

# **COMPUTER ANALYSIS & REINFORCED** **CONCRETE DESIGN OF BEAMS**

By  
**FADY R. S. ROSTOM**

## **ABSTRACT**

This project deals with the creation of a computer application that analyzes and designs structural beams. The project also aims at emphasizing the importance of computers in the solution of everyday engineering problems.

The program developed analyses one, two and three-span beams and includes a module for the design of reinforced concrete beams. This program was created using the relatively new Actionscript language.

The project also discusses various theoretical analysis techniques that can be implemented in developing a computer program. The main theoretical methods used in this project are Moment Distribution and Macaulay's Method. The Reinforced concrete design is based on the BS8110 code.

This report acts as a support document for the created software. It describes the program in detail and highlights the methodologies used in its development.

## **CONTENTS**

Acknowledgements	3
Abstract	4
Contents	5
<b>CH.1: INTRODUCTION</b>	<b>8</b>
1.1 Computer Application in the Civil & Structural Engineering Industry	8
1.1.1 Structural Analysis & Design Software	9
1.2 Scope & Aims of Project	10
1.3 Project Overview	11
<b>CH.2: LITERATURE REVIEW</b>	<b>12</b>
2.1 Programming Language Review	13
2.2.1 Basic Elements of Actionscript	13
2.2 Analytical Theories Review	16
2.2.1 Macaulay's Method	16
2.2.2 Moment Area Method	19
2.2.3 Conjugate Beam Method	22
2.2.4 Virtual Work Method	23
2.2.5 The Unit Load Method	24
2.2.6 Influence Line Theory	25

2.2.7	The Three Moment Equation (Clapeyron's Theorem)	27
2.2.8	Stiffness & Flexibility Methods	29
2.2.9	Slope Deflection Method	32
2.2.10	Moment Distribution Method	35
2.3	Reinforced Concrete Beam Design Review	42
2.3.1	Composite Action	42
2.3.2	Limit State Design	44
2.3.3	Bending & the Equivalent Stress Block	45
2.3.4	Rectangular Section with Compression Reinforcement at the Ultimate Limit state	48
<b>CH. 3: PROGRAM REVIEW &amp; APPLICATION</b>		<b>52</b>
3.1	Single Span Beams	53
3.2	Two Span Beam Analysis	59
3.3	Three Span Beam Analysis	65
3.4	Reinforced Concrete Beam Design	72
<b>CH. 4: DISCUSSION</b>		<b>80</b>
4.1	Single Span Beam Analysis	81
4.2	Two & Three Span Load Swap Modules	85
4.3	Two Span Beam Analysis	92
4.4	Three Span Beam Analysis	99
4.5	Reinforced Concrete Beam Design Module	113

<b>4.6</b>	Program Limitations	122
<b>4.6</b>	General Program Discussions	123
<b>CH. 5: CONCLUSION &amp; RECOMMENDATIONS</b>		<b>125</b>
<b>5.1</b>	Conclusion	125
<b>5.2</b>	Recommendations	128
<b>CH. 6: SELECTED BIBLIOGRAPHY</b>		<b>129</b>
<b>APPENDICES</b>		
	Appendix A: Code Printouts for the Main Program	131
	Appendix B: Code Printouts for the Profile Plotting Module	252

## **CHAPTER 1: INTRODUCTION**

### **1.1 Computer Application in the Civil & Structural Engineering Industry**

Civil engineers design and construct major structures and facilities that are essential in our every day lives. Civil engineering is perhaps the broadest of the engineering fields, for it deals with the creation, improvement and protection of the communal environment, providing facilities for living, industry and transportation, including large buildings, roads, bridges, canals, railroad lines, airports, water-supply systems, dams, irrigation, harbors, docks, tunnels, and other engineered constructions. Over the course of history, civil engineers have made significant contributions and improvements to the environment and the world we live in today.

The work of a civil engineer requires a lot of precision. This is mainly due to the fact that the final result of any project will directly or indirectly affect people's lives; hence safety becomes a critical issue. Designing structures and developing new facilities may take up to several months to complete. The volumes of work, as well as the seriousness of the issues considered in project planning, contribute to the amount of time required to complete the development of an adequate, safe and efficient design.

The introduction of software usage in the civil engineering industry has greatly reduced the complexities of different aspects in the analysis and design of projects, as well as reducing the amount of time necessary to complete the designs. Concurrently, this leads to greater savings and reductions in costs. More complex projects that were almost impossible to work out several years ago are now easily solved with the use of computers. In order to stay at the pinnacle of any industry, one needs to keep at par with the latest technological advancements which accelerate work timeframes and accuracy without decreasing the reliability and efficiency of the results.

### **1.1.1 Structural Analysis & Design Software:**

Currently, there are quite a number of structural analysis and design software applications present in the market. Although they are rather expensive, their use has become prevalent amongst a majority of structural engineers and engineering firms.

A majority of these applications are based on the Finite-Element method of analysis. This method facilitates computations in a wide range of physical problems including heat transfer, seepage, flow of fluids, and electrical & magnetic potential.

In the finite-element method, a continuum is idealized as an assemblage of finite elements with specified nodes. In essence, the analysis of a structure by the finite-element method is an application of the displacement/stiffness method. The use of a computer in the finite-element approach is essential because of the large number of degrees of freedom commonly involved. The computerized computations make use of the systematic sequences executed in a computer program as well as the high processing speeds.

Some common Structural Analysis & Design Software available in the market:

- **STADD III:**  
Comprehensive structural software that addresses all aspects of structural engineering- model development, analysis, design, visualization and verification.
- **AXIS VM:** (<http://www.axisvm.com>)  
Structural analysis and design with an updateable database of element sections and specifications available in the market.
- **ANSYS:** (<http://www.ansys.com>)  
All-inclusive engineering software dealing with structural analysis and other engineering disciplines such as fluid dynamics, electronics and magnetism and heat transfer
- **ETABS:**  
Offers a sophisticated 3-D analysis and design for multistory building structures.

## **1.2 Scope & Aims of Project**

The main aim of this project is to create a computer application for the analysis and design of reinforced concrete beams. The program is intended to be designed in such a way that the users will be guided through the analysis and design stages in a straightforward and understandable manner. The software is intended for use by civil/structural engineering students but is also quite appropriate for use by professional structural engineers. Unlike a majority of the current engineering software applications, it is aimed to develop the software in such a manner that is very user-friendly and easy to follow without having to memorize syntax commands or read a user manual.

The project also aims at establishing a relationship between theoretical structural analysis procedures and possible methods of correlating and implementing these concepts in a practical computer program.

### Personal Objectives:

- To develop an in-depth appreciation of theoretical concepts used in structural analysis.
- To learn the process of systematically creating and developing engineering software applications.
- To create a project that has continuity, i.e. one that can be worked on and improved by students and other users while being put to good use, not merely shelved away.

### Specific Program Scope:

- Analysis of Single Span Beams for Shear, Moment and Deflection values at every point on the beam span.
- Analysis of 2-Span and 3Span Beams; yielding support and midspan moments along the beam length.
- Design of Reinforced Concrete Beams; offers a recommended beam sizing and calculates the areas of tension and compression steel required.

### **1.3 Project Overview**

This section gives a guide on the main issues covered in the succeeding chapters of this report.

#### Chap. 2: Literature Review

This section offers a brief review on the following:

- Programming Language:  
Introduces Actionscript as the programming language of the Macromedia Flash Software. Explains what the language is all about and gives a brief description on the fundamentals of the Actionscript language.
- Analytical Theory:  
Brief explanations on the major structural analysis theories applicable in beam analysis with main emphasis on the theories used in this project, namely: Macaulay's Method & Moment Distribution.
- Reinforced Concrete Beam Design:  
An introduction to reinforced concrete design concepts. Also includes a summary of the process of design, with the applicable formulae derived from first principles. The applicable and relevant points extracted from the BS8110 code that were used in this project are also mentioned here.

### Chap. 3: Program Review & Application

This section summarizes the individual steps of the program. It explains each step in the Analysis modules as well as the RC Design module by including individual snapshots of the screen with instructions and information regarding that section. It is more or less like a guided tour on the use of the software with explanations on what happens at every stage and in the programming background after every command.

### Chap. 4: Discussion

This section displays the code written in the program for the single, double and triple span beam analyses as well as the code for the RC Design module. Every few lines of the code are explained in detail. Thus, the code sections become clear, even if the reader is not too familiar with the Actionscript Syntax. A General Discussion of the Program is also found in this section.

### Chap. 5: Conclusion & Recommendation

The project's concluding statements are found in this section. Program and general recommendations are also included here.

### Chap. 6: References & Bibliography

A List of all the text books and sources of information used in this project.

### Appendices

Printouts of all the code developed for this software.

## **CHAPTER 2: LITERATURE REVIEW**

### **2.1 PROGRAMMING LANGUAGE REVIEW**

Programming languages are used to send information to and receive information from computers. Hence, programming may be viewed as communicating with a computer using representative vocabulary and grammar. A program may be defined as a collection of code, that when properly executed, performs a required task.

“Actionscript” is the back-end programming language of Macromedia’s Flash Software. Flash is a relatively new software application. It was mainly created to enable the development of on-line animations and internet applications. However, the rapid growth and development of Actionscript has enabled the widespread use of this software in developing almost any software application.

Like almost any other “new age” programming language, Actionscript involves the use of variables, operators, statements, conditionals, loops, functions, objects & arrays.

A combination of good use of Flash and good programming in Actionscript allows an artistic application to be created, whether visually appealing or dynamically interactive. Actionscript also has the distinct advantage of being easily understood, even to non-programmers, due to it’s, more or less, use of English statements.

#### **2.1.1 Basic Elements of Actionscript**

##### Variables:

An individual piece of data is known as a datum. A datum and the label that defines it are together known as a variable. A variable’s label is called its name, and a variable’s datum is called its value. We say that the variable stores or contains its value. For this reason, one may conveniently think of a variable as a container, whether anything is in that container or not.

e.g.    BeamLength = 5m ;

Here, the variable name (container) is “BeamLength”, and its value is 5m.

### Arguments:

This is basically a datum sent to a command (also called parameters). Supplying an argument to a command is known as passing the argument. In common Actionscript syntax, arguments are usually enclosed within parentheses.

e.g.    command (argument);

### Operators:

All operators link phrases of code together, manipulating those phrases in the process. Whether the phrases are text, numbers or some other datatype, an operator nearly always performs some kind of transformation. Very commonly, operators combine two things together, as the plus operator does

e.g.    trace ( 5 + 2 )

### Expressions:

In a program, any phrase of code that yields a single datum when a program runs is referred to as an expression. They represent simple data that will be used when the program runs. Expressions get even more interesting when combined with operators. The expression 4 + 5 for example, is an expression with two operands, 4 and 5, but the plus operator makes the entire expression yield the single value 9. An expression may even be assigned to a variable.

e.g.    Moment = 45 + 67

### Conditionals and Loops:

In nearly all programs, conditionals are used to add logic to the program, and loops to perform repetitive tasks. Conditionals allow a specification of terms under which a section of code should – or should not – be executed. To perform highly repetitive tasks, a loop is used. This is a statement that allows a block of code to be repeated an arbitrary number of times.

e.g.    While ( distance < min ) {  
                distance = distance + 1  
          }

**Functions:**

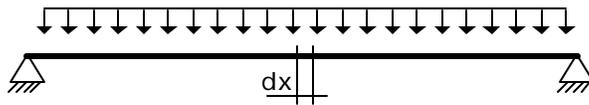
A function is a packaged series of statements. In practice, functions mostly serve as reusable blocks of code. It allows a clear way of managing code, especially when it becomes too large & cumbersome. After a function is created, the code it contains may be run from anywhere in the program by using its name.

## 2.2 ANALYTICAL THEORIES REVIEW

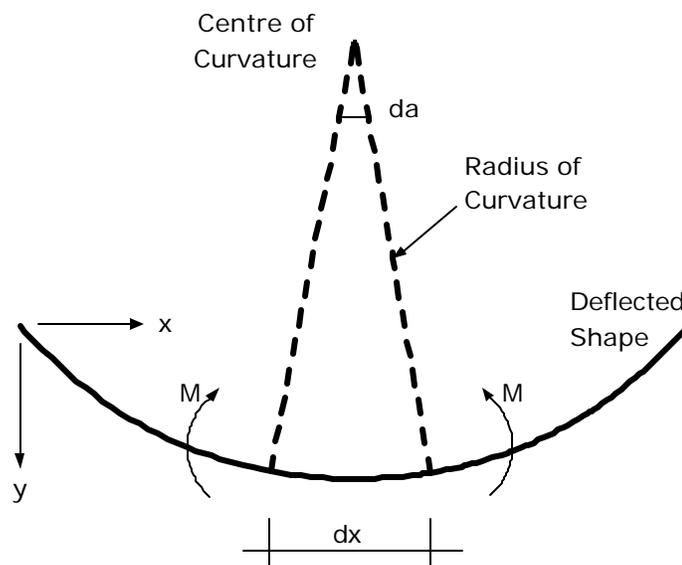
### 2.2.1 Macaulay's Method

This is a method suggested by W. H. Macaulay to relate the stiffness, radius of curvature, deflection and the bending moments in a beam by integration methods. The method enables discontinuous bending moment functions to be represented by a continuous function. It allows the contributions, from individual loads, to the bending moment at any cross section to be expressed as a single function, which takes zero value at those sections where particular loads don't contribute to the bending moment.

#### Beam Deflections using successive integration



Consider an infinitely small Section,  $dx$ , of the above loaded beam;



The bending moment (M) at section X is given by:

$$M = \frac{EI}{R}$$

where R = Radius of Curvature

I = Second Moment of Area

E = Young's Modulus of Elasticity

More exactly, positive (sagging) bending moment produces negative curvature,  $1/R$

i.e.

$$EI \frac{d^2 y}{dx^2} = -M$$

Nb.

$$Curvature = \frac{d}{dx} \left( \frac{dy}{dx} \right) = \frac{d^2 y}{dx^2} = \frac{1}{R}$$

where y = deflection at section X (measured positive downward)

To obtain the equation of the deflected shape, the bending moment expression (a function of x) is integrated twice with respect to x. The constants of integration formed are then evaluated from the boundary conditions.

Hence the differential equation of an elastic curve may be given as:

$$\boxed{\frac{d^2 y}{dx^2} = -\frac{M}{EI}}$$

Macaulay's method enables discontinuous bending moment functions to be represented by a continuous function, thus avoiding the need to deal with the beam section by section between discontinuities in the bending moment function. This is very desirable since it avoids the need to evaluate, and therefore eliminate, a large number of constants of integration.

Essentially, the method employs the use of a step function, allowing the individual loads to contribute to the bending moment.

In this method, the principle of superposition applies in all cases that involve several concentrated loads or discontinuous UDLs.

There are certain steps & rules that need to be followed in the analysis of a beam using Macaulay's method. These can be summarized as follows:

- An origin is selected at one end of the beam.
- The bending moment is written down for a section in the portion of the beam furthest from the origin taking the FBD (free body diagram) which includes the origin.
- The individual load contributions are grouped as bracket terms. (Nb. when the quantity within the bracket is negative, then the total value of the bracket shall be zero).
- It is essential that the bending moment at each & every section in the beam is expressed in such a way that the bracket concept can be maintained throughout the length of the beam and throughout the integration process. i.e. integrate expressions such as  $[z-a]$ , which only occur when positive, as  $[\frac{1}{2}(z-a)^2]$ .

In other words, bracket terms remain within the brackets throughout the integration process.

### 2.2.2 Moment Area Method

This is a method suitable for calculating slope & deflection at selected points on a beam. It is also effective for calculating the deflections of beams with various cross sections. The simplest way to evaluate the fixed-end moments, etc, will often be by the use of the moment area method.

There are two theorems associated with the moment area methods:

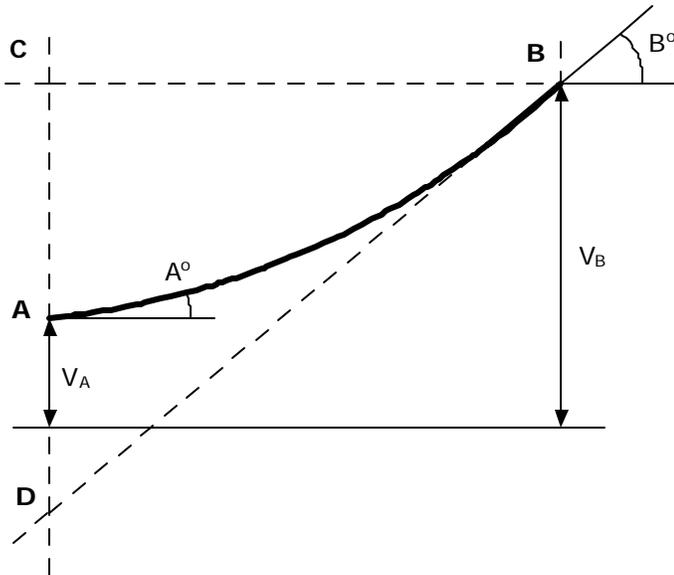
- First Moment Area Theorem:  
“the difference in slope between two points on a beam is equal to the area of the  $M/EI$  diagram between the two points.”
- Second Moment Area Theorem:  
“the moment about a point A of the  $M/EI$  diagram between points A and B will give the deflection of point A relative to the tangent at point B.”

To obtain the  $M/EI$  diagram, each ordinate of the bending-moment diagram is divided by the corresponding value of the beam flexural rigidity ( $EI$ ) at the ordinate.

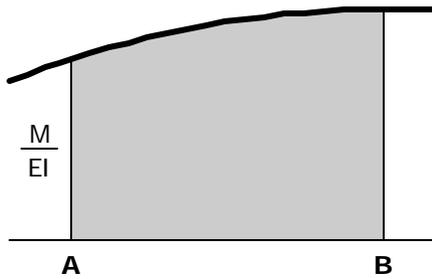
The above theorems follow directly from graphical interpretation of the successive integration technique and are exceptionally useful and easy to apply in several types of deflection problems and in deriving other results from the analysis of indeterminate structures.

Nb. this method is not applicable if there is a hinge (moment release) within the beam region being considered.

Consider a section of an elastic curve between points A & B:



$$M = EI \frac{d^2 y}{dx^2}$$



$$\frac{M}{EI} = \frac{d^2 y}{dx^2}$$

$$\int_A^B \frac{M}{EI} dx = \int_A^B \frac{d^2 y}{dx^2} dx = \left[ \frac{dy}{dx} \right]_A^B = B^o - A^o$$

... [1<sup>st</sup> Theorem]

$$\begin{aligned}\int_A^B \frac{Mx dx}{EI} &= \int_A^B \frac{d^2 y}{dx^2} x dx = \left[ x \frac{dy}{dx} \right]_A^B = \int_A^B \frac{dy}{dx} dx \\ &= x_B \left( \frac{dy}{dx} \right)_B - x_A \left( \frac{dy}{dx} \right)_A - y_B + y_A\end{aligned}$$

If the origin is now shifted until it is below A;

$$\int_A^B \frac{Mx dx}{EI} = x_B \theta_B - y_B + y_A \quad \dots [2^{\text{nd}} \text{ Theorem}]$$

where  $x = 0$  at A,  $x_B \theta_B$  is represented by CD in the elastic curve figure, and the complete expression is equal to the distance AD.

The procedure for beam analysis using the moment area method can be summarized as follows:

- calculate the support reactions
- draw the M/EI diagram
- select the reference tangent; either:
  - a known point with zero slope
  - determining tangential deviation of one support w.r.t. the other & finding the angle.

### **2.2.3 Conjugate Beam Method**

The conjugate beam may also be referred to as a fictitious/imaginary beam. This conjugate beam has the same length as the real beam but is supported and detailed in such a manner that when the conjugate beam is loaded by the  $M/EI$  diagram of the real beam as an elastic load, the elastic shear in the conjugate beam at any location is equal to the slope of the real beam at the corresponding location and the elastic bending moment in the conjugate beam is equal to the corresponding deflection of the real beam. These slopes and deflections of the real beam are measured with respect to its original position.

Two conjugate beam relations are recognized:

- The shear force  $V$ , in value & sign, at any point on the conjugate beam, is equal to the rotation slope  $\theta$ , at that point on the actual beam
- The moment  $M$ , in value & sign, at any point on the conjugate beam is equal to the deflection at that point on the actual beam.

Statically determinate real beams always have corresponding conjugate beams. However, such conjugate beams turn out to be in equilibrium since they are stabilized by the elastic loading corresponding to the  $M/EI$  diagram for the corresponding real beam.

### 2.2.4 Virtual Work Method

The general mathematical results concerning the virtual work done by an equilibrium system of forces moving through small virtual (imaginary) displacements are of great use in obtaining many structural analysis results. In particular, the principle of virtual work enables equilibrium equations to be written down very simply and is also useful in obtaining displacements of beams, frames and trusses.

The work done by external forces moving through small displacements compatible with the geometry of the structure is called external virtual work.

There are several principles involved in the virtual work method:

- Principle of Virtual Displacements:  
If a set of external forces acting on a structure are in equilibrium, then any virtual (imaginary) rigid-body displacements given to the system causes virtual work to be done by each force, and the total external virtual work is zero.
- Principle of Virtual Work:  
If any set of virtual (imaginary) displacements given to a body in equilibrium (these displacements being small and compatible with the geometry of the body and its supports), then the total external virtual work done by the external forces moving through the virtual displacements is equal to the total internal work done by the internal forces moving through corresponding virtual displacements.

### 2.2.5 The Unit Load Method

The unit load method considers the product of imaginary (dummy) loads and real displacements rather than considering the product of real loads & virtual displacements.

To determine the deflection of a beam, a unit load is applied at the point where deflection is to be determined.

The deflection of an elastic beam may be given as:

$$\Delta = \int_0^l \frac{Mm}{EI} dz$$

where  $M$  = moment due to external/applied loads

$m$  = applied unit moment

### **2.2.6 Influence Line Theory**

An influence line is a graphical representation of the value taken by an effect as a load moves along a structure. It is a curve, the ordinate to which, at any point, equals the value of the particular function for which the line has been constructed.

This is a function whose value at any given point represents the value of some structural quantity due to a unit force placed at that point. The influence line graphically shows how changing the position of a single load influences various significant structural quantities. (Structural quantities: Reactions, Shear, Moment, Deflection, etc.)

Influence lines may be used to advantage in the determination of simple beam reactions. In this case, the use of the unit influence line is necessary. The unit influence line represents the effects of unit: reactions (displacements), shears (separations) and moments (rotations) in a beam structure.

Influence lines can be used for two very important purposes;

- To determine what position of loading will lead to a maximum value of the particular function for which the influence line has been constructed. This is especially important for the design of members in structures that will be subjected to live loads (which vary in position and intensity)
- To compare the value of that function, for which the influence line has been constructed, with the loads placed for maximum effects, or for any loading combination.

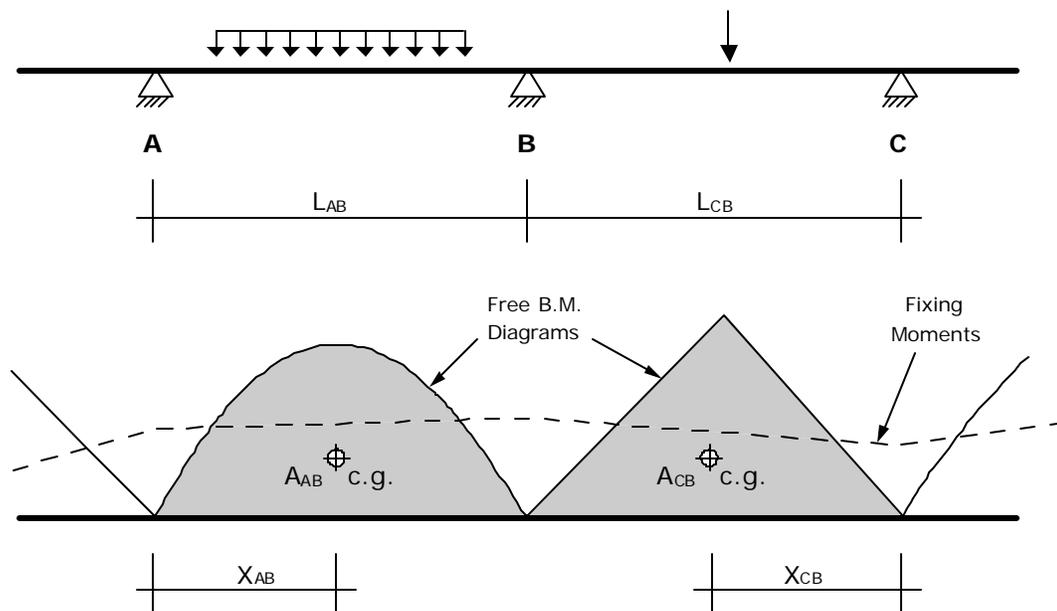
Since the ordinate to an influence line equals the value of a particular function due to a unit load acting at that point where the ordinate is measured, the following theorems hold:

- To obtain the maximum value of a function due to a single concentrated live load, the load should be placed at that point where the ordinate to the influence line for that function is a maximum.
- The value of a function due to the action of a single concentrated live load equals the product of the magnitude of the load and the ordinate to the influence line for that function, measured at the point of application of load.

### 2.2.7 The Three Moment Equation (Clapeyron's Theorem)

The three moment equation was first presented in 1857 by the French Engineer Clapeyron. This equation is a relationship that exists between the moments at three points on a continuous member. It is particularly helpful in solving for the moments at the supports of indeterminate beams. The three moment equation is applicable to any three points on a beam as long as there are no discontinuities, such as hinges, in the beam within this portion.

Consider three support points, A, B & C with  $L_{AB}$  and  $L_{BC}$  (distances),  $I_{AB}$  and  $I_{BC}$  (stiffnesses) between supports A & B and B & C respectively.



$M_{AB}, M_{BA}, M_{CB}$  = moments in statically indeterminate beam at points A, B, and C, respectively

$L_{AB}, L_{CB}$  = lengths of spans AB and BC

$I_{AB}, I_{BC}$  = moments of inertia of beam cross section between A & B and between C & B

$A_{AB}, A_{CB}$  = areas of moment diagrams, considering sections of beam between supports to be simply supported, between A & B and between C & B

$X_{AB}, X_{CB}$  = distance from A and C, respectively, to the centroids of areas  $A_{AB}$  and  $A_{CB}$

$\Delta_{AB}, \Delta_{CB}$  = deflection of A and C above B

$E$  = modulus of elasticity of beam material

It follows from direct application of the Second Moment Area Theorem that  $L_{AB}\theta_{BA}$  and  $L_{BC}\theta_{BA}$  can be written down in terms of the above parameters.

Hence, two equations can be written down for the quantity  $\theta_{BA}$ .

Equating the two results gives one equation linking the unknown support moments  $M_{AB}$ ,  $M_{BA}$  and  $M_{CB}$  in terms of the other (known) parameters:

$$M_{AB} \frac{L_{AB}}{I_{AB}} + 2M_{BA} \left( \frac{L_{AB}}{I_{AB}} + \frac{L_{CB}}{I_{CB}} \right) + M_{CB} \frac{L_{CB}}{I_{CB}} = - \frac{6A_{AB} \bar{X}_{AB}}{I_{AB} L_{AB}} - \frac{6A_{BC} \bar{X}_{BC}}{I_{BC} L_{BC}} + 6E \left( \frac{\Delta_{AB}}{L_{AB}} + \frac{\Delta_{BC}}{L_{BC}} \right)$$

This is the general statement of the three-moment equation which, though cumbersome in appearance when expressed generally, is particularly easy to apply to individual problems, especially when  $\Delta_1 = \Delta_2 = 0$

### **2.2.8 Stiffness & Flexibility Methods**

#### **Stiffness Method (Displacement Method of Analysis)**

The displacement method can be applied to statically determinate or indeterminate structures, but is more useful in the latter, particularly when the degree of statical indeterminacy is high.

In this method, one must first determine the degree of kinematic indeterminacy. A coordinate system is then established to identify the location and direction of joint displacements. Restraining forces equal in number to the degree of kinematic indeterminacy are introduced at the co-ordinates to prevent the displacement of the joints. The restraining forces are finally determined as a sum of the fixed end forces for the members meeting at a joint. (For most practical cases, the fixed-end force can be calculated with the aid of standard tables)

#### **Stiffness Matrix [S]**

$$\{D\} = [S]^{-1} \{-F\}$$

The elements of the vector  $\{D\}$  are the unknown displacements.

The elements of the matrix  $[S]$  are forces corresponding to unit values of displacements.

The column vector  $\{F\}$  depends on the loading on the structure

In general cases, the number of restraints introduced in the structure is  $n$ , the order of the matrices  $\{D\}$ ,  $[S]$  and  $\{F\}$  is  $n \times 1$ ,  $n \times n$  and  $n \times 1$  respectively.

The general steps followed in an analysis using the stiffness method are as follows:

- establish a relationship between the element forces and displacements (e.g. between moments and rotations, forces and deflections)
- Reassemble the elements to form original structure & apply compatibility to the joints.
- Apply equilibrium on the assembled structure at each joint.

### Flexibility Method (Force Method of Analysis)

In this method, the degree of statical indeterminacy is initially determined. Thereafter, a number of releases equal to the degree of statical indeterminacy is introduced, each release being made by the removal of an external or internal force. The magnitude of inconsistencies introduced by the releases is determined. Next, the displacements in the released structure due to unit values of the redundants are determined. This allows the values of the redundant forces necessary to eliminate the inconsistencies in the displacements to be determined. Hence, the forces on the original indeterminate structure are calculated as the sum of the correction forces (redundants) and forces on the released structure.

#### Flexibility Matrix $[f]$

$$[f]\{F\} = \{\Delta - D\}$$

$D$  represents inconsistencies in deformation while  $\{F\}$  represents the redundants.  $\Delta$  elements represent prescribed displacements at their respective coordinates.

The column vector  $\{\Delta - D\}$  thus depends on the external loading.

The elements of the matrix  $[f]$  are displacements due to the unit values of the redundants. Therefore  $[f]$  depends on the properties of the structure, and represents the flexibility of the released structure. For this reason,  $[f]$  is called the flexibility matrix and its elements are called flexibility coefficients.

The general steps followed in an analysis using the flexibility method are as follows:

- The structure is rendered indeterminate by the insertion of suitable releases, and is now called the primary structure (e.g. insert three releases for a degree of redundancy of three)
- By inserting a release, a condition of compatibility at that point is abandoned. Since the primary structure is now statically determinate, a

solution is carried out and the member forces are calculated by applying equilibrium conditions only.

- Release forces are introduced in the structure so as to restore conditions of compatibility at the releases. A complementary solution of the secondary structure is now carried out. Here, the displacements at the releases due to the release forces only are calculated.
- Next, the solutions of the primary structure and the complementary solution are combined to give the total displacement at the releases due to both the applied loads and the release forces. Finally, the member forces in the original structure may be obtained by the superposition effects from the particular and complementary solutions.

### Choice of Force or Displacement Method

In some structures, the formation of one of the matrices – stiffness or flexibility – may be easier than the formation of the other. This situation arises from the following general considerations.

In the force method, the choice of the released structure may affect the amount of calculation. For example, in the analysis of a continuous beam, the introduction of hinges above indeterminate supports produces a released structure consisting of a series of simple beams. In other structures, it may not be possible to find a released structure for which the redundants have a local effect only.

In the displacement method, generally all joint displacements are prevented regardless of the choice of the unknown displacement. A displacement of a joint affects only the members meeting at the given joint. These properties generally make the displacement method easy to formulate, and it is for this reason that the displacement method is more suitable for computer programming.

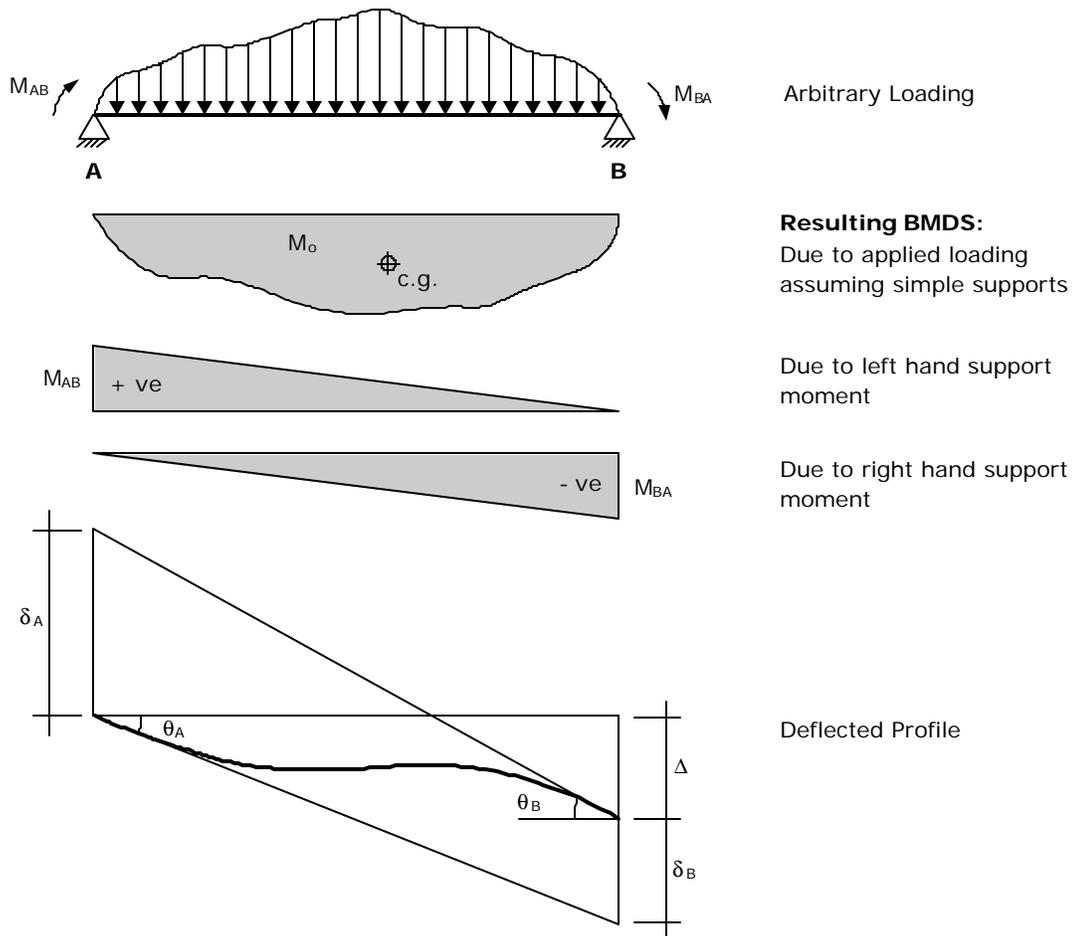
### **2.2.9 Slope Deflection Method**

The slope deflection method was presented by Prof. G. A. Maney in 1915 as a general method to be used in the analysis of rigid-jointed structures. The slope deflection method may be used to analyze all types of statically indeterminate beams or rigid frames. In this method, all joints are considered rigid. i.e. the angles between members at the joints are considered not to change in value as the loads are applied.

Thus, the joints at the interior supports of statically indeterminate beams can be considered as  $180^\circ$  rigid joints.

The fundamental slope deflection equations are derived by means of the moment-area theorems. These equations consider deformation caused by bending moment but neglect that due to shear and axial force.

Basically, a number of simultaneous equations are formed with the unknowns taken as the angular rotations and displacements of each joint. Once these equations have been solved, the moments at all joints may be determined.



The slope deflection equations may be written as:

$$M_{AB} = \frac{2EI}{L} \left( 2q_A + q_B - \frac{3d}{L} \right) + FEM_A$$

$$M_{BA} = \frac{2EI}{L} \left( 2q_B + q_A - \frac{3d}{L} \right) + FEM_B$$

where  $\theta$  = rotation of the tangent to the elastic curve at the end of a member

$\delta$  = rotation of the chord joining the ends of the elastic curve.

FEM = fixed end moments

The fundamental slope deflection equation is written as:

$$M_{PQ} = 2EK_{PQ} \left( 2\mathbf{q}_P + \mathbf{q}_Q - \frac{3\mathbf{d}}{l} \right) + FEM_{PQ}$$

where the stiffness factor,  $K_{PQ} = \frac{I_{PQ}}{L_{PQ}}$

This fundamental slope deflection equation is an expression for the moment on the end of a member in terms of four quantities, namely:

- The rotation of the tangent at each end of the elastic curve of a member
- The rotation of the chord joining the ends of the elastic curve
- The external loads applied to the member

### **2.2.10 The Moment Distribution Method**

The moment distribution method was first introduced by Prof. Hardy Cross in 1932, and is without doubt, one of the most important contributions to structural analysis in the twentieth century. It is an ingenious & convenient method of handling the stress analysis of rigid jointed structures.

The method of moment distribution usually does not involve as many simultaneous equations and is often much shorter than any of the methods of analysis of indeterminate beams previously discussed.

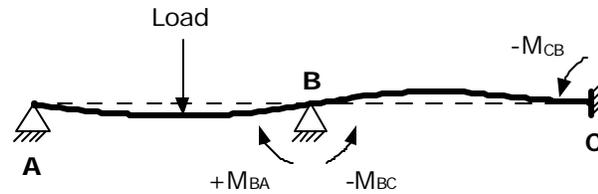
Essentially, Moment Distribution is a mechanical process dealing with indeterminate structures by means of successive approximations in which the moments themselves are treated directly, and the calculations involved being purely arithmetic.

It is basically a numerical technique which enables successive approximations to the final set of moments carried by a rigid-jointed structure to be made by a systematic “locking” and “relaxing” of the joints of the structural element(s). It has the advantage of being simply interpreted physically and of yielding solutions to any required degree of accuracy.

The method is unique in that all joints are initially considered to be fixed against rotation. The fixed end moments are determined for each member as though it were an encastred beam and then the joints are allowed to rotate, either separately or all at once, the moments induced by the rotations being distributed among the members until the algebraic sum of the moments at each internal joint is zero.

The sign convention most commonly adopted for Moment Distribution is that all moments acting on individual members from supports or other members of a structure are positive clockwise in application and negative if anti-clockwise.

Consider the following beam:



The three fundamental principles of Moment Distribution applicable to continuous beams on unyielding supports are listed as follows:

- Principle 1:

When a moment is applied at one end of a prismatic beam, that end remaining fixed in position but not in direction (pinned support), the other end being fixed both in position and direction (fixed support), a moment of half the amount and the same sign is induced at the second (fixed) end.

i.e. 
$$M_{AB} = \frac{1}{2} M_{BA}$$

- Principle 2:

When one end of a beam remains fixed in position and direction, the moment required to produce a rotation of a given angle at the other end of the beam, which remains fixed in position, is proportional to the value of  $I/L$  for the beam, provided that  $E$  is constant. The value  $I/L$  (known by the symbol,  $K$ ) is the stiffness factor for the particular beam in question.

$$M_{BA} = 4E \tan \mathbf{q} \frac{I}{L}$$

i.e.

$$= 4E \mathbf{q} \frac{I}{L} \quad \text{.....for small values of } \mathbf{q}$$

- Principle 3:

When one end of a beam is rotated through a given angle, remaining fixed in position, and the other end remains fixed in position but not in direction, the moment required at the first end is  $\frac{3}{4}$  of that required if the second end was fixed both in position and direction, i.e. the equivalent stiffness factor for the beam is  $\frac{3}{4}I/L = \frac{3}{4}K$

i.e. 
$$M_{BA} = 3E\mathbf{q} \frac{I}{L} \quad \dots\dots\text{for small values of } \mathbf{q}$$

The three foregoing principles alone are applied when the supports do not yield. Hence, the previous section applies solely to structures in which the only possible displacement at the joints is rotation.

The steps of the moment distribution process are summarized as follows:

- Step 1

Determine the internal joints which will rotate when the external load is applied to the frame.

Calculate the relative rotational stiffnesses of the ends of the members meeting at these joints, as well as the carry over factors from the joints to the far ends of these members.

Determine the distribution factors using the following equation:

$$(DF)_i = \frac{S_i}{\sum_{j=1}^n S_j}$$

where  $i$  refers to the near end of the member considered

$n$  = members meeting at the joint

$S$  = Stiffnesses of the beam span being considered  $S = \frac{I}{L}$

The rotational stiffness of either end of a prismatic member is  $4EI/L$  and the COF (carry over factor) from either end to the other is  $1/2$ . If one end of a prismatic member is hinged, the rotational end stiffnesses of the other end is  $3EI/L$ , and of course, no moment is carried over to the hinged end.

In a scenario where all the members are prismatic, the relative rotational end stiffnesses can be taken as  $K = I/L$ ; and when one end is hinged, the rotational stiffnesses at the other end is  $3/4(K) = 3/4(I/L)$

- Step 2:

With all joint rotations restrained, determine the fixed-end moments due to the lateral loading on all the members.

- Step 3:

Select the joints to be released in the first cycle. It may be convenient to select alternate internal joints in the case of a framed structure.

Calculate the balancing moment at the selected joints; this is equal to minus the algebraic sum of the fixed-end moments. If an external clockwise couple acts at any joint, its value is simply added to the balancing moment.

- Step 4:

Distribute the balancing moments to the ends of the members meeting at the released joints. The distributed moment is equal to the DF (distribution factor) multiplied by the balancing moment. The distributed moments are then multiplied by the COFs to give the carry over moments at the far ends. Thus the first cycle is terminated.

- Step 5:

Release the remaining internal joints, while further rotation is prevented at the joints released in the first cycle. The balancing moment at any joint is equal to minus the algebraic sum of FEMs and of the end-moments carried over the first cycle. The balancing moments are distributed and moments are carried over to the far ends in the same way as in *Step 3*. This completes the second cycle.

- Step 6:  
The joints released in *Step 3* are released again, while the rotation of the other joints is prevented. The balancing moment at a joint is equal to minus the algebraic sum of the end-moments carried over to the ends meeting at the joint in the previous cycle.
- Step 7:  
Repeat *Step 6* several times, for the two sets of joints in turn until the balancing moments become negligible.
- Step 8:  
Sum the end moments recorded in each of the *Steps 2 to 7* to obtain the final end-moments. The mid-span moments may then be calculated separately, depending on the type of loading within the span being considered. The Law of Superposition holds good.

#### **Various Moment Definitions:**

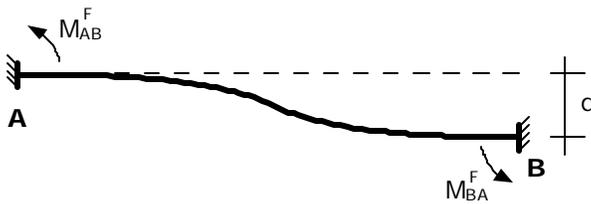
- Fixed End Moments – these are end moments developed when loads are applied to the structure with all joints locked against rotation.
- Unbalanced Moment – when a joint is unlocked, it will rotate if the algebraic sum of all the FEMs acting the joint does not add up to zero. This resultant moment acting on the joint is therefore called the unbalanced moment (or out-of-balance moment)
- Distributed Moments – when the unlocked joint rotates under this unbalanced moment, end moments are developed in the ends of the members meeting at the joint. These finally restore equilibrium at the joint and are called distributed moments.
- Carry Over Moments – As the joint rotated, and bent these members, end moments were likewise developed at the far ends of each. These are called carry-over moments.

So far, the theory and methodology considered only caters for conditions where the supports do not yield. i.e. it applies solely to structures in which the only possible displacement at the joints is rotation.

However, some rare scenarios do occur when other displacements contribute to the stresses and hence moments in the beam. These are:

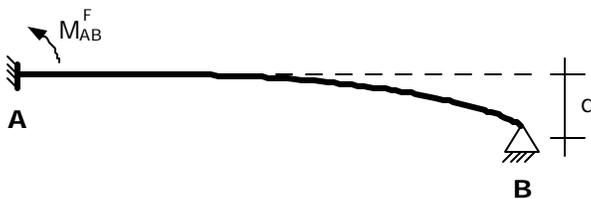
### Translational Yield

For a beam with fixed ends:



$$M_{AB}^F = M_{BA}^F = \frac{-6EId}{l^2}$$

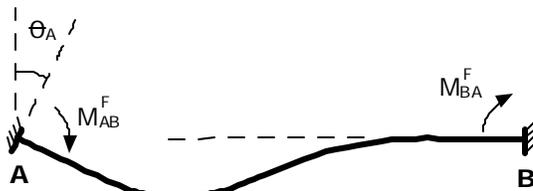
For a beam with a pinned end:



$$M_{AB}^F = \frac{-3EId}{l^2} \quad ; \quad M_{BA}^F = 0$$

### Rotational Yield

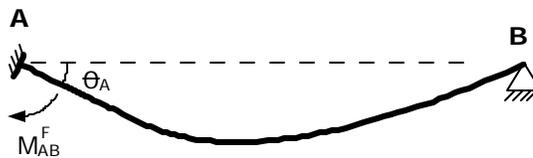
For a beam with fixed ends:



$$M_{AB}^F = \frac{4EI\theta_A}{l}$$

$$M_{BA}^F = \frac{2EI\theta_A}{l}$$

For a beam with a pinned end:



$$M_{AB}^F = \frac{3EI\theta_A}{l} \quad ; \quad M_{BA}^F = 0$$

The fixed end moments resulting from these support yields has to be factored in the moment distribution process. i.e. arithmetically added to the FEMs in *Step 2* (above).

## **2.3 REINFORCED CONCRETE BEAM DESIGN REVIEW**

Reinforced concrete is a strong durable building material that can be formed into many varied shapes and sizes ranging from simple rectangular columns, to curved domes & shells. Its utility and versatility is achieved by combining the best features of concrete and steel.

### **2.3.1 Composite Action**

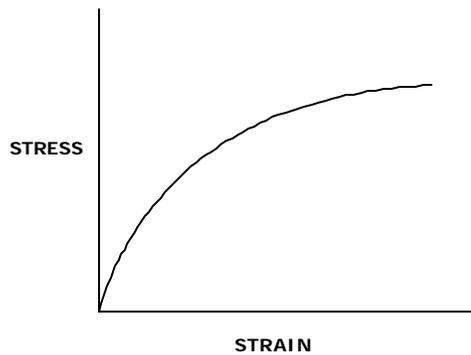
The tensile strength of concrete is only about 10% of its compressive strength. Because of this, nearly all reinforced concrete structures are designed on the assumption that the concrete does not resist any tensile forces, which are transferred by bond between the interfaces of the two materials. Thus, members should be detailed so that the concrete can be well compacted around the reinforcement during construction. In addition, some bars are ribbed or twisted so that there is an extra mechanical grip.

In the analysis and design of the composite reinforced concrete section, it is assumed that there is perfect bond, so that the strain in the reinforcement is identical to the strain in the adjacent concrete. This ensures that there is what is known as “compatibility of strains” across the cross-section of the member.

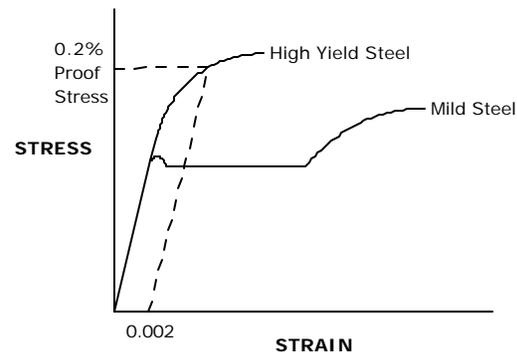
#### **Stress-Strain Curves for Concrete & Steel:**

To carry out an analysis and design of a member, it is necessary to have a knowledge of the relationship between the stresses and strains of the materials used in the member. This knowledge is particularly important when dealing with reinforced concrete, which is a composite material. In this case, the analysis of the stresses on a cross section of a member must consider the equilibrium of the forces in the concrete and steel, and also the compatibility of the strains across the cross-section.

The stress-strain curves for steel and concrete are given below:



Stress - Strain Curve for Concrete



Stress - Strain Curve for Steel

Concrete is a very variable material, having a wide range of stress-strain curves. A typical curve for concrete in compression is shown above. As the load is applied, the ratio between the stresses and strains is almost linear and the concrete behaves like an elastic material with virtually full recovery of displacement if the load is removed. Eventually, the curve is no longer linear and the concrete behaves like a plastic material, with incomplete displacement recovery during load removal at this stage. The ultimate strain for most structural concrete tends to be a constant value of approximately 0.0035, irrespective of the strength of concrete.

The figure above also shows the stress-strain curves for mild steel and high yield steel. Mild steel behaves as an elastic material up to the yield point, at which, there is a sudden increase in strain with no change in stress. After the yield point, mild steel becomes a plastic material and the strain increases rapidly up to the ultimate value.

High yield steel on the other hand, does not have a definite yield point but shows a more gradual change from elastic to plastic behavior.

$$\text{Modulus of Elasticity, } E = \frac{\text{stress}}{\text{strain}}$$

A satisfactory and economic design of a concrete structure depends on a proper theoretical analysis of individual member sections as well as deciding on a practical overall layout of the structure, careful attention to detail and sound constructional practice.

### **2.3.2 Limit State Design**

The design of an engineering structure must ensure that (1) under the worst loadings, the structure is safe, and (2) during normal working conditions the deformation of the members does not detract from the appearance, durability or performance of the structure. The Limit State method involves applying partial factors of safety, both to the loads and to the material strengths. The magnitude of the factors may be varied so that they may be used either with the plastic conditions in the ultimate state or with the more elastic stress range in the working loads.

The two principal types of limit state are the ultimate limit state and the serviceability limit state.

#### **Ultimate Limit State (ULS)**

This requires that the structure must be able to withstand, with an adequate factor of safety against collapse, the loads for which it is designed. The possibility of buckling or overturning must also be taken into account, as must the possibility of accidental damage as caused, for example, by an internal explosion.

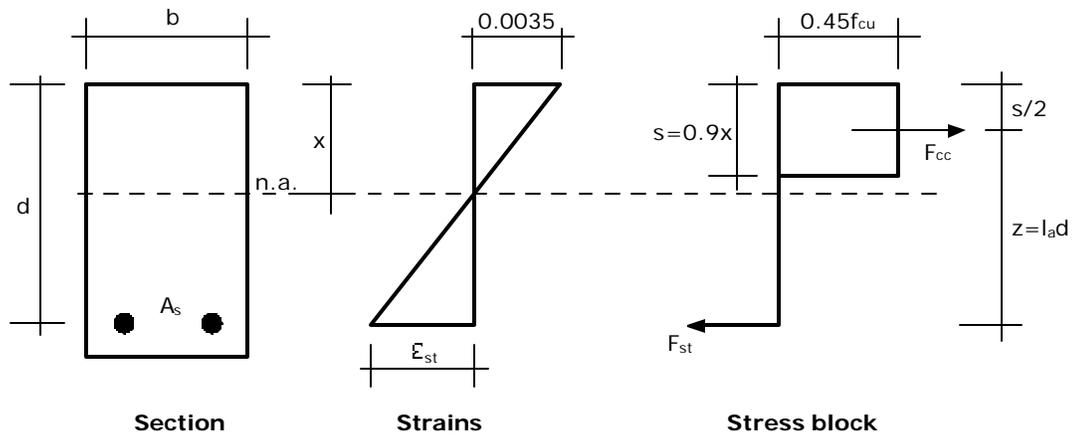
#### **Serviceability Limit State (SLS)**

This requires that the structural elements do not exhibit any preliminary signs of failure. Generally, the most important serviceability limit states are: Deflection (appearance or efficiency of any part of the structure must not be adversely affected by deflections), Cracking (local damage due to cracking and spalling must not affect the appearance, efficiency or durability of the structure) and Durability (in terms of the proposed life of the structure and its conditions of exposure). Other Limit States that may be reached include: Excessive Vibration, Fatigue & Fire Resistance.

The relative importance of each limit state will vary according to the nature of the structure. The usual procedure is to decide which the crucial limit state for a particular structure is, and base the design on this, although durability and fire resistance requirements may well influence the initial member sizing and concrete grade selection.

### 2.3.3 Bending and the Equivalent Rectangular Stress Block

For the design of most reinforced concrete structures it is usual to commence the design for the conditions at ultimate limit state, which is then followed by checks to ensure that the structure is adequate at the serviceability limit state.



Singly reinforced section with rectangular stress block

Bending in the section will induce a resultant tensile force  $F_{st}$  in the reinforcing steel, and a resultant compressive force in the concrete  $F_{cc}$  which acts through the centroids of the effective area of concrete in compression, as shown in the figure above.

For equilibrium, the ultimate design moment  $M$ , must be balanced by the moment of resistance of the section so that

$$M = F_{cc} \times z = F_{st} z \quad \dots (1)$$

where  $z$  is the lever arm between the resultant forces  $F_{cc}$  and  $F_{st}$ .

$$\begin{aligned} F_{cc} &= \text{stress} \times \text{area of section} \\ &= 0.45 f_{cu} \times bs \end{aligned}$$

and

$$z = d - \frac{s}{2} \quad \dots (2)$$

Substitution in equation (1);

$$M = 0.45 f_{cu} b s \times z$$

and replacing s from equation (2);

$$M = 0.9 f_{cu} b (d - z) z \quad \dots (3)$$

rearranging and substituting  $K = \frac{M}{f_{cu} b d^2}$  ;

$$\left(\frac{z}{d}\right)^2 - \left(\frac{z}{d}\right) + \frac{K}{0.9} = 0$$

solving this quadratic equation;

$$z = d \left[ 0.5 + \sqrt{0.25 - \frac{K}{0.9}} \right] \quad \dots (4)$$

which is the equation in the code of practice BS8110 for the lever arm, z, of a singly reinforced section.

In equation (1);

$$F_{st} = \left( \frac{f_y}{g_m} \right) A_s \quad \text{with } g_m = 1.15 \\ = 0.87 f_y A_s$$

Hence

$$A_s = \frac{M}{0.87 f_y z} \quad \dots (5)$$

Equations (4) and (5) are used to design the area of tension reinforcement in a concrete section to resist an ultimate moment, M.

As specified in BS8110;

$$z = l_a d \quad \text{with } (0.775 < l_a < 0.95)$$

using the lower limit ( $z = 0.775 d$ ) from equation (3);

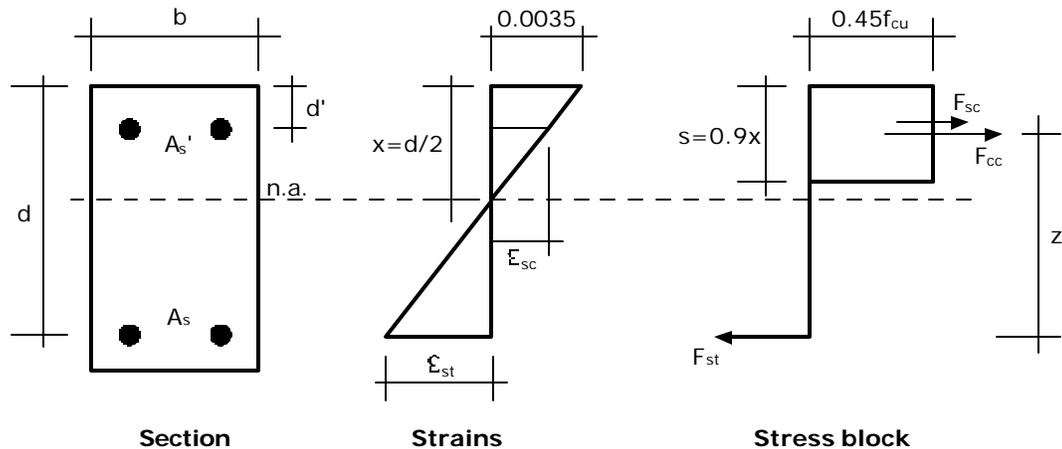
$$M = 0.156 f_{cu} b d^2 \quad \dots (6)$$

Therefore, when:

$$\frac{M}{f_{cu} b d^2} = K > 0.156$$

compression reinforcement is also required to supplement the moment of resistance of the concrete.

### 2.3.4 Rectangular Section with Compression Reinforcement at the Ultimate Limit State



Section with compression reinforcement

As previously discussed, if  $K > 0.156$  compression reinforcement is required. For this condition the depth of the neutral axis,  $x < 0.5d$ , the maximum value allowed by the code in order to ensure tension failure with a ductile section.

Therefore;

$$\begin{aligned} z &= d - \frac{s}{2} = d - 0.9 \frac{x}{2} \\ &= d - 0.9 \times 0.5 \frac{d}{2} \\ &= 0.775d \end{aligned}$$

For equilibrium of the section in the above figure;

$$F_{st} = F_{cc} + F_{sc}$$

so that with the reinforcement at yield

$$0.87 f_y A_s = 0.45 f_{cu} b s + 0.87 f_y A_s'$$

or with  $s = 0.9 \times \frac{d}{2} = 0.45d$

$$0.87 f_y A_s = 0.201 f_{cu} b d + 0.87 f_y A_s' \quad \dots (7)$$

and taking moments about the centroids of the tension steel,  $A_s$

$$\begin{aligned} M &= F_{cc} \times z + F_{sc} (d - d') \\ &= 0.201 f_{cu} b d \times 0.775d + 0.87 f_y A_s' (d - d') \\ &= 0.156 f_{cu} b d^2 + 0.87 f_y A_s' (d - d') \end{aligned} \quad \dots (8)$$

From equation (8):

$$A_s' = \frac{M - 0.156 f_{cu} b d^2}{0.87 f_y (d - d')} \quad \dots (9)$$

Multiplying both sides of equation (7) by  $(z = 0.775d)$  and rearranging;

$$A_s = \frac{0.156 f_{cu} b d^2}{0.87 f_y (d - d')} + A_s' \quad \dots (10)$$

Hence, the areas of compression steel,  $A_s'$  and tension steel,  $A_s$ , can be calculated from equations (9) and (10).

### 2.3.5 Important points mentioned in the BS8110 code used in this project

- Characteristic Strengths of Steel:

$$f_y = 460 \text{ N/mm}^2, 250 \text{ N/mm}^2 \text{ (UK) source: BS8110}$$

$$f_y = 425 \text{ N/mm}^2, 250 \text{ N/mm}^2 \text{ (Kenya) source: Kenya Bureau of Standards}$$

- Recommended Grades of Structural Concrete:

Grades 30, 35 and 40 ( $f_{cu} = 30, 35, 40 \text{ N/mm}^2$  respectively)

Lower grades are not recommended for use in structures and the use of higher concrete grades is rarely economically justified.

- Structural Analysis:

BS8110 allows a structure to be analyzed by partitioning it into subframes. The subframes that can be used depend on the type of structure being analyzed, namely braced or unbraced.

A braced frame is designed to resist vertical loads only, therefore the building must incorporate, in some other way, the resistance to lateral loading and sidesway. e.g. shear wall, tubular systems, etc.

An unbraced frame has to be designed to resist both vertical and lateral loads. i.e. the building does not incorporate any stiffening system.

#### Continuous Beam Simplification (BS8110: Clause 3.2.1.2.4.):

As a more conservative analysis of subframes, the moments and shear forces in the beams at one level may be obtained by considering the beams as a continuous beam over supports providing no restraint to rotation.

Where the continuous beam simplification is used, the column moments may be calculated by simple moment distribution procedure, on the assumption that the column and beam ends remote from the junction under consideration are fixed and that the beams possess half their actual stiffnesses.

### **Points mentioned in the in other references used this project**

#### Manual for the Design of Reinforced Concrete Building Structures:

(by the Institute of Structural Engineers, U.K.)

- Initial Estimations:

“To design even a simply supported beam, the designer needs to guess the beam size before he can include its self-weight in the analysis.”

- Span / Depth Ratios of Beams:

15	-	Continuous Beams
12	-	Simply Supported Beams
6	-	Cantilevers

- “Rules of Thumb”

the width (b) of a rectangular beam should be between  $1/3$  and  $2/3$  of the effective length (d). The larger fraction is used for relatively larger design moments.

- Degree of Accuracy:

“In every day design, final results quoted to more than three significant figures cannot normally be justified”

## **CHAPTER 3: PROGRAM REVIEW & APPLICATION**

In an effort to make the program as simple to use as possible and avoid any confusion, a color scheme has been adopted:

- Any RED object is a button and will perform a task if clicked.
- Any text within a WHITE box can either be selected or altered by clicking in it and inputting the desired figures.

If it is a blank white box, you will be required to click in it and input the appropriate figures.

Nb. For purposes of clarity of images in this report, some images have been inverted (e.g. like a photographic negative). Hence, the white input boxes will appear black.

What follows is a user guide and explanation of what goes on during the computer analyses processes of the beams.

### **3.1 SINGLE SPAN BEAM ANALYSIS**

Step 1:



#### User Instructions

This is the section where the user chooses whether to start with the Analysis or go straight to the RC Design process. In this step, choose the first button (above) to take you to the various single span beam analysis options.

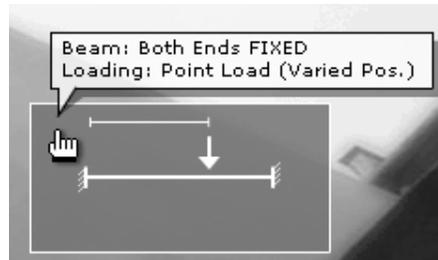
(Moving your mouse over any button will pop up a description of what it represents)

#### Section Information

At this section, no major code has been executed yet. The only active code is the expressions that pop up the description box and “stick” it to the mouse as long as it is within the button area.

When the button is clicked, we move to a new section of the program that displays the 9 single span beam analysis options.

Step 2:



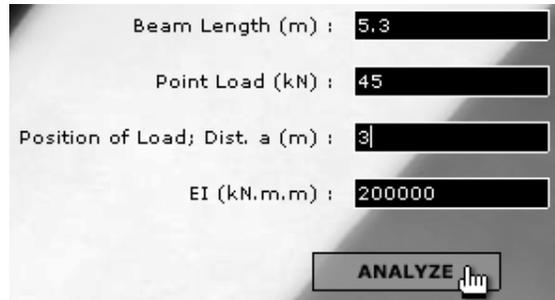
### User Instructions

You will be presented with 9 possible single span beams to analyze. If not obvious from the figures, move your mouse over the buttons to view the pop-up descriptions. For purposes of this example, the Fixed Ended beam with a non-central point load has been selected.

### Section Information

Selecting one of the beams here transfers the user to the section of the code where the user may input the relevant data for the beam type & loading selected. The same section of code for the “floating” label descriptions is still being executed.

Step 3:



Beam Length (m) : 5.3  
Point Load (kN) : 45  
Position of Load; Dist. a (m) : 3  
EI (kN.m.m) : 200000  
ANALYZE

### User Instructions

You are now at the Input Stage of the selected beam. The units relevant to each value are displayed in brackets alongside the input field. After inputting your beam dimensions and load values, click on the Analyze button.

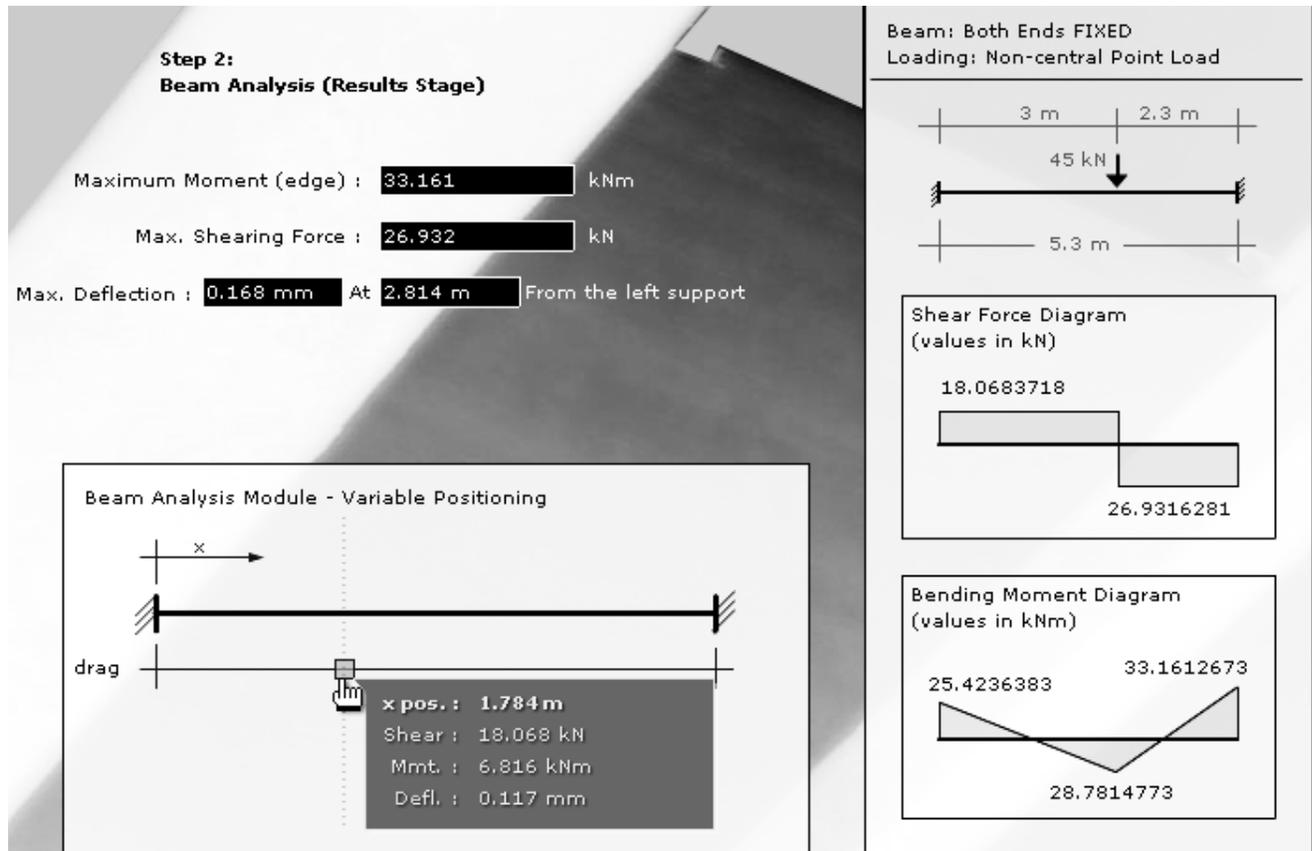
For this example, a beam length of 5.3m, a load of 45kN at 3.0m from the left support is chosen.

### Section Information

The variables have been initialized in this stage for the user to be able to input data to the program. In an event where the user inputs wrong data (e.g. distance to point load exceeds the beam length), an error pop up box has been programmed to display the wrong inputs to the user before the program can proceed with the analysis.

Once the user clicks the analyze button and the first few lines of code verify that all the inputted data is workable, the main section of code relevant to this individual beam is executed and the results displayed in the next step.

Step 4:



User Instructions

This is the analysis results stage where all the calculated values are clearly displayed. The Shear Force Diagram and the Bending Moment Diagram on the right column are only sketches showing the important peak values and are not drawn to scale. The “Beam Analysis Module” on the lower left hand side of the screen allows the red button to be dragged along the beam length to show the Shear, Moment and Deflection values at any point on the real beam.

### Section Information

After being processed, the results are displayed at this stage of the program. The Module that allows the user to drag the analysis across the beam basically scales the actual inputted beam length value to 300 pixels (width of the beam drawn), so that for every pixel moved, the computer processes a section of code that allows the “new” scaled position of the actual beam to be displayed on the screen (as “x pos”) as well as calculating the shear, moment and deflection values at that same point.

Step 5:



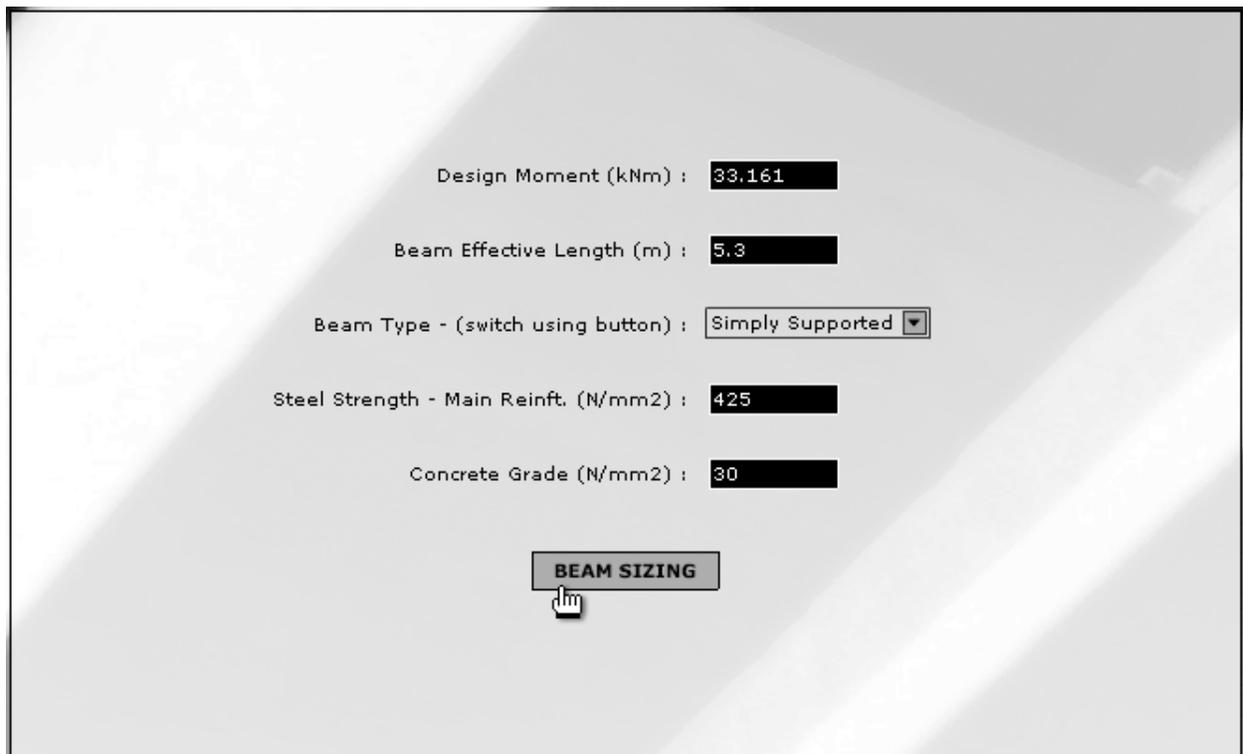
### User Instructions

At the top left corner of the screen, an “RC Design” button is visible. If clicked, this button picks the maximum moment as well as the beam span length and inputs them for you in the RC Design module. (you may alter these values at the module input stage if you wish).

### Section Information

Once the button is clicked, it executes several lines of code that pick the maximum moment value (regardless of being positive and negative) and transfer to user to the design stage where the relevant results from the analysis stage will be displayed.

In this case, the effective length would be the beam length inputted by the user (5.3m), the design moment would be the maximum moment in the beam (33.161kN) and the beam type would be set to a simply supported beam.



### 3.2 TWO SPAN BEAM ANALYSIS

#### Step 1:



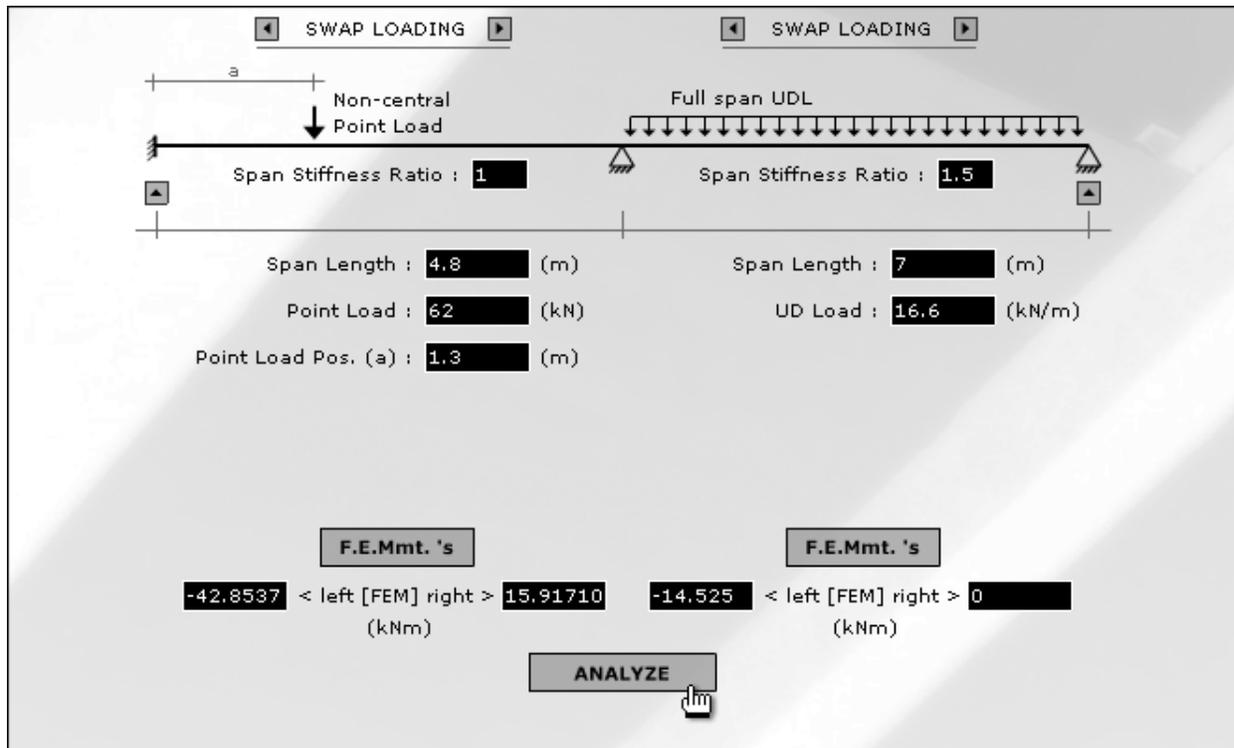
#### User Instructions

From the first program screen, the same options of analysis or design are displayed. This time round, we will select the second beam analysis option which is an analysis of a two-span beam.

#### Section Information

Once clicked, the program now moves to the section allowing the user to select his/her loading types, select the end fixity conditions (fixed or pinned) and input the span lengths & loading data.

Step 2:



User Instructions

The first step here is to select your type of loading using the two red “swap loading” arrow buttons at the top region of your screen. Once, the load types are selected (in this example, a point load on one span and a UDL on the other have been selected), you may wish to alter the end support types using the small red buttons at both ends of the beam. Next, input your data into the provided text fields (span stiffnesses, lengths, load positions and intensities) and click the F.E.Mmt.’s (fixed end moments) buttons. Once your fixed end moments have been calculated and displayed, click the Analyze button to proceed.

### Section Information

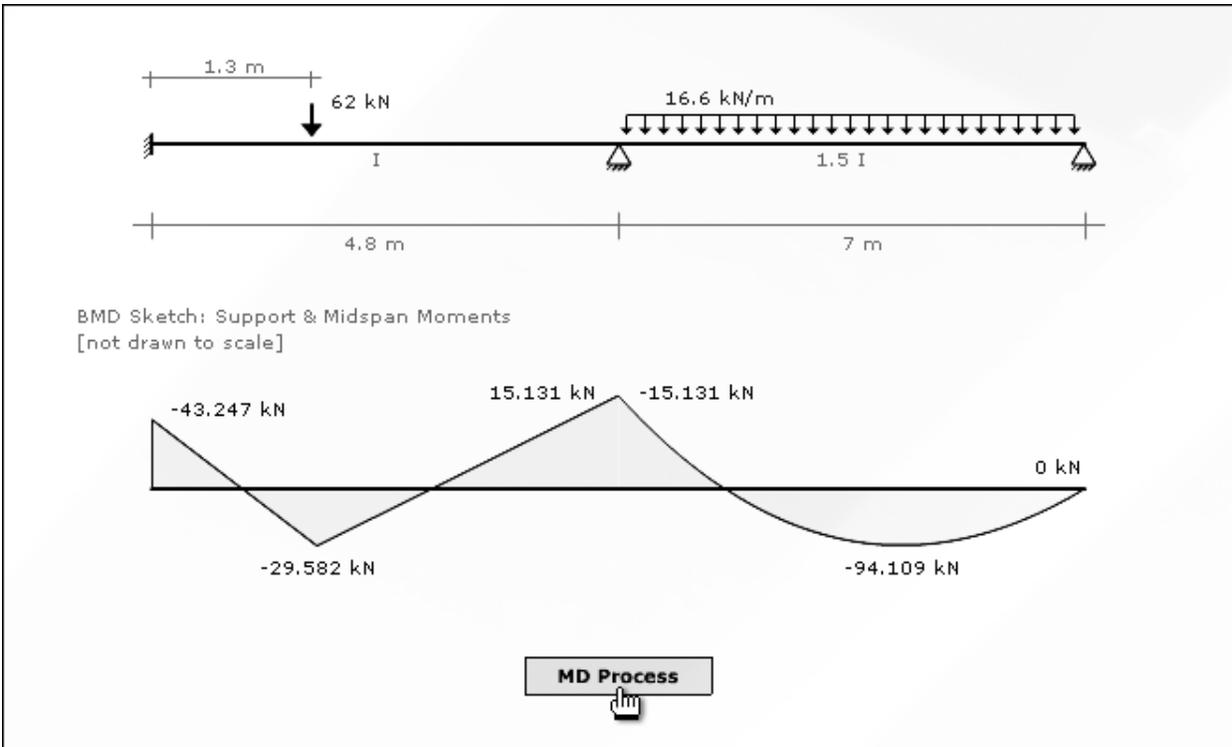
There are quite a number of activities carried out from the time the user gets into this stage to the time he/she moves on.

First of all, there are 5 main loading types and 1 extra blank one left open if the user has an uncommon loading arrangement and wishes to input his/her own FEMs. The blank loading option is also used if that span actually has no loading on it (though it may be rare to exclude the self weight of a beam from an analysis). Whatever loading setup a user may select, the appropriate section of code that calculates the FEM values for that specific loading is prepared. After the user inputs the length, loading arrangement(s) and intensities for one span, a click on the FEM button would execute the relevant code and display the FEM values on the same screen.

The user may also wish to introduce fixed end supports instead of the default pin ends. Changing these supports by clicking the edge buttons does not visually carry out any task but in actual sense, alters the whole moment distribution process (since moments cannot be distributed to pin supports but only to fixed ends).

Once all the input data and FEMs have been set up, the user clicks the Analyze button. This starts up the whole moment distribution process and displays the results on the next screen. The moment distribution process for a two span beam consists of only one iteration. (The full moment distribution code is explained in the Discussion, Chap. 4)

Step 3:



User Instructions

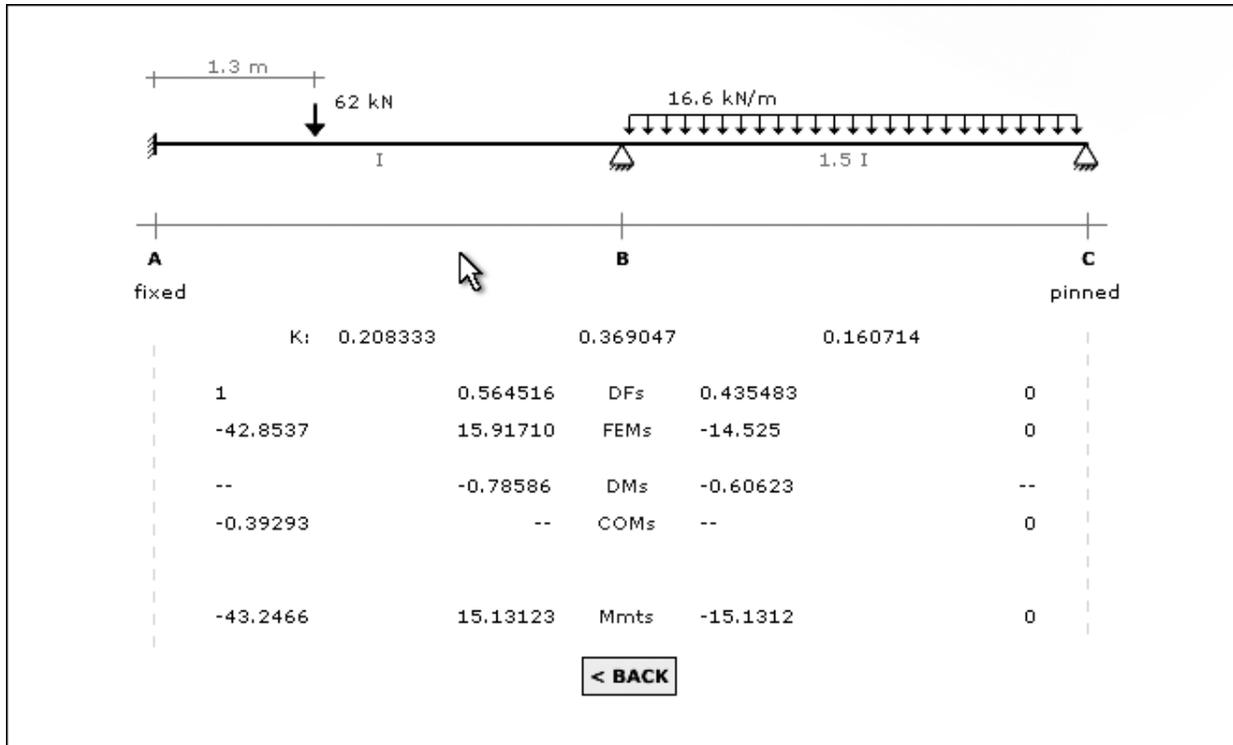
For the data inputted in step 2, the resulting support moments and midspan moments are displayed on a sketch of the bending moment diagram.

The user may wish to view the details of the moment distribution process. This may be possible by clicking on the MD Process button at the bottom of the screen.

Section Information

Depending on the fixity conditions and the type of loading initially selected, the appropriate BMD sketch (per span) is called up and displayed. The salient moment values calculated are displayed on their respective positions on the sketch.

The moment distribution process for this example is displayed as follows:



As can be seen, only one distribution was necessary to achieve a moment balance. It can also be seen that no moments have been carried over to support C since it is a pinned support.

Step 4:



### User Instructions

Similarly, a linkage between the analysis and design stages is possible at the click of a button. The RC Design button is located at the same place (top left corner) as it was for the single span beam analysis. Clicking this button will bring you to the RC Design section and the selected design values from the analysis will be pre-set for you. You may alter them if you so wish.

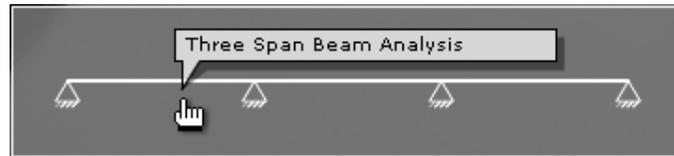
### Section Information

Once this button is clicked, it evaluates all the support moments as well as the midspan moments to obtain the maximum value (regardless of the sign. e.g. Design moment for this example = -94.109). The maximum span length is also determined and selected as the effective length of the beam to be designed (7.0m). The last piece of information swapped between these modules is the beam type, which is continuous in this case.

A screenshot of a software interface for reinforced concrete design. It features several input fields with values pre-filled: Design Moment (kNm) is 94.109, Beam Effective Length (m) is 7, Beam Type is set to Continuous in a dropdown menu, Steel Strength - Main Reinf. (N/mm<sup>2</sup>) is 425, and Concrete Grade (N/mm<sup>2</sup>) is 30. A mouse cursor is hovering over the Concrete Grade field. Below these fields is a button labeled "BEAM SIZING".

### 3.3 THREE SPAN BEAM ANALYSIS

#### Step 1:



#### User Instructions

Back to the opening screen, the last option for analysis is that for a three-span beam.

#### Section Information

As was with the previous sections, this stage displays the possible modules to the user. Clicking the three-span button will send the user to the input stages for the analysis of a three-span beam.

Step 2:

The screenshot shows a software interface for beam analysis. At the top, there are three 'SWAP LOADING' buttons. Below them is a diagram of a beam with two spans, supported by a fixed support on the left and two roller supports. The spans are labeled 'a', 'b', and 'c'. The total length is marked as 'l/2'. Loading options include 'No Loading (May input direct FEMs)', 'Partial UDL', and 'Central Point Load'. Below the diagram, input parameters are listed:

- No Loading (May input direct FEMs): Span Stiffness Ratio : 1
- Partial UDL: Span Stiffness Ratio : 3
- Central Point Load: Span Stiffness Ratio : 2.5

Below these are the input values for the spans:

- Span Length : 4.5 (m)
- Span Length : 5.5 (m)
- Span Length : 6 (m)
- UD Load : 32.3 (kN/m)
- Point Load : 50 (kN)
- UDL Pos. (b) : 1.2 (m)
- Extent of UDL (c) : 2 (m)

At the bottom, the resulting bending moment diagrams are shown in kNm:

- Span 1: 0 < left [FEM] right > 0
- Span 2: -48.0310 < left [FEM] right > 33.32576
- Span 3: -37.5 < left [FEM] right > 37.5

Buttons for 'ANALYZE' and 'F.E.Mmt. 's' are also visible.

### User Instructions

This setup is exactly the same as the two-span process; the only difference is, of course, the extra span.

In this example, the loading on the first span (left) has been left blank. This could either denote an unloaded span or may give the user flexibility in inputting FEMs for a unique loading type not catered for in the “swap loading” element. If you have no loading, you will be required to input 0 (zero) in the FEM fields.

### Section Information

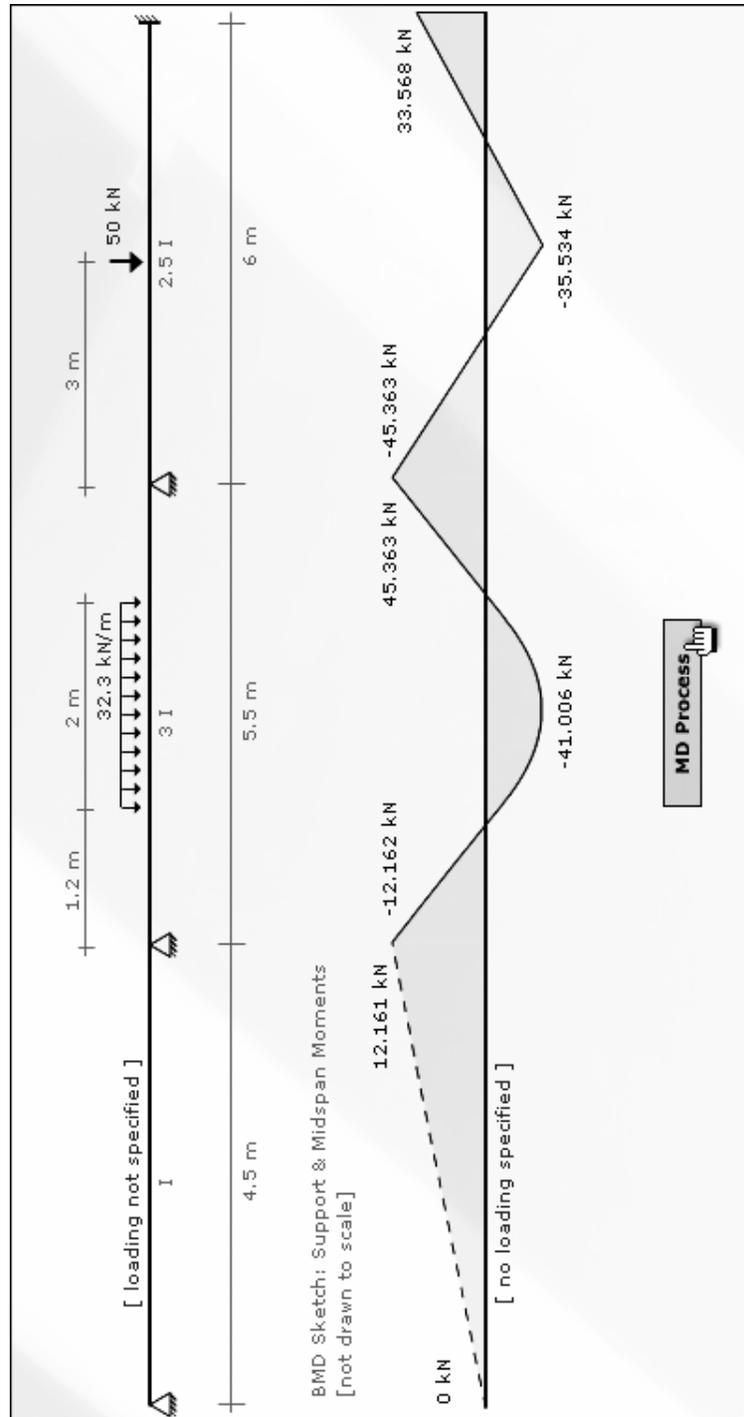
The same sections of code are executed for the FEM calculations here as were previously done for the 2-span input stage.

Once the user selects the loading setup, the appropriate section of code that calculates the FEM values for that specific loading is prepared. After the user inputs the length, loading arrangement and intensities for one span, a click on the FEM button would execute the relevant code and display the FEM values under the respective span.

The same conditions apply for changing the end support conditions (fixed or pinned). Changing these supports by clicking the edge buttons also alters the whole moment distribution process here (since moments cannot be distributed to pin supports but only to fixed ends).

Once all the input data and the three sets of FEMs have been set up, the user clicks the Analyze button. This starts up the whole moment distribution process and displays the results on the next screen. The moment distribution process for a three span beam consists of several iterations. In this case, 12 iterations are carried out before the moment values are displayed. (The full moment distribution code process for the three span analysis is also explained in the Discussion, Chap. 4)

Step 3:



### User Instructions

The bending moment diagrams here are also not drawn to scale but are representative of the type of loading selected and the end fixity conditions. The salient moment values (support and midspan moments) are displayed.

To view the moment distribution process, click on the MD Process button at the bottom.



To proceed straight to the design stage, click the RC Design button on the top left corner of the screen.

### Section Information

This stage is similar to the two-span display stage. The main difference however is in the number of iterations (12) in the moment distribution process.

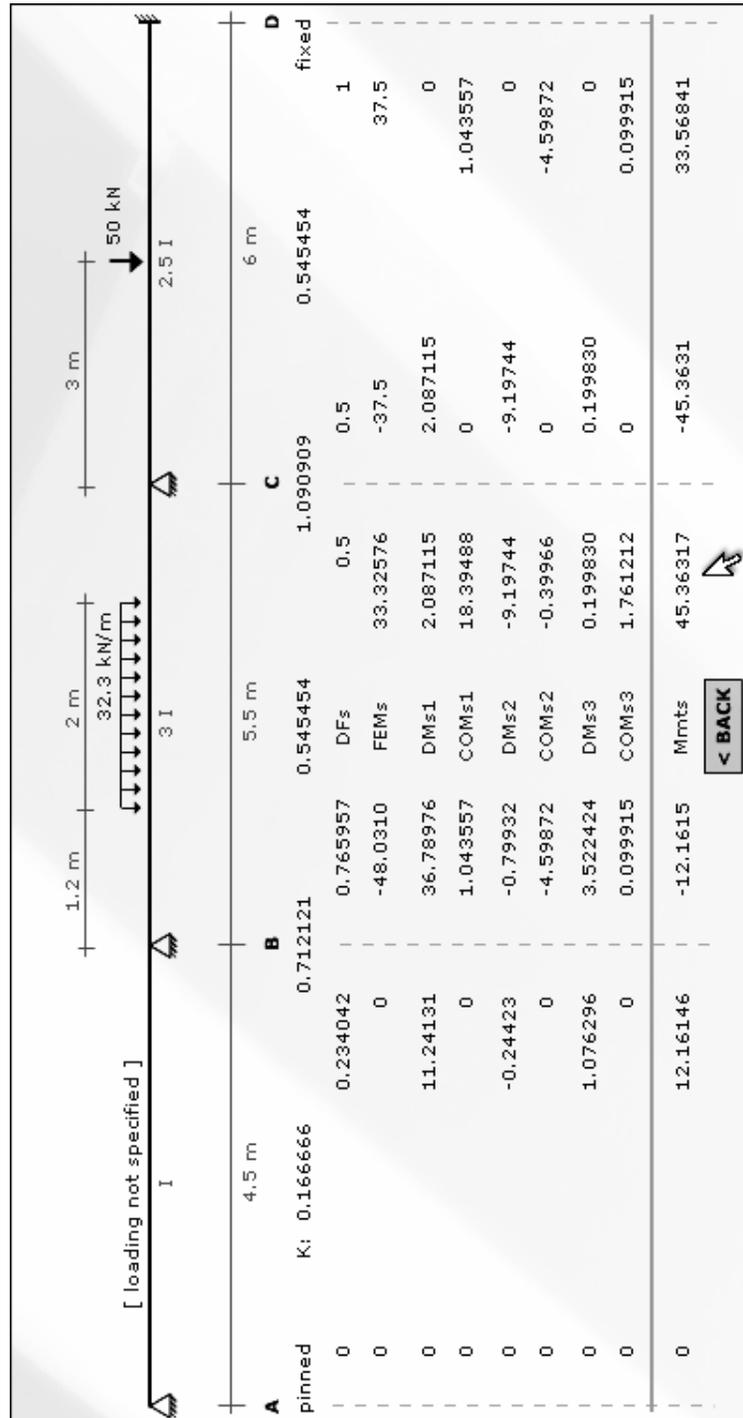
In the example shown above, the dotted BMD on the first span exists because the loading type was not specified; only a set of FEMs was inputted. Hence, a dotted parabolic shape with an accompanying comment was seen as suitable.

The moment distribution process for the above three-span example is shown below:

(However, due to screen size limitations, not all 12 iterations can be viewed. Only the first three distributions are displayed).

It has been attempted to display these moment distribution tables as clearly as possible, with a similar arrangement as one would typically adopt in a manual solution.

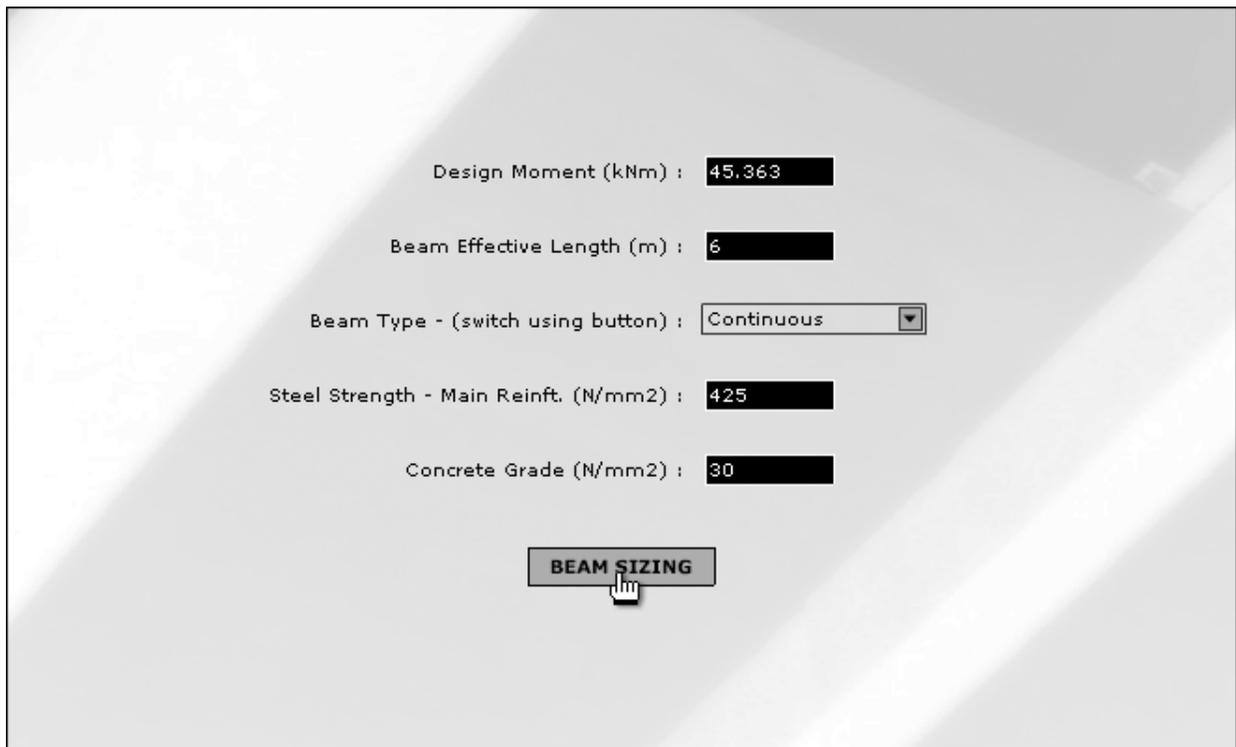
Computer Analysis & Reinforced Concrete Design of Beams



### Computer Analysis & Reinforced Concrete Design of Beams

---

Once the user clicks the RC Design button from the results stage of the three-span analysis, the moment values are evaluated to determine the maximum value and concurrently, send this value to the RC Design module as the design moment. In our example, the maximum moment from the analysis is  $-45.363\text{kN}$ . Similarly, the maximum span length (6m) is set as the effective length for the design. Finally, the beam type is set as Continuous.



Design Moment (kNm) : 45.363

Beam Effective Length (m) : 6

Beam Type - (switch using button) : Continuous

Steel Strength - Main Reinf. (N/mm<sup>2</sup>) : 425

Concrete Grade (N/mm<sup>2</sup>) : 30

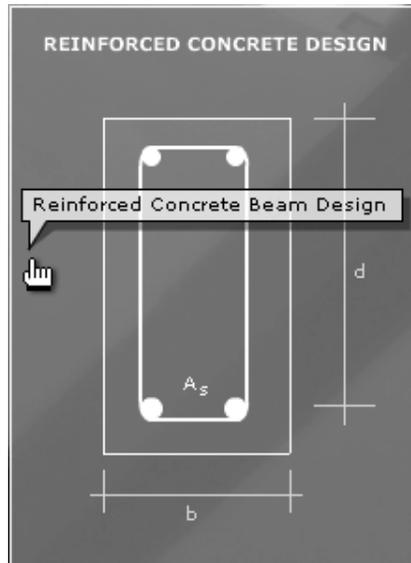
**BEAM SIZING**

### **3.4 REINFORCED CONCRETE DESIGN**

There are two ways of accessing the RC design module in this program. One is directly selecting it from the opening screen and the other is from the various result display stages in the one, two and three span analysis modules. The latter pre-inputs the design moment, effective length and beam type depending on the characteristics of the analysis. These figures may be altered if the user wishes to do so.

As was done for the analysis modules, the following is a user guide and explanation of what goes on during the computer design process.

Step 1:



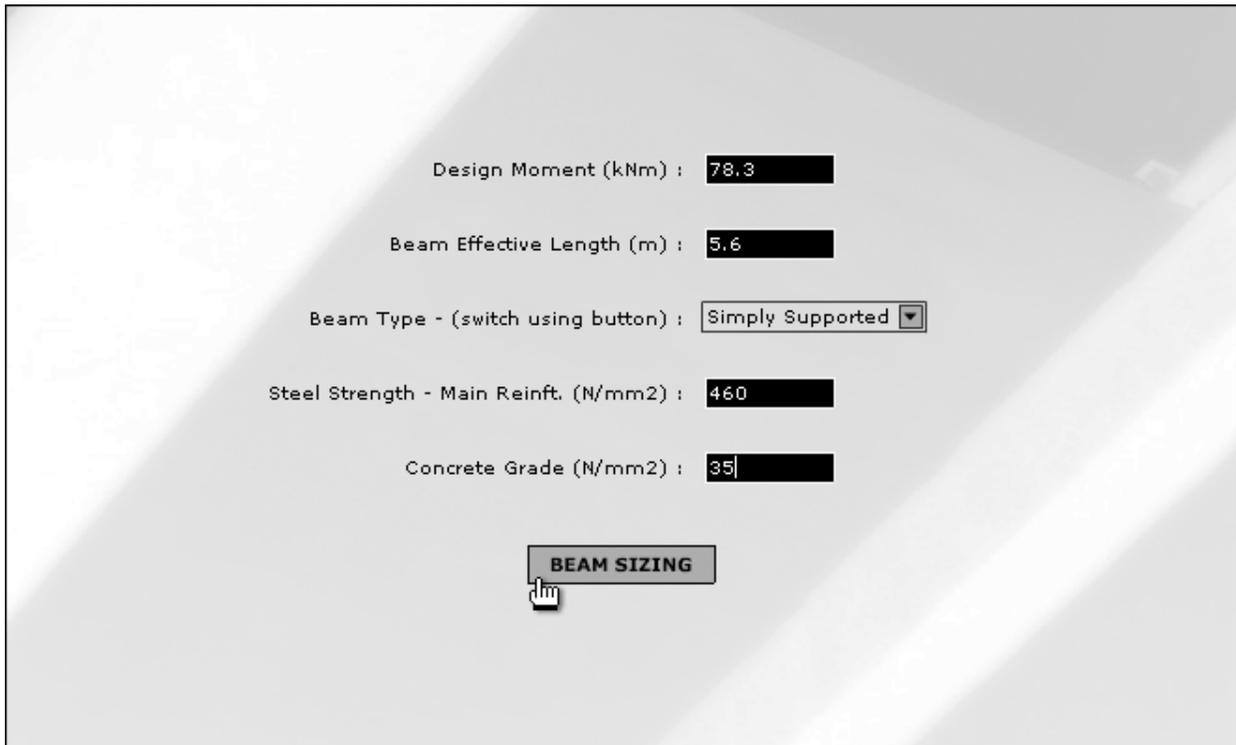
### User Instructions

From the opening screen, select the Reinforced Concrete Design button (as shown above) to proceed to the RC Design module.

### Section Information

Apart from the “floating” label code to make the description follow the mouse when the user moves over a button, no major code is being executed at this stage. When the user clicks the RC Design button, the input stage is brought to screen where the user inputs the design parameters as will be seen in the next stage.

Step 2:



Design Moment (kNm) : 78.3

Beam Effective Length (m) : 5.6

Beam Type - (switch using button) : Simply Supported ▾

Steel Strength - Main Reinf. (N/mm<sup>2</sup>) : 460

Concrete Grade (N/mm<sup>2</sup>) : 35

**BEAM SIZING**

User Instructions

Here, the required design parameters are to be inputted. The respective value units are denoted in parentheses beside the respective input field. After all the parameters are entered, click the Beam Sizing button to proceed.

### Section Information

In this stage, several design parameters necessary for the following stages are inputted. If the design module is accessed from the opening screen, the Design Moment and Beam Effective Length are blank and have to be filled by the user. If accessed from the beam analyses results, the values will already be present, but can be altered.

The steel strength and concrete grade are set to the default values ( $425\text{N/mm}^2$  and  $30\text{N/mm}^2$  respectively). As can be seen from the above values, these are flexible and can be changed by the user.

BS8110 specifies three main types of beams that can be designed. These are: Simply Supported, Continuous and Cantilevered Beams. These selections can be made using the dropdown button at the Beam Type input field. Every time the button is clicked, a new set of variables (BS8110 recommended sizing ratios; 20 for simply supported, 26 for continuous and 7 for cantilevered) is brought forward for use by the next section of code. In an event where the user inputs wrong data (e.g. a negative beam length), an error pop up box has been programmed to display the wrong inputs to the user before the program can proceed with the analysis.

Once the user clicks the analyze button and the first few lines of code verify that all the inputted data is workable, the main section of code dealing with beam section sizing is executed and the results displayed in the next step.

Step 3:

Calculated Beam Width (mm) : 233.3  
Suggested Beam Width (mm) : **240**

Calculated Effective Depth (mm) : 466.7  
Suggested Effective Depth (mm) : **470**

Suggested Concrete Cover (mm) : **30**

Suggested Tolerance (mm) : **10**

Old Design Moment - Excluding Self Weight of Beam (kNm) : 78.3  
New Design Moment - Including Self Weight of Beam (kNm) : **89.815**

RESIZE

DESIGN REINFORCEMENT

2Y12  
A<sub>s</sub>  
470 mm  
510 mm  
240 mm

The screenshot displays a software interface for reinforced concrete beam design. On the left, a list of parameters is shown with their calculated and suggested values. The suggested values are highlighted in black boxes. A 'RESIZE' button is located below the list. To the right, a diagram of a rectangular beam cross-section is shown. The width is labeled as 240 mm. The effective depth is labeled as 470 mm, and the total height is labeled as 510 mm. The diagram shows two reinforcement bars at the top, labeled '2Y12', and a reinforcement area at the bottom, labeled 'A<sub>s</sub>'. Below the diagram, the 'Old Design Moment - Excluding Self Weight of Beam (kNm)' is 78.3, and the 'New Design Moment - Including Self Weight of Beam (kNm)' is 89.815. A 'DESIGN REINFORCEMENT' button is located at the bottom center of the interface.

User Instructions

The calculated and suggested beam dimensions are displayed. The new design moment, which includes the self weight of the beam, is also displayed. If you do not wish to factor the self weight into the design moment, you may rectify the value to display the old design moment. The suggested beam breadth and effective depth may be altered but it is not recommended to input a new dimension smaller than that suggested. This is because any smaller dimensions will result in a lower concrete capacity and if this is too low, the design moment may not be resisted by the lowered concrete capacity.

A default cover of 30mm and a tolerance of 10mm are displayed. If you wish to change any dimension or figure, click the resize button after you input the new value(s). The new dimensions will be displayed on the figure alongside. The additional design moment will be either increased or decreased depending on how much you have increased or decreased the section area.

Once the resizing is complete and the new design moment attained, click the button at the bottom of the screen labeled Design Reinforcement to proceed.

### Section Information

To arrive at this information, the major part of the beam sizing code dealt with specifying a minimum effective depth that will:

- satisfy the BS8110 ratio values
- ensure that the concrete capacity of the section is adequate
- limit the excessive deflection for this beam (via modification factor)
- avoid a situation where the shearing force governs the design.

Next, the breadth is calculated as  $\frac{1}{2}$  the effective depth.

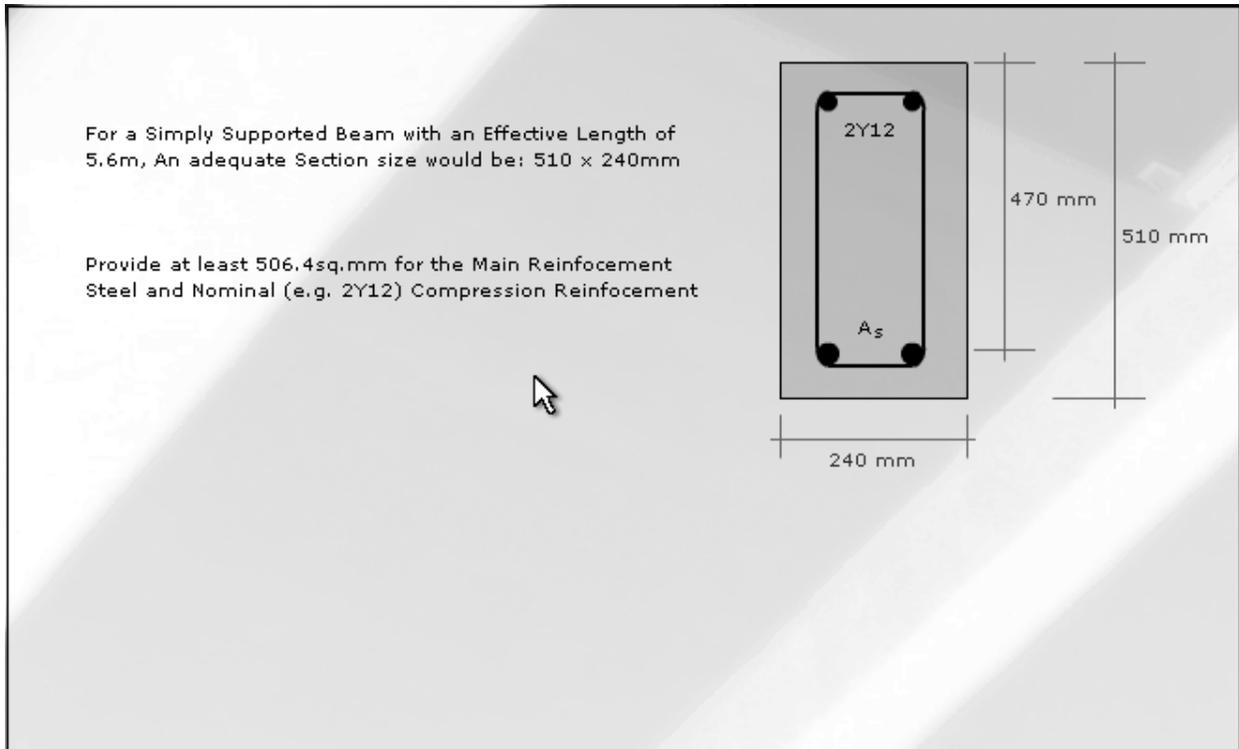
The calculated minimum effective depth and breadth of the beam section are displayed. However, the user is not required to follow these exact figures; after all, design is an art. The calculated dimensions are displayed and the suggested minimum dimensions displayed immediately below them. These suggested minimum dimensions are arrived at by rounding off the calculated breadth and effective depth to the nearest higher 10mm. (i.e. 23mm is rounded off to 30mm rather than 20mm).

The new design moment is essentially a summation of the old design moment inputted by the user and the moment induced by the self weight of the beam acting as a UDL along the beam length.

The user may alter any dimensions. After doing so, clicking the Resize button will set the new dimensions into the scripts as well as recalculate the new design moment by factoring in the added self weight of the beam.

Once the resizing is complete and the new design moment attained, click the button at the bottom of the screen labeled Design Reinforcement to proceed.

Step 4:



User Instructions

This is the final stage of the process that displays the areas of steel required for the design parameters specified. The final member sizing and area(s) of reinforcement displayed. If compression steel is required, its area will also be calculated and displayed alongside the area of tension steel.

The areas of steel shown in the figure above should serve as minimum areas of reinforcement to be provided in such a section. After reading off this area, an adequate arrangement of steel bars should be determined.

### Section Information

To get to this final stage, the main sections of code dealt with determining the area(s) of steel required. A check was made to calculate whether compression steel would be required to supplement the moment of resistance of the concrete. All figures such as  $K$ , lever arm,  $z$ , etc were determined in this section. These values facilitated the calculation of the designed areas of steel. If the design moment did not exceed the moment of resistance of the concrete, a statement within the script would jump to a different region of code that would only allow for the specification of a minimum area of steel required for the section, and provide the standard 2Y12 bars in the compression zone to act as shear hangars.

## **CHAPTER 4: DISCUSSION**

This chapter discusses the main code sections of the program. These are:

- Code for Single Span Beam Analysis
- Code for the Two & Three Span Beam Analysis (similar)
- Code for Reinforced Concrete Design Module

All code written for this program has been printed out in the Appendices.

### Note on Code:

Any line starting with two forward slashes indicates that it is a comment.

e.g.

```
// this is a comment
```

All other lines are executable functions, variables, expressions or strings.

## **STEP BY STEP CODE PRINTOUTS & EXPLANATIONS**

### **4.1 Single Span Beam Analysis**

#### **Code on Keyframe 01:**

```
stop();  
// clearing the Input Field Values  
s = "";  
w = "";
```

*The above code sets all text fields to <empty> so as to prepare the error box pop ups in case the user has not inputted any values and clicks the analyze button.*

#### **Code on the ANALYZE button:**

```
on (release) {  
    // this is the function controlling the error message popups for  
    the input fields  
    function error(msg) {  
        // the function name is "error" and the parameter is "msg"  
        with (_root.beam01.errorbox) {  
            gotoAndStop(5);  
            _root.beam01.errorbox.errormsg = msg;  
            // we just assigned _root.beam01.errorbox to MSG  
        }  
    }  
}
```

*A function has been created to check the validity of an input figure and display an error message.*

```
// conditions :  
if (s<0) {  
    // checking for negative lengths  
    error("Please insert a Positive Beam Length value");  
    // calling "error" funct & setting the param "msg"  
} else if (s == 0) {  
    // checking if the length is zero...  
    error("\Zero\" is not a valid Beam Length");  
} else if (s == "") {  
    error("Please insert a Beam Length value");  
} else if (w == 0) {  
    error("Please insert a larger Load value");  
} else if (w == "") {  
    error("Please insert a Load value");  
} else {
```

*This is the section where the parameters are sent to the error function to set the right error message for the respective wrong value inputted*

```
// here come the equations  
// Memory Note: (w=UDL), (s=beam span length)
```

*throughout the program, the letter “l” representing length has been avoided because it can be easily confused with the number “1”. In most scripts, “l” has been replaced with “s”.*

```
mmax = (w*s*s)/12;  
mcentre = (w*s*s)/24;  
dmax = (w*s*s*s*s)/(384*ei*0.001);  
sf = (w*s)/2;
```

*these are the equations for a single span fixed -ended beam with a UDL.  
Namely:*

$$M_{spt} = \frac{wl^2}{12}$$

$$M_{midspan} = \frac{wl^2}{24}$$

$$\Delta_{max} = \frac{wl^4}{384EI}$$

$$ShearForce = \frac{wl}{2}$$

```
// preparing values for display + rounding them off to 3.d.p.  
mmaxval = Math.round(mmax * 1000) / 1000;  
mcentreval = Math.round(mcentre * 1000) / 1000;  
dmaxval = Math.round(dmax * 1000) / 1000;  
sfval = Math.round(sf * 1000) / 1000;
```

*flash usually rounds off values to 12 significant figures. Such accuracies are not required; hence, the values have been rounded off to three decimal places for display.*

```
// we'll now go to the display section (Keyframe 5)  
gotoAndPlay(5);  
}  
}
```

### **Code Executed on the Dragger Module:**

```
onClipEvent (enterFrame) {  
  
    // to obtain the position of the dragger in the Beam01Module  
    dist = 150 + _root.beam01.b01module.dragger._x;  
    // rel. to CenterPoint of B01Module, "dragger" MC is -155px (left)  
  
    // DragButton's _xpos is -5px ... hence the 150 + initially  
    // next, localizing variables to avoid long target paths in eqns  
    x = (dist * _root.beam01.s) / 300;
```

*the x position is calculated at every pixel value of the drawn beam and the corresponding real x that would be displayed if the beam were actually 300pixels wide is calculated.*

```
    w = _root.beam01.w;  
    s = _root.beam01.s;  
    ei = _root.beam01.ei;  
  
    // ... and here come the equations  
    mx = w*(6*s*x - s*s - 6*x*x)/12;  
    dx = ((w*x*x) * ((s - x)*(s-x)))/(24*ei*0.001);  
    sfx = w*((s/2)-x);
```

*these equations are executed for every pixel moved by the dragger:*

$$M_x = \frac{w}{12}(6sx - s^2 - 6x^2)$$

$$\Delta_x = \frac{wx^2}{24EI}(s-x)^2$$

$$Shear_x = w\left(\frac{s}{2} - x\right)$$

```
    // next, rounding off to 3.d.p. and preparing results for display  
    xval = Math.round(x * 1000) / 1000 + " m";  
    mxval = Math.round(mx * 1000) / 1000 + " kNm";  
    dxval = Math.round(dx * 1000) / 1000 + " mm";  
    sfxval = Math.round(sfx * 1000) / 1000 + " kN";  
}
```

### **Code on the Button to be Dragged:**

```
on (press) {  
    // dragging the whole "dragger" MC  
    startDrag("_root.beam01.b01module.dragger", false, -150, 5, 150, 5);  
    // spans from -150px(left) to +150px(right) = 300pixels to calculate  
    // Nb. whatever length (s) the user inputted, it will be split into  
    // 300 for each pixel moved by the dragger  
}
```

*this section allows the small red button called “dragger” to be dragged when pressed only within the beam length and not to anywhere else on the screen. When the button is released, the last command stops the dragging process.*

```
on (release, releaseOutside) {  
    stopDrag();  
}
```

### **Code on the RC Design Button:**

```
on (release) {  
    _root.origin = "analysis";  
    _root.store.m = mmaxval;  
    _root.store.s = s;  
    _root.store.beamtype = "Simply Supported";  
}
```

*this section the maximum moment values, the beam length and beam type variables are sent to a temporary storage (called `_root.store`) so as to be picked up from the RC Design module later on.*

```
with (_root) {  
    gotoAndStop(115);  
    with (_root.design) {  
        gotoAndStop(2);  
    }  
}
```

*the last few lines send the program to the RC Design module, specifically to the section where a code b it asks for the 3 variables from the temporary storage.*

## 4.2 Two & Three Span Load Swapper Modules

### Code on Keyframe 01 of the Load Swapper Module:

```
lclleft = 1;  
s = "";  
w = "";  
a = "";  
b = "";  
c = "";  
p = "";  
femab = "";  
femba = "";  
stop();
```

*this code is executed as soon as the user clicks on any of the two or three span beam analysis modules. Its basic function is to set all the text field values to <empty> so that in case a user forgets to input a value, the error box pop up will work properly.*

### Code on the FEM button for the First Loading Case (UDL):

```
on (release) {  
  // the function controlling the error messages for the input fields  
  function error(msg) {  
    // the function name is "error" and the parameter is "msg"  
    with (_root.twospan.errorbox) {  
      gotoAndStop(5);  
      _root.twospan.errorbox.errormsg = msg;  
      // we just assigned _root.bla.bla.bla to MSG  
    }  
  }  
}
```

*this is almost the same error box function as was discussed earlier in the single span module. The only difference is in the naming of the text fields to be called and the descriptions to the user of what exactly the wrong input is.*

```
// conditions :  
if (s<0) {  
  // checking for negative lengths  
  error("Please insert a Positive Beam Length value");  
  // calling error funct & setting "msg" to "Please inse..."
```

*the first condition checks whether the beam length (s) is negative (<0), and if it is, it displays the appropriate error description to the user.*

```
} else if (s == 0) {  
  // Nb. "=" assigns and "==" is a condition equality  
  error("\Zero\" is not a valid Beam Length");  
  // Nb. the code reads \" as "  
} else if (s == "") {  
  error("Please insert a Beam Length value");  
} else if (stiff == "") {  
  error("Please insert a Stiffness Ratio");
```

```
} else if (stiff < 0) {  
    error("Please insert a Positive Stiffness Ratio");  
} else if (w == 0) {  
    error("Please insert a larger Load value");  
} else if (w == "") {  
    error("Please insert a Load Value");  
} else {
```

*the error checks are now complete. If they all pass, the code proceeds to the next section which calculates the fixed end moment values as well as the midspan moment according to the type and intensity of loading chosen as well as the end support conditions.*

```
    leftspt = _root.twospan.leftsptswapper.leftspt;  
    if (leftspt == "pinned") {  
        femab = 0;  
        femba = w*s*s/12 + w*s*s/24;  
  
    } else if (leftspt == "fixed") {  
        femab = -w*s*s/12;  
        femba = w*s*s/12;  
    }  
  
    mmidleft = w*s*s/8;  
} }
```

*the equations are as follows:*

*for a fixed ended span: (i.e. the edge support is set as fixed)*

$$FEM_{AB} = -\frac{wl^2}{12}$$

$$FEM_{BA} = \frac{wl^2}{12}$$

*for a propped cantilever: (i.e. the edge support is pinned)*

$$FEM_{AB} = 0$$

$$FEM_{BA} = \frac{wl}{8}$$

*midspan moment as a simply supported span with a UDL:*

$$M_{AB \text{ MIDSPAN}} = \frac{wl^2}{8}$$

*Nb. The remaining code sections carry out the exact task as this bit just discussed; the only difference is in the text fields to be checked for errors and the different moment equations for the different loading type setups and support conditions.*

**Code on the FEM button for the Second Loading Case (Partial UDL):**

```
on (release) {
  b = parseFloat (b);
  c = parseFloat (c);
  s = parseFloat (s);
  w = parseFloat (w);
  // this is the function controlling the error message popups for
the input fields
  function error(msg) {
    // the function name is "error" and the parameter is "msg"
    with (_root.twospan.errorbox) {
      gotoAndStop(5);
      _root.twospan.errorbox.errormsg = msg;
      // we just assigned _root.bla.bla.bla to MSG
    }
  }
  // conditions :
  if (s<0) {
    // checking for negative lengths
    error("Please insert a Positive Beam Length value");
    // calling error function & setting the param "msg".
  } else if (s == 0) {
    // checking if the length is zero...
    error("\\"Zero\\" is not a valid Beam Length");
    // Nb. the code reads  \\" as  "
  } else if (s == "") {
    error("Please insert a Beam Length value");
  } else if (stiff == "") {
    error("Please insert a Stiffness Ratio");
  } else if (stiff < 0) {
    error("Please insert a Positive Stiffness Ratio");
  } else if (w == 0) {
    error("Please insert a larger Load value");
  } else if (w == "") {
    error("Please insert a Load Value");
  } else if (b == "") {
    error("Please insert a Load Position Value");
  } else if (b < 0) {
    error("Please insert a Positive Load Position Value");
  } else if (b >= s) {
    error("Please reduce your Load Position Value");
  } else if (c == "") {
    error("Please insert a UDL Length Extent Value");
  } else if (c < 0) {
    error("Please insert a Positive UDL Length Value");
  } else if (c > s) {
    error("Please reduce your UDL Length Value");
  } else if (b+c > s) {
    error("Please reduce Load Position and UDL Length Values");
  } else {
```

*the code here now includes different equations for the respective loading setup:*

```
    d = b+c;
    g = s-d;
```

```
e = c+g;
d = b+c;
g = s-d;
e = c+g;
leftspt = _root.twospan.leftsptswapper.leftspt;
if (leftspt == "pinned") {
    femab = 0;
    femba = ((w)/(8*s*s*c))*(e*e-g*g)*(2*s*s-g*g-e*e);
} else if (leftspt == "fixed") {
    femab = ((-w)/(12*s*s))*((e*e*e)*((4*s)-(3*e))-
    (g*g*g)*((4*s)-(3*g)));
    femba = ((w)/(12*s*s))*((d*d*d)*((4*s)-(3*d))-
    ((b*b*b)*(4*s-3*b))) +
    ((w)/(24*s*s))*((e*e*e)*((4*s)-(3*e))-
    (g*g*g)*((4*s)-(3*g)));

    rl = ((w*c)/(2*s))*(2*g + c);
    mmidleft = rl*(b + (rl/(2*w)));
}
```

### **Code on the FEM button for the Third Loading Case (Central Point Load):**

```
on (release) {
    // function controlling error message popups for the input fields
    function error(msg) {
        // the function name is "error" and the parameter is "msg"
        with (_root.twospan.errorbox) {
            gotoAndStop(5);
            _root.twospan.errorbox.errormsg = msg;
            // we just assigned _root.bla.bla.bla to MSG
        }
    }
    // conditions :
    if (s<0) {
        // checking for negative lengths
        error("Please insert a Positive Beam Length value");
        // calling our "error" function and setting the param "msg"
    } else if (s == 0) {
        // Nb. "=" assigns and "==" is a condition equality
        error("\Zero\" is not a valid Beam Length");
        // Nb. the code reads \" as \"
    } else if (s == "") {
        error("Please insert a Beam Length value");
    } else if (stiff == "") {
        error("Please insert a Stiffness Ratio");
    } else if (stiff < 0) {
        error("Please insert a Positive Stiffness Ratio");
    } else if (p == 0) {
        error("Please insert a larger Load value");
    } else if (p == "") {
        error("Please insert a Load Value");
    } else {
        leftspt = _root.twospan.leftsptswapper.leftspt;
        if (leftspt == "pinned") {
            femab = 0;
            femba = p*s/8 + p*s/16;
        }
    }
}
```

```
    } else if (leftspt == "fixed") {  
        femab = -p*s/8;  
        femba = p*s/8;  
    }  
    mmidleft = p*s/4;    }  
}
```

### **Code on the FEM button for the Fourth Loading Case (Non-central Point Load):**

```
on (release) {  
    a = parseFloat (a);  
    s = parseFloat (s);  
    // funct controlling the error message popups for the input fields  
    function error(msg) {  
        // the function name is "error" and the parameter is "msg"  
        with (_root.twospan.errorbox) {  
            gotoAndStop(5);  
            _root.twospan.errorbox.errormsg = msg;  
            // we just assigned _root.bla.bla.bla to MSG  
        }  
    }  
    // conditions :  
    if (s<0) {  
        // checking for negative lengths  
        error("Please insert a Positive Beam Length value");  
        // calling our "error" function and setting the param "msg"  
    } else if (s == 0) {  
        // Nb. "=" assigns and "==" is a condition equality  
        error("\Zero\" is not a valid Beam Length");  
        // Nb. the code reads \" as "  
    } else if (s == "") {  
        error("Please insert a Beam Length value");  
    } else if (stiff == "") {  
        error("Please insert a Stiffness Ratio");  
    } else if (stiff < 0) {  
        error("Please insert a Positive Stiffness Ratio");  
    } else if (p == 0) {  
        error("Please insert a larger Load value");  
    } else if (p == "") {  
        error("Please insert a Load Value");  
    } else if (a == 0) {  
        error("Please insert a larger Load Position value");  
    } else if (a == "") {  
        error("Please insert a Load Position Value");  
    } else if (a >= s) {  
        error("Please reduce your Load Position Value");  
    } else if (a < 0) {  
        error("Please insert a Positive Load Position Value");  
    } else {  
        h = s-a;  
        leftspt = _root.twospan.leftsptswapper.leftspt;  
        if (leftspt == "pinned") {  
            femab = 0;  
            femba = (p*h*a*a)/(s*s) + (p*h*h*a)/(2*s*s);  
        } else if (leftspt == "fixed") {  
            femab = (-p*h*h*a)/(s*s);  
        }  
    }  
}
```

```
        femba = (p*h*a*a)/(s*s);
    }
    mmidleft = p*a*h/s;    }
}
```

**Code on the FEM button for the Fifth Loading Case (Combined Non-central Point Load and Partial UDL):**

```
on (release) {
  a = parseFloat (a);
  b = parseFloat (b);
  c = parseFloat (c);
  s = parseFloat (s);
  w = parseFloat (w);
  p = parseFloat (p);
  // funct controlling the error message popups for the input fields
  function error(msg) {
    // the function name is "error" and the parameter is "msg"
    with (_root.twospan.errorbox) {
      gotoAndStop(5);
      _root.twospan.errorbox.errormsg = msg;
      // we just assigned _root.bla.bla.bla to MSG
    }
  }
  // conditions :
  if (s<0) {
    // checking for negative lengths
    error("Please insert a Positive Beam Length value");
    // setting the param "msg" to "Please insert bla bla bla..."
  } else if (s == 0) {
    // checking if the length is zero...
    error("\\"Zero\\" is not a valid Beam Length");
    // Nb. the code reads  \\" as  "
  } else if (s == "") {
    error("Please insert a Beam Length value");
  } else if (stiff == "") {
    error("Please insert a Stiffness Ratio");
  } else if (stiff < 0) {
    error("Please insert a Positive Stiffness Ratio");
  } else if (w == 0) {
    error("Please insert a larger Load value");
  } else if (w == "") {
    error("Please insert a Load Value");
  } else if (p == 0) {
    error("Please insert a larger Load value");
  } else if (p == "") {
    error("Please insert a Load Value");
  } else if (a == 0) {
    error("Please insert a larger Load Position value");
  } else if (a == "") {
    error("Please insert a Load Position Value");
  } else if (a >= s) {
    error("Please reduce your Load Position Value");
  } else if (a < 0) {
    error("Please insert a Positive Load Position Value");
  } else if (b == "") {
```

```
        error("Please insert a Load Position Value");
    } else if (b < 0) {
        error("Please insert a Positive Load Position Value");
    } else if (b >= s) {
        error("Please reduce your Load Position Value");
    } else if (c == "") {
        error("Please insert a UDL Length Extent Value");
    } else if (c < 0) {
        error("Please insert a Positive UDL Length Value");
    } else if (c > s) {
        error("Please reduce your UDL Length Value");
    } else if (b+c > s) {
        error("Please reduce your Load Position and UDL Length
```

*all error checks are included here since this combined loading has all possible input fields for dimensions, stiffnesses*

```
Values");
    } else {
        d = b+c;
        g = s-d;
        e = c+g;
        h = s-a;
        leftspt = _root.twospan.leftsptswapper.leftspt;
        if (leftspt == "pinned") {
            femab = 0;
            femba = ((w)/(12*s*s))*(((d*d*d)*(4*s-3*d))-((b*b*b)*(4*s-3*b)))+(p*h*a*a)/(s*s)+
                ((w)/(24*s*s))*((e*e*e)*((4*s)-(3*e))-
                (g*g*g)*((4*s)-(3*g)))+(p*h*h*a)/(2*s*s);
        } else if (leftspt == "fixed") {
            femab = ((-w)/(12*s*s))*((e*e*e)*((4*s)-(3*e))-
                (g*g*g)*((4*s)-(3*g)))+(-p*h*h*a)/(s*s);
            femba = ((w)/(12*s*s))*(((d*d*d)*(4*s-3*d))-((b*b*b)*(4*s-3*b)))+(p*h*a*a)/(s*s);
        }
        rl = ((w*c)/(2*s))*(2*g + c);
        mmidleft = rl*(b + (rl/(2*w))) + p*a*h/s;    }
}
```

*the combined equations above use the principle of superposition. They consist of a sum of the equations for a non-central point load and those for a partial UDL superimposed on the same beam span:*

*The above 5 sets of code are repeated, with different labeling, for the Right Hand Side Span of the 2-span Module, the Left Hand Side Span of the 3-span Module and the Right Hand Side Span of the 3-span Module.*

*The Central Span of the 3-span Module also has a similar structure of code but has excluded the formulae for the propped cantilevers since it's edge supports are continuous and will therefore have Fixed End Moments.*

### **4.3 Two Span Beam Analysis**

The code for all Load Swapper Modules in the Two & Three Span Analyses is the same.  
(See Sect. 4.1.2 for Details)

#### **Code on the Two-Span ANALYZE button:**

```
on (release) {  
    // this is the function controlling the error message popups for  
the input fields  
    function error(msg) {  
        // the function name is "error" and the parameter is "msg"  
        with (_root.twospan.errorbox) {  
            gotoAndStop(5);  
            _root.twospan.errorbox.errormsg = msg;  
            // we just assigned _root.bla.bla.bla to MSG  
        }  
    }  
}
```

*this is almost the same error box function as was discussed earlier in the both the single and two-span modules. Difference is in the naming of the text fields to be called to the error function and the descriptions to the user of what exactly the wrong input is.*

```
// conditions :  
if (_root.twospan.lsleft.s == "") {  
    error("Please insert the Left Span Length");  
} else if (_root.twospan.lsleft.s <= 0) {  
    error("Please insert a Positive Left Span Length");  
} else if (_root.twospan.lsright.s == "") {  
    error("Please insert the Right Span Length");  
} else if (_root.twospan.lsright.s <= 0) {  
    error("Please insert a Positive Right Span Length");  
} else if (_root.twospan.lsleft.femab == "" or  
_root.twospan.lsleft.femba == "") {  
    error("Please insert the all FEM values for the Left Span");  
} else if (_root.twospan.lsright.fembc == "" or  
_root.twospan.lsright.femcb == "") {  
    error("Please insert the all FEM values for the Right Span");  
} else {
```

*the error function and checks are complete. Now on to the main moment distribution process:*

```
// localizing span stiffnesses (from "stiff" to "iab" or "ibc")  
iab = parseFloat (_root.twospan.lsleft.stiff);  
ibc = parseFloat (_root.twospan.lsright.stiff);  
// localizing dimensions & loadings from the LoadSwap Modules  
stiffleft = _root.twospan.lsleft.stiff;  
aleft = _root.twospan.lsleft.a + " m";  
bleft = _root.twospan.lsleft.b + " m";  
cleft = _root.twospan.lsleft.c + " m";  
s2left = (_root.twospan.lsleft.s)/2 + " m";  
pleft = _root.twospan.lsleft.p + " kN";
```

```
wleft = _root.twospan.lsleft.w + " kN/m";
lcleft = _root.twospan.lsleft.lcleft;
// .. and for the RHS
stiffright = _root.twospan.lsrigh.stiff;
aright = _root.twospan.lsrigh.a + " m";
bright = _root.twospan.lsrigh.b + " m";
cright = _root.twospan.lsrigh.c + " m";
s2right = (_root.twospan.lsrigh.s)/2 + " m";
pright = _root.twospan.lsrigh.p + " kN";
wright = _root.twospan.lsrigh.w + " kN/m";
lcrigh = _root.twospan.lsrigh.lcrigh;
// localizing the span lengths
sab = parseFloat (_root.twospan.lsleft.s);
sbc = parseFloat (_root.twospan.lsrigh.s);
// localizing the end support conditions
leftspt = _root.twospan.leftsptswapper.leftspt;
rightspt = _root.twospan.rightsptswapper.rightspt;
// localizing the FEMs
femab = parseFloat (_root.twospan.lsleft.femab);
femba = parseFloat (_root.twospan.lsleft.femba);
fembc = parseFloat (_root.twospan.lsrigh.fembc);
femcb = parseFloat (_root.twospan.lsrigh.femcb);
```

*the basic function of the chunk of code above is to pick the inputted and calculated values from the 6 load swapper modules (Discussed in Sect. 3.2), and to localize them in this code so as to avoid going to and fro every few lines.*

*the parseFloat (); function converts any string datatype to a number datatype. This is extremely important to avoid gross errors:*

*Eg. If not converted to number datatypes, 2 + 3 would yield “23” since it reads the values as letters rather than numbers.*

```
// calculating the stiffness factors (K) for each span
if (leftspt == "pinned") {
    kab = 0.75*(iab/sab);
} else if (leftspt == "fixed") {
    kab = (iab/sab);
    // note the reduction to 0.75 when an end support is pinned
    // but remains I/L when fixed
```

*we are now calculating the stiffness factors (K) as  $K = I/L$ . as is mentioned in the // comments, a reduction of  $\frac{3}{4}$  in the stiffness value occurs if an end support is pinned.*

```
}
if (rightspt == "pinned") {
    kbc = 0.75*(ibc/sbc);
} else if (rightspt == "fixed") {
    kbc = (ibc/sbc);
    // note the reduction to 0.75 when an end support is pinned
    // remains I/L when fixed
}
// summing stiffnesses at each end support B
sumkb = kab+kbc;
```

*the stiffness values of the spans meeting at one support are summed to enable the distribution factors (next step) to be calculated*

```
// now to calculate the distribution factors for each end
if (leftspt == "pinned") {
    dfab = 0;
} else if (leftspt == "fixed") {
    dfab = 1;
    // basic condition of (1) if fixed
    // no distribution mmts (0) to a pinned support
```

*rule of thumb: no moments can be distributed to a pinned support and the DF at a fixed end is always 1.*

```
}
dfba = kab/sumkb;
dfbc = kbc/sumkb;
if (rightspt == "pinned") {
    dfcb = 0;
} else if (rightspt == "fixed") {
    dfcb = 1;
    // basic condition of (1) if fixed
    // no distribution mmts (0) to a pinned support
}
```

*continuation of calculating the Distribution Factors.*

*The actual equation is:*

$$DF_{AB} = \frac{K_{AB}}{\sum K_B = K_{AB} + K_{BC}}$$

```
// now calculating Distribution Moments
dmba01 = -(femba+fembc)*dfba;
dmbc01 = -(femba+fembc)*dfbc;
```

*the distribution moments can be calculated as minus the sum of the fixed end moments meeting at one support multiplied by the directional distribution factor.*

```
// and now for the Carry Over Moments
// can only carry over to a Fixed Support
if (leftspt == "pinned") {
    comab01 = 0;
    femab = 0;
} else if (leftspt == "fixed") {
    comab01 = 0.5*dmba01;
}
```

*states that if a support is pinned, no moments can be carried over to it. Otherwise, if fixed, a half of the distributed moment (with the same sign) is carried over to the net support.*

```
if (rightspt == "pinned") {
```

```
comcb01 = 0;  
femcb = 0;
```

*note the last line that sets the FEM to zero if pinned, this is to avoid a situation where the user intentionally sets a Fixed End Moment, other than zero. This line ensures that the FEM value does not find its way into the summation as a final moment at a pinned end.*

```
} else if (rightspt == "fixed") {  
    comcb01 = 0.5*dmbc01;  
}
```

*since this is a two span beam, only one distribution and carry over is necessary.*

```
// the Final Moment = the sum of column FEMs, DMs and COMs  
mab = femab + comab01;  
mba = femba + dmba01;  
mbc = fembc + dmbc01;  
mcb = femcb + comcb01;
```

*the moment distribution process is complete. Next, the midspan moments are calculated making use of the principle of superposition.*

```
// calculating the MID-SPAN MOMENTS:  
// first, localizing the midspan mmts from the input stage  
mmidleft = _root.twospan.lsleft.mmidleft;  
mmidright = _root.twospan.lsrigh.mmidright;  
// next, calculating the avg. top-span moment from the end mmts  
// making end mmt values +ve (for the following calculation)  
pmab = mab;  
if (mab < 0) {  
    pmab = -1 * mab;  
}  
pmba = mba;  
if (mba < 0) {  
    pmba = -1 * mba;  
}  
pmbc = mbc;  
if (mbc < 0) {  
    pmbc = -1 * mbc;  
}  
pmcb = mcb;  
if (mcb < 0) {  
    pmcb = -1 * mcb;  
}  
// calculating the average top mmts  
mtopleft = (pmab+pmba)/2;  
mtopright = (pmbc+pmcb)/2;  
// calculating midspan mmts by subtracting from avg. top mmts  
mmidleft = mtopleft - mmidleft;  
mmidright = mtopright - mmidright;
```

*as can be observed from the // comments, the midspan moment was calculated from the superposition of the end support moments and the mid moment being simply supported as was calculated back in the load swap modules.*

```
// rounding off the MMT values to 3.d.p. for display
```

```
mabval = Math.round(mab * 1000) / 1000 + " kN";  
mbaval = Math.round(mba * 1000) / 1000 + " kN";  
mbcval = Math.round(mbc * 1000) / 1000 + " kN";  
mcbval = Math.round(mcb * 1000) / 1000 + " kN";  
mmidleftval = Math.round(mmidleft * 1000) / 1000 + " kN";  
mmidrightval = Math.round(mmidright * 1000) / 1000 + " kN";
```

*accuracy beyond three decimal places is not required. The moments are thus rounded off to 3.d.p and the respective units added to the values for display on the BMDs.*

```
        // sending the playhead to KF05 after completing the MD Process  
        gotoAndStop(5);  
    }  
}
```

### **Code at the Results Display Stage for the Two Span Analysis:**

```
// adding the dimensions to the lengths for display purposes  
sabm = sab + " m";  
sbcm = sbc + " m";  
// now altering the support displayed at the results stage depending on  
that selected initially  
if (leftspt == "pinned") {  
    with (_root.twospan.sptresultleft) {  
        gotoAndStop(1);  
    }  
} else if (leftspt == "fixed") {  
    with (_root.twospan.sptresultleft) {  
        gotoAndStop(2);  
    }  
}  
if (rightspt == "pinned") {  
    with (_root.twospan.sptresultright) {  
        gotoAndStop(1);  
    }  
} else if (rightspt == "fixed") {  
    with (_root.twospan.sptresultright) {  
        gotoAndStop(2);  
    }  
}  
}  
stop();
```

*the above section of code basically initializes what types of end supports the user has chosen as well as the type of loading so that the next section picks this information and displays the appropriate Bending Moment Diagram.*

### **Code on the BMD Sketch Swapper on the Left Span:**

```
onClipEvent (load) {  
    mab = _root.twospan.mabval;  
    mba = _root.twospan.mbaval;  
    mmidleft = _root.twospan.mmidleftval;  
    lclef = parseFloat (_root.twospan.lclef);  
    if (_root.twospan.leftspt == "pinned") {
```

```
        with (_root.twospan.bmdleft) {
            gotoAndStop(lcleft);
        }
    } else if (_root.twospan.leftspt == "fixed") {
        with (_root.twospan.bmdleft) {
            gotoAndStop(lcleft + 10);
        }
    }
}
```

*these two sections of code (above & Below) controlling the BMD diagrams to be displayed basically collect information about the load type and support conditions and using that information, are able to call up and display the appropriate sketch.*

### **Code on the BMD Sketch Swapper on the Right Span:**

```
onClipEvent (load) {
    mbc = _root.twospan.mbcval;
    mcb = _root.twospan.mcbval;
    mmidright = _root.twospan.mmidrightval;
    lcrightright = parseFloat (_root.twospan.lcrightright);
    if (_root.twospan.rightspt == "pinned") {
        with (_root.twospan.bmdright) {
            gotoAndStop(lcrightright);
        }
    } else if (_root.twospan.rightspt == "fixed") {
        with (_root.twospan.bmdright) {
            gotoAndStop(lcrightright + 10);
        }
    }
}
```

### **Code on the RC Design Button:**

```
on (release) {
    // function to calculate the maximum value in any array specified
    // to be mainly used in the transition between ANALYSIS and DESIGN
    // i.e. to use the maximum moment in the beam as the Design Moment
    function maxInArray(checkArray) {
        // function name = "maxInArray", parameter = "checkArray"
        var maxVal = - Number.MAX_VALUE;
        for (var i = 0; i < checkArray.length; i++) {
            maxVal = Math.max(checkArray[i], maxVal);
        }
        return maxVal;
    }
}
```

*the above function calculates the maximum value in any array specified later on (to be used in determining the maximum moments and span lengths) to be mainly used in the transition between ANALYSIS and DESIGN*

```
// removing all the "kN" from the strings & converting to numbers
ra = parseFloat (mabval);
rb = parseFloat (mbaval);
rc = parseFloat (mbcval);
rd = parseFloat (mcbval);
re = parseFloat (mmidleftval);
rf = parseFloat (mmidrightval);
// creating a funct. called modulus to transform any -ve no. to +ve
function modulus(r) {
  if (r < 0) {
    return (-1)*(r);
  } else if (r >= 0) {
    return r;
  }
}
```

*this function allows any set numerical datatype in a array to be converted from a negative value to a positive integer. It first checks if the number is less than 0, if it is negative, it will be multiplied by -1 so as to make it positive.*

```
// transforming all MMTs to +ve (only for the maxInArray Functions)
ra = modulus(ra);
rb = modulus(rb);
rc = modulus(rc);
rd = modulus(rd);
re = modulus(re);
rf = modulus(rf);
```

*above, new variables have been calculated to contain all the moment values that have been converted to positive integers. These values are converted to positive integers so as not to create any bias and exclude a negative moment from being selected as the design moment if it does, in fact have the maximum value as a modulus.*

```
_root.origin = "analysis";
_root.store.m = maxInArray([ra, rb, rc, rd, re, rf]);
_root.store.s = maxInArray([sab, sbc]);
_root.store.beamtype = "Continuous";
```

*the design moment, beam length and beam type have just been evaluated and sent to the \_root.storage to be picked up later on by the RC Design module.*

```
with (_root) {
  gotoAndStop(115);
  with (_root.design) {
    gotoAndStop(2);
  }
}
```

*the last bit of code sends the program to the Section in the RC Module that picks up the design parameters from the temporary storage.*

#### 4.4 Three Span Beam Analysis

The code for all Load Swapper Modules in the Two & Three Span Analyses is the same. (See Sect. 4.1.2 for Details)

##### Code for the Three-Span ANALYZE Button:

*Most of the comments here will borrow from those already encountered in the two-span discussion; this is because the code sections that carry out the moment distributions for the two and three span beams are quite similar. The only difference is in the number of iterations and the extra span (and support). Otherwise, the code structure and sequence is exactly the same.*

```
on (release) {
  // this is the function controlling the error message popups for
  the input fields
  function error(msg) {
    // the function name is "error" and the parameter is "msg"
    with (_root.threespan.errorbox) {
      gotoAndStop(5);
      _root.threespan.errorbox.errormsg = msg;
      // we just assigned _root.bla.bla.bla to MSG
    }
  }
}
```

*this is almost the same error box function as was discussed earlier in the both the single and two -span modules. Difference is in the naming of the text fields to be called to the error function and the descriptions to the user of what exactly the wrong input is.*

```
// conditions :
if (_root.threespan.lsleft.s == "") {
  error("Please insert the Left Span Length");
} else if (_root.threespan.lsleft.s <= 0) {
  error("Please insert a Positive Left Span Length");
} else if (_root.threespan.lsmid.s == "") {
  error("Please insert the Middle Span Length");
} else if (_root.threespan.lsmid.s <= 0) {
  error("Please insert a Positive Middle Span Length");
} else if (_root.threespan.lsrigh.s == "") {
  error("Please insert the Right Span Length");
} else if (_root.threespan.lsrigh.s <= 0) {
  error("Please insert a Positive Right Span Length");
} else if (_root.threespan.lsleft.femab == "" or
  _root.threespan.lsleft.femba == "") {
  error("Please insert the all FEM values for the Left Span");
} else if (_root.threespan.lsmid.fembc == "" or
  _root.threespan.lsmid.femcb == "") {
  error("Please insert the all FEM values for the Middle Span");
}
```

```
} else if (_root.threespan.lsrigh.femcd == "" or
           _root.threespan.lsrigh.femdc == "") {
    error("Please insert the all FEM values for the Right Span");
} else {
```

*the error function and checks are complete. Now on to the main moment distribution process:*

```
// LOCALIZING: dimensions & loadings from the LoadSwap
// left span:
stiffleft = _root.threespan.lsleft.stiff;
aleft = _root.threespan.lsleft.a + " m";
bleft = _root.threespan.lsleft.b + " m";
cleft = _root.threespan.lsleft.c + " m";
s2left = (_root.threespan.lsleft.s)/2 + " m";
pleft = _root.threespan.lsleft.p + " kN";
wleft = _root.threespan.lsleft.w + " kN/m";
lcleft = _root.threespan.lsleft.lcleft;
// mid span:
stiffmid = _root.threespan.lsmid.stiff;
amid = _root.threespan.lsmid.a + " m";
bmid = _root.threespan.lsmid.b + " m";
cmid = _root.threespan.lsmid.c + " m";
s2mid = (_root.threespan.lsmid.s)/2 + " m";
pmid = _root.threespan.lsmid.p + " kN";
wmid = _root.threespan.lsmid.w + " kN/m";
lcmid = _root.threespan.lsmid.lcmid;
// right span:
stiffright = _root.threespan.lsrigh.stiff;
aright = _root.threespan.lsrigh.a + " m";
bright = _root.threespan.lsrigh.b + " m";
cright = _root.threespan.lsrigh.c + " m";
s2right = (_root.threespan.lsrigh.s)/2 + " m";
pright = _root.threespan.lsrigh.p + " kN";
wright = _root.threespan.lsrigh.w + " kN/m";
lcrigh = _root.threespan.lsrigh.lcrigh;
// .. end of localizing for display
// now localizing for Mmt Distribution
// localizing span stiffnesses (from "stiff" to "iab" or "ibc")
iab = parseFloat (_root.threespan.lsleft.stiff);
ibc = parseFloat (_root.threespan.lsmid.stiff);
icd = parseFloat (_root.threespan.lsrigh.stiff);
// localizing the span lengths
sab = parseFloat (_root.threespan.lsleft.s);
sbc = parseFloat (_root.threespan.lsmid.s);
scd = parseFloat (_root.threespan.lsrigh.s);
// localizing the end support conditions
leftspt = _root.threespan.leftsptswapper.leftspt;
rightspt = _root.threespan.rightsptswapper.rightspt;
// localizing the FEMs
femab = parseFloat (_root.threespan.lsleft.femab);
femba = parseFloat (_root.threespan.lsleft.femba);
fembc = parseFloat (_root.threespan.lsmid.fembc);
femcb = parseFloat (_root.threespan.lsmid.femcb);
femcd = parseFloat (_root.threespan.lsrigh.femcd);
femdc = parseFloat (_root.threespan.lsrigh.femdc);
```

*the basic function of the chunk of code above is to pick the inputted and calculated values from the 6 load swapper modules (Discussed in Sect. 3.2), and to localize them in this code so as to avoid going to and fro every few lines.*

*the parseFloat (); function converts any string datatype to a number datatype. This is extremely important to avoid gross errors:*

*Eg. If not converted to number datatypes, 2 + 3 would yield "23" since it reads the values as letters rather than numbers.*

```
// calculating the STIFFNESS FACTORS (K) for each span
// depends on support conditions
if (leftspt == "pinned") {
    kab = 0.75*(iab/sab);
} else if (leftspt == "fixed") {
    kab = (iab/sab);
    // note the reduction to 0.75 when an end support is pinned
    // remains I/L when fixed
```

*we are now calculating the stiffness factors (K) as  $K = I/L$ . as is mentioned in the // comments, a reduction of  $\frac{3}{4}$  in the stiffness value occurs if an end support is pinned.*

```
}
kbc = (ibc/sbc);
if (rightspt == "pinned") {
    kcd = 0.75*(ibc/sbc);
} else if (rightspt == "fixed") {
    kcd = (ibc/sbc);
}

// summing the stiffnesses at the supports B and C
sumkb = kab+kbc;
sumkc = kbc+kcd;
```

*the stiffness values of the spans meeting at one support are summed to enable the distribution factors (next step) to be calculated*

```
// now to calculate the DISTRIBUTION FACTORS for each end
if (leftspt == "pinned") {
    dfab = 0;
} else if (leftspt == "fixed") {
    dfab = 1;
    // basic condition of (1) if fixed
    // no distribution mmts (0) to a pinned support
}
```

*rule of thumb: no moments can be distributed to a pinned support and the DF at a fixed end is always 1.*

```
dfba = kab/sumkb;  
dfbc = kbc/sumkb;  
dfcb = kbc/sumkc;  
dfcd = kcd/sumkc;
```

*continuation of calculating the Distribution Factors.  
The actual equation is:*

$$DF_{AB} = \frac{K_{AB}}{\sum K_B = K_{AB} + K_{BC}}$$

```
if (rightspt == "pinned") {  
    dfdc = 0;  
} else if (rightspt == "fixed") {  
    dfdc = 1;  
    // basic condition of (1) if fixed  
    // no distribution mmts (0) to a pinned support  
}
```

*the moment distribution iterations begin here:*

```
// 1st ITERATION  
// now calculating DISTRIBUTION MOMENTS  
dmab01 = 0;  
dmba01 = -(femba+fembc)*dfba;  
dmbc01 = -(femba+fembc)*dfbc;  
dmcb01 = -(femcb+femcd)*dfcb;  
dmcd01 = -(femcb+femcd)*dfcd;  
dmdb01 = 0;
```

*the distribution moments can be calculated as minus the sum of the fixed end moments meeting at one support multiplied by the directional distribution factor.*

```
// and now for the CARRY OVER MOMENTS  
//(can only carry over to a Fixed Support)  
if (leftspt == "pinned") {  
    comab01 = 0;  
    femab = 0;  
} else if (leftspt == "fixed") {  
    comab01 = 0.5*dmba01;  
}  
comba01 = 0;  
combc01 = 0.5*dmcb01;  
comcb01 = 0.5*dmbc01;  
comcd01 = 0;  
if (rightspt == "pinned") {  
    comdc01 = 0;  
    femdc = 0;  
} else if (rightspt == "fixed") {  
    comdc01 = 0.5*dmcd01;  
}
```

*the previous section calculates the carry over moments in the first iteration. If a support is pinned, no moments can be carried over to it. Otherwise, if fixed, a half of the distributed moment (with the same sign) is carried over to the net support.*

*note the last line that sets the FEM to zero if pinned, this is to avoid a situation where the user intentionally sets a Fixed End Moment, other than zero. This line ensures that the FEM value does not find its way into the summation as a final moment at a pinned end.*

```
// 2nd ITERATION
dmab02 = 0;
dmba02 = -(comba01+combc01)*dfba;
dmcb02 = -(comba01+combc01)*dfbc;
dmcb02 = -(comcb01+comcd01)*dfcb;
dmcd02 = -(comcb01+comcd01)*dfcd;
dmcd02 = 0;
if (leftspt == "pinned") {
    comab02 = 0;
} else if (leftspt == "fixed") {
    comab02 = 0.5*dmba02;
}
comba02 = 0;
combc02 = 0.5*dmcb02;
comcb02 = 0.5*dmcb02;
comcd02 = 0;
if (rightspt == "pinned") {
    comdc02 = 0;
} else if (rightspt == "fixed") {
    comdc02 = 0.5*dmcd02;
}
}
```

*in a similar pattern, 12 iterations follow*

```
// 3rd ITERATION
dmab03 = 0;
dmba03 = -(comba02+combc02)*dfba;
dmcb03 = -(comba02+combc02)*dfbc;
dmcb03 = -(comcb02+comcd02)*dfcb;
dmcd03 = -(comcb02+comcd02)*dfcd;
dmcd03 = 0;
if (leftspt == "pinned") {
    comab03 = 0;
} else if (leftspt == "fixed") {
    comab03 = 0.5*dmba03;
}
comba03 = 0;
combc03 = 0.5*dmcb03;
comcb03 = 0.5*dmcb03;
comcd03 = 0;
if (rightspt == "pinned") {
    comdc03 = 0;
} else if (rightspt == "fixed") {
    comdc03 = 0.5*dmcd03;
}
}
```

```
// 4th ITERATION
dmab04 = 0;
dmba04 = -(comba03+combc03)*dfba;
dmbc04 = -(comba03+combc03)*dfbc;
dmcb04 = -(comcb03+comcd03)*dfcb;
dmcd04 = -(comcb03+comcd03)*dfcd;
dmdc04 = 0;
if (leftspt == "pinned") {
    comab04 = 0;
} else if (leftspt == "fixed") {
    comab04 = 0.5*dmba04;
}
comba04 = 0;
combc04 = 0.5*dmcb04;
comcb04 = 0.5*dmbc04;
comcd04 = 0;
if (rightspt == "pinned") {
    comdc04 = 0;
} else if (rightspt == "fixed") {
    comdc04 = 0.5*dmcd04;
}

// 5th ITERATION
dmab05 = 0;
dmba05 = -(comba04+combc04)*dfba;
dmbc05 = -(comba04+combc04)*dfbc;
dmcb05 = -(comcb04+comcd04)*dfcb;
dmcd05 = -(comcb04+comcd04)*dfcd;
dmdc05 = 0;
if (leftspt == "pinned") {
    comab05 = 0;
} else if (leftspt == "fixed") {
    comab05 = 0.5*dmba05;
}
comba05 = 0;
combc05 = 0.5*dmcb05;
comcb05 = 0.5*dmbc05;
comcd05 = 0;
if (rightspt == "pinned") {
    comdc05 = 0;
} else if (rightspt == "fixed") {
    comdc05 = 0.5*dmcd05;
}

// 6th ITERATION
dmab06 = 0;
dmba06 = -(comba05+combc05)*dfba;
dmbc06 = -(comba05+combc05)*dfbc;
dmcb06 = -(comcb05+comcd05)*dfcb;
dmcd06 = -(comcb05+comcd05)*dfcd;
dmdc06 = 0;
if (leftspt == "pinned") {
    comab06 = 0;
} else if (leftspt == "fixed") {
    comab06 = 0.5*dmba06;
}
}
```

## Computer Analysis & Reinforced Concrete Design of Beams

---

```
comba06 = 0;
combc06 = 0.5*dmcb06;
comcb06 = 0.5*dmbc06;
comcd06 = 0;
if (rightspt == "pinned") {
    comdc06 = 0;
} else if (rightspt == "fixed") {
    comdc06 = 0.5*dmcd06;
}

// 7th ITERATION
dmab07 = 0;
dmba07 = -(comba06+combc06)*dfba;
dmbc07 = -(comba06+combc06)*dfbc;
dmcb07 = -(comcb06+comcd06)*dfcb;
dmcd07 = -(comcb06+comcd06)*dfcd;
dmdc07 = 0;
if (leftspt == "pinned") {
    comab07 = 0;
} else if (leftspt == "fixed") {
    comab07 = 0.5*dmba07;
}
comba07 = 0;
combc07 = 0.5*dmcb07;
comcb07 = 0.5*dmbc07;
comcd07 = 0;
if (rightspt == "pinned") {
    comdc07 = 0;
} else if (rightspt == "fixed") {
    comdc07 = 0.5*dmcd07;
}

// 8th ITERATION
dmab08 = 0;
dmba08 = -(comba07+combc07)*dfba;
dmbc08 = -(comba07+combc07)*dfbc;
dmcb08 = -(comcb07+comcd07)*dfcb;
dmcd08 = -(comcb07+comcd07)*dfcd;
dmdc08 = 0;
if (leftspt == "pinned") {
    comab08 = 0;
} else if (leftspt == "fixed") {
    comab08 = 0.5*dmba08;
}
comba08 = 0;
combc08 = 0.5*dmcb08;
comcb08 = 0.5*dmbc08;
comcd08 = 0;
if (rightspt == "pinned") {
    comdc08 = 0;
} else if (rightspt == "fixed") {
    comdc08 = 0.5*dmcd08;
}

// 9th ITERATION
```

```
dmab09 = 0;
dmba09 = -(comba08+combc08)*dfba;
dmbc09 = -(comba08+combc08)*dfbc;
dmcb09 = -(comcb08+comcd08)*dfcb;
dmcd09 = -(comcb08+comcd08)*dfcd;
dmdc09 = 0;
if (leftspt == "pinned") {
    comab09 = 0;
} else if (leftspt == "fixed") {
    comab09 = 0.5*dmba09;
}
comba09 = 0;
combc09 = 0.5*dmcb09;
comcb09 = 0.5*dmbc09;
comcd09 = 0;
if (rightspt == "pinned") {
    comdc09 = 0;
} else if (rightspt == "fixed") {
    comdc09 = 0.5*dmcd09;
}
```

*by now, the iterations have eliminated inconsistencies between support moments up to the first decimal place, even for a 3-figure moment.*

*The following iterations exist to refine the moment further so that when the final moments are rounded off to the nearest 3.d.p., the end support moments from two different spans meeting at the same support are the same.*

```
// 10th ITERATION
dmab10 = 0;
dmba10 = -(comba09+combc09)*dfba;
dmbc10 = -(comba09+combc09)*dfbc;
dmcb10 = -(comcb09+comcd09)*dfcb;
dmcd10 = -(comcb09+comcd09)*dfcd;
dmdc10 = 0;
if (leftspt == "pinned") {
    comab10 = 0;
} else if (leftspt == "fixed") {
    comab10 = 0.5*dmba10;
}
comba10 = 0;
combc10 = 0.5*dmcb10;
comcb10 = 0.5*dmbc10;
comcd10 = 0;
if (rightspt == "pinned") {
    comdc10 = 0;
} else if (rightspt == "fixed") {
    comdc10 = 0.5*dmcd10;
}

// 11th ITERATION
dmab11 = 0;
dmba11 = -(comba10+combc10)*dfba;
dmbc11 = -(comba10+combc10)*dfbc;
```

```
dmcb11 = -(comcb10+comcd10)*dfcb;
dmcd11 = -(comcb10+comcd10)*dfcd;
dmcd11 = 0;
if (leftspt == "pinned") {
    comab11 = 0;
} else if (leftspt == "fixed") {
    comab11 = 0.5*dmba11;
}
comba11 = 0;
combc11 = 0.5*dmcb11;
comcb11 = 0.5*dmbc11;
comcd11 = 0;
if (rightspt == "pinned") {
    comdc11 = 0;
} else if (rightspt == "fixed") {
    comdc11 = 0.5*dmcd11;
}

// 12th ITERATION
dmab12 = 0;
dmba12 = -(comba11+combc11)*dfba;
dmbc12 = -(comba11+combc11)*dfbc;
dmcb12 = -(comcb11+comcd11)*dfcb;
dmcd12 = -(comcb11+comcd11)*dfcd;
dmcd12 = 0;
if (leftspt == "pinned") {
    comab12 = 0;
} else if (leftspt == "fixed") {
    comab12 = 0.5*dmba12;
}
comba12 = 0;
combc12 = 0.5*dmcb12;
comcb12 = 0.5*dmbc12;
comcd12 = 0;
if (rightspt == "pinned") {
    comdc12 = 0;
} else if (rightspt == "fixed") {
    comdc12 = 0.5*dmcd12;
}
// -- iterations complete -
```

*The next section calculates the Final Support moments as the sum of the fixed end moments, distribution moments and carry over moments on each side of a support.*

```
// FINAL MOMENT = sum of the FEMs, DMs and COMs in each column
mab = femab + dmab01 + comab01 + dmab02+ comab02 + dmab03 +
      comab03 + dmab04 + comab04 + dmab05 + comab05 + dmab06 +
      comab06 + dmab07 + comab07 + dmab08 + comab08 + dmab09 +
      comab09 + dmab10 + comab10 + dmab11 + comab11 + dmab12 +
      comab12;
```

## Computer Analysis & Reinforced Concrete Design of Beams

---

```
mba = femba + dmba01 + comba01 + dmba02+ comba02 + dmba03 +  
      comba03 + dmba04 + comba04 + dmba05 + comba05 + dmba06 +  
      comba06 + dmba07 + comba07 + dmba08 + comba08 + dmba09 +  
      comba09 + dmba10 + comba10 + dmba11 + comba11 + dmba12 +  
      comba12;  
mbc = fembc + dmcb01 + combc01 + dmcb02+ combc02 + dmcb03 +  
      combc03 + dmcb04 + combc04 + dmcb05 + combc05 + dmcb06 +  
      combc06 + dmcb07 + combc07 + dmcb08 + combc08 + dmcb09 +  
      combc09 + dmcb10 + combc10 + dmcb11 + combc11 + dmcb12 +  
      combc12;  
mcb = femcb + dmcb01 + comcb01 + dmcb02+ comcb02 + dmcb03 +  
      comcb03 + dmcb04 + comcb04 + dmcb05 + comcb05 + dmcb06 +  
      comcb06 + dmcb07 + comcb07 + dmcb08 + comcb08 + dmcb09 +  
      comcb09 + dmcb10 + comcb10 + dmcb11 + comcb11 + dmcb12 +  
      comcb12;  
mcd = femcd + dmcd01 + comcd01 + dmcd02+ comcd02 + dmcd03 +  
      comcd03 + dmcd04 + comcd04 + dmcd05 + comcd05 + dmcd06 +  
      comcd06 + dmcd07 + comcd07 + dmcd08 + comcd08 + dmcd09 +  
      comcd09 + dmcd10 + comcd10 + dmcd11 + comcd11 + dmcd12 +  
      comcd12;  
mdc = femdc + dmcd01 + comdc01 + dmcd02+ comdc02 + dmcd03 +  
      comdc03 + dmcd04 + comdc04 + dmcd05 + comdc05 + dmcd06 +  
      comdc06 + dmcd07 + comdc07 + dmcd08 + comdc08 + dmcd09 +  
      comdc09 + dmcd10 + comdc10 + dmcd11 + comdc11 + dmcd12 +  
      comdc12;
```

*The moment distribution process is complete.*

*Next, the midspan moments are calculated making use of the principle of superposition.*

```
// calculating the MID-SPAN MOMENTS:  
// first, localizing the midspan mmts from the input stage  
mmidleft = _root.threespan.lsleft.mmidleft;  
mmidmid = _root.threespan.lsmid.mmidmid;  
mmidright = _root.threespan.lsrigh.mmidright;  
// next, calculating the avg. top-span moment from the end mmts  
// making end mmt values + ve for the following calculation  
pmab = mab;  
if (mab < 0) {  
    pmab = -1 * mab;  
}  
pmba = mba;  
if (mba < 0) {  
    pmba = -1 * mba;  
}  
pmbc = mbc;  
if (mbc < 0) {  
    pmbc = -1 * mbc;  
}  
pmcb = mcb;  
if (mcb < 0) {  
    pmcb = -1 * mcb;  
}  
pmcd = mcd;  
if (mcd < 0) {  
    pmcd = -1 * mcd;  
}  
pmdc = mdc;
```

```
if (mdc < 0) {
    pmdc = -1 * mdc;
}

// calculating the average top mmts
mtopleft = (pmab+pmba)/2;
mtopmid = (pmbc+pmcb)/2;
mtopright = (pmcd+pmdc)/2;

// calculating midspan mmts by subtracting from avg. top mmts
mmidleft = mtopleft - mmidleft;
mmidmid = mtopmid - mmidmid;
mmidright = mtopright - mmidright;
```

*as can be observed from the // comments, the midspan moment was calculated from the superposition of the end support moments and the plane mid moment calculated back in the load swap modules.*

```
// Display Prep:
// rounding of the Final Mmt values to 3.d.p. and adding "kN"
mabval = Math.round(mab * 1000) / 1000 + " kN";
mbaval = Math.round(mba * 1000) / 1000 + " kN";
mbcval = Math.round(mbc * 1000) / 1000 + " kN";
mcbval = Math.round(mcb * 1000) / 1000 + " kN";
mcdval = Math.round(mcd * 1000) / 1000 + " kN";
mdcval = Math.round(mdc * 1000) / 1000 + " kN";
mmidleftval = Math.round(mmidleft * 1000) / 1000 + " kN";
mmidmidval = Math.round(mmidmid * 1000) / 1000 + " kN";
mmidrightval = Math.round(mmidright * 1000) / 1000 + " kN";
```

*accuracy beyond three decimal places is not required. The moments are thus rounded off to 3.d.p and the respective units added to the values for display on the BMDs.*

```
// sending the playhead to KF05 after completing the MD Process
gotoAndStop(5);
}
}
```

### **Code at the Results Display Stage for the Three Span Analysis:**

```
// adding the dimensions to the lengths for display purposes
sabm = sab + " m";
sbcm = sbc + " m";
scdm = scd + " m";
// now altering the support displayed at the results stage depending on
that selected initially
if (leftspt == "pinned") {
    with (_root.threespan.sptresultleft) {
        gotoAndStop(1);
    }
} else if (leftspt == "fixed") {
    with (_root.threespan.sptresultleft) {
        gotoAndStop(2);
    }
}
}
```

```
if (rightspt == "pinned") {
    with (_root.threespan.sptresultright) {
        gotoAndStop(1);
    }
} else if (rightspt == "fixed") {
    with (_root.threespan.sptresultright) {
        gotoAndStop(2);
    }
}
stop();
```

*the above section of code basically initializes what types of end supports the user has chosen as well as the type of loading so that the next section picks this information and displays the appropriate Bending Moment Diagram.*

*The next three sections of code control the BMD diagrams to be displayed. They basically collect information about the load type and support conditions and using that information, are able to call up and display the appropriate sketch.*

#### **Code on the BMD Sketch Swapper on the Left Span:**

```
onClipEvent (load) {
    mab = _root.threespan.mabval;
    mba = _root.threespan.mbaaval;
    mmidleft = _root.threespan.mmidleftval;
    lcleft = parseFloat (_root.threespan.lcleft);
    if (_root.threespan.leftspt == "pinned") {
        with (_root.threespan.bmdleft) {
            gotoAndStop(lcleft);
        }
    } else if (_root.threespan.leftspt == "fixed") {
        with (_root.threespan.bmdleft) {
            gotoAndStop(lcleft + 10);
        }
    }
}
```

#### **Code on the BMD Sketch Swapper on the Middle Span:**

```
onClipEvent (load) {
    mbc = _root.threespan.mbcval;
    mcb = _root.threespan.mcbval;
    mmidmid = _root.threespan.mmidmidval;
    lcmid = parseFloat (_root.threespan.lcmid);
    with (_root.threespan.bmdmid) {
        gotoAndStop(lcmid);
    }
}
```

### **Code on the BMD Sketch Swapper on the Right Span:**

```
onClipEvent (load) {
    mcd = _root.threespan.mcdval;
    mdc = _root.threespan.mdcval;
    mmidright = _root.threespan.mmidrightval;
    lcrighth = parseFloat (_root.threespan.lcrighth);
    if (_root.threespan.rightspt == "pinned") {
        with (_root.threespan.bmdright) {
            gotoAndStop(lcrighth);
        }
    } else if (_root.threespan.rightspt == "fixed") {
        with (_root.threespan.bmdright) {
            gotoAndStop(lcrighth + 10);
        }
    }
}
```

### **Code on the RC Design Button for the Transition between Analysis & Design:**

```
on (release) {
    // function to calculate the maximum value in any array specified
    // to be mainly used in the transition between ANALYSIS and DESIGN
    // i.e. to use the maximum moment in the beam as the Design Moment
    function maxInArray(checkArray) {
        // the function name= "maxInArray" & parameter= "checkArray"
        var maxVal = - Number.MAX_VALUE;
        for (var i = 0; i < checkArray.length; i++) {
            maxVal = Math.max(checkArray[i], maxVal);
        }
        return maxVal;
    }
}
```

*the above function calculates the maximum value in any array specified later on (to be used in determining the maximum moments and span lengths) to be mainly used in the transition between ANALYSIS and DESIGN*

```
// removing all the "kN" from the strings & converting to numbers
rg = parseFloat (mmidleftval);
rh = parseFloat (mmidmidval);
rk = parseFloat (mmidrightval);
ra = parseFloat (mabval);
rb = parseFloat (mbaval);
rc = parseFloat (mbcval);
rd = parseFloat (mcbval);
re = parseFloat (mcdval);
rf = parseFloat (mdcval);
```

*the next function allows any set numerical datatype in a array to be converted from a negative value to a positive integer. It first checks if the number is less than 0, if it is negative, it will be multiplied by -1 so as to make it positive.*

```
// creating a funct. called modulus to transform any -ve no. to +ve
function modulus(r) {
    if (r < 0) {
        return (-1)*(r);
    } else if (r >= 0) {
        return r;
    }
}

// transforming all the MMT values to +ve for the maxInArray Funct.
rg = modulus(rg);
rh = modulus(rh);
rk = modulus(rk);
ra = modulus(ra);
rb = modulus(rb);
rc = modulus(rc);
rd = modulus(rd);
re = modulus(re);
rf = modulus(rf);
```

*above, new variables have been calculated to contain all the moment values that have been converted to positive integers. These values are converted to positive integers so as not to create any bias by excluding a negative moment from being selected as the design moment if it does, in fact have the maximum value as a modulus*

```
_root.origin = "analysis";
_root.store.m = maxInArray([rg, rh, rk, ra, rb, rc, rd, re, rf]);
_root.store.s = maxInArray([sab, sbc, scd]);
_root.store.beamtype = "Continuous";
```

*the design moment, beam length and beam type have just been evaluated and sent to the \_root.storage to be picked up later on by the RC Design module.*

```
with (_root) {
    gotoAndStop(115);
    with (_root.design) {
        gotoAndStop(2);
    }
}
}
```

*this last section of code sends the program to the Section in the RC Module that picks up the design parameters from the temporary storage.*

## **4.5 Reinforced Concrete Beam Design Module**

### **Code on the First Keyframe of the RC Design Module**

```
if (_root.origin == "analysis") {
    gotoAndStop(2);
} else if (_root.origin == "user") {
    s = "";
    minput = "";
    stop();
}
```

*this initial script checks where the user came from. i.e. from the analysis modules or from the main screen. If the user came from the main screen, then the beam design parameters (effective length and design moment) are set to <empty>. If the user came from the analysis modules, the next section of code (below) is executed.*

### **Code on the Second Keyframe of the RC Design Module:**

```
minput = _root.store.m;
s = _root.store.s;
beamtype = _root.store.beamtype;
stop();
```

*the above code picks the design parameters from the temporary \_root.storage. The parameters had been placed there from the analysis modules.*

### **Code on the BEAM SIZING button:**

```
on (release) {
    // function controlling error message popups for the input fields
    function error(msg) {
        // the function name is "error" and the parameter is "msg"
        with (_root.design.errorbox) {
            gotoAndStop(5);
            _root.design.errorbox.errormsg = msg;
        }
    }
}
```

*the same error function used before; this time round, it is has been customized for the design module.*

```
// conditions :
if (_root.design.minput == "") {
    error("Please insert the Design Moment for the Beam");
} else if (_root.design.s == "") {
    error("Please insert the Effective Length of the Beam");
} else if (_root.design.s < 0) {
    error("Please insert a Positive Length Value for the Beam");
} else if (_root.design.fy == "") {
    error("Please insert the Steel Strength Value");
} else if (_root.design.fy < 0) {
```

```
    error("Please insert a Positive Steel Strength Value");
} else if (_root.design.fcu == "") {
    error("Please insert the Concrete Grade Value");
} else if (_root.design.fcu < 0) {
    error("Please insert a Positive Concrete Grade Value");
} else {
```

*the error checks are now complete.*

```
m = parseFloat(minput);
// Precaution: Converting the Design Moment to a Positive Value
if (m < 0) {
    m = -1 * m;
}
s = parseFloat(s);
fy = parseFloat(fy);
fcu = parseFloat(fcu);
```

*as a precaution the design moment has been converted to a positive value if the user has inputted it as negative. The other design parameters have been converted to numbers to avoid a situation where the computer reads the values as figures (e.g. 2 + 3 = "23")*

```
// specifying BS8110 Ratio Values depending on Beam Type
if (beamtype == "Simply Supported") {
    den = 12;
    ratio = 20;
} else if (beamtype == "Continuous") {
    den = 15;
    ratio = 26;
} else if (beamtype == "Cantilevered") {
    den = 6;
    ratio = 7;
}
```

*BS8110 suggested (l/d) and m.f.sizing ratios are set as shown above.*

```
// calculating the Effective Depth
// according to the Beam Type & Effective Length (BS8110)
d = (s*1000)/den;
```

*the effective depth has been calculated using the ratios set in the previous stage.*

```
// Checking the Minimum Depth for the Concrete Capacity
//(for a b:d ratio of 1:2)
dmin = Math.pow(((m*1000000)/(0.312*fcu)), (1/3));
if (d < dmin) {
```

*to ensure that the design moment can be resisted by the capacity of the concrete section sized;*

$$M = 0.156 f_{cu} b d^2$$

$$\text{also, } d = 2b$$

$$\therefore M = 0.156 f_{cu} b (2b)^2$$

*rearranging and equating in terms of b:*

$$b = \left( \frac{M}{0.624} \right)^{\frac{1}{3}}$$

*this limits the minimum effective depth (as was seen in the above code section) to the minimum allowable depth (for a b/d ratio of 1/2)*

```
    // increasing the initial effective depth
    // to meet this minimum + 10mm extra
    d = dmin + 10;
}
```

*if the concrete capacity check failed and the initially sized depth was actually smaller than the minimum allowable effective depth, the code sets a new effective depth as the minimum allowed, and adds 10mm as a factor of safety.*

```
// suggested beam breadth = 1/2 the effective depth
b = 0.5*d;
```

*this line resets the new breadth from the corrected effective depth.*

*The next step is the deflection check (via BS8110 Modification Factor):*

```
// calculation the M/bdd ratio
mbdd = (m*1000000)/(b*d*d);
// fs is assumed to be 5/8 of fy for M(design)= M(elastic)
// above assumption source: BS8110
fs = 5*fy/8;
```

*BS8110 has charts for determining the m.f. value given the steel strengths and the  $M/bd^2$  values. However, it also gives a way of actually calculating the modification factor under the assumption that: “for a continuous beam, if the percentage of redistribution is not known but the design ultimate moment at mid-span is obviously the same as or greater than the elastic ultimate moment, the stress  $f_s$  may be taken as  $5/8 f_y$ .”*

*The equation is:*

$$\text{Modification Factor} = \frac{(477 - f_s)}{120 \left( 0.9 + \frac{M}{bd^2} \right)} \leq 2.0$$

*where M is the design ultimate moment at the center of the span, or for a cantilever, at the support.*

```
// Modification Factor (BS8110):
mf = 0.55 + ((477 - fs) / (120 * (0.9 + mbdd)));
if (mf > 2) {
    // Limiting the m.f. to a maximum of 2.0
    mf = 2;
}
```

## Computer Analysis & Reinforced Concrete Design of Beams

---

```
// calculating the min eff. depth to limit excessive deflection  
ddef = (s*1000)/(ratio*mf);
```

*this follows from the rule:*

$$\frac{l}{d} \leq \text{ratio} \times mf$$

*next, if the deflection check fails and the effective depth is smaller than the minimum depth to limit deflection, a new effective depth is calculated by setting the minimum depth to this new depth and adding 10mm to it as a factor of safety.*

```
if (d < ddef) {  
    // increasing d to prevent deflection  
    // increasing d by 10mm more than the minimum  
    d = ddef + 10;  
}  
// readjusting breadth to meet final adjusted effective depth  
b = 0.5*d;
```

*the beam dimensions are now readjusted to meet any new depth created.*

*Final effective depth check:*

```
// calculating the min d for high Mmts  
// shear governs design i.e. K < 0.225  
dshr = Math.pow(((M*1000000)/(0.1125*fcu)), 1/3);
```

*this last check deals with the minimum effective depth of a beam just before shearing forces govern the design as opposed to bending.*

*In the equation:*

$$l_a = 0.5 + \sqrt{0.25 - \frac{K}{0.9}}$$

*the value of K needs to be lower than 0.225 to prevent a negative value from occurring within the square root.*

*Generally*      *If the value of K exceeds 0.156, compression reinforcement is required*  
                  *If the value of K exceeds 2.225, shear governs the design.*

*From the equation:*

$$K = \frac{M}{f_{cu} b d^2}$$

*rearranging and setting the max value of K as 0.225.*

*In terms of the effective depth, d:*

$$d_{\min} = \sqrt[3]{\frac{M}{0.1125 f_{cu}}}$$

```
if (d < dshr) {  
    // increasing d by 10mm more than that necessary  
    // to prevent K > 0.225  
    d = dshr + 10;  
    // readjusting breadth to meet final adjusted effective depth  
    b = 0.5*d;
```

*if this minimum value of d to prevent shearing forces from governing the design was not met, then a new value of the effective depth is calculated by assigning this minimum depth value to the new value and adding 10mm as a factor of safety.*

*Again, the breadth b, is recalculated as 1/2 the new effective depth value.*

*The beam dimensions are now either the suggested by the BS code (calculated from the ratios for beam types and span lengths), or the minimum dimensions that meet any or all of the following limitations:*

- *The concrete capacity of the section is not exceeded*
- *The deflection is limited to acceptable values (via the Modification Factor)*
- *The beam dimensions are large enough to prevent shear from governing the design.*

```
// Rounding off Calculated Beam Dimensions to the nearest 0.1mm  
dcalc = Math.round(d*10)/10;  
bcalc = Math.round(b*10)/10;
```

*the suggested minimum beam dimensions are rounded off to the nearest 0.1mm. In RC Design, this quite an accurate result.*

```
// rounding off Suggested Width & Effective Depth  
// to Nearest 10mm (upper 10mm limit)  
dceil = Math.ceil(dcalc/10)*10;  
bceil = Math.ceil(bcalc/10)*10;
```

*the practical dimensions are established by rounding the minimum dimensions to the upper 10mm limit. (in design, dimensions, areas and loads are always rounded upwards while material strengths are rounded downwards; for reasons of safety) i.e. 23mm is rounded up to 30mm rather than 20mm.*

```
// Sending the Playhead to the Beam Sizing Display Section  
mold = parseFloat(m);  
gotoAndStop(5);  
}  
}
```

*the program now moves to the beam sizing display section where the user has the option to alter the sizes recommended by the program.*

### **Code executed at the beginning of the Beam Sizing Display Stage:**

*The purpose of the following code is to prepare the figures obtained in the previous section for display, as well as further manipulation in forthcoming sections. It also calculates the beam self weight and superimposes this as a simply supported UDL moment to the initial design moment inputted by the user.*

```
minit = parseFloat(mold);  
s = parseFloat(s);  
dceil = parseFloat(dceil);  
bceil = parseFloat(bceil);  
c = parseFloat(c);  
t = parseFloat(t);
```

*in these first few lines, the values calculated and entered are converted to string datatypes to avoid errors in arithmetic operations.*

```
// displaying the dimensions on the figure  
bfig = bceil + " mm";  
dfig = dceil + " mm";  
  
// calculating the total beam height as D + Cover + Tolerance  
hfig = dceil + c + t + " mm";
```

*the initial cover and tolerance have been set to 30mm and 10mm respectively. The above expression calculates the total beam height as a sum of the effective depth, cover and tolerance.*

*The next step deals with increasing the design moment by the effects of the beam self weight.*

```
// calculating the additional UDL due to the Beam Self Weight  
h = dceil + c + t;  
wadd = 24*(h/1000)*(bceil/1000);
```

*we have just calculated the beam's self weight as a UDL. A density of 24kN/m<sup>3</sup> is used.*

```
// calculating the additional moment due to the self weight  
madd = wadd*s*s/8;
```

*the additional moment has been calculated as a UDL spanning along a simply supported beam:*

$$M_{add} = \frac{wl^2}{8}$$

*next,  $M_{add}$  is added to the initial design moment and the final mmt rounded off to 3.d.p.*

```
// adding this additional moment to the initial design moment  
mnew = minit + madd;  
mnew = Math.round(mnew*1000)/1000;  
stop();
```

### **Code on the RESIZE button:**

*This section of code is executed when the user clicks the resize button. It updates the dimensions, self weight, additional moment and final moments according to the new depth, breadth cover and tolerance values entered by the user.*

*The code follows the same format and performs the same tasks as the previous section.*

```
on (release) {
  minit = parseFloat(mold);
  s = parseFloat(s);
  dceil = parseFloat(dceil);
  bceil = parseFloat(bceil);
  c = parseFloat(c);
  t = parseFloat(t);

  // displaying the dimensions on the figure
  bfig = bceil + " mm";
  dfig = dceil + " mm";

  // calculating the total beam height as D + Cover + Tolerance
  hfig = dceil + c + t + " mm";

  // calculating the additional UDL due to the Beam Self Weight
  h = dceil + c + t;
  wadd = 24*(h/1000)*(bceil/1000);

  // calculating the additional moment due to the self weight
  madd = wadd*s*s/8;

  // adding this additional moment to the initial design moment
  mnew = minit + madd;
  mnew = Math.round(mnew*1000)/1000;
}
```

### **Code on the DESIGN REINFORCEMENT button:**

*After the user has complete the resizing of the beam (if he/she felt it was necessary), the following chunk of code is executed once the Design Reinforcement button s clicked. In summary, this section calculates the areas tension and compression steel required for the beam size and deign parameters specified.*

```
on (release) {
  // localizing the Variables and converting them to number values
  m = parseFloat (mnew);
  b = parseFloat (bceil);
  d = parseFloat (dceil);
  fy = parseFloat(fy);
  fcu = parseFloat(fcu);
}
```

*the common conversion to number datatypes takes place in the above lines.*

```
// calculating K  
k = (m*1000000)/(fcu*b*d*d);
```

*the value of K has been calculated as:*

$$K = \frac{M}{f_{cu} b d^2}$$

```
// calculating the lever arm  
la = 0.5+Math.sqrt(0.25-(k/0.9));  
// limiting 0.77 < la < 0.95  
if (la < 0.77) {  
    la = 0.77;  
}  
if (la > 0.95) {  
    la = 0.95;  
}
```

*BS8110 limits the lever arm value to: (0.77 < l<sub>a</sub> < 0.95)*

*The equation for the lever arm is given as follows:*

$$l_a = 0.5 + \sqrt{0.25 - \frac{K}{0.9}}$$

```
// calculating Z  
z = la*d;
```

*the dimension z, is essentially the distance between the effective components of the tension force and the compression force.*

*It is the value of the effective depth factored down by the lever arm value:*

$$z = l_a d$$

```
// calculating area of steel  
if (k <= 0.156) {  
    // if K < 0.156, then no compression reinforcement required  
    ascc = "Nominal (e.g. 2Y12)";  
    dispa = "2Y12";  
    dispcc = "";  
    ast = (m*1000000)/(0.87*fy*z);  
    ast = Math.round(ast*10)/10;
```

*the main choice of whether to design compression steel or not lies on the basis of whether the value of K calculated is greater than or smaller than 0.156*

*In the above section, K is less than 0.156; hence, no compression steel is required and the area of tension steel is calculated using the following equation:*

$$A_{st} = \frac{M}{0.87 f_y z}$$

*the area of steel required is then rounded off to 1 decimal point.*

```
} else if (k > 0.156) {  
    // compression reinforcement required if K > 0.156  
    dcomp = parseFloat (50);  
    asc = ((m*1000000)-(0.156*fcu*b*d*d))/(0.87*fy*(d-dcomp));  
    ast = (0.156*fcu*b*d*d)/(0.87*fy*z) + asc;  
    ast = Math.round(ast*10)/10;  
    ascc = Math.round(asc*10)/10 + "sq.mm";
```

*the other possibility is that the value of K exceeds 0.156*

*Hence, (as can be seen above) compression reinforcement will be required and the areas of tension and compression steel are calculated using the following equations:*

$$A_{sc}' = \frac{M - 0.156 f_{cu} b d^2}{0.87 f_y (d - d')}$$

*and*

$$A_{st} = \frac{0.156 f_{cu} b d^2}{0.87 f_y z} + A_{sc}'$$

*The steel area design are complete.*

```
    dispa = "A";  
    dispsc = "sc";  
}  
statement01 = "For a " + beamtype + " Beam with an Effective Length  
of " + s + "m, An adequate Section size would be: " +  
h + " x " + b + "mm";  
statement02 = "Provide at least " + ast + "sq.mm for the Main  
Reinforcement Steel and " + ascc + " Compression  
Reinforcement";
```

*the first and second statements above summarize all the design parameters calculated in the form of an easy to read paragraph. The minimum areas of tension and compression steel are also displayed.*

*All that's left is to actually move on to the part of the program that displays the results; and that is exactly what the next line does.*

```
    gotoAndStop(10);  
}
```

#### **4.6 Program Limitations**

- The Beam Analysis Modules are limited to the most common loading cases.
- Shear values and deflection profiles are not available for the 2 & 3-span beam analyses.

The profile plotting module is not combined with the main software.

- Maximum span moments are not calculated for the 2 & 3-span beam analyses, but rather, only the mid-span moments.
- Shear Reinforcement is not designed in the RC Design Module.

#### **4.7 General Program Discussion**

Throughout the creation of this software application, ease-of use has been a major consideration. A lot of effort has been put to making the program simple to understand and straight-forward to use. The whole process of the computerized analysis and design follows the same style and procedure as one would generally follow in a manual solution.

Flexibility was also a major consideration in the execution of this application. Since the scope of all computer programs is always limited to a certain extent, special alternatives have been provided for the users to input their own data at different stages of the program, so as to be able to proceed further with the computerized solution.

#### **Choice of Programming Language**

Macromedia Flash was extensively used throughout the programming of this project. The software was initially created for web animations but through the years, has developed a full language, Actionscript, which provides programmers with the tools to create almost any application that they can imagine. The Flash program allows the interactive fusion between visually appealing graphics created and the back-end Actionscript language.

Flash was chosen as the programming language for this project mainly because my experience with this software has brought forward its main advantages of presenting a visually appealing and interactive graphical user interface as well as a powerful language to create code that properly executes the desired tasks when properly programmed.

To demonstrate just how extensive Flash is, all the diagrams and sketches in this report were created using its illustration tools.

### Choice of Structural Theory

The Moment Distribution Method and Macaulay's Method were both used as the main theoretical techniques of analysis in this project.

Macaulay's method was used mainly in deriving equations for the single span analyses. The method was chosen for its broad capacity in solving complex beam loading and support problems. In a few situations, analysis concepts were borrowed from the Moment Area Method to arrive at the equations of some single span systems.

The Moment Distribution Method was comprehensively used in the two and three span beam analyses. This method was chosen because of its simple process of arithmetic iterations in arriving at the final solution. Most structural analysis programs use the stiffness method. During the initial stages of this project, both methods were considered for use. The application of the stiffness method in computer solutions required more research in the proper formation of its large matrices, whereas the moment distribution method, though limited in its application to structural analysis, proved to be applicable to the task of beam analysis, as well as simple in understanding. The moment distribution method was thus decided upon since it seemed more promising in offering a shorter time for its actual programming vis-à-vis the stiffness approach.

## **CHAPTER 5: CONCLUSION & RECOMMENDATIONS**

### **5.1 CONCLUSION**

During the last few decades, computer software has become more and more critical in the analysis of engineering and scientific problems. Much of the reason for this change from manual methods has been the advancement of computer techniques developed by the research community and, in particular, universities.

As both the Technology and Engineering industries advance, new methodologies of interlinking and complementing the industries via computer applications will be created, with a similar improvement in hardware capacities. This in turn will facilitate the implementation of more efficient and professional engineering software. As these software applications advance in functionality, one can hope that they will be more affordable so as to promote their widespread usage amongst civil engineers at a global scale.

The following are the drawn up conclusions that have emanated from the research and implementation of this project:

- A user-friendly program for the computer analysis and reinforced concrete design of beams has been successfully created and tested for the following:
  - Single Span Beam Analysis with the following variable input parameters:
    - Type of end supports
    - Span length
    - Type and intensity of loading
    - EI (modulus of elasticity and moment of inertia)

The program instantaneously calculates and displays the following results using the above parameters:

- Bending Moment diagrams displaying the maximum and salient moments and their respective locations.
  - Shear force diagrams showing the salient shear force values along the beam's span.
  - A variable positioning module that allows the user to easily determine the bending moment, shear force and deflection values at any point on that beam span.
- Two and Three span Beam Analysis with the following variable input parameters:
- Number of spans (either two or three)
  - Individual span lengths
  - Loading types and intensities for each span.
  - Stiffness values for each span
  - End support types (fixed or pinned)

At the click of a button, the program calculates and displays the following output values instantaneously:

- Graphical display of all the input data
  - Support and Midspan Moment values
  - Bending moment diagram displaying the moment profiles for each span
  - Tabulation of the moment distribution process
- Reinforced Concrete Design of beams with the following variable input parameters:
- Design Moment
  - Beam Effective Length
  - Beam Type
  - Steel Strengths
  - Concrete Grade

The parameters may also be calculated and picked from any of the analysis modules.

The following output results are calculated and displayed:

- Section Dimensions
- Steel Area Design

The overall ease with which a user applies this program to everyday beam analysis and design tasks by entering parameters and instantaneously receiving the results in an understandable manner enables a great time saving, accuracy and hence, an optimized design.

The final results of this project were in line with the expectations and objectives.

## 5.2 **RECOMMENDATIONS**

The recommendations directly affiliated with this program are given as follows:

- To combine the Graphical Profile Plotting Module with the main program. When combined, this module will allow exact bending moment, shear force and deflection profiles for each beam to be plotted.  
All code developed for this plotting module has been printed in Appendix B.
- To continue developing, expanding and improving this software application hoping that one day, it will be a full structural analysis program catering for the analysis and design of frames, trusses and other structural elements.

Other general recommendations regarding the developments and advances in computer applications and civil engineering:

- The Department of Civil Engineering at the University of Nairobi should introduce a computer lab for use by students so as to promote the use of computers in the engineering profession.
- The department should encourage conducting similar final year projects dealing with computer applications in the future.
- More emphasis regarding computer technology and applications to engineering should be made at an academic level in different courses. This would broaden the intellect of students as well as expose them to new technologies in all engineering disciplines.

## **CHAPTER 6: SELECTED BIBLIOGRAPHY**

1. Coates R. C., Coutie M. G. & Kong F. K.; “*Structural Analysis*”, 3<sup>rd</sup> Edition, ELBS, 1987
2. “*Extracts from British Standards for Students of Structural Design*”, BSi, 1988
3. Ghali A. & Neville A. M.; “*Structural Analysis*”, 4<sup>th</sup> Edition, E & FN Spon, 1997
4. “*Manual for the Design of Reinforced Concrete Building Structures*”, Institute of Structural Engineers, 1985
5. Moock C.; “*Actionscript: The Definitive Guide*”, O’Reilly & Associates, 2001
6. Mosley W. H. & Bungey J. H.; “*Reinforced Concrete Design*”, 4<sup>th</sup> Edition, Macmillan Press, 1990
7. Norris C. H., Wilbur J. B. & Utslu S.; “*Elementary Structural Analysis*”, 3<sup>rd</sup> Edition, McGraw Hill, 1976
8. Onsongo W. M.; “*Statically Determinate Structures*”, Nairobi University Press, 1993
9. Perry J. H. & Perry R. H.; “*Engineering Manual*”, McGraw Hill, 1959
10. “*Steel Designers’ Manual*”, 4<sup>th</sup> Edition, Oxford, 1990
11. Swannell P.; “*Revision Notes on Theory of Structures*”, Butterworth & Co., 1972

12. Timoshenko S. P. & Young D. H.; "*Theory of Structures*", 2<sup>nd</sup> Edition, McGraw Hill, 1965
13. Todd J. D.; "*Structural Theory & Analysis*", 2<sup>nd</sup> Edition, Macmillan Press, 1981
14. Wang Chu-Kia & Eckel C. L.; "*Elementary Theory of Structures*", McGraw Hill, 1957

## **APPENDIX A**

**Full Program Source Code Excluded for Copyright Purposes.**

However, If the code already displayed in Chapter 4 (Discussion) is not sufficient for you (not likely), please feel free to email me:

[fadzter@yahoo.com](mailto:fadzter@yahoo.com)

Good Luck!

Fady Rostom  
Fadzter Media  
[www.fadzter.com](http://www.fadzter.com)  
[www.fadzter.com/engineering](http://www.fadzter.com/engineering)