\right) \quad (3.29)$$

$$f_{pe} = \frac{F_e}{A_{tf}} + \frac{F_e e_{c2}}{S_t} \quad (3.30)$$

where, f_r = modulus of rupture; f_{pe}= compressive stress in concrete due to effective prestress forces only (after allowance for all prestress losses) at extreme fiber of section where tensile stress is caused by externally applied load; S_t, S_{tc} = Section Modulus of top fiber of transformed precast & composite section respectively; e_{c2} = eccentricity of composite section at position of maximum negative moment; M_{P2} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum negative moment;

3.9 Design for Shear

The design and analysis of precast, prestressed concrete bridge members for vertical shear is presented in this section. Unlike flexural design, for which conditions at both service and factored load are evaluated, shear design is only evaluated for factored loads (strength limit state). The shear strength of a prestressed concrete member is taken as the s um of t he s hear s trength contributions by c oncrete a nd by t he w eb reinforcement. According t o AASHTO s pecification for de sign purposes t he relationship is written as,

$$V_u \leq \varphi\ (V_c + V_s) \tag{3.31}$$

where,

V_u = factored shear force at the section considered

V_c = the concrete contribution taken as lesser of flexural shear, V_{ci} and web shear, V_{cw}

V_s = shear carried by the steel.

The concrete contribution V_c, is taken as the shear required to produce shear cracking. Two types of shear cracking have been flexural shear and web shear as illustrated in Figure 3.5.

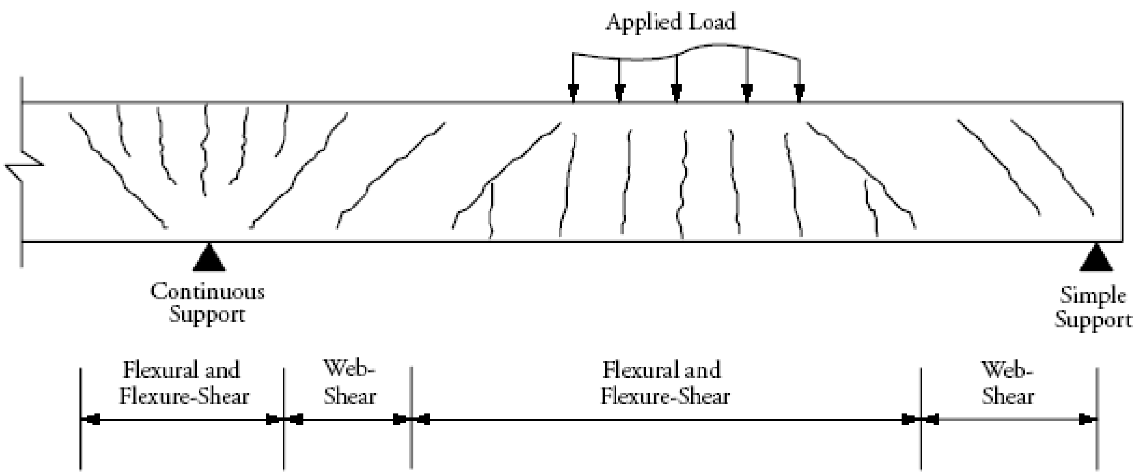


Figure 3.5 Types of cracking in prestressed concrete beams (PCI 2003)

Flexural s hear c rack do minate t he be havior of t he p ortion of t he gi rder w here hi gh flexural st resses co incide w ith si gnificant sh ear st resses. W eb sh ear c rack f orms i n regions of high shear and small flexural stress such as near the support of the simply supported beam. The shears that produce these two types of cracking are V_{ci} and V_{cw}. Therefore V_c is taken as lesser of the V_{ci} and V_{cw}. Procedures for computing these two shear capacity are presented below:

3.9.1 Flexure-shear, V_{ci}

A flexure-shear crack is initiated by a flexure crack forming at a distance d/2 from the section being considered. As the shear increases, the flexure crack inclines and becomes a shear crack with a horizontal projection equal to the distance d. V_{ci}, nominal shear strength provided by concrete when diagonal cracking results from combined shear and moment is calculated as follows:

$$V_{ci} = 0.05\sqrt{f_c}\, W_t d + V_d + \frac{V_i M_{cr}}{M_{max}} \leq 0.141\sqrt{f_c}\, W_t d \tag{3.32}$$

$$M_{cr} = S_{bc}\left(0.5\sqrt{f_c} + f_{pe} - f_d\right) \tag{3.33}$$

$$M_{max} = 1.3(M_D + 1.67\, M_{LL}) - M_D \tag{3.34}$$

$$V_i = 1.3(V_D + 1.67\, V_{LL}) - V_D \tag{3.35}$$

$$f_d = \frac{M_D}{S_{bc}} \tag{3.36}$$

where,

V_D = shear force at the section of investigation due to the unfactored dead load

M_{max} = maximum factored moment at the section due to externally applied loads

V_i = factored shear force at the section that occurs simultaneously with M_{max}

M_{cr} = moment due to external load required to crack the concrete at the critical section.

The term f_{pe} and f_d are the stresses at the extreme tension fiber due to the effective prestress forces only, after all losses, and due to the total unfactored dead load, respectively.

The AASHTO specifications state that V_{ci} need not be taken less than $0.141\sqrt{f_c}\, W_t d$ (kN) and that d need not be taken less than 0.8h, where h is the height of the section.

3.9.2 Web-shear, V_{cw}

V_{cw}, nominal shear strength provided by concrete when diagonal cracking results from excessive principal tensile stress in web is calculated as follows:

$$V_{cw} = \left(0.283\sqrt{f_c} + 0.3 f_{pc}\right) W_t d + V_p \tag{3.37}$$

$$V_p = \sum_{i=1}^{i=N_T} F_i \sin\theta_i \tag{3.38}$$

where,

V_p = the vertical component of the prestress force.

f_{pc} = stress that includes the effect of the prestress force after losses and the stresses due to any loads applied to the member as a non-composite section.

3.10 Design for Horizontal Interface Shear

Cast-in-place concrete decks designed to act compositely with precast concrete beams must be able to resist the horizontal shearing forces at the interface between the two elements. Design is carried out at various locations along the span, similar to vertical shear design. The Standard Specifications does not identify the location of the critical section. For convenience, it may be assumed to be the same location as the critical section for vertical shear. Other sections, generally at tenth-point intervals along the span, are also designed for composite-action shear. This may be necessary to ensure that adequate reinforcement is provided for horizontal shear because reinforcement for vertical shear, which is extended into the deck and used for horizontal shear reinforcement, may vary along the length of the member. Composite sections are designed for horizontal shear at the interface between the precast beam and deck using the equation:

$$V_u \le \varphi V_{nh} \tag{3.39}$$

Where,

V_u = factored shear force acting on the interface; φ = strength reduction factor for shear; V_{nh} = nominal shear capacity of the interface

The nominal shear capacity is obtained from one of the following conditions given as:

a) When the contact surface is intentionally roughened but minimum vertical ties are not provided:

$V_{nh} = 80 b_v d$

b) When minimum ties are provided but the contact surface is not intentionally roughened:

$V_{nh} = 80 b_v d$

c) When the contact surface is intentionally roughened to minimum amplitude of 1⁄4 in and minimum vertical ties are provided:

$V_{nh} = 350 b_v d$

d) When required area of ties, $A_v h$, exceeds the minimum area:

$V_{nh} = 330 b_v d + 0.40 A_v h f_y d/s$

For the above equations,

b_v = width of cross-section at the contact surface being investigated for horizontal shear

d = distance from extreme compression fiber to centroid of the prestressing force. As for vertical shear design, d need not be taken less than 0.80h.

s = maximum spacing not to exceed 4 times the least web width of support element, nor 24 in.

3.11 Design for Lateral Stability

Prestressed concrete members are generally stiff enough to prevent lateral buckling. However, during handling and transportation, support conditions may result in lateral displacements of the beam, thus producing lateral bending about the weak axis. For hanging beams, the tendency to roll is governed primarily by the properties of the beam. The equilibrium conditions for a hanging beam are shown in Figure 3.6 and Figure 3.7. When a beam hangs from lifting points, it may roll about an axis through the lifting points. The safety and stability of long beams subject to roll are dependent upon:

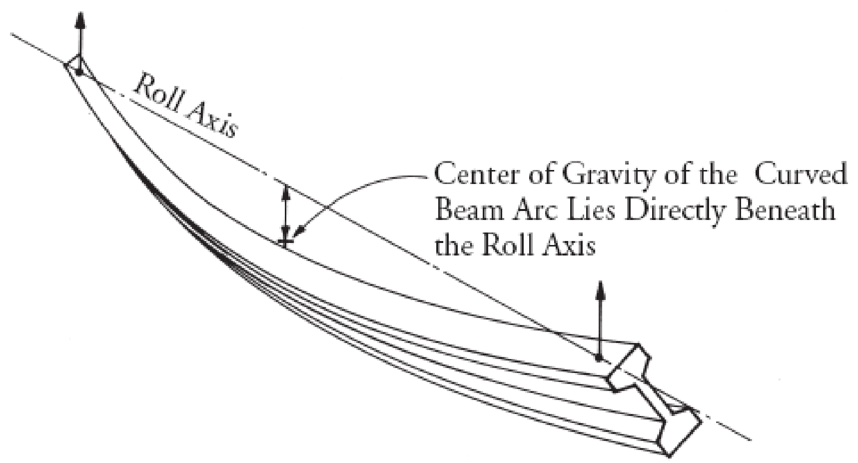


Figure 3.6 Perspective of a beam free to roll and deflect laterally (PCI 2003)

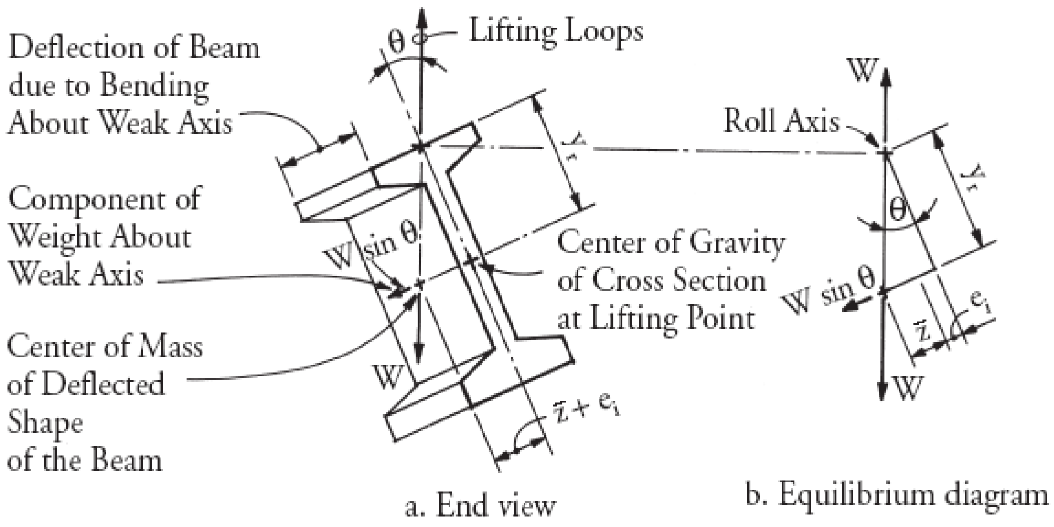


Figure 3.7 Equilibrium of beam in tilted position (PCI 2003)

Where,

e_i = the initial lateral eccentricity of the center of gravity with respect to the roll axis

y_r = the height of the roll axis above the center of gravity of the beam

z_o = the theoretical lateral deflection of the center of gravity of the beam, computed with the full weight applied as a lateral load, measured to the center of gravity of the deflected arc of the beam

θ_{max} = tilt angle at which cracking begins, based on tension at the top corner equal to the modulus of rupture.

For a beam with overall length, l, and equal overhangs of length, a, at each end:

$$\bar{z}_o = \frac{w}{12EI_g \ell}\left[0.1(\ell_1)^5 - a^2(\ell_1)^3 + 3a^4(\ell_1) + 1.2(a^5)\right] \tag{3.40}$$

Where $l_1 = l - 2a$

I_g = moment of inertia of beam about weak axis

For a beam with no overhangs, ($a = 0$, $l_1 = l$), and:

$$\bar{z}_o = \frac{w(\ell)^4}{120EI_g} \tag{3.41}$$

It is to note that, for a two span continuous post-tensioned girder (with each span having a length = l), l of aforementioned equations should be replaced with 2l.

The factor of safety against cracking, FS_c, is given by:

$$FS_c = \frac{1}{\frac{z_o}{y_r}+\frac{\theta_i}{\theta_{max}}} \geq 1.5 \tag{3.42}$$

where θ_i= the initial roll angle of a rigid beam.

It is recommended that e_i be based, as a minimum, on 1/4 in. plus one-half the PCI tolerance for sweep. The PCI sweep tolerance is 1/8 in. per 10 ft of member length. When cracking occurs, the lateral stiffness decreases and Z_o increases. Thus, failure may occur shortly after cracking as the tilt angle increases rapidly due to the loss of stiffness. Consequently, the factor of safety against failure, FS_f, is conservatively taken equal to FS_c. The necessary factor of safety can not be determined from scientific laws; it must be determined from experience. It is suggested to use a factor of safety of 1.0 against cracking, FS_c, and 1.5 against failure, FS_f.

3.12 Control of Deflection

Flexural members of bridge superstructures shall be designed to have adequate stiffness to limit deflections or any deformations that may adversely affect the strength or serviceability of the structure at service load plus impact. When making deflection computations the following criterion according to AASHTO Standard Specification is recommended.

Members having simple or continuous spans preferably should be designed so that the deflection due to service live load plus impact shall not exceed 1/800 of the span.

3.13 Composite Construction

Composite construction involves construction in which a precast member (usually a girder) acts in combination with cast-in-place concrete (usually a slab), that is poured at a later time and bonded to the member, with stirrups if necessary, to develop composite action. AASHTO (AASHTO 2007) defines a composite flexural member as one that "consists of precast and/or cast-in-place concrete elements constructed in separate placements but so interconnected that all elements respond to superimposed load as a unit".

3.14 Loads

3.14.1 General

There can be various types of load coming to a bridge structure. They are listed below:

1. Dead weight
2. Live load (vehicle load and pedestrian load)
3. Dynamic effect of live load
4. Wind load (directly on bridge, from vehicle, dynamic effect)
5. Earthquake load (static or dynamic)
6. Longitudinal forces (stopping vehicles)
7. Centrifugal forces (curved deck)
8. Thermal forces
9. Earth pressure
10. Buoyancy
11. Shrinkage stress
12. Rib shortening
13. Erection stresses
14. Ice loading
15. River current pressure

Among all these types of forces, in the present thesis, only the first three i.e. Dead Load, Live Load and Impact Load have been considered.

3.14.2 Dead Load

The dead loads on the bridge superstructure consist of self weight of the individual components (girder weight, deck slab weight), wearing surface on slab, sidewalks, curbs, railings and diaphragm etc.

While finding out dead load, load coming on the girder from its own weight, the weight of the deck and wearing surface that fall inside the girder's 'tributary area' should be considered (Figure 3.8).

In case of an exterior girder, extra load from the curb and railing has to be considered.

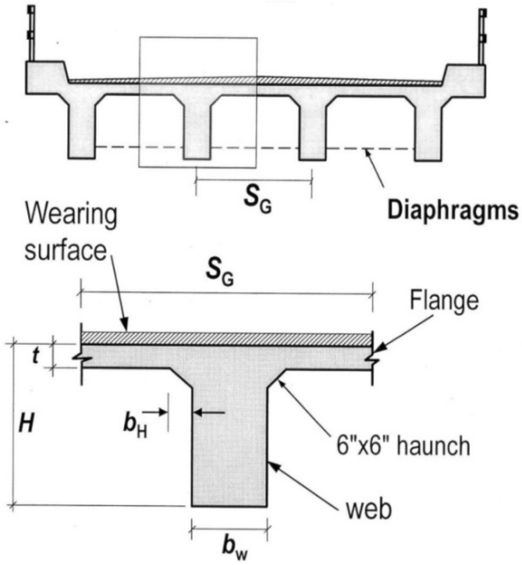


Figure 3.8 Tributary area of interior girder

3.14.3 Live Load

Vehicular live loading on the roadways of bridges or incidental structures, designated HL-93, shall consist of a combination of: i) Design truck or design tandem, and ii) Design lane load.

According to AASHTO LRFD HL 93 loading, each design lane should occupy either by the design truck or design tandem and lane load, which will be effective 3000mm transversely within a design lane. (AASHTO, 2007 3.6.1.2.1)

3.14.3.1 Truck Load

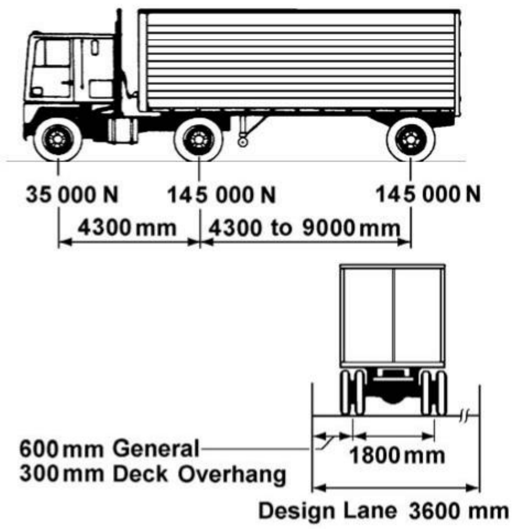


Figure 3.9 Truck load coming to bridge (AASHTO, 2007)

3.14.3.2 Tandem Load

The design tandem shall consist of a pair of 110,000 N axles spaced 1200 mm apart. The transverse spacing of wheels shall be taken as 1800 mm. (AASHTO, 2007, 3.6.1.2.3)

3.14.3.3 Lane Load

The design lane load shall consist of a load of 9.3 N/mm uniformly distributed in the longitudinal direction. Transversely, the design lane load shall be assumed to be uniformly distributed over a 3000 mm width. (AASHTO, 2007, 3.6.1.2.4)

3.14.3.4 Impact Load

Impact effect of live load is considered by increasing the live load effect by a certain factor.

Impact factor, $I = 50/(L+125) < 0.3$; Where, L = Loaded span in ft.

According to AASHTO 2007 explanation of the loaded length, the loaded length should be as follows:

(a) For roadway floors: the design span length.
(b) For transverse members, such as floor beams: the span length of member center to center of support.
(c) For computing truck load moments: the span length, or for cantilever arms the length from the moment center to the farthermost axle.
(d) For shear due to truck loads: the length of the loaded portion of span from the point under consideration to the far reaction; except, for cantilever arms, use a 30% impact factor.
(e) For continuous span: the length of span under consideration for positive moment, and the average of two adjacent loaded spans for negative moment.

3.14.3.5 Distribution Factor for Moment

As per AASHTO 2007, while calculating design moment, it should be multiplied by certain Distribution Factor. Value of Distribution Factor depends on several parameters like number of lane, span length of girder, deck slab thickness, lateral spacing of girders etc. For interior and exterior girder, AASHTO proposes different Distribution Factors.

For Interior Beam/ Girder

According to AASHTO 2007, Distribution Factors for moment for interior beam/ girder are as follows:

Table 3.1 Distribution Factor for Moment (Interior Girder) [AASHTO, 2007]

One Design Lane Loaded: $$0.06+\left(\frac{S}{4300}\right)^{0.4}\left(\frac{S}{L}\right)^{0.3}\left(\frac{K_g}{Lt_s^{\ 3}}\right)^{0.1}$$ Two or More Design Lanes Loaded: $$0.075+\left(\frac{S}{2900}\right)^{0.6}\left(\frac{S}{L}\right)^{0.2}\left(\frac{K_g}{Lt_s^{\ 3}}\right)^{0.1}$$	$1100 \le S \le 4900$ $110 \le t_s \le 300$ $6000 \le L \le 73\,000$ $N_b \ge 4$ $4\times10^9 \le K_g \le 3\times10^{12}$
use lesser of the values obtained from the equation above with $N_b = 3$ or the lever rule	$N_b = 3$

Here,

S = Spacing of Main Girder

t_s = Thickness of Slab

L = Span Length

N_b = Number of Beam/ Girder

$K_g = n(I + Ae_g^2)$

$n = E_B/E_D$

E_B = Modulus of elasticity of beam/ girder material (MPa)

E_D = Modulus of elasticity of deck/ slab material (MPa)

I = Moment of inertia of beam/ girder (mm^4)

e_g = distance between center of gravity of beam/ girder and deck/ slab (mm)

For Exterior Beam/ Girder

According to AASHTO 2007, Distribution Factors for moment for exterior beam/ girder are as follows:

Table 3.2 Distribution Factor for Moment (Exterior Girder) [AASHTO, 2007]

One Design Lane Loaded	Two or More Design Lanes Loaded	Range of Applicability
Lever Rule	$g = e\, g_{interior}$ $e = 0.77 + \frac{d_e}{2800}$	$-300 \le d_e \le 1700$
	use lesser of the values obtained from the equation above with N_b = 3 or the lever rule	$N_b = 3$

Here,

g = distribution factor for exterior beam/ girder

$g_{interior}$ = distribution factor for interior beam/ girder

e = eccentricity of a lane from centre of gravity of the pattern of girders (mm)

d_e = distance from the exterior web of exterior beam to the interior edge of curb or traffic barrier (mm).

3.14.3.6 Distribution Factor for Shear

As per AASHTO 2007, while calculating design shear, it should be multiplied by certain Distribution Factor. Value of Distribution Factor depends on several parameters like number of lane, span length of girder, deck slab thickness, lateral spacing of girders etc. For interior and exterior girder, AASHTO proposes different Distribution Factors.

For Interior Beam/ Girder

According to AASHTO 2007, Distribution Factors for shear for interior beam/ girder are as follows:

Table 3.3 Distribution Factor for Shear (Interior Girder) [AASHTO, 2007]

One Design Lane Loaded	Two or More Design Lanes Loaded	Range of Applicability
$0.36+\frac{S}{7600}$	$0.2+\frac{S}{3600}-\left(\frac{S}{10\,700}\right)^{2.0}$	$1100 \le S \le 4900$ $6000 \le L \le 73\,000$ $110 \le t_s \le 300$ $N_b \ge 4$
Lever Rule	Lever Rule	$N_b = 3$

Here,

S = Spacing of Main Girder

t_s = Thickness of Slab

L = Span Length

N_b = Number of Beam/ Girder

For Exterior Beam/ Girder

According to AASHTO 2007, Distribution Factors for shear for exterior beam/ girder are as follows:

Table 3.4 Distribution Factor for Shear (Exterior Girder) [AASHTO, 2007]

One Design Lane Loaded	Two or More Design Lanes Loaded	Range of Applicability
Lever Rule	$g = e\, g_{interior}$ $e = 0.6+\frac{d_e}{3000}$	$-300 \le d_e \le 1700$
	Lever Rule	$N_b = 3$

Here,

g = distribution factor for exterior beam/ girder

$g_{interior}$ = distribution factor for interior beam/ girder

e = eccentricity of a lane from centre of gravity of the pattern of girders (mm)

d_e = distance from the exterior web of exterior beam to the interior edge of curb or traffic barrier (mm).

Chapter 4

Optimization Method

4.1 Introduction

Optimization is the act of obtaining the best result under given circumstances. In the design, construction and maintenance of any engineering system, engineers have to take many technological and managerial decisions at several stages. The ultimate aim of all such decision is to either minimize the effort required or maximize the desire benefit. Since the effort required or the benefit desired in any practical situation can be expressed as a function of a certain design variables, optimization can be defined as the process of finding the conditions that give the minimum or maximum value of a function.

4.2 Classification of Optimization Problem

Generally optimization problems can be classified based on the nature of equation involved into two categories. This is based on the expression for the objective function and the constraints.

(i) **Linear optimization problems**: - where the expression for objective function and the expression for all constraints are linear function of design variable.

(ii) **Non-linear optimization problem**: - where the expression for objective function or the expressions for some or all of constraint are non linear function of the design variables.

4.3 Classification of Optimization Method

The available method of optimization may conveniently be divided into two distinctly different categories as follows:-

(i) **Analytical method**:-Which usually employ the mathematical theory of calculus (continuous differentiability, availability of gradient vectors and existence of second derivatives), variation method etc.

(ii) **Numerical method:** - which are usually employing a branch in the field of numerical mathematics called programming method. The recent developments in this branch are closely related to the rapid growth in computing capacity affected by the development of computers. In numerical methods, a near optimal design is automatically generated in iterative manner. An initial guess is used as starting points for a systematic search for better design. The search is terminated when certain criteria are satisfied; indicating that the current design is sufficiently close to the optimum.

Many mathematical programming methods have been developed for solving linear and nonlinear optimization problems during the last three decades. However, no single method has been found to be entirely efficient and robust for all different kinds of engineering optimization problems. Some methods, such as the penalty function method, the augmented Lagrangian method, and the conjugate gradient method, search for a local optimum by moving in a direction related to the local gradient. Other methods apply the first and second order necessary conditions to seek a local minimum by solving a set of nonlinear equations. For the optimum design of large structures, these methods become inefficient due to a large amount of gradient calculations and finite element analyses. These methods usually seek a solution in the neighborhood of the starting point similar to local hill climbing. If there is more than one local optimum in the problem, the result will depend on the choice of the starting point, and the global optimum cannot be guaranteed. Furthermore, when the objective function and constraints have multiple or sharp peaks, the gradient search becomes difficult and unstable.

So a truly versatile optimization algorithm for realistic problems should possess, at the very least, the following capabilities (Ghani, 1989).

(i) Ability to deal with nonlinear objective and constraining functions directly without the requirement of gradients or sub-gradients.
(ii) Objective and constraining functions allowed possessing finite number of discontinuities.
(iii) Restart facility to truly check the previously obtained minimum and high probability of directly locating the global minimum.

(iv) Ability to minimize objective functions with a mix of continuous, discrete and integer variables as arguments.

(v) Scaling of objective and constraining functions unnecessary.

(vi) The optimization problem allowed possessing simultaneously some or all features from above.

4.4 Global Optimization Algorithm

Global optimization algorithms are based on numerical or programming methods. These are an optimization algorithm that employs measures that prevent convergence to local optima and increase the probability of finding a global optimum. Figure 4.1 shows global and local optima of a two-dimensional function, f(X_1, X_2). Global optimization, so far, has been a rather difficult and illusive problem. It is still in its infancy, and consequently there is little in the literature compared to that for local optimization. Methods researched to date for global optimization are mainly for the unconstrained problem.

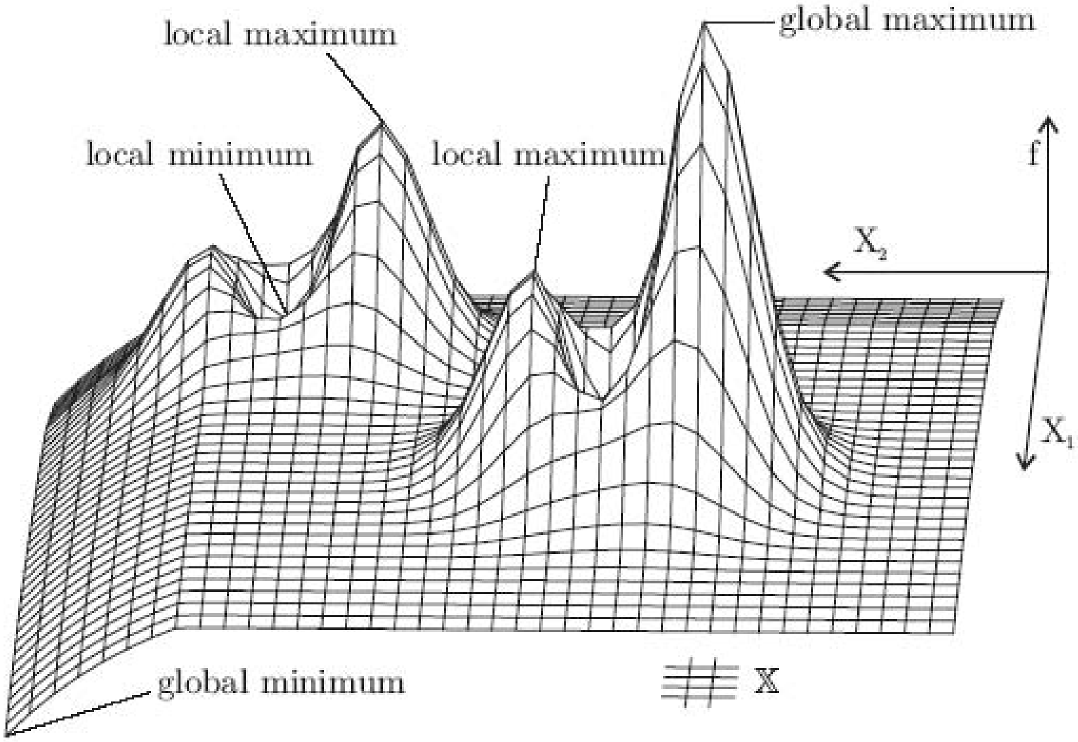


Figure 4.1Global and local optima of a two-dimensional function (Weise, 2008)

4.5 Statement of an Optimization Problem

An optimization or a mathematical programming problem can be stated as follows:-
Find $X = \{x_1, x_2... x_n\}$ which minimize or maximize, $F(X)$

Subject to constraints

$g_i(X) \leq 0$ and $h_j(X) = 0$ (j = 1,2,…….m) (4.1)
where, X is an n-dimensional vector called the design vector, $F(X)$ is called the objective function. $g_i(X)$ and $h_i(X)$ are, respectively, the equality and inequality constraints. The constrained stated in Eq. (4.1) is called a constrained optimization problem.
Some optimization problems do not involve any constraints and can be stated as:-
Find $X = \{x_1, x_2... x_n\}$ which minimize or maximize, $F(X)$
Such problems are called unconstrained optimization problems.

Design vector: - any engineering system or component is described by a set of quantities some of which are viewed as variables during the design process. In general certain quantities are usually fixed at the outset and these are called pre assigned parameter or constant design parameters. All the other quantities are treated as variables in the design process and are called design or decision variable, X_i, i = 1, 2… n. The design variables are collectively represented as a design vector.

In structural design, from physical point of view, the design variables $\boldsymbol{X}$ that are varied by optimization procedure may represent the following properties of the structure:

(i) The mechanical and physical properties of material
(ii) Topology of the structure i.e. the pattern of connection of members or the number of element in a structure.
(iii) The configuration or geometric layout of the structure
(iv) The cross-sectional dimensions or the member sizes

The types of design variables may be either continuous, integer, discrete or a combination of these types. Integer or discrete design variable is number of elements in the structure, for example. From mathematical point of view, it is important to distinguish between continuous and discrete variables.

Design Constraints: - In many practical problems, the design variable cannot be chosen arbitrarily; rather they have to satisfy certain specified functional, behavioral and other requirement. The restrictions that must be satisfied in order to produce an acceptable design are collectively called constraints. If the design meets the entire requirement placed on it, it is called a feasible design. From the physical point of view we may identify two kind of constraint:

(i) Constraints imposed on the design variables and which restrict their range. For reasons other than behavior considerations will be called **explicit constraint** or side constraints. These constraints, which are explicit in from, may derive from various considerations such as functionality, fabrication, or aesthetics thus, a side constraints is a specified limitation (upper or lower bound) on a design variable, or a relationship which fixes the relative value of a group of design variable, example of such constrain in structural design include minimum thickness of plate, maximum height of a shell structure, minimum slope of a roof structure.

(ii) Constraints that are derived from behavior requirements will be called behavior constraints or **implicit constraints** in structural design. For example, limitations on maximum stresses, deflections, flexural strength, or buckling strength are implicit constraints.

Explicit and implicit constraints are often given by formulas according to design codes or specifications. However implicit constraints are generally implicit in any case the constraint must be a computable function of the design variable. Form a mathematical point of view, both explicit and implicit constraints may usually be expressed as a set of inequalities, $g_i(X) \leq 0$; $(j = 1, 2 \ldots .m)$. Where m is the number of inequality constraints and $\boldsymbol{X}$ is the vector of design variables. In a structural design problem, one has also to consider equality constraints of the general form, $h_j(X) = 0$; $(j= 1\ldots p)$ .Where p is the number of equalities.

Objective function: - The conventional design procedures aim at finding an acceptable or adequate design, which merely satisfies the functional and other requirements of the problem. In general there will be more than one acceptable designs and the purpose of optimization is to choose the best out of the many acceptable design variable. Thus a criterion has to be chosen for comparing the different alternate acceptable design and selecting the best one. The criteria with respect to which the design is optimized when expressed as a function of the design variable is called objective function. The choice of the objective function is governed by the nature of the problem. For instant, in aircraft and aerospace structure design problem, the objective function is usually be weight of the structure, in civil engineering structure designs, the objective is usually taken as the minimization of cost.

4.6 Optimization Algorithm (EVOP)

A global optimization algorithm EVOP (Evolutionary Operation) for constrained parameter optimization has been presented. Few current methods cope with real world problems involving discontinuous objective and constraining functions where there is a combination of continuous, discrete and integer set of arguments and global minimum is sought. For noisy data, solutions are possible with genetic algorithms but costly parallel processing would be needed to locate the global minimum. Solutions remain elusive with genetic algorithms for problems with hard real-time constraints. The robust algorithm EVOP surmounts these difficulties with a much faster and more accurate solution.

Virtues of EVOP

Searching the Internet will yield a number of algorithms with the name EVOP. But this EVOP is unique in its speed and accuracy and flexibility. It appears this EVOP is the 'silver bullet' that has succeeded in slaying the dragon of dimensionality in multiple minima bound objective function. Nothing comparable is available to date. It has the capability to locate directly with high probability the global minimum. It is also capable to deal with possible finite number of discontinuities in the nonlinear objective and constraining functions. It has the ability to minimize directly an objective function without requiring information on gradient or sub-gradient. It can also deal with objective functions having a mix of integer, discrete and continuous variables as arguments. There is no requirement for scaling of objective and constraining functions. It has the capability for optimization even when there are more than one of the above difficulties simultaneously present. It has facility for automatic restarts to check whether the previously obtained minimum is the global minimum. It can optimize physical systems in real-time or accelerated time; e.g. optimal adaptive control of physical systems. Since objective function is never evaluated in the infeasible region, as a consequence the safety of the plant or system is not in jeopardy at any time because of optimization. Gradient or sub-gradient is not required thus ensuring that noise in measurement will not be accentuated to adversely affect the optimization process. It has inherent ability to cope with realistic hard time constraint requirement imposed by real-time systems.

An updated version of EVOP is available that is capable for minimization of objective function having combination of integer, discrete and continuous arguments (design variables). The method treats all arguments as continuous but for discrete and integer variables, picks values from thin strips centered on specified values. The procedure EVOP has successfully minimized a large number of internationally recognized test problems (Ghani, 1995). The problems were categorized as unconstrained, constrained, multiple minima and mixed variable problems.
The algorithm can minimize an objective function

$$F(X) = F(x_1, x_2 \ldots x_n) \tag{4.2}$$

Where, $F(x)$ is a function of n independent variables $(x_1, x_2 \ldots x_n)$. The n independent variables x_i's $(i = 1, 2 \ldots. n)$ are subject to explicit constraints

$$l_i \leq x_i \leq u_i \tag{4.3}$$

Where, l_i's and u_i's are lower and upper limits on the variables. They are either constants or functions of n independent variables (movable boundaries). These n independent variables x_i's are also subject to m numbers of implicit constraints

$$L_j \leq f_j(x_1, x_2 \ldots x_n) \leq U_j \tag{4.4}$$

Where, $j = 1, 2 \ldots. m$. L_j's and U_j's are lower and upper limits on the m implicit constraints. They are either constants or functions of n independent variables.

4.6.1 The Procedure

The method is subdivided into six fundamental processes (Figure 4.2) which are fully described in the reference (Ghani, 1989). They are,

(i) Generation of a 'complex',
(ii) Selection of a 'complex' vertex for penalization,
(iii) Testing for collapse of a 'complex',
(iv) Dealing with a collapsed 'complex',
(v) Movement of a 'complex' and
(vi) Convergence tests.

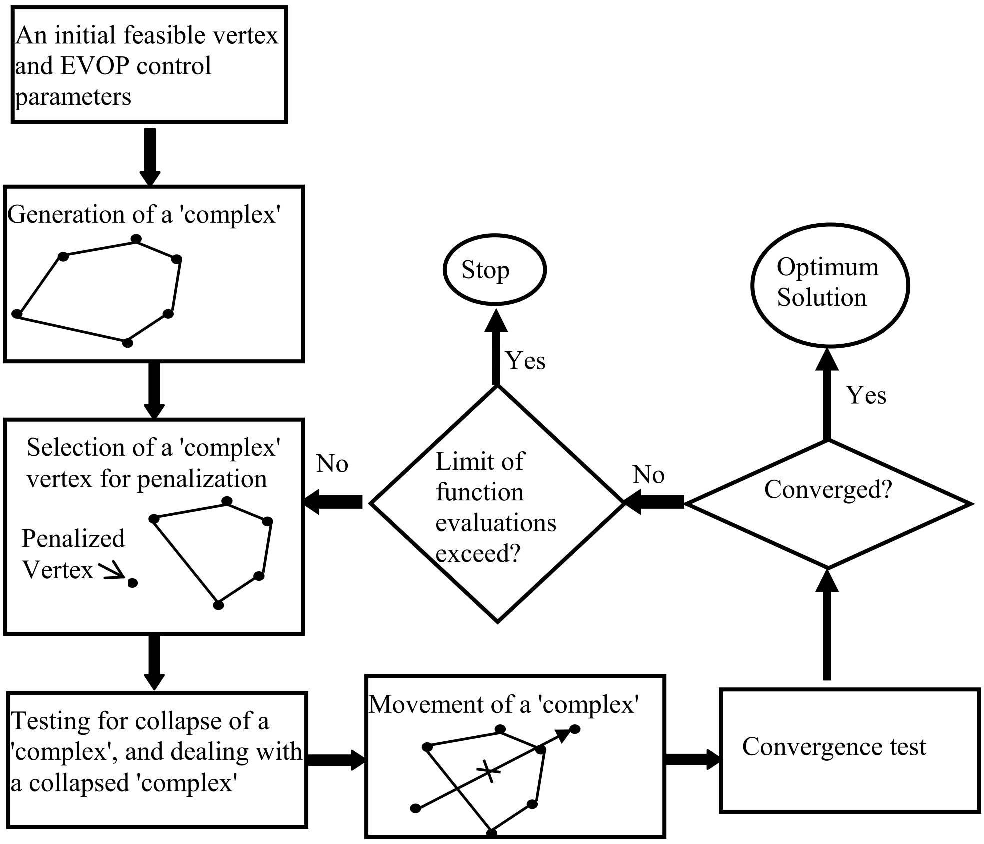


Figure 4.2 General outline of EVOP Algorithm (Rana, 2010)

A 'complex' is a 'living' object spanning an n-dimensional space defined by $k \geq (n+1)$ vertices inside the feasible region. It has the intelligence to move towards a minimum located on the boundary or inside the allowed space. It can rapidly change its shape and size for negotiating difficult terrain. Figure 4.3 shows a 'complex' with four vertices in a two dimensional parameter space. The 'complex' vertices are identified by lower case letters 'a', 'b', 'c' and 'd' in an ascending order of function values, i.e. f(a)<f(b)<f(c)<f(d). Straight line parallel to the co-ordinate axes are explicit constraints with fixed upper and lower limits. The curved lines represent implicit constraints set to either upper or lower limits. The hatched area is the two dimensional feasible search spaces.

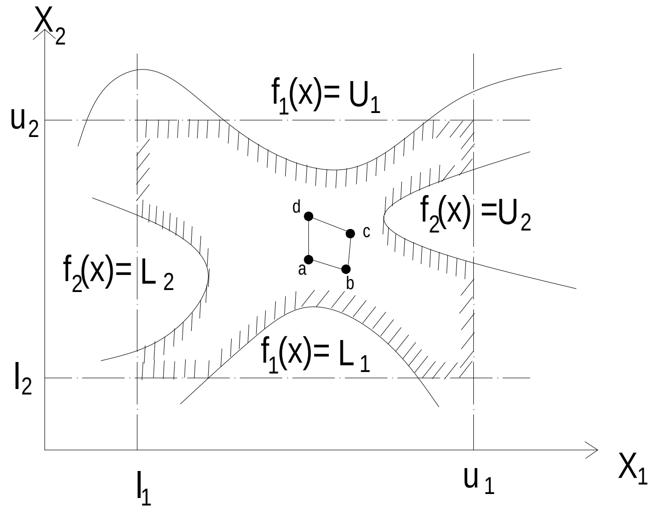


Figure 4.3 A "complex" with four vertices inside a two dimensional feasible search space (Ghani, 1989)

4.6.1.1 Generation of a 'complex'

Referring to Figure 4.4, for any feasible parameter space a random point is generated in such a way that all the explicit constraints are automatically satisfied. The co-ordinates of this random point are given by

$$x_i \triangleq l_i + r_i(u_i - l_i) \; (i = 1, 2 \ldots n) \qquad (4.5)$$

where r_i is a pseudo-random deviate of rectangular distribution over the interval (0,1).

The implicit constraints are satisfied by continually moving the newly generated point halfway towards the feasible centroid of all feasible vertices already generated. The new point **x** is obtained from the old feasible point **x'** and the feasible centroid **C** as follows,

$$\boldsymbol{x} = \frac{1}{2}(\boldsymbol{C} + \boldsymbol{x'}) \qquad (4.6)$$

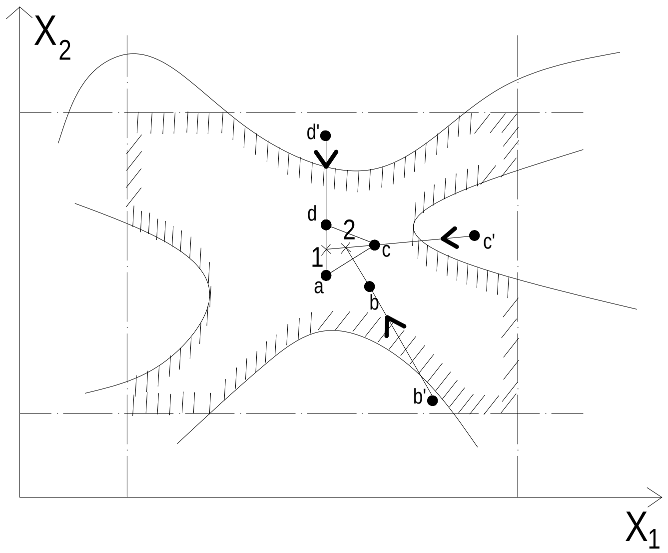


Figure 4.4 Generation of initial "complex" (Ghani, 1989)

Once x has satisfied all constraints it is added to the list of feasible complex vertices. This process is repeated till all k vertices that satisfy all explicit and all implicit constraints have been generated beginning from a single feasible starting point.

In simpler language, a starting point is required that satisfies all explicit and implicit constraint sets. A second point is randomly generated within the bounds defined by the explicit constraints. If this second point also happens to satisfy, all implicit constraints, everything is going fine. Centroid of the two feasible points is determined. If it satisfies all constraints, then things are really going fine and the 'complex' is updated with this second point. If, however, the randomly generated point fails to satisfy implicit constraints it is continually moved half way towards the feasible starting point till all constraints are satisfied. Feasibility of the centroid of the two points is next checked. If the centroid satisfies all constraints, then we have an acceptable 'complex', and we proceed to generate the third point for the 'complex'. If, however, the centroid fails to satisfy any of the constraints this second point is randomly once again generated in the space defined by the explicit constraints.

Referring to the Figure 4.4, the first random point is 'd', and the centroid is the feasible starting point 'a'. The point 'd' is moved halfway towards 'a' in order to satisfy the violated implicit constraint. Next the centroid of the feasible vertices 'a' and 'd' is determined which itself is feasible. Another random point 'c' is next generated and is moved to 'c' to satisfy the violated implicit constraint. Repeating the procedure all the (k-1) feasible vertices of the initial 'complex' are obtained. The initial 'complex' for the two dimensional example is the object 'abcd'. It can be seen that the same feasible point 'a' can be used again for generating a new initial 'complex' which would be of a completely different shape and size. The method allows repeated re-use of the same feasible starting point for checking whether the global minimum has been located.

For a convex feasible parameter space the above method would, without fail, succeed in generating a 'complex' with k vertices. If the parameter space is non-convex and the centroid happens to lie in the infeasible area, there is every chance that a 'complex' cannot be generated. Figure 4.5 shows such a possibility. Three vertices 'a', 'b' and 'c' in the feasible parameter space have already been generated. In order to generate the fourth feasible vertex a trial point 'T_1' satisfying the explicit constraints is created. However, 'T_1' is infeasible as it violates an implicit constraint. In order to make 'T_1' feasible it is continually moved halfway towards the centroid '**C**'. Since the centroid itself is infeasible no amount of such moves would make 'T_1' feasible, and a 'complex' with four vertices can never be generated. In such case if a new feasible 'complex' vertex results in the new centroid to lie in the infeasible area, that new vertex is discarded and another is generated until a feasible centroid is obtained.

In minimization involving combination of continuous, integer and discrete variables rounding off of appropriate co-ordinates of the trial point is conducted in the user written explicit constraint function. The co-ordinates of the centroid should never be rounded off.

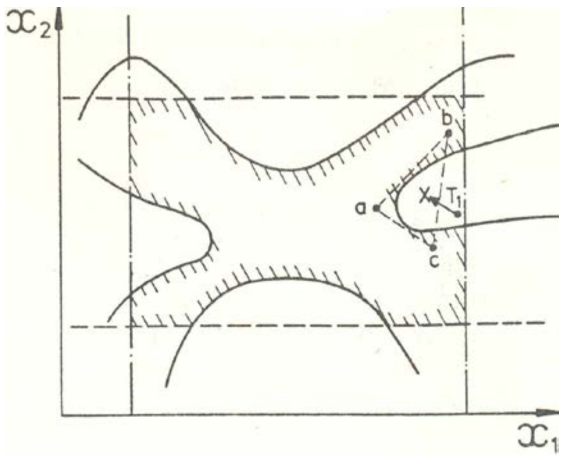


Figure 4.5 A 'complex' with four vertices 'abcd' cannot be generated

(Ghani, 1989)

4.6.1.2 Selection of a 'complex' vertex for penalization

In the present minimization process the worst vertex 'ng' of a 'complex' is that with the highest function value which is penalized by over-reflecting on the centroid. Referring to Figure 4.6, the penalized point 'd' is reflected over the centroid '**C**' to create a trial point 'T_l' which violates an implicit constraint. Since the centroid itself is in the infeasible region, repeated movement of point 'T_l' halfway towards the centroid would result in collapse of the point on the centroid. The new complex has now three vertices 'a', 'b' and 'c'. One more such collapse would result in complete collapse of the 'complex' because an object with two vertices cannot span a two dimensional space. In general a space of n dimension can only be spanned by objects defined by k vertices where k≥(n+1).

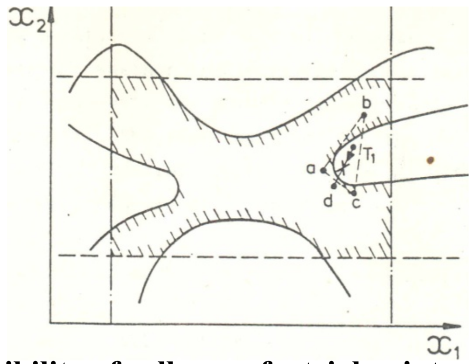


Figure 4.6 The possibility of collapse of a trial point onto the centroid.

(Ghani, 1989)

For selection of a 'complex' vertex for penalization the procedure as shown in Figure 4.7 is followed until a preset number of calls to the three functions (Implicit, Explicit and Objective) are collectively exceeded.

4.6.1.3 Testing for collapse of a 'complex'

A 'complex' is said to have collapsed in a subspace if the i^{th} coordinate of the centroid is identical to the same of all 'k' vertices of the 'complex'. This is a sufficiency condition and detects collapse of a 'complex' when it lies parallel along a coordinate axis. Once a 'complex' has collapsed to a subspace it can never again be able to span the original space. The word "identical" implies here "identical within the resolution of Φ_{cpx} which is a parameter for detection of 'complex' collapse. Numbers x and y are identical within the resolution of Φ_{cpx} if x and $\{x + \Phi_{cpx}(x-y)\}$ have the same numerical values. For Φ_{cpx} set to 10^{-2}, if x and y differs by not more than the last two significant digits they will be considered identical.

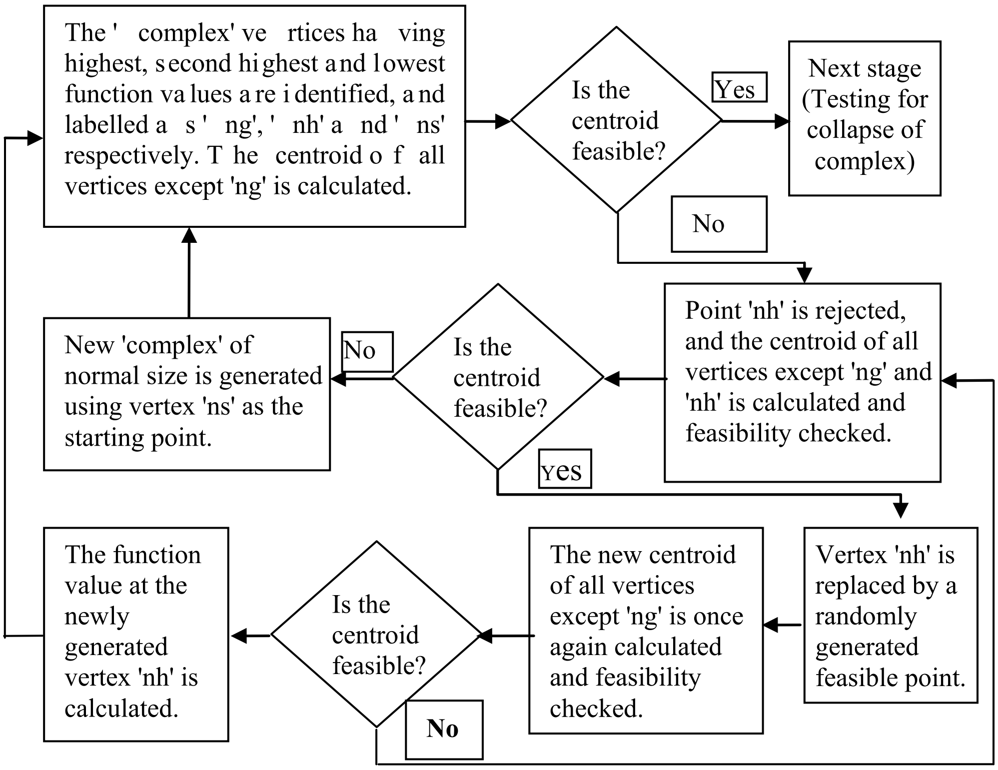


Figure 4.7 Selection of a 'complex' vertex for penalization (Rana, 2010)

Figure 4.8 shows a 'complex' with vertices 'a', 'b', 'c' and 'd', which has collapsed to a one dimensional search space. The X_2 coordinates of all vertices and the centroid are identical within the resolution of Φ_{cpx}. As can be seen the 'complex' vertices can now move only along the X_1 coordinate direction. Such collapses along other angular directions have not been accounted. Such a failure would rapidly lead to the type of collapse discussed above, albeit additional computations will be performed.

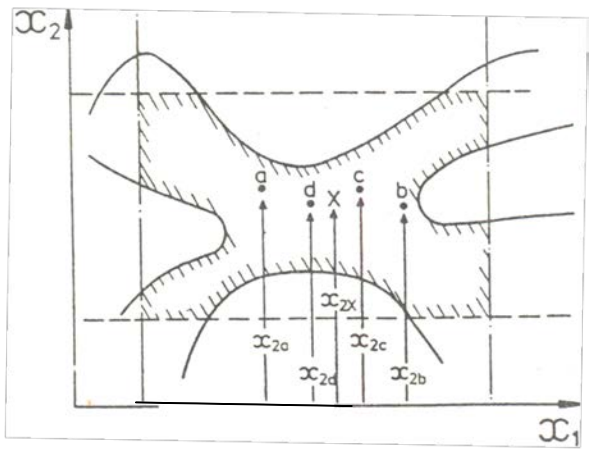


Figure 4.8 Collapse of a 'complex' to a one dimensional subspace. (Ghani, 1989)

4.6.1.4 Dealing with a collapsed 'complex'

On detecting collapse of a 'complex' some actions are taken such that a new ‘complex’ is generated within the full feasible space defined by the explicit and implicit constraints or a ‘complex’ spanning smaller feasible space. The process for movement of a ‘complex’ as explained below is continued.

4.6.1.5 Movement of a 'complex'

The process begins by over-reflecting the worst vertex '**ng**' of a 'complex' on the feasible centroid ‘**C**’ of the remaining vertices to generate a new trial point $\mathbf{x_r}$,

$$x_r = (1+\alpha)C - \alpha x_g \tag{4.7}$$

where α is reflection coefficient.

A check is then made to determine whether the trial point violates any constraints. If an explicit constraint is violated, the trial point is moved just inside the boundary by a small amount Δ called the explicit constraint retention coefficient. If any implicit constraint is violated the trial point is repeatedly moved halfway towards the centroid until the constraint is satisfied.

Figure 4.9 shows the penalized point 'd' on over-reflection violates an explicit constraint. The trial point 'T_1' is moved to 'T_2' just inside the constraint boundary by a factor, Δ.

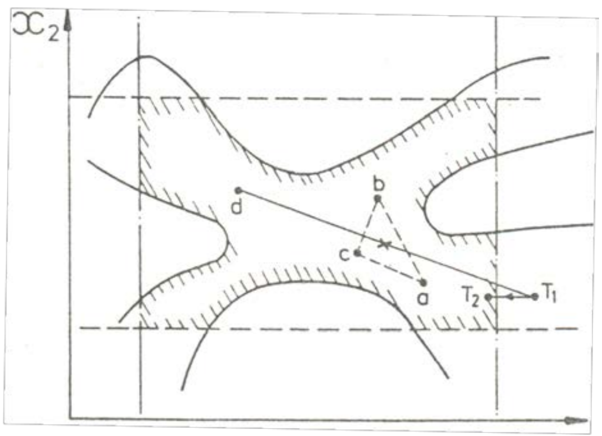


Figure 4.9 The reflected point violating an explicit constraint. (Ghani, 1989)

Figure 4.10 shows the case when an implicit constraint is violated. The infeasible trial point 'T_1' is moved halfway towards the feasible centroid '**C**' to 'T_2' which satisfies all constraints.

The function value at the feasible trial point is next evaluated. The reflection step is considered successful if the function value at this new trial point is lower than that at vertex 'nh', and the vertex 'ng' is replaced by the trial point. If, however, the function value at the trial point is greater than that at vertex 'nh' of the current 'complex', the trial point would be the worst vertex in the new 'complex' configuration. The reflection step is, therefore, considered unsuccessful and contraction step applied.

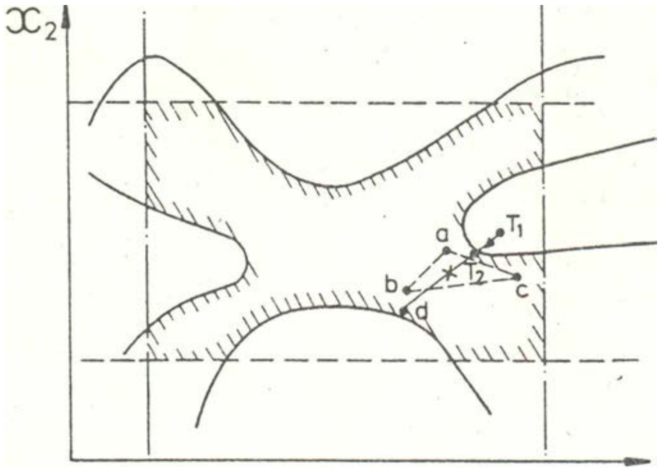


Figure 4.10 The reflected trial point violating an implicit constraint (Ghani, 1989)

Depending on situation anyone of the three stages (Stages 1-3) of the contraction step can be called. If the function value at the feasible trial point after over-reflection is less than that at vertex 'ng' but equal to or greater than that at vertex 'nh', Stage 1 of contraction step is applied. This is essentially an under-reflection, and the co-ordinates of the new trial point **x** is given by

$$x = (1+\beta)C - \beta x_g \tag{4.8}$$

where β is contraction coefficient.

Figure 4.11 shows the worst vertex 'd' of the current 'complex' 'abcd' over-reflected on the feasible centroid '**C**'. The trial point 'T_1' so obtained is moved just inside an explicit constraint resulting in trial point 'T_2' which still violates an implicit constraint.

'T_2' is moved halfway towards the centroid '**C**' along the line joining the two resulting in a completely feasible trial point 'T_3'. Function value at 'T_3' is calculated, and found to be intermediate between the second highest function value at vertex 'c', and the highest function value at vertex 'd'. If 'd' is replaced by the feasible trial point 'T_3' to form a new complex 'abcT_3', then 'T_3' would be the worst vertex in the new configuration. The reflection step is, therefore, considered unsuccessful and point 'T_3' is rejected, and Stage 1 of contraction step is applied. The worst vertex 'd' is under-reflected on the feasible centroid '**C**' to 'T_4'. Since the trial point 'T_4' violates an implicit constraint it is moved halfway towards the centroid to 'T_5'. The trial point 'T_5' is feasible, and replaces the worst vertex 'd' to form a new complex 'abcT_5'.

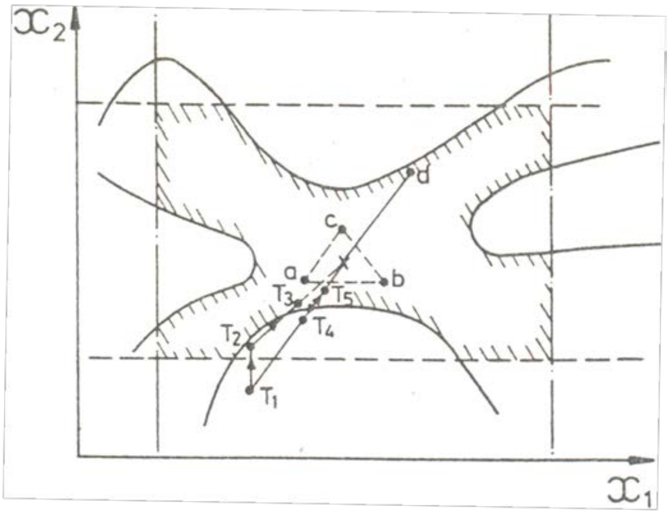


Figure 4.11 Unsuccessful over-reflection and Stage of contraction step applied.
(Ghani, 1989)

Stage 2 of contraction step is applied if at the end of over-reflection the function value at the feasible trial point is equal to or greater than that at the worst vertex 'ng' of the current complex. The co-ordinates of the new trial point are given by:

$$x = \beta x_g + (1-\beta)C \tag{4.9}$$

Figure 4.12 shows that the point 'd' of a current 'complex' 'abcd' is reflected over the centroid '**C**' of the remaining points 'a', 'b' and 'c' to obtain a trial point 'T_1' which violates both explicit and implicit constraints. The explicit constraint is satisfied by moving the point 'T_1' to 'T_2' just inside the boundary of the explicit constraint. The implicit constraint is satisfied by moving the point 'T_2' to 'T_3' halfway towards the feasible centroid '**C**'. The function value at the feasible trial point 'T_3' is found to be greater than the highest function value of the current 'complex' at vertex 'd'. The reflection step is, therefore, considered unsuccessful, and point 'T_3' is rejected. Stage 2 of contraction step is applied penalizing the worst vertex 'd' to 'T_4'. The trial point 'T_4' violates an implicit constraint. It is made feasible by moving halfway towards the feasible centroid '**C**' to 'T_5'. 'T_5' replaces the worst vertex 'd' to form a new 'complex' 'abcT_5'.

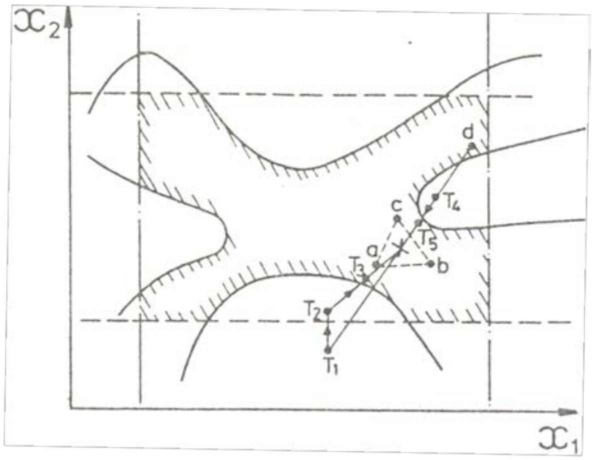


Figure 4.12 Unsuccessful over-reflection and Stage 2 of contraction step applied

(Ghani, 1989)

Stage 3 of contraction step is called only after Stages 1 and/or 2 have been previously applied consecutively for more than '2k' times. A small 'complex' is generated using vertex 'ns' as the starting point. If on over-reflection the trial point has not violated any constraints, has a function value lower than the lowest function value at vertex 'ns' of the current 'complex', and the previous move was not a contraction step, this over-reflection is considered over-successful. Expansion step is then applied to generate a new trial point further away from the feasible centroid along the same straight line used for over-reflection. The co-ordinates of this accelerated trial point is given by

$$x = \gamma x_r + (1-\gamma)C \tag{4.10}$$

Where, γ is expansion coefficient.

Feasibility of this accelerated trial point is next checked. If any constraint is violated, the acceleration step is considered unsuccessful, and a new 'complex' is formed with the over-reflected feasible trial point $\boldsymbol{x_r}$ as the updated vertex replacing the worst vertex 'ng' of the current 'complex'. Otherwise the function value at the feasible accelerated trial point is evaluated. If it is lower than that at x the acceleration step is considered successful. The accelerated point then replace the worst vertex ‘ng' to form the new 'complex'. Else, if the function value at the accelerated point equals or exceeds that at the over-reflected point $\boldsymbol{x_r}$ the acceleration step is also considered unsuccessful. The accelerated point is rejected in favor of the point $\boldsymbol{x_r}$ to form the new 'complex'.

Figure 4.13 shows a successful acceleration step. The over-reflected trial point 'T_1' does not violate any constraint, and has a function value lower than the lowest function value at vertex 'a' of the current 'complex' 'abcd'. Since contraction step was not applied previously acceleration step is called to obtain the trial point 'T_2'. 'T_2' does not violate any constraint and yet has a function value lower than that at 'T_1 '. Trial point 'T_2' replaces the worst vertex 'd' of the current 'complex' to form the updated 'complex' 'abcT_2'.

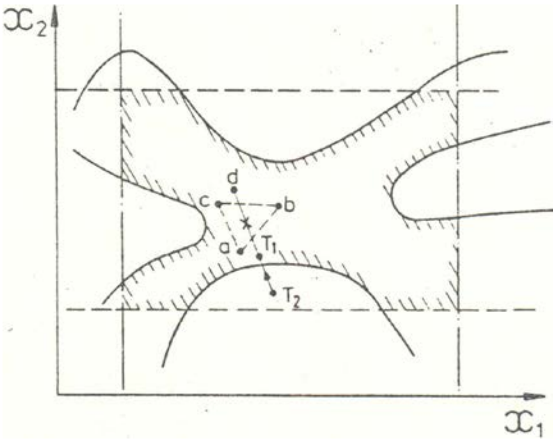


Figure 4.13 Successful acceleration steps (Ghani, 1989)

4.6.1.6 Convergence tests

While executing the process of movement of a ‘complex’, test for convergence are made periodically after certain preset number of calls to the objective function. There are two levels of convergence tests. The first convergence test would succeed only if a predefined number of consecutive function values are identical within the resolution of convergence parameter Φ, which should be finer than Φ_{cpx}. The second convergence test is attempted only if the first convergence test succeeds. This second test for convergence verifies whether function values at all vertices of the current 'complex' are also identical within the resolution of Φ.

CHAPTER 5

PROBLEM FORMULATION

5.1 Introduction

Appropriate optimization problem has to be formulated for the present bridge structure (a two-span continuous post-tensioned girder of bridge) to be solved by an optimization method (EVOP). Then, optimization method solves the problem and gives the optimum solutions. In this chapter the various components of the optimization problem of the present study are described. The various components are mathematical expression required for the design and analysis of the bridge system, an objective function, implicit constraints, explicit constraints and input control parameters for the optimization method.For design and analysis of the two-span continuous girder, eight sections/ positions along the span of the girder were considered. The structure being an indeterminate one, Stiffness Method was incorporated in the mathematical expression required for the design and analysis of it. Prestress losses, allowable and ultimate strength design criteria, ductility limits, lateral stability and deflection criteria were applied/ considered by taking into account the continuity effect of the structure.The program is finally linked to the optimization method to obtain the optimumsolutionsof cost optimum design.

5.2 Objective Function

In this study, the objective is the cost minimization of the present bridge systems by taking into account the cost of all materials, fabrication, and installation. The total cost of a bridge system is formulated as:

$$C_T = C_{GC} + C_{DC} + C_{PS} + C_{OS} \tag{5.1}$$

where, C_{GC}, C_{DC}, C_{PS} and C_{OS} are the cost of materials, fabrication and installation of Girder Concrete, Deck slab Concrete, Prestressing Steel and Ordinary Steel for deck reinforcement and girder's shear reinforcement respectively. Costs of individual components arecalculated as:

$$C_{GC} = (UP_{GC}\,V_{GC} + UP_{GF}\,SA_G)\,N_G \tag{5.2}$$

$$C_{DC} = (UP_{DC}V_{DC} + UP_{DF}(S\text{-}TF_w))\ N_G \quad (5.3)$$

$$C_{PS} = (UP_{PS}W_{PS} + 2\ UP_{ANC}\ N_T + UP_{SH}\ N_T\ L)\ N_G \quad (5.4)$$

$$C_{NS} = UP_{OS}\ (W_{OSD} + W_{OSG})\ N_G \quad (5.5)$$

where, UP_{GC}, UP_{DC}, UP_{PS} and UP_{OS} are the unit prices including materials, labor, fabrication and installation of (i) the precast girder concrete, (ii) deck concrete, (iii) prestressing steel and (iv) ordinary steel respectively; UP_{GF}, UP_{DF}, UP_{ANC}, UP_{SH} are the unit prices of girder formwork, deck formwork, anchorage set and metal sheath for duct respectively; V_{GC}, V_{DC}, W_{PS}, W_{OSD} and W_{OSG} are the volume of the precast girder concrete and deck slab concrete, weight of prestressing steel and ordinary steel in deck slab and in girder respectively; L is the girder span; N_G is number of girders; S is girder spacing.

5.3 Design Variables and Constant Design Parameters

For a particular girder span and bridge width, a large number of parameters control the design of the bridge such as girder spacing, cross sectional dimensions of girder, deck slab thickness, number of strands per tendon, number of tendons, deck slab reinforcement, configuration of tendons, anchorage system, pre-stress losses, concrete strength etc. The design variables and variable type considered in the study are tabulated in Table 5.1. A typical cross-section of the two-span continuous PC I-girder at 0.4L distance from end support is illustrated in Figure 5.1 to highlight several of the design variables. Besides, typical cross-sections i) above end support (at position of zero moment), ii) at positive moment region and iii) just above the interior support (at position of maximum negative moment) are illustrated in Figure 5.2 as well.

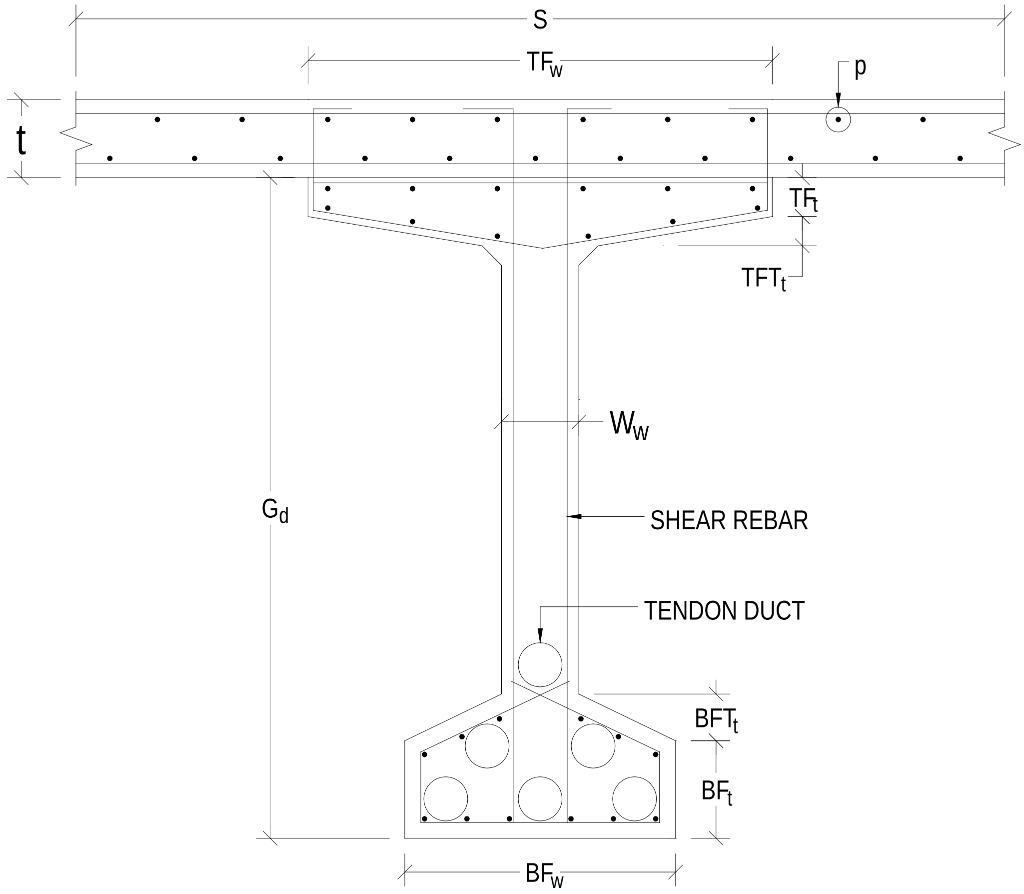


Figure 5.1 Section with design variables at positive moment region (Rana, 2010).

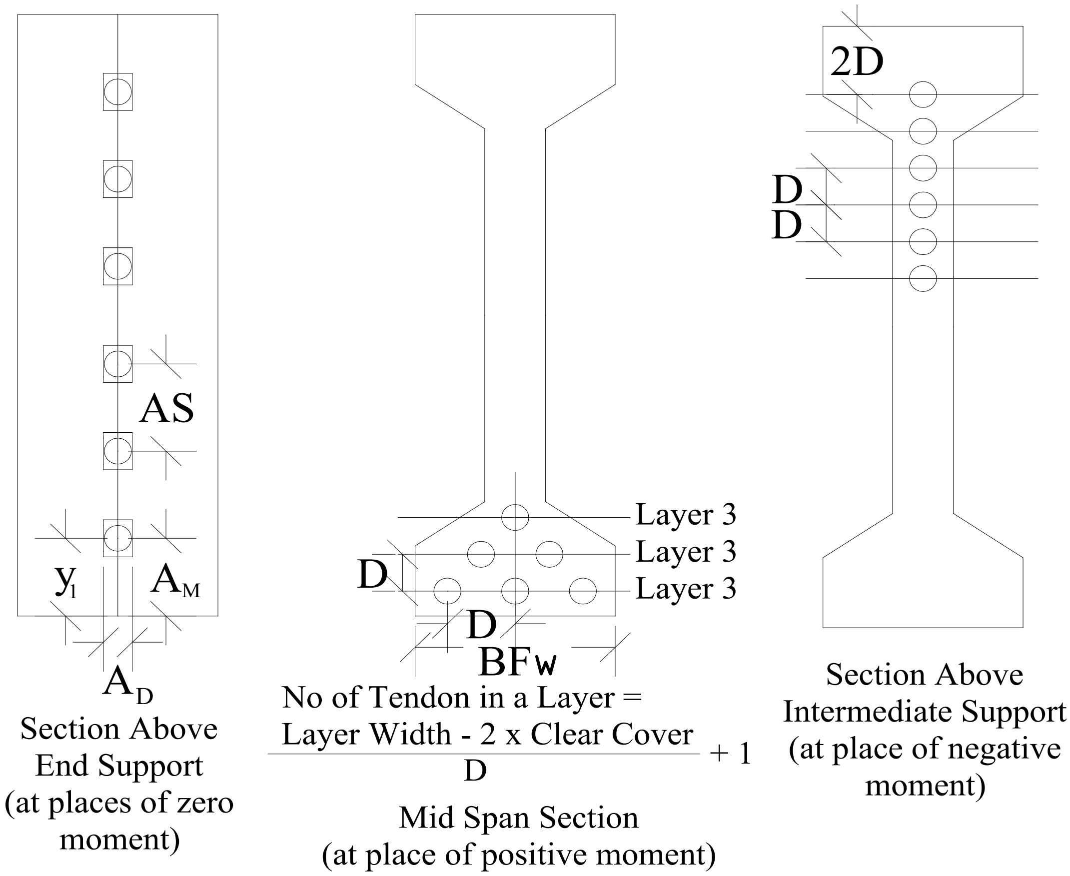


Figure 5.2 Typical cross-section at various positions along the span

The constant design parameters under consideration are various material properties, superimposed dead loads, AASHTO live load, strand size, post-tensioning anchorage system and unit costs of materials including fabrication and installation etc. Optimization is based on the analysis of an interior girder arranged as shown in Figure 5.3. The girder and the deck are assumed to act as a composite section during service condition. Prestress is considered to be applied in two stages, a percentage of total prestress at initial stage to carry only the girder self weight & stress produced during lifting and transportation and full prestress during casting of deck slab. In the present study the tendons arrangement is not assumed as fixed rather it is considered as design variable as it has significant effects on prestress losses and flexural stress at various sections along the girder. Tendons layout along the span is assumed as parabolic. The vertical and horizontal arrangement of tendons depends on various cross sectional dimensions of girder such as depth, bottom flange, and web. Typical arrangements of tendons at various sections are shown in Figure 5.2. The arrangement of tendons also depends on duct size and spacing, anchorage spacing and anchorage edge distance. These parameters depend on a design variable, namely, number of strand per tendon and on a constant parameter, namely, concrete strength, and are determined using values listed in the Table 5.2.

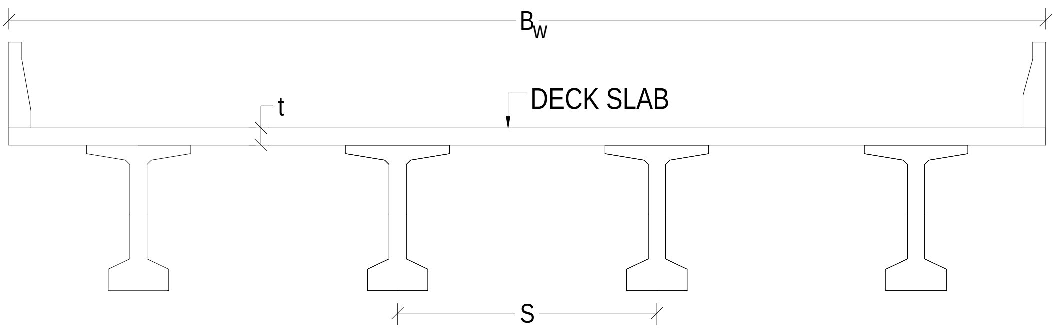


Figure 5.3 Girders arrangement in the bridge

Table 5.1 Design variables with variable type

Design variables	**Variabletype**
Girder spacing (S) (m)	Discrete
Girder depth (G_d) (mm)	Discrete
Top flange width (TF_w) (mm)	Discrete
Top flange thickness (TF_t) (mm)	Discrete
Top flange transition thickness (TFT_t) (mm)	Discrete
Bottom flange width (BF_w) (mm)	Discrete
Bottom flange thickness (BF_t) (mm)	Continuous
Web width (W_w) (mm)	Discrete
Number of strands per tendon (N_s)	Integer
Number of tendons per girder (N_T)	Integer
Lowermost tendon position at the end from bottom (y_1) (mm)	Continuous
Initial stage prestress (% of full prestress) (η)	Continuous
Slab thickness (t) (mm)	Discrete
Slab main reinforcement ratio (ρ)	Continuous

Table 5.2 Minimum dimensions for C range anchorage system

No. of strands per tendon	**1-3**	**4**	**5-7**	**8-9**	**10-12 (mm)**	**13**	**14-19**	**21-22**	**23-27**
Duct diameter	45	50	65	70	85	85	100	110	115
Duct clear spacing (D_S)	38	38	38	38	38	38	38	50	50
A_D	110	120	150	185	200	210	250	275	300
A_M	128	150	188	210	248	255	300	323	345

$AS = \frac{F}{f_c \times BF_w}$; f_c= concrete strength at stressing time; F = Prestressing force; A_D = Anchorage dimension; A_M = Anchorage minimum vertical edge distance

5.4 Explicit Constraints

These are specified limitation (upper or lower limit) on design variables which are derived from geometric requirements (superstructure depth, clearances etc.), minimum practical dimension for construction, code restriction etc. The constraint is defined as

$$X_L \leq X \leq X_U \tag{5.6}$$

Where, X = Design variable, X_L = Lower limit of the design variable, X_U = Upper limit of the design variable.

Explicit constraints for girder spacing: Lower and upper limit of girder spacing is considered such that number of girder in the bridge can vary from 1 to 10.

Explicit constraints for top flange: The lower limit of top flange width is assumed as 300 mm from lateral stability and bearing considerations and upper limit equal to girder spacing. The lower limit of top flange thickness is considered as 75 mm to resist damage during handling and proper placement of transverse reinforcement and upper limit is assumed as 300 mm. The lower limit of top flange transition thickness is considered as 50 mm to facilitate placement and consolidation of concrete and upper limit is assumed as 300 mm. The haunch thickness and width is assumed as 50 mm.

Explicit constraints for web: The lower limit of web width is equal to diameter of duct plus web rebars and clear cover (Figure 5.4) and upper limit is assumed as 300 mm.

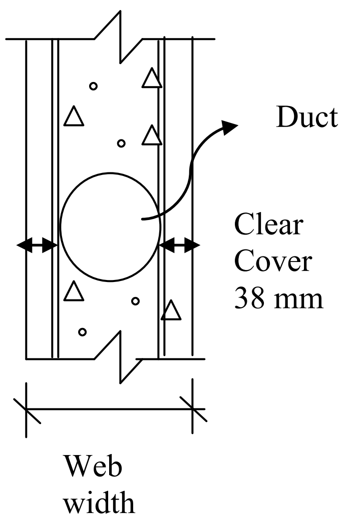


Figure 5.4 Width of web (Rana, 2010)

Explicit constraints for bottom flange: The lower limit of bottom flange width is assumed as 300 mm to accommodate anchorage setup and upper limit equal to girder spacing. The lower limit of thickness is equal to clear cover and duct diameter to fit at least one row of tendons. The upper limit is assumed as 600 mm. The width to thickness ratio of bottom flange transition area is assumed as 2 to 1 from practical construction point of view.

Explicit constraints for girder depth: The lower limit of girder depth is considered as 1000 mm and upper limit 3500 mm which is common range of girder depth to minimize the cost of substructure and approach roads and from aesthetics and limited clear space criterion.

Explicit constraints for number of strand per tendon: Within the available anchorage system one tendon may consist of several seven-wire strands like 1 to 55. Here the effect of number of strands in a tendon is studied. For this study it is considered that each tendon may consist of 1 to 27 strands.

Explicit constraints for number of tendon: The amount of pre-stressing force required for cost optimum design are directly associated with the number of tendons required in the girder. For this study it is considered that the number of tendon may vary from 1 to 20.

Explicit constraints for lowermost tendon position: To vary the profile of tendon along the girder span the lower most tendon position from bottom at the end section is considered as a design variable and the other tendon positions are determined from anchorage spacing. The lower limit of vertical position of the tendon is considered equal to anchorage minimum vertical edge distance and upper limit is assumed as 1000 mm.

Explicit constraints for deck slab: The lower limit of deck slab thickness is considered as 175 mm to control deflection and excessive crack and upper limit as 300 mm. The lower and upper limits of deck slab reinforcement are considered according to AASHTO standard specification. The explicit constraints for all the above design variables are shown in Table 5.3.

Table 5.3 Design variables with Explicit Constraints

Design variables	Explicit Constraint
Girder spacing (S) (m)	$B_W/10 \leq S \leq B_W$
Girder depth (G_d) (mm)	$1000 \leq G_d \leq 3500$
Top flange width (TF_w) (mm)	$300 \leq TF_w \leq S$
Top flange thickness (TF_t) (mm)	$75 \leq TF_t \leq 300$
Top flange transition thickness (TFT_t) (mm)	$50 \leq TFT_t \leq 300$
Bottom flange width (BF_w) (mm)	$300 \leq BF_w \leq S$
Bottom flange thickness (BF_t) (mm)	$a \leq BF_t \leq 600$
Web width (W_w) (mm)	$b \leq W_w \leq 300$
Number of strands per tendon (N_s)	$1 \leq N_s \leq 27$
Number of tendons per girder (N_T)	$1 \leq N_T \leq 20$
Lowermost tendon position at the end from bottom (y_1) (mm)	$A_M \leq y_1 \leq 1000$
Initial stage prestress (% of full prestress) (η)	$1\% \leq \eta \leq 100\%$
Slab thickness (t) (mm)	$175 \leq t \leq 300$
Slab main reinforcement ratio (ρ)	$\rho_{min} \leq \rho \leq \rho_{max}$

a = clear cover + duct diameter; b = clear cover + web rebar's diameter + duct diameter; A_M = Anchorage minimum vertical edge distance

5.5 Implicit Constraints

These constraints represent the performance requirements or response of the bridge system. A total 46 implicit constraints are considered according to the AASHTO Standard Specifications (AASHTO 2007). These constraints are categorized into eight groups:

(i) Flexural working stress constraints
(ii) Flexural ultimate strength constraints
(iii) Shear constraints (ultimate strength)
(iv) Ductility constraints
(v) Deflection constraints
(vi) Lateral stability constraint
(vii) Tendons eccentricity constraint and
(viii) Deck slab design constraint

These constraints are formulated as below:

5.5.1 Flexural working stress constraints:

These constraints limit the working stresses in concrete and are given by:

$$f^L \leq f_j \leq f^U \tag{5.7}$$

$$f_j = -\frac{F_j}{\mathrm{A}} \pm \frac{F_j e_j}{\mathrm{S_j}} \pm \frac{M_j}{\mathrm{S_j}} \tag{5.8}$$

Where, f^L = allowable compressive stress (lower limit), f^U = allowable tensile stress (upper limit) and f_j is the actual working stress in concrete; F_j, e_j, S_j, M_j = prestressing force, tendons eccentricity section modulus and moment at j[th] section respectively. These constraints are considered at eight critical sections along the span of the girder as shown in Figure 5.5 and for various loading stages (initial stage and service conditions). The eight sections are (describing from left support):

Section 4: Section where anchorages are placed

Section 2: End of anchorage and transition zone (Considered at a distance of 1.5 times girder depth).

Section 3: Section where prestress is at its maximum value.

Section 1: Section at 0.4L distance from end support.

Section 7: Midpoint of section 1 and 6.

Section 6: Section at 0.1L distance from interior support (where parabolic tendon changes its curvature)

Section 8: Midpoint of section 6 and 5.

Section 5: Section at interior support.

Allowable stresses for prestressed concrete (AASHTO 2007)

Compression stress:

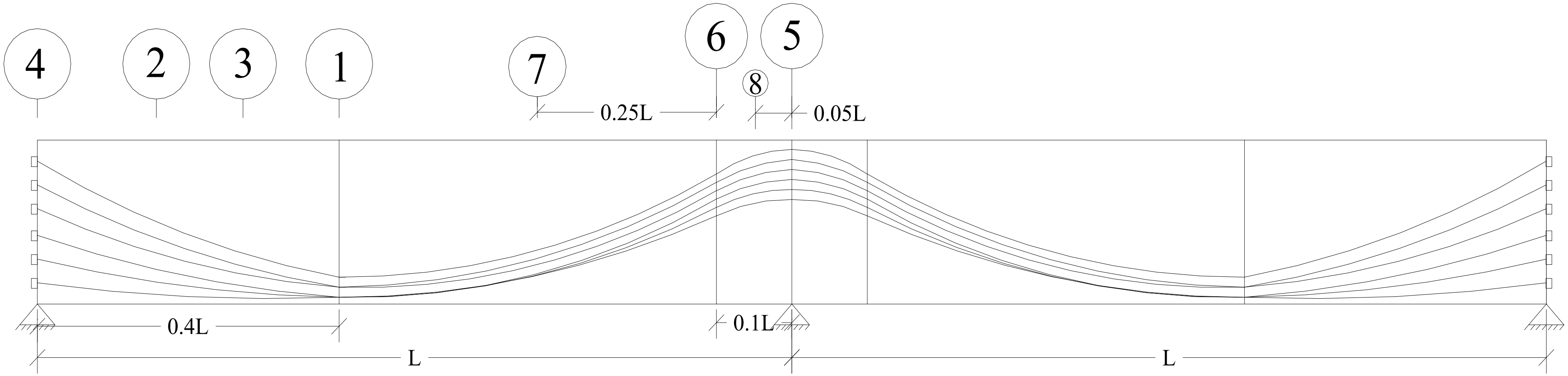
1. The stress limit due to the sum of the effective prestress, permanent loads, and transient loads and during shipping and handling is taken as $\boldsymbol{0.6f'_c}$ and $\boldsymbol{0.55f'_{ci}}$ at transfer.
2. The stress limit in prestressed concrete at the service limit state after losses for fully prestressed components in bridges other than segmentally constructed

due to the sum of effective prestress and permanent loads shall be taken as $\boldsymbol{0.45f'_c}$

3. The stress limit in prestressed concrete at the service limit state after losses for fully prestressed components in bridges other than segmentally constructed due to live load plus one-half the sum of the effective prestress and permanent loads shall be taken as $\boldsymbol{0.40f'_c}$

Tension stress:

The stress limit in prestressed concrete at the service limit state after losses for fully prestressed components in bridges other than segmentally constructed, which include bonded prestressing tendons and are subjected to not worse than moderate corrosion conditions shall be taken as the following: $\boldsymbol{0.25\sqrt{f'_{ci}}}$(MPa) (initial) and $\boldsymbol{0.5\sqrt{f'_c}}$(MPa) (Final).



Section 1: Section at 0.4L distance from end support; Section 2: End of anchorage and transition zone (Considered at a distance of 1.5 times girder depth); Section 3: Section where prestress is at its maximum value; Section 4: Section where anchorages are placed; Section 5: Section at interior support; Section 6: Section at 0.1L distance from interior support (where parabolic tendon changes its curvature); Section 7: Midpoint of section 1 and 6; Section 8: Midpoint of section 6 and 5.

Figure 5.5 Tendons profile along the girder

The initial loading stage includes the girder self weight and prestressing force after instantaneous losses (friction loss, anchorage loss and elastic shortening loss). In this stage net cross sectional properties of precast girder are used excluding duct. At initial stage a portion of total prestress is applied only to carry girder self weight. At service the first loading stage includes initial loading stage in addition slab and diaphragm weight. In this stage transformed cross sectional properties of precast girder are used and full prestress is applied. The second loading stage includes first loading stage in addition loads due to wearing course and median strip superimposed on composite section and prestress force after total losses is considered. The third loading stage includes live load and impact load superimposed on composite section in addition to second loading stage. The fourth loading stage includes half of dead load and prestress force plus full live load. Loading stages are summarized in Table 5.4.

Table 5.4 Loading stages and implicit constraints

Load stage	Resisting section	Section properties	Load Combination	Implicit constraint
Initial stage	Precast section	A_{net}, e_i, S_{net}	$\eta F+G$	Eq. (5.9)
1	Precast section	A_{tf}, e, S	$F_i+G+SB+DP$	Eq. (5.10)
2	Precast section + Composite section	A_{tf}, e, S + S_C	$F_e+G+SB+DP$ + SD	Eq. (5.11)
3	Precast section + Composite section	A_{tf}, e, S + S_C	$F_e+G+SB+DP$ + $SD+L+I$	Eq. (5.12)
4	Precast section + Composite section	A_{tf}, e, S + S_C	$0.5(F_e+DL)$ + $L+I$	Eq. (5.13)

G = Girder self weight; SB = slab weight; DP = diaphragm weight; SD = superimposed dead load for wearing coarse and curb weight; DL = total dead load; L = live load; I = impact load. F = Jacking force; F_i, F_e = Prestressing force after initial losses and total losses respectively;

$$-0.55f'_{ci} \leq f_i \leq 0.25\sqrt{f'_{ci}} \tag{5.9}$$

$$-0.60f_c' \leq f \leq 0.5\sqrt{f_c'} \tag{5.10}$$

$$-0.40f_c' \leq f \leq 0.5\sqrt{f_c'} \tag{5.11}$$

$$-0.60f_c' \leq f \leq 0.5\sqrt{f_c'} \tag{5.12}$$

$$-0.40f_c' \leq f \leq 0.5\sqrt{f_c'} \tag{5.13}$$

Prestress losses are estimated according to AASHTO 2007 instead of using lump sum value for greater accuracy because prestress losses are also implicit functions of some of design variables. The instantaneous losses depend on jacking equipment and anchorage hardware used and the design variables (number of tendons, number of strands per tendon, layout of tendon in the girder, prestressing of tendon and girder cross sectional properties). The long term losses are loss due to creep of concrete, loss due to shrinkage of concrete and loss due to steel relaxation and are also implicit functions of some of design variables. In post-tensioned girder, prestressing forces varies along the length of the girder due to friction losses and anchorage losses as shown in Figure 5.6. The prestressing forces after instantaneous losses at seven critical sections and at the end are determined as follows:

$$F_{1i} = F - \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + 0.4KL)}\right) - L_{ES} \tag{5.14}$$

$$F_{3i} = F - 0.5L_{ANC} - L_{ES} \tag{5.15}$$

$$F_{2i} = F_{3i} - 0.5L_{ANC}\left(\frac{x_2 - x_3}{x_2}\right) - L_{ES} \tag{5.16}$$

$$F_{4i} = F - L_{ANC} - L_{ES} \tag{5.17}$$

$$L_{ES} = K_{es}\frac{E_s}{E_{ci}}f_{cgp}A_s \tag{5.18}$$

$$L_{ANC} = \frac{4\,(F - F_{1i})}{L}\sqrt{\frac{LA_sE_s}{2(F - F_{1i})}\delta} \tag{5.19}$$

$$F_{5i} = F_{8i} - \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + 0.05KL)}\right) - L_{ES} \tag{5.20}$$

$$F_{6i} = F_{7i} - \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + 0.25KL)}\right) - L_{ES} \tag{5.21}$$

$$F_{7i} = F_{1i} - \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + 0.25KL)}\right) - L_{ES} \tag{5.22}$$

$$F_{8i} = F_{6i} - \sum_{i=1}^{i=N_T} F_i\left(1 - e^{-(\mu\theta_i + 0.05KL)}\right) - L_{ES} \tag{5.23}$$

The prestress forces after all losses at seven sections are F_{1e}, F_{2e}, F_{3e}, F_{5e}, F_{6e}, F_{7e} and F_{8e} respectively. F or p ost-tensioned m embers a ccording t o A ASHTO a llowable prestress immediately after seating at anchorage $0.7 f_{su}$, at the end of the seating loss zone $0.83f_y^*$ and st ress at ser vice l oad af ter l osses $0.80f_y^*$. In t he pr esent s tudy tensioning t o $0.9f_y^*$ (jacking s tress) f or s hort p eriod o f time p rior to s eating is considered to offset anchorage andfriction losses and implicit constraints are applied such th at th e s tresses in th e te ndon re main w ithin th e a llowable lim it. T he i mplicit constraints are as follows:

$$0.0 \le F_{4i} \le 0.7 f_{su} A_s \tag{5.20}$$

$$0.0 \le F_{3i} \le 0.83\, f_y^* A_s \tag{5.21}$$

$$0.0 \le F_{3e} \le 0.80\, f_y^* A_s \tag{5.22}$$

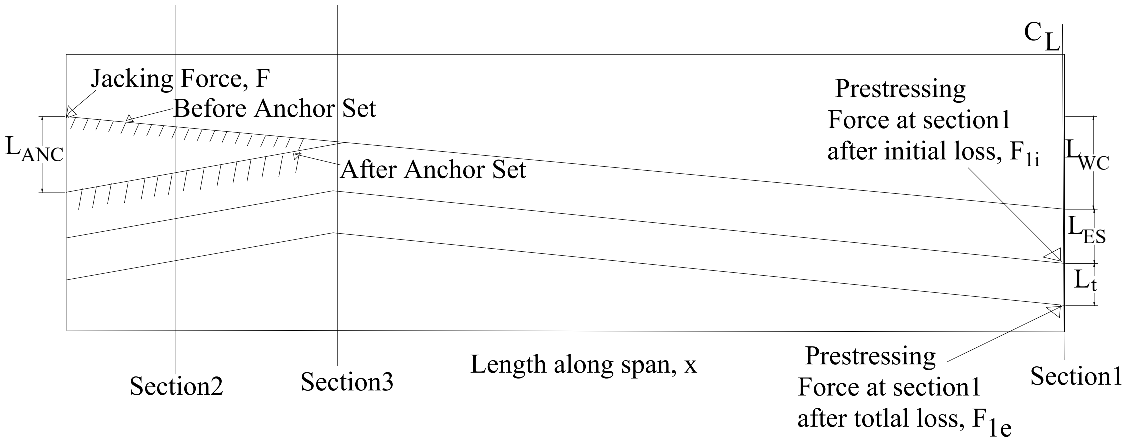


Figure 5.6 Variation of prestressing force along the length of girder

The working stresses at various loading stages are determined as follows:

Initial stage

At positive moment sections:

Stress at top fiber,

$$f_t = -\frac{\eta F_i}{A_{net}} + \frac{\eta F_i e_i}{S_{tnet}} - \frac{M_G}{S_{tnet}}$$

Stress at bottom fiber,

$$f_b = -\frac{\eta F_i}{A_{net}} - \frac{\eta F_i e_i}{S_{bnet}} + \frac{M_G}{S_{bnet}}$$

At negative moment sections:

Stress at top fiber,

$$f_t = -\frac{\eta F_i}{A_{net}} - \frac{\eta F_i e_i}{S_{tnet}} + \frac{M_G}{S_{tnet}}$$

Stress at bottom fiber,

$$f_b = -\frac{\eta F_i}{A_{net}} + \frac{\eta F_i e_i}{S_{bnet}} - \frac{M_G}{S_{bnet}}$$

First loading stage

At positive moment sections:

Stress at top fiber,

$$f_t = -\frac{F_i}{A_{tf}} + \frac{F_i e}{S_t} - \frac{M_P}{S_t}$$

Stress at bottom fiber,

$$f_b = -\frac{F_i}{A_{tf}} - \frac{F_i e}{S_b} + \frac{M_P}{S_b}$$

At negative moment sections:

Stress at top fiber,

$$f_t = -\frac{F_i}{A_{tf}} - \frac{F_i e}{S_t} + \frac{M_P}{S_t}$$

Stress at bottom fiber,

$$f_b = -\frac{F_i}{A_{tf}} + \frac{F_i e}{S_b} - \frac{M_P}{S_b}$$

Second loading stage

At positive moment sections:

Stress at top fiber,

$$f_t = -\frac{F_e}{A_{tf}} + \frac{F_e e}{S_t} - \frac{M_P}{S_t} - \frac{M_C}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{F_e}{A_{tf}} - \frac{F_e e}{S_b} + \frac{M_P}{S_b} + \frac{M_C}{S_{bc}}$$

At negative moment sections:

Stress at top fiber,

$$f_t = -\frac{F_e}{A_{tf}} - \frac{F_e e}{S_t} + \frac{M_P}{S_t} + \frac{M_C}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{F_e}{A_{tf}} + \frac{F_e e}{S_b} - \frac{M_P}{S_b} - \frac{M_C}{S_{bc}}$$

Third loading stage

At positive moment sections:

Stress at top fiber,

$$f_t = -\frac{F_e}{A_{tf}} + \frac{F_e e}{S_t} - \frac{M_P}{S_t} - \frac{M_C}{S_{tc}} - \frac{M_L}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{F_e}{A_{tf}} - \frac{F_e e}{S_b} + \frac{M_P}{S_b} + \frac{M_C}{S_{bc}} + \frac{M_L}{S_{bc}}$$

At negative moment sections:

Stress at top fiber,

$$f_t = -\frac{F_e}{A_{tf}} - \frac{F_e e}{S_t} + \frac{M_P}{S_t} + \frac{M_C}{S_{tc}} + \frac{M_L}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{F_e}{A_{tf}} + \frac{F_e e}{S_b} - \frac{M_P}{S_b} - \frac{M_C}{S_{bc}} - \frac{M_L}{S_{bc}}$$

Fourth loading stage

At positive moment sections:

Stress at top fiber,

$$f_t = -\frac{1}{2}\frac{F_e}{A_{tf}} + \frac{1}{2}\frac{F_e e}{S_t} - \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{1}{2}\frac{F_e}{A_{tf}} - \frac{1}{2}\frac{F_e e}{S_b} + \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{bc}}$$

At negative moment sections:

Stress at top fiber,

$$f_t = -\frac{1}{2}\frac{F_e}{A_{tf}} - \frac{1}{2}\frac{F_e e}{S_t} + \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{tc}}$$

Stress at bottom fiber,

$$f_b = -\frac{1}{2}\frac{F_e}{A_{tf}} + \frac{1}{2}\frac{F_e e}{S_b} - \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{bc}}$$

5.5.2 Ultimate flexural strength constraints

The ultimate flexural strength constraints for the precast section and composite section are considered as:

$$0.0 \le M_{pu} \le \varphi M_{pn} \tag{5.23}$$
$$0.0 \le M_{cu} \le \varphi M_{cn} \tag{5.24}$$

where, M_{pu} and M_{cu} are factored bending moments; φM_{pn} and φM_{cn} are flexural strength of the precast and composite section respectively.

To calculate the flexural strength of the composite section at position of positive moments, following four cases are considered and detail calculations are tabulated in Table 5.5.

Case 1: Compression block remains within the deck slab.

Case 2: Compression block remains within the Top flange.

Case 3: Compression block remains within the Top flange transition area.

Case 4: Compression block falls in web (Flanged section calculation is used assuming T shape stress block).

Table 5.5 Flexural strength calculations

Case	Equations
Case 1:	$b = EFW; \rho = \frac{A_s}{bd}$ $f_{su} = f_{pu}\left(1 - \frac{\gamma^*}{\beta}\rho\frac{f_{pu}}{f'_{cdeck}}\right); z = \frac{A_s f_{su}}{0.85 f_{cdeck}\, b}$ $\boldsymbol{M_n} = A_s f_{su}\left(d - \frac{z}{2}\right)$
Case 2:	$b = \left(\frac{f_{cdeck}}{f_c^{'}} EFW \times t + \frac{TF_w \times TF_t}{t + TF_t}\right); \rho = \frac{A_s}{bd}$ $f_{su} = f_{pu}\left(1 - \frac{\gamma^*}{\beta}\rho\frac{f_{pu}}{f_c^{'}}\right); z = \frac{A_s f_{su}}{0.85 f_c b}$ $\boldsymbol{M_n} = A_s f_{su}\left(d - \frac{z}{2}\right)$
Case 3:	$b = \left(\frac{f_{cdeck}}{f_c^{'}} EFW \times t + \frac{TF_w \times TF_t + \frac{2(TF_w - TFS_w)}{2} \times TFS_t}{t + TF_t + TFS_t}\right)$ $\rho = \frac{A_s}{bd}$ $f_{su} = f_{pu}\left(1 - \frac{\gamma^*}{\beta}\rho\frac{f_{pu}}{f_c^{'}}\right); z = \frac{A_s f_{su}}{0.85 f_c^{'} b}$ $\boldsymbol{M_n} = A_s f_{su}\left(d - \frac{z}{2}\right)$
Case 4:	$A_{sw} = A_s - \frac{0.85 * f_c * (b - W_w) \times (t + TF_t + TFS_t)}{f_{su}}$ $\rho = \frac{A_{sw}}{W_w d}$ $\boldsymbol{M_n} = 0.85 f_c (b - W_w)(t + TF_t + TFS_t)\left(d - \frac{t + TF_t + TFS_t}{2}\right)$ $+ A_{sw} + f_{su} d\left(1 - 0.6\frac{\rho f_{su}}{f_c^{'}}\right)$

To cal culate t he f lexural s trength o f th e g irder section at pos ition of ne gative moments, corresponding equations that are used are as follows:

$$b = b_w; \rho = \frac{A_s}{b_w d}$$

$$f_{su} = f_{pu}\left(1 - \frac{\gamma^*}{\beta}\rho\frac{f_{pu}}{f'_c}\right); z = \frac{A_s f_{su}}{0.85\, f'_c b_w}$$

$$\boldsymbol{M_n} = A_s f_{su}\left(d - \frac{z}{2}\right)$$

5.5.3 Ductility (maximum and minimum prestressing steel) constraints

The maximum prestressing steel constraint for the composite section is given below:

$$0 \leq \omega \leq \omega_u \tag{5.25}$$

Where, Reinforcement index, $\omega = \frac{\rho f_{su}}{f_c'}$ and ω_u= Upper limit to reinforcement index = $0.36\beta_1$

The constraints which limit the minimum value of reinforcement are,

$$1.2\ M_{cr}^* \leq \varphi M_n \tag{5.26}$$

Where, for composite girder,
At position of maximum positive moment,

$$M_{cr}^* = (f_r + f_{pe})S_{bc} - M_{P1}\left(\frac{S_{bc}}{S_b} - 1\right) \tag{5.27}$$

$$f_{pe} = \frac{F_e}{A_{tf}} + \frac{F_e e_{c1}}{S_b} \tag{5.28}$$

where, f_r = modulus of rupture; f_{pe}= compressive stress in concrete due to effective prestress forces only (after allowance for all prestress losses) at extreme fiber of section where tensile stress is caused by externally applied load; S_b, S_{bc} = Section Modulus of bottom fiber of transformed precast & composite section respectively; e_{c1} = eccentricity of composite section at position of maximum positive moment; M_{P1} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum positive moment;

At position of maximum negative moment,

$$M_{cr}^* = (f_r + f_{pe})S_{tc} - M_{P2}\left(\frac{S_{tc}}{S_t} - 1\right) \tag{5.29}$$

$$f_{pe} = \frac{F_e}{A_{tf}} + \frac{F_e e_{c2}}{S_t} \tag{5.30}$$

where, f_r = modulus of rupture; f_{pe}= compressive stress in concrete due to effective prestress forces only (after allowance for all prestress losses) at extreme fiber of section where tensile stress is caused by externally applied load; S_t, S_{tc} = Section Modulus of top fiber of transformed precast & composite section respectively; e_{c2} = eccentricity of composite section at position of maximum negative moment; M_{P2} = Non-composite dead load moment or Moment due to girder self weight, cross girder and deck slab at position of maximum negative moment;

For deck slab,

$$M^*_{crslab} = f_r S_{deck} \tag{5.29}$$

5.5.4 Ultimate shear strength and horizontal interface shear constraints

The ultimate shear strength is considered at two sections, section at the end of transition zone and section where the prestress is maximum and the related implicit constraint is defined as,

$$\varphi V_s = (V_u - \varphi V_c) \leq 0.666\sqrt{f_c'}\, W_w d_s \tag{5.30}$$

where, V_u = factored shear at a section, V_c = the concrete contribution taken as lesser of flexural shear, V_{ci} and web shear, V_{cw},, V_s = shear carried by the steel in kN. These two shear capacity are determined according to AASHTO specification.

Composite sections are designed for horizontal shear at the interface between the precast beam and deck and the related constraint is:

$$V_u \leq \varphi V_{nh} \tag{5.31}$$

Where, V_{nh} = nominal horizontal shear strength.

5.5.5 Deflection constraint

Deflection at mid span due to initial prestress (For parabolic tendon profile) is computed as:

$$\Delta_{PT} = \frac{13}{136}\sum_{i=1}^{i=N_T} \eta F_{1i} h_i \frac{L^2}{E_{ci} I_{net}} + \frac{1}{8}\sum_{i=1}^{i=N_T} \eta F_{1i} e_i \frac{L^2}{E_{ci} I_{net}} \tag{5.32}$$

Where, h_i, e_i are the sag and eccentricity of the ith tendon respectively.

Deflection due to dead load:

$$\Delta_{DL} = \frac{13}{136} M_G \frac{L^2}{EI} \tag{5.33}$$

$$\text{Initial camber} = \Delta_{PT} - \Delta_{DL} \tag{5.34}$$

Deflection due to live load (AISC Mkt 1986):

$$\Delta_{LL} = \frac{324}{EI_c} P_T (L^3 - 555L + 4780) \tag{5.35}$$

The live load deflection constraint is as follows:

$$\Delta_{LL} \leq L/800 \tag{5.36}$$

5.5.6 End section tendon eccentricity constraint

Eccentricity of tendons at the end section becomes a constraint because eccentricity has to remain within the kern distances of the section to avoid extreme fiber tension both at initial stage and at final stage. The following constraint limits the tendon eccentricity at end section so that the eccentricity remains within the kern distances,

$$\frac{G_d}{6} + 0.25\sqrt{f_{ci}}\frac{A_4 G_d}{6\,F_{4i}} \leq e_4 \leq \frac{G_d}{6} + 0.5\sqrt{f_c'}\frac{A_4 G_d}{6F_{4e}} \tag{5.37}$$

5.5.7 Lateral stability constraint

The following constraint according to PCI (PCI 2003) limits the safety and stability during lifting of long girder subject to roll about weak axis,

$$FS_c = \frac{1}{\frac{z_o}{y_r} + \frac{\theta_i}{\theta_{max}}} \geq 1.5 \tag{5.38}$$

Where, FS_c= factor of safety against cracking of top flange when the girder hangs from lifting loop.

5.5.8 Deck slab constraints

The co nstraint co nsidered f or d eck sl ab t hickness acco rding t o d esign cr iteria o f ODOT (ODOT 2000) is,

$$t \geq \frac{S_d + 17}{3} \tag{5.39}$$

The constraint which limit the required effective depth for deck slab is,

$$d_{min} \leq d_{req} \leq d_{prov} \tag{5.40}$$

Where, S_d = effective slab span in feet = $S\text{-}TF_w/2$; t = slab thickness in inch.

5.6 Stiffness method

5.6.1 General

One o f t he b asic ad vantages o f st iffness method i s t hat w hatever b e t he st ructural idealization the main steps of stiffness method are always same and as stated below:

1. Identify t he unknow n displacement f or e ach joint. T hat is, de termine t he degree of kinematic indeterminacy.
2. Make th e s tructure k inematically determinate b y re straining a ll d egrees o f freedom.
3. Apply l oads an d c alculate j oint f orces corresponding t o e ach D egree of Freedom (D.O.F.). That is finally obtain the member force vector [Pm].
4. Apply unknow n di splacements on e a t a t ime a nd c alculate j oint forces corresponding to each D.O.F. That is, calculate the stiffness terms and finally obtain the stiffness matrix [k].
5. Write e quilibrium e quations c orresponding t o each D .O.F. i .e. t he S tiffness Equations. Solve for unknown displacements.
6. Superimpose the effects of loads and displacements to obtain stress resultants and reactions.

5.6.2 Computer Application of Stiffness Method

Although the step by step approach is convenient as a common method to different structures, the method is not quite yet appropriate for writing programs to solve problems using stiffness method. For computer application the same stiffness method is used following a slightly different sequence.

First stiffness matrix of each member is derived. Then they are assembled to form a global stiffness matrix of the whole structure. Then global force vectors are derived. At last boundary conditions are imposed. Stiffness equations are then solved to determine the unknowns. The process of solving indeterminate structure following this approach will be demonstrated by a beam problem (Figure 5.7).

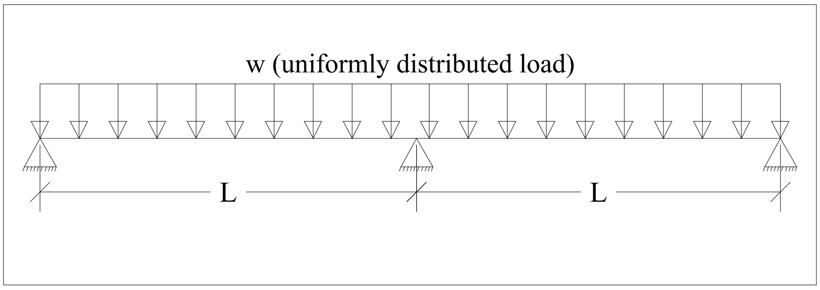


Figure 5.7 Two-span continuous indeterminate beam with UDL

First a single beam member without any boundary condition is considered:

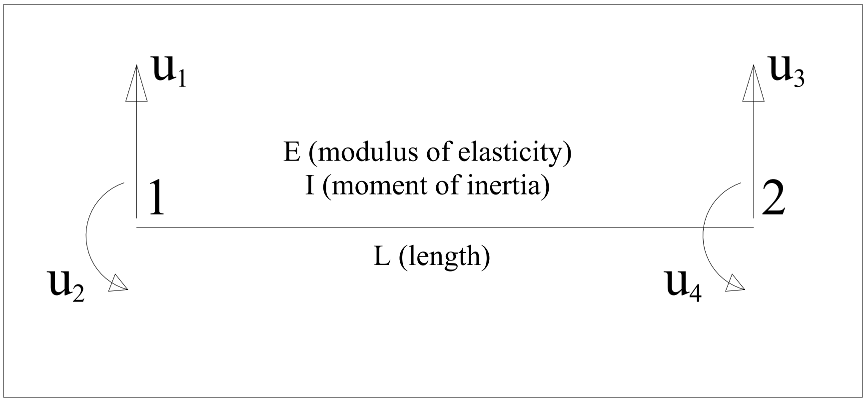


Figure 5.8 Single beam member without any boundary condition

The beam member has got four degrees of freedom (local), namely, one translation (u_1) and one rotation (u_2) at node 1 and one translation (u_3) and one rotation (u_4) at node 2 (Figure 5.8). The stiffness matrix (4x4) of the member will be:

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

Now, for the time being, if loading and boundary conditions are ignored, the present structure is basically an assemblage of two beam members connected at a node (Figure 5.9).

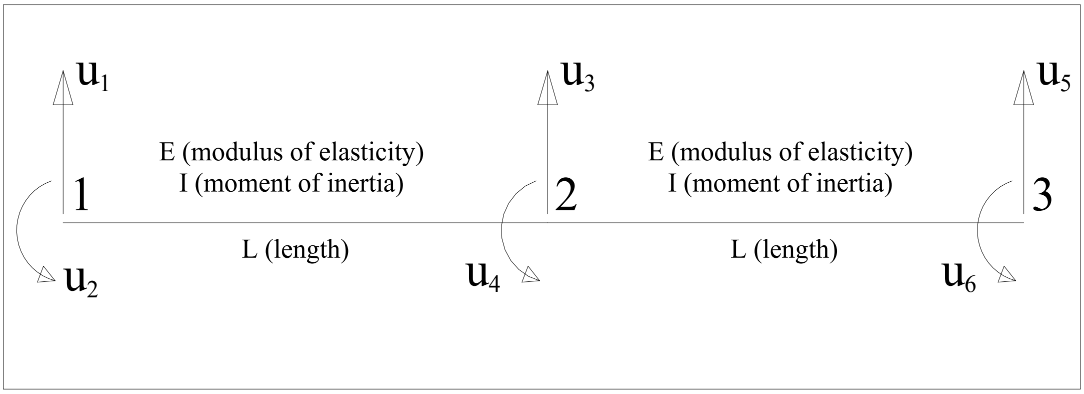


Figure 5.9 Assemblage of two beam members without boundary conditions

In total the structure has 3 nodes and 6 (global) corresponding degrees of freedom. The members have following attributes assigned to them:

Member No.	**i-node**	**j-node**	**Length**	**I**	**E**
1	1	2	L	I	E
2	2	3	L	I	E

Using the last three attributes member stiffness matrix for each member can easily be calculated. For the first member its local first (i) and second (j) node, respectively, corresponding to the global node 1 and 2. Thus the four rows and columns of the member stiffness matrix of member 1 will fill up the first four rows and columns of global stiffness matrix.

However for member 2, its first (i) and second (j) local node, respectively, corresponding to the global node 2 and 3. Thus the four rows and columns of the member stiffness matrix will fill up 3rd (2i-1), 4th (2i), 5th (2j-1) and 6th (2j) rows and columns of the global stiffness matrix. Thus the 6x6 global stiffness matrix will be:

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

The process of deriving global stiffness matrix from member stiffness matrices is called assembling. The global stiffness matrix for beam problem becomes banded.

Here it is easily noticeable that, now the same structure can have different loading and support conditions without any need for recalculating the stiffness matrix.

Now the loading will be considered. Member loads are considered as assignments to individual members and joint loads are considered as assignments to individual degrees of freedom (Figure 5.10).

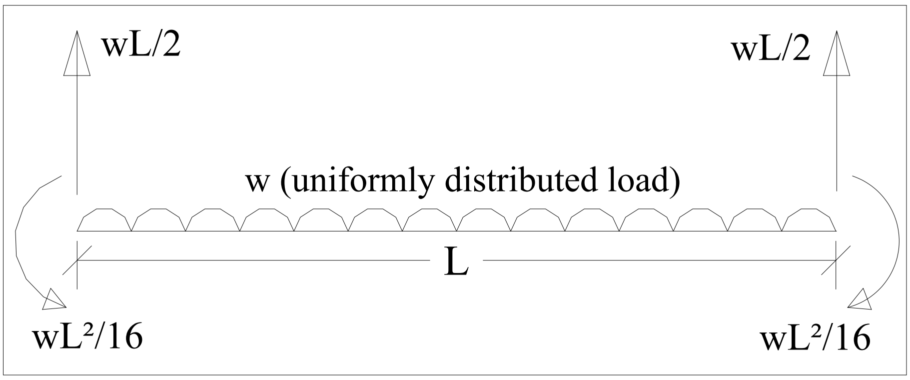


Figure 5.10 Assigning loads to degrees of freedom

The local member force matrix (4x1) for a single beam member will be:

$$\{Pm^1\} = \begin{bmatrix} \frac{wL}{2} \\ \frac{wL^2}{16} \\ \frac{wL}{2} \\ -\frac{wL^2}{16} \end{bmatrix}$$

After assem blage o f two s uch local member f orce m atrices o f t wo si ngle b eam members, the global member force matrix (6x1) will be found:

$$\{Pm\} = \begin{bmatrix} \frac{wL}{2} \\ \frac{wL^2}{16} \\ wL \\ 0 \\ \frac{wL}{2} \\ -\frac{wL^2}{16} \end{bmatrix}$$

Since there are no loads on joints, the joint force matrix {Pj} will be a null-vector:

$$\{Pj\} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Considering degrees of freedom matrix {u} to be:

$$\{u\} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

The stiffness equations then becomes,

{Pm} + [K]*{u} = {Pj}

$$\begin{bmatrix} \frac{wL}{2} \\ \frac{wL^2}{16} \\ wL \\ 0 \\ \frac{wL}{2} \\ -\frac{wL^2}{16} \end{bmatrix} + \begin{bmatrix} \frac{12EI}{L^3} & \frac{6EI}{L^2} & \frac{-12EI}{L^3} & \frac{6EI}{L^2} & 0 & 0 \\ \frac{6EI}{L^2} & \frac{4EI}{L} & \frac{-6EI}{L^2} & \frac{2EI}{L} & 0 & 0 \\ \frac{-12EI}{L^3} & \frac{-6EI}{L^2} & \frac{24EI}{L^3} & 0 & \frac{-12EI}{L^3} & \frac{6EI}{L^2} \\ \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & \frac{8EI}{L} & \frac{-6EI}{L^2} & \frac{2EI}{L} \\ 0 & 0 & \frac{-12EI}{L^3} & \frac{-6EI}{L^2} & \frac{12EI}{L^3} & \frac{-6EI}{L^2} \\ 0 & 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & \frac{-6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It is noticeable that the diagonal elements of the stiffness matrix are as usual positive and the matrix is symmetric. It is to remind that, support conditions have not yet considered in the stiffness equations. Thus, if these equations are solved, no unique solution can be found. That is, all the six stiffness equations above are not linearly independent.

Now, the support conditions will be considered. There are two ways of doing that. One method is easier but requires more memory space and computation time and the other method is more difficult but takes less memory and computation time. In this thesis, the easier procedure is used for simplicity.

In order to impose boundary condition it is required to know, number of restrained nodes.

No. of restrained nodes, NRN=3.

In order to make u_i=0,

(i) Make all the elements of the ith row and ith column of the assembled structure stiffness matrix = 0;

(ii) Make diagonal members of the stiffness matrix, K_{ii}= 1;

(iii) Make corresponding element of member force matrix, P_i= 0;

In this way, boundary conditions can be imposed and Modified Stiffness Matrix and Modified Stiffness Equation can be obtained.

So, finally the stiffness equations for the free degrees of freedom will be:

$$\begin{bmatrix} \frac{wL^2}{16} \\ 0 \\ -\frac{wL^2}{16} \end{bmatrix} + \begin{bmatrix} \frac{4EI}{L} & \frac{2EI}{L} & 0 \\ \frac{2EI}{L} & \frac{8EI}{L} & \frac{2EI}{L} \\ 0 & \frac{2EI}{L} & \frac{4EI}{L} \end{bmatrix} \begin{bmatrix} u_2 \\ u_4 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

5.7 Constructing Influence Line for Indeterminate Structure

To find out design moment and shear, the statically indeterminate structure had to solve for hundreds of times. For solving the indeterminate structure, stiffness method was used. To make the basic stiffness method applicable to computer aided analysis, 'computer application of stiffness method' was used with required sequence of logic.

To find out the live load and impact shear force and moment, it was necessary to construct influence lines for different sections. No general equation of influence line has been used; rather the coordinate values of different points of the influence line are determined using the basic stiffness concept. The general simple concept used to construct the influence lines is described below:

The total length of the bridge is divided into a series of 0.25 meter long segments. At every 0.25 meter, a node/ coordinate have been considered. For constructing influence line, a 1-kip load was placed at each of these nodes and for that 1-kip load, the whole structure is solved using stiffness method. After solving the whole structure, the shear force and bending moment values at every 0.25 meter apart nodes has been stored in the columns of two matrices of sufficient size. While storing the shear force and moment values, it was assured that, the nodal values were placed column wise. It is evident from the concept / definition of influence line that the n th row of those matrices will give the coordinate values of the influence line for shear force and moment for that particular section (i.e. n^{th} section) (Fig 5.11).

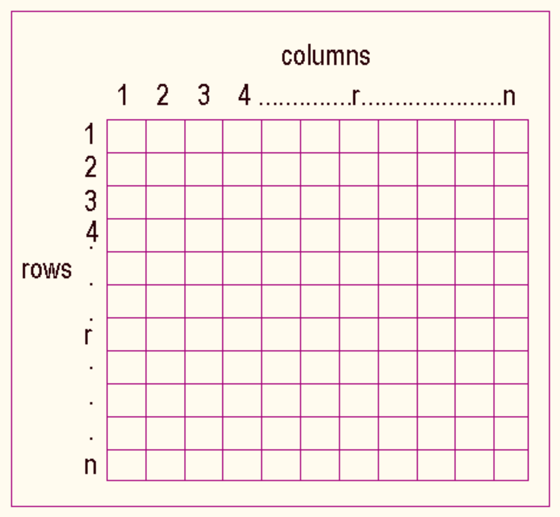


Fig. 5.11 Constructing Influence Line Matrix

For example, the coordinates of the influence line of shear force for the 60 meter long two-span continuous girder came out to be:

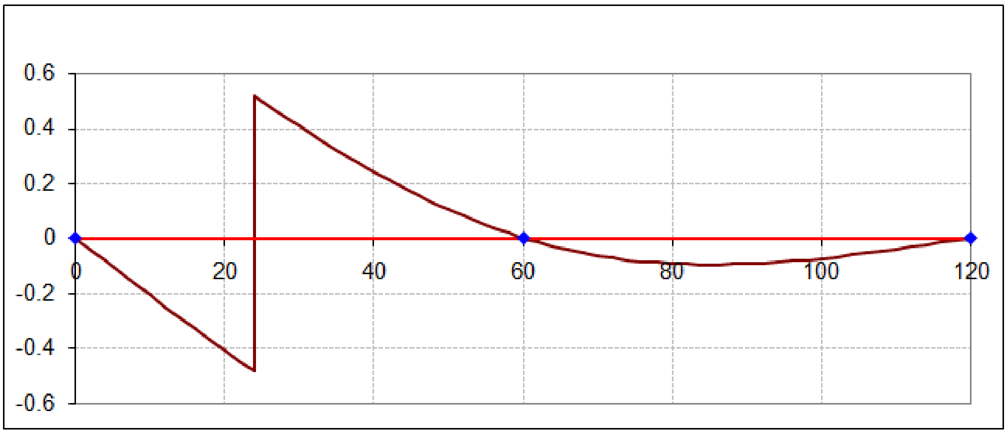


Fig. 5.12 Influence Line for Shear at section 1 (at 0.4L distance from left support) of first span for two-span continuous girder with 60 meter span length

For the 60 meter long two-span continuous girder, influence line for shear at section 1 which is at 0.4L (i.e. 24 ft) distance from left support has been plotted (Fig 5.12) using the values of shear force at every 0.25 m nodes along the span. In accordance with the concept with which the influence line matrix for shear has been developed, it is evident that, the 96th row (24X4) of the influence line matrix will have the 480 nos. (120X4=480) of nodal values (Fig 5.13) of the influence line.

	1	2	3	…	96	97	98	99	…	239	240	241	242	…	479	480
1																
2																
3																
⋮																
⋮																
95																
96	-0.02 kN-m	-0.04	-0.06	…	-0.48 / 0.52	0.50	0.47	0.44	…	0.03	0.00	-0.01	-0.02	…	-0.01	0.00
97																
98																
⋮																
⋮																
478																
479																
480																

Fig. 5.13 Coordinates of Influence Line for Shear at section 1 (at 0.4L distance from left support) of first span for two-span continuous girder with 60 meter span length

Similarly, the coordinates of the influence line of bending moment for the 60 meter long two-span continuous girder came out to be:

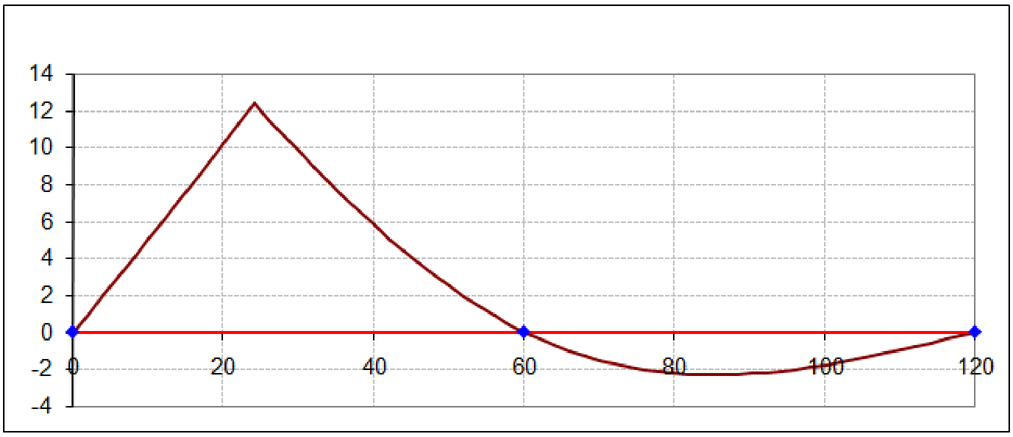


Fig. 5.14 Influence Line for Moment at section 1 (at 0.4L distance from left support) of first span for two-span continuous girder with 60 meter span length

While determining the design value of live load shear / moment, the wheel loads had to be placed on the influence lines and the pick value had to be found out. It is to be noted that, a HL-93 truck has the spacing of 4.3 meter between the front and middle wheel, and a spacing of 4.3 meter between the middle and rear wheel. Both these two numbers, i.e. 4.3 m and 4.3 m are almost dividable by 0.25 with a fraction of 0.05 meter. So, it will not be too much erroneous if the wheel loads are placed only on the nodes described before. For any particular position of the truck, the shear/ moment can be easily found by multiplying the wheel loads with corresponding values of influence line respective coordinates.

5.8 Linking Optimization Problem with EVOP and Solve

In the present optimization problem a large number of design variables and constraints are associated. The design variables are classified as combination of continuous, discrete and integer variables. Expressions for the objective function and the constraints are non linear functions of these design variables. So the optimal design problem becomes highly nonlinear and non-convex having multiple local minima which requires an optimization method to derive the global optimum. As a result the global optimization algorithm named EVOP (Ghani 1989) is used.

The algorithm EVOP requires three user written functions the objective function, the explicit constraint function and implicit constraint function, some user input control parameters and a starting point inside the feasible space (Figure 5.15). Given the coordinates of a feasible point in an N-dimensional space the objective function calculates the functional value. Explicit constraint function evaluates the upper and the lower limits of the explicit constraints. Implicit constraint function evaluates the implicit constraints values and their upper and lower limits. The input control parameters with their default values and ranges are, α = 1.2 (1.0 to 2.0); β = 0.5 (0 to 1.0); $\Delta = 10^{-12}$; γ = 2.0 (greater than 1.0 to upwards), $\Phi = 10^{-14}$(10^{-16} to 10^{-8}) ($\Phi = 10^{-12}$ will yield higher accuracy for convergence compared to $\Phi = 10^{-14}$) and $\Phi_{cpx} = 10^{-9}$ (10^{-16} to 10^{-8}).

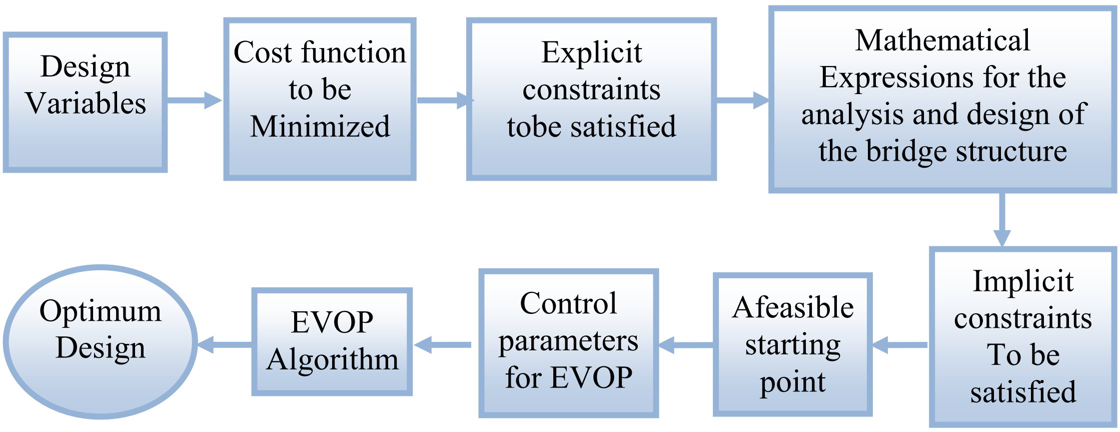


Figure 5.15 Steps for Optimization Problem Formulations and Linking with Optimization Algorithm (EVOP) (Rana, 2010)

The other parameters relevant to the usage of the program EVOP are as follows.

IJK --- For first entry, this variable should always be set to 1. It will subsequently be changed by 'EVOP'.

K --- Number of 'complex' vertices. If 'n' is the dimension of the parameter space, for n <= 5, k= 2n; and for n > 5, k >= (n + 1).

KNT --- Number of consecutive times the objective function is called after which tests are conducted for convergence. (Typically 25).

LIMIT --- Maximum number of times the three functions: the objective function, the explicit constraint function and the implicit constraint function can be collectively called.

NRSTRT --- Number of automatic restart of EVOP to check that the previously obtained value is the global minimum. If NRSTRT = 5, the EVOP program will execute 5 times. For first time execution a starting point of the complex inside the feasible space has to be given. For further restart the complex is generated taking the coordinates of the previous minimum (values obtained from previous execution of EVOP) as the starting point of the complex.

IER --- Error flag.

= 1 indicates user provided starting point is violating upper limit of an explicit constraint.

= 2 indicates user provided starting point is violating lower limit of an explicit constraint.

= 3 indicates user provided starting point is violating upper limit of an implicit constraint.

= 4 indicates user provided starting point is violating the lower limit of an implicit constraint.

= 5 indicates randomly generated (k – 1) tests points not obtainable in the 'LIMIT' to which the three functions can be collectively called.

= 6 indicates minimum of the objective function not obtainable within the desired accuracy of convergence. The results are those obtained after exceeding 'LIMIT'.

= 7 indicates final 'complex' has not reduced its size to satisfy convergence test2. Results are those obtained after exceeding 'LIMIT'.

= 8 indicates minimum of the objective function has been located to the desired degree of accuracy to satisfy both convergence tests.

XMAX(N) --- Array of dimension 'N' containing the upper limits of the explicit constraints. They are calculated and supplied by the explicit constraint function for a given trial point provided by 'EVOP'.

XMIN(N) --- Array of dimension 'N' containing the lower limits of the explicit constraints. They are calculated and supplied by the explicit constraint function for a given trial point provided by 'EVOP'.

XT(N) --- Array of dimension 'N' containing the coordinates of the trial point. On first entry 'XT(N)' contains the feasible trial point, and at the end of minimization it returns with the coordinates of the minimum located.

XX(NIC) --- Array of dimension 'NIC' containing the implicit constraint function values. They are calculated and supplied by the implicit constraint function, for a given trial point 'XT(N)' provided by 'EVOP'.

XXMAX(NIC) --- Array of dimension 'NIC' containing the upper limit of the implicit constraints. They are calculated and supplied by the implicit constraint function, for a given trial point 'XT(N)' provided by 'EVOP'.

XXMIN(NIC) --- Array of dimension 'NIC' containing the lower limit of the implicit constraints. They are calculated and supplied by the implicit constraint function, for a given trial point 'XT(N)' provided by 'EVOP'.

Values for 'α', 'β', 'γ', 'Φ' and 'Φ_{cpx}'

(i) Initially 'NRSTRT' has to be set to a high integer value, say 10 or 20.

(ii) Initially for low convergence accuracy, the value of $\Phi = 10^{-14}$ has to be set and the value of $\Phi_{cpx} = 10^{-9}$ has to be set.

(iii) **'α', 'β', 'γ'** have to be set to their default values of 1.2, 0.5 and 2.0 respectively, and the program has to be run.

(iv) Keeping **β** and **'γ'** fixed, **α** has to be varied from a value greater than 1.0 to a value less than 2.0 for convergence 'IER = 8', with lowest number of function evaluation 'NFUNC', and lowest function value 'F'.

(v) 'Φ' has to be increased upto 10^{-10} for double precision in steps for tighter convergence, and 'Φ' has to be set the highest value that would still yield 'IER = 8'. Note: **α** is the most sensitive parameter.

(vi)Keeping **α** and γ fixed β has to be varied above 0.0 to less than 1.0 for the criterion set out in (iv) above.

(vii) Keeping α and β fixed γ has to be varied from 2.0 up wards for the criterion set out in (iv) above.

(viii) Repeat from step (iv) only if lower values of 'NFUNC' and 'F' are required.

(ix) Φ_{cpx} has to be changed from a value two decades higher to two decades lower compared to Φ and observe the effects on 'NFUNC' and 'F'. Φ_{cpx} has to be chosen for least 'NFUNC' and 'F'.

(x) Using optimum 'XT(N)' and corresponding 'XMAX(N)', 'XMIN(N)', 'XXMAX(NIC)', 'XXMIN(NIC)', from (viii) above, the program has to be run with same values for **'α', 'β', 'γ', 'Φ' and 'Φ_{cpx}'**. Whether a better minimum is obtained has to be checked.

A computer program coded in C++ (Appendix A) is used to input control parameters and to define three functions: an objective function, an explicit constraint function and an implicit constraint function. First the values of the control parameters are assigned with their default values and other input parameters are set to specific numerical values. These other input parameters for the present optimization problem are: number of complex vertices, K = 15; maximum number of times the three functions can be collectively called, limit = 100000; dimension of the design variable space, N = 14; number of implicit constraint, NIC = 73 and number of EVOP restart, NRSTRT = 10.

Determination of a feasible starting point (the values of design variables corresponding to the feasible point satisfy all the explicit and implicit constraints) is simple. Before calling subroutine EVOP a random point satisfying all explicit constraints is generated and tested for satisfying all implicit constraints. If these constraints are also satisfied then control is passed to the function EVOP. Otherwise the process is repeated till a feasible starting point is found. Next the function EVOP is called. Next suitable values of the control parameters are obtained by varying the parameters within the range sequentially and setting Φ to highest value that would still yield convergence and number of function evaluation becomes lowest with least function value. The program is rerun using optimum design variables obtained previously as starting point with same values of control parameters and checked whether a better minimum is obtained.

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 General

The cost optimum design method has been performed for **40 m, 60 m, 80 m** girder span and for 3 Lane or 4 Lane bridges. AASHTO HL-93 live load is considered for each case and superimposed dead loads are according to AASHTO also. Optimum design is dependent on the design constant parameters i.e. unit cost of materials, labor, fabrication and installation, concrete strength, strand size, anchorage system etc. As different design constant parameters will result in different optimum design, the cost optimum design method has been performed for two types of girder concrete strengths (28 days) **40 MPa** and **50 MPa** and for three different unit costs of the materials as shown in Table 6.1 so that the variation in design with respect to change in the parameters can be observed. Concrete strength at initial stage is taken as 75% of 28 days strength. Deck slab concrete strength is considered as **25 MPa**. Ultimate strength of prestressing steel and yield strength of ordinary steel are considered as 1861 MPa and 410 MPa respectively. Freyssinet C-Range anchorage system is used for posttensioning tendons consisting of 15.2 mm diameter 7-wire strands. Unit cost of girder concrete and deck slab concrete is considered fixed and the unit costs of steels and anchorage systems are varied such that in Cost2, these costs are two times those in Cost1 and in Cost3, these costs are three times those in Cost1.

Table 6.1 Relative cost parameters used for cost minimum design
(As per RHD schedule of rates 2015)

Item	Unit	Cost1 (C1) (BDT)	Cost2 (C2) (BDT)	Cost3 (C3) (BDT)
Precast girder concrete-including equipment and labor	per m^3(50MPa)	19,500	19,500	19,500
(UP_{GC})	per m^3(40MPa)	13,500	13,500	13,500
Girder formwork (UP_{GF})	per m^2	550	550	550
Cast-in-place deck concrete(UP_{DC})	per m^3	8,000	8,000	8,000
Deck formwork-equipment and labor(UP_{DF})	per m^2	530	530	530
Girder posttensioning-tendon, equipment and labor(UP_{PS})	per ton	1,20,000	240,000	360,000
Anchorage set(UP_{ANC})	per set	7,000	14,000	21,000
Metal sheath for duct(UP_{SH})	per lin. meter	90	180	270
Mild steel reinforcement for deck and web in girder(UP_{OS})	per ton	60,000	1,20,000	180,000

6.2 Parametric Studies

6.2.1 Optimum design for 40 m double span continuous girder

Table 6.2 Optimum values of design variables for 3 Lane 40 m double span continuous girder and Concrete strength = 50 MPa

Cost	S (m)	Gd (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	4.0	1700	450	75	50	325	260	150	9	4	255	0.68	430	27
C2	4.0	2200	750	75	50	325	235	150	9	4	265	0.55	720	30
C3	4.0	2250	1250	75	50	300	195	150	8	3	285	0.48	680	50

Table 6.3 Optimum values of design variables for 3 Lane 40 m double span continuous girder and Concrete strength = 40 MPa

Cost	S (m)	Gd (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	4.0	1850	725	75	50	310	265	160	9	4	245	0.67	425	27
C2	4.0	2250	950	75	50	325	150	160	8	4	245	0.52	795	37
C3	4.0	2625	1375	75	50	305	130	145	7	3	290	0.45	810	29

Table 6.4 Cg of tendons from bottom fiber of girder in the optimum design

	Girder concrete strength = 50 MPa									Girder concrete strength = 40 MPa								
	Y_1	Y_2	Y_3	Y_{end}	Y_5	Y_6	Y_7	Y_8	AS	Y_1	Y_2	Y_3	Y_{end}	Y_5	Y_6	Y_7	Y_8	AS
C1	137	162	590	821	1525	1325	677	1425	260	130	171	625	910	1539	1348	654	1430	327
C2	115	186	789	1223	2003	1823	987	1906	300	119	176	683	1075	1877	1624	874	1756	290
C3	112	156	610	946	1950	1745	895	1877	260	104	176	655	1137	2013	1823	983	1932	286

Optimum d esign for 4 0 m double s pan c ontinuous girder (Girder co ncrete strength = 50 MPa)

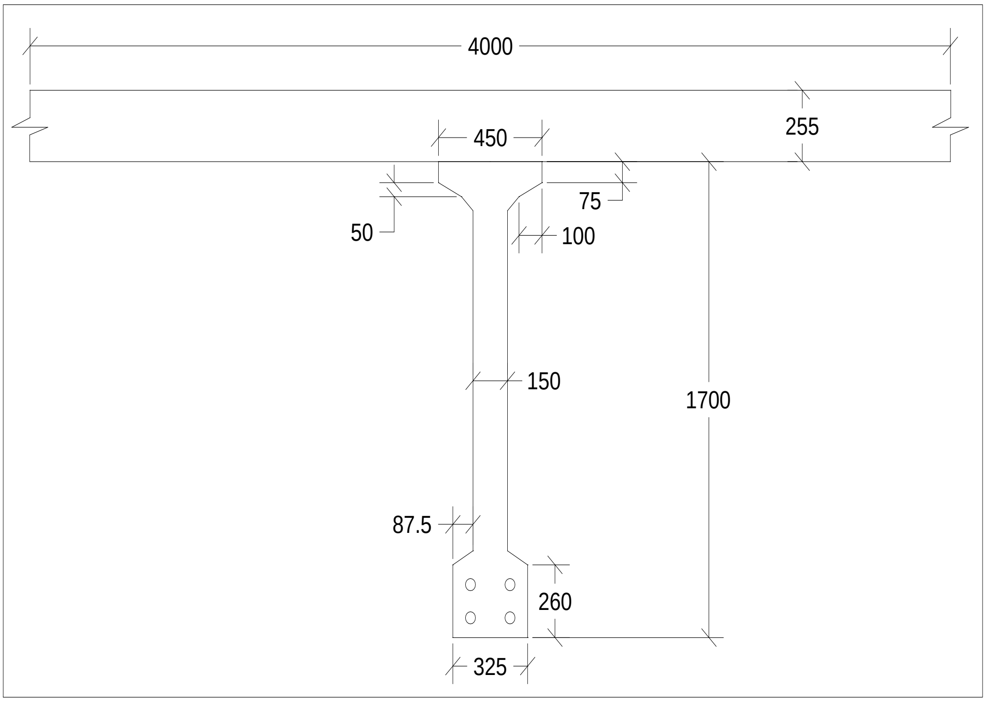


Figure 6.1(a) Optimum design for double span cont. girder at section 1 for Cost1

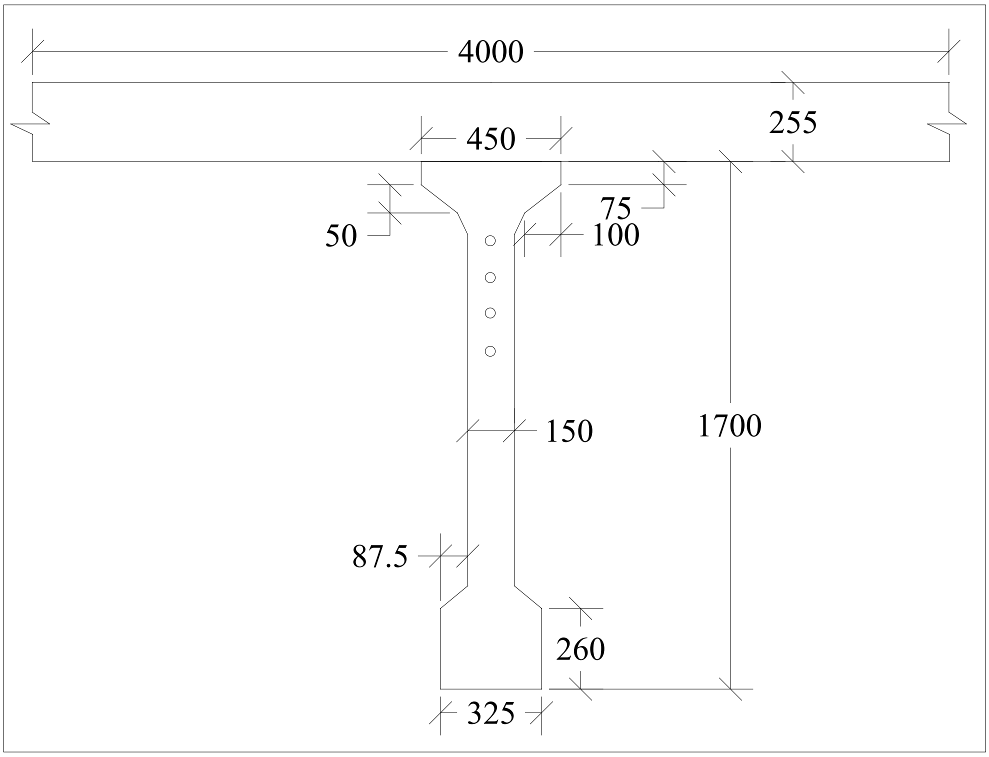


Figure 6.1(b) Optimum design for double span cont. girder at section 5 for Cost1

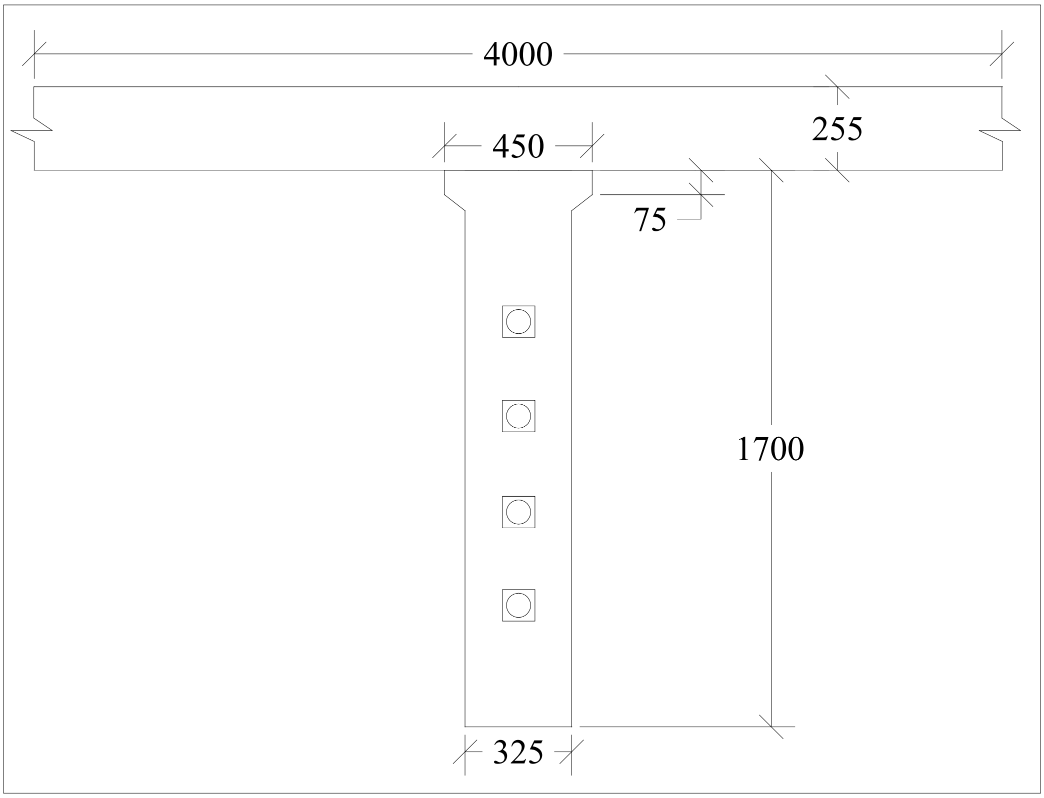


Figure 6.1(c) Optimum design for double span cont. girder at section 4 for Cost1

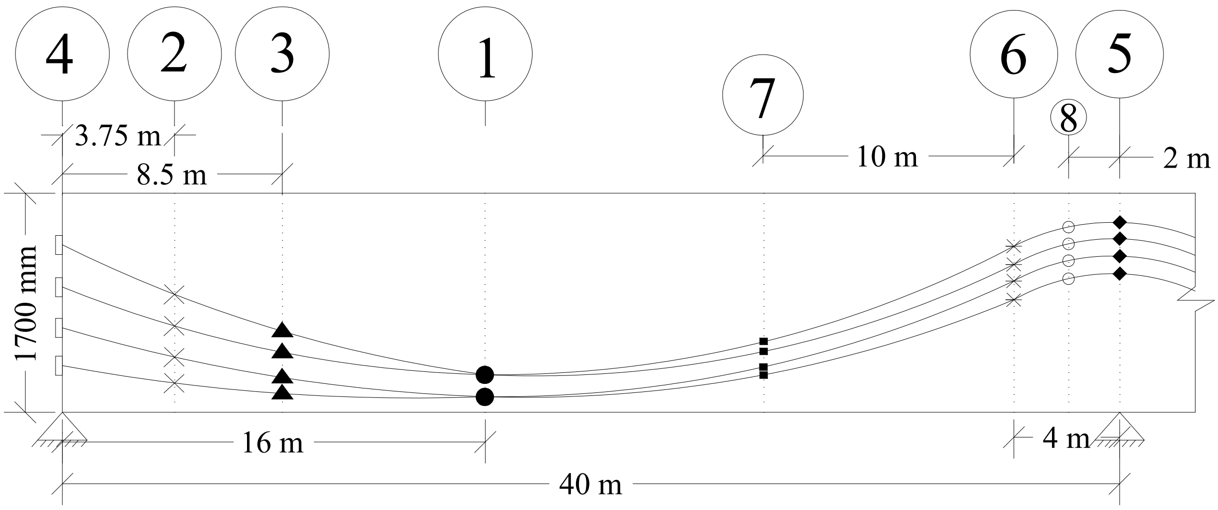


Figure 6.1(d) Optimum design for double span continuous girder for Cost1

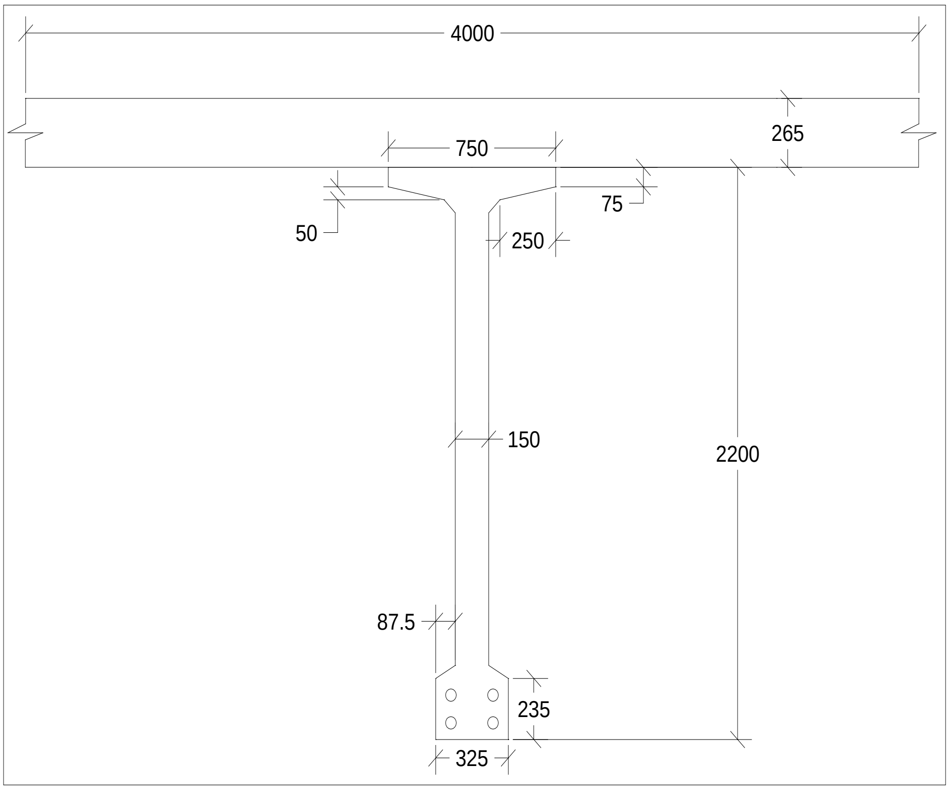


Figure 6.2(a) Optimum design for double span cont. girder at section 1 for Cost2

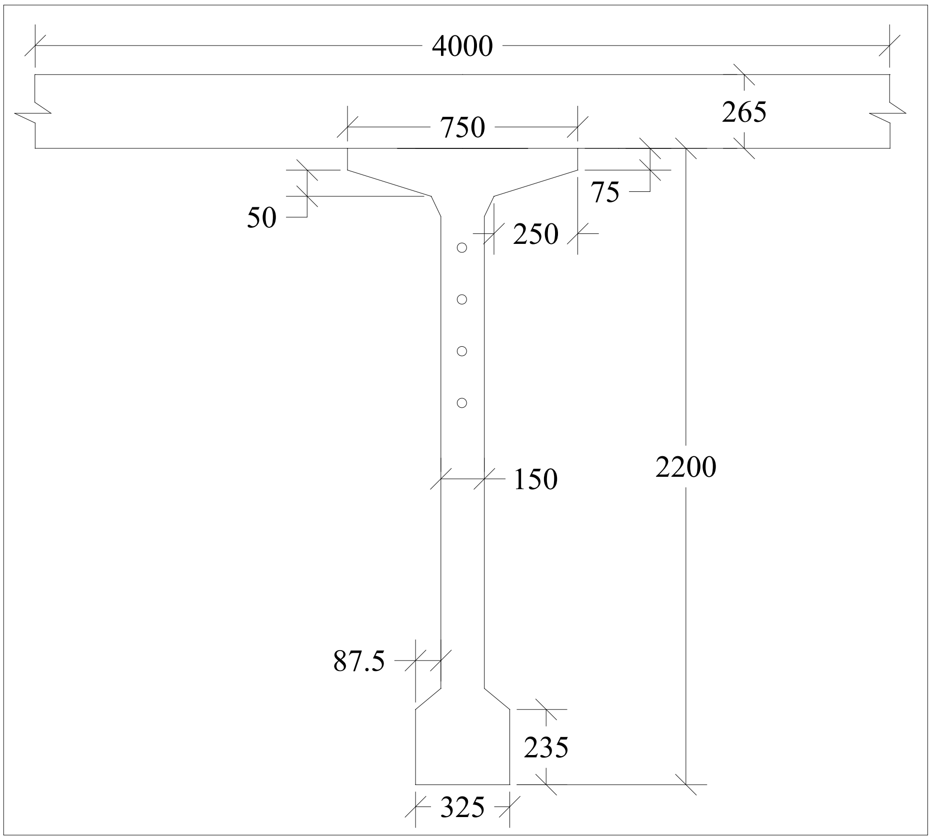


Figure 6.2(b) Optimum design for double span cont. girder at section 5 for Cost2

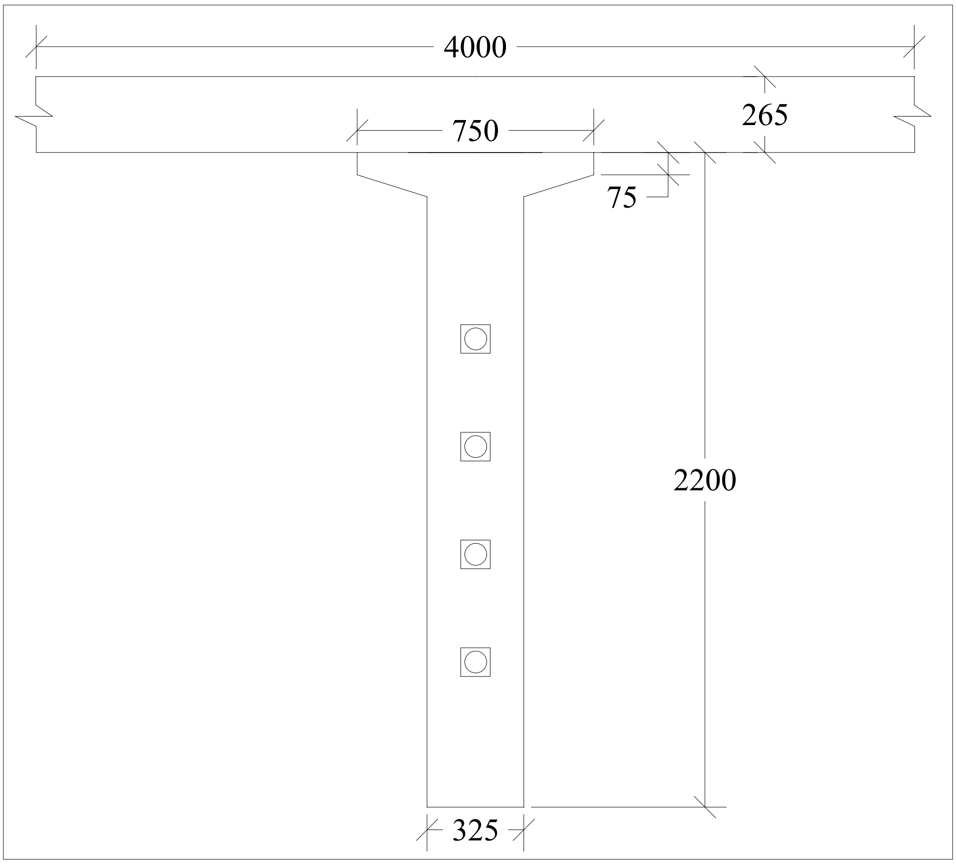


Figure 6.2(c) Optimum design for double span cont. girder at section 4 for Cost2

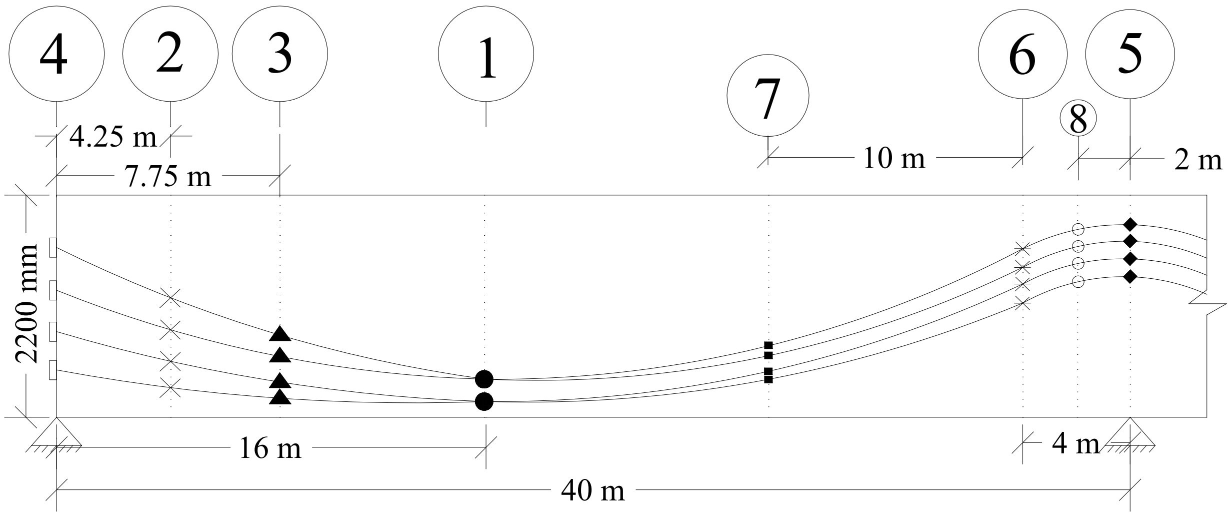


Figure 6.2(d) Optimum design for double span continuous girder for Cost2

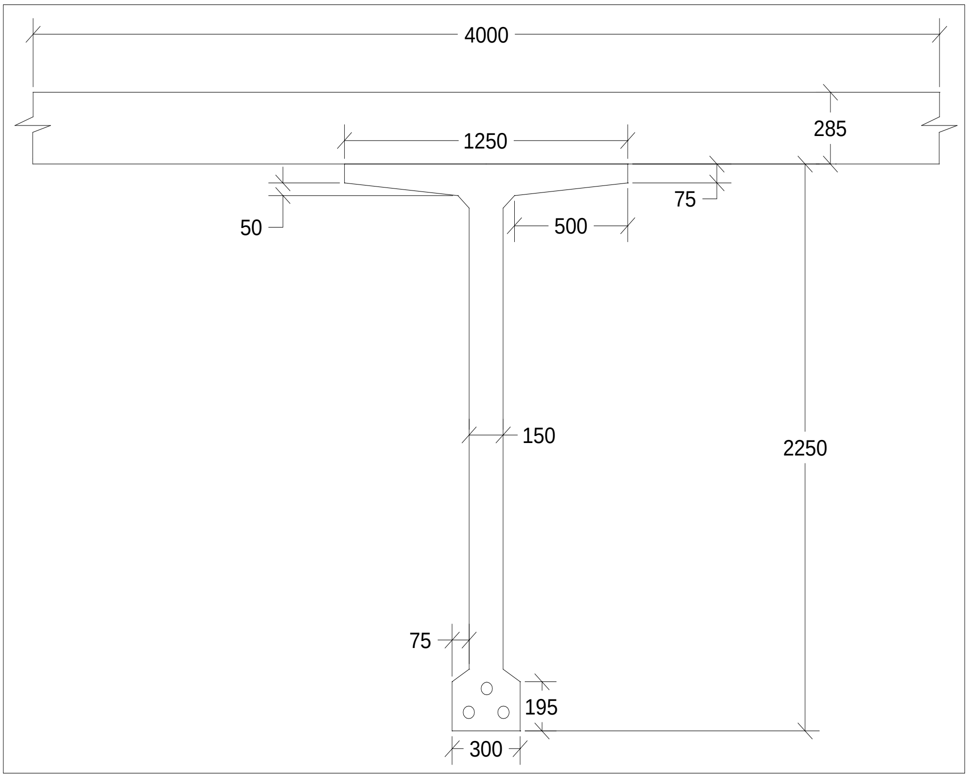


Figure 6.3(a) Optimum design for double span cont. girder at section 1 for Cost3

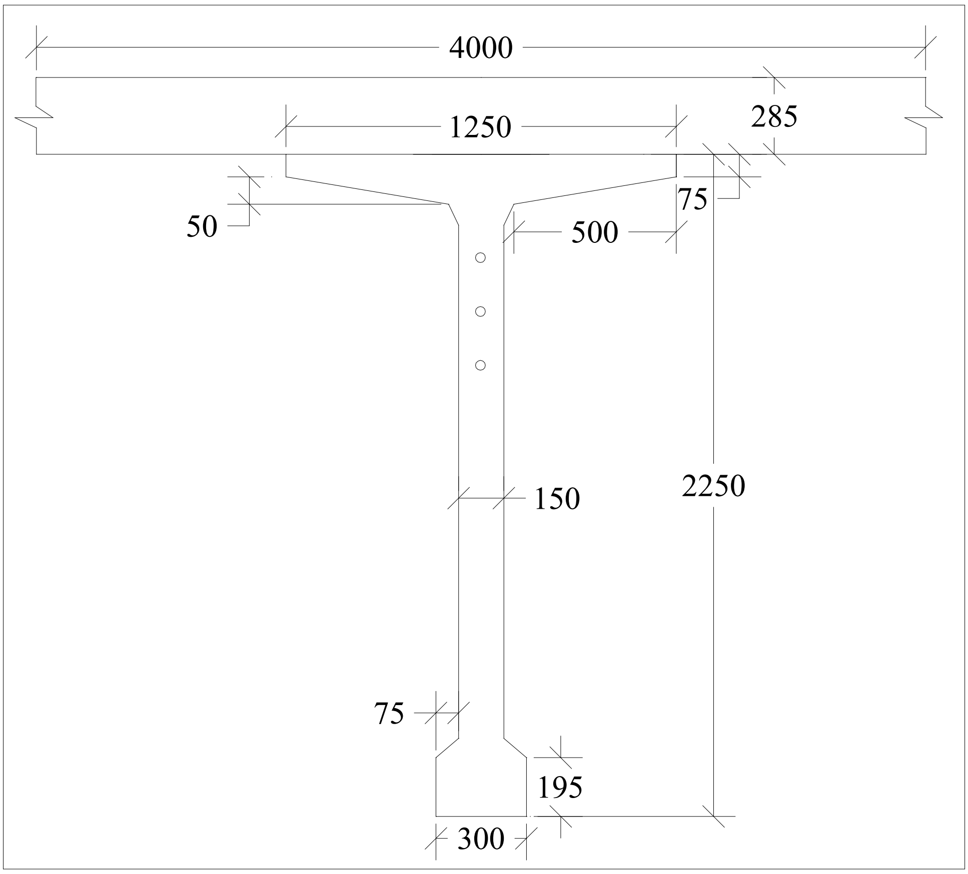


Figure 6.3(b) Optimum design for double span cont. girder at section 5 for Cost3

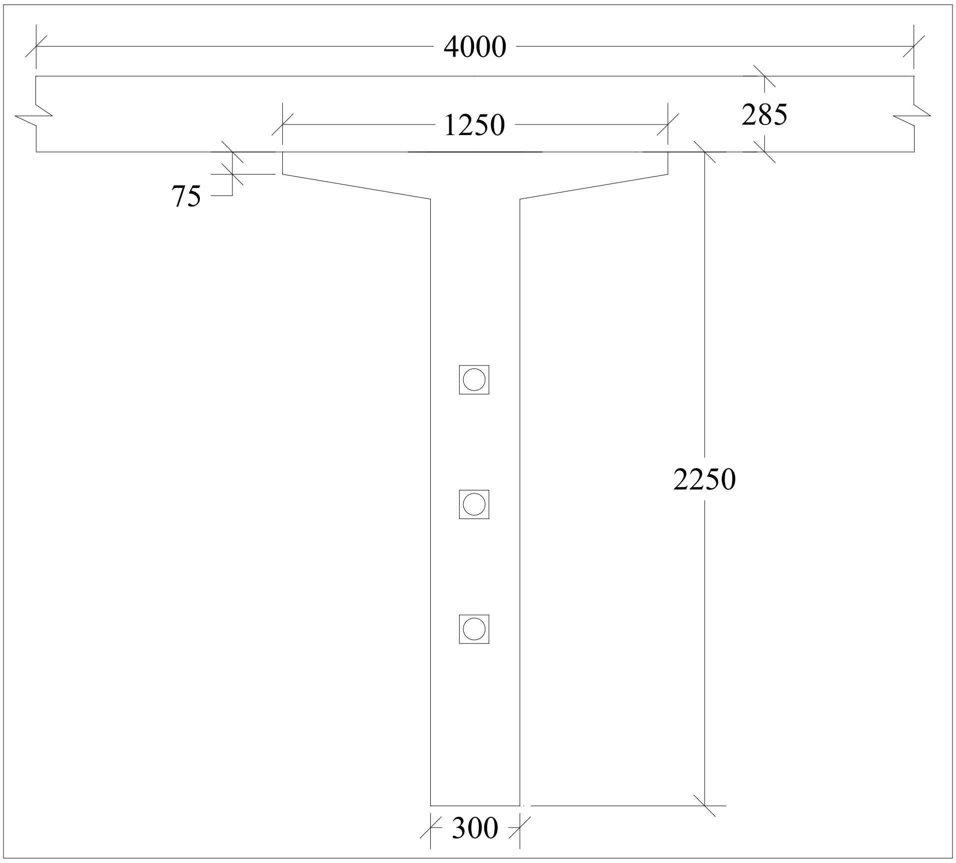


Figure 6.3(c) Optimum design for double span cont. girder at section 4 for Cost3

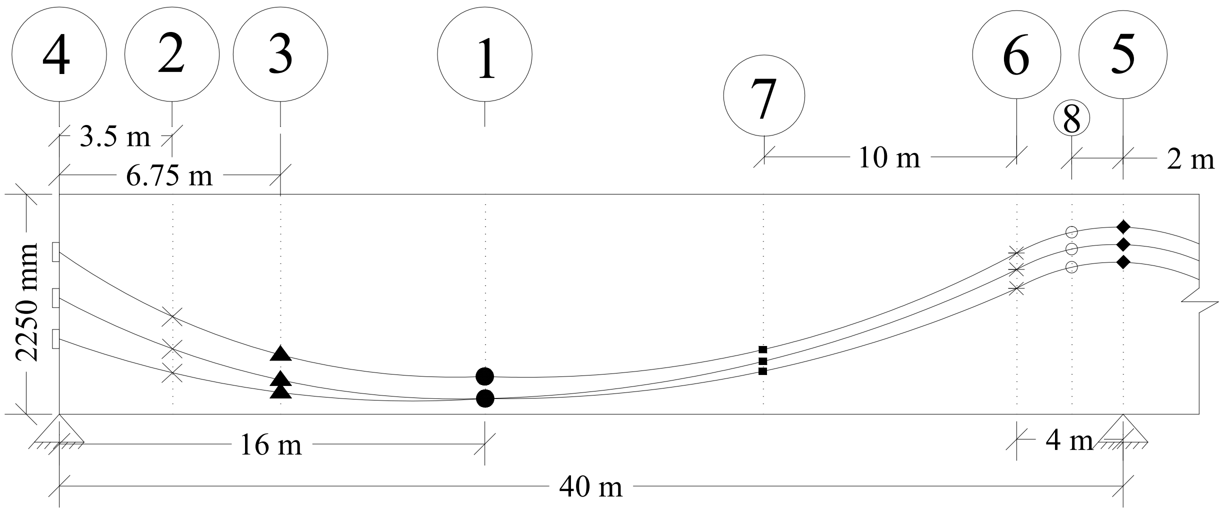


Figure 6.3(d) Optimum design for double span continuous girder for Cost3

Table 6.5 Cost of individual materials for 3 Lane 40 m double span cont girder

	Girder concrete strength = 50 MPa					Girder concrete strength = 40 MPa				
Cost **(BDT)**	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***
C1	1880	1790	1250	1250	**4,540**	1590	1690	1215	1265	**4,970**
C2	2290	1860	1975	2255	**7,980**	1930	1720	1750	2350	**7,420**
C3	2350	1720	2770	3380	**9,750**	2350	1930	2320	2910	**9,010**

* Cost in **(BDT)** per square meter of deck slab; TC = Total cost;

Table 6.6 Computational effort and control parameters used

	Girder concrete strength = 50 MPa								Girder concrete strength = 40 MPa							
	OF*	EC*	IC*	T (s)	α	β	γ	Φ	Φ_{cpx}	OF*	EC*	IC*	T (s)	α	β	γ	Φ	Φ_{cpx}
C1	263	567	457	7	1.2	0.5	2	10^{-13}	10^{-16}	245	1234	783	8	1.3	0.5	2	10^{-13}	10^{-16}
C2	452	986	907	8	1.4	0.5	2	10^{-13}	10^{-16}	125	673	453	9	1.8	0.5	2	10^{-13}	10^{-16}
C3	457	1783	1209	8	1.2	0.6	2	10^{-13}	10^{-16}	567	3248	1673	9	1.9	0.5	3	10^{-13}	10^{-16}

* Number of evaluations; OF = Objective function; EC = Explicit constraint; IC = Implicit constraint; T = Time (sec)

From the parametric study of the cost optimum design for **40 m** double span continuous girder of the present bridge system it is observed that:

(i) Optimum girder spacing for 3 Lane Bridge is **4.0 m** for both concrete strengths (50 MPa and 40 MPa) and for all the cost cases, Cost1, Cost2 and Cost3 (Table 6.2 and Table 6.3). It indicates that it is more economical to space the girder at the maximum practical spacing.

(ii) Optimum girder depth for Cost1, Cost2, Cost3 are 1700 mm, 2200 mm and 2250 mm respectively for 50 MPa concrete strength and 1850 mm, 2250 mm, 2625 mm respectively for 40MPa concrete strength (Table 6.2 and Table 6.3).

So optimum girder depth increases with increase in costs of steels in both cases which indicates that relative cost difference of materials influence optimum design of bridge. The optimum depth is smaller in higher concrete strength. Optimum designs of the 40 m double span continuous girder bridge for concrete strength of 50 MPa are shown in Figure 6.1, 6.2 and 6.3.

(iii) Due to composite construction, deck slab thickness is adequate to satisfy the compression area required for flexural strength of the girder and so top flange width is controlled by the effective span of deck slab to satisfy serviceability criteria of the deck and lateral stability effects of the girder. Top flange width increases in Cost2 and in Cost3. Optimum top flange widths are 450 mm, 750 mm and 1250 mm in Cost1, Cost2, Cost3 respectively in case of concrete strength of 50 MPa and 725 mm, 950 mm and 1375 mm respectively in case of concrete strength of 40 MPa. Optimum top flange width increases with relative increase in costs of steels. Optimum top flange width decreases with increases in concrete strength. Top flange thickness and top flange transition thickness remain to their lower limit.

(iv) Optimum bottom flange width is about 300 mm to 325 mm for both concrete strengths which is close to the lower limit. It indicates that it is not necessary to have large width to accommodate all the tendons in the lowermost position to have greater eccentricity. Thus bottom flange transition area is minimized to keep the concrete area smaller. Optimum bottom flange thickness decreases with increase in relative costs of steels as number of tendon decreases with increase in relative costs of steels.

(v) Optimum web width in all the three cases is about 145 mm to 160 mm and number of strands per tendon is 8 to 9 for concrete strength of 50 MPa and 7 to 9 for concrete strength of 40 MPa. It indicates that the C-Range anchorage system which accommodates 7 to 9 tendons is the optimum value (Table 6.2, Table 6.3). Number of tendons i.e. prestressing steel required decreases with increase in cost of steels.

(vi) Deck slab thickness increases a little which indicates that even the steel cost is high, deck thickness does not increase comparatively because larger thickness induces larger dead load. So optimum value of deck slab thickness is 255 mm to 285 mm for concrete strength of 50 MPa and 245 mm to 290 mm for concrete strength of 40 MPa. Reinforcement ratio in the deck decreases with increase in steel costs for cost minimization of the bridge.

(vii) Percentage of steel to be prestressed at initial stage increases with the increase in steel cost which indicates that as steel cost increase the girder weight also increases which require more prestress at initial stage. In this study tendons arrangement along the girder is considered as variables and the vertical position of tendons at various sections are shown in the Table 6.4.

(viii) Optimum costs of bridge for different relative costs of materials and for different concrete strengths are tabulated in Table 6.5. Total cost of steels is higher in higher concrete strength.

(ix) The most active constraints governing the optimum design are compressive stress at top fiber of girder for permanent dead load at service condition, tensile stress at bottom fiber due to all loads, prestress force at the end of seating loss zone, deck thickness and factor of safety against lateral stability, deflection at service condition due to full load in most of the three cases. When steel cost is higher, flexural strength of composite girder becomes an active constraint as amount of prestressing steel decreases.

(x) Computational efforts used by EVOP and control parameters used are tabulated in Table 6.6 which shows that the optimization problem with a large number of mixed type design variables and implicit constraints converges with a small number of function evaluations. Intel COREi5 processor has been used in this study and computational time required for optimization by EVOP is about only 7-8 seconds.

6.2.2 Optimum design for 60 m double span continuous girder

The optimum designs for 60 m double span continuous girder for various relative costs and concrete strengths are tabulated in Table 6.7 and Table 6.8. The optimized costs are tabulated in Table 6.10.

Table 6.7 Optimum values of design variables for 3 Lane 60 m double span continuous girder and Concrete strength = 50 MPa

Cost	S (m)	G_d (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	3.0	2430	1075	75	50	365	225	150	9	5	240	0.65	750	55
C2	3.0	2670	1150	75	50	350	240	150	9	4	230	0.59	880	68
C3	3.0	3030	1075	75	50	335	180	150	8	4	240	0.55	760	73

Table 6.8 Optimum values of design variables for 3 Lane 60 m double span continuous girder and Concrete strength = 40 MPa

Cost	S (m)	Gd (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	3.0	2690	1225	75	50	360	240	150	9	7	235	0.67	920	51
C2	3.0	3120	1350	75	50	370	270	150	9	5	240	0.58	820	59
C3	3.0	3450	1125	75	50	320	175	150	9	4	270	0.53	875	72

Table 6.9 Cg of tendons from bottom fiber of girder in the optimum design

	Girder concrete strength = 50 MPa									Girder concrete strength = 40 MPa								
	Y_1	Y_2	Y_3	Y_{end}	Y_5	Y_6	Y_7	Y_8	AS	Y_1	Y_2	Y_3	Y_{end}	Y_5	Y_6	Y_7	Y_8	AS
C1	137	453	1025	1387	2210	2019	1023	2106	225	135	522	1158	1630	2270	2019	1046	2140	258
C2	119	443	980	1375	2196	2018	1087	2102	229	122	438	946	1355	2201	1989	1093	2096	256
C3	116	389	829	1198	2504	2297	1198	2409	213	137	468	942	1422	2645	2424	1187	2532	346

AS = Anchorage spacing

Table 6.10 Cost of individual materials for 3 Lane 60 m double span cont girder

	Girder concrete strength = 50 MPa					Girder concrete strength = 40 MPa				
Cost **(BDT)**	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***
C1	3550	1390	2280	1160	**7,850**	2920	1390	2070	1160	**6,920**
C2	3860	1370	3650	2350	**10,700**	3290	1300	3300	2530	**9,760**
C3	4220	1370	4960	3430	**13,910**	3470	1440	4320	3650	**11,460**

* Cost in **(BDT)** per square meter of deck slab; TC = Total cost;

Optimum d esign for 6 0 m double s pan c ontinuous girder (Girder co ncrete strength = 40 MPa)

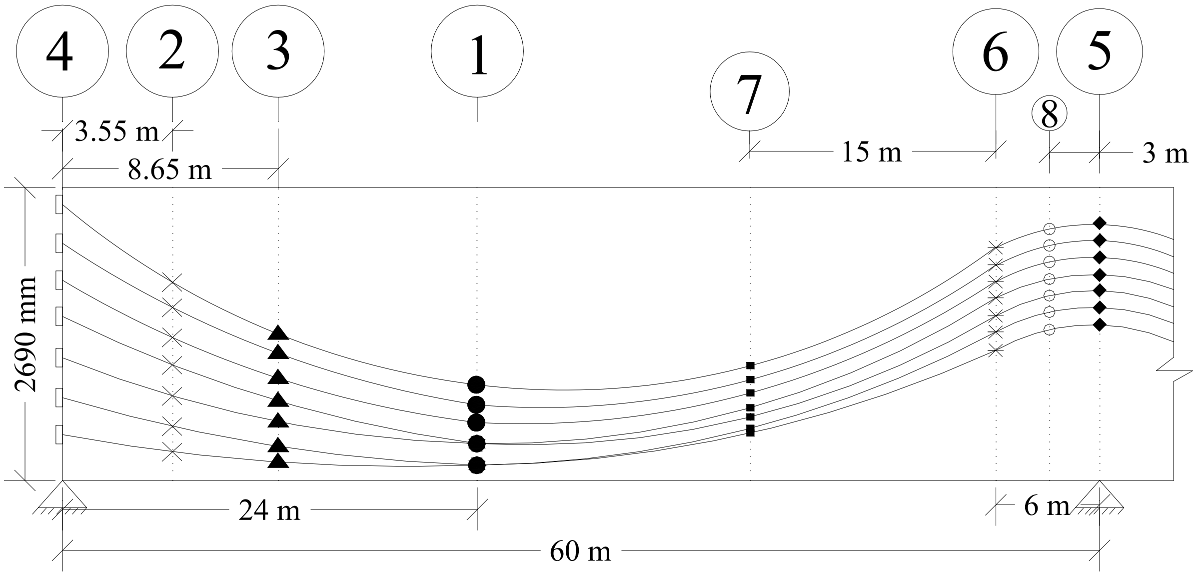


Figure 6.2(a) Optimum design for double span continuous girder for Cost1

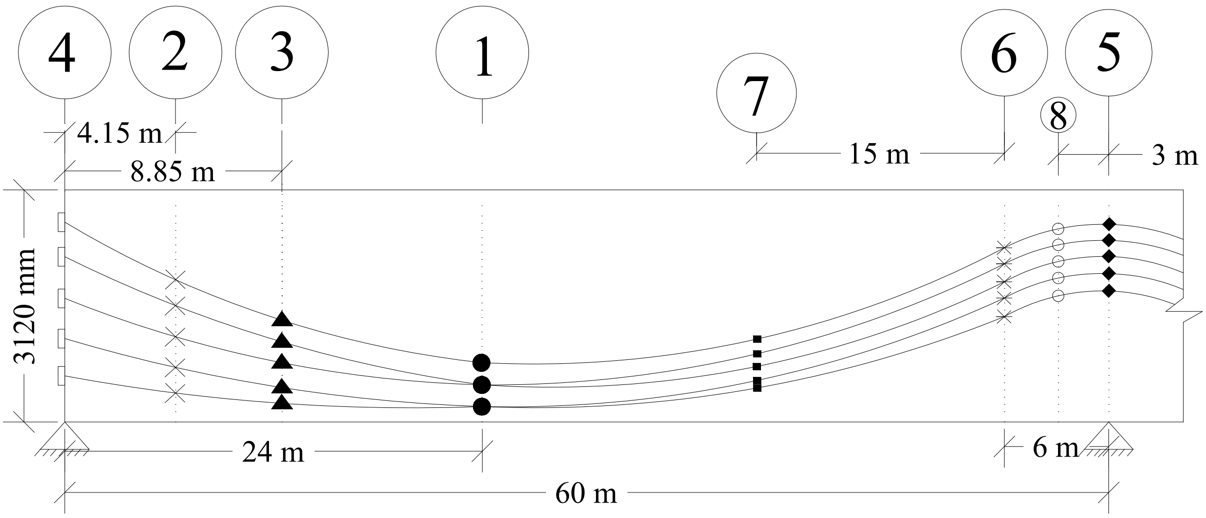


Figure 6.2(b) Optimum design for double span continuous girder for Cost2

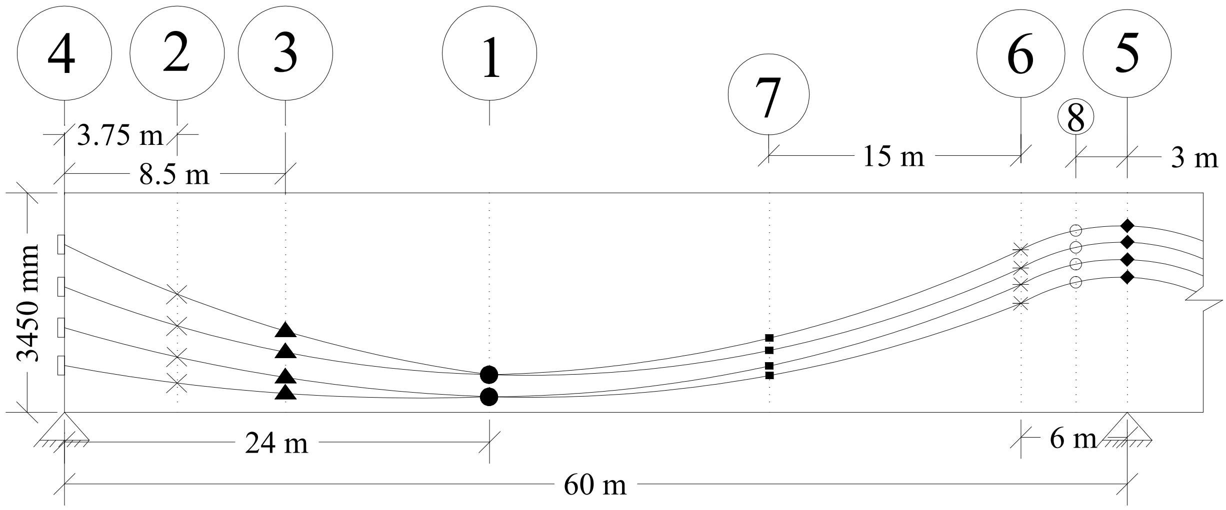


Figure 6.2(c) Optimum design for double span continuous girder for Cost3

Table 6.11 Computational effort and control parameters used

	Girder concrete strength = 50 MPa									Girder concrete strength = 40 MPa								
	OF*	EC*	IC*	T (s)	α	β	γ	Φ	$Φ_{cpx}$	OF*	EC*	IC*	T (s)	α	β	γ	Φ	$Φ_{cpx}$
C1	153	863	569	7	1.2	0.5	2	10^{-13}	10^{-16}	79	458	276	7	1.3	0.5	2	10^{-13}	10^{-16}
C2	184	652	652	8	1.5	0.5	2	10^{-13}	10^{-16}	99	568	547	7	1.6	0.5	2	10^{-13}	10^{-16}
C3	164	763	459	8	1.7	0.5	2	10^{-13}	10^{-16}	127	539	379	9	1.2	0.5	2	10^{-13}	10^{-16}

* Number of evaluations; OF = Objective function; EC = Explicit constraint; IC = Implicit constraint; T = Time (sec)

Table 6.12 Values of design variables for 4 Lane 60 m double span continuous girder and Concrete strength = 50 MPa

	S (m)	G_d (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %	TC*
C1	3.2	2530	950	75	50	355	240	150	9	6	210	0.72	837	41	7990
C2	3.2	2790	1150	75	50	365	190	150	9	5	210	0.67	714	45	10950
C3	3.2	2880	1150	75	50	375	220	150	9	5	220	0.58	950	46	13780

From the parametric study of the cost optimum design for **60 m** girder of the present bridge system it is observed that:

(i) Optimum girder spacing for 3 Lane Bridge is 3m for both concrete strengths and for 4 Lane Bridge is 3.2m for all the cost cases, Cost1, Cost2 and Cost3 (Table 6.7, Table 6.8 and Table 6.12).

(ii) Optimum girder depth for Cost1 is 2430 mm (3 lanes) and 2530 mm (4 lanes) for 50 Mpa concrete strength and 2690 mm (3 lanes) for 40Mpa concrete strength. Optimum girder depth increases with increase in costs of steels in both cases.

(iii) Optimum top flange width increases in Cost2 compared to Cost1 while it remains nearly unchanged for Cost3. In Cost3 required deck thickness is greater for optimization which need not smaller span. Optimum top flange width is in between 1075 mm to 1150 mm in case of concrete strength of 50 Mpa and 1125 mm to 1350 mm in case of concrete strength of 40 Mpa. It indicates that the wider top flange reduces the formwork cost of the deck slab and increase safety factor against lateral stability. Top flange thickness and top flange transition thickness remain to their lower limit.

(iv) Optimum bottom flange width is about 370 mm for both concrete strengths which is close to the lower limit which indicates that it is not necessary to have large width to accommodate all the tendons in the lowermost position to have greater eccentricity. Thus bottom flange transition area is minimized to keep the concrete area smaller. Bottom flange thickness increases a little with increase in cost of steel in Cost2 but decrease in Cost3.

(v) Optimum web width in all the three cases is 150 mm and number of strands per tendon is 8 or 9 for both concrete strengths. It indicates that the C-Range anchorage system which accommodates 9 tendons is the optimum value (Table 6.7, Table 6.8). Number of tendons required decreases with increase in cost of steels. Number of tendons required is lower in higher concrete strength.

(vi) Deck slab thickness increases a little which indicates that even the steel cost is high, deck thickness does not increase comparatively because larger thickness induces larger dead load. So optimum value of deck slab thickness is 215 mm to 230 mm irrespective of the relative cost differences. Reinforcement ratio in the deck decreases with increase in steel costs for cost minimization of the bridge.

(vii) The vertical positions of tendons at various sections are shown in the Table 6.9.

(viii) Optimum costs of bridge for different relative costs of materials and for different concrete strengths are tabulated in Table 6.10. Total cost of steels is higher in higher concrete strength.

(ix) The most active constraints governing the optimum design are compressive stress at top fiber of girder for permanent dead load at service condition, tensile stress at bottom fiber due to all loads, prestressing force at the end of seating loss zone, deck thickness and factor of safety against lateral stability, deflection at service condition due to full load in most of the three cost cases. When steel cost is higher, flexural strength of composite girder becomes an active constraint as amount of prestressing steel decreases.

(x) It is interesting to note that costs per square meter of deck are very close irrespective of number of lanes.

(xi) Computational efforts used by EVOP and control parameters used are tabulated in Table 6.11.

6.2.3 Optimum design for 80 m double span continuous girder

The optimum designs for 80 m double span continuous girder for various relative costs and concrete strengths are tabulated in Table 6.13 and Table 6.14. The optimized costs are tabulated in Table 6.15.

Table 6.13 Optimum values of design variables for 3 Lane 80 m double span continuous girder and Concrete strength = 50 MPa

Cost	S (m)	G_d (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	4.0	3450	725	75	50	300	255	145	9	8	235	0.61	832	42
C2	3.0	3675	800	75	50	300	220	145	9	7	210	0.55	843	59
C3	3.0	3750	700	75	50	300	210	135	7	7	220	0.51	762	52

Optimum design for 80 m double span continuous girder (Girder concrete strength = 50 MPa)

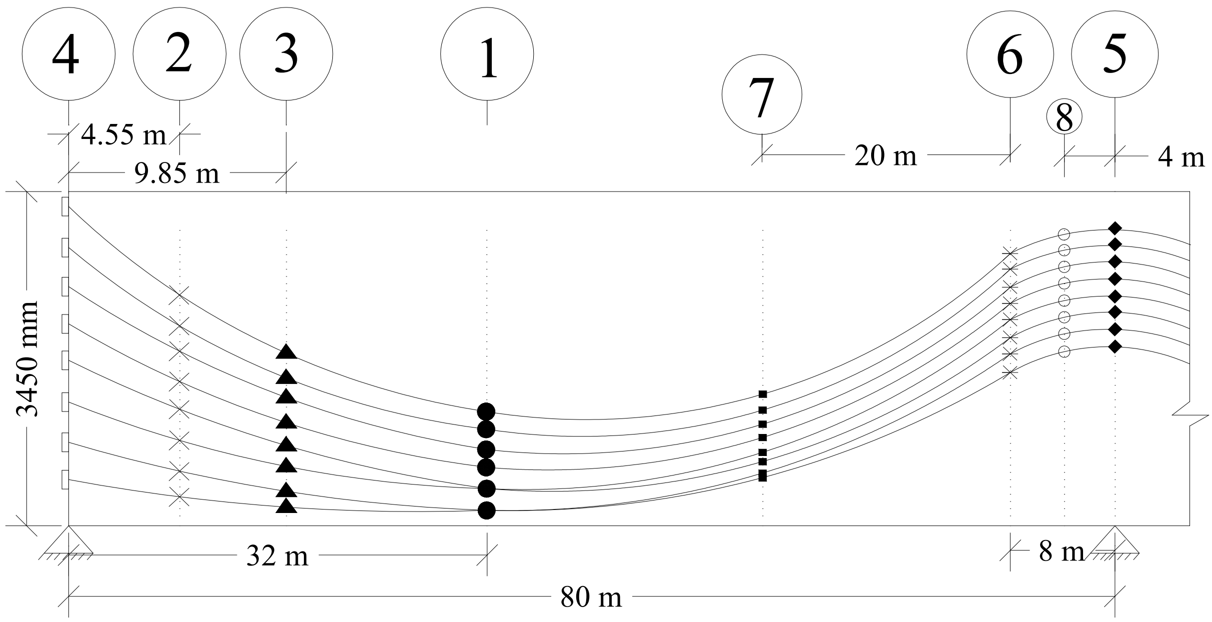


Figure 6.3(a) Optimum design for double span continuous girder for Cost1

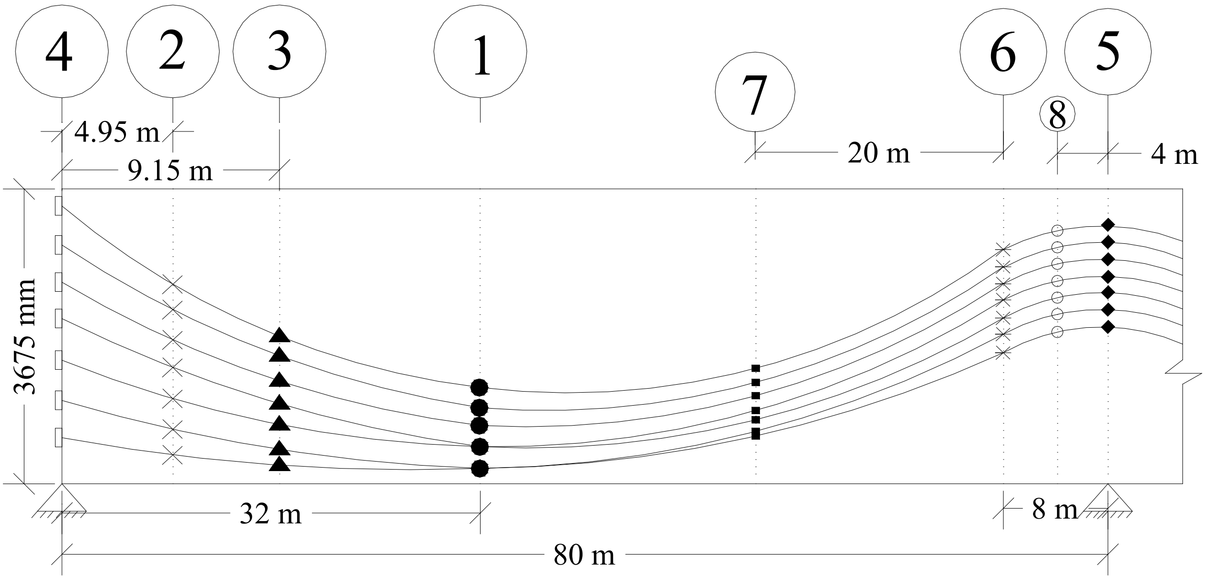


Figure 6.3(b) Optimum design for double span continuous girder for Cost2

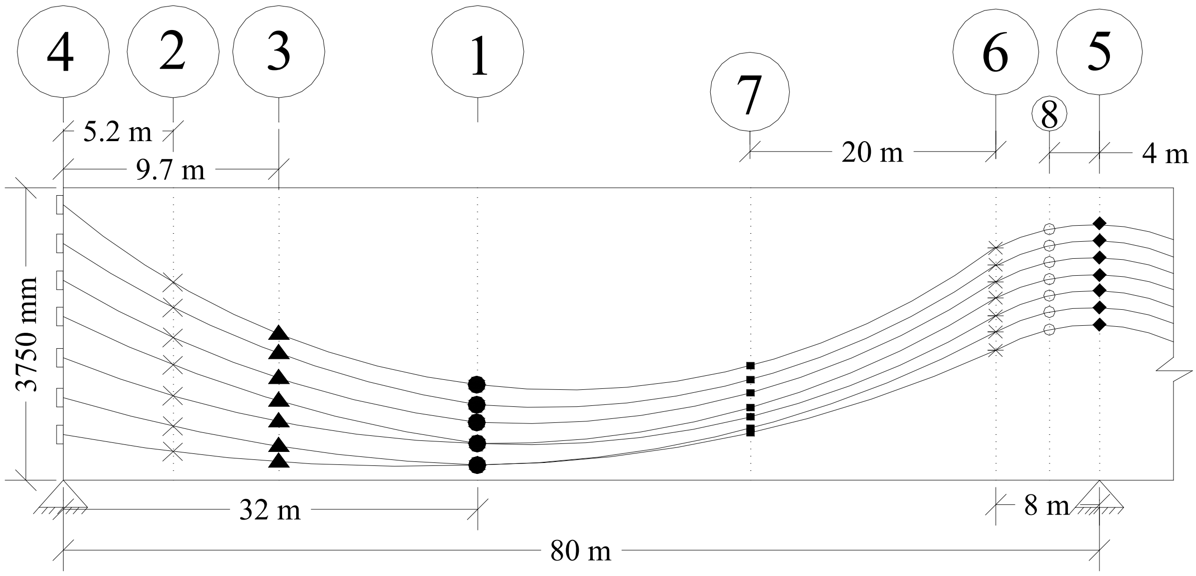


Figure 6.3(c) Optimum design for double span continuous girder for Cost3

Table 6.14 Optimum values of design variables for 3 Lane 80 m double span continuous girder and Concrete strength = 40 MPa

Cost	S (m)	G_d (mm)	TF_w (mm)	TF_t (mm)	TFS_t (mm)	BF_w (mm)	BF_t (mm)	W_w (mm)	N_S	N_T	t (mm)	ρ %	y_1 (mm)	η %
C1	3.0	3650	1050	75	50	300	200	145	8	8	230	0.61	921	45
C2	3.0	3850	950	75	50	300	270	145	9	8	235	0.57	932	42
C3	3.0	3900	1150	75	50	325	155	145	9	7	260	0.42	719	41

Table 6.15 Cost of individual materials for 3 Lane 80 m double span cont girder

	Girder concrete strength = 50 MPa					Girder concrete strength = 40 MPa				
Cost **(BDT)**	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***	C_{GC}*	C_{DC}*	C_{PS}*	C_{OS}*	**TC***
C1	2690	1740	1460	1230	**6,490**	2210	1690	1330	1210	**4,990**
C2	3780	1560	2320	2300	**9,750**	2360	1800	2400	2430	**8,390**
C3	3750	1150	3600	2580	**10,980**	2420	1800	3620	3410	**10,690**

* Cost in **(BDT)** per square meter of deck slab; TC = Total cost;

Table 6.16 shows total optimum number of prestressing strands (N_S x N_T) required for various girder span and concrete strength. Number of strands increases with the increase in span of girder.

Table 6.17 shows optimum girder spacing for various girder span and concrete strength. Girder spacing is higher in smaller span than larger span. Girder spacing depends on the maximum limit of girder depth. If maximum depth of girder can exceed the practical limit, the girder spacing will increase for more optimized design of bridge.

Table 6.16 Total optimum number of prestressing strands (N_S x N_T)

Cost	Girder concrete strength = 50 MPa			Girder concrete strength = 40 MPa		
	40 m	60 m	80 m	40 m	60 m	80 m
C1	36	45	72	36	63	64
C2	36	36	63	32	45	72
C3	24	32	49	21	36	63

Table 6.17 Optimum girder spacing (meter)

Cost	Girder concrete strength = 50 MPa			Girder concrete strength = 40 MPa		
	40 m	**60 m**	**80 m**	**40 m**	**60 m**	**80 m**
C1	4.0	3.0	4.0	4.0	3.0	3.0
C2	4.0	3.0	3.0	4.0	3.0	3.0
C3	4.0	3.0	3.0	4.0	3.0	3.0

Table 6.18 shows optimum deck slab thickness for various girder span and concrete strength. Optimum deck slab thickness is higher in shorter span as the girder spacing is higher in shorter span. The higher girder spacing, the higher effective span of deck slab which requires thicker depth of deck slab. Table 6.19 shows Optimum deck slab main reinforcement. It can be observed that girder concrete strength has no effect on optimum deck slab main reinforcement.

Table 6.18 Optimum deck slab thickness (mm)

Cost	Girder concrete strength = 50 MPa			Girder concrete strength = 40 MPa		
	40 m	**60 m**	**80 m**	**40 m**	**60 m**	**80 m**
C1	255	240	235	245	235	230
C2	265	230	210	245	240	235
C3	285	240	220	290	270	260

Table 6.19 Optimum deck slab main reinforcement (%)

Cost	Girder concrete strength = 50 MPa			Girder concrete strength = 40 MPa		
	40 m	**60 m**	**80 m**	**40 m**	**60 m**	**80 m**
C1	0.68	0.65	0.61	0.67	0.67	0.61
C2	0.55	0.59	0.55	0.52	0.58	0.57
C3	0.48	0.55	0.51	0.45	0.53	0.42

CHAPTER 7

CONCLUSIONS AND SUMMARY OF SUGGESTIONS

7.1 Conclusions

The present research work commenced with an aim to achieve the cost minimization of a double span continuous post-tensioned PC I-girder bridge superstructure system by adopting an optimization approach to obtain the optimum design and also to perform various parametric studies for the constant design parameters of the bridge system to observe the effects of such parameters on the optimum design. A global optimization algorithm named EVOP (Evolutionary Operation) is used which is capable of locating directly with high probability the global minimum. A program is developed for the optimization which may be beneficial to designers and contractors interested in cost minimum design of the present bridge system. Influence of constant design parameters such as unit costs of materials and concrete strength on the optimum design is presented.

Under the scope of the present study, following conclusions can be made:

(i) Optimum girder spacing is higher in smaller span than larger span bridges.

(ii) Optimum girder depth increases with increase in cost of steels. On an average, girder depth increases 22% with every 100% increase in cost of steel for 40 MPa concrete. On the other hand, for 50 MPa concrete, the average increase in girder depth came out to be 19%.

(iii) Optimum top flange width is controlled by deck slab span and lateral stability effect. Top flange thickness and top flange transition thickness remain to their lower limit.

(iv) Optimum bottom flange width remains nearly to the lower limit. Optimum bottom flange thickness decreases with increase in relative cost of steels.

(v) Optimum web width remains nearly constant irrespective of girder span and concrete strength.

(vi) Optimum number of strand is higher in higher span girder. Number of strand decreases 17% with every 100% increase in cost of steel for 40 MPa concrete. On the other hand, for 50 MPa concrete, the average decrease in number of strand came out to be 16%.

(vii) Optimum number of strand per tendon is 8 or 9 for both concrete strengths for 80 m girder and 7 to 9 for 40 m and 60 m girder spans studied. Number of strands required is higher in higher concrete strength.

(viii) Optimum deck slab thickness is higher in shorter span as the girder spacing is higher in shorter span. The higher girder spacing, the higher effective span of deck slab which requires thicker depth of deck slab.

(ix) The present constrained optimization problem of 14 number design variables having a combination of continuous, integer, discrete types and 73 numbers of implicit constraints is easily solved by EVOP with a relatively small number of function evaluations by simply adjusting the EVOP control parameters.

7.2 Summary of Suggestions

It is recommended that the study can be extended further in the following fields:

(i) Application of optimization approach on the I-Girder bridges system or other types of bridge system considering both superstructure and substructure.

(ii) Application of high strength concrete in the optimization of I- Girder bridge system. High strength concrete (HSC) has several advantages over conventional strength concrete. These include increased compressive strength, modulus of elasticity, tensile strength. In addition, high strength concrete is nearly always enhanced by these other benefits, a smaller creep coefficient, less shrinkage strain, lower permeability and improved durability.

(iii) Application of optimization approach on various types of prestressed concrete structures under flexure considering the various classes (Class U, Class T and Class C) of flexure.

APPENDIX-A

COMPUTER PROGRAM

(Written in C++ Language)

```
//********************************************************************************
//__01__01__01__01____________Header Files Declaration Zone____________01__01__01__01__
//********************************************************************************
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cmath>
#include <string>
#include <time.h>
#include<math.h>
#define SWAP(a,b){temp=(a);(a)=(b);(b)=temp;}
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*
using namespace std;

extern "C"
{
    void __stdcall EVOP(double*,double*,double*,double*,double*,double*,double*,
                        int*,int*,int*,int*,int*,int*,int*,int*,int*,int*,double*,double*,double*,
                        double*,double*,double*,double*,double*,double*,double*,double*,double*);
          void __stdcall DINTG2(int*,double*,double*,double*,double*);
          void __stdcall DISCR2(double*,int*,int*,double*,double*,double*,double*);
          void __stdcall EXPCON(int*,int*,int*,int*,double*,double*,double*);
          void __stdcall FUNC(double*,int*,int*,int*,double*);
          double __stdcall RNDOFF(double*);
          void __stdcall IMPCON(int*,int*,double*,double*,double*,double*);
}
//********************************************************************************
//__02__02__02__02____________Variable Declaration Zone____________02__02__02__02__02__
//********************************************************************************
int No_of_Span=2;
int No_of_Node=No_of_Span+1;
int y=1;
int xx=1;
double xw;
```

```
//Area and other
double Ag,Gd,Gdc,Anet,Atf,Atfc,EFW;

//I
double I,Inet,Ic,Itf;

//Y
double Y1,Y2,Y3,Y_end,Y_int_sup,Y_inf,Y7,Y8;    //cg of strands
//double Y1i,Y2i,Y3i,Y_end_i,Y_int_sup_i,Y_inf_i,Y7i,Y8i;  //cg of strands
double
Yb,Y1bnet,Y1tnet,Y2bnet,Y2tnet,Y3bnet,Y3tnet,Y_int_sup_bnet,Y_int_sup_tnet,Y_inf_bnet,Y_inf_tnet,Y
7bnet,Y7tnet,Y8bnet,Y8tnet,Y1b,Y2b,Y3b,Y_int_sup_b,Y_inf_b,Y7b,Y8b,Y1t,Y2t,Y3t,Y_int_sup_t,Y_inf_t,Y
7t,Y8t,Y1bc,Y1tc,Y2bc,Y2tc,Y3bc,Y3tc,Y_int_sup_bc,Y_int_sup_tc,Y_inf_bc,Y_inf_tc,Y7bc,Y7tc,Y8bc,Y8tc;
        //cg of section

//e
double
e1,e2,e3,e_end,e_int_sup,e_inf,e7,e8,e1i,e2i,e3i,e_end_i,e_int_sup_i,e_inf_i,e7i,e8i,ec1,ec2,ec3,ec_en
d,ec_int_sup,ec_inf,ec7,ec8; //eccentricity

//s
double
S1tnet,S1bnet,S1t,S1b,S1tc,S1bc,S2tnet,S2bnet,S2t,S2b,S2tc,S2bc,S3tnet,S3bnet,S3t,S3b,S3tc,S3bc,S_en
d_tnet,S_end_bnet,S_end_t,S_end_b,S_end_tc,S_end_bc;
double
S_int_sup_tnet,S_int_sup_bnet,S_int_sup_t,S_int_sup_b,S_int_sup_tc,S_int_sup_bc,S_inf_tnet,S_inf_b
net,S_inf_t,S_inf_b,S_inf_tc,S_inf_bc,S7tnet,S7bnet,S7t,S7b,S7tc,S7bc,S8tnet,S8bnet,S8t,S8b,S8tc,S8bc;

double
Cable_Loc_mid[31],Cable_Loc_end[31],Cable_Loc_int_sup[31],Cable_Loc_inf[31],Cable_Loc_8[31],Cable
_Loc_7[31],alpha[31],alpha_xw[31],alpha3[31],alpha7[31],alpha8[31],Layer_dist_bottom_mid[31],Layer
_dist_bottom_inf[31];
double
TFRd,TFRw,TFFHd,TFFHtw,TFFHw,TFFHbw,TFSHd=75,TFSHtw,TFSHw=75,TFSHbw,W,Wt,BFHd,BFHw,BFR
d, BFRw,ts,GS,Nstrand,Ncable,cable_1st_position_end;
double Wd,Rh,Rw,FPw,FPt,Kcr,T,UPcondeck,UPnonprest,UPcon,UPst;
double
Duct_dia,Ancg_C2C,Ancg_Edge_dist,Ancg_C2C_Lay1,Ancg_Edge_dist_Lay1,Ancg_Edge_dist_vertical,Duc
t_clear_spacing,fricncoeff,Nstrandt,Mu,Wri,Anchor_dim;
double deflectiont,deflectione,deflectionf,deflection;
double
MG1,MG2,MG3,MG4,MG5,MG6,MG7,MG8,MCG1,MCG2,MCG3,MCG4,MCG5,MCG6,MCG7,MCG8,MS1,
MS2,MS3,MS4,MS5,MS6,MS7,MS8,MWC1,MWC2,MWC3,MWC4,MWC5,MWC6,MWC7,MWC8;
double
MMS1,MMS2,MMS3,MMS4,MMS5,MMS6,MMS7,MMS8,MFP,MC1,MC2,MC3,MC4,MC5,MC6,MC7,MC8
,DF,MLL1,MLL2,MLL3,MLL4,MLL5,MLL6,MLL7,MLL8;
```

```
double
IMF,MT1,MT2,MT3,MT4,MT5,MT6,MT7,MT8,MP1,MP2,MP3,MP4,MP5,MP6,MP7,MP8,MD1,MD2,MD3,
MD4,MD5,MD6,MD7,MD8,MFP_xw,MFP3;   //??
double Fend,F3i,F1i,F11,As,F2i,F21,F31,x,Fend2;
double f4ti,F4i,e4i,S4tnet,f5ti,F5i,e5i,S5tnet,f6ti,F6i,e6i,S6tnet,f7ti,F7i,f8ti,F8i;
double
fti,fbi,ftc,fbc,ftt,fbt,fttt,fbtt,f3ti,f3bi,f3tc,f3bc,f3tt,f3bt,f3ttt,f3btt,fti_xw,fbi_xw,ftc_xw,fbc_xw,ftt_xw,fbt
_xw,fttt_xw,fbtt_xw;
double f4bi,f5bi,f6bi,f7bi,f8bi,f4tc,f5tc,f6tc,f7tc,f8tc,F41,F51,F61,F71,F81,f4bc,f5bc,f6bc,f7bc,f8bc;
double
f4tt,f5tt,f6tt,f7tt,f8tt,f4bt,f5bt,f6bt,f7bt,f8bt,f4ttt,f5ttt,f6ttt,f7ttt,f8ttt,f4btt,f5btt,f6btt,f7btt,f8btt;
double
VDL2,VLL2,IMF2,Vc,Mcr2,fpe,Vu,dshear,Vs,Vnh,ds,R,Asnp,Asnpd,rho,d_min,Muslab,IMFS,MSS,MSWC,M
SDL,MSLL,dreq;
int LayerNo_mid,LayerNo_inf,Cable_Layer_mid[31],Cable_Layer_inf[31];
double NoGirder;
double DX[8],pt1,pt2,pt3,Ig,DECKT[26],DX1[69],DX2[69],DX4[101],DX11[10], DX12[11], DX13[36];
int const nv = 14;
int const icn = 73;
double Cable_Loc_3[31],Cable_Loc_xw[31];
int cost = 1;
double Anchcost,sheathcost,UPgf,UPdf;
double Cpcon,Cpst,Cdconc,Cnpst,SA;

//                  Bridge Data:
//                  Length of girder,
double  L = 40000;                //mm
//                  Width of bridge =
double  BW = 12000;               //mm

//          Cross girder, wearing coarse and Median strip constant
int         NCG = 9;
double  CGt = 250;        //mm
double  WCt = 50;         //mm
double  MSh = 600;        //mm
double  MSw = 450;        //mm

//            Material constants:
double  Gammawc = 25;            //!KN/m3
//                  Unit weight of concrete,
double  Gammacon = 24;           //!KN/m3
//                  Unitweight of steel,
double  Gammast=7850e-9;      //Kg/mm3

double  Astrand = 140.0;  //mm2
//                  Astrand = 98.7;

//                  Wobble coefficient,
```

```
double  Kwc =0.000005;//!per mm
//                  Anchorage Slip
double  Delta= 6;           //!mm
//                  Modulus of elasticiy of steel
double  Es= 197000; //   !MPa  //AASHTO LRFD (2007) 5.4.4.2
//                  Ultimate strength of prestressing steel,
double  fpu = 1860; //ASTM A416 M
//                  Concrete:
//                  Compressive strength of concrete, MPa
double  fc=40;
//                  Compressive strength of concrete for deck slab, MPa
double  fcdeck=25;
//                  Concrete compressive strength at transfer,
double  fci =0.75*fc;

//                  Coefficient of elastic shortening,
double  Kes = 0.5;
//                  Design Data:
//                  Specification AASHTO 2007
//                  Live Load HL-93
//                  Load from frontal wheel,
double  P1=35;  //!KN
//                  Load from Rear wheel,
double  P2=145;             //!KN
//                  Modulus of elasticiy of concrete
double  Ec = 33.0*pow(150,1.5)*sqrt(fc*145.0)/145.0; // !MPa
//                  Modulus of elasticiy of concrete at initial stage
double  Eci = 33.0*pow(150,1.5)*sqrt(fci*145.0)/145.0;
double  Ecdeck = 33.0*pow(150,1.5)*sqrt(fcdeck*145.0)/145.0;
double  mratio = Ecdeck/Ec;

int whichSection_intermsof_ILmatrixrow=0;

double max_wheelLoad_shearPositive=0;
double max_wheelLoad_shearNegative=0;
double max_wheelLoad_momentPositive=0;
double max_wheelLoad_momentNegative=0;
double max_laneLoad_shearPositive=0;
double max_laneLoad_shearNegative=0;
double max_laneLoad_momentPositive=0;
double max_laneLoad_momentNegative=0;

double laneLoad_pick=0;

double loadedLength_max_wheelLoad_shearPositive=0;
double loadedLength_max_wheelLoad_shearNegative=0;
double loadedLength_max_wheelLoad_momentPositive=0;
double loadedLength_max_wheelLoad_momentNegative=0;
```

```
double loadedLength_max_laneLoad_shearPositive=0;
double loadedLength_max_laneLoad_shearNegative=0;
double loadedLength_max_laneLoad_momentPositive=0;
double loadedLength_max_laneLoad_momentNegative=0;

double ImpactFactor_max_wheelLoad_shearPositive=0;
double ImpactFactor_max_wheelLoad_shearNegative=0;
double ImpactFactor_max_wheelLoad_momentPositive=0;
double ImpactFactor_max_wheelLoad_momentNegative=0;
double ImpactFactor_max_laneLoad_shearPositive=0;
double ImpactFactor_max_laneLoad_shearNegative=0;
double ImpactFactor_max_laneLoad_momentPositive=0;
double ImpactFactor_max_laneLoad_momentNegative=0;

double w_dyn=0  ;
double UDL_SW_girder=0;
double UDL_SW_CG=0;
double UDL_SW_slab=0;
double UDL_SW_WC=0;
double UDL_SW_MS=0;
double UDL_SW_LL=0;
//*********************************************************************************
//__03__03__03__03____________Matrix/Array Declaration Zone____________03__03__03__03__
//*********************************************************************************
double local_stiffness_matrix[4][4];
double global_stiffness_matrix[32][32];
double **global_stiffness_matrix_pointer;
double global_member_force_matrix[32][1];
double **global_member_force_matrix_pointer;
double final_DOF_matrix[16][1];
double total_DL_end_shearforce_bendingmoment_matrix[15][4];
double for_ILvalue_LL_end_shearforce_bendingmoment_matrix[15][4];
double total_DL_sectionwise_shearforce_matrix[1][3500];
double total_DL_sectionwise_bendingmoment_matrix[1][3500];
double for_ILvalue_LL_sectionwise_shearforce_matrix[1][3500];
double for_ILvalue_LL_sectionwise_bendingmoment_matrix[1][3500];
double IL_matrix_shear[3500][3500];
double IL_matrix_moment[3500][3500];
double matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[1][3500];
double matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[1][3500];
double matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[1][3500];
double matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[1][3500];
//*********************************************************************************
//__04__04__04__04__04____________Matrix Initialization Zone____________04__04__04__04__04_
//*********************************************************************************
void matrix_initialization_function()
{
//		cout<<"matrix_initialization function"<<"\t";
```

```
for(int i=0;i<32;i++)
{
	for(int j=0;j<32;j++)
	{
		global_stiffness_matrix[i][j]=0.0;  //global_stiffness_matrix[32][32];
	}
}

for(int k=0;k<32;k++)
{
	global_member_force_matrix[k][0]=0.0;  //global_member_force_matrix[32][1];

	//DOF_matrix[32][1];
	//Pj_matrix[32][1];
}

for(int l=0;l<16;l++)
{
	final_DOF_matrix[l][0]=0.0;   //final_DOF_matrix[16][1];
}

for(int m=0;m<15;m++)
{
	for(int n=0;n<4;n++)
	{
		total_DL_end_shearforce_bendingmoment_matrix[m][n]=0.0;
			//total_DL_end_shearforce_bendingmoment_matrix[15][4];
		for_ILvalue_LL_end_shearforce_bendingmoment_matrix[m][n]=0.0;
			//for_ILvalue_LL_end_shearforce_bendingmoment_matrix[15][4];
	}
}

for(int o=0;o<3500;o++)
{
	total_DL_sectionwise_shearforce_matrix[0][o]=0.0;
		//total_DL_sectionwise_shearforce_matrix[1][3500];

	total_DL_sectionwise_bendingmoment_matrix[0][o]=0.0;
		//total_DL_sectionwise_bendingmoment_matrix[1][3500];

	for_ILvalue_LL_sectionwise_shearforce_matrix[0][o]=0.0;
		//for_ILvalue_LL_sectionwise_shearforce_matrix[1][3500];

	for_ILvalue_LL_sectionwise_bendingmoment_matrix[0][o]=0.0;
		//for_ILvalue_LL_sectionwise_bendingmoment_matrix[1][3500];
}

for(int p=0;p<3500;p++)
```

```
    {
        for(int q=0;q<3500;q++)
        {
            IL_matrix_shear[p][q]=0.0;    //IL_matrix_shear[3500][3500];
        IL_matrix_moment[p][q]=0.0;   //IL_matrix_moment[3500][3500];

        }
    }

    for(int r=0;r<3500;r++)
    {
        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][r]=0.0;
              //double matrix_forSorting_maXof_wheelLoadorlaneLoad_shear[1][3500];

        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][r]=0.0;
     //double matrix_forSorting_maXof_wheelLoadorlaneLoad_moment[1][3500];

        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][r]=0.0;
              //double matrix_forSorting_maXof_wheelLoadorlaneLoad_shear[1][3500];

        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][r]=0.0;
     //double matrix_forSorting_maXof_wheelLoadorlaneLoad_moment[1][3500];
    }
}
//******************************************************************************
//__05__05__05__05_____________Function Declaration Zone____________05__05__05__05__05__
//******************************************************************************
void local_stiffness_matrix_function();              //calculates [k]_local matrix
void global_stiffness_matrix_function();   //calculates [k]_global matrix
void global_member_force_matrix_function(double w_dyn);   //calculates [Pm]_golbal matrix.
void impose_boundarycondition_function();   /*imposes boundary condition by making kii=1;
                              Pi=0; ith row and column values=0 to make
                              ui=0*/
void modified_global_stiffness_matrix_function();   //considers boundary condition
void modified_global_member_force_matrix_function();    //considers boundary condition
void DOF_matrix_solution_function(double **a,int n,double **b,int m);
void final_DOF_matrix_function();  /*call within DOF_matrix_solution_function(); ...not from main()*/
void total_DL_end_shearforce_bendingmoment_function(double w_dyn);
void for_ILvalue_LL_end_shearforce_bendingmoment_function(int increment_a,int increment_c);
void total_DL_sectionwise_shearforce_function(double w_dyn);
void total_DL_sectionwise_bendingmoment_function(double w_dyn);
void influence_line_function();
void global_member_force_matrix_function_forIL(int increment_a,int increment_c);
                         //calculates [Pm]_golbal matrix for influence line
void for_ILvalue_LL_sectionwise_shearforce_function(int increment_a,int increment_c);
void for_ILvalue_LL_sectionwise_bendingmoment_function(int increment_a,int increment_c);
void matrix_initialization_function();  //initializes all matrices with zero value
void test_function();      //common output function to test different funtions
```

```
void nrerror(char error_text[]);
int *ivector(long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void dynamic_allocation_01();
void dynamic_allocation_02();
void factored_DLplusLL_shearCombination_atSpecificSection_function
(int whichSection_intermsof_ILmatrixrow);
void factored_DLplusLL_momentCombination_atSpecificSection_function
(int whichSection_intermsof_ILmatrixrow);
double total_LL_atSpecific_section_positive_shearforce_function
(int whichSection_intermsof_ILmatrixrow);
double total_LL_atSpecific_section_negative_shearforce_function
(int whichSection_intermsof_ILmatrixrow);
double total_LL_atSpecific_section_positive_bendingmoment_function
(int whichSection_intermsof_ILmatrixrow);
double total_LL_atSpecific_section_negative_bendingmoment_function
(int whichSection_intermsof_ILmatrixrow);
double maxm(double value_a, double value_b);
double minm(double value_c, double value_d);
double absolue_value_function(double value_r)
	if (value_r>=0)
		return value_r;

		else
		return -(value_r);
}
void Anchorage_system() //  Duct and Anchorage System
{
//	cout<<"anchorage_system"<<"\t";
	double Fcable;
	Nstrandt = RNDOFF(&Nstrand);
	Fcable = 0.7*fpu*Astrand*Nstrand/1000;
	if(Nstrandt <= 3)
	{
		Duct_dia = 45;
		Duct_clear_spacing = 38;
		Ancg_Edge_dist_vertical = 127.5;//Ancg_Edge_dist_vertical = 1.5*boo
		fricncoeff = 0.25;
		Anchor_dim = 110;
	}
	else if(Nstrandt <= 4 )
	{
		Duct_dia = 50;
		Duct_clear_spacing = 38;
		Ancg_Edge_dist_vertical = 150;
		fricncoeff = 0.25;
		Anchor_dim = 120;
	}
```

```
else if(Nstrandt <= 7 )
{
	Duct_dia = 65;
	Duct_clear_spacing = 38;
	Ancg_Edge_dist_vertical = 187.5;
	fricncoeff = 0.25;
	Anchor_dim = 150;
}
else if(Nstrandt <= 9 )
{
	Duct_dia = 70;
	Duct_clear_spacing = 38;
	Ancg_Edge_dist_vertical = 210;
	fricncoeff = 0.20;
	Anchor_dim = 185;
}
else if(Nstrandt <= 12 )
{
	Duct_dia = 85;
	Duct_clear_spacing = 38;
	Ancg_Edge_dist_vertical = 247.5;
	fricncoeff = 0.20;
	Anchor_dim = 200;
}
else if(Nstrandt <= 13 )
{
	Duct_dia = 85;
	Duct_clear_spacing = 38;
	Ancg_Edge_dist_vertical = 255;
	fricncoeff = 0.20;
	Anchor_dim = 210;
}
else if(Nstrandt <= 19)
{
	Duct_dia = 100;
	Duct_clear_spacing = 38;
	Ancg_Edge_dist_vertical = 300;
	fricncoeff = 0.20;
	Anchor_dim = 250;
}
else if(Nstrandt <= 22)
{
	Duct_dia = 110;
	Duct_clear_spacing = 50;
	Ancg_Edge_dist_vertical = 322.5;
	fricncoeff = 0.20;
	Anchor_dim = 275;
}
```

```
        else
        {
                Duct_dia = 115;
                Duct_clear_spacing = 50;
                Ancg_Edge_dist_vertical = 345;
                fricncoeff = 0.20;
                Anchor_dim = 300;
        }
        Ancg_Edge_dist_Lay1 = Ancg_Edge_dist_vertical/1.50;
        Ancg_C2C_Lay1 = Fcable*1000.0/fci/Ancg_Edge_dist_Lay1;
        Ancg_Edge_dist = BFRw/2;
        Ancg_C2C = Fcable*1000.0/fci/Ancg_Edge_dist;
        Wt = Duct_dia + 80;
}
double minm(double a, double b)
{
        double temp;
        if(a<b)
                temp = a;
        else
                temp = b;
        return temp;
}
double maxm(double a, double b)
{
        double temp;
        if(a>b)
                temp = a;
        else
                temp = b;
        return temp;
}
void Sectional_Properties()
{
//      cout<<"sectional_properties"<<"\t";
//      Non Composite Section Properties
        Ag =
(TFRd*TFRw)+((TFFHtw+TFFHbw)/2*TFFHd)+((TFSHtw+TFSHbw)/2*TFSHd)+Wd*Wt+((Wt+BFRw)/2*BFH
d)+(BFRd*BFRw);
        Anet = Ag-Ncable*3.1416/4*(Duct_dia)*(Duct_dia);
        Atf = Ag + (Es/Ec-1)*As;
        Yb = ((TFRd*TFRw)*(Gd-TFRd/2))+((TFFHw*TFFHd)*(Gd-TFRd-TFFHd/3));
        Yb = Yb +((TFFHtw-2*TFFHw)*TFFHd*(Gd-TFRd-TFFHd/2));
        Yb = Yb +((TFSHd*TFSHw)*(Gd-TFRd-TFFHd-TFSHd/3))+((Wt*TFSHd)*(Gd-TFRd-TFFHd-
TFSHd/2));
        Yb = Yb +((Wt*Wd)*(BFRd+BFHd+Wd/2))+((BFHw*BFHd)*(BFRd+BFHd/3));
        Yb = Yb +((BFHd*Wt)*(BFRd+BFHd/2))+((BFRd*BFRw)*BFRd/2);
//-----------------------------------------------------------------------------
```

```
        Y1bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y1;          // section 1 and 2
        Y2bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y2;

        Y1b = Yb + (Es/Ec-1)*As*Y1;
        Y2b = Yb + (Es/Ec-1)*As*Y2;

        Y1b = Y1b/Atf;
        Y2b = Y2b/Atf;

        Y1t = Gd-Y1b;
        Y2t = Gd-Y2b;

        Y1bnet = Y1bnet/Anet;
        Y2bnet = Y2bnet/Anet;

        Y1tnet = Gd-Y1bnet;
        Y2tnet = Gd-Y2bnet;
//------------------------------------------------------------------------------
        Y3bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y3;          // section 3
        Y3b = Yb + (Es/Ec-1)*As*Y3;
        Y3b = Y3b/Atf;
        Y3t = Gd-Y3b;
        Y3bnet = Y3bnet/Anet;
        Y3tnet = Gd-Y3bnet;
//------------------------------------------------------------------------------
        Y_int_sup_bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y_int_sup;   //section int support
        Y_int_sup_b = Yb + (Es/Ec-1)*As*Y_int_sup;
        Y_int_sup_b = Y_int_sup_b/Atf;
        Y_int_sup_t = Gd-Y_int_sup_b;
        Y_int_sup_bnet = Y_int_sup_bnet/Anet;
        Y_int_sup_tnet = Gd-Y_int_sup_bnet;
//------------------------------------------------------------------------------
        Y_inf_bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y_inf;    // section Inflection point
        Y_inf_b = Yb + (Es/Ec-1)*As*Y_inf;
        Y_inf_b = Y_inf_b/Atf;
        Y_inf_t = Gd-Y_inf_b;
        Y_inf_bnet = Y_inf_bnet/Anet;
        Y_inf_tnet = Gd-Y_inf_bnet;
//------------------------------------------------------------------------------
        Y7bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y7;          // section 7
        Y7b = Yb + (Es/Ec-1)*As*Y7;
        Y7b = Y7b/Atf;
        Y7t = Gd-Y7b;
        Y7bnet = Y7bnet/Anet;
        Y7tnet = Gd-Y7bnet;
//------------------------------------------------------------------------------
        Y8bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y8;          // section 8
        Y8b = Yb + (Es/Ec-1)*As*Y8;
```

```
        Y8b = Y8b/Atf;
        Y8t = Gd-Y8b;
        Y8bnet = Y8bnet/Anet;
        Y8tnet = Gd-Y8bnet;
//      Ecentricity,
//--------------------------------------------------------------------------
        e1 = Y1b-Y1;
        e1i = Y1bnet-Y1;          //!eccentricity at section 1
//--------------------------------------------------------------------------
        e2 = Y2b - Y2;    //!eccentricity at section 2 (i.e section xw)
        e2i = Y2bnet - Y2;
//--------------------------------------------------------------------------
        e3 = Y3b - Y3;    //!eccentricity at section 3 (i.e. section x)
        e3i = Y3bnet - Y3;
//--------------------------------------------------------------------------
        e_int_sup = Y_int_sup_b - Y_int_sup;     //!eccentricity at section int_sup
        e_int_sup_i = Y_int_sup_bnet - Y_int_sup;
//--------------------------------------------------------------------------
        e_inf = Y_inf_b - Y_inf;   //!eccentricity at section inflection
        e_inf_i = Y_inf_bnet - Y_inf;
//--------------------------------------------------------------------------
        e7 = Y7b - Y7;    //!eccentricity at section 7
        e7i = Y7bnet - Y7;
//--------------------------------------------------------------------------
        e8 = Y8b - Y8;    //!eccentricity at section 8
        e8i = Y8bnet - Y8;
        I = pow(TFRd,3)*TFRw/12+(TFRd*TFRw)*pow((Y1t-TFRd/2),2);
        I = I + TFFHw*pow(TFFHd,3)/36*2+(TFFHw*TFFHd)*pow((Y1t-TFRd-TFFHd/3),2);
        I = I + (pow(TFFHd,3)*TFFHbw/12)+(TFFHbw*TFFHd)*pow((Y1t-TFRd-TFFHd/2),2);
        I = I + (TFSHw*pow(TFSHd,3)/36)*2+(TFSHw*TFSHd)*pow((Y1t-TFRd-TFFHd-TFSHd/3),2);
        I = I + (Wt*pow(TFSHd,3)/12)+(Wt*TFSHd)*pow((Y1t-TFRd-TFFHd-TFSHd/2),2);
        I = I + (Wt*pow(Wd,3)/12)+(Wt*Wd)*pow((Y1b-BFRd-BFHd-Wd/2),2);
        I = I + (BFHw*pow(BFHd,3)/36)*2+(BFHw*BFHd)*pow((Y1b-BFRd-BFHd/3),2);
        I = I + Wt*pow(BFHd,3)/12+(Wt*BFHd)*pow((Y1b-BFRd-BFHd/2),2);
        I = I + (BFRw*pow(BFRd,3)/12)+(BFRd*BFRw)*pow((Y1b-BFRd/2),2);
        Inet = I - (3.1416*pow(Duct_dia,4)/32*Ncable + 3.1416*pow(Duct_dia,2)/4*Ncable*e1*e1);
        Itf = I + (Es/Ec-1)*As*e1*e1;
//-------------------------------------------------------------------------------------
        S1tnet = Inet/Y1tnet;                   //section 1 and 2
        S2tnet = Inet/Y2tnet;

        S1bnet = Inet/Y1bnet;
        S2bnet = Inet/Y2bnet;

        S1t = Itf/Y1t;
        S2t = Itf/Y2t;

        S1b = Itf/Y1b;
```

```
	S2b = Itf/Y2b;
//-----------------------------------------------------------------------------------------
	S3tnet = Inet/Y3tnet;				//section 3
	S3bnet = Inet/Y3bnet;

	S3t = Itf/Y3t;
	S3b = Itf/Y3b;
//-----------------------------------------------------------------------------------------
	S_int_sup_tnet = Inet/Y_int_sup_tnet;			//section int_sup
	S_int_sup_bnet = Inet/Y_int_sup_bnet;

	S_int_sup_t = Itf/Y_int_sup_t;
	S_int_sup_b = Itf/Y_int_sup_b;
//-----------------------------------------------------------------------------------------
	S_inf_tnet = Inet/Y_inf_tnet;				//section inf
	S_inf_bnet = Inet/Y_inf_bnet;

	S_inf_t = Itf/Y_inf_t;
	S_inf_b = Itf/Y_inf_b;
//-----------------------------------------------------------------------------------------
	S7tnet = Inet/Y7tnet;				//section 7
	S7bnet = Inet/Y7bnet;

	S7t = Itf/Y7t;
	S7b = Itf/Y7b;
//-----------------------------------------------------------------------------------------
	S8tnet = Inet/Y8tnet;				//section 8
	S8bnet = Inet/Y8bnet;

	S8t = Itf/Y8t;
	S8b = Itf/Y8b;
//--------------------------------------------------------------------------------------
}
void Comp_Sectional_Properties()
{
//	cout<<"comp_sectional_properties"<<"\t";
//	Composite Section Properties
	double EWW;
	EWW = minm(TFRw,12*(TFRd+TFFHd)+Wt+2*TFSHd);
	EFW = 12*ts+EWW;
	EFW = minm(L/4,EFW);
	EFW = minm(EFW,GS);
	Gdc = Gd + ts;
	Atfc = Atf + mratio*EFW*ts;
//---------------------------------------------------------------------------
	Y1bc = (Y1b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;
	Y2bc = (Y2b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;
```

```
            ec1= Y1bc-Y1;                //section 1 and 2
            ec2 = Y2bc - Y2;
            Y1tc = Gdc-Y1bc;
            Y2tc = Gdc-Y2bc;
//------------------------------------------------------------------------
            Y3bc = (Y3b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;

            ec3= Y3bc-Y3;                //section 3
            Y3tc = Gdc-Y3bc;
//------------------------------------------------------------------------
            Y_int_sup_bc = (Y_int_sup_b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;

            ec_int_sup= Y_int_sup_bc-Y_int_sup;             //section int_sup
            Y_int_sup_tc = Gdc-Y_int_sup_bc;
//------------------------------------------------------------------------
            Y_inf_bc = (Y_inf_b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;

            ec_inf= Y_inf_bc-Y_inf;            //section inf
            Y_inf_tc = Gdc-Y_inf_bc;
//------------------------------------------------------------------------
            Y7bc = (Y7b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;

            ec7= Y7bc-Y7;                //section 7
            Y7tc = Gdc-Y7bc;
            //------------------------------------------------------------------------
            Y7bc = (Y7b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;

            ec7= Y7bc-Y7;                //section 8
            Y7tc = Gdc-Y7bc;
//------------------------------------------------------------------------------
            Ic = (mratio*EFW*pow(ts,3)/12)+((mratio*EFW*ts)*pow((Y1tc-ts/2),2));
            Ic = Ic +pow(TFRd,3)*TFRw/12+(TFRd*TFRw)*pow((Y1tc-ts-TFRd/2),2);
            Ic = Ic +((TFFHw*pow(TFFHd,3)/36)*2+((TFFHw*TFFHd)*pow((Y1tc-ts-TFRd-TFFHd/3),2)));
            Ic = Ic +((pow(TFFHd,3)*TFFHbw/12)+(TFFHbw*TFFHd)*pow((Y1tc-ts-TFRd-TFFHd/2),2));
            Ic = Ic +((TFSHw*pow(TFSHd,3)/36)*2+(TFSHw*TFSHd)*pow((Y1tc-ts-TFRd-TFFHd-TFSHd/3),2));
            Ic = Ic +((Wt*pow(TFSHd,3)/12)+(Wt*TFSHd)*pow((Y1tc-ts-TFRd-TFFHd-TFSHd/2),2));
            Ic = Ic +((Wt*pow(Wd,3)/12)+(Wt*Wd)*pow((Y1bc-BFRd-BFHd-Wd/2),2));
            Ic = Ic +((BFHw*pow(BFHd,3)/36)*2+(BFHw*BFHd)*pow((Y1bc-BFRd-BFHd/3),2));
            Ic = Ic +((Wt*pow(BFHd,3)/12+(Wt*BFHd)*pow((Y1bc-BFRd-BFHd/2),2)));
            Ic = Ic +((BFRw*pow(BFRd,3)/12)+(BFRd*BFRw)*pow((Y1bc-BFRd/2),2));
            Ic = Ic +(Es/Ec-1)*As*ec1*ec1;
//------------------------------------------------------------------------------
            S1tc = Ic/(Y1tc - ts);             //section 1 and 2
            S2tc = Ic/(Y2tc - ts);

            S1bc = Ic/Y1bc;
            S2bc = Ic/Y2bc;
//------------------------------------------------------------------------------
```

```
          S3tc = Ic/(Y3tc - ts);                  //section 3
          S3bc = Ic/Y3bc;
//-----------------------------------------------------------------------------
          S_int_sup_tc = Ic/(Y_int_sup_tc - ts);                 //section int_sup
          S_int_sup_bc = Ic/Y_int_sup_bc;
//-----------------------------------------------------------------------------
          S_inf_tc = Ic/(Y_inf_tc - ts);                  //section inf
          S_inf_bc = Ic/Y_inf_bc;
//-----------------------------------------------------------------------------
          S7tc = Ic/(Y7tc - ts);                  //section 7
          S7bc = Ic/Y7bc;
//-----------------------------------------------------------------------------
//        S8tc = Ic/(Y8tc - ts);                  //section 8
//        S8bc = Ic/Y8bc;
//-----------------------------------------------------------------------------
}
//*****************************************************************************************
//__06__06__06__06______________Stiffness Equation Development Zone_____________06__06__06
//*****************************************************************************************
void local_stiffness_matrix_function()
{
//        cout<<"local_stiffness_matrix_function"<<"\t";
           local_stiffness_matrix[0][0]=12*Ec*Itf/(L*L*L);
           local_stiffness_matrix[0][1]=6*Ec*Itf/(L*L);
           local_stiffness_matrix[0][2]=-12*Ec*Itf/(L*L*L);
           local_stiffness_matrix[0][3]=6*Ec*Itf/(L*L);
           local_stiffness_matrix[1][0]=6*Ec*Itf/(L*L);
           local_stiffness_matrix[1][1]=4*Ec*Itf/(L);
           local_stiffness_matrix[1][2]=-6*Ec*Itf/(L*L);
           local_stiffness_matrix[1][3]=2*Ec*Itf/(L);
           local_stiffness_matrix[2][0]=-12*Ec*Itf/(L*L*L);
           local_stiffness_matrix[2][1]=-6*Ec*Itf/(L*L);
           local_stiffness_matrix[2][2]=12*Ec*Itf/(L*L*L);
           local_stiffness_matrix[2][3]=-6*Ec*Itf/(L*L);
           local_stiffness_matrix[3][0]=6*Ec*Itf/(L*L);
           local_stiffness_matrix[3][1]=2*Ec*Itf/L;
           local_stiffness_matrix[3][2]=-6*Ec*Itf/(L*L);
           local_stiffness_matrix[3][3]=4*Ec*Itf/L;
}
//*****************************************************************************************
void global_stiffness_matrix_function()
{
//        cout<<"global_stiffness_matrix_function"<<"\t";
          int increment=0;
          for(int i=0;i<No_of_Span;i++)
          {
                    for(int j=0;j<4;j++)
                              for(int k=0;k<4;k++)
```

```
                    {
                                        global_stiffness_matrix[j+increment][k+increment]=
                                global_stiffness_matrix[j+increment][k+increment]+
           local_stiffness_matrix[j][k];
                    }
                              increment=increment+2;
          }
}
//******************************************************************************
void global_member_force_matrix_function(double w_dyn)
{
//          cout<<"global_member_force_matrix_function"<<"\t";
double local_member_force_matrix[4][1]={{w_dyn*L/2},{w_dyn*L*L/12},
                         {w_dyn*L/2},{-w_dyn*L*L/12}};
          int increment=0;
          for(int i=0;i<No_of_Span;i++)
          {
                    for(int j=0;j<4;j++)
                    {
                              global_member_force_matrix[j+increment][0]=
                       global_member_force_matrix[j+increment][0]+
                              local_member_force_matrix[j][0];
                    }
                    increment=increment+2;
          }
          for(int k=0;k<2*No_of_Node;k++)
          {
                    global_member_force_matrix[k][0]=
                    (-global_member_force_matrix[k][0]);
          }
}
//******************************************************************************
//__07__07__07_____________Boundary Condition Application Zone____________07__07__07__07
//******************************************************************************
void impose_boundarycondition_function()
{
//          cout<<"impose_boundary_condition_function"<<"\t";
          modified_global_stiffness_matrix_function();
          modified_global_member_force_matrix_function();
}
//*****************************************************************************
void modified_global_stiffness_matrix_function()
{
//          cout<<"modified_global_stiffness_matrix_function"<<"\t";
          int increment=0;
          for(int i=0;i<No_of_Node;i++)
          {
                    for(int j=0;j<2*No_of_Node;j++)
```

```
				{
					global_stiffness_matrix[0+increment][j]=0.0;
				}
				for(int k=0;k<2*No_of_Node;k++)
				{
					global_stiffness_matrix[k][0+increment]=0.0;
				}

				global_stiffness_matrix[increment][increment]=1.0;

				increment=increment+2;
		}
}
//*****************************************************************************
void dynamic_allocation_01()
{
//		cout<<"dynamic_allocation_01"<<"\t";
		int i,j,M,N;
		N = 2*No_of_Node;
		M = 2*No_of_Node;
		global_stiffness_matrix_pointer= new double* [N];
		for(i=0; i<N; i++)
		{
				global_stiffness_matrix_pointer[i] = new double[M];
				for(j=0; j<M; j++)
				{
					global_stiffness_matrix_pointer[i][j] = global_stiffness_matrix[i][j];
				}
		}
}
//*****************************************************************************
void modified_global_member_force_matrix_function()
{
//		cout<<"modified_global_member_force_matrix_function"<<"\t";
		int increment=0;
		for(int i=0;i<No_of_Node;i++)
		{
				global_member_force_matrix[increment][0]=0.0;
				increment=increment+2;
		}
}
//*****************************************************************************
void dynamic_allocation_02()
{
//		cout<<"dynamic_allocation_02"<<"\t";
		int i,j,M,N;
N = 2*No_of_Node;
M = 1;
```

```
global_member_force_matrix_pointer= new double* [N];
for(i=0; i<N; i++)
{
	global_member_force_matrix_pointer[i] = new double[M];
	for(j=0; j<M; j++)
	{
		global_member_force_matrix_pointer[i][j] = global_member_force_matrix[i][j];
	}
}
}
//*********************************************************************************
//__08__08__08__08__08____________Stiffness Equation Solution Zone____________08__08__08__
//*********************************************************************************
//Modified stiffness equation (boundary condition apllied)/stiffness equation solution zone
void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
	fprintf(stderr,"Numerical Recipes run-time error...\n");
	fprintf(stderr,"%s\n",error_text);
	fprintf(stderr,"...now exiting to system...\n");
	exit(1);
}
//*********************************************************************************
int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
int *v;
v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
if (!v) nrerror("allocation failure in ivector()");
return v-nl+NR_END;
}
//*********************************************************************************
void free_ivector(int *v, long nl, long nh)
/* free an int vector allocated with ivector() */
{
free((FREE_ARG) (v+nl-NR_END));
}
//*********************************************************************************
void DOF_matrix_solution_function(double **a,int n,double **b,int m)
{
//	cout<<"DOF_matrix_solution_function"<<"\t";
	int *indxc,*indxr,*ipiv;
	int i,icol,irow,j,k,l,ll;
	double big,dum,pivinv,temp;
	indxc=ivector(1,n);   /*the integer arrays ipiv,indxr,and indxc are used for
						bookkeeping on the pivoting*/
	indxr=ivector(1,n);
   ipiv=ivector(1,n);
```

```
	for(j=0;j<n;j++) ipiv[j]=0;
	for(i=0;i<n;i++)   //this is the main loop over the column to be reduced
	{
		big=0.0;
		for(j=0;j<n;j++)  //this is the outer loop of the search for a pivot element
			if(ipiv[j]!=1)
				for(k=0;k<n;k++)
				{
					if(ipiv[k]==0)
					{
						if(fabs(a[j][k])>=big)
						{
							big=fabs(a[j][k]);
							irow=j;
							icol=k;
						}
					}
				}

				++(ipiv[icol]);
				if(irow!=icol)
				{
					for(l=0;l<n;l++) SWAP(a[irow][l],a[icol][l])
						for(l=0;l<n;l++) SWAP(a[irow][l],a[icol][l])
				}
/*we are now ready to divide the pivot row by the pivot element,located at irow and icol*/
				indxr[i]=irow;
				indxc[i]=icol;
				if(a[icol][icol]==0.0) nrerror("gaussj:Singular Matrix");
				pivinv=1.0/a[icol][icol];
				a[icol][icol]=1.0;

				for(l=0;l<n;l++) a[icol][l]*=pivinv;
				for(l=0;l<m;l++) b[icol][l]*=pivinv;

				for(ll=0;ll<n;ll++)
					if(ll!=icol)
					{
						dum=a[ll][icol];
						a[ll][icol]=0.0;
						for(l=0;l<n;l++) a[ll][l]-=a[icol][l]*dum;
						for(l=0;l<m;l++) b[ll][l]-=b[icol][l]*dum;
					}
	}

	for(l=n;l>=1;l--)
	{
```

```
            if(indxr[l]!=indxc[l])
                for(k=0;k<n;k++)
                    SWAP(a[k][indxr[l]],a[k][indxc[l]]);
        }

        free_ivector(ipiv,1,n);
        free_ivector(indxr,1,n);
        free_ivector(indxc,1,n);

        for(int p=0;p<2*No_of_Node;p++)
        {
            final_DOF_matrix[p][0]=global_member_force_matrix_pointer[p][0];
        }
}
//********************************************************************************
//__09__09__09__09_____________End Shear and Moment Calculation Zone____________09__09__
//********************************************************************************
/*For each member- end shear and moment calculation zone (using solved values of DOFs)
(matrix form)*/
void total_DL_end_shearforce_bendingmoment_function(double w_dyn)
{
//      cout<<"total_DL_end_shearforce_bendingmoment_function"<<"\t";
        int increment=0;
        for(int i=0;i<No_of_Span;i++)
        {

            total_DL_end_shearforce_bendingmoment_matrix[i][0]=
            +w_dyn*L/2.0
            +final_DOF_matrix[i+increment+1][0]*6.0*Ec*Itf/(L*L)
            +final_DOF_matrix[i+increment+3][0]*6.0*Ec*Itf/(L*L);   //left end shear

            total_DL_end_shearforce_bendingmoment_matrix[i][1]=
            -w_dyn*L*L/12.0
          -final_DOF_matrix[i+increment+1][0]*4.0*Ec*Itf/L
            -final_DOF_matrix[i+increment+3][0]*2.0*Ec*Itf/L;     //left end moment

            total_DL_end_shearforce_bendingmoment_matrix[i][2]=
            -w_dyn*L/2.0
            +final_DOF_matrix[i+increment+1][0]*6.0*Ec*Itf/(L*L)
          +final_DOF_matrix[i+increment+3][0]*6.0*Ec*Itf/(L*L);   //right end shear

            total_DL_end_shearforce_bendingmoment_matrix[i][3]=
            -w_dyn*L*L/12.0
            +final_DOF_matrix[i+increment+1][0]*2.0*Ec*Itf/L
            +final_DOF_matrix[i+increment+3][0]*4.0*Ec*Itf/L;     //right end moment

            increment=increment+1;
        }
```

```
}
//*****************************************************************************
void for_ILvalue_LL_end_shearforce_bendingmoment_function(int increment_a,int increment_c)
{
//	cout<<"for_ILvalue_LL_end_shearforce_bendingmoment_function"<<"\t";
	int constant=0;
	int increment=0;
	for(int i=0;i<No_of_Span;i++)
	{
		for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][0]=
		+final_DOF_matrix[i+increment+1][0]*6.0*Ec*Itf/(L*L)
		+final_DOF_matrix[i+increment+3][0]*6.0*Ec*Itf/(L*L);	//left end shear

		for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][1]=
	   -final_DOF_matrix[i+increment+1][0]*4.0*Ec*Itf/L
		-final_DOF_matrix[i+increment+3][0]*2.0*Ec*Itf/L;	//left end moment

    for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][2]=
		+final_DOF_matrix[i+increment+1][0]*6.0*Ec*Itf/(L*L)
	   +final_DOF_matrix[i+increment+3][0]*6.0*Ec*Itf/(L*L);	//right end shear

		for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][3]=
		+final_DOF_matrix[i+increment+1][0]*2.0*Ec*Itf/L
		+final_DOF_matrix[i+increment+3][0]*4.0*Ec*Itf/L;	//right end moment

		increment=increment+1;
	}
	constant=increment_c/2;

	for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][0]=
	+(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	-(increment_a*0.25*y)*(increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	+(50-increment_a*0.25)/50.0
	+final_DOF_matrix[increment_c+1][0]*6.0*Ec*Itf/(L*L)
	+final_DOF_matrix[increment_c+3][0]*6.0*Ec*Itf/(L*L);	//left end shear

	for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][1]=
	-(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y)
	-final_DOF_matrix[increment_c+1][0]*4.0*Ec*Itf/L
	-final_DOF_matrix[increment_c+3][0]*2.0*Ec*Itf/L;	//left end moment

  for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][2]=
	-(increment_a*0.25*y)*(increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	+(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
```

```
        *(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
        -(increment_a*0.25)/50.0
        +final_DOF_matrix[increment_c+1][0]*6.0*Ec*Itf/(L*L)
  +final_DOF_matrix[increment_c+3][0]*6.0*Ec*Itf/(L*L);   //right end shear

        for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][3]=
        -(increment_a*0.25*y)*(increment_a*0.25*y)
        *(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y)
        +final_DOF_matrix[increment_c+1][0]*2.0*Ec*Itf/L
        +final_DOF_matrix[increment_c+3][0]*4.0*Ec*Itf/L;     //right end moment
}
/*above formula are written for following sign convention:
---> left end/section:   upward shear positive;     clockwise moment positive
---> right end/section:  downward shear positive;   anti-clockwise moment positive*/
//*******************************************************************************
//__10__10__10__10_______Sectionwise DL Shear And Moment Calculation Zone____10__10__10__
//*******************************************************************************
// for each member sectionwise shear and moment calculation zone (matrix form)
/*for finding shear at every 0.25 metre interval*/
 void total_DL_sectionwise_shearforce_function(double w_dyn)
{
//      cout<<"total_DL_sectionwise_shearforce_function"<<"\t";
        int increment=0;

        for(int i=0;i<No_of_Span;i++)
        {
                for(int j=0;j<201;j++)
                {
                        total_DL_sectionwise_shearforce_matrix[0][j+increment]=
                        +total_DL_end_shearforce_bendingmoment_matrix[i][0]
                        -w_dyn*j*(0.25)*(y);   //in fps
                }
                increment=increment+201;
        }
}
//*******************************************************************************
/*for finding moment at every 0.25 metre*/
void total_DL_sectionwise_bendingmoment_function(double w_dyn)
{
//      cout<<"total_DL_sectionwise_bendingmoment_function"<<"\t";
        int increment=0;

        for(int i=0;i<No_of_Span;i++)
        {
                for(int j=0;j<201;j++)
                {
                        total_DL_sectionwise_bendingmoment_matrix[0][j+increment]=
                        +total_DL_end_shearforce_bendingmoment_matrix[i][1]
```

```
                    +total_DL_end_shearforce_bendingmoment_matrix[i][0]*j*(0.25)*(y)
                          -w_dyn*j*(0.25)*(y)*j*(0.25)*(y)/2.0;
                }
                increment=increment+201;
        }
}
//*********************************************************************************
//__11__11__11__11_____________Influence Line Development Zone____________11__11__11__11
//*********************************************************************************
void influence_line_function()
{
//      cout<<"influence_line_function"<<"\t";
        int constant=0;
        int increment_a=1;
        int increment_b=1;
        int increment_c=0;

        for(int k=0;k<No_of_Span;k++)
        {
                for(int l=0;l<199;l++)  /*for influence line values at every 0.25 metre interval.*/
                {
                        dynamic_allocation_01();
                        global_member_force_matrix_function_forIL(increment_a,increment_c);
                        modified_global_member_force_matrix_function();
                        dynamic_allocation_02();
                    DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                                                 global_member_force_matrix_pointer,x);

        for_ILvalue_LL_end_shearforce_bendingmoment_function(increment_a,increment_c);
                        for_ILvalue_LL_sectionwise_shearforce_function(increment_a,increment_c);

        for_ILvalue_LL_sectionwise_bendingmoment_function(increment_a,increment_c);

                        for(int n=0;n<No_of_Span*201;n++)
                        {
                                IL_matrix_shear[n][0+increment_b]=
                                for_ILvalue_LL_sectionwise_shearforce_matrix[0][n];

                                IL_matrix_moment[n][0+increment_b]=
                                for_ILvalue_LL_sectionwise_bendingmoment_matrix[0][n];
                        }
                        increment_a=increment_a+1;
                        increment_b=increment_b+1;
                }
                increment_a=1;
                increment_b=increment_b+1;
                increment_c=increment_c+2;
        }
```

```
}
//*********************************************************************************
void global_member_force_matrix_function_forIL(int increment_a,int increment_c)
{
//	cout<<"global_member_force_matrix_function_forIL"<<"\t";
	for(int m=0;m<2*No_of_Node;m++)
	{
		global_member_force_matrix[m][0]=0.0;
	}

	global_member_force_matrix[0+increment_c][0]=
	+(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	-(increment_a*0.25*y)*(increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	+(50-increment_a*0.25)/50.0;

	global_member_force_matrix[1+increment_c][0]=
	(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y);

   global_member_force_matrix[2+increment_c][0]=
	+(increment_a*0.25*y)*(increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	-(increment_a*0.25*y)*(50.0*y-increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y*50.0*y)
	+(increment_a*0.25)/50.0;

   global_member_force_matrix[3+increment_c][0]=
	-(increment_a*0.25*y)*(increment_a*0.25*y)
	*(50.0*y-increment_a*0.25*y)/(50.0*y*50.0*y);

	for(int k=0;k<2*No_of_Node;k++)
	{
		global_member_force_matrix[k][0]=
		(-global_member_force_matrix[k][0]);
	}
}
//*********************************************************************************
void for_ILvalue_LL_sectionwise_shearforce_function(int increment_a,int increment_c)
{
//	cout<<"for_ILvalue_LL_sectionwise_shearforce_function"<<"\t";
	int constant=0;
	int increment=0;				/*for finding shear at every 0.25 metre interval*/

	for(int i=0;i<No_of_Span;i++)
	{
		for(int j=0;j<201;j++)
```

```
			{
				for_ILvalue_LL_sectionwise_shearforce_matrix[0][j+increment]=
				+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][0];
			}
			increment=increment+201;
		}
		constant=increment_c/2;
		for(int k=0;k<=increment_a;k++)
		{
			for_ILvalue_LL_sectionwise_shearforce_matrix[0][k+201*constant]=
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][0];
		}
		for(int l=1;l<=200-increment_a;l++)
		{
			for_ILvalue_LL_sectionwise_shearforce_matrix[0][l+201*constant+increment_a]=
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][0]-1;
		}
}
//*****************************************************************************
void for_ILvalue_LL_sectionwise_bendingmoment_function(int increment_a,int increment_c)
{
//		cout<<"for_ILvalue_LL_sectionwise_bendingmoment_function"<<"\t";
		int constant=0;
		int increment=0;				/*for finding moment at every 0.25 metre interval*/
		for(int i=0;i<No_of_Span;i++)
		{
			for(int j=0;j<201;j++)
			{
				for_ILvalue_LL_sectionwise_bendingmoment_matrix[0][j+increment]=
				+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][1]
				+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[i][0]*j*(0.25)*(y);
			}

			increment=increment+201;
		}
		constant=increment_c/2;
		for(int k=0;k<=increment_a;k++)
		{
			for_ILvalue_LL_sectionwise_bendingmoment_matrix[0][k+201*constant]=
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][1]
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][0]*k*(0.25)*(y);
		}
		for(int l=1;l<=200-increment_a;l++)
		{
			for_ILvalue_LL_sectionwise_bendingmoment_matrix[0][l+201*constant+increment_a]=
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][1]
			+for_ILvalue_LL_end_shearforce_bendingmoment_matrix[constant][0]
			*(l+increment_a)*(0.25)*(y)
```

```
                -1*l*(0.25)*(y);
        }
}

//*********************************************************************************
//__12__12__12________Girder DL & LL (Moment & Shear) Combination Zone______12__12__12__
//*********************************************************************************
void factored_DLplusLL_shearCombination_atSpecificSection_function
(int whichSection_intermsof_ILmatrixrow)
{
//cout<<"factored_DLplusLL_shearCombination_atSpecificSection_function"<<"\t";

        double factored_DLplusLL_shear_01=0;
        double factored_DLplusLL_shear_02=0;
        double factored_DLplusLL_shear_absolute=0;

        factored_DLplusLL_shear_01=1.3*total_DL_sectionwise_shearforce_matrix
                        [0][whichSection_intermsof_ILmatrixrow]+2.17

*total_LL_atSpecific_section_positive_shearforce_function
                (whichSection_intermsof_ILmatrixrow);

        factored_DLplusLL_shear_01=absolue_value_function(factored_DLplusLL_shear_01);

        factored_DLplusLL_shear_02=1.3*total_DL_sectionwise_shearforce_matrix
                        [0][whichSection_intermsof_ILmatrixrow]+2.17

*total_LL_atSpecific_section_negative_shearforce_function
                (whichSection_intermsof_ILmatrixrow);

        factored_DLplusLL_shear_02=absolue_value_function(factored_DLplusLL_shear_02);

        factored_DLplusLL_shear_absolute=maxm(factored_DLplusLL_shear_01,
                                factored_DLplusLL_shear_02);
}
//*********************************************************************************
void factored_DLplusLL_momentCombination_atSpecificSection_function
(int whichSection_intermsof_ILmatrixrow)
{
//cout<<"factored_DLplusLL_momentCombination_atSpecificSection_function"<<"\t";
        double factored_DLplusLL_moment_01=0;
        double factored_DLplusLL_moment_02=0;
        double factored_DLplusLL_moment_positive=0;
        double factored_DLplusLL_moment_negative=0;

        factored_DLplusLL_moment_01=1.3*total_DL_sectionwise_bendingmoment_matrix
                        [0][whichSection_intermsof_ILmatrixrow]+2.17
```

```
*total_LL_atSpecific_section_positive_bendingmoment_function
                        (whichSection_intermsof_ILmatrixrow);

        factored_DLplusLL_moment_02=1.3*total_DL_sectionwise_bendingmoment_matrix
                              [0][whichSection_intermsof_ILmatrixrow]+2.17

*total_LL_atSpecific_section_negative_bendingmoment_function
                        (whichSection_intermsof_ILmatrixrow);

        if(factored_DLplusLL_moment_01>0 && factored_DLplusLL_moment_02>0)
        {
                factored_DLplusLL_moment_positive=
                maxm(factored_DLplusLL_moment_01,factored_DLplusLL_moment_02);
        }

        else if(factored_DLplusLL_moment_01<0 && factored_DLplusLL_moment_02<0)
        {
                factored_DLplusLL_moment_negative=
                minm(factored_DLplusLL_moment_01,factored_DLplusLL_moment_02);
        }

        else if(factored_DLplusLL_moment_01>0 && factored_DLplusLL_moment_02<0)
        {
                factored_DLplusLL_moment_positive=factored_DLplusLL_moment_01;
                factored_DLplusLL_moment_negative=factored_DLplusLL_moment_02;
        }
        else
        {
                factored_DLplusLL_moment_positive=factored_DLplusLL_moment_02;
                factored_DLplusLL_moment_negative=factored_DLplusLL_moment_01;
        }
}
//*******************************************************************************
double total_LL_atSpecific_section_positive_shearforce_function
(int whichSection_intermsof_ILmatrixrow)
{
//      cout<<"total_LL_atSpecific_section_positive_shearforce_function"<<"\t";
//maxm positive 'wheel load' shear calculation
//-------------------------------------------
//for vehicle going from left to right
        for(int i=0;i<=whichSection_intermsof_ILmatrixrow-1;i++)
        {
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][i]=
          IL_matrix_shear[whichSection_intermsof_ILmatrixrow][i];
        }

        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01
```

```
[0][whichSection_intermsof_ILmatrixrow]=
IL_matrix_shear
[whichSection_intermsof_ILmatrixrow][whichSection_intermsof_ILmatrixrow]-1;

for(int i2=whichSection_intermsof_ILmatrixrow;i2<200*No_of_Span+1;i2++)
{
        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][i2+1]=
  IL_matrix_shear[whichSection_intermsof_ILmatrixrow][i2];
}

for(int j=0;j<17;j++)
{
        if(wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j]>0
                &&
          wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j]>
          max_wheelLoad_shearPositive)
        {
                max_wheelLoad_shearPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j];
        }
}

for(int k=17;k<34;k++)
{
        if(wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
          wheel_load_rearAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17]>0
                &&
          wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
          wheel_load_rearAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17]>
          max_wheelLoad_shearPositive)
        {
                max_wheelLoad_shearPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17];
        }
}

for(int l=34;l<=200*No_of_Span+2;l++)
{
```

```
            if(wheel_load_frontAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
              wheel_load_rearAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
              wheel_load_rearAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34]>0
                    &&
              wheel_load_frontAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
              wheel_load_rearAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
              wheel_load_rearAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34]>
              max_wheelLoad_shearPositive)
            {
                    max_wheelLoad_shearPositive=
                    wheel_load_frontAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
                    wheel_load_rearAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
                    wheel_load_rearAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34];
            }
    }

//for vehicle going from right to left

    for(int ii=0;ii<200*No_of_Span+2;ii++)
    {
            matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ii]=
            matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][200*No_of_Span+1-ii];
    }

    for(int jj=0;jj<17;jj++)
    {
            if(wheel_load_frontAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj]>0
                    &&
              wheel_load_frontAxle*
              matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj]>
              max_wheelLoad_shearPositive)
            {
                    max_wheelLoad_shearPositive=
                    wheel_load_frontAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj];
            }
    }
```

```
for(int kk=17;kk<34;kk++)
{
	if(wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17]>0
		&&
	  wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17]>
	  max_wheelLoad_shearPositive)
	{
		max_wheelLoad_shearPositive=
		wheel_load_frontAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
		wheel_load_rearAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17];
	}
}

for(int ll=34;ll<200*No_of_Span+2;ll++)
{
	if(wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34]>0
		&&
	  wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34]>
	  max_wheelLoad_shearPositive)
	{
		max_wheelLoad_shearPositive=
		wheel_load_frontAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
		wheel_load_rearAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
		wheel_load_rearAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34];
	}
}
```

```
//Impact Load Factor//loaded length
//---------------------------------

   loadedLength_max_wheelLoad_shearPositive=(whichSection_intermsof_ILmatrixrow%200)*y;

        if (loadedLength_max_wheelLoad_shearPositive<100*y)
        {
                loadedLength_max_wheelLoad_shearPositive=
                200*y-loadedLength_max_wheelLoad_shearPositive;
        }
        ImpactFactor_max_wheelLoad_shearPositive=
        50/(loadedLength_max_wheelLoad_shearPositive+125);

        if(ImpactFactor_max_wheelLoad_shearPositive>0.3)
        {
                ImpactFactor_max_wheelLoad_shearPositive=0.3;
        }

        max_wheelLoad_shearPositive=max_wheelLoad_shearPositive
                      *(1+ImpactFactor_max_wheelLoad_shearPositive);
//*****************************************************************************
//maxm positive 'lane load' shear calculation

        for(int m=0;m<200*No_of_Span+2;m++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m]>=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m+1]>=0)
                {
                        max_laneLoad_shearPositive=
                        max_laneLoad_shearPositive+
                        lane_load_UDL_forShear*0.25*y*
                        (matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m]+

        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m+1])/2;//checksequence

                        loadedLength_max_laneLoad_shearPositive=
                        loadedLength_max_laneLoad_shearPositive+1;
                }
        }

//finding laneLoad_pick

        for(int n=0;n<200*No_of_Span+2;n++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n]>=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n+1]>=0)
```

```
			{
				laneLoad_pick=
				maxm(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n],
					matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n+1]);
			}
		}

		laneLoad_pick=laneLoad_pick*lane_load_Concentrated_forShear;

		max_laneLoad_shearPositive=
		max_laneLoad_shearPositive+laneLoad_pick;

//Impact Load Factor//loaded length

loadedLength_max_laneLoad_shearPositive=loadedLength_max_laneLoad_shearPositive*0.25*y;

		ImpactFactor_max_laneLoad_shearPositive=
		50/(loadedLength_max_laneLoad_shearPositive+125);

		if(ImpactFactor_max_laneLoad_shearPositive>0.3)
		{
			ImpactFactor_max_laneLoad_shearPositive=0.3;
		}

		max_laneLoad_shearPositive=max_laneLoad_shearPositive
			*(1+ImpactFactor_max_laneLoad_shearPositive);

		return maxm(max_wheelLoad_shearPositive,max_laneLoad_shearPositive);
}
//*******************************************************************************
double total_LL_atSpecific_section_negative_shearforce_function
(int whichSection_intermsof_ILmatrixrow)
{
//	cout<<"total_LL_atSpecific_section_negative_shearforce_function"<<"\t";

//maxm negative 'wheel load' shear calculation
//-------------------------------------------

//for vehicle going from left to right

		for(int i=0;i<=whichSection_intermsof_ILmatrixrow-1;i++)
		{
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][i]=
		   IL_matrix_shear[whichSection_intermsof_ILmatrixrow][i];
		}

		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01
		[0][whichSection_intermsof_ILmatrixrow]=
```

```
IL_matrix_shear
[whichSection_intermsof_ILmatrixrow][whichSection_intermsof_ILmatrixrow]-1;

for(int i2=whichSection_intermsof_ILmatrixrow;i2<200*No_of_Span+1;i2++)
{
	matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][i2+1]=
   IL_matrix_shear[whichSection_intermsof_ILmatrixrow][i2];
}

for(int j=0;j<17;j++)
{
	if(wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j]<0
		&&
	  wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j]<
	  max_wheelLoad_shearNegative)
	{
		max_wheelLoad_shearNegative=
		wheel_load_frontAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][j];
	}
}

for(int k=17;k<34;k++)
{
	if(wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17]<0
		&&
	  wheel_load_frontAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
	  wheel_load_rearAxle*
	  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17]<
	  max_wheelLoad_shearNegative)
	{
		max_wheelLoad_shearNegative=
		wheel_load_frontAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k]+
		wheel_load_rearAxle*
		matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][k-17];
	}
}

for(int l=34;l<=200*No_of_Span+2;l++)
{
	if(wheel_load_frontAxle*
```

```
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34]<0
                        &&
                  wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34]<
                  max_wheelLoad_shearNegative)
                {
                        max_wheelLoad_shearNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-17]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][l-34];
                }
        }

//for vehicle going from right to left

        for(int ii=0;ii<200*No_of_Span+2;ii++)
        {
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ii]=
                matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][200*No_of_Span+1-ii];
        }

        for(int jj=0;jj<17;jj++)
        {
                if(wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj]<0
                        &&
                  wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj]<
                  max_wheelLoad_shearNegative)
                {
                        max_wheelLoad_shearNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][jj];
                }
        }

        for(int kk=17;kk<34;kk++)
```

```
	{
		if(wheel_load_frontAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17]<0
			&&
		  wheel_load_frontAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17]<
		  max_wheelLoad_shearNegative)
		{
			max_wheelLoad_shearNegative=
			wheel_load_frontAxle*
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk]+
			wheel_load_rearAxle*
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][kk-17];
		}
	}

	for(int ll=34;ll<200*No_of_Span+2;ll++)
	{
		if(wheel_load_frontAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34]<0
			&&
		  wheel_load_frontAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
		  wheel_load_rearAxle*
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34]<
		  max_wheelLoad_shearNegative)
		{
			max_wheelLoad_shearNegative=
			wheel_load_frontAxle*
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll]+
			wheel_load_rearAxle*
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-17]+
			wheel_load_rearAxle*
			matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_02[0][ll-34];
		}
	}

//Impact Load Factor//loaded length
```

```
//---------------------------------

  loadedLength_max_wheelLoad_shearNegative=(whichSection_intermsof_ILmatrixrow%200)*y;

        if(loadedLength_max_wheelLoad_shearNegative<100*y)
        {
                loadedLength_max_wheelLoad_shearNegative=
                200*y-loadedLength_max_wheelLoad_shearNegative;
        }

        ImpactFactor_max_wheelLoad_shearNegative=
        50/(loadedLength_max_wheelLoad_shearNegative+125);

        if(ImpactFactor_max_wheelLoad_shearNegative>0.3)
        {
                ImpactFactor_max_wheelLoad_shearNegative=0.3;
        }

        max_wheelLoad_shearNegative=max_wheelLoad_shearNegative
                *(1+ImpactFactor_max_wheelLoad_shearNegative);

//******************************************************************************
//maxm Negative 'lane load' shear calculation

        for(int m=0;m<200*No_of_Span+2;m++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m]<=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m+1]<=0)
                {
                        max_laneLoad_shearNegative=
                        max_laneLoad_shearNegative+
                        lane_load_UDL_forShear*0.25*y*
                        (matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m]+

        matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][m+1])/2;//checksequence

                        loadedLength_max_laneLoad_shearNegative=
                        loadedLength_max_laneLoad_shearNegative+1;
                }
        }

//finding laneLoad_pick

        for(int n=0;n<200*No_of_Span+2;n++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n]<=0
                        &&
```

```
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n+1]<=0)
                {
                        laneLoad_pick=
                        minm(matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n],
                           matrix_forSorting_maXof_wheelLoadorlaneLoad_shear_01[0][n+1]);
                }
        }

        laneLoad_pick=laneLoad_pick*lane_load_Concentrated_forShear;

        max_laneLoad_shearNegative=
        max_laneLoad_shearNegative+laneLoad_pick;

//Impact Load Factor//loaded length

loadedLength_max_laneLoad_shearNegative=loadedLength_max_laneLoad_shearNegative*0.25*y;

        ImpactFactor_max_laneLoad_shearNegative=
        50/(loadedLength_max_laneLoad_shearNegative+125);

        if(ImpactFactor_max_laneLoad_shearNegative>0.3)
        {
                ImpactFactor_max_laneLoad_shearNegative=0.3;
        }

        max_laneLoad_shearNegative=max_laneLoad_shearNegative
                    *(1+ImpactFactor_max_laneLoad_shearNegative);

        return minm(max_wheelLoad_shearNegative,max_laneLoad_shearNegative);
}
//*******************************************************************************
double total_LL_atSpecific_section_positive_bendingmoment_function
(int whichSection_intermsof_ILmatrixrow)
{
//      cout<<"total_LL_atSpecific_section_positive_bendingmoment_function"<<"\t";
//maxm positive 'wheel load' moment calculation
//-------------------------------------------
//for vehicle going from left to right
        for(int i=0;i<200*No_of_Span+1;i++)
        {
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][i]=
          IL_matrix_moment[whichSection_intermsof_ILmatrixrow][i];
        }

        for(int j=0;j<17;j++)
        {
                if(wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j]>0
```

```
                &&
          wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j]>
          max_wheelLoad_momentPositive)
        {
                max_wheelLoad_momentPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j];
        }
    }

    //cout<<max_wheelLoad_momentPositive<<endl;

    for(int k=17;k<34;k++)
    {
        if(wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
          wheel_load_rearAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17]>0
                &&
          wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
          wheel_load_rearAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17]>
          max_wheelLoad_momentPositive)
        {
                max_wheelLoad_momentPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17];
        }
    }

    //cout<<max_wheelLoad_momentPositive<<endl;
    //cout<<max_wheelLoad_momentPositive<<endl;
//for vehicle going from right to left

    for(int ii=0;ii<200*No_of_Span+1;ii++)
    {
        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ii]=
        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][200*No_of_Span-ii];
    }

    for(int jj=0;jj<17;jj++)
    {
        if(wheel_load_frontAxle*
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj]>0
```

```
                &&
            wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj]>
            max_wheelLoad_momentPositive)
        {
                max_wheelLoad_momentPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj];
        }
}

for(int kk=17;kk<34;kk++)
{
        if(wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17]>0
                &&
            wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17]>
            max_wheelLoad_momentPositive)
        {
                max_wheelLoad_momentPositive=
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17];
        }
}

for(int ll=34;ll<=200*No_of_Span+1;ll++)
{
        if(wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34]>0
                &&
            wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34]>
            max_wheelLoad_momentPositive)
```

```
                {
                        max_wheelLoad_momentPositive=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34];
                }
        }
        //cout<<max_wheelLoad_momentPositive<<endl;
//Impact Load Factor//loaded length
//---------------------------------
        loadedLength_max_wheelLoad_momentPositive=50*y;

        ImpactFactor_max_wheelLoad_momentPositive=
        50/(loadedLength_max_wheelLoad_momentPositive+125);

        if(ImpactFactor_max_wheelLoad_momentPositive>0.3)
        {
                ImpactFactor_max_wheelLoad_momentPositive=0.3;
        }

        max_wheelLoad_momentPositive=max_wheelLoad_momentPositive
                    *(1+ImpactFactor_max_wheelLoad_momentPositive);
//*****************************************************************************
//maxm positive 'lane load' moment calculation

        for(int m=0;m<200*No_of_Span;m++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m]>=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m+1]>=0)
                {
                        max_laneLoad_momentPositive=
                        max_laneLoad_momentPositive+
                        lane_load_UDL_forMoment*0.25*y*
                        (matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m]+

        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m+1])/2;//checksequenc

                        loadedLength_max_laneLoad_momentPositive=
                        loadedLength_max_laneLoad_momentPositive+1;
                }
        }

//finding laneLoad_pick
```

```
	for(int n=0;n<200*No_of_Span;n++)
	{
		if(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n]>=0
			&&
		  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n+1]>=0)
		{
			laneLoad_pick=
			maxm(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n],
		    matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n+1]);
		}
	}

	//cout<<laneLoad_pick<<endl;

	laneLoad_pick=laneLoad_pick*lane_load_Concentrated_forMoment;

	max_laneLoad_momentPositive=
	max_laneLoad_momentPositive+laneLoad_pick;

//Impact Load Factor//loaded length

	loadedLength_max_laneLoad_momentPositive=
	loadedLength_max_laneLoad_momentPositive*0.25*y;

 //  cout<<loadedLength_max_laneLoad_momentPositive<<endl;

	ImpactFactor_max_laneLoad_momentPositive=
	50/(loadedLength_max_laneLoad_momentPositive+125);

	if(ImpactFactor_max_laneLoad_momentPositive>0.3)
	{
		ImpactFactor_max_laneLoad_momentPositive=0.3;
	}

	max_laneLoad_momentPositive=max_laneLoad_momentPositive
		      *(1+ImpactFactor_max_laneLoad_momentPositive);

	return maxm(max_wheelLoad_momentPositive,max_laneLoad_momentPositive);
}
//*******************************************************************************
double total_LL_atSpecific_section_negative_bendingmoment_function
(int whichSection_intermsof_ILmatrixrow)
{
//	cout<<"total_LL_atSpecific_section_negative_bendingmoment_function"<<"\t";

//maxm negative 'wheel load' moment calculation
//-------------------------------------------
```

```
//for vehicle going from left to right
        for(int i=0;i<200*No_of_Span+1;i++)
        {
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][i]=
          IL_matrix_moment[whichSection_intermsof_ILmatrixrow][i];
        }

        for(int j=0;j<17;j++)
        {
                if(wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j]<0
                        &&
                  wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j]<
                  max_wheelLoad_momentNegative)
                {
                        max_wheelLoad_momentNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][j];
                }
        }

        for(int k=17;k<34;k++)
        {
                if(wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17]<0
                        &&
                  wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
                  wheel_load_rearAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17]<
                  max_wheelLoad_momentNegative)
                {
                        max_wheelLoad_momentNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][k-17];
                }
        }

        for(int l=34;l<=200*No_of_Span+1;l++)
        {
                if(wheel_load_frontAxle*
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l]+
                  wheel_load_rearAxle*
```

```
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-17]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-34]<0
                    &&
            wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-17]+
            wheel_load_rearAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-34]<
            max_wheelLoad_momentNegative)
          {
                    max_wheelLoad_momentNegative=
                    wheel_load_frontAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l]+
                    wheel_load_rearAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-17]+
                    wheel_load_rearAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][l-34];
          }
      }

//for vehicle going from right to left

      for(int ii=0;ii<200*No_of_Span+1;ii++)
      {
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ii]=
          matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][200*No_of_Span-ii];
      }

      for(int jj=0;jj<17;jj++)
      {
          if(wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj]<0
                    &&
            wheel_load_frontAxle*
            matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj]<
            max_wheelLoad_shearNegative)
          {
                    max_wheelLoad_momentNegative=
                    wheel_load_frontAxle*
                    matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][jj];
          }
      }

      for(int kk=17;kk<34;kk++)
      {
          if(wheel_load_frontAxle*
```

```
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17]<0
                        &&
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17]<
                max_wheelLoad_momentNegative)
              {
                        max_wheelLoad_shearNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][kk-17];
              }
        }

        for(int ll=34;ll<=200*No_of_Span+1;ll++)
        {
              if(wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34]<0
                        &&
                wheel_load_frontAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
                wheel_load_rearAxle*
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34]<
                max_wheelLoad_momentNegative)
              {
                        max_wheelLoad_momentNegative=
                        wheel_load_frontAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-17]+
                        wheel_load_rearAxle*
                        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_02[0][ll-34];
              }
        }
//Impact Load Factor//loaded length
//--------------------------------
        loadedLength_max_wheelLoad_momentNegative=50*y;
```

```
        ImpactFactor_max_wheelLoad_momentNegative=
        50/(loadedLength_max_wheelLoad_momentNegative+125);

        if(ImpactFactor_max_wheelLoad_momentNegative>0.3)
        {
                ImpactFactor_max_wheelLoad_momentNegative=0.3;
        }

        max_wheelLoad_shearNegative=max_wheelLoad_momentNegative
                    *(1+ImpactFactor_max_wheelLoad_momentNegative);
//*****************************************************************************
//maxm Negative 'lane load' moment calculation

        for(int m=0;m<201*No_of_Span;m++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m]<=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m+1]<=0)
                {
                        max_laneLoad_momentNegative=
                        max_laneLoad_momentNegative+
                        lane_load_UDL_forMoment*0.25*y*
                        (matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m]+

        matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][m+1])/2;//checsequence

                        loadedLength_max_laneLoad_momentNegative=
                        loadedLength_max_laneLoad_momentNegative+1;
                }
        }

//finding laneLoad_pick

        for(int n=0;n<201*No_of_Span;n++)
        {
                if(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n]<=0
                        &&
                  matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n+1]<=0)
                {
                        laneLoad_pick=
                        minm(matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n],
                matrix_forSorting_maXof_wheelLoadorlaneLoad_moment_01[0][n+1]);
                }
        }

        laneLoad_pick=laneLoad_pick*lane_load_Concentrated_forMoment;

        max_laneLoad_momentNegative=
```

```
	max_laneLoad_momentNegative+laneLoad_pick;

//Impact Load Factor//loaded length

	loadedLength_max_laneLoad_momentNegative=
	loadedLength_max_laneLoad_momentNegative*0.25*y;

	ImpactFactor_max_laneLoad_momentNegative=
	50/(loadedLength_max_laneLoad_momentNegative+125);

	if(ImpactFactor_max_laneLoad_momentNegative>0.3)
	{
		ImpactFactor_max_laneLoad_momentNegative=0.3;
	}

	max_laneLoad_momentNegative=max_laneLoad_momentNegative
		*(1+ImpactFactor_max_laneLoad_momentNegative);

	return minm(max_wheelLoad_momentNegative,max_laneLoad_momentNegative);
}
//*****************************************************************************
//13__13__13__13__13____________Cable layout Function Zone____________13__13__13__13__13_
//*****************************************************************************

void Cable_layout(double cable_1st_position_end)
{
//	cout<<"cable_layout"<<"\t";

	double Layer_dist_bottom_end[31],N_total,Ncablet;
	int Cable_Layer_end[31],LayerNo_end,k1;
//	Cable Layout
	Anchorage_system();
// No of cable per Layer determination at mid section
	Cable_Layer_mid[1] = (int)(BFRw-76-Duct_dia)/(Duct_clear_spacing+Duct_dia)+1;
	Layer_dist_bottom_mid[1] = 38 + Duct_dia/2;

	LayerNo_mid = 1;
	N_total = 0;
	Ncablet = RNDOFF(&Ncable);
	for( ; ; )
	{
		N_total = N_total+Cable_Layer_mid[LayerNo_mid];
		if(N_total<Ncablet)
		{
			LayerNo_mid = LayerNo_mid+1;
			Layer_dist_bottom_mid[LayerNo_mid] = Layer_dist_bottom_mid[LayerNo_mid-
1] + Duct_clear_spacing+Duct_dia;
			if(Layer_dist_bottom_mid[LayerNo_mid]<= BFRd)
```

```
                    Cable_Layer_mid[LayerNo_mid] = Cable_Layer_mid[LayerNo_mid-1];
                else if(Layer_dist_bottom_mid[LayerNo_mid]<= (BFRd+BFHd))
                    Cable_Layer_mid[LayerNo_mid] = int((BFRw-
(Layer_dist_bottom_mid[LayerNo_mid]-BFRd)/BFHd*BFHw*2-76-
Duct_dia)/(Duct_clear_spacing+Duct_dia) + 1);
                else
                    Cable_Layer_mid[LayerNo_mid] = 1;
            }
            else
            {
                Cable_Layer_mid[LayerNo_mid] = int(Ncablet-(N_total -
Cable_Layer_mid[LayerNo_mid]));
                break;
            }
        }
//      Cg of cables at mid section from bottom,
        Y1 = 0;
        for(int i=1;i<=LayerNo_mid;i++)
        {
            Y1      = Y1 + Cable_Layer_mid[i]*Layer_dist_bottom_mid[i];
            if(i==LayerNo_mid) Y1 = Y1/Ncablet;
        }
//***********************************************************************************
//No of cable per Layer determination at end section
        Cable_Layer_end[1] = (int)(BFRw-2*Ancg_Edge_dist_Lay1)/(Anchor_dim+30)+1;
        Layer_dist_bottom_end[1] = cable_1st_position_end;

        LayerNo_end = 1;
        N_total = 0;
        for( ; ; )
        {
            N_total = N_total+Cable_Layer_end[LayerNo_end];
            if(N_total<Ncablet)
            {
                LayerNo_end = LayerNo_end+1;
                if(Cable_Layer_end[1]>1)
                    Layer_dist_bottom_end[LayerNo_end] =
Layer_dist_bottom_end[LayerNo_end-1]+ Ancg_C2C_Lay1;
                else
                    Layer_dist_bottom_end[LayerNo_end] =
Layer_dist_bottom_end[LayerNo_end-1]+ Ancg_C2C;
                Cable_Loc_xw[LayerNo_end] =
Layer_dist_bottom_mid[1]+4*(Layer_dist_bottom_end[LayerNo_end]-
Layer_dist_bottom_mid[1])*pow((0.4L-xw),2)/pow(0.8L,2);
                if(Cable_Loc_xw[LayerNo_end]<= (BFRd + BFHd))
                {
                    Cable_Layer_end[LayerNo_end] = Cable_Layer_end[LayerNo_end-1];
                }
```

```
                    else
                    {
                              Cable_Layer_end[LayerNo_end] = 1;
                              Layer_dist_bottom_end[LayerNo_end] =
Layer_dist_bottom_end[LayerNo_end-1]+ Ancg_C2C;
                    }
          }
          else
          {
                    Cable_Layer_end[LayerNo_end] = int(Ncablet-(N_total -
Cable_Layer_end[LayerNo_end]));
                    break;
          }
     }
//   Cg of cables at end section from bottom,
     Y_end = 0;
     for(i = 1;i<=LayerNo_end;i++)
     {
          Y_end = Y_end + Cable_Layer_end[i] * Layer_dist_bottom_end[i];
          if(i == LayerNo_end) Y_end = Y_end/Ncablet;
     }
//*******************************************************************************
//No of cable per Layer determination at int_sup section
     double Layer_dist_bottom_int_sup[31];
     int Cable_Layer_int_sup[31],LayerNo_int_sup;

     Cable_Layer_int_sup[1] = 1;
     Layer_dist_bottom_int_sup[1] = Gd-2*(Duct_clear_spacing+Duct_dia)-(Ncablet-
1)*(Duct_clear_spacing+Duct_dia)-Duct_dia/2;

     LayerNo_int_sup = 1;
     N_total = 0;
     for( ; ; )
     {
          N_total = N_total+Cable_Layer_int_sup[LayerNo_int_sup];
          if(N_total<Ncablet)
          {
                    LayerNo_int_sup = LayerNo_int_sup+1;
                    Cable_Layer_int_sup[LayerNo_int_sup]=1;

     Layer_dist_bottom_int_sup[LayerNo_int_sup]=Layer_dist_bottom_int_sup[LayerNo_int_sup-
1]+Duct_clear_spacing+Duct_dia;

          }
          else
          {
                    break;
          }
```

```
	}
//	Cg of cables at int sup section from bottom,
	Y_int_sup = 0;
	for(i = 1;i<=LayerNo_int_sup;i++)
	{
		Y_int_sup = Y_int_sup + Cable_Layer_int_sup[i] * Layer_dist_bottom_int_sup[i];
		if(i == LayerNo_int_sup) Y_int_sup = Y_int_sup/Ncablet;
	}
//*****************************************************************************
//No of cable per Layer determination at inf section
double Layer_dist_bottom_inf[31];
	int Cable_Layer_inf[31],LayerNo_inf;

	Cable_Layer_inf[1] = 1;
	Layer_dist_bottom_inf[1] = Gd-4*(Duct_clear_spacing+Duct_dia)-(Ncablet-
1)*(Duct_clear_spacing+Duct_dia)-Duct_dia/2;

	LayerNo_inf = 1;
	N_total = 0;
	for( ; ; )
	{
		N_total = N_total+Cable_Layer_inf[LayerNo_inf];
		if(N_total<Ncablet)
		{
			LayerNo_inf = LayerNo_inf+1;
			Cable_Layer_inf[LayerNo_inf]=1;
			Layer_dist_bottom_inf[LayerNo_inf]=Layer_dist_bottom_inf[LayerNo_inf-
1]+Duct_clear_spacing+Duct_dia;

		}
		else
		{
			break;
		}
	}
//	Cg of cables at int sup section from bottom,
	Y_inf = 0;
	for(i = 1;i<=LayerNo_inf;i++)
	{
		Y_inf = Y_inf + Cable_Layer_inf[i] * Layer_dist_bottom_inf[i];
		if(i == LayerNo_inf) Y_inf = Y_inf/Ncablet;
	}
//*****************************************************************************
// Location of individual cable
	k1 = 0;
	for(i = 1;i<=LayerNo_mid;i++)
	{
		for(int j = 1;j<=Cable_Layer_mid[i];j++)
```

```
			{
				k1 = k1 + 1;
				Cable_Loc_mid[k1] = Layer_dist_bottom_mid[i];
			}
		}

		k1 = 0;
		for(i = 1;i<=LayerNo_end;i++)
		{
			for(int j = 1;j<=Cable_Layer_end[i];j++)
			{
				k1 = k1 + 1;
				Cable_Loc_end[k1] = Layer_dist_bottom_end[i];
			}
		}

		k1 = 0;
		for(i = 1;i<=LayerNo_int_sup;i++)
		{
			for(int j = 1;j<=Cable_Layer_int_sup[i];j++)
			{
				k1 = k1 + 1;
				Cable_Loc_int_sup[k1] = Layer_dist_bottom_int_sup[i];
			}
		}

		k1 = 0;
		for(i = 1;i<=LayerNo_inf;i++)
		{
			for(int j = 1;j<=Cable_Layer_inf[i];j++)
			{
				k1 = k1 + 1;
				Cable_Loc_inf[k1] = Layer_dist_bottom_inf[i];
			}
		}
		k1 = 0;
		for(i = 1;i<=LayerNo_mid;i++)
		{
			for(int j = 1;j<=Cable_Layer_mid[i];j++)
			{
				k1 = k1 + 1;
				Cable_Loc_xw[k1] = Layer_dist_bottom_mid[i]+4*(Cable_Loc_end[k1]-
Cable_Loc_mid[k1])*pow((0.4L-xw),2)/pow(0.8L,2);
				alpha_xw[k1] = 4*(Cable_Loc_xw[k1]-Cable_Loc_mid[k1])/(0.8L);
			}
		}
//*********************************************************************************
 //	Cg of steel at section xw-xw for shear
```

```
	Y2 = 0;
	for( i = 1;i<=k1;i++)
	{
		Y2 = Y2 + Cable_Loc_xw[i];
		if(i==k1)
		{
			Y2 = Y2/Ncablet;
		}
	}
}
//*******************************************************************************
//14__14__14__14__14____________Flexural Strength Determination Zone____________14__14__14
//*******************************************************************************
double Flexural_Strength(double As,double &Wri)
{
//	cout<<"Flexural_Strength"<<"\t";
	double gamma,beff,deff,Pp,fsu,Mu,Asw,z;
//	Flexural Strength
	gamma  = 0.28;
	deff = Gdc-Y1;
	Pp = As/(EFW*deff);
	fsu = fpu*(1-gamma/0.85*Pp*fpu/fcdeck);
	Wri = Pp*fsu/fcdeck;
	z = As*fsu/(0.85*fcdeck*EFW);
	if(z <=ts)
		Mu = 0.95*As*fsu*(deff-z/2)/1000;

	else if(As*fsu>0.85*fcdeck*EFW*ts)
	{
		beff = (fcdeck/fc*EFW*ts + TFRw*TFRd)/(ts + TFRd);
		Pp = As/(beff*deff);
		fsu = fpu*(1-gamma/0.75*Pp*fpu/fc);
		Wri = Pp*fsu/fc;
		z = As*fsu/(0.85*fc*beff);
		if(z<=(TFRd+ts))
			Mu = 0.95*As*fsu*(deff-z/2)/1000;

		else
		{
			beff = (fcdeck/fc*EFW*ts + TFRw*TFRd+(TFFHtw+TFFHbw)/2*TFFHd)/(ts +
TFRd+TFFHd);
			Pp = As/(beff*deff);
			fsu = fpu*(1-gamma/0.75*Pp*fpu/fc);
			Wri = Pp*fsu/fc;
			z = (As*fsu)/(0.85*fc*beff);
			if(z<=(ts + TFRd+TFFHd))
				Mu = 0.95*As*fsu*(deff-z/2)/1000;
```

```
                    else
                    {
                            Asw = As - 0.85*fc*(beff-Wt)*(ts + TFRd+TFFHd)/fsu;
                            Pp = Asw/(Wt*deff);
                            Wri = Pp*fsu/fc;
                            Mu = 0.95*(0.85*fc*(beff-Wt)*(ts + TFRd+TFFHd)*(deff-(ts +
TFRd+TFFHd)/2)
                                    + Asw*fsu*deff*(1-0.6*(Asw*fsu/Wt/deff/fc)))/1000;
                    }
            }
    }
    return Mu;
}
double Flexural_Strength_precastgirder(double &Wri2)
{
//  cout<<"Flexural_Strength_precastgirder"<<"\t";
    double gamma,beff,deff,Pp,fsu,Mu2,Asw,z;
//  Flexural Strength
    gamma  = 0.28;
    deff = Gd-Y1;
    Pp = pt1*As/(TFRw*deff);
    fsu = fpu*(1-gamma/0.75*Pp*fpu/fc);
    Wri2 = Pp*fsu/fc;
    z = pt1*As*fsu/(0.85*fc*TFRw);
    if(z <= TFRd)
            Mu2 = 0.95*pt1*As*fsu*(deff-z/2)/1000;

    else if(pt1*As*fsu>0.85*fc*TFRw*TFRd)
    {
            beff = (TFRw*TFRd+(TFFHtw+TFFHbw)/2*TFFHd)/(TFRd+TFFHd);
            Pp = pt1*As/(beff*deff);
            fsu = fpu*(1-gamma/0.75*Pp*fpu/fc);
            Wri2 = Pp*fsu/fc;
            z = pt1*As*fsu/(0.85*fc*beff);
            if(z<=(TFRd+TFFHd))
                    Mu2 = 0.95*pt1*As*fsu*(deff-z/2)/1000;

            else
            {
                    Asw = pt1*As - 0.85*fc*(beff-Wt)*(TFRd+TFFHd)/fsu;
                    Pp = Asw/(Wt*deff);
                    Wri2 = Pp*fsu/fc;
                    Mu2 = 0.95*(0.85*fc*(beff-Wt)*(TFRd+TFFHd)*(deff-(TFRd+TFFHd)/2)
                                    + Asw*fsu*deff*(1-0.6*(Asw*fsu/Wt/deff/fc)))/1000;
            }
    }
    return Mu2;
}
```

```
//*******************************************************************************
//15__15__15__15__15____________Deflection Calculation Zone____________15__15__15__15__15_
//*******************************************************************************
void Deflection(double Ncablet,double MD1,double MG1,double MLL1)
{
//	cout<<"Deflection"<<"\t";
	//	Deflection

	deflectiont = 0.0;
	deflectione = 0.0;
	deflectionf = 0.0;

	for(int i = 1; i<= Ncablet;i++)
	{
		deflectiont = deflectiont+(13.0/136.0*F1i*pt1/Ncablet*(Cable_Loc_end[i] - 
Cable_Loc_mid[i])-F1i*pt1/Ncablet*(Cable_Loc_end[i] - Gd/2)/8)*L*L/Eci/I*1000;
	}

	for(i = 1;i<= Ncablet;i++)
	{
		deflectione = deflectione +(13.0/136.0*F1i/Ncablet*(Cable_Loc_end[i] - 
Cable_Loc_mid[i])-F1i/Ncablet*(Cable_Loc_end[i] - Gd/2)/8)*L*L/Ec/Itf*1000;
	}

	deflectionf = deflectione*2.2;

	deflectiont = 13.0/136.0*MG1*1000*L*L/Ec/Itf - deflectiont;
	deflectione = 13.0/136.0*(MG1*1.85+(MP1-MG1))*L*L/Ec/Itf*1000 - deflectione;
	deflectionf = 13.0/136.0*((2.4*MG1/Itf+3.0*((MP1-MG1)/Itf+MC1/Ic)+MLL1/Ic))*1000*L*L/Ec - 
deflectionf;
//	deflection =  13.0/136.0*MLL1*1000*L*L/Ec/Ic;
	deflection =  324.0*pow(25.4,4)/(Ec*0.145*Ic)*24*DF*(1+IMF)*(pow(L/1000*3.28,3)-
555*L/1000*3.28+4780)*25.4/NoGirder;
//	deflection = 
22.5*pow(L/1000*3.28,3)*pow(25.4,4)/(Ec*0.145*Ic)*(1.6*9+0.32)*3*DF*(1+IMF)*25.4/NoGirder;
}

//*******************************************************************************
//16__16__16__16__16____________Prestress Loss Calculation Zone____________16__16__16__16__
//*******************************************************************************
void Prestress_Loss(double MG1,double MP1,double MD1)
{
//	cout<<"Prestress_Loss"<<"\t";
	double Lt,LWC,LAN,LES,Lo,LCR,LSR,LSH,Fof,Fmid;
	int Ncablet;
//	Loss Calculation
	As = Astrand*Nstrand*Ncable;
//	Jacking force,
```

```
        Fend=0.9*0.9*fpu*As/1000;       //!yield strength = 0.9 * ultimate  KN
        Ncablet = RNDOFF(&Ncable);                                  // !jacking force = 0.9*yield strength

//      Wobble and curvature loss,
        LWC = 0;
        for(int i = 1;i<=Ncablet;i++)
        {
                alpha[i] = (0.4L)*8*(Cable_Loc_end[i]-Cable_Loc_mid[i])/(L*L);
                LWC = LWC + Fend/Ncablet * (1-exp(-fricncoeff*alpha[i]-0.4*Kwc*L));
        }

        Fmid = Fend - LWC;
//      Anchorage Loss,
        x = sqrt(Delta*Es*L/2/((Fend-Fmid)*1000/As));
        LAN = 2*(Fend-Fmid)*x/(L/2);
        F3i = Fend - LAN/2;
        Fend = Fend - LAN;
        F2i = F3i-(x-xw)*LAN/2/x;

//      Elastic shortening loss,
        F1i = Fmid;

        for( ; ;)
        {
                LES = Kes*Es/Ec*(F1i/Atf+F1i*e1*e1/Itf-MG1*e1/Itf)*As;
                Fof = Fmid - LES;
                if (fabs((F1i-Fof)/F1i) <= 0.0001)
                        break;
                else
                        F1i = Fof;
        }

        for(int i = 1;i<=Ncablet;i++)
        {
                alpha[i] = (0.25L)*8*(Cable_Loc_inf[i]-Cable_Loc_mid[i])/(L*L);
                LWC = LWC + Fend/Ncablet * (1-exp(-fricncoeff*alpha[i]-0.25*Kwc*L));
        }

        F7i = F1i - LWC;

        for(int i = 1;i<=Ncablet;i++)
        {
                alpha[i] = (0.25L)*8*(Cable_Loc_inf[i]-Cable_Loc_mid[i])/(L*L);
                LWC = LWC + Fend/Ncablet * (1-exp(-fricncoeff*alpha[i]-0.25*Kwc*L));
        }

        F6i = F7i - LWC;
```

```
          for(int i = 1;i<=Ncablet;i++)
          {
                    alpha[i] = (0.05L)*8*(Cable_Loc_inf[i]-Cable_Loc_mid[i])/(L*L);
                    LWC = LWC + Fend/Ncablet * (1-exp(-fricncoeff*alpha[i]-0.05*Kwc*L));
          }

          F8i = F6i - LWC;

          for(int i = 1;i<=Ncablet;i++)
          {
                    alpha[i] = (0.05L)*8*(Cable_Loc_inf[i]-Cable_Loc_mid[i])/(L*L);
                    LWC = LWC + Fend/Ncablet * (1-exp(-fricncoeff*alpha[i]-0.05*Kwc*L));
          }

          F5i = F8i - LWC;

          F2i = F2i - LES;
          F3i = F3i - LES;
          Fend = Fend - LES;
          F5i = F5i - LES;
          F6i = F6i - LES;
          F7i = F7i - LES;
          F8i = F8i - LES;

//        Losses of prestress at transfer,
          Lo = LWC+LES+LAN;

// Time dependent Loss

//        Loss due to creep of concrete
          LCR = (12*(F1i/Atf+F1i*e1*e1/Itf-MG1*e1/Itf)-7*((MP1-MG1)*e1/Itf+MC1*ec1/Ic))*1000*145;
//!psi
          LSH = 0.8*(17000-150*77.916);   //!psi  RH = 77.916
          LSR = 5000-0.10*LES*1000/As*145-0.05*(LSH+ LCR);//     !psi
          if(LSR<0) LSR = 2185;   //!2.0%Loss of initial prestress considering

//        Time dependent Loss of prestress,
//        Lt= (LCR+LSH+LSR)/145.0*As/1000;
          Lt= (LCR+LSH+LSR)/145.0*As/1000*(1-pt1/20.0);
//        Effective force,

          F11= F1i-Lt;
          F21 = F2i - Lt;
          F31 = F3i - Lt;
          Fend2 = Fend - Lt;
          F51 = F5i - Lt;
          F61 = F6i - Lt;
          F71 = F7i - Lt;
```

```
		F81 = F8i - Lt;
}
/* void Moment3()
{
		MG3 = w_stage_03/2.0*(L*x-x*x)*1.0e-9;
		MCG3=(GS*Gd-BFRd*(GS-BFRw)-Ag)*Gammacon*CGt*(NCG/L)/2*(L*x-x*x)*1.0e-9;
		MS3 = (ts+12.5)*GS*Gammacon/2*(L*x-x*x)*1.0e-9;
		MP3=MG3+MCG3+MS3;
		MWC3 =WCt*Gammawc*GS/2*(L*x-x*x)*1.0e-9;
		MMS3=MSh*MSw*Gammacon/2*(L*x-x*x)/(NoGirder)*1.0e-9;
		MC3 = MWC3 + MMS3;
		MD3=MP3+MC3;
		MLL3 = maxm((4*P2*((L-x)/L + (L-x-4.27*1000)/L + (L-x-
8.54*1000)/L/4))*DF*x,(0.5*L*9.34/1000+80.064)*x*(L-x)/L*DF/2)*(1+IMF);
		MT3 = MLL3+MD3;

}  */
void cablelayout3()
{
//		cout<<"cablelayout3"<<"\t";
		int k1,   Ncablet;
		k1 = 0;
		for(int i = 1;i<=LayerNo_mid;i++)
		{
			for(int j = 1;j<=Cable_Layer_mid[i];j++)
			{
				k1 = k1 + 1;
				Cable_Loc_3[k1] = Layer_dist_bottom_mid[i]+4*(Cable_Loc_end[k1]-
Cable_Loc_mid[k1])*pow((0.4L-x),2)/pow(0.8L,2);
				alpha3[k1] = 4*(Cable_Loc_3[k1]-Cable_Loc_mid[k1])/(0.8L);
			}
		}
		Ncablet = RNDOFF(&Ncable);

		Y3 = 0;
		for( i = 1;i<=k1;i++)
		{
			Y3 = Y3 + Cable_Loc_3[i];
			if(i==k1)
			{
				Y3 = Y3/Ncablet;
			}
		}

		Y3bnet = Yb - Ncable*3.1416/4*pow((Duct_dia),2)*Y3;
		Y3b = Yb + (Es/Ec-1) * As* Y3;
		Y3b = Y3b/Atf;
		Y3t = Gd-Y3b;
```

```
		Y3bnet = Y3bnet/Anet;
		Y3tnet = Gd-Y3bnet;
		Y3bc = (Y3b*Atf+(mratio*EFW*ts)*(Gdc-ts/2))/Atfc;
		Y3tc = Gdc-Y3bc;
		e3i = Y3bnet - Y3;
		e3 = Y3b - Y3;
		ec3 = Y3bc - Y3;
		S3tnet = Inet/Y3tnet;
		S3bnet = Inet/Y3bnet;
		S3t = Itf/Y3t;
		S3b = Itf/Y3b;
		S3tc = Ic/(Y3tc - ts);
		S3bc = Ic/Y3bc;
}

void cablelayout7()
{
//	cout<<"cablelayout7"<<"\t";
	int k1,   Ncablet;
	k1 = 0;
	for(int i = 1;i<=LayerNo_mid;i++)
	{
		for(int j = 1;j<=Cable_Layer_mid[i];j++)
		{
			k1 = k1 + 1;
			Cable_Loc_7[k1] = Layer_dist_bottom_mid[i]+4*(Cable_Loc_inf[k1]-
Cable_Loc_mid[k1])*pow((L/2-L/4),2)/pow(L,2);
			alpha7[k1] = 4*(Cable_Loc_7[k1]-Cable_Loc_mid[k1])/L;
		}
	}
	Ncablet = RNDOFF(&Ncable);

	Y7 = 0;
	for( i = 1;i<=k1;i++)
	{
		Y7 = Y7 + Cable_Loc_7[i];
		if(i==k1)
		{
			Y7 = Y7/Ncablet;
		}
	}
}

void cablelayout8()
{
//	cout<<"cablelayout8"<<"\t";
	int k1,   Ncablet;
	k1 = 0;
```

```
	for(int i = 1;i<=LayerNo_inf;i++)
	{
		for(int j = 1;j<=Cable_Layer_inf[i];j++)
		{
			k1 = k1 + 1;
			Cable_Loc_8[k1] = Layer_dist_bottom_inf[i]+4*(Cable_Loc_int_sup[k1]-
Cable_Loc_inf[k1])*pow((L/10-L/20),2)/pow(L/5,2);
			alpha8[k1] = 4*(Cable_Loc_8[k1]-Cable_Loc_inf[k1])/(0.2L);
		}
	}
	Ncablet = RNDOFF(&Ncable);

	Y8 = 0;
	for( i = 1;i<=k1;i++)
	{
		Y8 = Y8 + Cable_Loc_8[i];
		if(i==k1)
		{
			Y8 = Y8/Ncablet;
		}
	}
}

//*****************************************************************************
//17__17__17__17__17____________Moment Calculation Zone____________17__17__17__17__
//*****************************************************************************

void Moment()
{
//	cout<<"Moment"<<"\t";
//		cout<<"Moment SWg"<<"\t";
//	Girder selfweight moment @ different postitions :

	UDL_SW_girder= Ag*Gammacon/1000000;   //N/mm

	matrix_initialization_function();
	local_stiffness_matrix_function();
	global_stiffness_matrix_function();
	global_member_force_matrix_function(UDL_SW_girder);
	impose_boundarycondition_function();
	dynamic_allocation_01();
   dynamic_allocation_02();
   DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                                   global_member_force_matrix_pointer,x);
	total_DL_end_shearforce_bendingmoment_function(UDL_SW_girder);
	total_DL_sectionwise_shearforce_function(UDL_SW_girder);
	total_DL_sectionwise_bendingmoment_function(UDL_SW_girder);
```

```
        MG1 = total_DL_sectionwise_bendingmoment_matrix[0][80]; //@ section 1
        MG2 = total_DL_sectionwise_bendingmoment_matrix[0][30]; //@ section 2
        MG3 = total_DL_sectionwise_bendingmoment_matrix[0][50];  //@ section 3
        MG4 = total_DL_sectionwise_bendingmoment_matrix[0][0];  //@ section 4
        MG5 = total_DL_sectionwise_bendingmoment_matrix[0][200]; //@ section 5
        MG6 = total_DL_sectionwise_bendingmoment_matrix[0][180]; //@ section 6
        MG7 = total_DL_sectionwise_bendingmoment_matrix[0][130]; //@ section 7
        MG8 = total_DL_sectionwise_bendingmoment_matrix[0][190]; //@ section 8
//--------------------------------------------------------------------------------
 // cout<<"Moment SWcg"<<"\t";

//      Cross Girder Moment @ different postitions :

        UDL_SW_CG= (GS*Gd-BFRd*(GS-BFRw)-Ag)*Gammacon*CGt*(NCG/L)/1000000;

        matrix_initialization_function();
        local_stiffness_matrix_function();
        global_stiffness_matrix_function();
        global_member_force_matrix_function(UDL_SW_CG);
        modified_global_stiffness_matrix_function();
        dynamic_allocation_01();
        modified_global_member_force_matrix_function();
  dynamic_allocation_02();
  DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                               global_member_force_matrix_pointer,x);
        total_DL_end_shearforce_bendingmoment_function(UDL_SW_CG);
        total_DL_sectionwise_shearforce_function(UDL_SW_CG);
        total_DL_sectionwise_bendingmoment_function(UDL_SW_CG);

        MCG1 = total_DL_sectionwise_bendingmoment_matrix[0][80]; //@ section 1
        MCG2 = total_DL_sectionwise_bendingmoment_matrix[0][30]; //@ section 2
        MCG3 = total_DL_sectionwise_bendingmoment_matrix[0][50];  //@ section 3
        MCG4 = total_DL_sectionwise_bendingmoment_matrix[0][0];  //@ section 4
        MCG5 = total_DL_sectionwise_bendingmoment_matrix[0][200]; //@ section 5
        MCG6 = total_DL_sectionwise_bendingmoment_matrix[0][180]; //@ section 6
        MCG7 = total_DL_sectionwise_bendingmoment_matrix[0][130]; //@ section 7
        MCG8 = total_DL_sectionwise_bendingmoment_matrix[0][190]; //@ section 8
//----------------------------------------------------------------------------------
//cout<<"Moment SWs"<<"\t";
//      Slab Moment @ different positions:

        UDL_SW_slab=(ts+12.5)*GS*Gammacon/1000000;

        matrix_initialization_function();
        local_stiffness_matrix_function();
        global_stiffness_matrix_function();
        global_member_force_matrix_function(UDL_SW_slab);
        modified_global_stiffness_matrix_function();
```

```
        dynamic_allocation_01();
        modified_global_member_force_matrix_function();
  dynamic_allocation_02();
  DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                               global_member_force_matrix_pointer,x);
        total_DL_end_shearforce_bendingmoment_function(UDL_SW_slab);
        total_DL_sectionwise_shearforce_function(UDL_SW_slab);
        total_DL_sectionwise_bendingmoment_function(UDL_SW_slab);

        MS1 = total_DL_sectionwise_bendingmoment_matrix[0][80]; //@ section 1
        MS2 = total_DL_sectionwise_bendingmoment_matrix[0][30]; //@ section 2
        MS3 = total_DL_sectionwise_bendingmoment_matrix[0][50];  //@ section 3
        MS4 = total_DL_sectionwise_bendingmoment_matrix[0][0];  //@ section 4
        MS5 = total_DL_sectionwise_bendingmoment_matrix[0][200]; //@ section 5
        MS6 = total_DL_sectionwise_bendingmoment_matrix[0][180]; //@ section 6
        MS7 = total_DL_sectionwise_bendingmoment_matrix[0][130]; //@ section 7
        MS8 = total_DL_sectionwise_bendingmoment_matrix[0][190]; //@ section 8
//----------------------------------------------------------------------------
//      Moment due to self weight, cross girder, deck slab @ different positions

        MP1=MG1+MCG1+MS1;               //@ section 1
        MP2=MG2+MCG2+MS2;               //@ section 2
        MP3=MG1+MCG1+MS1;               //@ section 3
        MP4=MG2+MCG2+MS2;               //@ section 4
        MP5=MG1+MCG1+MS1;               //@ section 5
        MP6=MG2+MCG2+MS2;               //@ section 6
        MP7=MG1+MCG1+MS1;               //@ section 7
        MP8=MG2+MCG2+MS2;               //@ section 8

//----------------------------------------------------------------------------
//cout<<"Moment SWwc"<<"\t";
//      Wearing course moment @ different positions

        UDL_SW_WC=WCt*Gammawc*GS/1000000;

        matrix_initialization_function();
        local_stiffness_matrix_function();
        global_stiffness_matrix_function();
        global_member_force_matrix_function(UDL_SW_WC);
        modified_global_stiffness_matrix_function();
        dynamic_allocation_01();
        modified_global_member_force_matrix_function();
  dynamic_allocation_02();
  DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                               global_member_force_matrix_pointer,x);
        total_DL_end_shearforce_bendingmoment_function(UDL_SW_WC);
        total_DL_sectionwise_shearforce_function(UDL_SW_WC);
        total_DL_sectionwise_bendingmoment_function(UDL_SW_WC);
```

```
        MWC1 = total_DL_sectionwise_bendingmoment_matrix[0][80]; //@ section 1
        MWC2 = total_DL_sectionwise_bendingmoment_matrix[0][30]; //@ section 2
        MWC3 = total_DL_sectionwise_bendingmoment_matrix[0][50];  //@ section 3
        MWC4 = total_DL_sectionwise_bendingmoment_matrix[0][0];  //@ section 4
        MWC5 = total_DL_sectionwise_bendingmoment_matrix[0][200]; //@ section 5
        MWC6 = total_DL_sectionwise_bendingmoment_matrix[0][180]; //@ section 6
        MWC7 = total_DL_sectionwise_bendingmoment_matrix[0][130]; //@ section 7
        MWC8 = total_DL_sectionwise_bendingmoment_matrix[0][190]; //@ section 8
//----------------------------------------------------------------------------------
//cout<<"Moment SWms"<<"\t";
//      Medain strip moment @ different positions

        UDL_SW_MS=MSh*MSw*Gammacon/(NoGirder)/1000000;

        matrix_initialization_function();
        local_stiffness_matrix_function();
        global_stiffness_matrix_function();
        global_member_force_matrix_function(UDL_SW_MS);
        modified_global_stiffness_matrix_function();
        dynamic_allocation_01();
        modified_global_member_force_matrix_function();
  dynamic_allocation_02();
  DOF_matrix_solution_function(global_stiffness_matrix_pointer,2*No_of_Node,
                              global_member_force_matrix_pointer,x);
        total_DL_end_shearforce_bendingmoment_function(UDL_SW_MS);
        total_DL_sectionwise_shearforce_function(UDL_SW_MS);
        total_DL_sectionwise_bendingmoment_function(UDL_SW_MS);

        MMS1 = total_DL_sectionwise_bendingmoment_matrix[0][80]; //@ section 1
        MMS2 = total_DL_sectionwise_bendingmoment_matrix[0][30]; //@ section 2
        MMS3 = total_DL_sectionwise_bendingmoment_matrix[0][50];  //@ section 3
        MMS4 = total_DL_sectionwise_bendingmoment_matrix[0][0];  //@ section 4
        MMS5 = total_DL_sectionwise_bendingmoment_matrix[0][200]; //@ section 5
        MMS6 = total_DL_sectionwise_bendingmoment_matrix[0][180]; //@ section 6
        MMS7 = total_DL_sectionwise_bendingmoment_matrix[0][130]; //@ section 7
        MMS8 = total_DL_sectionwise_bendingmoment_matrix[0][190]; //@ section 8
//----------------------------------------------------------------------------------
//      Composite dead load moment
        MC1 = MWC1 + MMS1;
        MC2 = MWC2 + MMS2;
        MC3 = MWC3 + MMS3;
        MC4 = MWC4 + MMS4;
        MC5 = MWC5 + MMS5;
        MC6 = MWC6 + MMS6;
        MC7 = MWC7 + MMS7;
        MC8 = MWC8 + MMS8;
//----------------------------------------------------------------------------------
```

```
//      Total dead load Moment
        MD1=MP1+MC1;
        MD2=MP2+MC2;
        MD3=MP3+MC3;
        MD4=MP4+MC4;
        MD5=MP5+MC5;
        MD6=MP6+MC6;
        MD7=MP7+MC7;
        MD8=MP8+MC8;
//------------------------------------------------------------------------------------
//      Live load moment @ different positions
   influence_line_function();
        MLL1 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(80),
         total_LL_atSpecific_section_negative_bendingmoment_function(80)); //@ section 1
        MLL2 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(30),
         total_LL_atSpecific_section_negative_bendingmoment_function(30)); //@ section 2
        MLL3 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(50),
         total_LL_atSpecific_section_negative_bendingmoment_function(50)); //@ section 3
        MLL4 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(0),
         total_LL_atSpecific_section_negative_bendingmoment_function(0)); //@ section 4
        MLL5 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(200),
         total_LL_atSpecific_section_negative_bendingmoment_function(200)); //@ section 5
        MLL6 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(180),
         total_LL_atSpecific_section_negative_bendingmoment_function(180)); //@ section 6
        MLL7 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(130),
         total_LL_atSpecific_section_negative_bendingmoment_function(130)); //@ section 7
        MLL8 = maxm(total_LL_atSpecific_section_positive_bendingmoment_function(190),
         total_LL_atSpecific_section_negative_bendingmoment_function(190)); //@ section 8
//-----------------------------------------------------------------------------------
//      Total Moment.
        MT1 = MLL1+MD1;
        MT2 = MLL2+MD2;
        MT3 = MLL3+MD3;
        MT4 = MLL4+MD4;
        MT5 = MLL5+MD5;
        MT6 = MLL6+MD6;
        MT7 = MLL7+MD7;
        MT8 = MLL8+MD8;
}

void momentslab()
{
//      cout<<"momentslab"<<"\t";
        IMFS= minm(50/((GS-TFRw/2)/1000*3.28+125),0.3);
        MSS = (ts+12.5)*Gammacon*pow((GS-TFRw/2),2)/10*1.0e-6;
        MSWC =WCt*Gammawc*pow((GS-TFRw/2),2)/10*1.0e-6;
        MSDL = MSS + MSWC;
        if(NoGirder >= 2.98)
```

```
            MSLL = ((GS-TFRw/2)/1000*3.28+2)/32.0*16.0*4451*0.8;
        else
            MSLL = ((GS-TFRw/2)/1000*3.28+2)/32.0*16.0*4451;

        Muslab = 1.3*(MSDL + 1.67*MSLL*(1+IMFS));
        dreq = sqrt(Muslab/(0.9*410*rho*(1-0.59*rho*fcdeck/410)));
        d_min = sqrt(Muslab/(0.9*410*0.0195*(1-0.59*0.0195*fcdeck/410)));
        ds = ts - 57;
//      R = Muslab/(0.9*ds*ds); //ref pci chap 8.2.3
//      Asnp = 0.85*fcdeck/410*(1-sqrt(1-(2/0.85/fcdeck)*R))*ds;
}
//*****************************************************************************
//18__18__18__18__18____________Shear Calculation Zone____________18__18__18__18__
//*****************************************************************************
void Shear()
{
//      cout<<"shear"<<"\t";
        double Vi,fd,Mcr,Mmax,Vci,Vp,Vcw,d_xw,fpe_xw,fpc,Ncablet,V2s;
        double V3i,fd3,Mcr3,Mmax3,V3ci,V3cw,d3,fpe3,fpc3,V3s,IMF3,VDL3,VLL3,V3c,V3u;
        IMF2 = minm( 50/((L-xw)/1000*3.28+125),0.3);
        VDL2 = 4*MD1/L - 8*MD1/(L*L)*xw;
        VLL2 = maxm((4*P2*((L-xw)/L + (L-xw-4.27*1000)/L + (L-xw-8.54*1000)/L/4))*DF,
            (0.5*9.34*(L-xw)/1000+115.65)*DF/2)*(1+IMF2);
//      Evaluation of Vci
        Vi =1.3*(VDL2+1.67*VLL2)-VDL2;
        d_xw = Gdc - Y2;
        e2 = Y2b - Y2;
        ec2 = Y2bc - Y2;
        Vnh = 350*(TFRw*d_xw)/(25.4*25.4)/1000*4.45;
        if(d_xw< 0.8*Gdc)
            d_xw = 0.8*Gdc;         // !As per AASHTO 2007

        fpe_xw = (F21/Atf+F21*e2/S2b)*1000;
        fd = (MP2/S2b+MC2/S2bc)*1000;
        Mcr =S2bc*(0.5* sqrt(fc) + fpe_xw - fd)/1000;
        Mmax = 1.3*(MD2+1.67*MLL2)-MD2;
        Vci = 0.05*sqrt(fc)*Wt*d_xw/1000 + VDL2 + Vi*Mcr/Mmax;
        if(Vci < 0.141*sqrt(fc)*Wt*d_xw/1000)
            Vci = 0.141*sqrt(fc)*Wt*d_xw/1000;
//*****************************************************************************
//      Evaluation of Vcw
        Ncablet = RNDOFF(&Ncable);

        Vp = 0;
        for(int i = 1;i<=Ncablet;i++)
        {
            Vp = Vp + F21/Ncablet*sin(alpha_xw[i]);
        }
```

```
        fpc = F21/Atf - F21*e2*(Y2bc-Y2b)/Itf + MP2*(Y2bc-Y2b)/Itf;
        Vcw = (0.283*sqrt(fc)/1000 + 0.3*fpc)*Wt*d_xw + Vp;
        dshear = d_xw;
        Vc = minm(Vci,Vcw);
        Vu =1.3*(VDL2+1.67*VLL2); //!KN
        V2s = maxm((Vu/0.9 - Vc),0.1);    //!phi = 0.9 for shear  unit = KN
//*******************************************************************//
        IMF3 = minm( 50/((L-x)/1000*3.28+125),0.3);
        VDL3 = 4*MD3/L - 8*MD3/(L*L)*x;
        VLL3 = maxm((4*P2*((L-x)/L + (L-x-4.27*1000)/L + (L-x-8.54*1000)/L/4))*DF,
                (0.5*9.34*(L-x)/1000+115.65)*DF/2)*(1+IMF2);
//      Evaluation of Vci
        V3i =1.3*(VDL3+1.67*VLL3)-VDL3;
        d3 = Gdc - Y3;
        e3 = Y3b - Y3;
        if(d3< 0.8*Gdc)
                d3 = 0.8*Gdc;     // !As per AASHTO 2007
        fpe3 = (F31/Atf+F31*e3/S3b)*1000;
        fd3 = (MP3/S3b+MC3/S3bc)*1000;
        Mcr3 =S3bc*(0.5* sqrt(fc) + fpe3 - fd3)/1000;
        Mmax3 = 1.3*(MD3+1.67*MLL3)-MD3;
        V3ci = 0.05*sqrt(fc)*Wt*d3/1000 + VDL3 + V3i*Mcr3/Mmax3;
        if(V3ci < 0.141*sqrt(fc)*Wt*d3/1000)
                V3ci = 0.141*sqrt(fc)*Wt*d3/1000;
//*****************************************************************************
//      Evaluation of Vcw
        Ncablet = RNDOFF(&Ncable);
        Vp = 0;
        for( i = 1;i<=Ncablet;i++)
        {
                Vp = Vp + F31/Ncablet*sin(alpha3[i]);
        }
        fpc3 = F31/Atf - F31*e3*(Y3bc-Y3b)/Itf + MP3*(Y3bc-Y3b)/Itf;
        V3cw = (0.283*sqrt(fc)/1000 + 0.3*fpc3)*Wt*d3 + Vp;
        V3c = minm(V3ci,V3cw);
        V3u =1.3*(VDL3+1.67*VLL3); //!KN
        V3s = maxm((V3u/0.9 - V3c),0.1);            //!phi = 0.9 for shear  unit = KN
        Vs = maxm(V2s,V3s);
        if(Vs == V2s)
                dshear = d_xw;
        else
                dshear = d3;
}
//*****************************************************************************
//19__19__19__19__19_____________EXPCON function  Zone_____________19__19__19__19__
//*****************************************************************************
void __stdcall EXPCON(int *IFLG,int *ISKP,int *KKT,int *KOUNT,double XMAX[nv],double
XMIN[nv],double XT[nv])
```

```
{
//	cout<<"EXPCON"<<"\t";
	double STRIP;
	int nx = 8,ny = 26,nt=69,nb=69,nd=101,na=10,ne=11,nf=36;
	Nstrand = XT[5];
	*KOUNT = *KOUNT+1;
	*KKT = *KKT+1;
	Anchorage_system();

	XMIN[0]=1499.99;
//	XMIN[0]=2399.99;
//	XMIN[0]=2999.99;

	XMIN[1]=300;
	XMIN[2]=300;
	XMIN[3]=Duct_dia + Duct_clear_spacing;
	XMIN[4]=1000.01;
	XMIN[5]=2.99;
	XMIN[6]=0.99;
   XMIN[7]=Ancg_Edge_dist_vertical;
	XMIN[8]=0.001;
	XMIN[9]=174.99;
	XMIN[10]=0.0015;
	XMIN[11]=75;
	XMIN[12]=50;
	XMIN[13]=Duct_dia + 80;

	XMAX[0]= 12000.01;
//	XMAX[0]= 2400.01;
//	XMAX[0]= 3000.01;
	XMAX[1]= 2000.01;
	XMAX[2]= 1150.01;
	XMAX[3]= 600.01;
	XMAX[4]= 3500.01;
	XMAX[5]= 19.001;
	XMAX[6]= 15.001;
	XMAX[7]= 1000.01;
	XMAX[8]= 0.999;
	XMAX[9]= 300.01;
	XMAX[10]=0.32*fcdeck/410.0;
	XMAX[11]= 300;
	XMAX[12]= 300;
	XMAX[13]= 300;

	if(*IFLG == 0)
	{
		STRIP=1e-4;
//		DX[0] = 2400;
```

```
//			DX[0] = 3000;
			DX[0]= 1500.0;
			DX[1]= 1715.0;
			DX[2]= 2000.0;
			DX[3]= 2400.0;
			DX[4]= 3000.0;
			DX[5]= 4000.0;
			DX[6]= 6000.0;
			DX[7]= 12000.0;

/*			DX[0]= 2000.0;
			DX[1]= 2300.0;
			DX[2]= 2650.0;
			DX[3]= 3200.0;
			DX[4]= 4000.0;
			DX[5]= 5330.0;
			DX[6]= 8000.0;
			DX[7]= 16000.0;*/

			DISCR2(DX,ISKP,&nx,&STRIP,&XT[0],&XMAX[0],&XMIN[0]);
			DISCR2(DX1,ISKP,&nt,&STRIP,&XT[1],&XMAX[1],&XMIN[1]);
			DISCR2(DX2,ISKP,&nb,&STRIP,&XT[2],&XMAX[2],&XMIN[2]);
			DISCR2(DX4,ISKP,&nd,&STRIP,&XT[4],&XMAX[4],&XMIN[4]);
	DINTG2(ISKP,&STRIP,&XT[5],&XMAX[5],&XMIN[5]);
			DINTG2(ISKP,&STRIP,&XT[6],&XMAX[6],&XMIN[6]);
			DISCR2(DECKT,ISKP,&ny,&STRIP,&XT[9],&XMAX[9],&XMIN[9]);
			DISCR2(DX11,ISKP,&na,&STRIP,&XT[11],&XMAX[11],&XMIN[11]);
			DISCR2(DX12,ISKP,&ne,&STRIP,&XT[12],&XMAX[12],&XMIN[12]);
			DISCR2(DX13,ISKP,&nf,&STRIP,&XT[13],&XMAX[13],&XMIN[13]);
		}
}
//*****************************************************************************
//20__20__20__20__20____________stdcall FUNC function Zone____________20__20__20__20__
//*****************************************************************************
void __stdcall FUNC(double *F,int *KOUNT,int *KUT,int *N,double XT[nv])
{
//		cout<<"FUNC"<<"\t";

		double
Av,smax,s,shearbar_length,shearbar_no,Ncablet,Wtst,Volcon,Wtnonprestd,Wtnonprestg;

		GS = XT[0];
		TFRw = XT[1];
		BFRw = XT[2];
		BFRd = XT[3];
		Gd  = XT[4];
		Nstrand = XT[5];
		Ncable = XT[6];
```

```
        cable_1st_position_end = XT[7];
        pt1 = XT[8];
        ts = XT[9];
        rho = XT[10];
        TFRd = XT[11];
        TFFHd = XT[12];
        Wt = XT[13];

//*******************************************************************************
        TFSHbw = Wt;
        TFSHtw = TFSHbw + 2*TFSHw;
        TFFHbw = TFSHtw;
        TFFHtw = TFRw;
        TFFHw = (TFFHtw - TFFHbw)/2;
        BFHw = (BFRw-Wt)/2;
        BFHd = BFHw/2;
        NoGirder = BW/GS;

        SA =
(TFRd+sqrt(pow(TFFHw,2)+pow(TFFHd,2))+TFSHd*1.414+Wd+sqrt(pow(BFHw,2)+pow(BFHd,2))+BFRd)*
L;
        Ncablet = RNDOFF(&Ncable);
        Anchorage_system();
        Wd = Gd - (TFRd+TFFHd+TFSHd+BFHd+BFRd);
        xw  =1.5*Gd;
        Cable_layout(cable_1st_position_end);
        Sectional_Properties();
        Comp_Sectional_Properties();
        Ag =
(TFRd*TFRw)+((TFFHtw+TFFHbw)/2*TFFHd)+((TFSHtw+TFSHbw)/2*TFSHd)+Wd*Wt+((Wt+BFRw)/2*BFH
d)+(BFRd*BFRw);
        Anet = Ag-Ncable*3.1416/4*(Duct_dia)*(Duct_dia);
        Volcon =Anet*(L-1.5*Gd)+(BFRw*Gd-Ncable*3.1416/4*(Duct_dia)*(Duct_dia))*Gd +
((Anet+BFRw*Gd)/2.0-Ncable*3.1416/4*(Duct_dia)*(Duct_dia))*Gd/2.0 ;
        As = Astrand*Nstrand*Ncable;
        Moment();
        Prestress_Loss(MG1,MP1,MD1);
//      Moment3();
        cablelayout3();
        cablelayout7();
        cablelayout8();
        Shear();
        Av = maxm(Vs*1000.0/(410.0*dshear),50*Wt/410*0.00683);
        if(Vs <= 0.333 * sqrt(fc)*Wt*dshear/1000)
                smax = minm(0.75*(Gd + ts + 12.5),610);
        else
                smax = minm(0.75*(Gd + ts + 12.5)/2.0,305);
        s = minm(226.0/Av,smax);
```

```
/*        AASHTO 8.20 - 12.7 mm @ 18'' c/c temperature reinforcement at top = As = 0.265 mm2/mm*/
//        shearbar_length = 113.0*(2.0*(Gd+ts)+2*(150+4.123*BFHd+(BFRd-40))+(BFRw-80)-60 +
2*(200+TFFHw+TFRd+ts-40+120-30));
          shearbar_length = 113.0*(2.0*(Gd+ts)+2*(150+4.123*BFHd+(BFRd-
40))+2*(200+TFFHw+TFRd+ts-40+120-30));
          if (s == 610)
                    shearbar_no = 2*((L/2-Gd)/3/s+(L/2-Gd)/3/s+(L/2-Gd)/3/s);
          else if(s <=510)
                    shearbar_no = 2*((L/2-Gd)/3/s+(L/2-Gd)/3/(s+50)+(L/2-Gd)/3/(s+100));
          else
                    shearbar_no = 2*((L/2-Gd)/3/s+(L/2-Gd)/3/(s+50)+(L/2-Gd)/3/(s+50));
          ds = ts - 57;
          Asnp = rho*ds;
          Asnpd = minm(220/sqrt((GS-TFRw/2)/1000*3.28),67)/100*Asnp;
          Wtnonprestd = (2*Asnp*GS*L + Asnpd*L*GS+ 0.265*L*GS)*Gammast;
          Wtnonprestg = shearbar_length*shearbar_no*Gammast;
          Wtst = Astrand*Nstrand*Ncable*L*Gammast;

          Cpcon = Volcon*UPcon+UPgf*2*SA;
          Cdconc = GS*ts*L*UPcondeck+UPdf*(GS-TFRw)*L;
          Cpst = Wtst*UPst+Anchcost*2*Ncable+sheathcost*(Duct_dia/50.0)*Ncable*(L/1000);
          Cnpst = Wtnonprestd*UPnonprest + Wtnonprestg*UPnonprest;
//        Cnpst = Wtnonprestd*UPnonprest;

   *KOUNT=*KOUNT+1;
   *KUT=*KUT+1;
   *F=(Cpcon+Cpst+Cdconc+Cnpst)*NoGirder;
}
//*********************************************************************************
//21__21__21__21__21_____________IMPCON function  Zone_____________21__21__21__21__
//*********************************************************************************
void __stdcall IMPCON(int *KOUNT,int *M,double XT[nv],double XX[icn],double XXMAX[icn],double
XXMIN[icn])
{
//        cout<<"IMPCON"<<"\t";
          double Mfactored,Ncablet;

          *KOUNT = *KOUNT + 1;
          *M = *M + 1;

          GS = XT[0];
          TFRw = XT[1];
          BFRw = XT[2];
          BFRd = XT[3];
          Gd  = XT[4];
          Nstrand = XT[5];
          Ncable = XT[6];
          cable_1st_position_end = XT[7];
```

```
        pt1 = XT[8];
        ts = XT[9];
        rho = XT[10];
        TFRd = XT[11];
        TFFHd = XT[12];
        Wt = XT[13];
//****************************************************************************
        TFSHbw = Wt;
        TFSHtw = TFSHbw + 2*TFSHw;
        TFFHbw = TFSHtw;
        TFFHtw = TFRw;
        TFFHw = (TFFHtw - TFFHbw)/2;
        BFHw = (BFRw-Wt)/2;
        BFHd = BFHw/2;

        Wd = Gd - (TFRd+TFFHd+TFSHd+BFHd+BFRd);
        xw  =1.5*Gd;

        Ncablet = RNDOFF(&Ncable);

        As = Astrand*Nstrand*Ncable;
        Cable_layout(cable_1st_position_end);
        Sectional_Properties();
        Comp_Sectional_Properties();

        NoGirder = BW/GS;
//****************************************************************************
        Moment();
//      Factored Moment
        Mfactored = 1.3*(MD1+1.67*MLL1);
//****************************************************************************

        Prestress_Loss(MG1,MP1,MD1);
//      Moment3();
        cablelayout3();
        cablelayout7();
        cablelayout8();
//****************************************************************************
//      Flexural stress at transfer
//      Initial stress at top,
        fti=(-F1i*pt1/Anet+F1i*pt1*e1i/S1tnet-MG1/S1tnet)*1000;
        fti_xw=(-F2i*pt1/Anet+F2i*pt1*e2i/S2tnet-MG2/S2tnet)*1000;
        f3ti=(-F3i*pt1/Anet+F3i*pt1*e3i/S3tnet-MG3/S3tnet)*1000;
//      f4ti=(-Fend*pt1/Anet+Fend*pt1*e4i/S4tnet-MG4/S4tnet)*1000;
        f5ti=(-F5i*pt1/Anet+F5i*pt1*e_int_sup_i/S_int_sup_tnet-MG5/S_int_sup_tnet)*1000;
        f6ti=(-F6i*pt1/Anet+F6i*pt1*e_inf_i/S_inf_tnet-MG6/S_inf_tnet)*1000;
        f7ti=(-F7i*pt1/Anet+F7i*pt1*e7i/S7tnet-MG7/S7tnet)*1000;
//      f8ti=(-F8i*pt1/Anet+F8i*pt1*e8i/S8tnet-MG8/S8tnet)*1000;
```

```
        XX[0]= fti;
        XXMAX[0]= 0.25* sqrt(fci);
        XXMIN[0]= -0.55*fci;

        XX[1]= fti_xw;
        XXMAX[1]= 0.25* sqrt(fci);
        XXMIN[1]= -0.55*fci;

        XX[2]= f3ti;
        XXMAX[2]= 0.25* sqrt(fci);
        XXMIN[2]= -0.55*fci;

        XX[3]= f5ti;
        XXMAX[3]= 0.25* sqrt(fci);
        XXMIN[3]= -0.55*fci;

        XX[4]= f6ti;
        XXMAX[4]= 0.25* sqrt(fci);
        XXMIN[4]= -0.55*fci;

        XX[5]= f7ti;
        XXMAX[5]= 0.25* sqrt(fci);
        XXMIN[5]= -0.55*fci;

//      Initial stress at bottom,
        fbi =-(F1i*pt1/Anet+F1i*pt1*e1i/S1bnet-MG1/S1bnet)*1000;
        fbi_xw =-(F2i*pt1/Anet+F2i*pt1*e2i/S2bnet-MG2/S2bnet)*1000;
        f3bi =-(F3i*pt1/Anet+F3i*pt1*e3i/S3bnet-MG3/S3bnet)*1000;
//      f4ti=-(Fend*pt1/Anet+Fend*pt1*e4i/S4tnet-MG4/S4tnet)*1000;
        f5bi=-(F5i*pt1/Anet+F5i*pt1*e_int_sup_i/S_int_sup_bnet-MG5/S_int_sup_bnet)*1000;
        f6bi=-(F6i*pt1/Anet+F6i*pt1*e_inf_i/S_inf_bnet-MG6/S_inf_bnet)*1000;
        f7bi=-(F7i*pt1/Anet+F7i*pt1*e7i/S7bnet-MG7/S7bnet)*1000;
        f8bi=-(F8i*pt1/Anet+F8i*pt1*e8i/S8bnet-MG8/S8bnet)*1000;

        XX[6]= fbi;
        XXMAX[6]= 0.25* sqrt(fci);
        XXMIN[6]= -0.55*fci;

        XX[7]= fbi_xw;
        XXMAX[7]= 0.25* sqrt(fci);
        XXMIN[7]= -0.55*fci;

        XX[8]= f3bi;
        XXMAX[8]= 0.25* sqrt(fci);
        XXMIN[8]= -0.55*fci;

        XX[9]= f5bi;
```

```
        XXMAX[9]= 0.25* sqrt(fci);
        XXMIN[9]= -0.55*fci;

        XX[10]= f6bi;
        XXMAX[10]= 0.25* sqrt(fci);
        XXMIN[10]= -0.55*fci;

        XX[11]= f7bi;
        XXMAX[11]= 0.25* sqrt(fci);
        XXMIN[11]= -0.55*fci;

//      Flexural stress at (Service II)Moment due to self weight, cross girder, deck slab,Wearing
course,Median strip
//      Stress at top fiber of girder,

        ftc= (-F11/Atf+F11*e1/S1t-MP1/S1t-MC1/S1tc)*1000;
        ftc_xw= (-F21/Atf+F21*e2/S2t- MP2/S2t-MC2/S2tc)*1000;
        f3tc= (-F31/Atf+F31*e3/S3t- MP3/S3t-MC3/S3tc)*1000;
//      f4tc= (-F41/Atf+F41*e4/S4t- MP4/S4t-MC4/S4tc)*1000;
        f5tc= (-F51/Atf+F51*e_int_sup/S_int_sup_t- MP5/S_int_sup_t-MC5/S_int_sup_tc)*1000;
        f6tc= (-F61/Atf+F61*e_inf/S_inf_t- MP6/S_inf_t-MC6/S_inf_tc)*1000;
        f7tc= (-F71/Atf+F71*e7/S7t- MP7/S7t-MC7/S7tc)*1000;
//      f8tc= (-F81/Atf+F81*e8/S8t- MP8/S8t-MC8/S8tc)*1000;

        XX[12]= ftc;
        XXMAX[12]= 0.5* sqrt(fc);
        XXMIN[12]= -0.40*fc;

        XX[13]= ftc_xw;
        XXMAX[13]= 0.5* sqrt(fc);
        XXMIN[13]= -0.40*fc;

        XX[14]= f3tc;
        XXMAX[14]= 0.5* sqrt(fc);
        XXMIN[14]= -0.40*fc;

        XX[15]= f5tc;
        XXMAX[15]= 0.5* sqrt(fc);
        XXMIN[15]= -0.40*fc;

        XX[16]= f6tc;
        XXMAX[16]= 0.5* sqrt(fc);
        XXMIN[16]= -0.40*fc;

        XX[17]= f7tc;
        XXMAX[17]= 0.5* sqrt(fc);
        XXMIN[17]= -0.40*fc;
```

```
//      Stress at bottom fiber,
        fbc = -(F11/Atf+F11*e1/S1b-MP1/S1b-MC1/S1bc)*1000;
        fbc_xw = -(F21/Atf+F21*e2/S2b-MP2/S2b-MC2/S2bc)*1000;
        f3bc = -(F31/Atf+F31*e3/S3b-MP3/S3b-MC3/S3bc)*1000;
//      f4bc = -(F41/Atf+F41*e4/S4b-MP4/S4b-MC4/S4bc)*1000;
        f5bc = -(F51/Atf+F51*e_int_sup/S_int_sup_b-MP5/S_int_sup_b-MC5/S_int_sup_bc)*1000;
        f6bc = -(F61/Atf+F61*e_inf/S_inf_b-MP6/S_inf_b-MC6/S_inf_bc)*1000;
        f7bc = -(F71/Atf+F71*e7/S7b-MP7/S7b-MC7/S7bc)*1000;
//      f8bc = -(F81/Atf+F81*e8/S8b-MP8/S8b-MC8/S8bc)*1000;

        XX[18]= fbc;
        XXMAX[18]= 0.5* sqrt(fc);
        XXMIN[18]=-0.40*fc;

        XX[19]= fbc_xw;
        XXMAX[19]= 0.5* sqrt(fc);
        XXMIN[19]=-0.40*fc;

        XX[20]= f3bc;
        XXMAX[20]= 0.5* sqrt(fc);
        XXMIN[20]=-0.40*fc;

        XX[21]= f5bc;
        XXMAX[21]= 0.5* sqrt(fc);
        XXMIN[21]=-0.40*fc;

        XX[22]= f6bc;
        XXMAX[22]= 0.5* sqrt(fc);
        XXMIN[22]=-0.40*fc;

        XX[23]= f7bc;
        XXMAX[23]= 0.5* sqrt(fc);
        XXMIN[23]=-0.40*fc;
//      Flexural stress at (Service III)Moment due to all dead Load + Live load
//      Stress at top fiber of girder,

        ftt= (-F11/Atf+F11*e1/S1t-MP1/S1t-(MC1+MLL1)/S1tc)*1000;
        ftt_xw= (-F21/Atf+F21*e2/S2t-MP2/S2t-(MC2+MLL2)/S2tc)*1000;
        f3tt= (-F31/Atf+F31*e3/S3t-MP3/S3t-(MC3+MLL3)/S3tc)*1000;
//      f4tt= (-F41/Atf+F41*e4/S4t-MP4/S4t-(MC4+MLL4)/S4tc)*1000;
        f5tt= (-F51/Atf+F51*e_int_sup/S_int_sup_t-MP5/S_int_sup_t-(MC5+MLL5)/S_int_sup_tc)*1000;
        f6tt= (-F61/Atf+F61*e_inf/S_inf_t-MP6/S_inf_t-(MC6+MLL6)/S_inf_tc)*1000;
        f7tt= (-F71/Atf+F71*e7/S7t-MP7/S7t-(MC7+MLL7)/S7tc)*1000;
//      f8tt= (-F81/Atf+F81*e8/S8t-MP8/S8t-(MC8+MLL8)/S8tc)*1000;

        XX[24]= ftt;
        XXMAX[24]=  0.5* sqrt(fc);
        XXMIN[24]=  -0.6*fc;
```

```
        XX[25]= ftt_xw;
        XXMAX[25]=  0.5* sqrt(fc);
        XXMIN[25]=  -0.6*fc;

        XX[26]= f3tt;
        XXMAX[26]=  0.5* sqrt(fc);
        XXMIN[26]=  -0.6*fc;

        XX[27]= f5tt;
        XXMAX[27]=  0.5* sqrt(fc);
        XXMIN[27]=  -0.6*fc;

        XX[28]= f6tt;
        XXMAX[28]=  0.5* sqrt(fc);
        XXMIN[28]=  -0.6*fc;

        XX[29]= f7tt;
        XXMAX[29]=  0.5* sqrt(fc);
        XXMIN[29]=  -0.6*fc;

//      Stress at bottom fiber,
        fbt = -(F11/Atf+F11*e1/S1b-MP1/S1b-(MC1+MLL1)/S1bc)*1000;
        fbt_xw = -(F21/Atf+F21*e2/S2b-MP2/S2b-(MC2+MLL2)/S2bc)*1000;
        f3bt = -(F31/Atf+F31*e3/S3b-MP3/S3b-(MC3+MLL3)/S3bc)*1000;
//      f4bt = -(F41/Atf+F41*e4/S4b-MP4/S4b-(MC4+MLL4)/S4bc)*1000;
        f5bt = -(F51/Atf+F51*e_int_sup/S_int_sup_b-MP5/S_int_sup_b-
(MC5+MLL5)/S_int_sup_bc)*1000;
        f6bt = -(F61/Atf+F61*e_inf/S_inf_b-MP6/S_inf_b-(MC6+MLL6)/S_inf_bc)*1000;
        f7bt = -(F71/Atf+F71*e7/S7b-MP7/S7b-(MC7+MLL7)/S7bc)*1000;
//      f8bt = -(F81/Atf+F81*e8/S8b-MP8/S8b-(MC8+MLL8)/S8bc)*1000;

        XX[30]= fbt;
        XXMAX[30]= 0.5* sqrt(fc);
        XXMIN[30]= -0.6*fc;

        XX[31]= fbt_xw;
        XXMAX[31]= 0.5* sqrt(fc);
        XXMIN[31]= -0.6*fc;

        XX[32]= f3bt;
        XXMAX[32]= 0.5* sqrt(fc);
        XXMIN[32]= -0.6*fc;

        XX[33]= f5bt;
        XXMAX[33]= 0.5* sqrt(fc);
        XXMIN[33]= -0.6*fc;
```

```
        XX[34]= f6bt;
        XXMAX[34]= 0.5* sqrt(fc);
        XXMIN[34]= -0.6*fc;

        XX[35]= f7bt;
        XXMAX[35]= 0.5* sqrt(fc);
        XXMIN[35]= -0.6*fc;

//      Flexural stress at (Service IIII)Moment due to 1/2( dead Load + PS) + Live load
//      Stress at top fiber of girder,

        fttt= (-(F11/2)/Atf+(F11/2)*e1/S1t-(MLL1+MD1/2)/S1tc)*1000;
        fttt_xw= (-(F21/2)/Atf+(F21/2)*e2/S2t-(MLL2+MD2/2)/S2tc)*1000;
        f3ttt= (-(F31/2)/Atf+(F31/2)*e3/S3t-(MLL3+MD3/2)/S3tc)*1000;
//      f4ttt= (-(F41/2)/Atf+(F41/2)*e4/S4t-(MLL4+MD4/2)/S4tc)*1000;
        f5ttt= (-(F51/2)/Atf+(F51/2)*e_int_sup/S_int_sup_t-(MLL5+MD5/2)/S_int_sup_tc)*1000;
        f6ttt= (-(F61/2)/Atf+(F61/2)*e_inf/S_inf_t-(MLL6+MD6/2)/S_inf_tc)*1000;
        f7ttt= (-(F71/2)/Atf+(F71/2)*e7/S7t-(MLL7+MD7/2)/S7tc)*1000;
//      f8ttt= (-(F81/2)/Atf+(F81/2)*e8/S8t-(MLL8+MD8/2)/S8tc)*1000;

        XX[36]= fttt;
        XXMAX[36]=  0.5*sqrt(fc);
        XXMIN[36]=  -0.40*fc;

        XX[37]= fttt_xw;
        XXMAX[37]=  0.5* sqrt(fc);
        XXMIN[37]=  -0.40*fc;

        XX[38]= f3ttt;
        XXMAX[38]=  0.5* sqrt(fc);
        XXMIN[38]=  -0.40*fc;

        XX[39]= f5ttt;
        XXMAX[39]=  0.5* sqrt(fc);
        XXMIN[39]=  -0.40*fc;

        XX[40]= f6ttt;
        XXMAX[40]=  0.5* sqrt(fc);
        XXMIN[40]=  -0.40*fc;

        XX[41]= f7ttt;
        XXMAX[41]=  0.5* sqrt(fc);
        XXMIN[41]=  -0.40*fc;

//      Stress at bottom fiber,
        fbtt = -((F11/2)/Atf+(F11/2)*e1/S1b-(MLL1+MD1/2)/S1bc)*1000;
        fbtt_xw = -((F21/2)/Atf+(F21/2)*e2/S2b-(MLL2+MD2/2)/S2bc)*1000;
        f3btt = -((F31/2)/Atf+(F31/2)*e3/S3b-(MLL3+MD3/2)/S3bc)*1000;
```

```
//      f4btt = -((F41/2)/Atf+(F41/2)*e4/S4b-(MLL4+MD4/2)/S4bc)*1000;
        f5btt = -((F51/2)/Atf+(F51/2)*e_int_sup/S_int_sup_b-(MLL5+MD5/2)/S_int_sup_bc)*1000;
        f6btt = -((F61/2)/Atf+(F61/2)*e_inf/S_inf_b-(MLL6+MD6/2)/S_inf_bc)*1000;
        f7btt = -((F71/2)/Atf+(F71/2)*e7/S7b-(MLL7+MD7/2)/S7bc)*1000;
//      f8btt = -((F81/2)/Atf+(F81/2)*e8/S8b-(MLL8+MD8/2)/S8bc)*1000;

        XX[42]= fbtt;
        XXMAX[42]= 1* sqrt(fc);
        XXMIN[42]= -0.4*fc;

        XX[43]= fbtt_xw;
        XXMAX[43]= 1* sqrt(fc);
        XXMIN[43]= -0.4*fc;

        XX[44]= f3btt;
        XXMAX[44]= 1* sqrt(fc);
        XXMIN[44]= -0.4*fc;

        XX[45]= f5btt;
        XXMAX[45]= 1* sqrt(fc);
        XXMIN[45]= -0.4*fc;

        XX[46]= f6btt;
        XXMAX[46]= 1* sqrt(fc);
        XXMIN[46]= -0.4*fc;

        XX[47]= f7btt;
        XXMAX[47]= 1* sqrt(fc);
        XXMIN[47]= -0.4*fc;

        Mu = Flexural_Strength(As,Wri);
        XX[48]= Mfactored;
        XXMAX[48]= Mu;
        XXMIN[48]= 0.0;

        Shear();
        XX[49]= Vs;
        XXMAX[49]= 0.666 * sqrt(fc)*Wt*dshear/1000;
        XXMIN[49]= 0.0;

//      Ductility Limit
        fpe = (F11/Atf+F11*e1/S1b)*1000;
        Mcr2 = S1bc*(0.625*sqrt(fc)+fpe)/1000-MP1*(S1bc/S1b-1);

        XX[50]= Mcr2;
        XXMAX[50]= Mu/1.2;
        XXMIN[50]= 0.0;
```

```
XX[51]= Wri;
XXMAX[51]= 0.36*0.75;
XXMIN[51]= 0.0;

XX[52]= Y_end;
XXMAX[52]= Gd/2+Gd/6+0.5* sqrt(fc)* (BFRw*Gd)*Gd/6/(Fend2*1000);
XXMIN[52]= Gd/2-Gd/6-0.25* sqrt(fci)* (BFRw*Gd)*Gd/6/(Fend*1000);

Deflection(Ncablet,MD1,MG1,MLL1);
XX[53]= fabs(deflectiont);
XXMAX[53]= L/360;
XXMIN[53]= 0.0;

XX[54]= fabs(deflectione);
XXMAX[54]= L/360;
XXMIN[54]= 0.0;

XX[55]= fabs(deflectionf);
XXMAX[55]= L/100;
XXMIN[55]= 0.0;

XX[56]= fabs(deflection);
XXMAX[56]= L/800;
XXMIN[56]= 0.0;

XX[57]= Fend;
XXMAX[57]= 0.70*fpu*As/1000;
XXMIN[57]= 0.0;

XX[58]= F3i;
XXMAX[58]= 0.747*fpu*As/1000;
XXMIN[58]= 0.0;

XX[59]= F31;
XXMAX[59]= 0.72*fpu*As/1000;
XXMIN[59]= 0.0;

XX[60]= Vnh;
XXMAX[60]= 100000;
XXMIN[60]= Vu/0.9;

XX[61]= ts;
XXMAX[61]= 300.0;
XXMIN[61]= ((GS-TFRw/2)/1000*3.28+17)/3*25.4;

momentslab();

XX[62]= dreq;
```

```
	XXMAX[62]= ds;
	XXMIN[62]= d_min;

	double Wri2;
	double  Mupregirder = Flexural_Strength_precastgirder(Wri2);
	XX[63]= 1.3*MG1;
	XXMAX[63]= Mupregirder;
	XXMIN[63]= 0.0;

	double Mcrslab = 0.625*sqrt(fcdeck)*ts*ts/6;

	XX[64]= Mcrslab;
	XXMAX[64]= Muslab/1.2;
	XXMIN[64]= 0.0;

	XX[65]= Wri2;
	XXMAX[65]= 0.36*0.75;
	XXMIN[65]= 0.0;

	double a1 = 0.1*L;
	double L1 = L-2*a1;

	double ei = (1.025*(pow(L1/L,2)-0.333)+0.25)*25.4;
	double yr = Y1t-deflectiont;
	Ig = pow(TFRw,3)*TFRd/12;
	Ig = Ig + TFFHd*pow(TFFHw,3)/36*2+(TFFHw*TFFHd)*pow((TFFHbw/2+TFFHw/3),2);
	Ig = Ig + (pow(TFFHbw,3)*TFFHd/12);
	Ig = Ig + (TFSHd*pow(TFSHw,3)/36)*2+(TFSHw*TFSHd)*pow((Wt/2+TFSHw/3),2);
	Ig = Ig + (TFSHd*pow(Wt,3)/12);
	Ig = Ig + (Wd*pow(Wt,3)/12);
	Ig = Ig + (BFHd*pow(BFHw,3)/36)*2+(BFHw*BFHd)*pow((Wt/2+BFHw/3),2);
	Ig = Ig + BFHd*pow(Wt,3)/12;
	Ig = Ig + (BFRd*pow(BFRw,3)/12);
	double zo = UDL_SW_slab/(12*Eci*Ig*L)*(0.1*pow(L1,5)-
pow(L1,3)*a1*a1+3*pow(a1,4)*L1+1.2*pow(a1,5))*1e-6;
//	double oi = maxm(ei/yr,0.0001);
	double oi = ei/yr;

	//double res1=(0.625*sqrt(fci)+(-fti));
  //res1=fabs(res1);

	double Mlat = fabs((0.625*sqrt(fci)+(-fti)))*Ig/(TFRw/2);
	double Mg1 = (UDL_SW_slab*L1*L1/8.0)*1e-6;
	double omax = maxm(Mlat/Mg1,0.00001);

//	double ab = maxm((zo/yr),0.0001);
	double ab = zo/yr;
//	double cd = maxm((oi/omax),0.0001);
```

```
        double cd = oi/omax;
        double Fsc = 1/(ab+cd);

        XX[66]= Fsc;
        XXMAX[66]= 100.0;
        XXMIN[66]= 1.5;

        double ft,ft_xw,f3t,fb,fb_xw,f3b;
        ft= (-F1i/Atf+F1i*e1/S1t- MP1/S1t)*1000;
        ft_xw= (-F2i/Atf+F2i*e2/S2t-MP2/S2t)*1000;
        f3t = (-F3i/Atf+F3i*e3/S3t-MP3/S3t)*1000;

        XX[67]= ft;
        XXMAX[67]= 0.5* sqrt(fc);
        XXMIN[67]= -0.60*fc;

        XX[68]= ft_xw;
        XXMAX[68]= 0.5* sqrt(fc);
        XXMIN[68]= -0.60*fc;

        XX[69]= f3t;
        XXMAX[69]= 0.5* sqrt(fc);
        XXMIN[69]= -0.60*fc;

//      Stress at bottom fiber,

        fb = -(F1i/Atf+F1i*e1/S1b-MP1/S1b)*1000;
        fb_xw = -(F2i/Atf+F2i*e2/S2b-MP2/S2b)*1000;
        f3b = -(F3i/Atf+F3i*e3/S3b-MP3/S3b)*1000;

        XX[70]= fb;
        XXMAX[70]= 0.5* sqrt(fc);
        XXMIN[70]= -0.60*fc;

        XX[71]= fb_xw;
        XXMAX[71]= 0.5* sqrt(fc);
        XXMIN[71]= -0.60*fc;

        XX[72]= fb;
        XXMAX[72]= 0.5* sqrt(fc);
        XXMIN[72]= -0.60*fc;
 }
//*****************************************************************************
//22__22__22__22__22____________MAIN function  Zone____________22__22__22__22__
//*****************************************************************************
  void main()
  {
//        cout<<"main"<<"\t";
```

```
	time_t start, stop;
	time(&start);

	double
C[nv],FF[nv+1],H[nv*(nv+1)],OLDCC[nv],XDN[nv],XG[nv],XMAX[nv],XMIN[nv],XUP[nv],XX[icn],XXMAX[icn
],XXMIN[icn],XT[nv];
	double ALPHA,BETA,DEL,GAMA,PHI,PHICPX;
	int ICON,IJK,IMV,IPRINT,K,KNT,LIMIT,N,NRSTRT,NIC;

	//	initial Value:

	GS = 1600;
	TFRw = 400;
	BFRw = 400;
	BFRd = 250;
	Gd = 1500;	//3lane 50mpa
	Nstrand = 5.0;
	Ncable = 4.0;
	cable_1st_position_end = 350.0;
	pt1 = 0.44;
	ts = 200.00;
	rho = 0.001773;
	TFRd = 160;
	TFFHd = 55;
	Wt =250;

	XT[0]= GS;
	XT[1]= TFRw;
	XT[2]= BFRw;
	XT[3]= BFRd;
	XT[4]= Gd;
	XT[5]= Nstrand;
	XT[6]= Ncable;
	XT[7]= cable_1st_position_end;
	XT[8]= pt1;
	XT[9]= ts;
	XT[10] = rho;
	XT[11] = TFRd;
	XT[12] = TFFHd;
	XT[13] = Wt;

       matrix_initialization_function();
//*********************************************************************************
	for(int i = 0;i<26;i++)
	{
		DECKT[i] = 175+5*i;
	}
```

```
 for( i = 0;i<69;i++)
 {
        DX1[i] = 300+25*i;
        DX2[i] = 300+25*i;
 }
  for( i = 0;i<101;i++)
 {
        DX4[i] = 1000+25*i;
 }
  for( i = 0;i<10;i++)
 {
        DX11[i] = 75+25*i;
 }
  for( i = 0;i<11;i++)
 {
        DX12[i] = 50+25*i;
 }
  for( i = 0;i<36;i++)
 {
        DX13[i] = 125+5*i;
 }

 cout<<"cost = ?"<<endl;
 cin>>cost;

if(cost == 1)
{
       UPcon=12500e-9;          //!per mm3
       UPcondeck=6000e-9;     // !per mm3
       UPst=90;// !per Kg
       UPnonprest = 45;//UPst/4.0;
       Anchcost = 4500;
       sheathcost = 90;
       UPgf = 400e-6;
       UPdf = 415e-6;
}
else if(cost == 2)
{
       UPcon=12500e-9;          //!per mm3
       UPcondeck=6000e-9;     // !per mm3
       UPst=180;// !per Kg
       UPnonprest = 90;//UPst/4.0;
       Anchcost = 9000;
       sheathcost = 180;
       UPgf = 400e-6;
       UPdf = 415e-6;
}
```

```
        else
        {
                UPcon=12500e-9;          //!per mm3
                UPcondeck=6000e-9;    // !per mm3
                UPst=270;// !per Kg
                UPnonprest = 135;//UPst/4.0;
                Anchcost = 13500;
                sheathcost = 270;
                UPgf = 400e-6;
                UPdf = 415e-6;
        }
//*************************************************************************************
//    CONTROL PARAMETERS FOR "EVOP"
         ALPHA = 1.2;
         BETA=0.5;
         GAMA=2.0;
         DEL=1e-12;
         PHI=1e-13;
         PHICPX=1e-8;
         ICON=5;
         LIMIT=100000;
         KNT=25;
         N=nv;
         NIC=icn;
         if(nv<=5)
         {
                 K=2*nv;
         }
         else
         {
                 K=nv+1;
         }
         IPRINT=2;
         NRSTRT=10;
         IMV=0;
         IJK=1;
line1:
EVOP(&ALPHA,&BETA,C,&DEL,FF,&GAMA,H,&ICON,&IJK,&IMV,&IPRINT,&K,&KNT,&LIMIT,&N,&NRSTRT,
                  &NIC,OLDCC,&PHI,&PHICPX,XDN,XG,XMAX,XMIN,XT,XUP,XX,XXMAX,XXMIN);
         if (IJK < 9) goto line1;
         time(&stop);
         cout<<difftime(stop, start)<<endl;

        cout<<"Cpcon"<<Cpcon/(GS*L*70)*1e6<<endl;
        cout<<"Cdconc"<<Cdconc/(GS*L*70)*1e6<<endl;
        cout<<"Cpst"<<Cpst/(GS*L*70)*1e6<<endl;
        cout<<"Cnpst"<<Cnpst/(GS*L*70)*1e6<<endl;
        cout<<UPgf*2*SA/(GS*L*70)*1e6<<endl;
```

```
        cout<<UPdf*(GS-TFRw)*L/(GS*L*70)*1e6<<endl;
        cout<<(Cpcon+Cpst+Cdconc+Cnpst)*NoGirder/(BW*L*70)*1e6<<endl;

        ofstream fout("output.txt");
        fout<<"Cpcon"<<Cpcon/(GS*L*70)*1e6<<endl;
        fout<<"Cdconc"<<Cdconc/(GS*L*70)*1e6<<endl;
        fout<<"Cpst"<<Cpst/(GS*L*70)*1e6<<endl;
        fout<<"Cnpst"<<Cnpst/(GS*L*70)*1e6<<endl;
        fout<<UPgf*2*SA/(GS*L*70)*1e6<<endl;
        fout<<UPdf*(GS-TFRw)*L/(GS*L*70)*1e6<<endl;
        fout<<(Cpcon+Cpst+Cdconc+Cnpst)*NoGirder/(BW*L*70)*1e6<<endl;
        fout<<Y1<<endl;
        fout<<Y2<<endl;
        fout<<Y3<<endl;
        fout<<Y_end<<endl;
        fout<<Ancg_C2C<<endl;

test_function();

        fout.close();
}
```

APPENDIX-B

Output Summary of Optimization Results of the 40 m double span continuous girder from EVOP Program

INPUT PARAMETERS FOR OPTIMISATION SUBROUTINE EVOP

REFLECTION COEFFICIENT ALPHA =.13000000E+01

CONTRACTION COEFFICIENT BETA = .50000000E+00

EXPANSION COEFFICIENT GAMA = .20000000E+01

EXPLICIT CONSTRAINT RETENTION COEFFICIENT

DEL =.10000000E-11

ACCURACY PARAMETER FOR CONVERGENCE PHI = .10000000E-12

PARAMETER FOR DETERMINING COLLAPSE OF A COMPLEX IN A SUBSPACE PHICPX = .10000000E-15

GLOBAL LIMIT ON THE NUMBER OF CALLS TO FUNCTION SUBROUTINE

LIMIT = 100000

NUMBER OF COMPLEX RESTARTS NRSTRT = 10

NUMBER OF CALLS TO FUNCTION SUBROUTINE AFTER WHICH CONVERGENCE TESTS ARE MADE KNT = 25

NUMBER OF CONSECUTIVE CONVERGENCE TEST_1 ICON = 5

NUMBER OF VARIABLES = NUMBER OF EXPLICIT CONSTRAINTS N =14

NUMBER OF IMPLICIT CONSTRAINTS NIC = 73

NUMBER OF COMPLEX VERTICES K = 15

COORDINATES OF THE STARTING POINT

Serial No.	Design variables	Design variables
1	S = 3500;	XT(1) = .35000000E+04
2	TF_w = 650;	XT(2) = .65000000E+03
3	BF_w = 450;	XT(3) = .45000000E+03
4	BF_t = 325;	XT(4) = .32500000E+03
5	G_d = 2500;	XT(5) = .25000000E+04
6	N_S = 9.0;	XT(6) = .90000000E+01

7	$N_T = 7.0;$	XT(7) = .70000000E+01
8	$y_1 = 645;$	XT(8) = .64500000E+03
9	$\eta = 0.55;$	XT(9) = .55000000E+00
10	$t = 250;$	XT(10) = .25000000E+03
11	$\rho = 0.005373;$	XT(11) = .53730000E-02
12	$TF_t = 125;$	XT(12) = .12500000E+03
13	$TFT_t = 75;$	XT(13) = .75000000E+02
14	$W_w = 190;$	XT(14) = .19000000E+03

FUNCTION VALUE AT THE STARTING POINT **FF(1)** = **.54087839E+07**

UPPER BOUND OF EXPLICIT CONSTRAINTS AT THE STARTING POINT	LOWER BOUND OF EXPLICIT CONSTRAINTS AT THE STARTING POINT
XMAX(1) = .12000010E+05	XMIN(1) = .14999900E+04
XMAX(2) = .20000100E+04	XMIN(2) = .30000000E+03
XMAX(3) = .11500100E+04	XMIN(3) = .30000000E+03
XMAX(4) = .60001000E+03	XMIN(4) = .10800000E+03
XMAX(5) = .35000100E+04	XMIN(5) = .99999000E+03
XMAX(6) = .19001000E+02	XMIN(6) = .29900000E+01
XMAX(7) = .15001000E+02	XMIN(7) = .99000000E+00
XMAX(8) = .10000100E+04	XMIN(8) = .21000000E+03
XMAX(9) = .99900000E+00	XMIN(9) = .10000000E-02
XMAX(10) = .30001000E+03	XMIN(10) = .17499000E+03
XMAX(11) = .19512195E-01	XMIN(11) = .15000000E-02
XMAX(12) = .30001000E+03	XMIN(12) = .74990000E+02
XMAX(13) = .30001000E+03	XMIN(13) = .49990000E+02
XMAX(14) = .30001000E+03	XMIN(14) = .14999000E+03

IMPLICIT CONSTRAINTS

Implicit Constraints	**Description**
XX(1) = f1ti XX(2) = f2ti XX(3) = f3ti XX(4) = f5ti XX(5) = f6ti XX(6) = f7ti	f1ti, f2ti, f3ti, f5ti, f6ti, f7ti are top fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at initial stage $$f_{ti} = -\frac{\eta F_i}{A_{net}} \pm \frac{\eta F_i e_i}{S_{tnet}} \mp \frac{M_G}{S_{tnet}}$$
XX(7) = f1bi XX(8) = f2bi XX(9) = f3bi XX(10) = f5bi XX(11) = f6bi XX(12) = f7bi	f1bi, f2bi, f3bi, f5bi, f6bi, f7bi are bottom fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at initial stage $$f_{bi} = -\frac{\eta F_i}{A_{net}} \mp \frac{\eta F_i e_i}{S_{bnet}} \pm \frac{M_G}{S_{bnet}}$$
XX(13) = f1tc XX(14) = f2tc XX(15) = f3tc XX(16) = f5tc XX(17) = f6tc XX(18) = f7tc	f1tc, f2tc, f3tc, f5tc, f6tc, f7tc are top fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at second loading stage $$f_t = -\frac{F_e}{A_{tf}} \pm \frac{F_e e}{S_t} \mp \frac{M_P}{S_t} \mp \frac{M_C}{S_{tc}}$$
XX(19) = f1bc XX(20) = f2bc XX(21) = f3bc XX(22) = f5bc XX(23) = f6bc XX(24) = f7bc	f1bc, f2bc, f3bc, f5bc, f6bc, f7bc are bottom fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at second loading stage $$f_b = -\frac{F_e}{A_{tf}} \mp \frac{F_e e}{S_b} \pm \frac{M_P}{S_b} \pm \frac{M_C}{S_{bc}}$$
XX(25) = f1tt XX(26) = f2tt XX(27) = f3tt XX(28) = f5tt XX(29) = f6tt XX(30) = f7tt	f1tt, f2tt, f3tt, f5tt, f6tt, f7tt are top fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at third loading stage $$f_t = -\frac{F_e}{A_{tf}} \pm \frac{F_e e}{S_t} \mp \frac{M_P}{S_t} \mp \frac{M_C}{S_{tc}} \mp \frac{M_L}{S_{tc}}$$

XX(31) = f1bt XX(32) = f2bt XX(33) = f3bt XX(34) = f5bt XX(35) = f6bt XX(36) = f7bt	f1bt, f2bt, f3bt, f5bt, f6bt, f7bt are bottom fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at third loading stage $$f_b = -\frac{F_e}{A_{tf}} \mp \frac{F_e e}{S_b} \pm \frac{M_P}{S_b} \pm \frac{M_C}{S_{bc}} \pm \frac{M_L}{S_{bc}}$$
XX(37) = f1ttt XX(38) = f2ttt XX(39) = f3ttt XX(40) = f5ttt XX(41) = f6ttt XX(42) = f7ttt	f1ttt, f2ttt, f3ttt, f5ttt, f6ttt, f7ttt are top fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at fourth loading stage $$f_t = -\frac{1}{2}\frac{F_e}{A_{tf}} \pm \frac{1}{2}\frac{F_e e}{S_t} \mp \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{tc}}$$
XX(43) = f1btt XX(44) = f2btt XX(45) = f3btt XX(46) = f5btt XX(47) = f6btt XX(48) = f7btt	f1btt, f2btt, f3btt, f5btt, f6btt, f7btt are bottom fiber flexural stresses at section1, section2, section3, section5, section6, section7 respectively at fourth loading stage $$f_b = -\frac{1}{2}\frac{F_e}{A_{tf}} \mp \frac{1}{2}\frac{F_e e}{S_b} \pm \frac{\left(M_L + \frac{M_D}{2}\right)}{S_{bc}}$$
XX(49) = M_{cu}	Calculations are done according to Table 5.5
XX(50) = V_S	Eq. (3.28) and Eq. (3.34); Detail calculations are done in Appendix-A
XX(51) = M^*_{cr}	Eq. (5.27) and Eq. (5.28)
XX(52) = w_c	Eq. (3.23); Reinforcement index of composite girder
XX(53) = Y_{end}	Centroidal distance of tendons at end section
XX(54) = Δ_{LL}	Eq. (5.35)
XX(55) = F_{4i}	Eq. (5.12)
XX(56) = F_{2i}	Eq. (5.10)
XX(57) = F_{2e}	$F_{2i} - Time\ dependent\ losses$
XX(58) = V_{nh}	350*(TF_w*d_S)/(25.4*25.4)/1000*4.45;
XX(59) = t	Deck slab thickness
XX(60) = d_{req}	Detail calculations are done in Appendix-A
XX(61) = M_{pu}	Calculations are done according to Table 5.5
XX(62) = M^*_{crslab}	Eq. (5.29)

XX(66) = w_p XX(67) = F_{sc}	Eq. (3.23); Reinforcement index of precast girder Eq. (3.36) to Eq. (3.38); Detail calculations are done in Appendix-A
XX(68) = f1t XX(69) = f2t XX(70) = f3t	f1t, f2t, f3t are top fiber flexural stresses at section1, section2, section3 respectively at first loading stage $$f_t = -\frac{F_i}{A_{tf}} \pm \frac{F_i e}{S_t} \mp \frac{M_P}{S_t}$$
XX(71) = f1b XX(72) = f2b XX(73) = f3b	f1b, f2b, f3b are bottom fiber flexural stresses at section1, section2, section3 respectively at first loading stage $$f_b = -\frac{F_i}{A_{tf}} \mp \frac{F_i e}{S_b} \pm \frac{M_P}{S_b}$$

UPPER BOUND OF IMPLICIT CONSTRAINTS	LOWER BOUND OF IMPLICIT CONSTRAINTS
XXMAX(1) = $0.25\sqrt{f'_{ci}}$	XXMIN(1) =$0.55f'_{ci}$
XXMAX(2) = $0.25\sqrt{f'_{ci}}$	XXMIN(2) =$0.55f'_{ci}$
XXMAX(3) = $0.25\sqrt{f'_{ci}}$	XXMIN(3) =$0.55f'_{ci}$
XXMAX(4) = $0.25\sqrt{f'_{ci}}$	XXMIN(4) =$0.55f'_{ci}$
XXMAX(5) = $0.25\sqrt{f'_{ci}}$	XXMIN(5) =$0.55f'_{ci}$
XXMAX(6) = $0.25\sqrt{f'_{ci}}$	XXMIN(6) =$0.55f'_{ci}$
XXMAX(7) = $0.25\sqrt{f'_{ci}}$	XXMIN(7) =$0.55f'_{ci}$
XXMAX(8) = $0.25\sqrt{f'_{ci}}$	XXMIN(8) =$0.55f'_{ci}$
	XXMIN(9) =$0.55f'_{ci}$
	XXMIN(10) =$0.55f'_{ci}$
	XXMIN(11) =$0.55f'_{ci}$

XXMAX(9) = $0.25\sqrt{f'_{ci}}$	XXMIN(12) =$0.55f'_{ci}$
XXMAX(10) = $0.25\sqrt{f'_{ci}}$	XXMIN(13) = $0.40f'_c$
XXMAX(11) = $0.25\sqrt{f'_{ci}}$	XXMIN(14) = $0.40f'_c$
XXMAX(12) = $0.25\sqrt{f'_{ci}}$	XXMIN(15) = $0.40f'_c$
XXMAX(13) = $0.5\sqrt{f'_c}$	XXMIN(16) = $0.40f'_c$
XXMAX(14) = $0.5\sqrt{f'_c}$	XXMIN(17) = $0.40f'_c$
XXMAX(15) = $0.5\sqrt{f'_c}$	XXMIN(18) = $0.40f'_c$
XXMAX(16) = $0.5\sqrt{f'_c}$	XXMIN(19) = $0.40f'_c$
XXMAX(17) = $0.5\sqrt{f'_c}$	XXMIN(20) = $0.40f'_c$
XXMAX(18) = $0.5\sqrt{f'_c}$	XXMIN(21) = $0.40f'_c$
XXMAX(19) = $0.5\sqrt{f'_c}$	XXMIN(22) = $0.40f'_c$
XXMAX(20) = $0.5\sqrt{f'_c}$	XXMIN(23) = $0.40f'_c$
XXMAX(21) = $0.5\sqrt{f'_c}$	XXMIN(24) = $0.40f'_c$
XXMAX(22) = $0.5\sqrt{f'_c}$	XXMIN(25) = $0.60f'_c$
XXMAX(23) = $0.5\sqrt{f'_c}$	XXMIN(26) = $0.60f'_c$
XXMAX(24) = $0.5\sqrt{f'_c}$	XXMIN(27) = $0.60f'_c$
XXMAX(25) = $0.5\sqrt{f'_c}$	XXMIN(28) = $0.60f'_c$
XXMAX(26) = $0.5\sqrt{f'_c}$	XXMIN(29) = $0.60f'_c$
XXMAX(27) = $0.5\sqrt{f'_c}$	XXMIN(30) = $0.60f'_c$
XXMAX(28) = $0.5\sqrt{f'_c}$	XXMIN(31) = $0.60f'_c$
XXMAX(29) = $0.5\sqrt{f'_c}$	XXMIN(32) = $0.60f'_c$
XXMAX(30) = $0.5\sqrt{f'_c}$	XXMIN(33) = $0.60f'_c$
XXMAX(31) = $0.5\sqrt{f'_c}$	XXMIN(34) = $0.60f'_c$
XXMAX(32) = $0.5\sqrt{f'_c}$	XXMIN(35) = $0.60f'_c$
XXMAX(33) = $0.5\sqrt{f'_c}$	XXMIN(36) = $0.60f'_c$
XXMAX(34) = $0.5\sqrt{f'_c}$	XXMIN(37) = $0.40f'_c$
	XXMIN(38) = $0.40f'_c$
	XXMIN(39) = $0.40f'_c$
	XXMIN(40) = $0.40f'_c$
	XXMIN(41) = $0.40f'_c$
	XXMIN(42) = $0.40f'_c$

XXMAX(35) = $0.5\sqrt{f_c'}$	XXMIN(43) = $0.40f_c'$
XXMAX(36) = $0.5\sqrt{f_c'}$	XXMIN(44) = $0.40f_c'$
XXMAX(37) = $0.5\sqrt{f_c'}$	XXMIN(45) = $0.40f_c'$
XXMAX(38) = $0.5\sqrt{f_c'}$	XXMIN(46) = $0.40f_c'$
XXMAX(39) = $0.5\sqrt{f_c'}$	XXMIN(47) = $0.40f_c'$
XXMAX(40) =$0.5\sqrt{f_c'}$	XXMIN(48) = $0.40f_c'$
XXMAX(41) =$0.5\sqrt{f_c'}$	XXMIN(49) = 0.0
XXMAX(42) = $0.5\sqrt{f_c'}$	XXMIN(50) = 0.0
XXMAX(43) = $0.5\sqrt{f_c'}$	XXMIN(51) = 0.0
XXMAX(44) = $0.5\sqrt{f_c'}$	XXMIN(52) = 0.0
XXMAX(45) = $0.5\sqrt{f_c'}$	XXMIN(53) = $\frac{G_d}{2} - (\frac{G_d}{6} + 0.25\sqrt{f_{ci}}\ \frac{A_4\,G_d}{6\,F_{4i}})$
XXMAX(46) =$0.5\sqrt{f_c'}$	XXMIN(54) = 0.0
XXMAX(47) =$0.5\sqrt{f_c'}$	XXMIN(55) = 0.0
XXMAX(48) = $0.5\sqrt{f_c'}$	XXMIN(56) = 0.0
XXMAX(49) = φM_{cn}	XXMIN(57) = 0.0
XXMAX(50) =$0.666\sqrt{f_c}W_w d_s$	XXMIN(58) = $V_u/0.9$
XXMAX(51) = $\varphi M_{cn}/1.2$	XXMIN(59) = (S_d +17)/3*25.4
XXMAX(52) = $0.36\beta_1$	XXMIN(60) = d_{min}
XXMAX(53) = $\frac{G_d}{2} + (\frac{G_d}{6} + 0.5\sqrt{\mathrm{f_c'}}\ \frac{A_4\,G_d}{6\,F_{4e}})$	XXMIN(61) = 0.0
XXMAX(54) = L/800	XXMIN(62) = 0.0
XXMAX(55) = $0.7f_{su}A_s$	XXMIN(63) = 0.0
XXMAX(56) = $0.83\ f_y^*A_s$	XXMIN(64) = 0.0
XXMAX(57) = $0.80\ f_y^*A_s$	XXMIN(65) = 0.0
XXMAX(58) =100000	XXMIN(66) = 0.0
XXMAX(59) = 300.0	XXMIN(67) = 1.5
XXMAX(60) = d_{prov}	XXMIN(68) = $0.60f_c'$
XXMAX(61) = φM_{pn}	XXMIN(69) = $0.60f_c'$
XXMAX(62) = $\varphi M_{nslab}/1.2$	XXMIN(70) = $0.60f_c'$
XXMAX(66) = $0.36\beta_1$	XXMIN(71) = $0.60f_c'$
XXMAX(67) = 100.0	XXMIN(72) = $0.60f_c'$
	XXMIN(73) = $0.60f_c'$

XXMAX(68) = $0.5\sqrt{f_c'}$ XXMAX(69) = $0.5\sqrt{f_c'}$ XXMAX(70) = $0.5\sqrt{f_c'}$ XXMAX(71) = $0.5\sqrt{f_c'}$ XXMAX(72) = $0.5\sqrt{f_c'}$ XXMAX(73) = $0.5\sqrt{f_c'}$	

UPPER BOUND OF IMPLICIT CONSTRAINTS AT THE STARTING POINT	LOWER BOUND OF IMPLICIT CONSTRAINTS AT THE STARTING POINT
XXMAX(1) = .13693064E+01	XXMIN(1) = -.16500000E+02
XXMAX(2) = .13693064E+01	XXMIN(2) = -.16500000E+02
XXMAX(3) = .13693064E+01	XXMIN(3) = -.16500000E+02
XXMAX(4) = .13693064E+01	XXMIN(4) = -.16500000E+02
XXMAX(5) = .13693064E+01	XXMIN(5) = -.16500000E+02
XXMAX(6) = .13693064E+01	XXMIN(6) = -.16500000E+02
XXMAX(7) = .13693064E+01	XXMIN(7) = -.16500000E+02
XXMAX(8) = .13693064E+01	XXMIN(8) = -.16500000E+02
XXMAX(9) = .13693064E+01	XXMIN(9) = -.16500000E+02
XXMAX(10) = .13693064E+01	XXMIN(10) = -.16500000E+02
XXMAX(11) = .13693064E+01	XXMIN(11) = -.16500000E+02
XXMAX(12) = .13693064E+01	XXMIN(12) = -.16500000E+02
XXMAX(13) = .31622777E+01	XXMIN(13) = -.16000000E+02
XXMAX(14) = .31622777E+01	XXMIN(14) = -.16000000E+02
XXMAX(15) = .31622777E+01	XXMIN(15) = -.16000000E+02
XXMAX(16) = .31622777E+01	XXMIN(16) = -.16000000E+02
XXMAX(17) = .31622777E+01	XXMIN(17) = -.16000000E+02
XXMAX(18) = .31622777E+01	XXMIN(18) = -.16000000E+02
XXMAX(19) = .31622777E+01	XXMIN(19) = -.16000000E+02
XXMAX(20) = .31622777E+01	XXMIN(20) = -.16000000E+02
XXMAX(21) = .31622777E+01	XXMIN(21) = -.16000000E+02
XXMAX(22) = .31622777E+01	XXMIN(22) = -.16000000E+02

XXMAX(23) =	.31622777E+01	XXMIN(23) =	-.16000000E+02
XXMAX(24) =	.31622777E+01	XXMIN(24) =	-.16000000E+02
XXMAX(25) =	.31622777E+01	XXMIN(25) =	-.24000000E+02
XXMAX(26) =	.31622777E+01	XXMIN(26) =	-.24000000E+02
XXMAX(27) =	.31622777E+01	XXMIN(27) =	-.24000000E+02
XXMAX(28) =	.31622777E+01	XXMIN(28) =	-.24000000E+02
XXMAX(29) =	.31622777E+01	XXMIN(29) =	-.24000000E+02
XXMAX(30) =	.31622777E+01	XXMIN(30) =	-.24000000E+02
XXMAX(31) =	.31622777E+01	XXMIN(31) =	-.24000000E+02
XXMAX(32) =	.31622777E+01	XXMIN(32) =	-.24000000E+02
XXMAX(33) =	.31622777E+01	XXMIN(33) =	-.24000000E+02
XXMAX(34) =	.31622777E+01	XXMIN(34) =	-.24000000E+02
XXMAX(35) =	.31622777E+01	XXMIN(35) =	-.24000000E+02
XXMAX(36) =	.31622777E+01	XXMIN(36) =	-.24000000E+02
XXMAX(37) =	.31622777E+01	XXMIN(37) =	-.16000000E+02
XXMAX(38) =	.31622777E+01	XXMIN(38) =	-.16000000E+02
XXMAX(39) =	.31622777E+01	XXMIN(39) =	-.16000000E+02
XXMAX(40) =	.31622777E+01	XXMIN(40) =	-.16000000E+02
XXMAX(41) =	.31622777E+01	XXMIN(41) =	-.16000000E+02
XXMAX(42) =	.31622777E+01	XXMIN(42) =	-.16000000E+02
XXMAX(43) =	.31622777E+01	XXMIN(43) =	-.16000000E+02
XXMAX(44) =	.31622777E+01	XXMIN(44) =	-.16000000E+02
XXMAX(45) =	.31622777E+01	XXMIN(45) =	-.16000000E+02
XXMAX(46) =	.31622777E+01	XXMIN(46) =	-.16000000E+02
XXMAX(47) =	.31622777E+01	XXMIN(47) =	-.16000000E+02
XXMAX(48) =	.31622777E+01	XXMIN(48) =	-.16000000E+02
XXMAX(49) =	.36280958E+08	XXMIN(49) =	.00000000E+00
XXMAX(50) =	.17478754E+04	XXMIN(50) =	.00000000E+00
XXMAX(51) =	.30234132E+08	XXMIN(51) =	.00000000E+00
XXMAX(52) =	.27000000E+00	XXMIN(52) =	.00000000E+00
XXMAX(53) =	.18561776E+04	XXMIN(53) =	.77051056E+03
XXMAX(54) =	.61000000E+02	XXMIN(54) =	.00000000E+00
XXMAX(55) =	.11489814E+05	XXMIN(55) =	.00000000E+00

XXMAX(56) =	.12261273E+05	XXMIN(56) =	.00000000E+00
XXMAX(57) =	.11818094E+05	XXMIN(57) =	.00000000E+00
XXMAX(58) =	.10000000E+06	XXMIN(58) =	.17714722E+04
XXMAX(59) =	.30000000E+03	XXMIN(59) =	.19704473E+03
XXMAX(60) =	.17300000E+03	XXMIN(60) =	.79049716E+02
XXMAX(61) =	.14889011E+08	XXMIN(61) =	.00000000E+00
XXMAX(62) =	.37443427E+05	XXMIN(62) =	.00000000E+00
XXMAX(66) =	.27000000E+00	XXMIN(66) =	.00000000E+00
XXMAX(67) =	.10000000E+03	XXMIN(67) =	.15000000E+01
XXMAX(68) =	.31622777E+01	XXMIN(68) =	-.24000000E+02
XXMAX(69) =	.31622777E+01	XXMIN(69) =	-.24000000E+02
XXMAX(70) =	.31622777E+01	XXMIN(70) =	-.24000000E+02
XXMAX(71) =	.31622777E+01	XXMIN(71) =	-.24000000E+02
XXMAX(72) =	.31622777E+01	XXMIN(72) =	-.24000000E+02
XXMAX(73) =	.31622777E+01	XXMIN(73) =	-.24000000E+02

IMPLICIT CONSTRAINTS AT THE STARTING POINT

XX(1) =	-.51534951E+01	XX(38) =	.91364319E+04
XX(2) =	-.78275140E+01	XX(39) =	.38575891E+04
XX(3) =	-.58464857E+01	XX(40) =	.23000000E+03
XX(4) =	-.90183803E+01	XX(41) =	.15055625E+03
XX(5) =	-.62977662E+01	XX(42) =	.66089805E+07
XX(6) =	-.94254173E+01	XX(43) =	.27552083E+05
XX(7) =	-.12798985E+02	XX(44) =	.13905571E+00
XX(8) =	-.13424276E+02	XX(45) =	.18554627E+01
XX(9) =	-.12657727E+02	XX(46) =	-.93922106E+01
XX(10) =	-.78140814E+01	XX(47) =	-.15481798E+02
XX(11) =	-.85757223E+01	XX(48) =	-.11272259E+02
XX(12) =	-.10593743E+02	XX(49) =	-.18917406E+02
XX(13) =	-.14913684E+02	XX(50) =	-.13616007E+02
XX(14) =	-.13994308E+02	XX(51) =	-.18917406E+02
XX(15) =	-.14200353E+02	XX(52) =	.38575891E+04
XX(16) =	-.29470536E+01	XX(53) =	.23000000E+03

XX(17) = -.71646409E+01	XX(54) = .15055625E+03
XX(18) = -.69501662E+01	XX(55) = .66089805E+07
XX(19) = -.16537192E+01	XX(56) = .27552083E+05
XX(20) = -.54253256E+01	XX(57) = .13905571E+00
XX(21) = -.28598239E+01	XX(58) = .18554627E+01
XX(22) = -.15041360E+01	XX(59) = -.93922106E+01
XX(23) = -.36658537E+01	XX(60) = -.15481798E+02
XX(24) = -.85757223E+01	XX(61) = -.11272259E+02
XX(25) = -.10593743E+02	XX(62) = -.18917406E+02
XX(26) = -.14913684E+02	XX(63) = -.13616007E+02
XX(27) = -.13994308E+02	XX(64) = -.18917406E+02
XX(28) = -.14200353E+02	XX(65) = .13905571E+00
XX(29) = -.29470536E+01	XX(66) = .18554627E+01
XX(30) = -.71646409E+01	XX(67) = -.93922106E+01
XX(31) = -.69501662E+01	XX(68) = -.15481798E+02
XX(32) = -.16537192E+01	XX(69) = -.11272259E+02
XX(33) = -.54253256E+01	XX(70) = -.18917406E+02
XX(34) = -.28598239E+01	XX(71) = -.13616007E+02
XX(35) = -.15041360E+01	XX(72) = -.18917406E+02
XX(36) = -.36658537E+01	XX(73) = -.18917406E+02
XX(37) = -.36658537E+01	

INITIAL COMPLEX CONFIGURATION

All the vertices of the initial complex shown below are feasible solution or design of the bridge. It indicates that a lot of design of the bridge can be done with the different costs of the bridge. The design of the bridge which yields the minimum cost is the optimum design.

VERTEX NUMBER 1	VERTEX NUMBER 9
FUNCTION VALUE = .53064839E+07	FUNCTION VALUE = .57796738E+07
COORDINATES	COORDINATES
XT(1) = .3563000E+04	XT(1) = .3583000E+04
XT(2) = .5680000E+03	XT(2) = .5730000E+03
XT(3) = .26800000E+03	XT(3) = .34100000E+03
XT(4) = .34700000E+03	XT(4) = .33400000E+03
XT(5) = .19000000E+04	XT(5) = .19400000E+04
XT(6) = .80000000E+01	XT(6) = .80000000E+01
XT(7) = .50000000E+01	XT(7) = .50000000E+01
XT(8) = .56400000E+03	XT(8) = .56400000E+03
XT(9) = .27170000E+00	XT(9) = .27170000E+00
XT(10) = .35500000E+03	XT(10) = .35500000E+03
XT(11) = .58849000E-02	XT(11) = .58849000E-02
XT(12) = .67000000E+02	XT(12) = .67000000E+02
XT(13) = .50000000E+02	XT(13) = .50000000E+02
XT(14) = .15000000E+03	XT(14) = .15000000E+03
	VERTEX NUMBER 10
VERTEX NUMBER 2	FUNCTION VALUE = .50476636E+07
FUNCTION VALUE = .52784440E+07	COORDINATES
COORDINATES	XT(1) = .3563000E+04
XT(1) = .36900000E+04	XT(2) = .5680000E+03
XT(2) = .3200000E+03	XT(3) = .26800000E+03
XT(3) = .42900000E+03	XT(4) = .34700000E+03
XT(4) = .32000000E+03	XT(5) = .19000000E+04
XT(5) = .19000000E+04	XT(6) = .80000000E+01
XT(6) = .90000000E+01	XT(7) = .50000000E+01
XT(7) = .40000000E+01	XT(8) = .56400000E+03
XT(8) = .39000000E+03	XT(9) = .27170000E+00
XT(9) = .34170000E+00	XT(10) = .35500000E+03
XT(10) = .32500000E+03	XT(11) = .58849000E-02
XT(11) = .54989000E-02	XT(12) = .67000000E+02
XT(12) = .75000000E+02	XT(13) = .50000000E+02

XT(13) = .50000000E+02	XT(14) = .15000000E+03
XT(14) = .15000000E+03	
VERTEX NUMBER 3	VERTEX NUMBER 11
FUNCTION VALUE = .54864721E+07	FUNCTION VALUE = .52410747E+07
COORDINATES	COORDINATES
XT(1) = .3583000E+04	XT(1) = .36900000E+04
XT(2) = .5730000E+03	XT(2) = .3200000E+03
XT(3) = .34100000E+03	XT(3) = .42900000E+03
XT(4) = .33400000E+03	XT(4) = .32000000E+03
XT(5) = .19400000E+04	XT(5) = .19000000E+04
XT(6) = .80000000E+01	XT(6) = .90000000E+01
XT(7) = .50000000E+01	XT(7) = .40000000E+01
XT(8) = .56400000E+03	XT(8) = .39000000E+03
XT(9) = .27170000E+00	XT(9) = .34170000E+00
XT(10) = .35500000E+03	XT(10) = .32500000E+03
XT(11) = .58849000E-02	XT(11) = .54989000E-02
XT(12) = .67000000E+02	XT(12) = .75000000E+02
XT(13) = .50000000E+02	XT(13) = .50000000E+02
XT(14) = .15000000E+03	XT(14) = .15000000E+03
VERTEX NUMBER 4	
FUNCTION VALUE = .56230794E+07	VERTEX NUMBER 12
COORDINATES	FUNCTION VALUE = .58728819E+07
XT(1) = .36900000E+04	COORDINATES
XT(2) = .3200000E+03	XT(1) = .3583000E+04
XT(3) = .42900000E+03	XT(2) = .5730000E+03
XT(4) = .32000000E+03	XT(3) = .34100000E+03
XT(5) = .19000000E+04	XT(4) = .33400000E+03
XT(6) = .90000000E+01	XT(5) = .19400000E+04
XT(7) = .40000000E+01	XT(6) = .80000000E+01
XT(8) = .39000000E+03	XT(7) = .50000000E+01
XT(9) = .34170000E+00	XT(8) = .56400000E+03
XT(10) = .32500000E+03	XT(9) = .27170000E+00

XT(11) = .54989000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 5

FUNCTION VALUE = .55566911E+07

COORDINATES

XT(1) = .3563000E+04

XT(2) = .5680000E+03

XT(3) = .26800000E+03

XT(4) = .34700000E+03

XT(5) = .19000000E+04

XT(6) = .80000000E+01

XT(7) = .50000000E+01

XT(8) = .56400000E+03

XT(9) = .27170000E+00

XT(10) = .35500000E+03

XT(11) = .58849000E-02

XT(12) = .67000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 6

FUNCTION VALUE = .55130258E+07

COORDINATES

XT(1) = .3583000E+04

XT(2) = .5730000E+03

XT(3) = .34100000E+03

XT(4) = .33400000E+03

XT(5) = .19400000E+04

XT(6) = .80000000E+01

XT(7) = .50000000E+01

XT(8) = .56400000E+03

XT(10) = .35500000E+03

XT(11) = .58849000E-02

XT(12) = .67000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 13

FUNCTION VALUE = .57308934E+07

COORDINATES

XT(1) = .36900000E+04

XT(2) = .3200000E+03

XT(3) = .42900000E+03

XT(4) = .32000000E+03

XT(5) = .19000000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .39000000E+03

XT(9) = .34170000E+00

XT(10) = .32500000E+03

XT(11) = .54989000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 14

FUNCTION VALUE = .60453512E+07

COORDINATES

XT(1) = .3563000E+04

XT(2) = .5680000E+03

XT(3) = .26800000E+03

XT(4) = .34700000E+03

XT(5) = .19000000E+04

XT(9) = .27170000E+00
XT(10) = .35500000E+03
XT(11) = .58849000E-02
XT(12) = .67000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 7
FUNCTION VALUE = .50098596E+07
COORDINATES

XT(1) = .3563000E+04
XT(2) = .5680000E+03
XT(3) = .26800000E+03
XT(4) = .34700000E+03
XT(5) = .19000000E+04
XT(6) = .80000000E+01
XT(7) = .50000000E+01
XT(8) = .56400000E+03
XT(9) = .27170000E+00
XT(10) = .35500000E+03
XT(11) = .58849000E-02
XT(12) = .67000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 8
FUNCTION VALUE = .55388064E+07
COORDINATES

XT(1) = .36900000E+04
XT(2) = .3200000E+03
XT(3) = .42900000E+03
XT(4) = .32000000E+03
XT(5) = .19000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(6) = .80000000E+01
XT(7) = .50000000E+01
XT(8) = .56400000E+03
XT(9) = .27170000E+00
XT(10) = .35500000E+03
XT(11) = .58849000E-02
XT(12) = .67000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 15
FUNCTION VALUE = .56236387E+07
COORDINATES

XT(1) = .3583000E+04
XT(2) = .5730000E+03
XT(3) = .34100000E+03
XT(4) = .33400000E+03
XT(5) = .19400000E+04
XT(6) = .80000000E+01
XT(7) = .50000000E+01
XT(8) = .56400000E+03
XT(9) = .27170000E+00
XT(10) = .35500000E+03
XT(11) = .58849000E-02
XT(12) = .67000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

XT(8) = .39000000E+03 XT(9) = .34170000E+00 XT(10) = .32500000E+03 XT(11) = .54989000E-02 XT(12) = .75000000E+02 XT(13) = .50000000E+02 XT(14) = .15000000E+03	

OUTPUT SUMMARY FROM SUBROUTINE EVOP

MINIMUM OF THE OBJECTIVE FUNCTION HAS BEEN LOCATED TO THE DESIRED DEGREE OF ACCURACY FOR CONVERGENCE. IER = 8

TOTAL NUMBER OF OBJECTIVE FUNCTION EVALUATION.

NFUNC = 1267

NUMBER OF TIMES THE SUBROUTINE FUNCTION IS CALLED DURING THE PRESENT CONVERGENCE TESTS. KUT = 6

NUMBER OF TIMES THE EXPLICIT CONSTRAINTS WERE EVALUATED

KKT = 4346

NUMBER OF TIMES THE IMPLICIT CONSTRAINTS WERE EVALUATED

M = 2142

COORDINATES OF THE MINIMUM

XT(1) = .40000000E+04	XT(8) = .43000000E+03
XT(2) = .4500000E+03	XT(9) = .27170000E+00
XT(3) = .32500000E+03	XT(10) = .25500000E+03
XT(4) = .26000000E+03	XT(11) = .62849000E-02
XT(5) = .17000000E+04	XT(12) = .75000000E+02
XT(6) = .90000000E+01	XT(13) = .50000000E+02
XT(7) = .40000000E+01	XT(14) = .15000000E+03

OBJECTIVE FUNCTION VALUE AT THE MINIMUM **F = .43409822E+07**

IMPLICIT CONSTRAINT VALUES AT THE MINIMUM

XX(1) =	-.51534951E+01	XX(38) =	.91364319E+04
XX(2) =	-.78275140E+01	XX(39) =	.38575891E+04
XX(3) =	-.58464857E+01	XX(40) =	.23000000E+03
XX(4) =	-.90183803E+01	XX(41) =	.15055625E+03
XX(5) =	-.62977662E+01	XX(42) =	.66089805E+07
XX(6) =	-.94254173E+01	XX(43) =	.27552083E+05
XX(7) =	-.12798985E+02	XX(44) =	.13905571E+00
XX(8) =	-.13424276E+02	XX(45) =	.18554627E+01
XX(9) =	-.12657727E+02	XX(46) =	-.93922106E+01
XX(10) =	-.78140814E+01	XX(47) =	-.15481798E+02
XX(11) =	-.85757223E+01	XX(48) =	-.11272259E+02
XX(12) =	-.10593743E+02	XX(49) =	-.18917406E+02
XX(13) =	-.14913684E+02	XX(50) =	-.13616007E+02
XX(14) =	-.13994308E+02	XX(51) =	-.18917406E+02
XX(15) =	-.14200353E+02	XX(52) =	.38575891E+04
XX(16) =	-.29470536E+01	XX(53) =	.23000000E+03
XX(17) =	-.71646409E+01	XX(54) =	.15055625E+03
XX(18) =	-.69501662E+01	XX(55) =	.66089805E+07
XX(19) =	-.16537192E+01	XX(56) =	.27552083E+05
XX(20) =	-.54253256E+01	XX(57) =	.13905571E+00
XX(21) =	-.28598239E+01	XX(58) =	.18554627E+01
XX(22) =	-.15041360E+01	XX(59) =	-.93922106E+01
XX(23) =	-.36658537E+01	XX(60) =	-.15481798E+02
XX(24) =	-.85757223E+01	XX(61) =	-.11272259E+02
XX(25) =	-.10593743E+02	XX(62) =	-.18917406E+02
XX(26) =	-.14913684E+02	XX(63) =	-.13616007E+02
XX(27) =	-.13994308E+02	XX(64) =	-.18917406E+02
XX(28) =	-.14200353E+02	XX(65) =	.13905571E+00
XX(29) =	-.29470536E+01	XX(66) =	.18554627E+01
XX(30) =	-.71646409E+01	XX(67) =	-.93922106E+01
XX(31) =	-.69501662E+01	XX(68) =	-.15481798E+02
XX(32) =	-.16537192E+01	XX(69) =	-.11272259E+02

XX(33) = -.54253256E+01 XX(34) = -.28598239E+01 XX(35) = -.15041360E+01 XX(36) = -.36658537E+01 XX(37) = -.36658537E+01	XX(70) = -.18917406E+02 XX(71) = -.13616007E+02 XX(72) = -.18917406E+02 XX(73) = -.18917406E+02

FINAL COMPLEX CONFIGURATION

The co ordinates o f t he v ertices o f t he f inal co mplex af ter co nvergence ar e sh own below.

VERTEX NUMBER 1 FUNCTION VALUE = .42824304E+07 COORDINATES XT(1) = .389000000E+04 XT(2) = .4450000E+03 XT(3) = .32000000E+03 XT(4) = .25500000E+03 XT(5) = .18000000E+04 XT(6) = .90000000E+01 XT(7) = .40000000E+01 XT(8) = .42500000E+03 XT(9) = .26370000E+00 XT(10) = .26500000E+03 XT(11) = .62849000E-02 XT(12) = .75000000E+02 XT(13) = .50000000E+02 XT(14) = .15000000E+03 VERTEX NUMBER 2 FUNCTION VALUE = .42834434E+07 COORDINATES XT(1) = .39900000E+04 XT(2) = .4520000E+03	VERTEX NUMBER 9 FUNCTION VALUE = .42826755E+07 COORDINATES XT(1) = .387500000E+04 XT(2) = .4320000E+03 XT(3) = .32500000E+03 XT(4) = .25000000E+03 XT(5) = .19000000E+04 XT(6) = .80000000E+01 XT(7) = .40000000E+01 XT(8) = .43000000E+03 XT(9) = .26370000E+00 XT(10) = .26500000E+03 XT(11) = .62849000E-02 XT(12) = .75000000E+02 XT(13) = .50000000E+02 XT(14) = .15000000E+03 VERTEX NUMBER 10 FUNCTION VALUE = .42812646E+07 COORDINATES XT(1) = .389000000E+04 XT(2) = .4450000E+03

XT(3) = .32000000E+03

XT(4) = .25500000E+03

XT(5) = .16500000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .27170000E+00

XT(10) = .25500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 3

FUNCTION VALUE = .42809856E+07

COORDINATES

XT(1) = .387500000E+04

XT(2) = .4320000E+03

XT(3) = .32500000E+03

XT(4) = .25000000E+03

XT(5) = .19000000E+04

XT(6) = .80000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .26370000E+00

XT(10) = .26500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 4

FUNCTION VALUE = .42824698E+07

COORDINATES

XT(3) = .32000000E+03

XT(4) = .25500000E+03

XT(5) = .18000000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .42500000E+03

XT(9) = .26370000E+00

XT(10) = .26500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 11

FUNCTION VALUE = .42828413E+07

COORDINATES

XT(1) = .39900000E+04

XT(2) = .4520000E+03

XT(3) = .32000000E+03

XT(4) = .25500000E+03

XT(5) = .16500000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .27170000E+00

XT(10) = .25500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 12

FUNCTION VALUE = .42819999E+07

COORDINATES

XT(1) = .387500000E+04

XT(1) = .39900000E+04	XT(2) = .4320000E+03
XT(2) = .4520000E+03	XT(3) = .32500000E+03
XT(3) = .32000000E+03	XT(4) = .25000000E+03
XT(4) = .25500000E+03	XT(5) = .19000000E+04
XT(5) = .16500000E+04	XT(6) = .80000000E+01
XT(6) = .90000000E+01	XT(7) = .40000000E+01
XT(7) = .40000000E+01	XT(8) = .43000000E+03
XT(8) = .43000000E+03	XT(9) = .26370000E+00
XT(9) = .27170000E+00	XT(10) = .26500000E+03
XT(10) = .25500000E+03	XT(11) = .62849000E-02
XT(11) = .62849000E-02	XT(12) = .75000000E+02
XT(12) = .75000000E+02	XT(13) = .50000000E+02
XT(13) = .50000000E+02	XT(14) = .15000000E+03
XT(14) = .15000000E+03	VERTEX NUMBER 13
VERTEX NUMBER 5	FUNCTION VALUE = .42820959E+07
FUNCTION VALUE = .42830152E+07	COORDINATES
COORDINATES	XT(1) = .39900000E+04
XT(1) = .389000000E+04	XT(2) = .4520000E+03
XT(2) = .4450000E+03	XT(3) = .32000000E+03
XT(3) = .32000000E+03	XT(4) = .25500000E+03
XT(4) = .25500000E+03	XT(5) = .16500000E+04
XT(5) = .18000000E+04	XT(6) = .90000000E+01
XT(6) = .90000000E+01	XT(7) = .40000000E+01
XT(7) = .40000000E+01	XT(8) = .43000000E+03
XT(8) = .42500000E+03	XT(9) = .27170000E+00
XT(9) = .26370000E+00	XT(10) = .25500000E+03
XT(10) = .26500000E+03	XT(11) = .62849000E-02
XT(11) = .62849000E-02	XT(12) = .75000000E+02
XT(12) = .75000000E+02	XT(13) = .50000000E+02
XT(13) = .50000000E+02	XT(14) = .15000000E+03
XT(14) = .15000000E+03	
VERTEX NUMBER 6	
FUNCTION VALUE = .42826158E+07	VERTEX NUMBER 14

COORDINATES	FUNCTION VALUE = .42837932E+07
XT(1) = .387500000E+04	COORDINATES
XT(2) = .4320000E+03	XT(1) = .389000000E+04
XT(3) = .32500000E+03	XT(2) = .4450000E+03
XT(4) = .25000000E+03	XT(3) = .32000000E+03
XT(5) = .19000000E+04	XT(4) = .25500000E+03
XT(6) = .80000000E+01	XT(5) = .18000000E+04
XT(7) = .40000000E+01	XT(6) = .90000000E+01
XT(8) = .43000000E+03	XT(7) = .40000000E+01
XT(9) = .26370000E+00	XT(8) = .42500000E+03
XT(10) = .26500000E+03	XT(9) = .26370000E+00
XT(11) = .62849000E-02	XT(10) = .26500000E+03
XT(12) = .75000000E+02	XT(11) = .62849000E-02
XT(13) = .50000000E+02	XT(12) = .75000000E+02
XT(14) = .15000000E+03	XT(13) = .50000000E+02
	XT(14) = .15000000E+03
VERTEX NUMBER 7	VERTEX NUMBER 15
FUNCTION VALUE = .42815025E+07	FUNCTION VALUE = .42828669E+07
COORDINATES	COORDINATES
XT(1) = .389000000E+04	XT(1) = .387500000E+04
XT(2) = .4450000E+03	XT(2) = .4320000E+03
XT(3) = .32000000E+03	XT(3) = .32500000E+03
XT(4) = .25500000E+03	XT(4) = .25000000E+03
XT(5) = .18000000E+04	XT(5) = .19000000E+04
XT(6) = .90000000E+01	XT(6) = .80000000E+01
XT(7) = .40000000E+01	XT(7) = .40000000E+01
XT(8) = .42500000E+03	XT(8) = .43000000E+03
XT(9) = .26370000E+00	XT(9) = .26370000E+00
XT(10) = .26500000E+03	XT(10) = .26500000E+03
XT(11) = .62849000E-02	XT(11) = .62849000E-02
XT(12) = .75000000E+02	XT(12) = .75000000E+02
XT(13) = .50000000E+02	XT(13) = .50000000E+02
XT(14) = .15000000E+03	XT(14) = .15000000E+03

VERTEX NUMBER 8	
FUNCTION VALUE = .42824856E+07	
COORDINATES	
XT(1) = .39900000E+04	
XT(2) = .4520000E+03	
XT(3) = .32000000E+03	
XT(4) = .25500000E+03	
XT(5) = .16500000E+04	
XT(6) = .90000000E+01	
XT(7) = .40000000E+01	
XT(8) = .43000000E+03	
XT(9) = .27170000E+00	
XT(10) = .25500000E+03	
XT(11) = .62849000E-02	
XT(12) = .75000000E+02	
XT(13) = .50000000E+02	
XT(14) = .15000000E+03	

FIRST RESTART OF "EVOP" TO CHECK THE MINIMUM

Automatic r estart o f E VOP t akes p lace t o check w hether t he p reviously o btained minimum i s t he g lobal minimum. The i nitial c omplex a s s hown be low i s ge nerated taking t he coordinates of t he pr evious m inimum (values obt ained from pr evious execution of EVOP) as the starting point of the complex.

INITIAL COMPLEX CONFIGURATION

VERTEX NUMBER 1	VERTEX NUMBER 9
FUNCTION VALUE = .42805835E+07	FUNCTION VALUE = .43559536E+07
COORDINATES	COORDINATES
XT(1) = .39900000E+04	XT(1) = .36900000E+04
XT(2) = .4520000E+03	XT(2) = .3200000E+03
XT(3) = .32000000E+03	XT(3) = .42900000E+03
XT(4) = .25500000E+03	XT(4) = .32000000E+03

XT(5) = .16500000E+04	XT(5) = .19000000E+04
XT(6) = .90000000E+01	XT(6) = .90000000E+01
XT(7) = .40000000E+01	XT(7) = .40000000E+01
XT(8) = .43000000E+03	XT(8) = .39000000E+03
XT(9) = .27170000E+00	XT(9) = .34170000E+00
XT(10) = .25500000E+03	XT(10) = .32500000E+03
XT(11) = .62849000E-02	XT(11) = .54989000E-02
XT(12) = .75000000E+02	XT(12) = .75000000E+02
XT(13) = .50000000E+02	XT(13) = .50000000E+02
XT(14) = .15000000E+03	XT(14) = .15000000E+03
VERTEX NUMBER 2	
FUNCTION VALUE = .42278859E+07	VERTEX NUMBER 10
COORDINATES	FUNCTION VALUE = .44066776E+07
XT(1) = .36900000E+04	COORDINATES
XT(2) = .3200000E+03	XT(1) = .39900000E+04
XT(3) = .42900000E+03	XT(2) = .4520000E+03
XT(4) = .32000000E+03	XT(3) = .32000000E+03
XT(5) = .19000000E+04	XT(4) = .25500000E+03
XT(6) = .90000000E+01	XT(5) = .16500000E+04
XT(7) = .40000000E+01	XT(6) = .90000000E+01
XT(8) = .39000000E+03	XT(7) = .40000000E+01
XT(9) = .34170000E+00	XT(8) = .43000000E+03
XT(10) = .32500000E+03	XT(9) = .27170000E+00
XT(11) = .54989000E-02	XT(10) = .25500000E+03
XT(12) = .75000000E+02	XT(11) = .62849000E-02
XT(13) = .50000000E+02	XT(12) = .75000000E+02
XT(14) = .15000000E+03	XT(13) = .50000000E+02
VERTEX NUMBER 3	XT(14) = .15000000E+03
FUNCTION VALUE = .44438093E+07	VERTEX NUMBER 11
COORDINATES	FUNCTION VALUE = .47584549E+07
XT(1) = .389000000E+04	COORDINATES
XT(2) = .4450000E+03	XT(1) = .389000000E+04
XT(3) = .32000000E+03	XT(2) = .4450000E+03

XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 4
FUNCTION VALUE = .45273709E+07
COORDINATES
XT(1) = .389000000E+04
XT(2) = .4450000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03
VERTEX NUMBER 5
FUNCTION VALUE = .52749319E+07
COORDINATES
XT(1) = .389000000E+04

XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 12
FUNCTION VALUE = .43274612E+07
COORDINATES
XT(1) = .39900000E+04
XT(2) = .4520000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .16500000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .43000000E+03
XT(9) = .27170000E+00
XT(10) = .25500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 13

XT(2) = .4450000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 6
FUNCTION VALUE = .46799539E+07
COORDINATES
XT(1) = .389000000E+04
XT(2) = .4450000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 7
FUNCTION VALUE = .57948675E+07
COORDINATES
XT(1) = .389000000E+04
XT(2) = .4450000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .18000000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .42500000E+03
XT(9) = .26370000E+00
XT(10) = .26500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

VERTEX NUMBER 14
FUNCTION VALUE = .47675648E+07
COORDINATES
XT(1) = .39900000E+04
XT(2) = .4520000E+03
XT(3) = .32000000E+03
XT(4) = .25500000E+03
XT(5) = .16500000E+04
XT(6) = .90000000E+01
XT(7) = .40000000E+01
XT(8) = .43000000E+03
XT(9) = .27170000E+00
XT(10) = .25500000E+03
XT(11) = .62849000E-02
XT(12) = .75000000E+02
XT(13) = .50000000E+02
XT(14) = .15000000E+03

FUNCTION VALUE = .42092963E+07

COORDINATES

XT(1) = .40000000E+04

XT(2) = .4500000E+03

XT(3) = .32500000E+03

XT(4) = .26000000E+03

XT(5) = .17000000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .27170000E+00

XT(10) = .25500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 8

FUNCTION VALUE = .49540199E+07

COORDINATES

XT(1) = .40000000E+04

XT(2) = .4500000E+03

XT(3) = .32500000E+03

XT(4) = .26000000E+03

XT(5) = .17000000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .27170000E+00

XT(10) = .25500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

VERTEX NUMBER 15

FUNCTION VALUE = .45442755E+07

COORDINATES

XT(1) = .40000000E+04

XT(2) = .4500000E+03

XT(3) = .32500000E+03

XT(4) = .26000000E+03

XT(5) = .17000000E+04

XT(6) = .90000000E+01

XT(7) = .40000000E+01

XT(8) = .43000000E+03

XT(9) = .27170000E+00

XT(10) = .25500000E+03

XT(11) = .62849000E-02

XT(12) = .75000000E+02

XT(13) = .50000000E+02

XT(14) = .15000000E+03

OUTPUT SUMMARY FROM SUBROUTINE EVOP

MINIMUM OF THE OBJECTIVE FUNCTION HAS BEEN LOCATED TO THE DESIRED DEGREE OF ACCURACY FOR CONVERGENCE. IER = 8

TOTAL NUMBER OF OBJECTIVE FUNCTION EVALUATION.

NFUNC = 371

NUMBER OF TIMES THE SUBROUTINE FUNCTION IS CALLED DURING THE PRESENT CONVERGENCE TESTS. KUT = 6

NUMBER OF TIMES THE EXPLICIT CONSTRAINTS WERE EVALUATED

KKT = 3514

NUMBER OF TIMES THE IMPLICIT CONSTRAINTS WERE EVALUATED

M = 2186

COORDINATES OF THE MINIMUM

XT(1) = .40000000E+04	XT(8) = .43000000E+03
XT(2) = .4500000E+03	XT(9) = .29170000E+00
XT(3) = .32000000E+03	XT(10) = .25500000E+03
XT(4) = .26000000E+03	XT(11) = .62549000E-02
XT(5) = .17500000E+04	XT(12) = .75000000E+02
XT(6) = .90000000E+01	XT(13) = .50000000E+02
XT(7) = .40000000E+01	XT(14) = .15000000E+03

OBJECTIVE FUNCTION VALUE AT THE MINIMUM **F = .41296593E+07**

SECOND RESTART OF "EVOP" TO CHECK THE MINIMUM

OUTPUT SUMMARY FROM SUBROUTINE EVOP

MINIMUM OF THE OBJECTIVE FUNCTION HAS BEEN LOCATED TO THE DESIRED DEGREE OF ACCURACY FOR CONVERGENCE. IER = 8

TOTAL NUMBER OF OBJECTIVE FUNCTION EVALUATION.

NFUNC = 342

NUMBER OF TIMES THE SUBROUTINE FUNCTION IS CALLED DURING THE PRESENT CONVERGENCE TESTS. KUT = 6

NUMBER OF TIMES THE EXPLICIT CONSTRAINTS WERE EVALUATED

KKT = 4592

NUMBER OF TIMES THE IMPLICIT CONSTRAINTS WERE EVALUATED

M = 2810

COORDINATES OF THE MINIMUM

XT(1) = .40000000E+04	XT(8) = .43000000E+03
XT(2) = .4500000E+03	XT(9) = .27170000E+00
XT(3) = .32500000E+03	XT(10) = .25500000E+03
XT(4) = .26000000E+03	XT(11) = .62849000E-02
XT(5) = .17000000E+04	XT(12) = .75000000E+02
XT(6) = .90000000E+01	XT(13) = .50000000E+02
XT(7) = .40000000E+01	XT(14) = .15000000E+03

OBJECTIVE FUNCTION VALUE AT THE MINIMUM **F = .41137607E+07**

THIRD RESTART OF "EVOP" TO CHECK THE MINIMUM

OUTPUT SUMMARY FROM SUBROUTINE EVOP

MINIMUM OF THE OBJECTIVE FUNCTION HAS BEEN LOCATED TO THE DESIRED DEGREE OF ACCURACY FOR CONVERGENCE. IER = 8

TOTAL NUMBER OF OBJECTIVE FUNCTION EVALUATION.

NFUNC = 122

NUMBER OF TIMES THE SUBROUTINE FUNCTION IS CALLED DURING THE PRESENT CONVERGENCE TESTS. KUT = 6

NUMBER OF TIMES THE EXPLICIT CONSTRAINTS WERE EVALUATED

KKT = 2122

NUMBER OF TIMES THE IMPLICIT CONSTRAINTS WERE EVALUATED

M = 1209

COORDINATES OF THE MINIMUM

XT(1) = .40000000E+04	XT(8) = .43000000E+03
XT(2) = .4500000E+03	XT(9) = .27170000E+00
XT(3) = .32500000E+03	XT(10) = .25500000E+03
XT(4) = .26000000E+03	XT(11) = .62849000E-02
XT(5) = .17000000E+04	XT(12) = .75000000E+02

XT(6) = .90000000E+01	XT(13) = .50000000E+02
XT(7) = .40000000E+01	XT(14) = .15000000E+03

OBJECTIVE FUNCTION VALUE AT THE MINIMUM **F = .41883095E+07**

FOURTH RESTART OF "EVOP" TO CHECK THE MINIMUM

OUTPUT SUMMARY FROM SUBROUTINE EVOP

MINIMUM OF THE OBJECTIVE FUNCTION HAS BEEN LOCATED TO THE DESIRED DEGREE OF ACCURACY FOR CONVERGENCE. IER = 8

TOTAL NUMBER OF OBJECTIVE FUNCTION EVALUATION.

NFUNC = 32

NUMBER OF TIMES THE SUBROUTINE FUNCTION IS CALLED DURING THE PRESENT CONVERGENCE TESTS. KUT = 7

NUMBER OF TIMES THE EXPLICIT CONSTRAINTS WERE EVALUATED

KKT = 256

NUMBER OF TIMES THE IMPLICIT CONSTRAINTS WERE EVALUATED

M = 171

COORDINATES OF THE MINIMUM

XT(1) = .40000000E+04	XT(8) = .43000000E+03
XT(2) = .4500000E+03	XT(9) = .27170000E+00
XT(3) = .32500000E+03	XT(10) = .25500000E+03
XT(4) = .26000000E+03	XT(11) = .62849000E-02
XT(5) = .17000000E+04	XT(12) = .75000000E+02
XT(6) = .90000000E+01	XT(13) = .50000000E+02
XT(7) = .40000000E+01	XT(14) = .15000000E+03

OBJECTIVE FUNCTION VALUE AT THE MINIMUM **F = .41413647E+07**

Further restart of EVOP gives the same coordinates of the minimum as obtained in the fourth restart. So these coordinates of the minimum obtained in this restart are the optimum solutions and objective function value at the minimum is, **F = .46413647E+07.** The program is rerun using these optimum design variables as starting point with same values of control parameters and the minimum remains same.

So the global minimum is **F = .41137607E+07** and optimum value of the design variables are as follows:

Serial No.	Design variables	Design variables
1	$S = 4000;$	XT(1) = .40000000E+04
2	$TF_w = 450;$	XT(2) = .4500000E+03
3	$BF_w = 325;$	XT(3) = .32500000E+03
4	$BF_t = 260;$	XT(4) = .26000000E+03
5	$G_d = 1700;$	XT(5) = .17000000E+04
6	$N_S = 9.0;$	XT(6) = .90000000E+01
7	$N_T = 4.0;$	XT(7) = .40000000E+01
8	$y_1 = 430;$	XT(8) = .43000000E+03
9	$\eta = 0.27;$	XT(9) = .27170000E+00
10	$t = 255;$	XT(10) = .25500000E+03
11	$\rho = 0.006284;$	XT(11) = .62849000E-02
12	$TF_t = 75;$	XT(12) = .75000000E+02
13	$TFT_t = 50;$	XT(13) = .50000000E+02
14	$W_w = 150;$	XT(14) = .15000000E+03