A SIMPLIFIED FINITE ELEMENT MODEL FOR TIME-DEPENDENT DEFLECTIONS OF FLAT SLABS

by

RENIER CLOETE

Submitted in partial fulfilment of the requirements for the degree

MASTER OF ENGINEERING (STRUCTURAL ENGINEERING)

In the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND INFORMATION TECHNOLOGY

UNIVERSITY OF PRETORIA

August 2004

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

SAMEVATTING

‘n VEREENVOUDIGDE EINDIGE ELEMENT MODEL VIR DIE BEREKENING VAN TYD-AFHANKLIKE DEFLEKSIES VAN PLAT BLAAIE

Deur: Renier Cloete
Leier: Prof. B.W.J. van Rensburg
Medeleier: Dr. J.M. Robberts
Departement: Siviele Ingenieurswese, Universiteit van Pretoria
Graad: Magister in Ingenieurswese (Struktuuringenieurswese)

Die doel van hierdie verhandeling is om ‘n vereenvoudigde eindige element model te ontwikkel vir die analise van tyd-afhanklike defleksies van plat blaaie.

Nuwe materiale en ontwerpbenaderings het tot gevolg dat diensbaarheidsfalings meer algemeen voorkom. Hierdie tipe van faling is geneig om eers lank na die voltooiing van konstruksie voor te kom en kan dus duur wees om te herstel.

Verskeie gesofistikeerde metodes, gebaseer op gelaagde elementmodelle, nie-lineêre materiaalwette en reologiese benaderings tot krimp and kruip probleme is al deur navorsers voorgestel. Hierdie metodes het die nadele van kompleksiteit en uitermatige rekenaar verwerkingstyd. Gesofistikeerde modelle is in baie gevalle nie geoorloof nie omdat die fisiese prosesse betrokke nie altyd goed verstaan óf akkuraat voorspel kan word nie.

Ontwerpers besef nie altyd die erns van sulke falings nie en ignoreer soms tyd-afhanklike defleksies van blaaie. ‘n Eenvoudige metode, wat maklik inskakel by bestaande ontwerp-metodiek, sal dus ‘n gaping vul in huidige praktyk.

Die veld van studie sluit , in breë trekke, die volgende in:

- Die keuse van eindige element formulering vir die model. ’n 8-node Serendipity element, gebaseer op ‘n Mindlin plaatanalise, vertoon stabiele resultate in ’n groot verskeidenheid van toepassings.
- Kraak van blaaie en die trek-verstywingseffek: Branson se effektiewe traagheidsmoment en die Bilineêre metode is in dié opsig vergelyk.
- Die gekombineerde invloed van krimp en kruip: Die bekende „Effective Modulus Method“ is gebruik om hierdie invloed te modelleer.

Die voorgestelde metode is saamgevat in 'n rekenaarprogram, waarvan die bronkode saamgevat word in die aanhangsels, en word vergelyk met ander metodes en eksperimentele resultate.

Die program vertoon goeie resultate in vergelyking met eksperimentele werk, veral in verband met kraak en trek-verstywing. Branson se benadering tot hierdie probleem lewer beter resultate as die bilineêre metode, veral waar wapening verhoudings laag is. Kruip en krimp resultate vergelyk ook goed met die balk wat ondersoek is, asook die handberekening van 'n bladpaneel se langtermyn defleksies.

Die program maak gebruik van parameters wat algemeen voorkom in betonontwerp en vereis dus nie spesialis kennis van materiaal eienskappe nie. Die metode, soos toegepas in 'n eindige element analise, maak voorsiening vir die plaat gedrag van blaaie en 'n meer realistiese skatting van langtermyn defleksies kan dus gemaak word.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\{\delta\}$	Nodal deflection vector
$\{\delta^e\}$	Element deflection vector
$\{\delta_i\}$	Nodal deflection vector at node i
$\{\epsilon\}$	Strain vector
$\{\phi\}$	Nodal shear rotation vector
$\{\psi\}$	Curvature vector
α	Reduction factor to account for cracking
α_e	Modular ratio
α_{ae}	Age adjusted modular ratio
β_1	CEB-FIP parameter for reinforcement type
β_2	CEB-FIP parameter for duration of loading
δ	Deflection
$\Delta\epsilon_s$	Difference of the mean steel strain and the steel strain at condition 2
$\Delta\psi$	Change in curvature
ϵ_1	Concrete strain at steel level of a singly reinforced concrete section subjected to uniform shrinkage
ϵ_2	Concrete strain at the compression face of a singly reinforced concrete section subjected to uniform shrinkage
$\epsilon_c(t)$	Concrete strain at time *t*
ϵ_{cs}	Free shrinkage strain
ϵ_e	Instantaneous elastic concrete strain
ϵ_{s1}	Tension reinforcement strain at condition 1
ϵ_{s2}	Tension reinforcement strain at condition 2
ϵ_{sm}	Mean strain of the tension reinforcement of a partially cracked concrete member
ζ	Cracking parameter in the Bilinear Method
θ	Plate flexural rotation
κ	Creep modification factor
κ_c	Creep curvature coefficient
κ_s	Interpolated crack curvature coefficient
κ_{s1}	Crack curvature coefficient at condition 1
κ_{s2}	Crack curvature coefficient at condition 2
ν	Poisson's ratio
ξ, η	Axes of the natural coordinate system

Π	Potential energy functional
ρ_c	Percentage compression reinforcement
ρ_t	Percentage tension reinforcement
σ_{s1}	Tension reinforcement stress at condition 1
σ_{s2}	Tension reinforcement stress at condition 2
ϕ	Plate shear deformation
$\chi(t,\tau)$	Age adjustment factor at time t for loading applied at time τ
ψ	Curvature
ψ_1	Curvature at condition 1
ψ_2	Curvature at condition 2
ψ_e	Elastic curvature at time τ
ψ_{sh}	Shrinkage curvature
$\phi(t,\tau)$	Creep coefficient at time t for loading applied at time τ
τ	Age at loading

[B]	Element strain matrix
[D]	Total elasticity matrix
$[D_f]$	Flexural elasticity matrix
$[D_s]$	Shear elasticity matrix
[J]	Jacobian matrix
$[K^e]$	Element stiffness matrix
{M}	Bending moment vector
$\{P^e\}$	Vector of element nodal forces
$\{P^e{}_p\}$	Vector of element nodal forces due to distributed loads
$\{P_{sh}\}$	Vector of nodal forces due to shrinkage
{Q}	Shear force vector
A_c	Effective concrete area
d	Distance from the compression face of a section to the centroid of the tension reinforcement
d_t	Distance from the neutral axis of a section to the level of tension reinforcement.
E_c	Short term concrete elastic modulus
$E_e(t,\tau)$	Effective elastic modulus at time t for loading applied at time τ
E_s	Reinforcement modulus of elasticity
e_s	Eccentricity of the steel centroid with respect to the centroid of the transformed section
f_{c1}	Concrete tensile stress at the tension reinforcement level due to shrinkage

f_{c1}	Concrete tensile stress at the top fibre of the section due to shrinkage
f_r	Modulus of rupture
f_s	Steel stress due to shrinkage
G	Shear modulus
h	Full depth of the concrete section
$\bar{I}$	Moment of inertia of the age adjusted transformed section about an axis through its centroid
I_1	Moment of inertia of the concrete section at condition 1
I_2	Moment of inertia of the concrete section at condition 2
I_c	Moment of inertia based on the cracked, transformed concrete section
I_e	Effective moment of inertia
I_g	Moment of inertia based on the uncracked, untransformed concrete section
I_t	Moment of inertia based on the uncracked, transformed concrete section
Δl	Change in length
l	Length
M	Bending moment
M_r	Cracking moment, based on the modulus of rupture
N_i	Shape function at node i
p	uniformly distributed load
w	Lateral plate deflection
y_c	y-coordinate of the centroid of A_c, measured from the centroid of the age-adjusted, transformed section
Δy	y-coordinate of the centroid of the age adjusted transformed section, measured downwards from the centroid of the transformed section at time τ.

SUMMARY

A SIMPLIFIED FINITE ELEMENT MODEL FOR THE CALCULATION OF TIME-DEPENDENT DEFLECTIONS OF FLAT SLABS

By: Renier Cloete
Supervisor: Prof. B.W.J. van Rensburg
Co-supervisor: Dr. J.M. Robberts
Department: Civil Engineering, University of Pretoria
Degree: Master of Engineering (Structural Engineering)

The aim of this dissertation is to develop a simplified finite element model for the analysis of time-dependent deflections of flat slabs.

New materials and design approaches have caused an increase in the incidence of serviceability failures. This type of failure tends to occur long after the completion of construction and can thus be quite expensive to repair, if at all possible.

Various sophisticated methods based on layered element models, non-linear constitutive laws and rheological creep and shrinkage models have been proposed by various authors. These methods suffer from a high degree of complexity and become prohibitive in terms of computer memory storage requirements and processing time. In many cases sophisticated models are uncalled for due to a lack of understanding of the physical processes involved or an inability to accurately predict the influence of these processes.

Broadly, the field of study includes the following:

- The choice of a finite element formulation for the model. An 8-noded Serendipity element, based on a Mindlin plate analysis has proven itself as a good performer in a wide range of problems and is the formulation of choice for this dissertation.
- Cracking and the tension-stiffening effect: Branson's Effective Moment of Inertia method and the bilinear method are compared in this regard.
- The combined effect of shrinkage and creep: The Effective Modulus method is used to model creep and shrinkage is included using a free shrinkage parameter.

The proposed method was implemented in a computer program, the source code of which is reproduced in the appendices, and compared to other methods and experimental results.

The program achieved good results when compared to experimental work, especially as far as cracking and tension stiffening are concerned. The bilinear method was found to produce results inferior to Branson's approach to tension stiffening, particularly when reinforcement ratios are low. Creep and shrinkage results compared well with the beam considered as well as the hand-calculation of the long-term deflections of a slab panel.

The program utilizes parameters commonly used in routine design, such as the creep coefficient and free shrinkage, and therefore does not require specialist knowledge of material properties. The method, as applied to plates in the finite element analysis, includes the plate behaviour of slabs and thus allows a more realistic estimate of deflection than the equivalent frame method.

1 INTRODUCTION

1.1 Background

The calculation of two-way slab deflections poses the dual difficulties of solving complex governing differential equations and taking into account the effects of material non-linearity, i.e. cracking and creep.

Various simplified methods of elastic analysis for two-way slab systems, suitable for hand calculation, have been proposed and adopted in national building codes. Unfortunately these methods have limited applicability when the slab supports are not rectangular in plan. In contrast, the finite element approach provides a convenient method for analysis where support layouts are irregular. The discretisation of the slab allows the analyst to model almost any shape of slab and the support layout is limited only by node positions. Although the finite element method is a numerical approximation of the 'exact' solution, careful modelling and choice of formulation yields results in good agreement with classical methods.

Load induced cracking, which for the purposes of this dissertation occurs when the stress at the tension face exceeds the modulus of rupture of the concrete, influences both the magnitude of deflection and the distribution of bending moments and shear forces in the slab. At the cracked section, it is assumed that the concrete is free of tensile stress, but this does not hold true for the uncracked zones between fully cracked sections where tensile stresses are transferred to the concrete by bonded reinforcement. The ability of uncracked concrete to contribute to the overall stiffness of a member is referred to as tension stiffening. The simplest method of modelling this effect involves the use of an average cracked section. Numerically, this implies a modification of the second moment of area.

Concrete response to load comprises instantaneous and time-dependent components. The time-dependent component can be attributed to the related effects of creep and shrinkage.

The study of the creep deflection consists of two aspects:

- Prediction of the creep and shrinkage behaviour of a concrete element which includes material and environmental factors. This prediction usually takes the form of a creep factor or function and a shrinkage strain.
- Incorporating the predictive parameters in the analysis of a reinforced concrete member.

This dissertation focuses on the latter aspect and no attempt is made to investigate the various creep and shrinkage models that are currently in use. Numerous methods are available for time-dependent analyses, ranging from complex rheological models to simple effective modulus methods. Faber's Effective Modulus method (Gilbert, 1988), where the elastic modulus of the concrete is adjusted with a creep factor, is probably the best known method.

The method proposed here employs the finite element method in a semi-iterative approach to the deflection problem. The instantaneous deflection is calculated using Branson's Effective Moment of Inertia (Branson, 1968) or the bilinear method (Favre *et al*, 1985). The resulting member actions are then used in a time analysis to establish the creep and shrinkage contribution to the final deflection.

The method derives from a simplified approach to slab tension stiffening suggested by Polak (1996). Polak applied Branson's effective moment of inertia to the finite element method using reduction factors to account for cracking and tension stiffening. Reinforcement has the effect of reducing the magnitude of creep deflections in concrete slabs. This effect is also accounted for using reduction factors.

Rigorous approaches to the long term deflection of concrete slabs, as applied to the finite element method, abound in journals such as *Advances in Engineering Software* published by *Elsevier* and are even commercially available in structural simulation software such as *DIANA,* which is developed by TNO Diana, a company based in the Netherlands. This dissertation proposes a simple method for use and elaboration by practicing engineers.

1.2 Objectives of the Study

The study has the broad objective of establishing a simple method of analysis, based on the finite element method, for the calculation of the long-term deflections of reinforced concrete flat slabs.

This objective is pursued under four components of deflection:

- Elastic deflection:
  The Mindlin Serendipity plate element, as applied to slab problems, is investigated. Convergence characteristics and the influence of aspect ratios are studied to verify the applicability of the element in the current context.

- Cracked deflection:
  The results of previous studies of tension stiffening are reproduced and the bilinear method is investigated as an alternative to Branson's method. The alternative is attractive due to the simple manner in which the influence of creep on the behaviour of a cracked section is modelled.
- Creep deflection:
  The objective of this section is to find a technique of incorporating the method of section curvatures in the finite element formulation.
- Shrinkage deflection:
  The resulting deformation of shrinkage is transformed into applied loading and the results are compared to experimental results to verify the accuracy of the procedure.

1.3 Scope of the Study

1.3.1 Geometry and Supports

Although nothing in the method prohibits non-rectangular layouts, only this type of layout was studied, mainly to avoid the use of distorted rectangular or triangular elements since distortion impacts negatively on the accuracy of the finite element analysis.

1.3.2 Finite Element Formulation

An eight-noded, Serendipity plate element based on the Mindlin formulation was chosen for the dissertation and no other element or plate formulation types were considered, although consideration is given to the numerical issues pertaining to this element. Membrane effects are not considered.

An important aspect of the finite element formulation is the accurate modelling of supports or boundary conditions. However, discontinuous boundary conditions such as those encountered with the column supports of flat slabs pose some problems. The simplest approach is a support, fixed in both translation and rotation, at a node placed at the column centre. Theoretically this implies infinite stress resultants at the point of support, although with the finite element approximation this value is finite but large.

This dissertation follows this simple approach and no consideration is given to alternative approaches such as elastic supports. Admittedly the simple approach will tend to overestimate the actual long term deflection, due to the large numerical values of the support moments and the associated higher degree of cracking.

1.3.3 Reinforcement Layout

Reinforcement directions are assumed to be parallel with the global x and y-axis. With the rectangular layout constraint introduced in paragraph 1.3.1, this will almost always be the case.

1.3.4 Constitutive model

Concrete and reinforcement behaviour are assumed to be linear elastic, although modifications are made for concrete creep and cracking. Reference is made to rigorous models, but no comparison is done with the results of these methods.

1.4 Methodology

A finite element program was developed specifically for this dissertation. The program follows the Object Orientated Programming (OOP) paradigm, which allows for easy extension and modular problem solution. The finite element is encapsulated in an object and the various effects of cracking, creep and shrinkage were added to this element object.

The results of the program are compared with the following analytical, experimental and hand-calculation results:

- Linear elastic analysis: Classical thin plate and moderately thick plate solutions for simply supported and clamped plates.
- Short term crack analysis: Experimental slab deflection results from two sources.
- Long term creep, crack and shrinkage analysis: Experimental results for a beam and hand calculation results for a flat slab panel.

1.5 Software Development

The source provided by Hinton and Owen (1983) was used as the basis for the implementation. The source was originally written in FORTRAN which was translated by the author to Object Pascal. The source was compiled for the Microsoft Windows environment using Borland Delphi 6.0, Borland (2001).

The user interface components used in the program were donated by Prokon Software Consultants which facilitated the process of input as well as the output of analysis results.

The OOP paradigm mentioned in section 1.4, adds a level of abstraction to software which allows for modular programming and extensibility. The main program remains unaware of the implementation details of the element object and the element formulation can thus be changed without influencing the logic of the main program.

1.6 Organisation of the Report

The report is split into the following chapters:

Chapter 2 provides theoretical background to the topics introduced above under the following headings:

- Analysis: The finite element and a hand-calculation method are considered.
- Cracked Sections: The bilinear method and the effective moment of inertia are compared.
- Creep: A single approach, based on the effective modulus method is presented.
- Shrinkage: A single approach based on basic theory is presented.

Chapter 3 presents the proposed method under the same headings as above and discusses the implementation of the method in computer code. Selected parts of the program are included in the Appendix for reference. Section 7.2 contains the source of the entire, self contained, finite element object. Sections 7.3 to 7.6 contain the various analysis procedures.

Chapter 4 documents the results of various analysis runs compared to alternative methods and published experimental data.

Conclusions regarding the accuracy and applicability of the proposed method are presented in Chapter 5. Chapter 6 lists the references and chapter 7 contains the appendix.

2 THEORETICAL BACKGROUND

2.1 Analysis

Analysis methods can be categorized into classical and numerical approaches. The classical approach involves finding stress or displacement solutions that satisfy the differential equations of equilibrium, compatibility requirements as well as stress-strain relationships, subject to the given boundary conditions. Due to these stringent requirements, very few classical solutions are available for practical plate bending problems.

This difficulty can be overcome by approximating plate behaviour with a crossing beam analogy such as the equivalent frame method, Corley and Jirsa (1970). The results of such an equivalent frame analysis, as well as the required reinforcement, are assumed known in the hand-calculation method presented in 3.1.1.

Numerical approaches require discretisation of the problem, i.e. a structure with an infinite number of degrees of freedom is reduced to a finite number to simplify the calculation process. Two of the best known methods are the method of finite differences and the finite element method. The method of finite differences has the disadvantage of difficulty in satisfying irregular boundary conditions. The finite element method, which is the method of choice in this dissertation, is presented in section 2.1.2.

2.1.1 Hand-Calculation Method

Short and long term deflections of two way slab systems can be calculated by the simplified method outlined below. This method is recommended by various authors, such as Gilbert (1988), Ghali and Favre (1986), as well as the ACI 318 (1999).

For flat slabs, the method involves calculating the mid-span deflections of the middle strip relative to the column strip deflections, x and y-direction strips being treated independently. The middle strip deflections are then added to the average of the column strip deflections and finally, the x and y-direction mid-span deflections are averaged to arrive at a total mid-panel deflection.

The deflection calculations outlined above, make use of known curvatures which are modified with factors to account for cracking, shrinkage and creep. These factors are discussed in detail in sections 3.2 through 3.4.

The main drawbacks of the method are as follows:

- Only rectangular slabs are considered,
- Bending moment magnitudes are assumed to be known, although these moments can easily be calculated using the Direct Design (ACI 318, 1999) or Equivalent Frame Methods,
- Curvatures are assumed to be parabolically distributed over the length of the strip considered,
- Simple support or continuity is assumed.

Deflection at the centre of a strip is given by (Ghali & Favre, 1986):

$$\delta = \frac{l^2}{96}(\psi_1 + 10\psi_2 + \psi_3) \qquad (2.1)$$

where:

l	= length of the strip
ψ_1, ψ_2, ψ_3	= curvatures at the left support, centre and right support of the strip.

Deflection at the centre of a panel can be expressed as the sum of the middle strip deflection and the average of the column strip deflections.

$$D_1 = \delta_{EF} + \frac{1}{2}(\delta_{AB} + \delta_{DC}) \qquad (2.2)$$

$$D_2 = \delta_{HI} + \frac{1}{2}(\delta_{AD} + \delta_{BC}) \qquad (2.3)$$

$$D_{final} = {}^{1}\!/_{2}(D_1 + D_2) \qquad (2.4)$$

The various deflection components in the equations (2.2) through (2.4) are illustrated in figure 2-1.

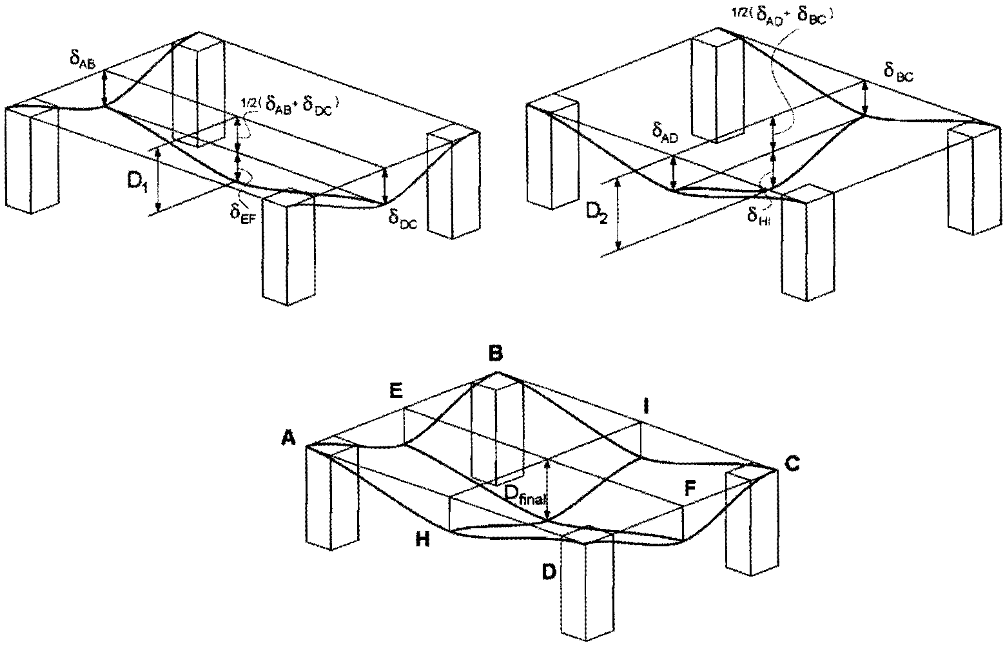

Figure 2-1: Displacement components

Curvature and bending moment can be related with the following equations (Ghali & Favre, 1986):

$$\psi_x = \frac{1}{E_c I_g}\left(M_x + \nu M_y\right) \tag{2.5}$$

$$\psi_y = \frac{1}{E_c I_g}\left(M_y + \nu M_x\right) \tag{2.6}$$

$$I_g = \frac{h^3}{12\left(1-\nu^2\right)} \tag{2.7}$$

where:

E_c	= Young's modulus of the concrete,
I_g	= Effective moment of inertia of the gross concrete area of the strip,
M_x, M_y	= Moments at the section under consideration,
ν	= Poisson's ratio of the concrete (usually taken as 0.2),
h	= Slab thickness.

2.1.2 Finite Element Method

This method involves the discretisation of continua problems into finite sub regions termed finite elements. The approach yields approximate results based on an assumed stress field, displacement field or a mixed, hybrid approach. These fields are defined by points, or nodes, in the finite element. The approach presented here is limited to the displacement approach, due to its widespread use for matrix analysis software.

The element used in the study is an eight-noded Serendipity element. These elements differ from elements such as Lagrange elements in the derivation of their shape functions. Langrage elements can contain nodes interior to the element and the Lagrange interpolation function is used to find these shape functions. Serendipity elements on the other hand, usually consist of only edge nodes and the shape functions are found by inspection.

Shape functions are approximations of element geometry and deflection behaviour. The same shape functions (second degree polynomials) are used for the definition of geometry and displacement, classifying this element as parabolic isoparametric.

The assumptions for the flexural formulation, due to Mindlin (1951), are as follows:

- The lateral deflection of the plate is small compared to its plan dimensions.
- Planes normal to the plate mid-surface remain plane, but not necessarily normal to the mid-surface, after bending.
- Stresses normal to the plate mid-surface are negligible.

The Kirchhoff, or thin-plate, assumptions (Ugural, 1999) for plate bending differ only in the second point above: Planes remain plane *and* normal to the mid-surface after bending. This implies that the Kirchhoff model neglects shear effects.

The Mindlin elements were developed mainly to overcome inter-element continuity problems that arose from the use of their Kirchhoff counterparts. Mindlin elements do pose some numerical problems such as "shear locking" which is elaborated upon at the end of this section.

The generalised displacements of the plate are completely described using three degrees of freedom per node where w denotes a displacement in the z direction and θ_x and θ_y orthogonal rotations about the y and x axis, respectively. This leads to following expressions for the plate deformations

$$\{\delta\} = \begin{bmatrix} w \\ \theta_x \\ \theta_y \end{bmatrix} = \begin{bmatrix} w \\ \dfrac{\partial w}{\partial x} + \phi_x \\ \dfrac{\partial w}{\partial y} + \phi_y \end{bmatrix} \tag{2.8}$$

These physical quantities and the orientation of the global Cartesian coordinate system are illustrated in figure 2-2.

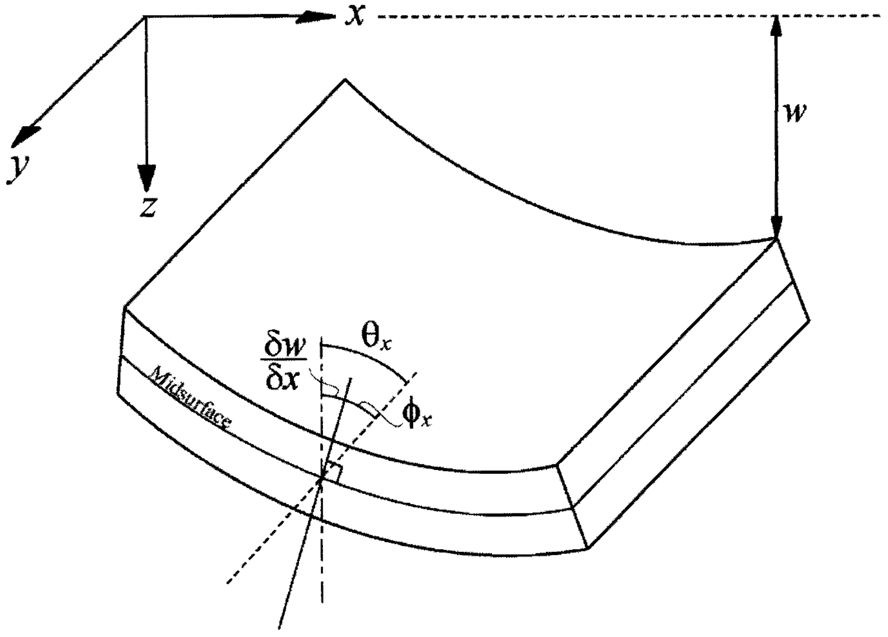

Figure 2-2: Plate deformations

The formulation makes use of a natural (dimensionless), curvilinear coordinate system in ξ and η with origin at the geometric centre of the element. Although the directions of the ξ and η axes vary within the element, the general positive directions are:

- Positive ξ is taken in the same direction as indicated by the nodal sequence 1-2-3.
- Positive η is taken in the same direction as indicated by the nodal sequence 3-4-5.

The nodal numbering starts at any corner of the element and proceeds in an anti-clockwise direction as shown in figure 2-3.

i 17512396
b16427460

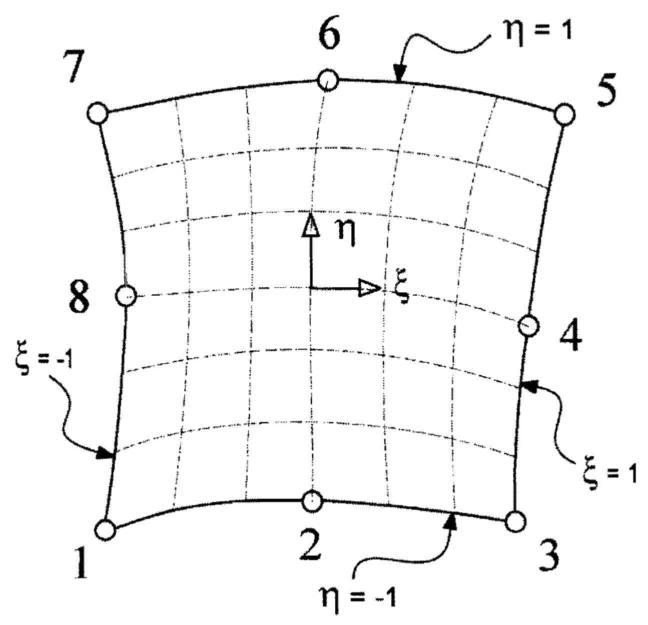

Figure 2-3: Element numbering

The geometry of each element is defined by

$$\begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{bmatrix} = \sum_{i=1}^{8} \begin{bmatrix} N_i & 0 \\ 0 & N_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{2.9}$$

where N_i is the parabolic shape function, or interpolation function, associated with node i. These functions have a value of unity at the associated nodes and zero at every other node.

Similarly the displacement field over the element is defined by

$$\boldsymbol{\delta} = \begin{bmatrix} w \\ \theta_x \\ \theta_y \end{bmatrix} = \sum_{i=1}^{8} \begin{bmatrix} N_i & 0 & 0 \\ 0 & N_i & 0 \\ 0 & 0 & N_i \end{bmatrix} \begin{bmatrix} w_i \\ \theta_{xi} \\ \theta_{yi} \end{bmatrix} \tag{2.10}$$

The shape functions are given by

$$\begin{aligned}
N_1(\xi,\eta) &= -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta)\\
N_2(\xi,\eta) &= \frac{1}{2}(1-\xi^2)(1-\eta)\\
N_3(\xi,\eta) &= \frac{1}{4}(1+\xi)(1-\eta)(\xi-\eta-1)\\
N_4(\xi,\eta) &= \frac{1}{2}(1+\xi)(1-\eta^2)\\
N_5(\xi,\eta) &= \frac{1}{4}(1+\xi)(1+\eta)(\xi+\eta-1)\\
N_6(\xi,\eta) &= \frac{1}{2}(1-\xi^2)(1+\eta)\\
N_7(\xi,\eta) &= \frac{1}{4}(1-\xi)(1+\eta)(-\xi+\eta-1)\\
N_8(\xi,\eta) &= \frac{1}{2}(1-\xi)(1-\eta^2)
\end{aligned} \tag{2.11}$$

The variational approach to the formulation of the stiffness matrix is used, specifically the minimisation of potential energy principle. The potential energy functional consists of terms for bending, shear and external work done by the applied lateral pressure denoted by p:

$$\Pi = \frac{1}{2}\int_A \left(\{M\}^T\{\psi\} + \{Q\}^T\{\phi\}\right)dA - \int_A pw\,dA \tag{2.12}$$

where {M} and {Q} are as defined in equations (2.15) and (2.16) respectively.

Positions of equilibrium are denoted by positions of stationary potential energy or $\delta\Pi = 0$, which leads to equation (2.30). The curvatures and shear strains are defined as

$$\{\psi\} = \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_{xy} \end{bmatrix} = \begin{bmatrix} -\dfrac{\partial\theta_x}{\partial x} \\ -\dfrac{\partial\theta_y}{\partial y} \\ -\left(\dfrac{\partial\theta_x}{\partial y} + \dfrac{\partial\theta_y}{\partial x}\right) \end{bmatrix} \tag{2.13}$$

$$\{\phi\} = \begin{bmatrix} -\phi_x \\ -\phi_y \end{bmatrix} \tag{2.14}$$

The stress resultants $\{M\}$ (bending moment) and $\{Q\}$ (shear force) are calculated by pre-integrating the relevant stresses over the depth of the plate which is denoted by h. The resultants are related to element strains by the following expressions, illustrated in the figure below:

$$\{M\} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} = [D_f]\{\psi\} \tag{2.15}$$

$$\{Q\} = \begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = [D_s]\{\phi\} \tag{2.16}$$

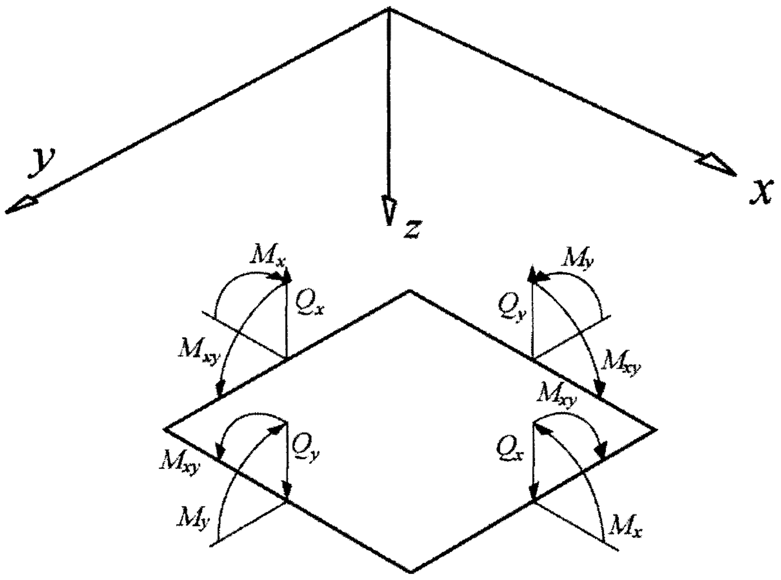

Figure 2-4: Sign convention (positive directions)

where $[D_f]$ and $[D_s]$ are the elasticity matrices,

$$[D_f] = \frac{Eh^3}{12(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{2.17}$$

$$[D_s] = \frac{Eh\kappa}{2(1+\nu)}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = Gh\kappa\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{2.18}$$

for flexure and shear, respectively. The elasticity matrices above are usually combined as the total elasticity matrix:

$$[D] = \begin{bmatrix} \frac{E_c h^3}{12(1-\nu^2)} & \frac{E_c \nu h^3}{12(1-\nu^2)} & 0 & 0 & 0 \\ \frac{E_c \nu h^3}{12(1-\nu^2)} & \frac{E_c h^3}{12(1-\nu^2)} & 0 & 0 & 0 \\ 0 & 0 & \frac{Gh^3}{12} & 0 & 0 \\ 0 & 0 & 0 & Gh\kappa & 0 \\ 0 & 0 & 0 & 0 & Gh\kappa \end{bmatrix} \tag{2.19}$$

where $\kappa = 5/6$ is a shear correction factor, applicable only to rectangular sections. This factor is used to transform the assumed parabolic shear stress distribution over the depth of the plate to an equivalent constant stress distribution.
Strains are related to displacements by

$$\{\varepsilon\} = [L]\{\delta\} \tag{2.20}$$

where $\{\varepsilon\}$ is a generalised strain vector,

$$\{\varepsilon\} = \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_{xy} \\ -\phi_x \\ -\phi_y \end{bmatrix} \tag{2.21}$$

$\{\delta\}$ is the generalised displacement vector and $[L]$ is the matrix of displacement differential operators. With the finite element method, $\{\delta\}$ is approximated by $N\delta^e$ within an element, where

$$[N]=\begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & . & 0 & 0 & N_8 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & . & 0 & 0 & N_8 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & . & 0 & 0 & N_8 \end{bmatrix} \tag{2.22}$$

and

$$\{\delta^e\}^T=[\delta_1 \quad \delta_2 \quad . \quad . \quad \delta_i \quad . \quad . \quad \delta_8] \tag{2.23}$$

is the vector of nodal displacement components and δ_i is given by

$$\{\delta_i\}=\begin{bmatrix} w_i \\ \theta_{xi} \\ \theta_{yi} \end{bmatrix} \tag{2.24}$$

Equation (2.20) then becomes

$$\{\varepsilon\}=[L][N]\{\delta^e\}=[B]\{\delta^e\} \tag{2.25}$$

where $[B]$ is known as the element strain matrix,

$$[B]=[B_1 \quad B_2 \quad . \quad . \quad B_i \quad . \quad . \quad B_8] \tag{2.26}$$

and $[B]_i$ is calculated for each node as

$$[B]_i=\begin{bmatrix} 0 & -\frac{\partial N_i}{\partial x} & 0 \\ 0 & 0 & -\frac{\partial N_i}{\partial y} \\ 0 & -\frac{\partial N_i}{\partial y} & -\frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial x} & -N_i & 0 \\ \frac{\partial N_i}{\partial y} & 0 & -N_i \end{bmatrix} \tag{2.27}$$

Finally the element stiffness matrix can be assembled as

$$[K^e] = \iint [B]^T [D][B]\,dxdy = \iint [B]^T [D][B]|J|d\xi d\eta \tag{2.28}$$

where $|J|$ is the determinant of the Jacobian matrix.

Distributed loads, denoted by p, must be represented as equivalent nodal loads in this method and the following equation is used for this purpose:

$$\{P_p^e\} = \int_{A_e} N_i p dA = \iint N_i p |J| d\xi d\eta \tag{2.29}$$

where A_e is the elemental area.

The linear system of equations per element is then

$$\{P_p^e\} + \{P^e\} = [K^e]\{\delta^e\} \tag{2.30}$$

where P^e denotes the nodal forces.

The element stiffness matrices are then assembled in the global stiffness matrix with the direct stiffness method and the problem reduces to a system of linear algebraic equations. This system can be solved with any suitable numerical method such as Gauss reduction or Cholesky decomposition.

A discussion on "shear locking", introduced at the beginning of the section, follows. Thin plates, modelled using Mindlin elements, often exhibit a high, incorrect shear stiffness which is termed shear locking. Mathematically this can be studied using the energy contributions of shear and flexure to equation (2.12). The flexural strain energy varies cubically with thickness whereas shear strain energy varies linearly, this implies that flexural strain energy decreases more rapidly with reductions in depth than shear strain energy.

When one takes into account that the shear strain energy is given by:

$$\Pi = \frac{1}{2}\int_A \left(\{\phi\}^T [D_s]\{\phi\}\right) dA \quad \text{and} \tag{2.31}$$

$$[D_s] = Gh\kappa \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.32)$$

it appears that the shear strain energy should disappear as h→ 0. Due to the approximate nature of the finite element method, zero strain energy is rarely achieved and under various conditions, the stiffness matrix becomes ill-conditioned, Tesler & Hughes (1983), which leads to a gross overestimation of stiffness.

This locking problem can be avoided with the use of reduced or selective numerical integration (Zienkiewicz, Taylor and Too, 1971) . The basis of these integration schemes resides in the assumption that shear locking can be avoided by not integrating the shear strain energy exactly. This dissertation employs a reduced 2x2 Gaussian integration scheme.

Shear locking tendencies can be assessed using the constraint ratio, Hughes (1987), or the Kirchhoff mode concept, Hughes & Tezduyar (1981).

The entire analysis process for a linear-elastic analysis is illustrated in figure 7-1.

2.2 Cracked Sections

Concrete members crack when the tensile stress at a section exceeds the tensile strength, usually taken as the modulus of rupture for members subjected to flexure. The flexural stiffness along the member then varies between two extremes:

- Condition 1: Where the tensile stress is below the modulus of rupture, the concrete remains uncracked and the full section contributes to the stiffness.
- Condition 2: At sections where the tensile stress exceeds the modulus of rupture, the concrete cracks over the full depth of the tension zone. Cracks at these sections are often referred to as primary cracks. The flexural stiffness at such a cracked section can be estimated from the fully cracked, transformed section.

Assuming a stiffness based on condition 2 would overestimate the deflection of the member, since the regions between primary cracks remain uncracked or partially cracked. In these regions, the concrete in tension contributes to the flexural stiffness and this is referred to as tension stiffening.

Including tension stiffening in the deflection analysis of a concrete member involves interpolating between conditions 1 and 2. Two empirical methods are considered here, the Bilinear Method and Branson's Effective Moment of Inertia.

2.2.1 The Bilinear Method

This method, first proposed for beam cracking problems by Favre *et al* (1985), is developed below:

Assuming that plain sections remain plane in bending for uncracked and cracked sections, strains remain linearly distributed over the depth of a section. Although this is not strictly true at the cracked section, the *average* strain measured over a number of primary cracks retains proportionality to the distance from the neutral axis.

Subject to the assumption that no bond slip occurs, the strain in the tension reinforcement at uncracked sections (condition 1), just prior to cracking, can be expressed as:

$$\varepsilon_{s1} = \frac{M_r d_t}{I_I E_c} \qquad (2.33)$$

where M_r is the cracking moment at the section under consideration, d_t is the depth from the neutral axis of the section to the level of the tension reinforcement, I_I is the moment of inertia based on the uncracked transformed section and E_c is the secant modulus of elasticity of the concrete.

The cracking moment for a rectangular section can be expressed as

$$M_r = \frac{f_r I_g}{h/2} \qquad (2.34)$$

where f_r is the modulus of rupture, I_g is the moment of inertia of the gross concrete section neglecting reinforcement and h is the section depth.

At fully cracked sections (condition 2), equation (2.33) changes to

$$\varepsilon_{s2} = \frac{Md_t}{I_2 E_c} \tag{2.35}$$

where I_2 is the moment of inertia based on the fully cracked transformed section and $M \geq M_r$. The steel stresses for condition 1 and 2 would then be

$$\sigma_{s1} = E_s \varepsilon_{s1} \tag{2.36}$$

$$\sigma_{s2} = E_s \varepsilon_{s2} \tag{2.37}$$

assuming linear elastic reinforcement behaviour as illustrated in figure 2-5.

Between primary cracks, the reinforcement strain increases to a value larger than ε_{s1} but smaller than ε_{s2}. The steel strain will gradually decrease from its maximum value at the crack to ε_{s1} as bond transfers tension from the reinforcement to the concrete. When the strain reaches ε_{s1}, another primary crack forms and the process repeats itself.

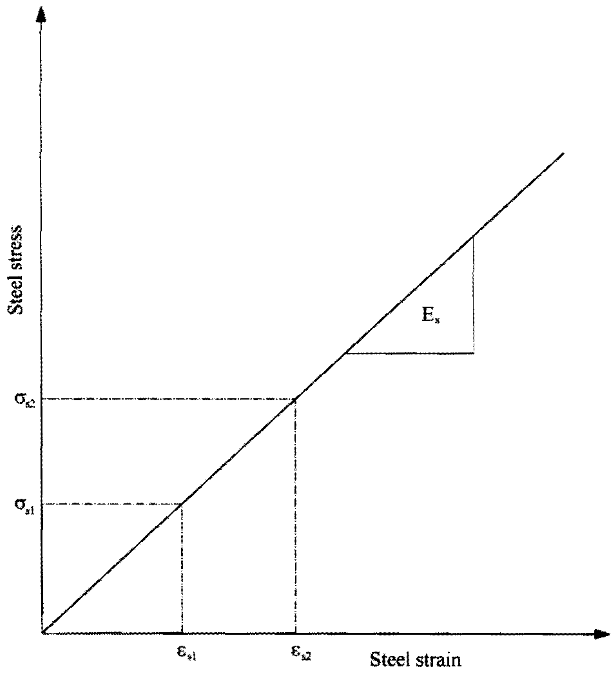

Figure 2-5: Reinforcement stress-strain relation

Let ε_{sm} represent the mean strain of the reinforcement in the member

$$\varepsilon_{sm} = \frac{\Delta l}{l} \tag{2.38}$$

where *Δl* is the change in member length at the level of the reinforcement and *l* the original length. As mentioned in the preceding paragraph ε_{sm} will be smaller than ε_{s2} with the difference $\Delta\varepsilon_s$, thus

$$\varepsilon_{sm} = \varepsilon_{s2} - \Delta\varepsilon_s \tag{2.39}$$

This difference has a maximum value of $\Delta\varepsilon_{smax}$ at the onset of cracking. Experimental evidence has shown that $\Delta\varepsilon_s$ can be related to σ_{s2} as follows, Ghali & Favre (1986)

$$\Delta\varepsilon_s = \Delta\varepsilon_{s\,\max}\frac{f_r}{\sigma_{s2}} = \Delta\varepsilon_{s\,\max}\frac{M_r}{M} \tag{2.40}$$

Note that equation (2.40) is based on the assumption that the uncracked concrete has the same effect on the mean reinforcement strain in flexure as is the case with axial loading.

Equations (2.39) and (2.40) are shown graphically in figure 2-6 below.

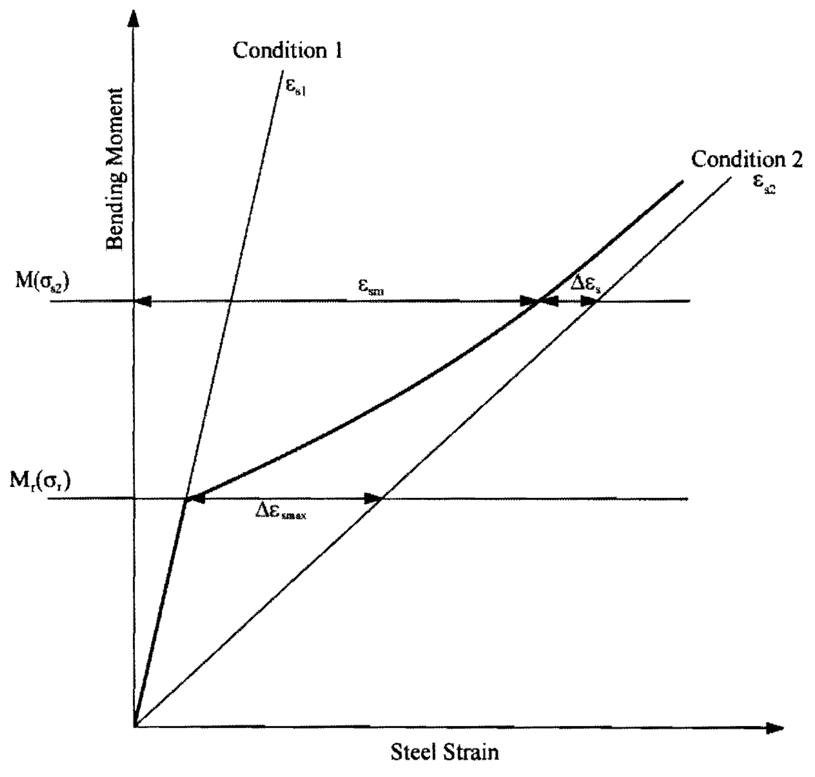

Figure 2-6: Variation of steel strains versus bending moment

From the geometry of the graph, $\Delta\varepsilon_{smax}$ can be written as

$$\Delta\varepsilon_{s\,\max} = (\varepsilon_{s2} - \varepsilon_{s1})\frac{M_r}{M} \tag{2.41}$$

Substituting equation (2.41) and (2.40) into equation (2.39) yields

$$\varepsilon_{sm} = (1-\zeta)\varepsilon_{s1} + \zeta\varepsilon_{s2} \quad \text{or} \tag{2.42}$$

$$\psi_{sm} = (1-\zeta)\psi_{s1} + \zeta\psi_{s2} \tag{2.43}$$

where ζ is a dimensionless parameter that measures the extent of cracking, zero for an uncracked section and between zero and unity for a fully cracked section. The parameters ψ_{s1} and ψ_{s2} represent the curvatures at the uncracked and cracked states, respectively.

$$\zeta = 1 - \left(\frac{M_r}{M}\right)^2 \quad \text{with } M > M_r \tag{2.44}$$

The CEB-FIP Model Code 1990 (1993) introduces two additional parameters to account for the difference in bond characteristics of deformed and plain bars, as well as long term effects

$$\zeta = 1 - \beta_1\beta_2\left(\frac{M_r}{M}\right)^2 \tag{2.45}$$

β_1 equals 1 and 0.5 for deformed and plain bars, respectively. β_2 equals 1 and 0.5 for immediate loading and sustained loading, respectively.

This approach to tension stiffening can be used in combination with the hand-calculation method described in section 2.1.1. A curvature coefficient is calculated for condition 1 and 2:

$$\kappa_{s1} = \frac{I_g}{I_1} \tag{2.46}$$

$$\kappa_{s2} = \frac{I_g}{I_2} \tag{2.47}$$

These coefficients represent the influence of the reinforcement on uncracked and cracked curvatures.

An effective coefficient is then found by interpolating with equation (2.44):

$$\kappa_s = (1-\zeta)\kappa_{s1} + \zeta\kappa_{s2} \qquad (2.48)$$

where:

I_g	= Moment of inertia of the gross concrete area, neglecting reinforcement;
I_1	= Moment of inertia of the uncracked, transformed section;
I_2	= Moment of inertia of the fully cracked transformed section;
κ_{s1}	= Curvature coefficient for condition 1;
κ_{s2}	= Curvature coefficient for condition 2.

2.2.2 Branson's Effective Moment of Inertia

Similar to the bilinear method, this method proposes an effective moment of inertia, constant over the length of a member, for the computation of deflections. The effective moment of inertia, developed by Branson (1968), is expressed as:

$$I_e = \left(\frac{M_r}{M}\right)^m I_g + \left[1 - \left(\frac{M_r}{M}\right)^m\right] I_2 \qquad (2.49)$$

where I_g is the moment of inertia of the gross concrete section neglecting reinforcement and m is a power usually set to 3, although Branson suggested a value of 4 for calculating I_e at a specific section.

Although this equation was developed for beams, a study undertaken by Polak (1996), suggests that sufficiently accurate deflection results are achieved for slabs using the equation in conjunction with the finite element method. This method is set out below.

The difficulty in applying Branson's equation to plate bending problems concerns the definition of flexural rigidity. In the case of beams the flexural rigidity is simply the product EI, whereas in the plate formulation, flexural rigidity is represented by the matrix $[D_f]$ as shown in equation (2.17). Polak circumvented this difficulty by modifying E and ν instead of I.

The ratio of the cracked to gross second moments of inertia is used to modify $[D]$ per element. This implies that the cracked section properties are averaged across all nodes belonging to an element to arrive at a single partially cracked element. This is achieved by calculating average moments in the x and y-directions for use in equation (2.49),

$$M_{xavg} = \frac{1}{n}\sum_{i=1}^{n}\left(|M_{xi}| + |M_{xyi}|\right) \tag{2.50}$$

$$M_{yavg} = \frac{1}{n}\sum_{i=1}^{n}\left(|M_{yi}| + |M_{xyi}|\right) \tag{2.51}$$

where *n* is the number of Gaussian sampling points and M_{xi}, M_{yi} are the moments calculated at sampling point *i*.

The elasticity matrix, modified for tension stiffening, takes the following orthotropic form:

$$[D] = \begin{bmatrix} \frac{E_x h^3}{12(1-\nu_x\nu_y)} & \frac{E_y \nu_x h^3}{12(1-\nu_x\nu_y)} & 0 & 0 & 0 \\ \frac{E_x \nu_y h^3}{12(1-\nu_x\nu_y)} & \frac{E_y h^3}{12(1-\nu_x\nu_y)} & 0 & 0 & 0 \\ 0 & 0 & \frac{G_1 h^3}{12} & 0 & 0 \\ 0 & 0 & 0 & G_2 h & 0 \\ 0 & 0 & 0 & 0 & G_3 h \end{bmatrix} \tag{2.52}$$

where:

$$E_x = \alpha_x E_c, \qquad E_y = \alpha_y E_c \tag{2.53}$$

$$\nu_x = \alpha_x \nu, \qquad \nu_y = \alpha_y \nu \tag{2.54}$$

$$G_1 = G\alpha_x\alpha_y, \quad G_2 = G\alpha_x, \quad G_3 = G\alpha_y \tag{2.55}$$

$$\alpha_x = \frac{I_{ex}}{I_g}, \qquad \alpha_y = \frac{I_{ey}}{I_g} \tag{2.56}$$

I_{ex} and I_{ey} are calculated using the average moments obtained from equations (2.50) and (2.51).

Bensalem (1997) pointed out weaknesses in Polak's proposed method, many of which are intentional approximations with a simple method in mind as pointed out in Polak's closure.

One of these weaknesses involves the calculation of the average moments. Bensalem argues that the approach would only be valid for conditions when the signs of the moments are the same. Should these signs differ, over- or underestimation of the average moments would occur. Typical rectangular layouts lead to same sign moments and the approach remains valid.

The analysis algorithm, as given by Polak, is restated in simplified form in figure 7-2 for reference.

2.2.3 Rigorous methods

A number of sophisticated approaches to post-cracking behaviour and tension stiffening have been proposed. These models usually incorporate non-linear constitutive relations and multi-layer elements as the main components.

Chan et al (1994), utilises strain hardening plasticity theory to develop the constitutive model for a finite element analysis and also present a bond stress distribution function to model tension stiffening. Principal stresses at integration points are evaluated and compared to the cracking strength of the concrete. Should the cracking strength be exceeded, tangential concrete moduli are calculated using the bond stress distribution. The constitutive matrix is modified with the tangential moduli and the analysis proceeds in an iterative manner with continuous model updating.

Hu et al (1991), use a similar approach as above, except that an explicit tension stiffening function and a layered finite element formulation is used. Crack directions are modified during the analysis to ensure that cracks remain normal to the maximum principal stresses.

Due to the non-linear nature of the methods briefly outlined above, and in fact almost all sophisticated methods, iteration, and all the associated numerical difficulties, is required. Large finite element models using these approaches become bulky in terms of storage requirements and computing time.

2.3 Creep

Creep is a progressive increase in strain under sustained loading and is responsible for the largest portion of long term deflections in concrete structures. The current state of the art with regards to creep is perhaps best described in a paper by Bažant (2001) : "…despite major successes, the phenomenon of creep and shrinkage is still far from being completely understood, even though it has occupied some of the best minds in the field on cement and concrete research and material science…"

A large number of predictive physical models have been proposed in the past and are still being developed. Neville & Dilger (1970) and Bažant (2001) have provided detailed overviews of these models and they will not be repeated here.

These physical models lead to mathematical models that facilitate structural analysis:

- Effective modulus method.
- Age-adjusted effective modulus method.
- Rate of creep method.
- Improved Dischinger method.
- Rheological models.

Analysis methods fall into two classes:

- Single step approximations.
- Step-by-step iterative solutions.

Of these, the simplest choice would be the age-adjusted effective modulus method using a single time-step approximation. These single step approximations yield acceptable upper bound deflections for routine design and are not plagued with the numerical problems of iterative rheological approaches, such as creep divergence, Bažant (1993).

The discussion below is therefore limited to the effective modulus method and the related age-adjusted effective modulus method.

In the study of creep effects, it is convenient to separate creep strain into the following components (figure 2-7):

- Irrecoverable creep or flow, designated by ε_f.
- ε_f is further divided into basic creep and drying creep.
- Recoverable creep or delayed elastic strain, designated by ε_d.

One of the main disadvantages of the effective modulus method is its inability to deal with decreasing stress histories. As can be seen from figure 2-7, creep involves a measure of irrecoverable strain. The effective modulus method, due to its elastic nature, predicts complete recovery of creep strains, i.e. a return to zero strain at unloading. This will lead to a severe underestimation of deflection for structures subjected to cyclic loading.

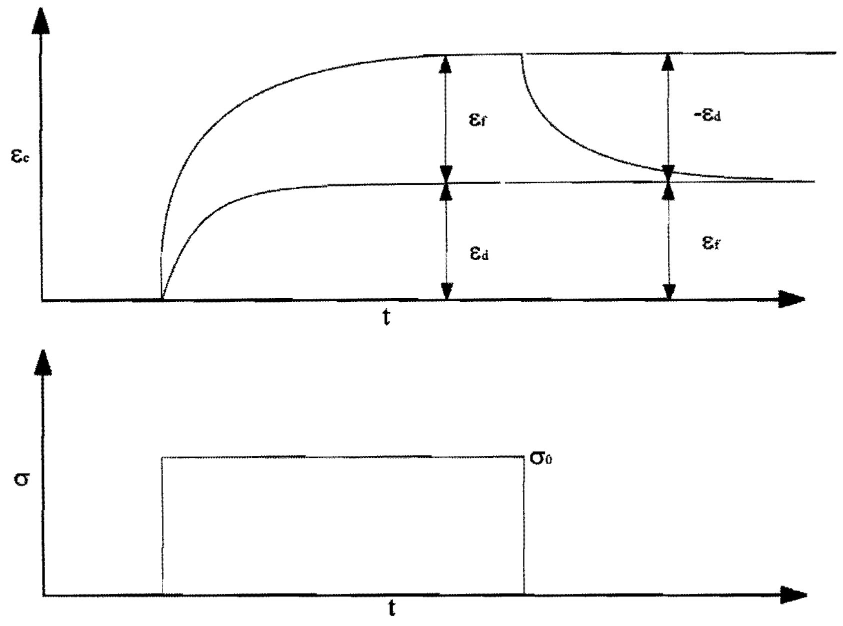

Figure 2-7: Creep components due to a load pulse

The total creep potential of a concrete specimen is usually described by a creep coefficient, $\phi(t, \tau)$, which is expressed as,

$$\phi(t,\tau) = \frac{\varepsilon_c(t)}{\varepsilon_e} \qquad (2.57)$$

where:

$\varepsilon_c(t)$	= Creep strain at time *t*.
ε_e	= Instantaneous elastic strain.
τ	= Age at loading.

This coefficient increases with time and is highly dependent on the concrete maturity at first loading. One of the largest uncertainties in creep problems is the magnitude of ϕ. A vast number of procedures are available for the calculation of ϕ, many of which calculate contributing portions to ϕ for the various components of the total creep strain.

The effective modulus method involves replacing the modulus used in the analysis with an artificial, effective modulus:

$$E_e(t,\tau) = \frac{E_c}{1+\phi(t,\tau)} \tag{2.58}$$

where:

$E_e(t,\tau)$	= Effective modulus at time t for a specimen loaded at time τ.
E_c	= Concrete modulus of elasticity at time zero.

Equation (2.58) allows the calculation of total deflection at time t, If only the creep increment in deflection is sought, equation (2.58) becomes:

$$E_e(t,\tau) = \frac{E_c}{\phi(t,\tau)} \tag{2.59}$$

The age-adjusted method allows an improved estimation of ϕ. Since the full loading is rarely instantaneously applied at time τ, as suggested by figure 2-7, the total load causing deflection is only active at some time later than τ. As mentioned earlier, ϕ is very sensitive to τ and should therefore be modified to account for this gradual increase in load. The age-adjusted method suggests the use of a factor, $\chi(t,\tau)$ smaller than unity, to reduce the magnitude of ϕ.

Equation (2.58) then becomes,

$$E_e(t,\tau) = \frac{E_c}{1+\chi(t,\tau)\phi(t,\tau)} \tag{2.60}$$

As for ϕ, various national building codes suggest procedures for the calculation of $\chi(t,\tau)$. For the purposes of this dissertation it is assumed that ϕ and χ are known and the focus falls on the implementation of these quantities in a creep analysis.

The method set out below is often referred to as the "Section Curvature Method" and is taken from Ghali and Favre (1986). The equation variables are illustrated in figure 2-8.

The creep curvature increment of a plain concrete member subjected to flexure and assumed to be uncracked may be expressed as:

$$\Delta\psi = \phi(t,\tau)\psi_e \tag{2.61}$$

where:

$\Delta\psi$	= Creep increment in curvature;
$\phi(t,\tau)$	= Creep coefficient at time t for loading at time τ;
ψ_e	= Elastic curvature at time τ.

Similar to equation (2.57), equation (2.61) modifies the elastic curvature for creep, based on the creep coefficient and the assumption that strain is linearly related to curvature.

Reinforcement tends to restrict concrete creep and the magnitude of this influence is a function of section geometry and reinforcement ratio. The effect of reinforcement on concrete creep can be expressed by a dimensionless parameter κ_c, as shown in equation (2.62):

$$\Delta\psi = \kappa_c\left[\phi(t,\tau)\left(\psi_e + \varepsilon_O\frac{y_c}{r_c^{\,2}}\right)\right] \tag{2.62}$$

where:

κ_c	=	Creep curvature coefficient, defined by equation (2.64);
ε_O	=	Axial strain at point O at time τ, point O is a reference point chosen at the centroid of the age adjusted transformed section;
y_c	=	y-coordinate of the centroid of A_c at time τ, measured downwards from the centroid of the age-adjusted transformed section.
$r_c^{\,2}$	=	I_c/A_c
I_c	=	Moment of inertia of A_c about an axis through the centroid of the age-adjusted transformed section.
A_c	=	Effective concrete area, full area for uncracked sections and the concrete compression zone for cracked members.

This dissertation neglects the effect of membrane action in the slab and equation (2.62) can then be simplified to:

$$\Delta\psi = \kappa_c\left[\phi(t,\tau)\psi_e\right] \tag{2.63}$$

The parameter κ_c can be calculated from:

$$\kappa_c = \frac{I_c + A_c y_c \Delta y}{\bar{I}} \tag{2.64}$$

where:

Δy = y-coordinate of the centroid of the age adjusted transformed section, measured downwards from the centroid of the transformed section at time τ.

$\bar{I}$ = Moment of inertia of the age adjusted transformed section about an axis through its centroid.

Figure 2-8, taken from Ghali & Favre, illustrates the variables used in equations (2.62) to (2.64). This approach is applied to the finite element method in section 3.2.

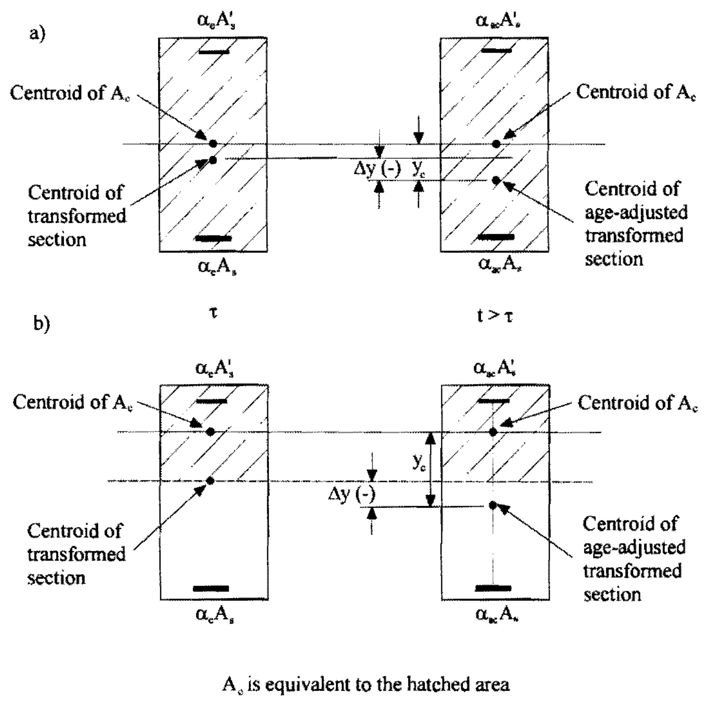

Figure 2-8: Creep section parameters for a) uncracked section; b) cracked section

(Reproduced from Ghali & Favre (1986))

2.4 Shrinkage

Shrinkage occurs when a hygral gradient exists between a concrete member and the surrounding environment. Pore and adsorbed water migrates from the concrete and causes a change in volume. Should this change in volume be restrained by reinforcement or support conditions, tensile stresses develop which could in turn cause cracking. When free shrinkage, denoted by ε_{cs}, is restrained by an unsymmetrical arrangement of reinforcement about the neutral axis of a member, an increase in curvature occurs.

The derivation below is taken from Kong & Evans (1987).

From basic theory, the shrinkage curvature can be written as,

$$\psi = \frac{\varepsilon_2 - \varepsilon_1}{d} \tag{2.65}$$

From the geometry of figure 2-9,

$$\varepsilon_2 = \varepsilon_{cs} - \frac{f_{c2}}{E_c} \text{ and} \tag{2.66}$$

$$\varepsilon_1 = \varepsilon_{cs} - \frac{f_{c1}}{E_c} \tag{2.67}$$

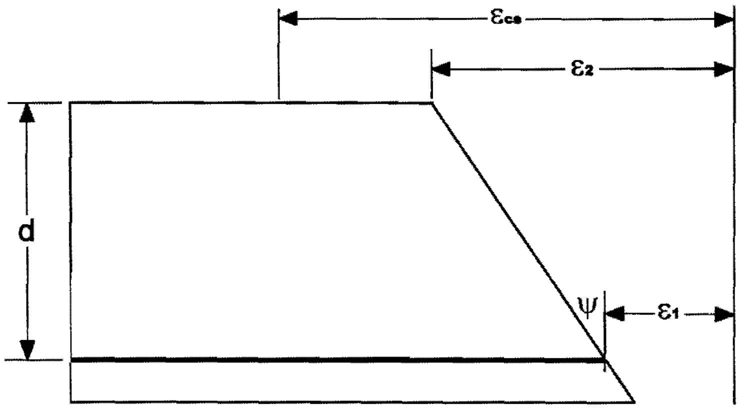

Figure 2-9: Shrinkage strains in a singly reinforced member

From the requirements of equilibrium,

$$f_{c1} = \frac{(f_s A_s)}{A} - \frac{(f_s A_s)e_s}{I}(d - e_s) \text{ and} \tag{2.68}$$

$$f_{c2} = \frac{(f_s A_s)}{A} + \frac{(f_s A_s)e_s}{I}e_s \tag{2.69}$$

where:

f_s	= Steel stress due to shrinkage ($E_s\epsilon_l$);
f_{c1}	= Concrete tensile stress at the tension reinforcement level due to shrinkage;
f_{c2}	= Concrete tensile stress at the top fibre of the section due to shrinkage;
e_s	= Eccentricity of the steel centroid with respect to the centroid of the transformed section;
A	= Concrete cross sectional area;
A_s	= Area of reinforcement;
I	= Moment of inertia of the age-adjusted transformed section.

Equation (2.65) can then be rewritten as:

$$\psi_{sh} = \frac{\varepsilon_{cs}\alpha_e A_s e_s}{I} \tag{2.70}$$

where α_e is defined as

$$\alpha_e = \frac{E_s}{E_c} \tag{2.71}$$

It should be noted that α_e is based on the effective concrete modulus and I is based on the age-adjusted transformed section.

Equation (2.70) holds for singly reinforced uncracked members, and very little error is involved by applying the equation to cracked sections as well (Kong & Evans, 1987).

This approach for singly reinforced sections is applied to the finite element method in section 3.3.

3 IMPLEMENTATION OF THE PROPOSED METHOD

3.1 Cracked Sections

Polak's approach to the problem of tension stiffening was applied almost without change. The author modified the algorithm suggested by Polak to allow for iteration after each model update, figure 7-3.

Both the Bilinear and Branson's method were used in conjunction with Polak's approach and compared to experimental results in section 4.2. The bilinear method required further development before being utilised in a manner similar to Branson's method.

Assuming that elastic relations still hold on average for cracked sections:

$$\psi_1 = \frac{M}{EI_1} \tag{3.1}$$

$$\psi_2 = \frac{M}{EI_2} \tag{3.2}$$

where the subscripts 1 and 2 refer to conditions 1 and 2 as described in section 2.2.
Substituting equations (3.1) and (3.2) into equation (2.43) yields an effective moment of inertia

$$I_e = \frac{I_1 I_2}{(1-\zeta)I_2 + \zeta I_1} \tag{3.3}$$

I_e can then be used to calculate α_x and α_y as described in section 2.2.2. It should be noted that the procedure for instantaneous cracked deflection and long-term cracked deflection differs. For long-term deflections a shrinkage analysis should precede the crack analysis, as shrinkage normally causes additional member actions that contribute to cracking.

A very simple convergence check was used in the crack analysis as follows:

- Step 1 : Calculate deflections using $I_{(i)}$, where i denotes the iteration step. For the first iteration $I_{(i)}$ corresponds to I_1.
- Step 2 : Calculate $I_{e(i+1)}$ using either of the two tension stiffening methods.
- Step 3 : Average $I_{e(i+1)}$ and $I_{(i)}$ and calculate the reduction factors α_x and α_y.
- Step 4 : Loop back to step 1 and repeat until $I_{e(i+1)} \approx I_{(i)}$.

This approach converged very quickly when using Branson's method but the bilinear method often exhibited oscillating divergence. This phenomenon was model and loading dependent and the algorithm had to be modified on a case by case basis to achieve a convergent solution.

3.2 Creep

Using equation (2.64) the factors κ_x and κ_y can be calculated based on the reinforcement ratios in those two directions, similar to α_x and α_y in section 2.2.2. To account for the different creep characteristics of cracked and uncracked sections, a creep analysis must be preceded by a crack analysis as described in section 3.1.

The elasticity matrix in equation (2.19) can then be modified as follows for the calculation of creep deflection increments for a cracked element:

$$[D]=\begin{bmatrix} \frac{E_x h^3}{12(1-\nu_x\nu_y)} & \frac{E_y \nu_x h^3}{12(1-\nu_x\nu_y)} & 0 & 0 & 0 \\ \frac{E_x \nu_y h^3}{12(1-\nu_x\nu_y)} & \frac{E_y h^3}{12(1-\nu_x\nu_y)} & 0 & 0 & 0 \\ 0 & 0 & \frac{G_1 h^3}{12} & 0 & 0 \\ 0 & 0 & 0 & G_2 h & 0 \\ 0 & 0 & 0 & 0 & G_3 h \end{bmatrix} \tag{3.4}$$

where:

$$E_x=\alpha_x\frac{E_c}{\kappa_x\phi}, \qquad E_y=\alpha_y\frac{E_c}{\kappa_y\phi} \tag{3.5}$$

$$G_1=\alpha_x\alpha_y\frac{G}{\kappa_x\kappa_y\phi}, \quad G_2=\alpha_x\frac{G}{\kappa_x\phi}, \quad G_3=\alpha_y\frac{G}{\kappa_y\phi} \tag{3.6}$$

$$\kappa_x=\frac{I_{cx}+A_{cx}y_{cx}\Delta y_x}{\bar{I}_x}, \qquad \kappa_y=\frac{I_{cy}+A_{cy}y_{cy}\Delta y_y}{\bar{I}_y} \tag{3.7}$$

$$\nu_x=\alpha_x\nu, \qquad \nu_y=\alpha_y\nu \tag{3.8}$$

with $\alpha_x \le 1$, $\alpha_y \le 1$ and ϕ the creep coefficient.

The parameters α_x and α_y are based on short term properties and κ_x and κ_y are parameters smaller than unity that modify the creep coefficient to account for the presence of the reinforcement.

The variables in equations (3.4) to (3.8) apply to cracked and uncracked section parameters as needed. Uncracked elements and fully cracked sections pose little difficulty. Partially cracked sections, on the other hand, require the calculation of an effective neutral axis.

It is proposed that the neutral axis for partially cracked sections be calculated based on the assumption that since the parameter α provides a measure of the extent of cracking it can also be used directly to modify the depth of the neutral axis:

$$y_e = y_1 \alpha \qquad (3.9)$$

where:

y_e	=	y-coordinate of the neutral axis of the partially cracked section, measured from the top of the section. This value should be larger than the cracked neutral axis coordinate and smaller than the uncracked value.
y_I	=	y-coordinate of the neutral axis for the uncracked section, measured from the top of the section.

The creep analysis algorithm is illustrated in figure 7-4.

3.3 Shrinkage

Equation (2.70) can be used to calculate x and y curvatures for each element, independent of loading. These curvatures need to be transformed into equivalent nodal loads in order to model the effect of boundary conditions on shrinkage in a finite element analysis.

Equivalent nodal loads are calculated simply from the following equation, utilising Gaussian numerical integration over the 4 sampling points:

$$\{P_{sh}\} = \int_A [B]^T [D] \{\varepsilon_{sh}\} \, dA \qquad (3.10)$$

where $[B]$ is calculated from equation (2.26) and $[D]$ is calculated from equation (3.4).

$\{\varepsilon_{sh}\}$ is the vector of shrinkage strains:

$$\{\varepsilon_{sh}\} = \begin{bmatrix} \chi_x \\ \chi_y \\ \chi_{xy} \\ \phi_x \\ \phi_y \end{bmatrix} = \begin{bmatrix} \psi_{xsh} \\ \psi_{ysh} \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (3.11)$$

The vector of shrinkage forces for each element node is calculated as:

$$\{P_{sh}\} = \begin{bmatrix} P_z \\ M_x \\ M_y \end{bmatrix} = \begin{bmatrix} 0 \\ M_{xsh} \\ M_{ysh} \end{bmatrix} \qquad (3.12)$$

All these forces are then assembled into a global force vector and the shrinkage deflections and forces are calculated with $[D]$ modified for creep.

The shrinkage analysis algorithm is illustrated in figure 7-5.

4 COMPUTATIONAL EVALUATION

4.1 Elastic Analysis

4.1.1 Aspect ratio studies

Shear locking was introduced in section 2.1.2 as a numerical issue where the Serendipity Mindlin element is concerned. This section evaluates extent of the shear locking problem and whether the issue is significant in the analysis of concrete slabs.

Shear locking causes an overestimation of plate stiffness for "thin" plates and it follows that plate thickness is the most significant factor influencing locking. Plates are therefore investigated over a range of span to thickness ratios.

The finite element analysis (FEA) mid-plate deflections of a simply supported and a clamped square plate, subjected to uniform transverse loading, are compared to the deflections obtained from classical methods. Navier's approach is used to calculate the exact plate deflection for the simply supported plate and Levy's solution is used for the clamped case, Ugural (1999). In both cases the Kirchhoff model of plate bending, i.e. thin plate theory, was employed. The resulting equations for a square plate are shown below:

$$w = 0.004066\frac{pl^4}{D} \text{ (simply supported)} \tag{4.1}$$

$$w = 0.001264\frac{pl^4}{D} \text{ (clamped)} \tag{4.2}$$

where p is the uniform load, l the plate length and

$$D = \frac{Eh^3}{12\left(1-\nu^2\right)} \tag{4.3}$$

The finite element layout and boundary conditions are illustrated in figure 4-1 and the results of the study are plotted on figure 4-2 and figure 4-3. In the aforementioned figures, w_{FEA} denotes lateral deflection at the centre of the plate as calculated with the finite element method for two integration schemes and w_{KIR}, the lateral deflection as calculated with equations (4.1) and (4.2). The finite element analysis employs Mindlin assumptions and the "exact" solutions employ Kirchhoff assumptions.

Clearly, numerical instabilities occur as the span to thickness ratio becomes large, regardless of the integration scheme employed. Reduced integration improves the performance of the element, but does *not* eliminate locking. For the serendipity element a 3x3 point Gaussian quadrature is exact, whereas a 2x2 point quadrature is a reduced integration scheme.

Although this finding is significant in analyses dealing with thin plates, reinforced concrete slabs rarely exhibit span to thickness ratios larger than 32. This ratio is represented by the vertical line in figure 4-2 and figure 4-3. As can be seen from these figures the deflection is at least overestimated, if not entirely accurate for ratios smaller than 32, even with exact integration.

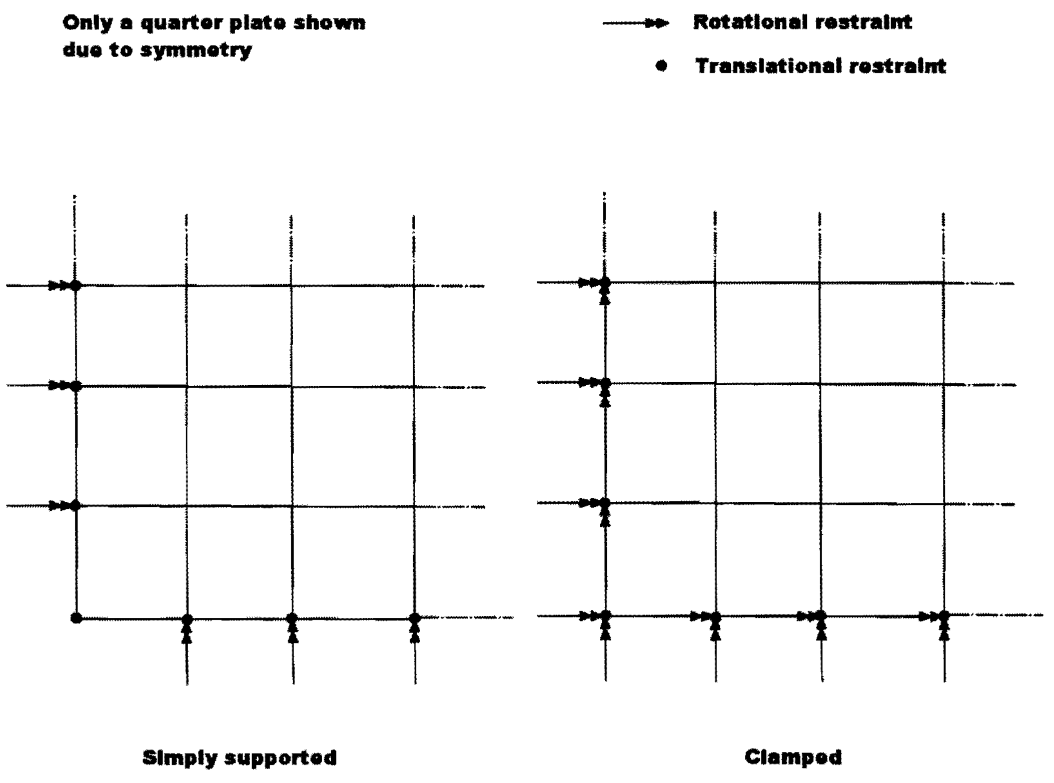

Figure 4-1: Element layout and boundary conditions

4.1.2 Convergence Studies

An issue that often arises in a finite element analysis is that of mesh density. The analyst always attempts to use the least number of elements and still obtain reliable results. This section studies various mesh densities on plates subjected to uniform loading in an attempt to find the optimum number of elements on a rectangular grid for slab problems.

The simply supported and clamped square plates illustrated in figure 4-1 are used with a varying number of elements. The plate analysed is a 6m square plate, 600mm thick subjected to a 5kPa distributed load. Both the Mindlin and Kirchhoff models for plate bending are used for analytical comparison. The analytical results for maximum deflection using the Kirchhoff assumptions are given in equations (4.1) through (4.3) and the results for the Mindlin model are given below, Liu (2002):

$$w = 0.00427\frac{pl^4}{D} \text{ (simply supported)} \tag{4.4}$$

$$w = 0.0015\frac{pl^4}{D} \text{ (clamped)} \tag{4.5}$$

where all variables are as defined in section 4.1.1.

As can be seen from figure 4-4 and figure 4-5, very little is gained from a mesh finer than 6x6, as far as accuracy is concerned. One should note that this result is valid for square plates subjected to uniform pressures only.

The curves labelled *Mindlin* plot the ratio of w_{FEA}/w_{exact}, where w_{exact} is calculated using equations (4.4) and (4.5). The curves labelled *Kirchhoff* uses a w_{exact} calculated from equations (4.1) and (4.2), w_{FEA} refers to the results of a finite element analysis (Mindlin assumptions, and 2x2 integration) throughout.

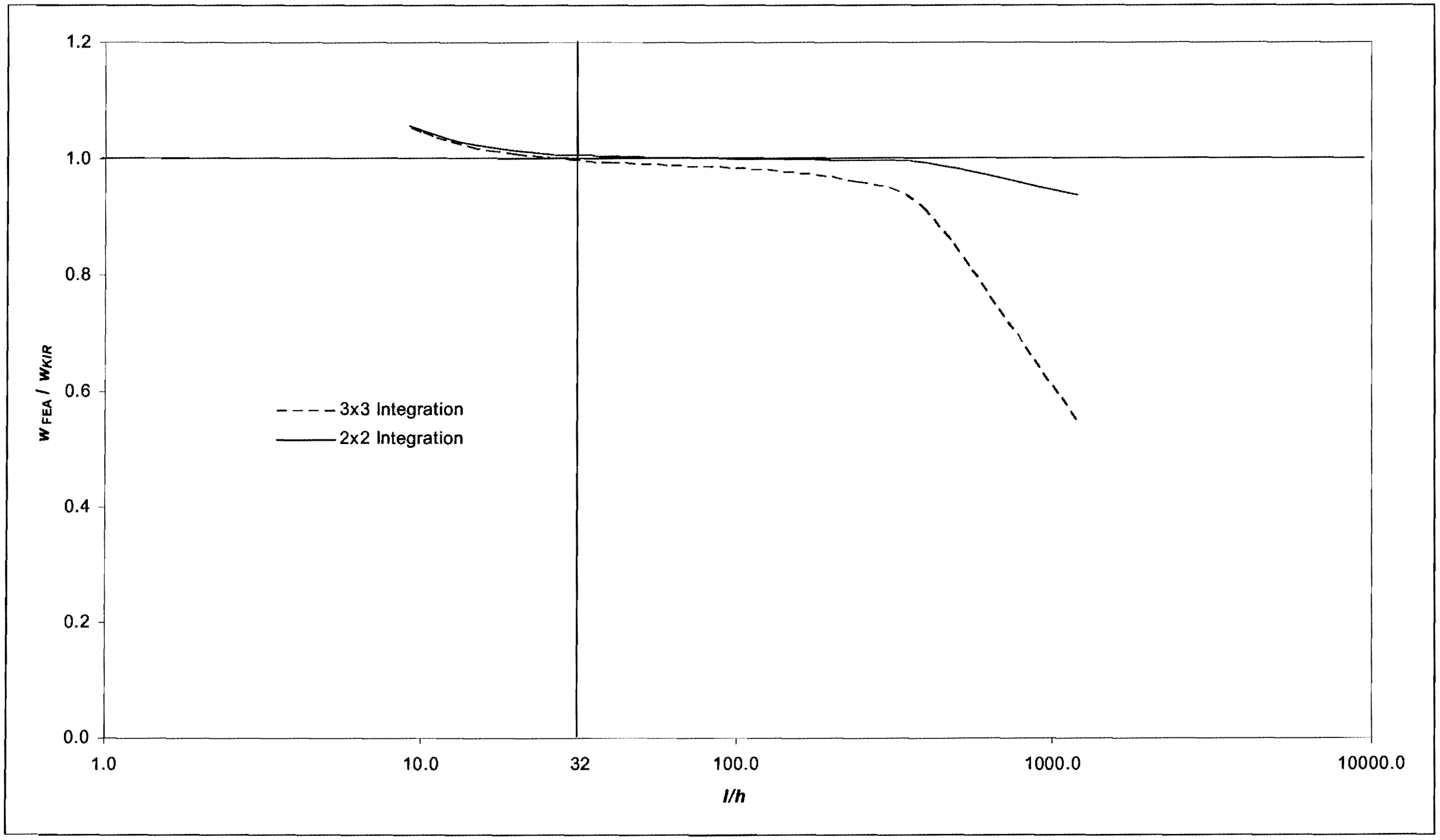

Figure 4-2: Aspect ratio study for a simply supported square plate

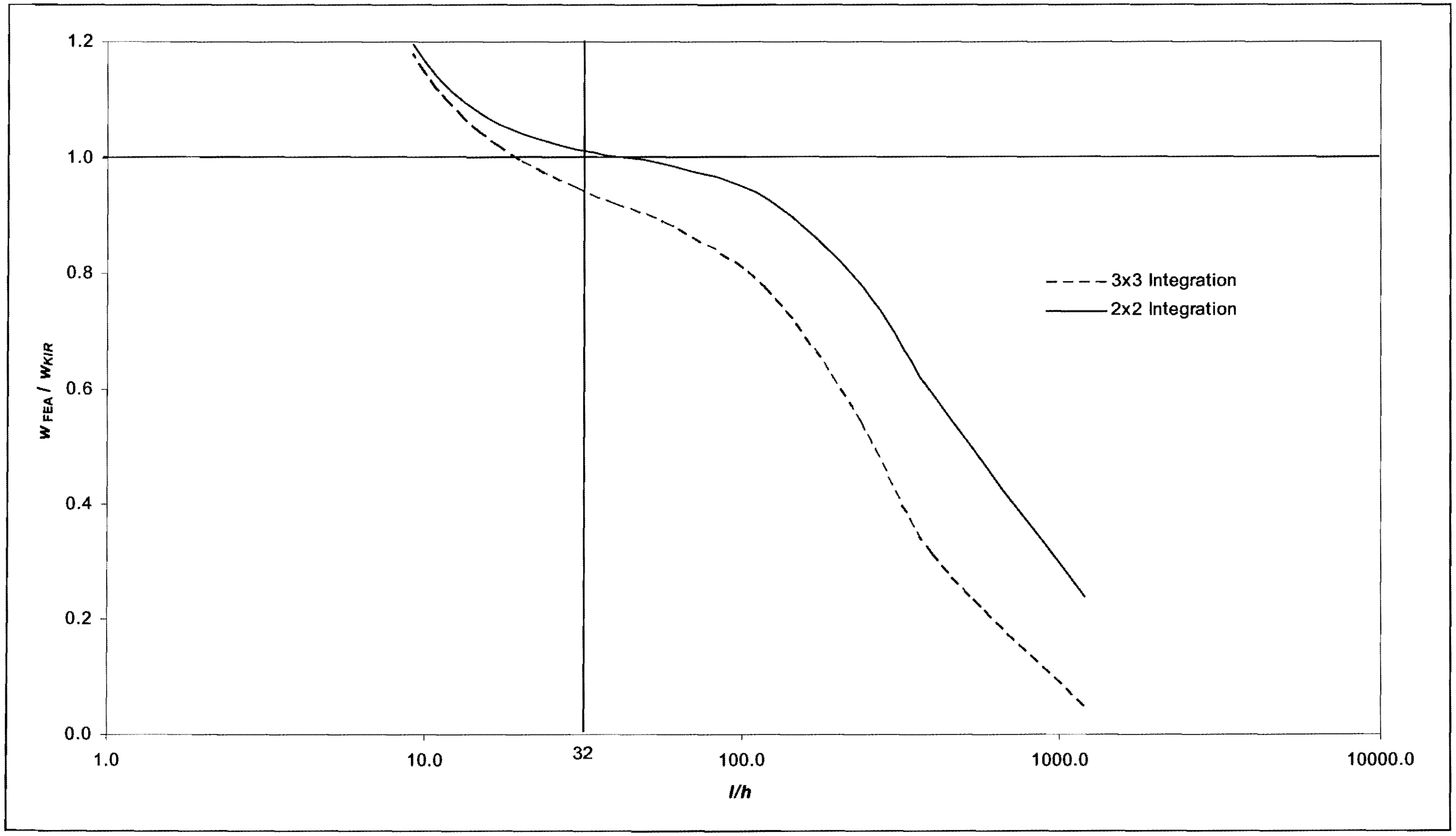

Figure 4-3: Aspect ratio study for a clamped square plate

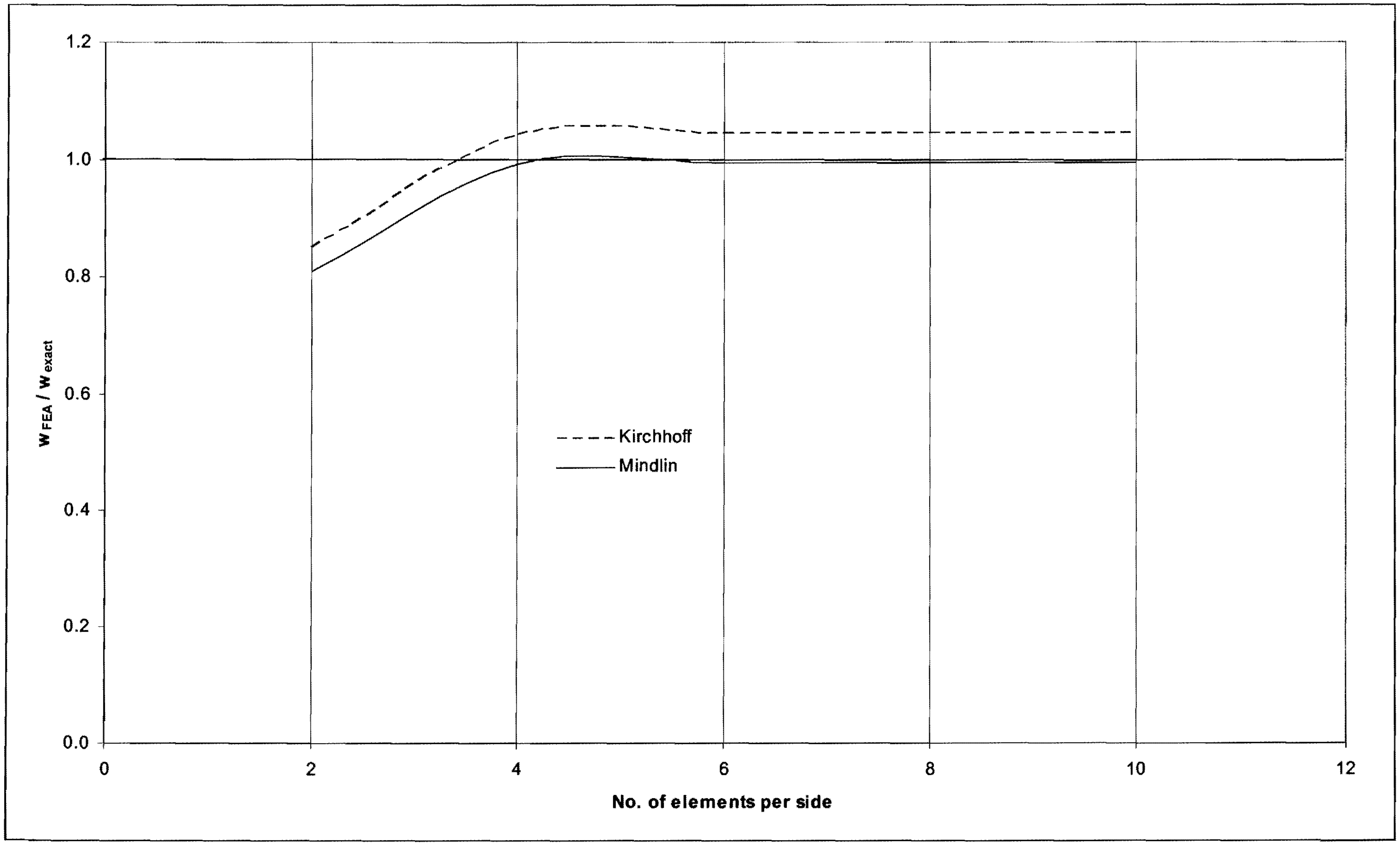

Figure 4-4: Convergence study for a simply supported square plate

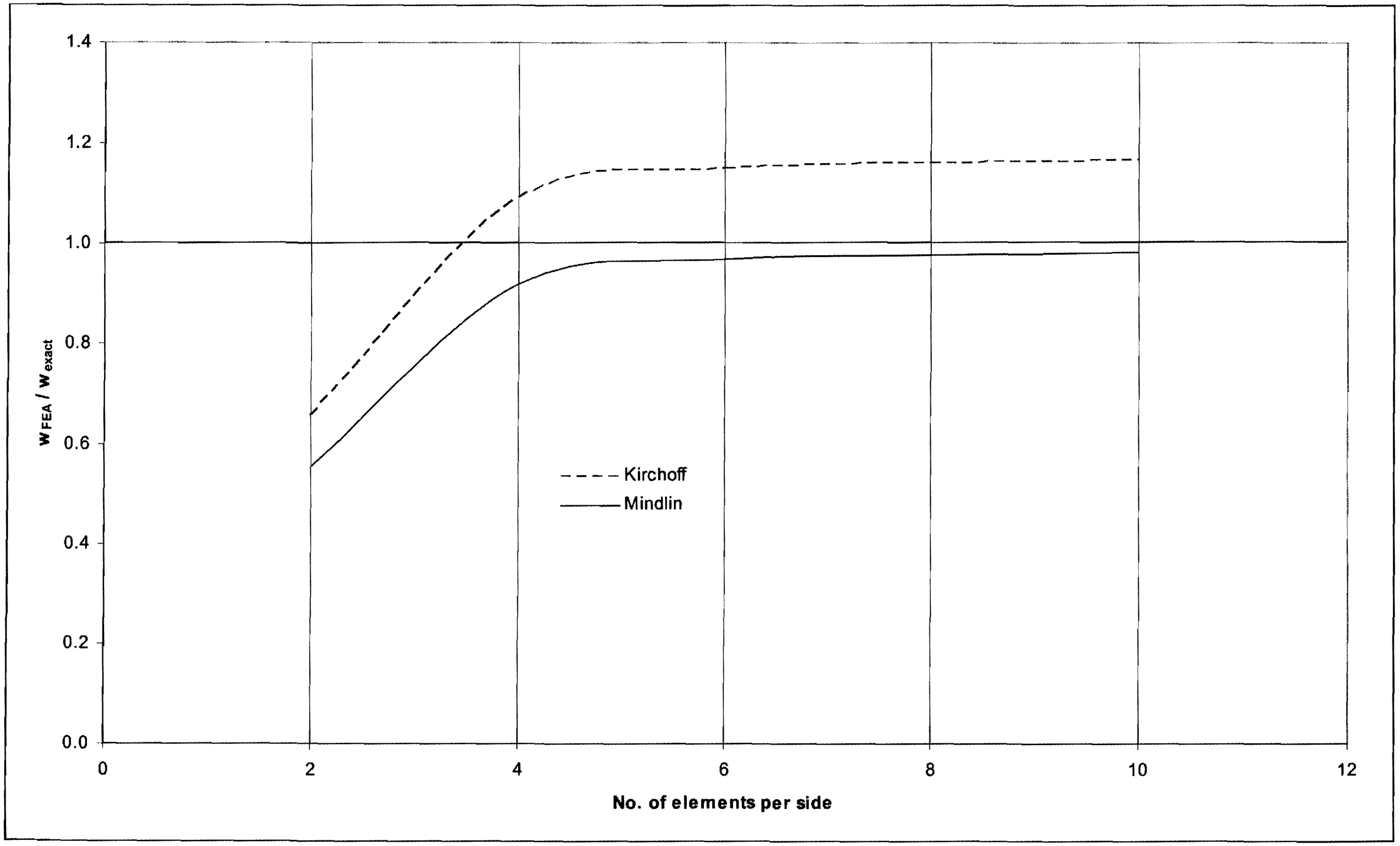

Figure 4-5: Convergence study for a clamped square plate

It is interesting to note that the element deteriorates in the case of the clamped plate, this indicates that the influence of shear deformation on flexural deflections of a plate is not solely dependent on the span to depth ratio, but also on boundary conditions.

4.2 Polak Slab Specimen

A slab tested by Polak (1994) was used to corroborate the results yielded by the effective stiffness method presented in section 2.2.2. The data from these slabs are used in this section to verify the software developed by the author and to test the applicability of the tension stiffening method presented in sections 2.2.1 and 3.1.

The specimen employed for comparison, labelled SM1, is illustrated in Table 4-1 and figure 4-6.

Dimensions (mm)	E_c **(GPa)**	ρ_x*	ρ_y*	d_x **(mm)**	d_y**(mm)**	ν
1625 x 1625 x 316	34.278	1.25%	0.42%	281	256	0.2

Table 4-1: Specimen properties (*per layer)

Specimen SM1, simply supported on two opposite edges, was loaded with uniaxial moments on the supported edges. The loading conditions and finite element model for the slab are shown in figure 4-6 and figure 4-7.

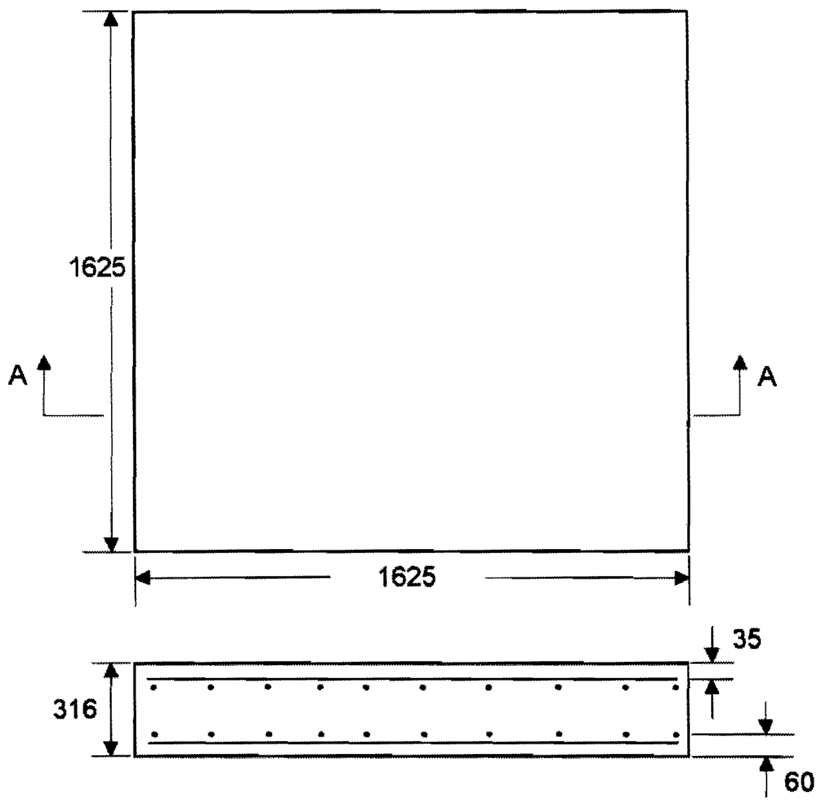

Figure 4-6: Specimen Geometry and Reinforcement

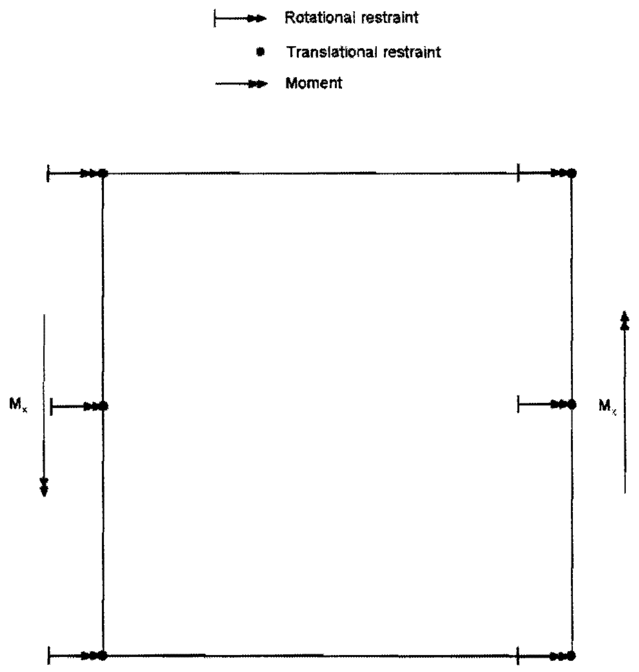

Figure 4-7: Finite element model for specimen SM1

A comparison of the results of the experimental and numerical analysis of specimen SM1 are plotted in figure 4-10. It is evident from figure 4-10 that the author's implementation of both the Bilinear and Branson's approach to tension stiffening compares favourably with the experimental data of specimen SM1 and the results of Polak.

4.3 Jofriet & McNeice Slab

Jofriet and McNeice (1971) performed a point loading test on a corner supported slab, the properties of which are indicated in Table 4-2. The point load was applied to the centre of the slab.

Dimensions (mm)	E_c **(GPa)**	ρ_x	ρ_y	$d_{x,y}$**(mm)**	v
914 x 914 x 44	28.623	0.85%	0.85%	33	0.15

Table 4-2: Specimen properties

The specimen geometry is illustrated in figure 4-8 and the finite element model in figure 5-9.

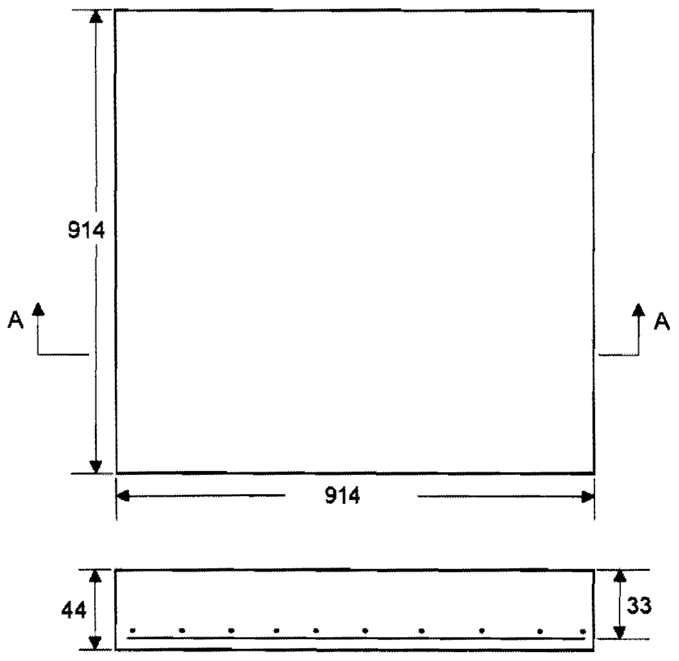

Figure 4-8: Specimen Geometry and Reinforcement

The slab model consists of a 6x6 mesh with the translational degrees of freedom restrained at the corner nodes. The slab was subjected to a central point load and deflections measured at point A as indicated. With the element mesh as shown, this point fortuitously coincides with a mid-edge node of a central element.

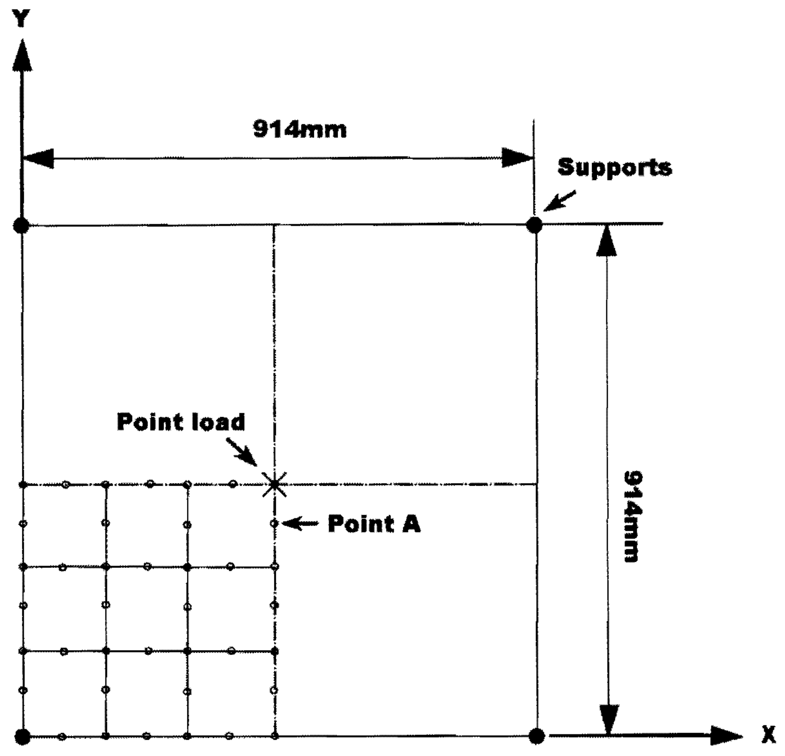

Figure 4-9: Jofriet and McNeice slab model

The results of both Polak and the author's analysis are plotted against the experimental data of Jofriet and McNeice in figure 4-11. It should be noted that Branson's approach yields results far superior to the bilinear approach. Careful investigation of the parameters influencing these two methods reveals that the bilinear method is very sensitive to changes in reinforcement ratio.

The curve of the effective moment of inertia versus applied moment curve changes shape with lower reinforcement ratios when using the Bilinear method, whereas the curves retain a similar shape when using Branson's method, see figure 4-12 for details. The figure implies that the bilinear method becomes unreliable with lower reinforcement ratios. This finding casts significant doubt on the usefulness of bilinear method in flat slab problems where reinforcement ratios are typically fairly low.

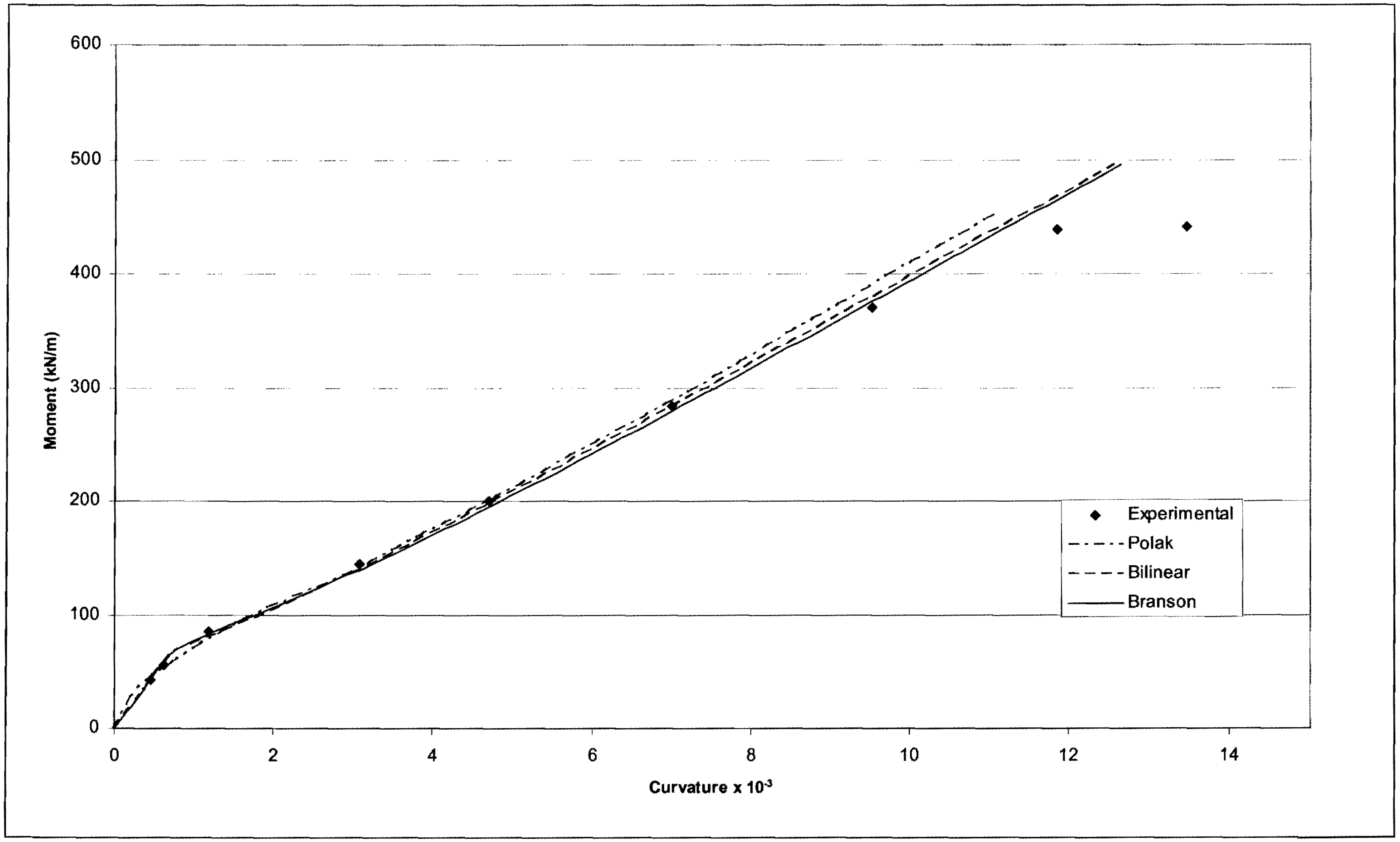

Figure 4-10: Specimen SM1 analysis results

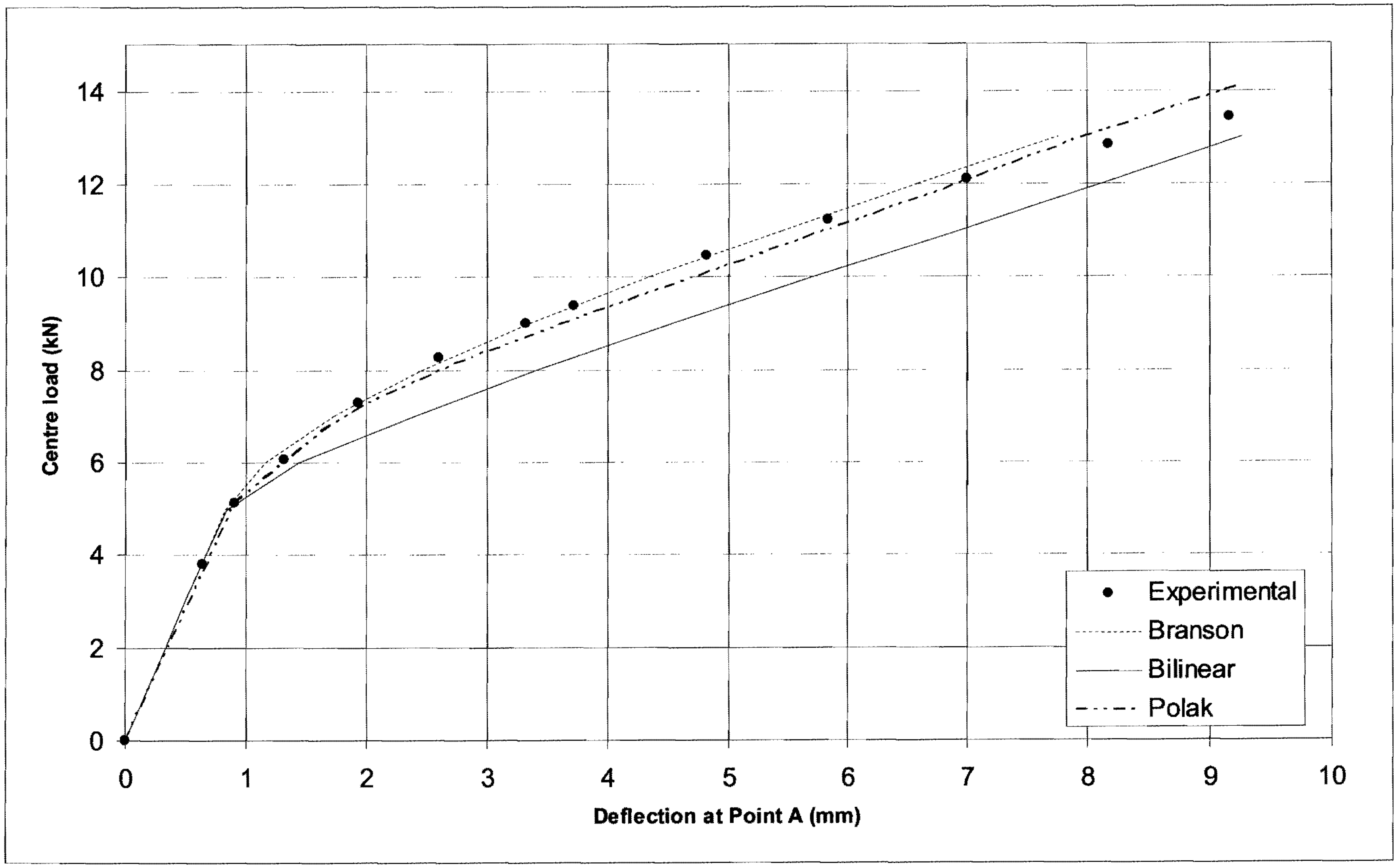

Figure 4-11: Jofriet and McNeice slab results

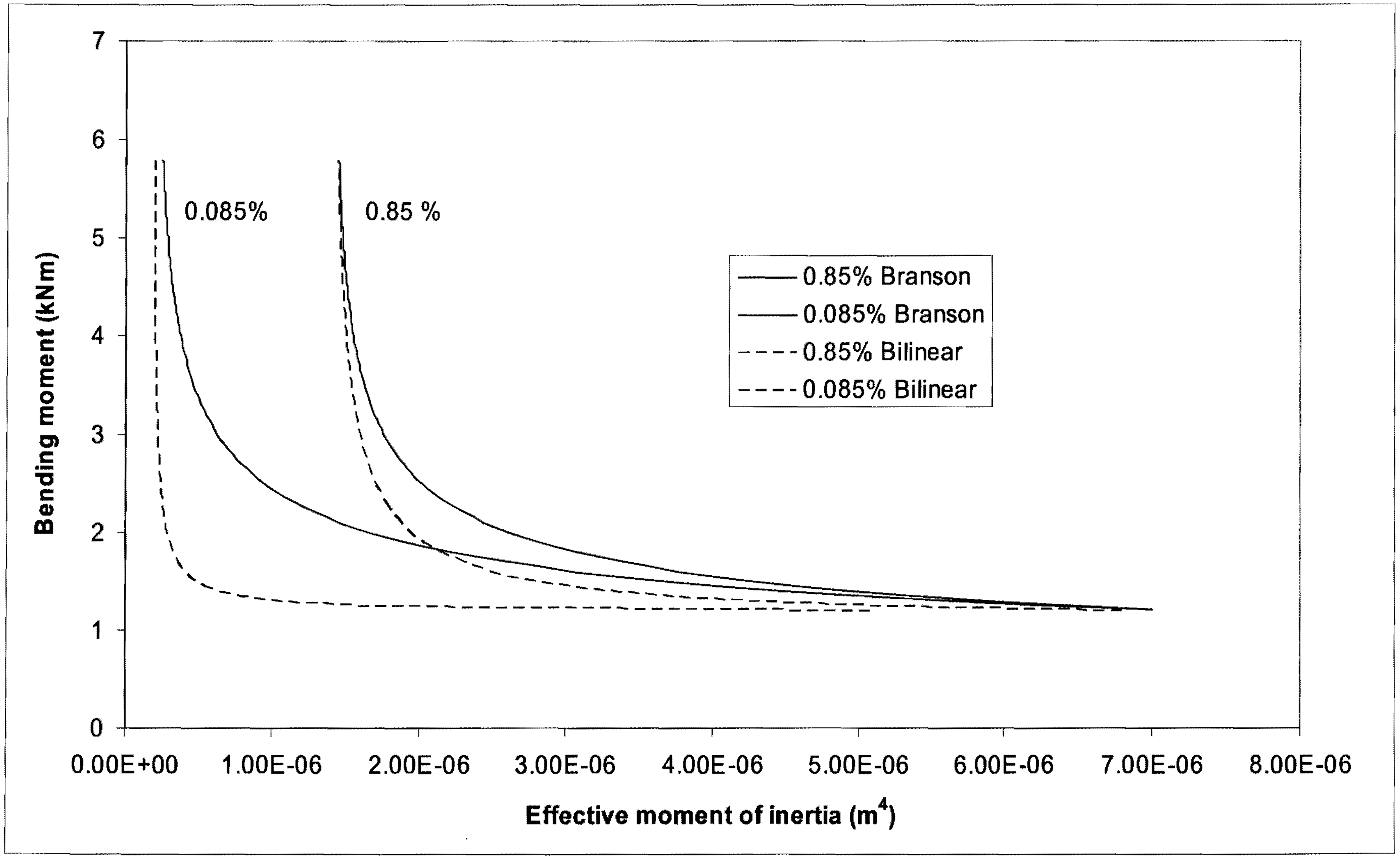

Figure 4-12: Comparison of the two tension stiffening methods

4.4 Haddad's Beam

The data for this beam test as well as the hand-calculation deflection results were obtained from Neville (1970). The cross sectional properties, layout and loading of the tested beam are illustrated in figure 4-13. It should be noted that the original test was carried out using imperial units.

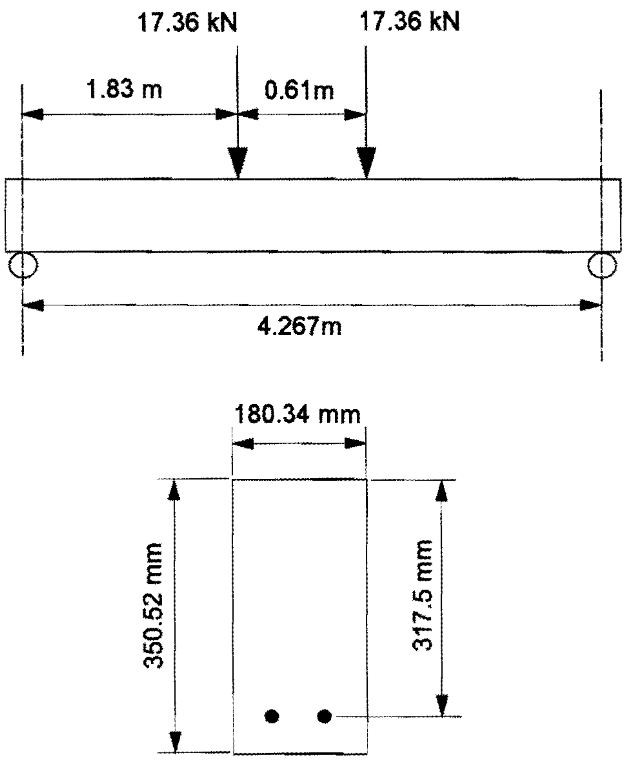

Figure 4-13: Geometry and loading of the beam tested by Haddad

Tabulated below are some material and geometric properties as established by Haddad:

Concrete cylinder strength, f_c'	26.34 MPa
Rupture modulus, f_r	3.1 MPa
Young's modulus, E	22.76 GPa
Free shrinkage strain, ε_{cs}	-204×10^{-6}
Creep coefficient, ϕ	2
Percentage tension reinforcement	1.42%

Table 4-3: Material properties of Haddad's beam

The rupture modulus was calculated, Neville (1970), from:

$$f_r = 0.6\sqrt{f_c'} \qquad (4.6)$$

The beam, modelled as a slab of the same width, was approximated with a 1x20 element mesh, as illustrated in figure 4-14. Three separate analyses were performed and compared with Haddad's experimental data as well as the results obtained by Neville with hand-calculation methods.

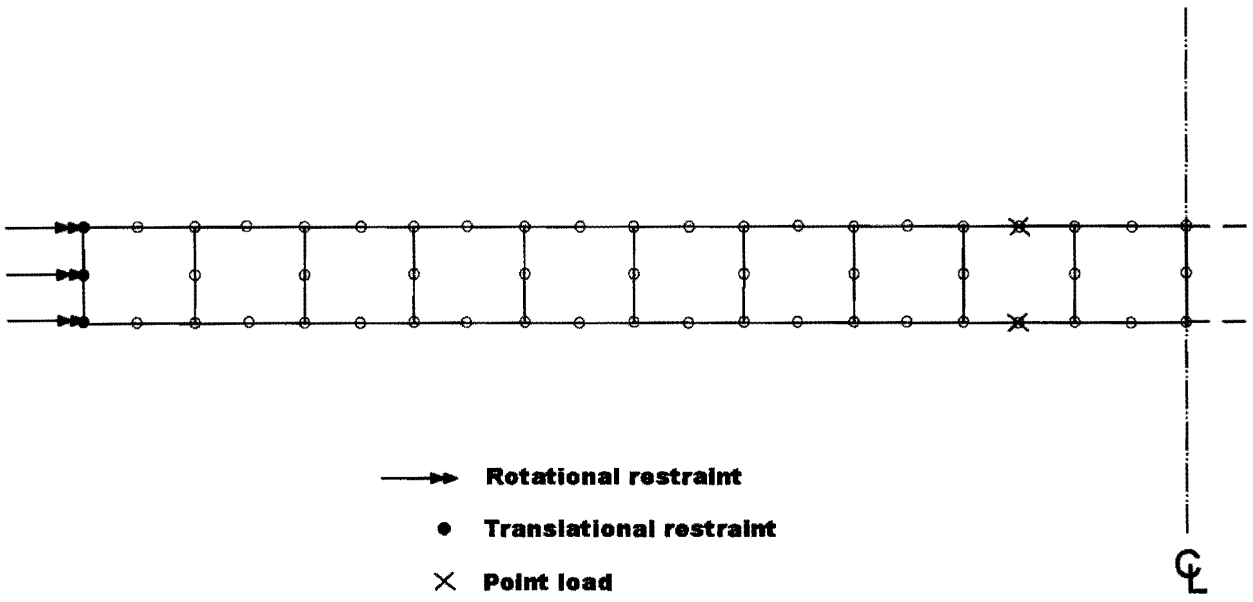

Figure 4-14: Plan view of the element layout

A simple elastic analysis, neglecting cracking and tension stiffening, yields a mid-span deflection of 3.67 mm which compares well with the value of 3.71 mm as predicted by analytical methods.

The table below compares the mid-span deflections at time infinity of the finite element analysis, Neville's results and Haddad's data.

	FEA (mm)	Neville (mm)	Haddad (mm)
Elastic with cracking (Branson)	5.8	5.28	5.84
Creep with cracking	3.14	3.81	
Shrinkage	1.28	1.04	
Total long term	10.22	10.13	10.9

Table 4-4: Comparison between the proposed model, Neville and Haddad's results

It should be noted that the method proposed in this dissertation assumes that the principle of superposition applies to the four stated components of time dependent deflection: elastic, cracked, shrinkage and creep.

The table indicates that the finite element analysis correlates extremely well with both the hand calculation methods employed by Neville and the actual results obtained by Haddad.

4.5 Simplified Analysis of a Slab Panel

As a further verification, a slab panel analysed with the hand-calculation method set out in section 2.1.1 is compared with the proposed finite element method. This panel is taken from Ghali and Favre (1986) and the detail is given below and in figure 4-15.

For the purposes of the hand calculation it is assumed that the moments and required reinforcement are known and only the final, long-term deflection is sought. Naturally, for the finite element approach, only slab geometry, required reinforcement and material properties are needed.

The panel is loaded with a uniformly distributed load q = 8.42 kN/m^2 on a 7m x 7m span. The depth of the slab, h = 200mm, and the average effective depth of the tension reinforcement in the x and y directions, d_t = 160mm. The modulus of elasticity at the time of loading E_c = 25GPa with the creep and aging coefficients $\phi(t, \tau)$ = 2.5 and $\chi(t, \tau)$ = 0.8, respectively. The modulus of rupture is given as f_r = 2MPa and the modulus of elasticity of the reinforcement E_s = 200GPa.

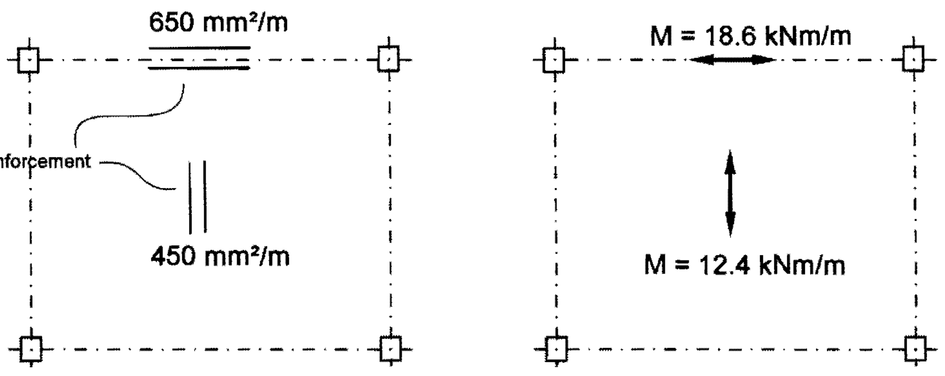

Figure 4-15: Reinforcement layout and moments of the slab panel

Top reinforcement is conspicuous in its absence in the figure above, and the assumptions made for the finite element analysis are elaborated upon in section 4.5.2. The hand calculation method on the other hand, oddly neglects the influence of negative reinforcement and cracking at the column supports.

4.5.1 Hand Calculation

The calculation in this sub-section is taken directly from Ghali and Favre (1986).
Equation (2.7) yields an uncracked moment of inertia, neglecting reinforcement as,

$$I_g = \frac{0.2^3}{12\left(1-0.2^2\right)} = 694\text{x}10^{-6}\text{m}^4/\text{m}$$

Using a deflection coefficient table based on equation (2.1) Ghali and Favre (1986), D, δ_{EF} and δ_{AB} are calculated as:

$$D = 0.00482\frac{ql^4}{E_cI_g} = 5.6mm$$

$$\delta_{AB} = 0.00342\frac{ql^4}{E_cI_g} = 3.97mm$$

$$\delta_{EF} = D - \delta_{AB} = 1.63mm$$

Column strip crack curvature coefficients are calculated using equations (2.46), (2.47) and interpolated with equation (2.48):

$$\kappa_{s1} = \frac{I_g}{I_1} = 0.98$$

$$\kappa_{s2} = \frac{I_g}{I_2} = 7$$

The cracking moment and the crack interpolation coefficient are calculated as:

$$M_r = \frac{f_rI_g}{y} = 13.33kNm/m$$

$$\zeta = 1-\beta_1\beta_2\left(\frac{M_r}{M}\right)^2 = 0.74$$ with $\beta_2 = 0.5$ for long-term loading.

The effective crack coefficient and cracked mid-span column strip deflection is then:

$$\kappa_s = (1-\zeta)\kappa_{s1} + \zeta\kappa_{s2} = 5.45$$

$$\delta_{AB} = 5.45 \times 3.97 = 21.65mm$$

The creep curvature coefficients are found using equation (2.64) and interpolated in a similar manner to calculate an ultimate creep deflection of 9.55mm. The middle strip deflections can be calculated in the exact same manner, Tables 4.5, 4.6 and 4.7 summarise the results of the comparison.

4.5.2 Finite Element Analysis

The finite element model consists of a 6x6 element mesh with the corners fixed against all displacements and the edges fixed against rotation about an axis parallel to the edge as shown in figure 4-16.

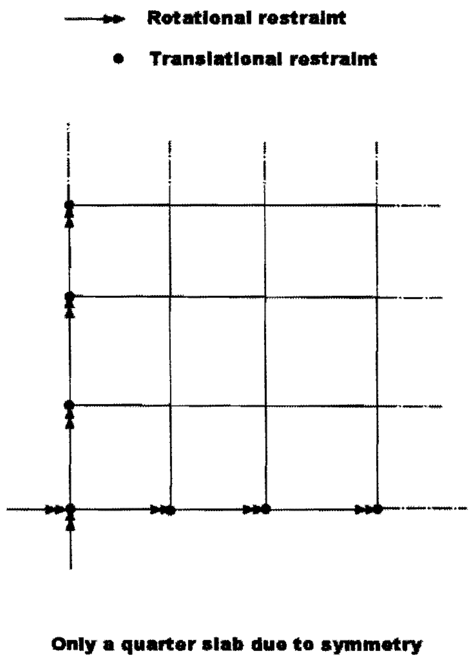

Figure 4-16: Finite element model

For the purposes of the finite element analysis, symmetric double reinforcement (top and bottom) is assumed as shown in figure 4-17.

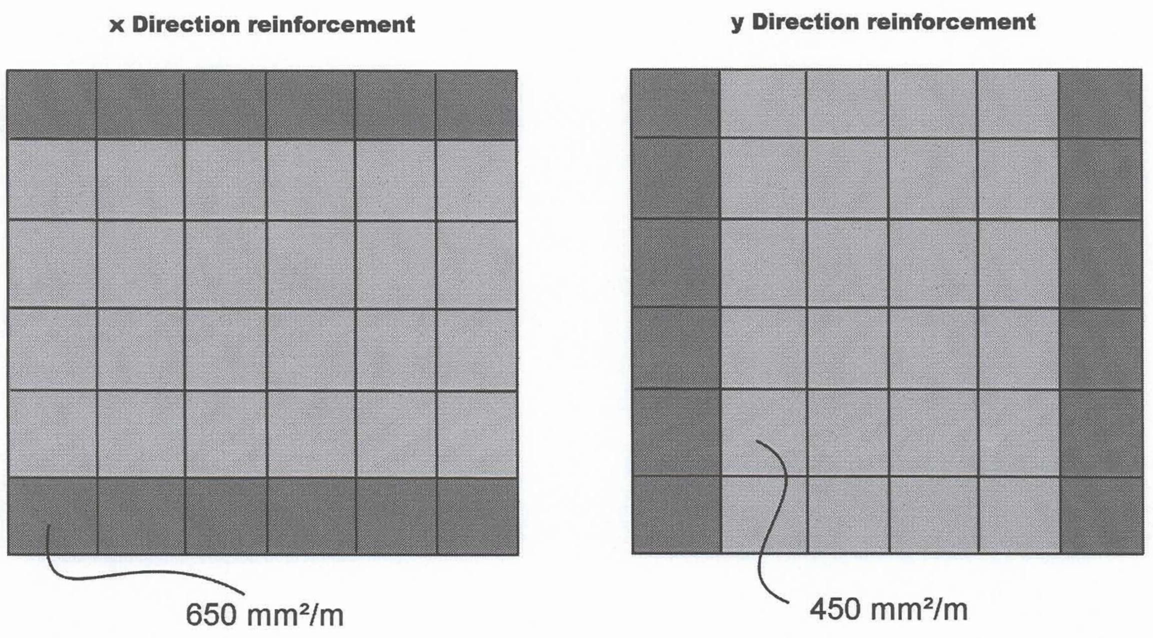

Figure 4-17: Assumed reinforcement layout

An uncracked, elastic finite element analysis yields a total mid-panel deflection, D = 6.97mm and a mid-column strip deflection δ_{AB} = 5.27mm, compared to 5.6mm and 3.97mm as calculated in the preceding section.

The results for long-term cracked deflection and creep deflection are set out in the tables below.

	Hand calculation (mm) (Ghali & Favre, 1986)	FEA (mm)
Cracked: Branson		29.73
Cracked: Bilinear	21.65	36.28
Creep: Branson		5.56
Creep: Bilinear	9.55	6.44
Total: Branson		35.29
Total: Bilinear	31.2	42.72

Table 4-5: Column strip deflections

	Hand calculation (mm) (Ghali & Favre, 1986)	**FEA (mm)**
Cracked: Branson		2.08
Cracked: Bilinear	1.6	2.26
Creep: Branson		2.56
Creep: Bilinear	3.04	2.74
Total: Branson		4.64
Total: Bilinear	4.64	5

Table 4-6: Relative middle strip deflections

	Hand calculation (mm) (Ghali & Favre, 1986)	**FEA (mm)**
Cracked: Branson		31.81
Cracked: Bilinear	23.25	38.54
Creep: Branson		8.12
Creep: Bilinear	12.59	9.18
Total: Branson		39.93
Total: Bilinear	35.84	47.72

Table 4-7: Total mid-panel deflections

It is clear that use of the bilinear method consistently results in a higher deflection than is the case when Branson's method is employed. When viewed in the light of the discussion in section 4.3, it must be said that the bilinear method is unsuited for the purposes of this dissertation.

4.6 Cardington Slabs

A full scale seven storey concrete frame was erected and investigated at the BRE's (Building Research Establishment) Large-Building Test Facility in Cardington in the UK as part of the European Concrete Building project. The floors consists of flat slabs and deflection measurements were published by Vollum & Hossain (1998).

The publications concerning this building do not mention the exact reinforcement ratios but the project brief, Chana et al. (1998), contains enough data, table 4-8, to infer the designed reinforcement from a design calculation to Eurocode 2.

Parameter	Value
Dead load (Load combination factor)	5 kPa (1.35)
Live Load (Load combination factor)	2.5 kPa (1.5)
Panel dimensions	7.5m x 7.5m x 250mm
Column dimensions (Internal)	400mm x 400mm x 3.75m
Concrete	C37

Table 4-8: Cardington slabs parameters

A design calculation utilising the equivalent frame method, yields required reinforcement in the order of 360mm²/m for both the hogging and sagging moment regions. This reinforcement area and the parameters shown in Table 4-9 and Table 4-10, are used to calculate the long-term deflection with the method proposed by the author. The finite element mesh used is identical to that of figure 4-16 and the reinforcement shown in figure 4-17 is modified to 360mm²/m.

Parameter	Value
t_0, t_1, t_2, t_3 (Time)	2 days, 12 days, 300 days, 1000 days
w_0, w_1, w_2 (Sustained service load)	6.75 kPa, 10.7 kPa, 9kPa
E_0, E_1, E_2 (Modulus of elasticity)	27GPa, 33GPa, 33GPa
f_{r0}, f_{r1}, f_{r2} (Modulus of rupture)	2.7MPa, 3.6MPa, 3.6MPa
Concrete	C37, (35MPa)

Table 4-9: Cardington time dependent slab parameters

Time (days)	$\phi(t,t_0)$	$\phi(t,t_2)$
$t_0 = 2$	0	-
$t_1 = 12$	0.57	-
$t_2 = 300$	1.42	0
$t_3 = 1000$	1.72	1.03

Table 4-10: Creep coefficients

Two analyses were performed:

- FEA (2 steps) – In this analysis, creep deflection was calculated using properties from t_0 to t_2 in the first step, and a second step calculated creep deflections from t_2 to t_3. The first step used the 6.75kPa load and the second step 9kPa.
- FEA (1 step) – Here creep deflection was calculated in a single step from t_0 to t_3 using 9kPa.

The measured deflections are plotted against the results of the finite element analysis in figure 4-18.

The finite element analysis correlates well with the experimental data up to the application of the 9kPa load at t = 300 days. It is clear that the load history is of great importance when calculating long-term deflections and that the proposed method does not perform extremely well when faced with varying load histories. This problem would be exacerbated were the sustained load to decrease at any time, since full creep recovery would erroneously be shown by the proposed method.

Despite these failings, the method predicts the 1000 day creep deflection within +12% when load history is included and within -40% when load history is neglected.

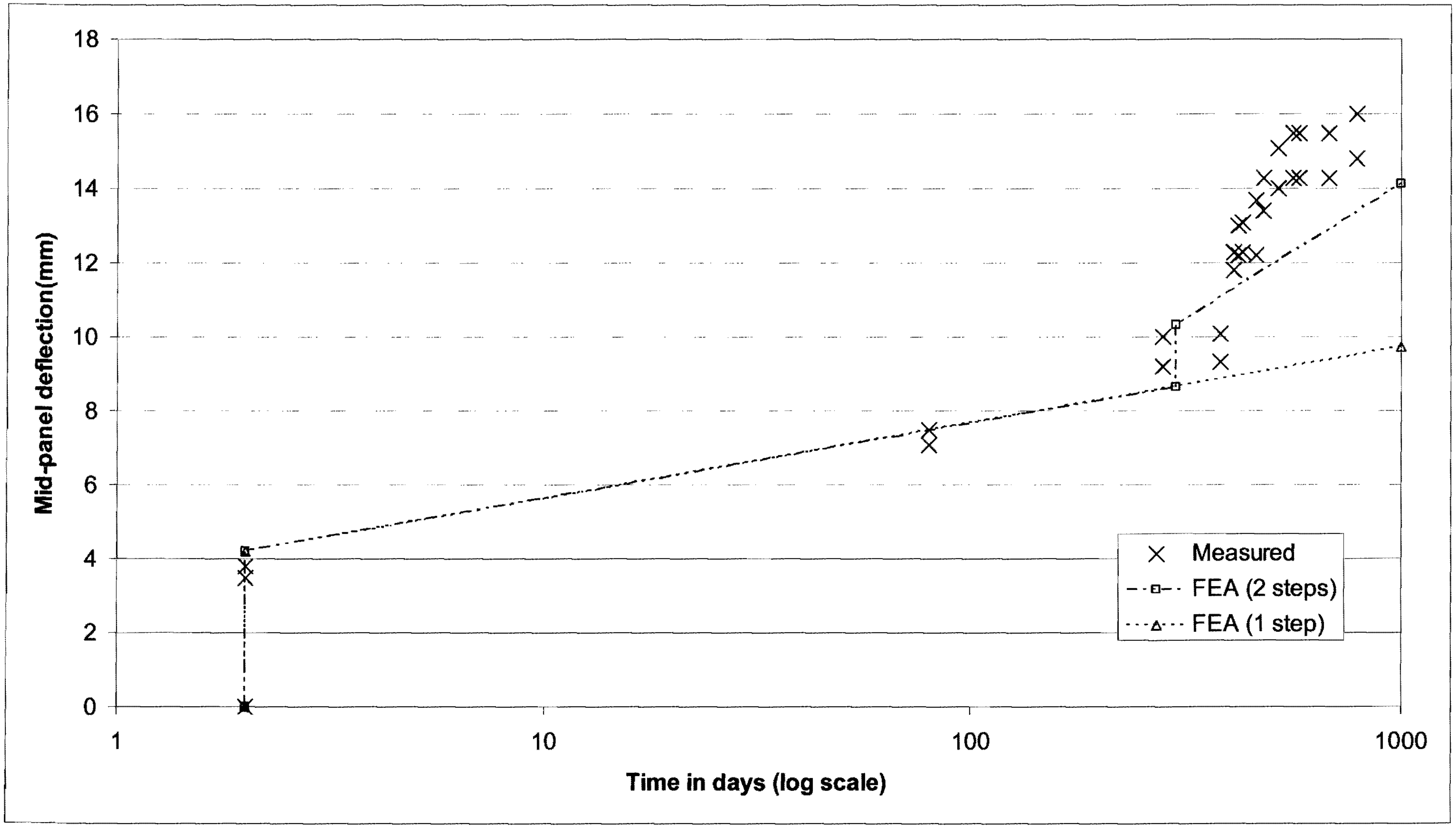

Figure 4-18: Finite element analysis plotted on the Cardington data

5 DISCUSSION AND CONCLUSIONS

The basic premise of the proposed method is the extrapolation of concrete beam behaviour to slab problems. The cracked and creep behaviour of beams is applied with slight modification to a plate or slab. Since slabs are subjected to biaxial stress states, the assumption that biaxial behaviour is equivalent to the superposition of two uniaxial solutions is an obvious simplification.

The error introduced by the simplification is of dubious importance when the inaccurate prediction models for the calculation of creep and shrinkage influences are taken into account. The equivalent frame approach will produce conservative estimates of long term deflection as slabs exhibit higher flexural stiffness due to torsional interaction.

This assumption has been the basis of many a simplified method in the past and the proposed method merely applies it to the finite element method.

5.1 Elastic deflections

Figures 4.2 through 4.5 demonstrate that the Mindlin element compares very well with classical solutions for simply supported plates. As expected, some divergence occurs at smaller plate thicknesses due to numerical issues. Since practical span to depth ratios of concrete slabs are rarely large, the Mindlin formulation yields elastic results of acceptable accuracy for this class of problems (span to depth ratios less than 32).

5.2 Cracked deflections

Polak's effective slab stiffness method produces acceptable results for both uniaxial and biaxial moment conditions as illustrated by figures 4.10 and 4.11. Although the Bilinear and Branson's method yield very similar results in the finite element analysis of slab SM1, the bilinear method performs poorly with low reinforcement ratios.

Authors such as Park and Gamble (2000) have objected to the use of Branson's method in slab problems due to the fact that this entirely empirical equation was developed for beams. They argue that slab steel ratios are often orders of magnitude smaller than those of beams and it follows that the tension stiffening effect in slabs would be far lower than predicted by Branson's equation.

Based on the findings of this dissertation, Branson's effective moment of inertia is preferred over the bilinear method for use in a finite element analysis.

5.3 Creep deflection

The proposed method of incorporating creep effects into a finite element analysis compares well with the experimental results of Haddad's beam and Neville's simple analysis. Branson's method again yields results superior to the bilinear method as far as the influence of cracking on creep deflections is concerned.

The proposed method has much to recommend it:

- The influence of reinforcement on creep is taken into account.
- Movement of the neutral axis due to cracking is incorporated.
- Only two parameters are required to quantify the creep strains of the material.

The creep method developed in this dissertation compares favourably with the concrete frame tested at the Cardington facility when constant loading is assumed. As mentioned in the previous section, some work is required to accommodate varying load histories.

5.4 Shrinkage deflection

As with creep, very little data is available on the shrinkage behaviour of flat slabs and the same procedure used for beams was implemented, based on the single free shrinkage parameter. The proposed method compares very well with Haddad's beam. It should be noted that the proposed method to include shrinkage deflection in the analysis fails with clamped beams or plates since rotation, and not only axial deformation, is prevented in these cases. Further analytical work is required to apply a complete shrinkage analysis to models with clamped boundary conditions.

5.5 Recommendations and Suggestions

The applicability of Polak's method using Branson's effective moment of inertia has been well demonstrated, although the boundary conditions for flat slabs do raise some concerns as to its use in the serviceability design of these structures.

Further work could include different plate formulations and more rigorous methods of estimating the magnitudes of the reduction factors for cracking and creep. Specifically the average moment for use in Branson's equation and the influence of both cracking and creep on the shear characteristics of flat slabs. Varying, and especially decreasing, load histories need to be considered in greater detail as few practical slabs are subjected to lifetime constant loading.

A point raised by one of the research groups at Cardington, was that with slender slabs dynamic effects also impact the magnitude of long-term deflections. Further work in this aspect would make the method widely applicable.

6 REFERENCES

American Concrete Institute 1999, Building Code Requirements for Structural Concrete, *ACI 318-99.*

Armer, G.S.T. 1968, Discussion of the Wood Paper (1968), *Concrete*, Vol 2, pp 319-320.

Bathe, K. 1982, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall Inc., New Jersey.

Bažant, Z.P. 2001, Prediction of concrete creep and shrinkage: past, present and future, *Nuclear Engineering and design*, Vol 203, pp 27-38.

Bažant, Z.P. Baweja, S. 1995, Creep and shrinkage prediction model for analysis and design of concrete structures – model B3, *Mater. Struct*, Vol 28, pp 357-365.

Bažant, Z.P. 1993, Creep and Shrinkage of Concrete, *Proceedings of the 5th International RILEM Symposium.*

Bensalem, A. 1997, Discussion of the Effective Stiffness Model for Reinforced Concrete Slabs, *Journal of Structural Engineering*, Vol 122, No 9, pp 1695-1696.

Borland Software Corporation 2001, *Borland Delphi Developer's Guide.*

Branson, D.E. 1968, Design Procedures for Computing Deflection, *ACI Journal*, Vol 65, No 9, pp 730-742.

Chan, H.C. Cheung, Y.K. Huang, Y.P. 1994, Nonlinear Finite Element Analysis of Reinforced Concrete Plates and Shells under Monotonic Loading, *Computers and Structures*, Vol 53, No 5, pp 1099-1107.

Chana, P. Judge and C. Moss, R. 1998, In-situ Building Performance Research – An Overview, *Proceedings of the 3rd Cardington Conference.*

Cope, R.J. Clark and L.A. 1984, *Concrete Slabs Analysis and design*, Elsevier Applied Science Publishers.

Corley, W.G. and Jirsa, J.O. 1970, Equivalent Frame Analysis for Slab Design, *ACI Journal, Proceedings*, Vol 11, pp 875-884.

Comite Euro-International du Beton Federation Internationale de la Precontrainte 1993, *CEB-FIP Model Code 1990*, Thomas Telford Ltd, London.

Crisfield, M.A. 2001, *Non-Linear Finite Element Analysis of Solids and Structures: Volume 1*, John Wiley & Sons.

Cullen, M.R. Zill, D.G. 1992, *Advanced Engineering Mathematics*, PWS Publishing Company, Boston.

Favre, R. Beeby, A.W. Falkner, H. Koprna, M. and Schiessl, P. 1985, Cracking and Deformations, *CEB Manual.*

Gallagher, R.H. 1975, *Finite Element Analysis Fundementals*, Prentice-Hall.

Ghali, A. Favre, R. 1986, *Concrete Structures: Stresses and Deformations*, Chapman and Hall, London.

Ghali, A. Neville, A.M. 1997, *Structural Analysis: A Unified Classical and Matrix Approach*, E & FN Spon, London.

Gilbert, R.I 1988, *Time Effect in Concrete Structures*, Elsevier Science Publishers B.V., Netherlands.

Hinton, E. Owen, D.R.J. 1983, *Finite Element Programming*, Fourth Edition Academic Press Inc., London.

Hu, H.T. Schnobrich, W.C. 1991, Nonlinear Modelling of Reinforced Concrete Structures, *Computers and Structures*, Vol 38, No 5/6, pp 637-651.

Hughes, T.J.R 1987, *The Finite Element Method*, Prentice-Hall Inc., New Jersey.

Hughes, T.J.R Tezduyar, T.E. 1981, Finite Elements Based on Mindlin Plate Theory with Particular Reference to the Four-Node Bilinear Isoparametric Element, *Journal of Applied Mechanics*, pp 587-596.

Jofriet, J.C. McNeice, G.M. 1971, Finite Element Analysis of Reinforced Concrete Slabs, *Journal of the Structural Division, ASCE,* Vol 97, No 3, pp 785-806.

Kong, F.K. Evans, R.H. 1987, *Reinforced and Prestressed Concrete*, Chapman and Hall, London.

Liu, Y.J. Riggs, H.R., College of Engineering, University of Hawaii 2002, *Development of the min-n Family of Triangular Anisoparametric Mindlin Plate Elements*, [online] available: http://www.eng.hawaii.edu/~riggs/MIN-N-UHM-CE-02-02.pdf, [May 2004].

Miller, A.L. 1958, Warping of Reinforced Concrete Due to Shrinkage, *ACI Journal, Proceedings*, Vol 54, pp 939-950.

Mindlin, R.D. 1951, Influence of Rotary Inertia and Shear on Flexural Motions of Isotropic Elastic Plates, *Journal of Applied Mechanics*, 18, pp 31-38.

Neville, A.M. Dilger, W. 1970, *Creep of Concrete: Plain, Reinforced and Prestressed*, North-Holland Publishing Company, Amsterdam.

Park, .R Gamble, W.L. 2000, *Reinforced Concrete Slabs*, John Wiley & Sons.

Polak, M.A. Vecchio 1994, Reinforced Concrete Shell Elements Subjected to Bending and Membrane Loads, *ACI Structural Journal*, Vol 91, No 2, pp 261-268.

Polak, M.A. 1996, Effective Stiffness Model for Reinforced Concrete Slabs, *Journal of Structural Engineering*, Vol 122, No 9.

Tesler A, Hughes T.J.R. 1983, An Improved Treatment of Transverse Shear in Mindlin Type Four Node Quadrilateral Elements, *Computational Methods in Applied Mechanics and Engineering*, Vol 39, pp 311-335.

Ugural, A.C. 1999, *Stresses in Plates and Shells*, McGraw-Hill International Editions, Boston.

Vanderbilt, M.D. Sozen, M.A. Siess, C.P. 1965, Deflections of multiple-panel reinforced concrete floor slabs, *Journal of the Structural Division, ASCE,* Vol 91, No ST4, pp 77-101.

Vollum, R.L., Hossain, T.T. 1998, Assessment of Slab Deflections in the in-situ Concrete Frame Building, *Proceedings of the 3rd Cardington Conference.*

Weaver, W. Johnston, P.R. 1984, *Finite Elements for Structural Analysis*, Prentice-Hall.

Wood, R.H. 1968, The Reinforcement of Slabs in accordance with a Predetermined Field of Moments, *Concrete*, Vol 1, pp 69-76.

Zienkiewicz, O.C., Taylor, R.L. Too, J.M. 1971, Reduced Integration Technique in the General Analysis of Plates and Shells, *International Journal of Numerical Methods in Engineering*, Vol 43, pp 275-290.

7 Appendix

7.1 Algorithms

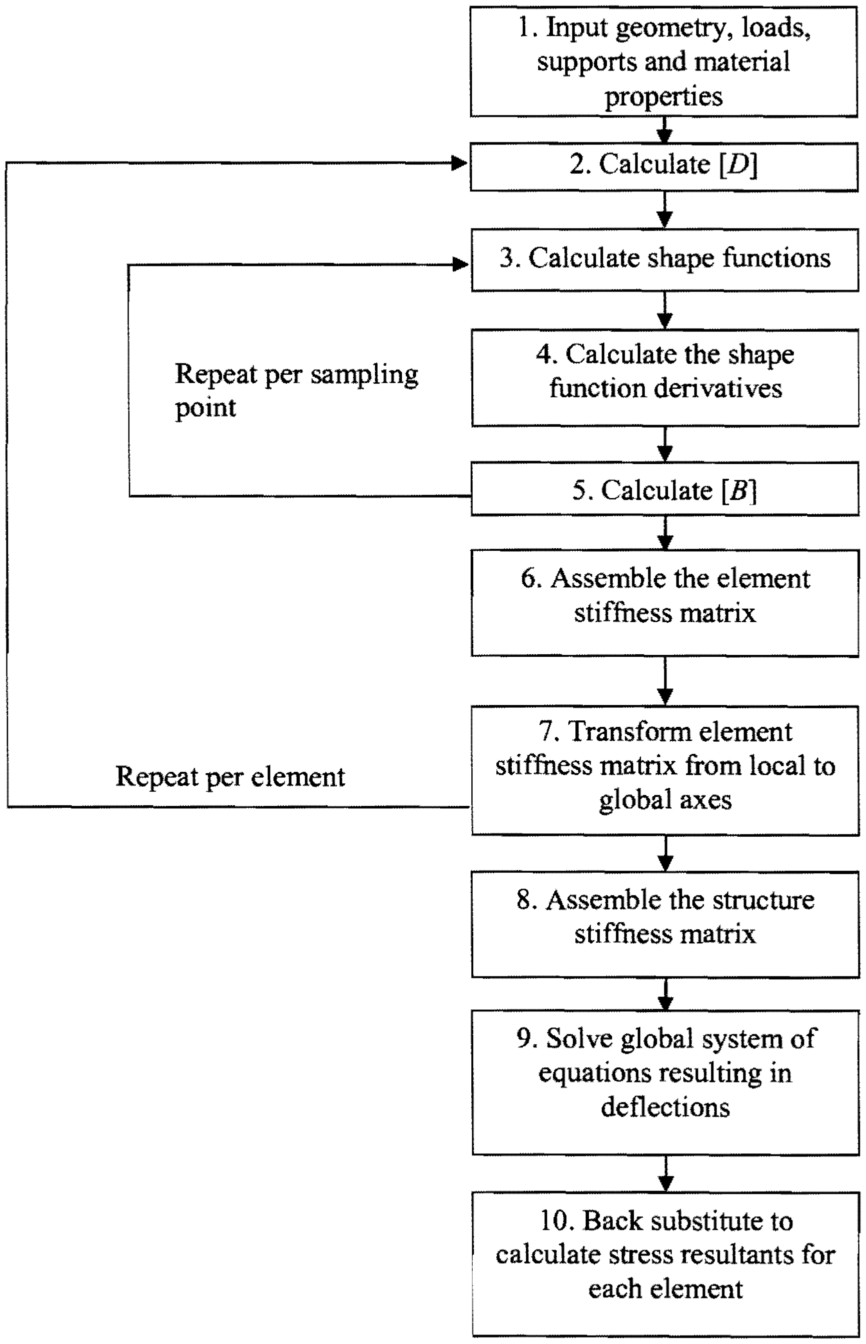
1. Input geometry, loads, supports and material properties
2. Calculate [D]
3. Calculate shape functions
4. Calculate the shape function derivatives
5. Calculate [B]
Repeat per sampling point
6. Assemble the element stiffness matrix
7. Transform element stiffness matrix from local to global axes
Repeat per element
8. Assemble the structure stiffness matrix
9. Solve global system of equations resulting in deflections
10. Back substitute to calculate stress resultants for each element

Figure 7-1: Linear finite element analysis algorithm

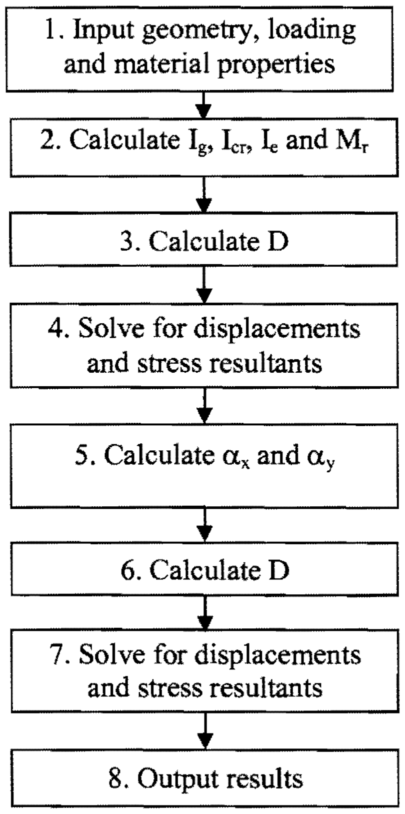
1. Input geometry, loading and material properties
2. Calculate I_g, I_{cr}, I_e and M_r
3. Calculate D
4. Solve for displacements and stress resultants
5. Calculate α_x and α_y
6. Calculate D
7. Solve for displacements and stress resultants
8. Output results

Figure 7-2: Polak's tension stiffening algorithm

Note: The calculation of [*D*] follows the procedure described in section 2.2.2.

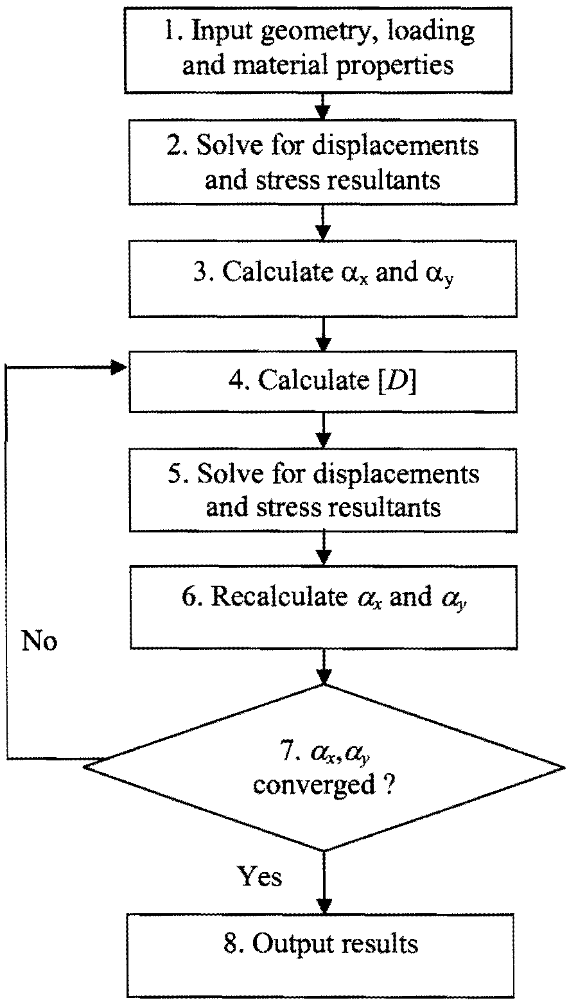
1. Input geometry, loading and material properties
2. Solve for displacements and stress resultants
3. Calculate α_x and α_y
4. Calculate [D]
5. Solve for displacements and stress resultants
6. Recalculate α_x and α_y
No
7. α_x, α_y converged ?
Yes
8. Output results

Figure 7-3: Modification of Polak's algorithm

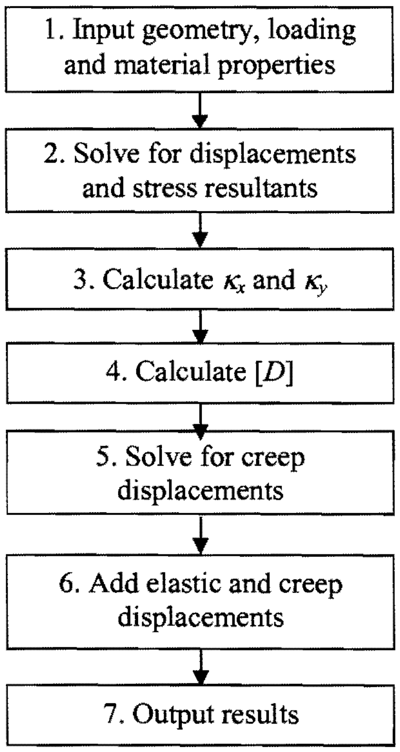
1. Input geometry, loading and material properties
2. Solve for displacements and stress resultants
3. Calculate κ_x and κ_y
4. Calculate $[D]$
5. Solve for creep displacements
6. Add elastic and creep displacements
7. Output results

Figure 7-4: Creep analysis algorithm

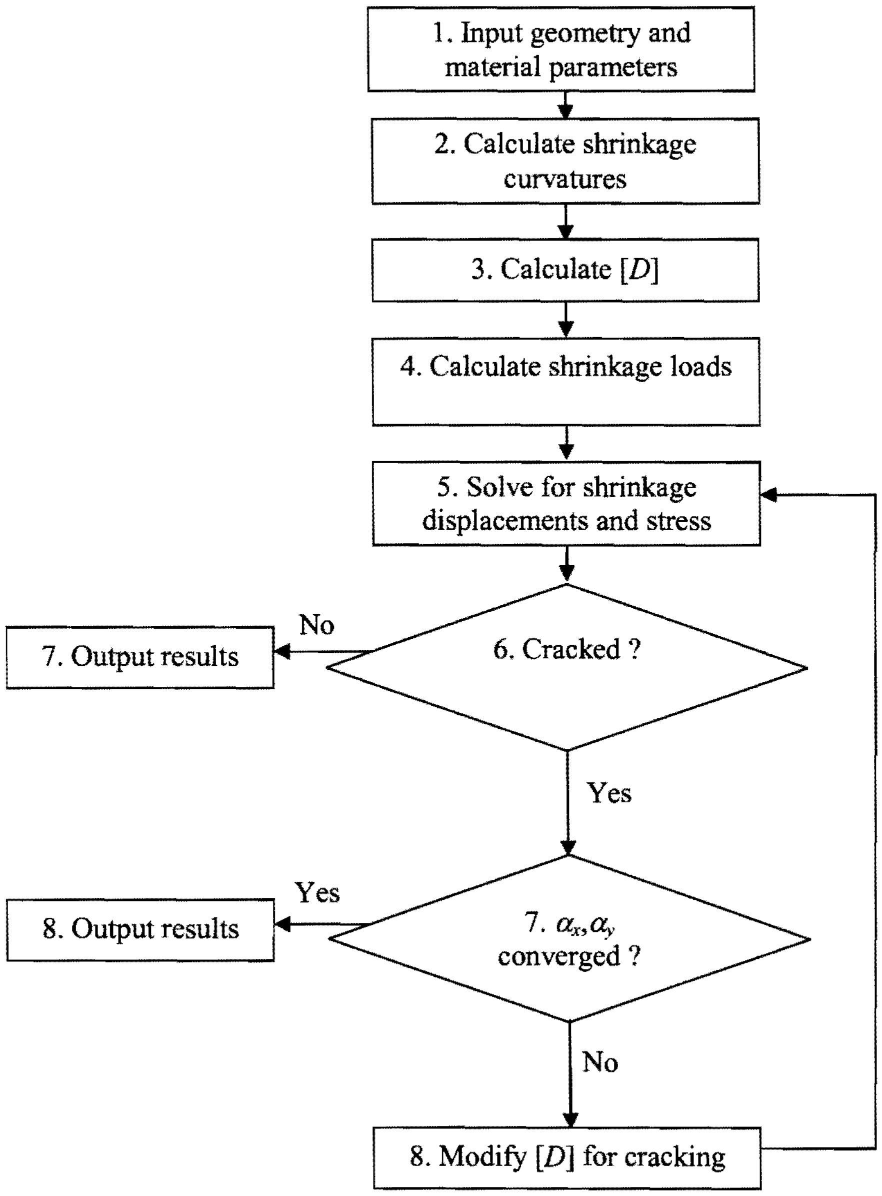
1. Input geometry and material parameters
2. Calculate shrinkage curvatures
3. Calculate [D]
4. Calculate shrinkage loads
5. Solve for shrinkage displacements and stress
6. Cracked ?
No
7. Output results
Yes
7. α_x, α_y converged ?
Yes
8. Output results
No
8. Modify [D] for cracking

Figure 7-5: Shrinkage analysis algorithm

7.2 Unit Mindlin code listing

```
unit Mindlin;

interface

type

  TElement = class
    public
      //Public member variables
      Number       : Integer;                        //Element index in global array
      Es           : Double;                         //Young's modulus of reinforcement
      E            : Double;                         //Young's modulus of concrete
      Pois         : Double;                         //Poisson's ratio
      h            : Double;                         //Element depth
      G            : Double;                         //Shear modulus of concrete
      NumIntOrder  : Integer;                        //Order of numerical integration
      phi          : double;                         //Creep factor

      NodesCoord   : array[1..3,1..8] of Double;     //Nodal point coordinates
      NodesNum     : array[1..8] of Integer;         //Global node numbers

      ElStiffp     : array[1..24,1..24] of Double;   //Element stiffness matrix - plate
      ElStiffm     : array[1..16,1..16] of Double;   //Element stiffness matrix - membrane
      ElStiff      : array[1..40,1..40] of Double;   //Superposed Element stiffness matrix

      SRes         : array[1..4,1..8] of Double;     //Stress resultants
      SResM        : array[1..8,1..8] of Double;     //Smoothed stress resultants
      Def          : array[1..5,1..8] of Double;     //Nodal deflections

      GaussDBp     : array[1..5,1..24,1..4] of Double; //DxB per sampling point - plate
      GaussDBm     : array[1..3,1..16,1..4] of Double; //DxB per sampling point – membrane

      ElLoad       : array[1..24] of Double;         //Element load vector
      UDL          : Double;                         //UDL on element

      Mxavg, Myavg          : Double;                //Average moments for crack analysis
      alphax, alphay        : Double;                //Crack modification factors
      ksix, ksiy            : Double;                //Bilinear crack modification factors
      alphaOldx, alphaOldy : Double;                 //Previous crack modification factors

      G1, G2, G3          : Double;                  //Orthotropic shear moduli
      Ex, Ey              : Double;                  //Orthotropic Young's moduli
      Poisx, Poisy        : Double;                  //Orthotropic Poisson's ratios

      Astx, Asty          : Double;                  //Area of tension steel mm²/m
      dtx, dty            : Double;                  //Effective depth of tension steel
      rocx, rocy          : Double;                  //Steel percentages, compression
      rox, roy            : Double;                  //Steel percentages, tension

      Igx, Igy            : Double;                  //Gross moments of inertia
      Iex, Iey            : Double;                  //Effective moments of inertia
      IeOldx, IeOldy      : Double;                  //Previous effective moments of inertia
      Icx, Icy            : Double;                  //Cracked moments of inertia
      Mcrx, Mcry          : Double;                  //Cracking moments
      Crackedx, Crackedy  : Boolean;                 //True/False crack flags

      Cx, Cy, Cxy         : Double;                  //Curvature variables
      xg, yg              : Double;                  //NA depth, gross, transformed
      xc, yc              : Double;                  //NA depth, cracked, transformed

      kappaX, kappaY      : Double;                  //Creep modification factors
      ecs                 : Double;                  //Free shrinkage strain
      MxShrink, MyShrink : double;                   //Shrinkage forces
    //Public procedures and functions
    Constructor Create(AOwner:TObject);   //Object creation - override
    Destructor Destroy; override;         //Object destruction - override

    Procedure CalcBp;                     //Calculates the plate strain matrix
    Procedure CalcBm;                     //Calculates the membrane strain matrix

    Procedure CalcDp;                     //Calculates the plate elastic matrix
    Procedure CalcDm;                     //Calculates the membrane elastic matrix

    Procedure CalcDBp;                    //Calculates product of B x D - plate
    Procedure CalcDBm;                    //Calculates product of B x D – membrane

    Procedure CalcJ( var JDet : Double;   //Calculates the jacobian matrix and its inverse
                     GNum     : Integer );
    Procedure CalcShape(u, v : Double);   //Calculates shape funtions and derivatives

    Procedure CalcStiffp;                 //Calculates the plate element stiffness matrix
    Procedure CalcStiffm;                 //Calculates the membrane element stiffness matrix
    Procedure CalcStiff;                  //Calculates the total element stiffness matrix

    Procedure SetupNumInt;                       //Sets up numerical integration
    Procedure CalcUDLoad;                        //Reduce UDL to nodal loads
    Procedure CalcShrinkLoad( CurvX : Double,    //Calculate shrinkage loads
                              CurvY : Double,
                              CurvXY: Double,
                              ShearX: Double,
                              ShearY: Double
    Procedure CalcAvgs;                          //Calculate average moments
    Procedure CalcModFactors(CrackType:integer); //Calculate crack modification factors
    Procedure CalcCurvatures;                    //Calculate curvatures from deflections
    Procedure CalcCreepFactors;                  //Calculate creep modification factors
    Procedure InitInertia;                       //Initialize moments of inertia vars
    Procedure CalcShrinkageCurvatures;           //Calculate shrinkage curvatures

  private
    //Private member variables
    Bp      : array[1..5,1..24] of Double; //Strain matrix - plate
    Bm      : array[1..3,1..16] of Double; //Strain matrix - membrane

    Dp      : array[1..5,1..5]  of Double; //Elasticity matrix - plate
    Dm      : array[1..3,1..3]  of Double; //Elasticity matrix - membrane

    DBp     : array[1..5,1..24] of Double; //Product of D x B - plate
    DBm     : array[1..3,1..16] of Double; //Product of D x B – membrane

    J       : array[1..2,1..2]  of Double; //Jacobian matrix
    JI      : array[1..2,1..2]  of Double; //Jacobian matrix inverse
    GP      : array[1..2,1..4]  of Double; //Sampling point coordinates

    SFunc   : array[1..8]       of Double; //Shape funtions per node;
    SFDeriv: array[1..3,1..8]   of Double; //Shape function derivatives per node
    CDeriv  : array[1..3,1..8]  of Double; //Cartesian shape function derivatives per node

    GaussPos : array[1..2]      of Double; //Sampling point position
    GaussWgt : array[1..2]      of Double; //Sampling point weighting factor

    calcForCreep : Boolean;                //True/False creep analysis flag
  published

end;
```

```
implementation

uses
  DTools, Math;

/////////////////////////////////////////////////////////
// Object creator - simply initializes a few variables //
/////////////////////////////////////////////////////////

Constructor TElement.Create(AOwner:TObject);
begin
  inherited Create;

  Crackedx := False;
  Crackedy := False;
  calcForCreep := False;
  InitInertia;
end;

//////////////////////////////////////////////////
// Object destructor - calls default destructor //
//////////////////////////////////////////////////

Destructor TElement.Destroy;
begin
  inherited Destroy;
end;

////////////////////////////////////////
// Calculates the plate strain matrix //
////////////////////////////////////////

Procedure TElement.CalcBp;
var
  iC,jC,kC : Integer;  //Loop counters

begin

  //Zero all matrix entries
  for iC := 1 to 5 do
    for jC := 1 to 3 do
      Bp[iC,jC] := 0;

  //Calculate the B matrix
  jC := 0;
  for iC := 1 to 8 do
  begin
    kC := jC + 1;
    Bp[4,kC] := CDeriv[1,iC];
    Bp[5,kC] := CDeriv[2,iC];
    kC := kC + 1;
    jC := kC + 1;
    Bp[1,kC] := -CDeriv[1,iC];
    Bp[3,kC] := -CDeriv[2,iC];
    Bp[4,kC] := -SFunc[iC];
    Bp[2,jC] := -CDeriv[2,iC];
    Bp[3,jC] := -CDeriv[1,iC];
    Bp[5,jC] := -SFunc[iC];
  end;

end;
```

```
///////////////////////////////////////////
// Calculates the membrane strain matrix //
///////////////////////////////////////////

Procedure TElement.CalcBm;
var
  iC,jC,kC : Integer;  //Loop counters

begin

  //Zero all matrix entries
  for iC := 1 to 3 do
    for jC := 1 to 2 do
      Bm[iC,jC] := 0;

  //Calculate the B matrix
  jC := 0;
  for iC := 1 to 8 do
  begin
    kC := jC + 1;
    jC := kC + 1;
    Bm[1,kC] := CDeriv[1,iC];
    Bm[1,jC] := 0;
    Bm[2,kC] := 0;
    Bm[2,jC] := CDeriv[2,iC];
    Bm[3,kC] := CDeriv[2,iC];
    Bm[3,jC] := CDeriv[1,iC];
  end;

end;

////////////////////////////////////////////
// Calculates the plate elasticity matrix //
////////////////////////////////////////////

Procedure TElement.CalcDp;
var
  FactorM, FactorV : Double;   //Temporary storage variables
  iC, jC           : Integer;  //Loop counters

begin

  //Zero all matrix entries
  for iC := 1 to 5 do
    for jC := 1 to 5 do
      Dp[iC,jC] := 0;

  if not (CalcForCreep) then
  begin
    //Do not use the creep modification factors
    if not (CrackedX or CrackedY) then
    begin
      //Calculate the gross D Matrix
      FactorM := E*power(h,3)/(12*(1-power(Pois,2)));
      FactorV := G*h/2.4;

      Dp[1,1] := FactorM;
      Dp[1,2] := Pois*FactorM;
      Dp[2,1] := Pois*FactorM;
      Dp[2,2] := FactorM;
      Dp[3,3] := G*pow(h,3)/12;
      Dp[4,4] := FactorV;
      Dp[5,5] := FactorV;
    end
    else
```

```
      begin
        //Calculate the cracked D Matrix
        Ex := E*alphax; Ey := E*alphay;
        Poisx := Pois*alphax; Poisy := Pois*alphay;
        G1 := G*alphax*alphay; G2 := G*alphax; G3 := G*alphay;


        Dp[1,1] := Ex*power(h,3)/(12*(1-Poisx*Poisy));
        Dp[1,2] := Poisx*Ey*power(h,3)/(12*(1-Poisx*Poisy));
        Dp[2,1] := Poisy*Ex*power(h,3)/(12*(1-Poisx*Poisy));
        Dp[2,2] := Ey*power(h,3)/(12*(1-Poisx*Poisy));
        Dp[3,3] := G1*power(h,3)/12;
        Dp[4,4] := G2*h;
        Dp[5,5] := G3*h;
      end;
    end
    else
    begin
      //Use the creep modification factors
      Ex := alphax*E/(phi*kappaX); Ey := alphay*E/(phi*kappaY);
      G1 := alphax*alphay*G/(phi*kappaX*kappaY);
      G2 := alphax*G/(phi*kappaX);
      G3 := alphay*G/(phi*kappaY);

      Dp[1,1] := Ex*power(h,3)/(12*(1-Pois*Pois));
      Dp[1,2] := Pois*Ey*power(h,3)/(12*(1-Pois*Pois));
      Dp[2,1] := Pois*Ex*power(h,3)/(12*(1-Pois*Pois));
      Dp[2,2] := Ey*power(h,3)/(12*(1-Pois*Pois));
      Dp[3,3] := G1*power(h,3)/12;
      Dp[4,4] := G2*h;
      Dp[5,5] := G3*h;
    end;

  end;

  //////////////////////////////////////////////////
  // Calculates the membrane elasticity matrix    //
  //                                              //
  // Note that no modifications have been made to //
  // account for cracking, creep and shrinkage    //
  //////////////////////////////////////////////////

  Procedure TElement.CalcDm;
  var
    Constant : Double;    //Temporary storgare variable
    iC, jC   : Integer;   //Loop counters
  begin

    //Zero all matrix entries
    for iC := 1 to 3 do
      for jC := 1 to 3 do
        Dm[iC,jC] := 0;

    Constant := E/(1-pow(Pois,2));
    Dm[1,1] := Constant;
    Dm[1,2] := Pois*Constant;
    Dm[2,1] := Pois*Constant;
    Dm[2,2] := Constant;
    Dm[3,3] := (1-Pois)/2*Constant;

  end;
```

```
/////////////////////////////////////////////////////////
// Calculates the product of B and D for re-use (plate) //
/////////////////////////////////////////////////////////

Procedure TElement.CalcDBp;
var
  iC,jC,kC : Integer;  //Loop counter

begin

  //Calculate B x D
  for iC := 1 to 5 do
  begin
    for jC := 1 to 24 do
    begin
      DBp[iC,jC] := 0;
      for kC := 1 to 5 do
        DBp[iC,jC] := DBp[iC,jC] + Dp[iC,kC]*Bp[kC,jC];
    end;
  end;

end;

////////////////////////////////////////////////////////////
// Calculates the product of B and D for re-use (membrane) //
////////////////////////////////////////////////////////////

Procedure TElement.CalcDBm;
var
  iC,jC,kC : Integer;  //Loop counters
begin

  //Calculate B x D
  for iC := 1 to 3 do
  begin
    for jC := 1 to 16 do
    begin
      DBm[iC,jC] := 0;
      for kC := 1 to 3 do
        DBm[iC,jC] := DBm[iC,jC] + Dm[iC,kC]*Bm[kC,jC];
    end;
  end;

end;

///////////////////////////////////////////////////////
// Calculates the the Jacobian matrix and its inverse //
///////////////////////////////////////////////////////

Procedure TElement.CalcJ(var JDet : Double; GNum : Integer);
var
  iC,jC,kC : Integer;  //Loop counters

begin

  //Zero all matrix entries
  for iC := 1 to 2 do
    for jC := 1 to 2 do
    begin
      JI[iC,jC] := 0;
    end;

  //Calculate Gaussian point coordinates
  for iC := 1 to 2 do
  begin
    GP[iC,GNum] := 0;
    for jC := 1 to 8 do
    begin
      GP[iC,GNum] := GP[iC,GNum] + NodesCoord[iC,jC]*SFunc[jC];
      end;
    end;

    //Calculate the Jacobian matrix
    for iC := 1 to 2 do
    begin
      for jC := 1 to 2 do
      begin
        J[iC,jC] :=0;
        for kC := 1 to 8 do
        begin
          J[iC,jC] := J[iC,jC] + SFDeriv[iC,kC]*NodesCoord[jC,kC];
        end;
      end;
    end;

    //Calculate determinant and inverse of the Jacobian
    JDet := J[1,1]*J[2,2]-J[1,2]*J[2,1];
    JI[1,1] := J[2,2]/JDet;
    JI[2,2] := J[1,1]/JDet;
    JI[1,2] := -J[1,2]/JDet;
    JI[2,1] := -J[2,1]/JDet;

    //Calculate the cartesian derivatives
    for iC := 1 to 2 do
    begin
      for jC := 1 to 8 do
      begin
        CDeriv[iC,jC] := 0;
        for kC := 1 to 2 do
        begin
          CDeriv[iC,jC] := CDeriv[iC,jC] + JI[iC,kC]*SFDeriv[kC,jC];
        end;
      end;
    end;

  end;

  //////////////////////////////////////////////////////////////////////
  // Calculates the shape functions and their cartesian derivatives //
  //////////////////////////////////////////////////////////////////////

  Procedure TElement.CalcShape( u,v : Double );
  begin
    //Shape functions
    SFunc[1] := -1/4 * (1-u)*(1-v)*(1+u+v);
    SFunc[2] :=  1/2 * (1-u*u)*(1-v);
    SFunc[3] :=  1/4 * (1+u)*(1-v)*(u-v-1);
    SFunc[4] :=  1/2 * (1+u)*(1-v*v);
    SFunc[5] :=  1/4 * (1+u)*(1+v)*(u+v-1);
    SFunc[6] :=  1/2 * (1-u*u)*(1+v);
    SFunc[7] :=  1/4 * (1-u)*(1+v)*(-u+v-1);
    SFunc[8] :=  1/2 * (1-u)*(1-v*v);

    //Shape function derivatives
    SFDeriv[1,1] :=  1/4 * (v+2*u-2*u*v-v*v);
    SFDeriv[1,2] :=  -u+u*v;
    SFDeriv[1,3] :=  1/4 * (-v+2*u-2*u*v+v*v);
    SFDeriv[1,4] :=  1/2 * (1-v*v);
    SFDeriv[1,5] :=  1/4 * (v+2*u+2*u*v+v*v);
    SFDeriv[1,6] :=  -u-u*v;
    SFDeriv[1,7] :=  1/4 * (-v+2*u+2*u*v-v*v);
    SFDeriv[1,8] :=  1/2 * (-1+v*v);
    SFDeriv[2,1] :=  1/4 * (u+2*v-u*u-2*u*v);
    SFDeriv[2,2] :=  1/2 * (-1+u*u);
    SFDeriv[2,3] :=  1/4 * (-u+2*v-u*u+2*u*v);
    SFDeriv[2,4] :=  -v-u*v;
    SFDeriv[2,5] :=  1/4 * (u+2*v+u*u+2*u*v);
    SFDeriv[2,6] :=  1/2 * (1-u*u);
    SFDeriv[2,7] :=  1/4 * (-u+2*v+u*u-2*u*v);
    SFDeriv[2,8] :=  -v+u*v;
```

```
end;


/////////////////////////////////////////////////
// Sets up the numerical integration constants //
/////////////////////////////////////////////////

Procedure TElement.SetupNumInt;
begin

  case NumIntOrder of
    2 : begin
          GaussPos[1] := -0.577350269189626;
          GaussWgt[1] := 1;
          GaussPos[2] := 0.577350269189626;
          GaussWgt[2] := 1;
        end;

    3 : begin
          GaussPos[1] := -0.774596669241483;
          GaussPos[2] := 0;
          GaussWgt[1] := 0.555555555555556;
          GaussWgt[2] := 0.888888888888889;
          GaussPos[3] := 0.774596669241483;
          GaussPos[4] := 0;
          GaussWgt[4] := 0.555555555555556;
          GaussWgt[4] := 0.888888888888889;
        end;
  end;

end;

///////////////////////////////////////////////////
// Calculates the plate element stiffness matrix //
///////////////////////////////////////////////////

Procedure TElement.CalcStiffp;
var
  iCount, jCount, kCount : Integer;   //Loop counters
  lCount, mCount, nCount : Integer;   //Loop counters
  uPoint, vPoint         : Double;    //Sampling point indices
  Area, JDet             : Double;    //Differential area and Jacobian determinant

begin

  //Zero all matrix entries
  for iCount := 1 to 24 do
    for jCount := 1 to 24 do
      ElStiffp[iCount,jCount] := 0;

  //Calculate the D matrix
  SetupNumInt;
  CalcDp;
  kCount := 0;

  //Numerical integration
  for iCount := 1 to 2 do
  begin
    uPoint := GaussPos[iCount];
    for jCount := 1 to 2 do
    begin
      vPoint := GaussPos[jCount];
      inc(kCount);

      CalcShape(uPoint,vPoint);
      CalcJ(JDet,kCount);
      Area := JDet*GaussWgt[iCount]*GaussWgt[jCount];

      CalcBp;
      CalcDBp;
      for lCount := 1 to 24 do
      begin
        for mCount := lCount to 24 do
        begin
          for nCount := 1 to 5 do
          begin
            ElStiffp[lCount,mCount] := ElStiffp[lCount,mCount]
                                       + Bp[nCount,lCount] * DBp[nCount,mCount] * Area;
          end;
        end;
      end;

      for lCount := 1 to 5 do
      begin
        for mCount := 1 to 24 do
        begin
          GaussDBp[lCount,mCount,kCount] := DBp[lCount,mCount];
        end;
      end;

    end;
  end;

  for lCount := 1 to 24 do
  begin
    for mCount := 1 to 24 do
    begin
      ElStiffp[mCount,lCount] := ElStiffp[lCount,mCount];
    end;
  end;

end;

/////////////////////////////////////////////////////
// Calculates the membrane element stiffness matrix //
/////////////////////////////////////////////////////

Procedure TElement.CalcStiffm;
var
  iCount, jCount, kCount : Integer; //Loop counters
  lCount, mCount, nCount : Integer; //Loop counters
  uPoint, vPoint         : Double;  //Sampling point indices
  Area, JDet             : Double;  //Differential area and Jacobian determinant

begin

  //Zero all matrix entries
  for iCount := 1 to 16 do
    for jCount := 1 to 16 do
      ElStiffm[iCount,jCount] := 0;

  //Calculate the D matrix
  SetupNumInt;
  CalcDm;
  kCount := 0;

  //Numerical integration
  for iCount := 1 to 2 do
  begin
    uPoint := GaussPos[iCount];
    for jCount := 1 to 2 do
    begin
      vPoint := GaussPos[jCount];
      inc(kCount);

      CalcShape(uPoint,vPoint);
      CalcJ(JDet,kCount);
      Area := JDet*GaussWgt[iCount]*GaussWgt[jCount]*h;

      CalcBm;
      CalcDBm;

      for lCount := 1 to 16 do
      begin
        for mCount := lCount to 16 do
        begin
          for nCount := 1 to 3 do
          begin
            ElStiffm[lCount,mCount] := ElStiffm[lCount,mCount]
                                    + Bm[nCount,lCount] * DBm[nCount,mCount] * Area;
          end;
        end;
      end;

      for lCount := 1 to 3 do
      begin
        for mCount := 1 to 16 do
        begin
          GaussDBm[lCount,mCount,kCount] := DBm[lCount,mCount];
        end;
      end;

    end;
  end;

  for lCount := 1 to 16 do
  begin
    for mCount := 1 to 16 do
    begin
      ElStiffm[mCount,lCount] := ElStiffm[lCount,mCount];
    end;
  end;

end;

////////////////////////////////////////////////
// Assemble the total element stiffness matrix //
////////////////////////////////////////////////

Procedure TElement.CalcStiff;
var
  i, j, k, l, m : Integer;  //Loop counters

begin
  CalcStiffm;
  CalcStiffp;
  for i := 1 to 8 do
  begin
    for j := 1 to 8 do
    begin
      for k := 1 to 2 do
      begin
        for l := 1 to 2 do
        begin
          ElStiff[5*i-(k+2),5*j-(l+2)] := ElStiffm[i*k,j*l];
        end;
      end;
      for k := 1 to 3 do
      begin
        for l := 1 to 3 do
        begin
          ElStiff[5*i-(k-1),5*j-(l-1)] := ElStiffp[i*k,j*l];
        end;
      end;
    end;
  end;
```

```
end;




/////////////////////////////////////////
// Reduce UDL to equivalent nodal loads //
/////////////////////////////////////////

Procedure Telement.CalcUDLoad;
var
  iCount, jCount, kCount : Integer;  //Loop counters
  lCount, pos            : Integer;  //Loop counters
  u, v                   : Double;   //Gauss point coordinates
  DArea, JDet            : Double;   //Differential area and Jacobian determinant

begin
  //Zero all matrix entries
  for iCount := 1 to 24 do
    ElLoad[iCount] := 0;

  //Numerical integration
  kCount := 0;
  for iCount := 1 to 2 do
  begin
    u := GaussPos[iCount];
    for jCount := 1 to 2 do
    begin
      v := GaussPos[jCount];
      inc(kCount);

      CalcShape(u,v);
      CalcJ(JDet,kCount);
      DArea := JDet*GaussWgt[iCount]*GaussWgt[jCount];

      for lCount := 1 to 8 do
      begin
        pos := (lCount-1)*3+1;
        ElLoad[pos] := ElLoad[pos] + SFunc[lCount]*UDL*DArea;
      end;

    end;
  end;
end;

///////////////////////////////////////////////
// Calculate shrinkage loads given curvatures //
///////////////////////////////////////////////


Procedure Telement.CalcShrinkLoad(CurvX, CurvY, CurvXY, ShearX, ShearY : double);
var
  i, j, k, l, m, posi : Integer;                      //Loop counters
  u, v                : Double;                       //Gauss point coordinates
  JDet                : Double;                       //Jacobian determinant
  DArea               : Double;                       //Differential area
  iStrain             : array[1..5] of double;        //Initial strain matrix
  BpT                 : array[1..24,1..5] of double;  //Transponent of [B]
  BpTD                : array[1..24,1..5] of double;  //Product [B^T][D]

  //Calculate transponent of [B]
  Procedure CalcBpTransponent;
  var
    i, j : integer;  //Loop counters

  begin
    for i := 1 to 5 do
    begin
      for j := 1 to 24 do
      begin
        BpT[j,i] := Bp[i,j];
      end;
    end;
  end;



  //Calculate the product of [B^T] and [D]
  Procedure CalcBpTD;
  var
    i, j, k : integer;  //Loop counters

  begin
    for i := 1 to 24 do
      for j := 1 to 5 do
        BpTD[i,j] := 0;
    for i := 1 to 24 do
      for j := 1 to 5 do
        for k := 1 to 5 do
          BpTD[i,j] := BpTD[i,j] + BpT[i,k]*Dp[k,j];
  end;

begin
  //Assign the initial strain matrix and calculate D
  iStrain[1] := CurvX;
  iStrain[2] := CurvY;
  iStrain[3] := CurvXY;
  iStrain[4] := ShearX;
  iStrain[5] := ShearY;
  CalcDp;



  //Zero all matrix entries
  for i := 1 to 24 do
    ElLoad[i] := 0;

  //Numerical integration
  k := 0;
  for i := 1 to 2 do
  begin
    u := GaussPos[i];
    for j := 1 to 2 do
    begin
      v := GaussPos[j];
      inc(k);

      CalcShape(u,v);
      CalcJ(JDet,k);
      DArea := JDet*GaussWgt[i]*GaussWgt[j];
      CalcBp;
      CalcBpTransponent;
      CalcBpTD;

      for l := 1 to 24 do
        for m := 1 to 5 do
            ElLoad[l] := ElLoad[l]+BpTD[l,m]*iStrain[m]*DArea;

    end;
  end;
end;

////////////////////////////////////////////////////////
// Calculate the average moments for a crack analysis //
////////////////////////////////////////////////////////

Procedure TElement.CalcAvgs;
var
  iCount : Integer;
begin
```

```
  MxAvg := 0;
  MyAvg := 0;

  for iCount := 1 to 4 do
  begin
    MxAvg := MxAvg + abs(SRes[iCount,4]) + abs(SRes[iCount,6]);
    MyAvg := MyAvg + abs(SRes[iCount,5]) + abs(SRes[iCount,6]);
  end;

  MxAvg := MxAvg/4;
  MyAvg := MyAvg/4;

end;

///////////////////////////////////////////////////////////
// Calculates the curvatures due to restrained shrinkage //
///////////////////////////////////////////////////////////


Procedure TElement.CalcShrinkageCurvatures;
var
  xs, ys           : Double; //Eccentricity of steel with respect to the cracked NA
  x, y             : Double; //NA of the transformed section
  Ec               : Double; //Effective concrete modulus
  n                : Double; //Modular ratio
  CurveX, CurveY : Double; //Shrinkage curvatures
  Ix, Iy           : Double; //Moments of inertia of the transformed section
Begin

  Ec := E/(1+phi);
  n  := Es/Ec;

  x  := (0.5*pow(h,2) + n*Astx*dtx)/(h+n*Astx);
  Ix := pow(h,3)/12+h*pow(h/2-x,2)+n*Astx*pow(dtx-x,2);
  xs := dtx - x;
  CurveX := ecs*n*Astx*xs/Ix;

  y  := (0.5*pow(h,2) + n*Asty*dty)/(h+n*Asty);
  Iy := pow(h,3)/12+h*pow(h/2-y,2)+n*Astx*pow(dty-y,2);
  ys := dty - y;
  CurveY := ecs*n*Asty*ys/Iy;

  CalcShrinkLoad(CurveX,CurveY,0,0,0);

end;

///////////////////////////////////////////////////////
// Calculates curvatures due given moment resultants //
///////////////////////////////////////////////////////

Procedure TElement.CalcCurvatures;
var
  Mx, My, Mxy : Double;  //Moments
  a, b, c, d  : Double;  //Temporary storage variables
  i           : Integer; //Loop counter

begin

  Mx := 0;
  My := 0;
  Mxy := 0;

  a := Dp[1,1]; b := Dp[1,2];
  c := Dp[2,1]; d := Dp[2,2];

  for i := 1 to 4 do
  begin
    Mx  := Mx  + SRes[i,4];
    My  := My  + SRes[i,5];
    Mxy := Mxy + SRes[i,5];
  end;
  Mx  := Mx/4;
  My  := Mxy/4;
  Mxy := My/4;

  Cxy := Mxy/Dp[3,3];
  Cy  := (My-c/a*Mx)/(d-c*b/a);
  Cx  := (Mx-b*Cy)/a;

end;
//////////////////////////////////
// Initializes several variables //
//////////////////////////////////

Procedure TElement.InitInertia;
begin
  Igx := 1/12*pow(h,3);
  Igy := 1/12*pow(h,3);
  Iex := Igx;
  Iey := Igy;
  IeOldx := Igx;
  IeOldy := Igy;
  alphax := 1;
  alphay := 1;
end;

/////////////////////////////
// Calculates creep factors //
/////////////////////////////

Procedure TElement.CalcCreepFactors;
var
  Ibarx, Ibary, Ix, Iy, x, y, AeffX, AeffY, ycx, ycy, delyx, delyy : double;
  xga, yga, xca, yca, Icxa, Icya : double;
  na, n, Ecreep      : double;
  Igbarx, Icbarx, Igbary, Icbary : double;
  xcprime, ycprime, xcaprime, ycaprime : double;
begin

  ECreep := E/(1+phi);
  na := Es/Ecreep;
  n := Es/E;

  xg  := (t*t/2 + n*Astx*dtx  + n*Ascx*(t-dcx))/ (t + n*Astx  + n*Ascx);
  yg  := (t*t/2 + n*Asty*dty  + n*Ascy*(t-dcy))/ (t + n*Asty  + n*Ascy);
  xga := (t*t/2 + na*Astx*dtx + na*Ascx*(t-dcx))/(t + na*Astx + na*Ascx);
  yga := (t*t/2 + na*Asty*dty + na*Ascy*(t-dcy))/(t + na*Asty + na*Ascy);

  xcprime := dtx*( -n*(rox+rocx) + sqrt(pow(n*(rox+rocx),2) + 2*n*(rox+(t-dcx)/dtx*rocx)) );
  ycprime := dty*( -n*(roy+rocy) + sqrt(pow(n*(roy+rocy),2) + 2*n*(roy+(t-dcy)/dty*rocy)) );

  xcaprime := dtx*( -na*(rox+rocx) + sqrt(pow(na*(rox+rocx),2)
              + 2*na*(rox+(t-dcx)/dtx*rocx)) );
  ycaprime := dty*( -na*(roy+rocy) + sqrt(pow(na*(roy+rocy),2)
              + 2*na*(roy+(t-dcy)/dty*rocy)) );

  xc  := xg*alphax; if xc<xcprime then xc:=xcprime else if xc>xg then xc:=xg;
  yc  := yg*alphay; if yc<ycprime then yc:=ycprime else if yc>yg then yc:=yg;
  xca := xga*alphax; if xca<xcaprime then xca:=xcaprime else if xca>xga then xca:=xga;
  yca := yga*alphay; if yca<ycaprime then yca:=ycaprime else if yca>yga then yca:=yga;

  IgBarx := pow(t,3)/12+t*pow(t/2-xga,2)+na*Astx*pow(dtx-xga,2)+na*Ascx*pow((t-dtx)-xga,2);
  IcBarx := 1/3*pow(xca,3) + na*rocx*pow(xca-(t-dcx),2)*dtx + na*rox*pow(dtx-xca,2)*dtx;
  IgBary := pow(t,3)/12+t*pow(t/2-yga,2)+na*Asty*pow(dty-yga,2)+na*Ascy*pow((t-dcy)-yga,2);
  IcBary := 1/3*pow(yca,3) + na*rocy*pow(yca-(t-dcy),2)*dty + na*roy*pow(dty-yca,2)*dty;

  if ((not CrackedX) or (alphax>9.99)) then
  begin
    Aeffx := t;
    Ix := pow(t,3)/12+t*pow(t/2-xga,2);
    ycx := Aeffx/2; ycx := ycx-xga;
    delyx := xga-xg;
    IBarx := IgBarx;
  end
  else
  begin
    Aeffx := xc;
    Ix := pow(xc,3)/12+xc*(xca-xc/2)*(xca-xc/2);

    ycx := Aeffx/2; ycx := ycx-xca;
    delyx := xca-xc;
    IBarx := IcBarX;
  end;

  if ((not CrackedY) or (alphaY>9.99)) then
  begin
    Aeffy := t;
    Iy := pow(t,3)/12+t*pow(t/2-yga,2);

    ycy := Aeffy/2; ycy := ycy-yga;
    delyy := yga-yg;
    IBary := IgBary;
  end
  else
  begin
    Aeffy := yc;
    Iy := pow(yc,3)/12+yc*(yca-y/2)*(yca-y/2);

    ycy := Aeffy/2; ycy := ycy-yca;
    delyy := yca-yc;
    IBary := IcBary;
  end;

  kappaX := (Ix + Aeffx*ycx*delyx)/IBarx;
  kappaY := (Iy + Aeffy*ycy*delyy)/IBary;

  CalcForCreep := True;
end;

//////////////////////////////
// Calculates crack factors //
//////////////////////////////

Procedure TElement.CalcModFactors(CrackType:integer);
var
  alpha      : Double;  //Modular ratio
  fr         : Double;  //Modulus of rupture (Should be an element property)
  yx, yy     : Double;  //NA positions of the gross concrete section
  dtxc, dtyc : Double;  //Embedment depth of compression reinforcement (Should be an element
                          property)

begin

  //Initialize
  Crackedx := False;
  Crackedy := False;
  IeOldx := Iex;
  IeOldy := Iey;
  alphaOldx := alphax;
  alphaOldy := alphay;
  alpha := Es/E;

  //Calculate NA's and moments of inertia
  xc := dtx*( -alpha*(rox+rocx) + sqrt(pow(alpha*(rox+rocx),2)
        + 2*alpha*(rox+(t-dcx)/dtx*rocx)) );
  Icx := 1/3*pow(xc,3)/12 + alpha*rocx*pow(xc-(t-dcx),2)*dtx + alpha*rox*pow(dtx-xc,2)*dtx;

  yc := dty*( -alpha*(roy+rocy) + sqrt(pow(alpha*(roy+rocy),2)
        + 2*alpha*(roy+(t-dcy)/dty*rocy)) );
  Icy := 1/3*pow(yc,3)/12 + alpha*rocy*pow(yc-(t-dcy),2)*dty + alpha*roy*pow(dty-yc,2)*dty;
  Igx := 1/12*pow(h,3);
  yx := h/2;

  Igy := 1/12*pow(h,3);
  yy := h/2;

  //Calculate cracking moments
  Mcrx := fr*Igx/yx;
  Mcry := fr*Igy/yy;

  //Bilinear method cracking factors
  if(MxAvg >= Mcrx) then ksix := 1-pow(Mcrx/MxAvg,2) else ksix := 0;
  if(MyAvg >= Mcry) then ksiy := 1-pow(Mcry/MyAvg,2) else ksiy := 0;

  if(MxAvg >= Mcrx) then
  begin
    if CrackType = 0 then
    begin
      Iex := pow(Mcrx/MxAvg,4)*Igx + (1 - pow(Mcrx/MxAvg,4))*Icx;
      Iex := IeOldx-(IeOldx - Iex)/2;
    end
    else
    begin
      Iex := (Igx*Icx) / ( (1-ksix)*Icx+ksix*Igx );
      Iex := IeOldx-(IeOldx - Iex)/2;
    end;
    if Iex > Igx then Iex := Igx
    else if Iex < Icx then Iex := Icx;
    alphax := Iex/Igx;
    CrackedX := True;
  end
  else
  begin
    Iex := Igx;
    IeOldx := Iex;
    alphax := 1;
    CrackedX := False;
  end;

  if(MyAvg >= Mcry) then
  begin
    if (CrackType = 0) then
    begin
      Iey := pow(Mcry/MyAvg,4)*Igy + (1 - pow(Mcry/MyAvg,4))*Icy;
      Iey := IeOldy-(IeOldy - Iey)/2;
    end
    else
    begin
      Iey := (Igy*Icy) / ( (1-ksiy)*Icy+ksiy*Igy );
      Iey := IeOldy-(IeOldy - Iey)/2;
    end;
    if Iey > Igy then Iey := Igy
    else if Iey < Icy then Iey := Icy;
    alphay := Iey/Igy;
    CrackedY := True;
  end
  else
  begin
    Iey := Igy;
    alphay := 1;
    CrackedY := False;
  end;

  if alphax>0.99 then crackedx := false;
  if alphay>0.99 then crackedy := false;

end;

end.
```

7.3 Elastic Analysis Procedure

```
procedure ElasticAnalysis;
var
  iCount: Integer;  //Loop counters

begin

  //Obtain FEA geometry, loading & material input
  readInput;

  //Initialize the model and global arrays
  Initialize;

  //Assemble the element and global stiffness matrices
  Assemble;

  //Assemble the load vector
  LoadVector;

  //Solve the linear system
  Solve;

  //Update global deflection array
  UpdateGlobalDVec;

  //Update each element's deflection array
  UpdateElementDVec;

  //Calculate and smooth stress resultants
  CalcResultants(true);

  UpdateOuput;

end;
```

7.4 Crack Analysis Procedure

```
procedure CrackAnalysis;
var
  iCount  : Integer;  //Loop counter
  El      : TElement; //Element object pointer
  Conv    : Double;   //Convergence variable
  limit   : Double;   //Convergence limit

  //Returns current convergence index
  function testConv:double;
  var
    i, j : integer;
    x,y : double;
  begin
    x := 0;
    y := 0;
    for i := 1 to num*num do
    begin
      x := x + abs(slab[i].alphax-slab[i].alphaOldx)/slab[i].alphaoldx;
      y := y + abs(slab[i].alphay-slab[i].alphaOldy)/slab[i].alphaoldy;
    end;
    x := x/(num*num);
    y := y/(num*num);
    result := max(x,y);
  end;

begin

  //Obtain FEA geometry, loading & material input
  readInput;

  //Initialize the model and global arrays
  Initialize;

  limit := 1e-3;  //Convergence limit

  //Initialize element moments of inertia
  for iCount := 1 to numElements do Slab[iCount].InitInertia;

  conv := 1;  //Convergence tester
  while ( abs(conv)>limit ) do
  begin
    //Record current deflection field
    StoreOrigDVec;

    //Assemble element and global stiffness matrices
    Assemble;

    //Assemble the load vector
    LoadVector;

    //Solve the linear system
    Solve;

    //Update global deflection array
    UpdateGlobalDVec;

    //Update elements' deflection array
    UpdateElementDVec;

    //Calculate stress resultants but do not smooth
    CalcResultants(false);

    //Calculate average moments and modification factors for each element
    for jCount := 1 to numElements do
    begin
      Slab[jCount].CalcAvgs;
      Slab[jCount].CalcModFactors(CrackType);
    end;
    //Update the convergence variable
    conv := testConv;

    if(conv < limit) then
      //If solution converged, smooth stress resultants
      CalcResultants(true);
    else
      //If solution not converged, restore original deflection field
      RegressGlobalDVec;

  end;

  UpdateOutput;
end;
```

7.5 Creep Analysis Procedure

```
Procedure CreepClickAnalysis;
var
  i  : Integer;  //Loop counter;

begin

  //Perform a crack analysis
  CrackAnalysis;

  //Initialize global arrays
  Initialize;

  //Calculate creep factors for each element
  for i := 1 to numElements do
  begin
    Slab[i].CalcCreepFactors;
  end;

  //Assemble element and global stiffness matrices
  Assemble;

  //Assemble the load vector
  LoadVector;

  //Solve the linear system
  Solve;

  //Update global deflection array
  UpdateGlobalDVec;

  //Update elements' deflection array
  UpdateElementDVec;

  //Calculate and smooth stress resultants
  CalcResultants(true);

  UpdateOutput;

end;
```

7.6 Shrinkage Analysis Procedure

```
Procedure ShrinkAnalysis;
var
  i : integer;

begin
  //Obtain FEA geometry, loading & material input
  readInput;

  //Initialize the model and global arrays
  Initialize;

  //Assemble element and global stiffness matrices
  Assemble;

  //Calculate shrinkage forces for each element
  for i := 1 to num*num do
  begin
    slab[i].ecs := ecs;
    slab[i].CalcShrinkageMoments;
  end;

  //Assemble the load vector
  LoadVector;

  //Solve the linear system
  Solve;

  //Update global deflection array
  UpdateGlobalDVec;

  //Update elements' deflection array
  UpdateElementDVec;

  //Calculate and smooth stress resultants
  CalcResultants(true);

  UpdateOutput;

end;
```

7.7 Stiffness Matrix Assembly Procedure

```
Procedure Assemble;
var
  iCount, jCount, kCount : Integer;  //Loop counters
  GNodeRow, GNodeCol     : Integer;  //Row, column indices
  DOF                    : Integer;  //Degrees of freedom variables
begin

  DOF := 5;

  for iCount := 1 to numElements do
  begin
    //Calculate element stiffness matrices
    Slab[iCount].CalcStiff;
    for jCount := 1 to 8 do
    begin

      GNodeRow := Slab[iCount].NodesNum[jCount];
      for kCount := 1 to 8 do
      begin
        //Place elemnt matrix in structure stiffness matrix
        GNodeCol := Slab[iCount].NodesNum[kCount];
        GStiff[5*GNodeRow-4,5*GNodeCol-4] := GStiff[5*GNodeRow-4,5*GNodeCol-4]
                                             + Slab[iCount].ElStiffm[2*jCount-1,2*kCount-1];
        GStiff[5*GNodeRow-4,5*GNodeCol-3] := GStiff[5*GNodeRow-4,5*GNodeCol-3]
                                             + Slab[iCount].ElStiffm[2*jCount-1,2*kCount-0];

        GStiff[5*GNodeRow-3,5*GNodeCol-4] := GStiff[5*GNodeRow-3,5*GNodeCol-4]
                                             + Slab[iCount].ElStiffm[2*jCount-0,2*kCount-1];
        GStiff[5*GNodeRow-3,5*GNodeCol-3] := GStiff[5*GNodeRow-3,5*GNodeCol-3]
                                             + Slab[iCount].ElStiffm[2*jCount-0,2*kCount-0];

        GStiff[3*GNodeRow-2,3*GNodeCol-2] := GStiff[3*GNodeRow-2,3*GNodeCol-2]
                                             + Slab[iCount].ElStiffp[3*jCount-2,3*kCount-2];
        GStiff[3*GNodeRow-2,3*GNodeCol-1] := GStiff[3*GNodeRow-2,3*GNodeCol-1]
                                             + Slab[iCount].ElStiffp[3*jCount-2,3*kCount-1];
        GStiff[3*GNodeRow-2,3*GNodeCol-0] := GStiff[3*GNodeRow-2,3*GNodeCol-0]
                                             + Slab[iCount].ElStiffp[3*jCount-2,3*kCount-0];

        GStiff[3*GNodeRow-1,3*GNodeCol-2] := GStiff[3*GNodeRow-1,3*GNodeCol-2]
                                             + Slab[iCount].ElStiffp[3*jCount-1,3*kCount-2];
        GStiff[3*GNodeRow-1,3*GNodeCol-1] := GStiff[3*GNodeRow-1,3*GNodeCol-1]
                                             + Slab[iCount].ElStiffp[3*jCount-1,3*kCount-1];
        GStiff[3*GNodeRow-1,3*GNodeCol-0] := GStiff[3*GNodeRow-1,3*GNodeCol-0]
                                             + Slab[iCount].ElStiffp[3*jCount-1,3*kCount-0];

        GStiff[3*GNodeRow-0,3*GNodeCol-2] := GStiff[3*GNodeRow-0,3*GNodeCol-2]
                                             + Slab[iCount].ElStiffp[3*jCount-0,3*kCount-2];
        GStiff[3*GNodeRow-0,3*GNodeCol-1] := GStiff[3*GNodeRow-0,3*GNodeCol-1]
                                             + Slab[iCount].ElStiffp[3*jCount-0,3*kCount-1];
        GStiff[3*GNodeRow-0,3*GNodeCol-0] := GStiff[3*GNodeRow-0,3*GNodeCol-0]
                                             + Slab[iCount].ElStiffp[3*jCount-0,3*kCount-0];
      end;
    end;
  end;

end;
```

7.8 Gauss Reduction Procedure

```
Procedure Solve;
var
  Pivot, Factor, Residual : Double;
  i, j, k, l              : Integer;  //Loop counters
  iRow, iCol              : Integer;  //Row & column indices
  NBack, NBac1, Neqns1    : Integer;
  DOF, SR                 : Integer;  //Degrees of freedom and no of stress resultants
  Diff : Double;
begin
  DOF := 5;
  SR := 8;

  //Gauss Reduction
  for i := 1 to numNodes*DOF do
  begin
    if (SVec[i]<>1) then //free DOF
    begin
      Pivot := GStiff[i,i];
      if(abs(pivot)>1e-10) then
      begin //valid pivot
        if (i<numNodes*DOF) then
        begin
          j := i+1;
          for iRow := j to numNodes*DOF do
          begin
            Factor := GStiff[iRow,i]/Pivot;
            if (Factor<>0) then
            begin
              for iCol := i to numNodes*DOF do
              begin
                GStiff[iRow,iCol] := GStiff[iRow,iCol] - Factor*GStiff[i,iCol];
              end;
              LVec[iRow] := LVec[iRow] - Factor*LVec[i];
            end;
          end;
        end;
      end
      else //invalid pivot
      begin
        infomsg('Non positive definite matrix');
        exit;
      end;
    end
    else //restrained DOF
    begin
      for iRow := i to numNodes*DOF do
      begin
        LVec[iRow] := LVec[iRow] - GStiff[iRow,i]*FVec[i];
        GStiff[iRow,i] := 0;
      end;
    end;
  end;

  //Back substitution process
  j := numNodes*DOF+1;
  for i := 1 to numNodes*DOF do
  begin
    k := j-i; //nback
    Pivot := GStiff[k,k];
    Residual := LVec[k];

    if (k<>numNodes*DOF) then
    begin
      l := k+1;
      for iCol := l to numNodes*DOF do
      begin
        Residual := Residual - GStiff[k,iCol]*DVec[iCol];
      end;
      end;

      If (SVec[k]=0) then
      begin
        DVec[k] := Residual/Pivot;
      end
      else
      begin
        DVec[k] := FVec[k];
        RVec[k] := -Residual;
      end;

    end;

  end;
```

7.9 Stress Resultant Calculation Procedure

```
Procedure CalcResultants(smooth:boolean);
var
  iCount, jCount, kCount, lCount : Integer;                    //Loop counters
  mCount, nCount, oCount, pCount : Integer;                    //Loop counters
  trans                          : array[1..4,1..4] of Double; //Extrapolation matrix
  temp                           : array[1..8] of Double;      //Holder matrix
  Sum                            : array[1..8] of Double;      //Holder matrix
  SR : Integer;
  NodesArray : array[1..4] of integer;

begin
  SR := 8;

  //Calculate stress resultants at sampling points - membrane
  for iCount := 1 to numElements do
  begin

    lCount := 0; //1 to 4
    for jCount := 1 to 2 do
    begin
      for kCount := 1 to 2 do
      begin
        inc(lCount);
        for mCount := 1 to 3 do
        begin
          nCount := 0;
          Slab[iCount].SRes[lCount,mCount] := 0;

          for oCount := 1 to 8 do
          begin
            for pCount := 1 to 2 do
            begin
              inc(nCount);
              Slab[iCount].SRes[lCount,mCount] := Slab[iCount].SRes[lCount,mCount] +
                (Slab[iCount].GaussDBm[mCount,nCount,lCount] *
                  Slab[iCount].Def[pCount,oCount]);
            end;
          end;
        end;
      end;
    end;

  end;

  //Calculate stress resultants at sampling points - plate
  for iCount := 1 to numElements do
  begin

    lCount := 0; //1 to 4
    for jCount := 1 to 2 do
    begin
      for kCount := 1 to 2 do
      begin
        inc(lCount);
        for mCount := 4 to 8 do
        begin
          nCount := 0;
          Slab[iCount].SRes[lCount,mCount] := 0;

          for oCount := 1 to 8 do
          begin
            for pCount := 3 to 5 do
            begin
              inc(nCount);
              Slab[iCount].SRes[lCount,mCount] := Slab[iCount].SRes[lCount,mCount] +
                (Slab[iCount].GaussDBp[mCount-3,nCount,lCount] *
                  Slab[iCount].Def[pCount,oCount]);
            end;
        end;
      end;
    end;
  end;
  Slab[iCount].CalcCurvatures;
end;

if not smooth then exit;

//Smooth by bi-linear extrapolation to the corner nodes
trans[1,1]:=1.866025404;
trans[1,2]:=-0.5;
trans[1,3]:=0.133974596;
trans[1,4]:=-0.5;
trans[2,1]:=-0.5;
trans[2,2]:=1.866025404;
trans[2,3]:=-0.5;
trans[2,4]:=0.133974596;
trans[3,1]:=0.133974596;
trans[3,2]:=-0.5;
trans[3,3]:=1.866025404;
trans[3,4]:=-0.5;
trans[4,1]:=-0.5;
trans[4,2]:=0.133974596;
trans[4,3]:=-0.5;
trans[4,4]:=1.866025404;

for iCount := 1 to numElements do
begin
  for jCount := 1 to SR do
      temp[jCount] := Slab[iCount].SRes[2,jCount];
  for kCount := 1 to SR do
      Slab[iCount].SRes[2,kCount] := Slab[iCount].SRes[3,kCount];
  for kCount := 1 to SR do
      Slab[iCount].SRes[3,kCount] := Slab[iCount].SRes[4,kCount];
  for kCount := 1 to SR do
      Slab[iCount].SRes[4,kCount] := temp[kCount];
end;

for iCount := 1 to num*num do
begin
  for jCount := 1 to SR do
  begin
    Slab[iCount].SResM[1,jCount] :=
      Slab[iCount].SRes[1,jCount]*trans[1,1] +
        Slab[iCount].SRes[2,jCount]*trans[1,2] +
          Slab[iCount].SRes[3,jCount]*trans[1,3] +
            Slab[iCount].SRes[4,jCount]*trans[1,4];
    Slab[iCount].SResM[3,jCount] :=
      Slab[iCount].SRes[1,jCount]*trans[2,1] +
        Slab[iCount].SRes[2,jCount]*trans[2,2] +
          Slab[iCount].SRes[3,jCount]*trans[2,3] +
            Slab[iCount].SRes[4,jCount]*trans[2,4];
    Slab[iCount].SResM[2,jCount] := (Slab[iCount].SResM[1,jCount]
                                     + Slab[iCount].SResM[3,jCount])/2;

    Slab[iCount].SResM[5,jCount] :=
      Slab[iCount].SRes[1,jCount]*trans[3,1] +
        Slab[iCount].SRes[2,jCount]*trans[3,2] +
          Slab[iCount].SRes[3,jCount]*trans[3,3] +
            Slab[iCount].SRes[4,jCount]*trans[3,4];
    Slab[iCount].SResM[4,jCount] := (Slab[iCount].SResM[3,jCount]
                                     + Slab[iCount].SResM[5,jCount])/2;

    Slab[iCount].SResM[7,jCount] :=
      Slab[iCount].SRes[1,jCount]*trans[4,1] +
        Slab[iCount].SRes[2,jCount]*trans[4,2] +
          Slab[iCount].SRes[3,jCount]*trans[4,3] +
            Slab[iCount].SRes[4,jCount]*trans[4,4];
    Slab[iCount].SResM[6,jCount] := (Slab[iCount].SResM[5,jCount]
                                     + Slab[iCount].SResM[7,jCount])/2;
      Slab[iCount].SResM[8,jCount] := (Slab[iCount].SResM[1,jCount]
                                     + Slab[iCount].SResM[7,jCount])/2;
    end;
  end;

  //Average stresses at nodal points to obtain unique values
  for iCount := 1 to numNodes do
  begin
    kCount := 0;
    for jCount := 1 to SR do
      Sum[jCount]:=0;
    for jCount := 1 to num*num do
    begin
      lCount := scanl(iCount,Slab[jCount].NodesNum,8);
      if lCount<>0 then
      begin
        inc(kCount);
        for mCount := 1 to SR do
          Sum[mCount] := Sum[mCount] + Slab[jCount].SResM[lCount,mCount];
      end;
    end;
    if kCount >0 then
    begin
      for jCount := 1 to SR do
        SRes[iCount,jCount] := Sum[jCount]/kCount;
    end;
  end;

  jCount := 1;
  for iCount := 1 to num*num do
  begin
    for kCount := 1 to 8 do
    begin
      if pos = Slab[iCount].NodesNum[kCount] then
      begin
        NodesArray[jCount] := iCount;
        jCount := jCount+1;
      end;
    end;
  end;

end;
```