Stevin Laboratory
Faculty of Civil Engineering and Geosciences
Delft University of Technology

Report 25.5-13.02

17 September 2013

Numerical Investigation on Time-Dependent Flexural and Shear Crack Growth

Finite Element Modelling

Ir. R. Sarkhosh
Ir. J.A. den Uijl
Prof.dr.ir. J.C. Walraven

Mailing address:
Delft University of Technology (TU-Delft)
Faculty of Civil Engineering and Geosciences
Concrete Structures Section
Stevin Laboratory II
Stevinweg 1
2628 CN Delft
The Netherlands

TUDelft Delft University of Technology

Challenge the future

Acknowledgment

The analytical modelling of this research has been carried out in Delft University of Technology (***TU Delft***).

The authors are greatly indebted to the Ministry of Transport, Public Works and Water Management of the Netherlands (RWS) and the Netherlands Organisation for Applied Scientific Research (TNO) for their support.

Table of Contents

CONTENTS *page*

Preface

Crack initiation, crack growth and time-dependency of the crack in unreinforced and reinforced concrete have been an interesting topic for decades. Parallel to the research on the 'shear capacity of concrete beams under sustained loading', in which the shear capacity of large-scale beams with longitudinal reinforcement under high sustained loading is being investigated, this research is done to investigate the behaviour of crack in unreinforced concrete.

A series of short-term and long-term loading has been performed to acquire the ultimate capacity of the beams (in case of short-term loading), displacements, and time of failure with different load ratios (in case of long-term loading).

Subsequently, a finite element model is proposed to develop the acquired results into a general finite element analysis.

The aim of this research is to investigate the time-dependent growth of single crack and multiple cracks in plain concrete beams subjected to sustained loading.

Summary

.

1. Fictitious crack

1.1. Softening behaviour of concrete

For quasi-brittle materials such as concrete, a complete tensile stress-strain curve can be achieved by means of displacement controlled testing, as shown in Fig. 1. At first, the material behaves almost linear elastic but when the stress increases, the curve becomes non-linear due to micro-cracks, which are distributed over the entire specimen. When the maximum stress is reached, one cross section is unable to carry more loads. It is fair to assume that when the specimen becomes more deformed, the development of micro-cracks is concentrated in a small material volume close to this cross section. This means that, after the maximum load is reached, additional deformations take place in the micro-cracked material volume, or fracture zone, while the material outside the fracture zone is elastically unloaded. The load decreases when the first fracture zone develops and consequently only a single zone develops [26].

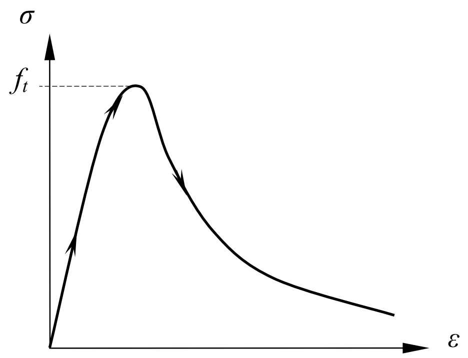


Fig. 1: Complete stress-strain curve

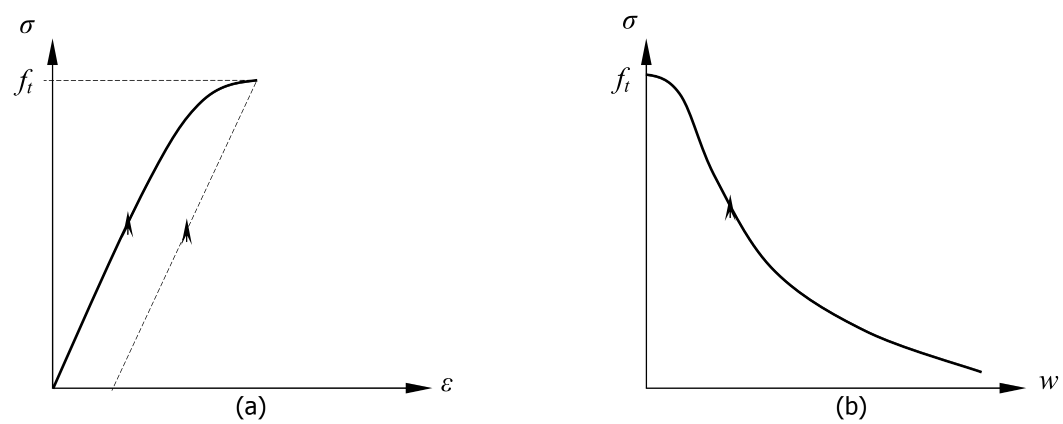


Fig. 2: (a) The deformation properties of the material outside the fracture zone are given by a relation between the stress and the relative strain, i.e. a σ-ε curve. (b) The deformation properties of the fracture zone are given by a relation between the stress and the absolute widening of the zone in the stressed direction, i.e. a σ-w curve

During the tensile test on a concrete specimen, after the maximum stress f_t is reached, the deformation of the fracture zone affects the mean strain and consequently the stress-strain curve of concrete depends on the specimen length. This means that it is unsuitable to use the stress-strain curve as a material property in modelling, unless the strain is obtained in a very small length. A better way of describing the deformation properties of a material therefore is to use two relations; one relation between the stress and the relative strain for the material outside the fracture zone (Fig. 2a) and one relation between the stress and the absolute deformation of the fracture zone (Fig. 2b).

1.2. Stress distribution in front of notch tip

When a notched concrete specimen is subjected to a load, a zone of micro-cracks develops in front of the notch. This fracture zone considerably reduces the stress concentration, which results in a much more realistic description of the stress distribution than the linear elastic solution, see Fig. 3.

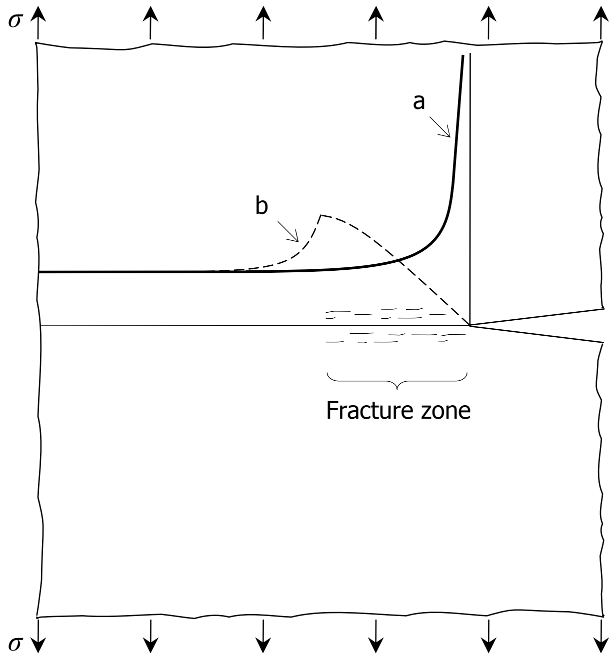


Fig. 3: Probable stress distribution in front of a crack/notch for: (a) a linear elastic material (b) a non-yielding material with a micro-cracked zone in front of the notch tip

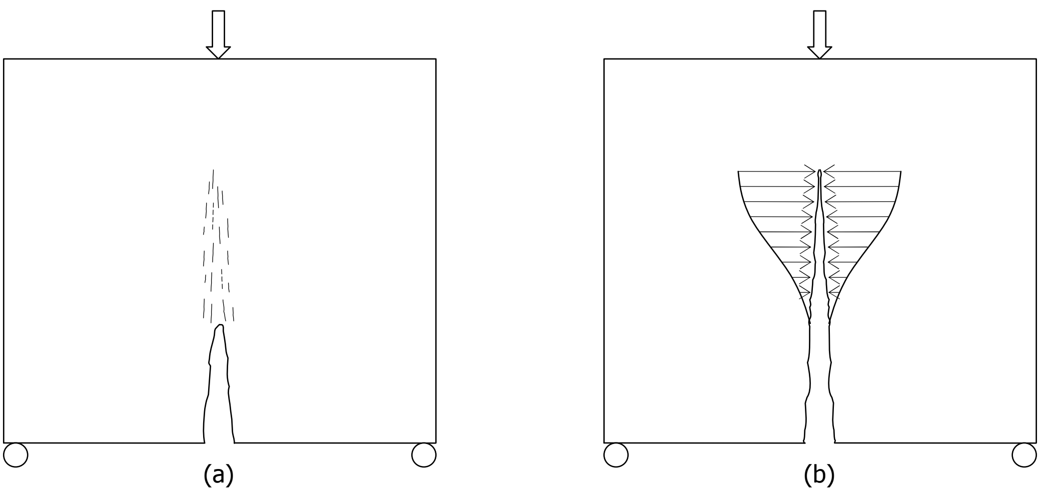


Fig. 4: (a) The fracture zone in front of a crack tip is replaced by a crack that is able to transfer stress. (b) The stress transferring capability depends on the width of the crack according to a σ-w curve, see also Fig. 2b.

The fracture zone in front of a notch or a crack normally develops in a tensile stress field and consequently the properties of this zone are similar to those of the fracture zone in a direct tensile test. This means that it should be possible to approximate the fracture zone in front of a notch or crack. The stress transferring capability depends on the width of the slit in the stressed direction. In Fig. 4, the load is represented by a point-load but of course this description is relevant for all types of loads, including volume stresses due to shrinkage or temperature gradients.

The stress-transferring crack is not a real crack but can be considered as a fictitious crack and therefore the model described above is called the Fictitious Crack Model (FCM) [20]. When using the Fictitious Crack Model the following assumptions are made:

The fracture zone starts developing at one point when the first principal stress reaches the tensile strength of concrete. Of course, other more complicated fracture criteria can be used but often the simple tensile strength criterion is sufficient.

The fracture zone develops perpendicular to the first principal tensile stress.

The material in the fracture zone is partly destroyed but is still able to transfer stress. The stress transferring capability depends on the local deformation of the fracture zone in the direction of the first principal stress. In the calculations, the fracture zone is normally replaced by a stress-transferring crack and the stress transferring capability depends on the width of the crack in the stressed direction according to a σ-w curve, see Fig. 2b.

The width of the fracture zone in the stressed direction is assumed to be equal to the widening of the zone, i.e. the width of the zone is zero when it starts developing. For non-yielding materials like concrete, this should be a fair assumption.

The properties of the material outside the fracture zone are given in a σ-ε curve, see Fig. 2a.

By using the FCM, it is possible to study the development of the fracture zone, the initiation of crack and the crack growth though the material [3]. However, the description of the FCM is relevant for a homogeneous material, i.e. a material that has the same properties in all points. In reality, no materials are perfectly homogenous, at least not at the atomic scale. Nevertheless, if the analysed structure is a few times greater than the largest irregularities in the material, then the material in the structure can be assumed to be approximately homogenous. The σ-w curve is then a function of the fraction and the properties of the components of the material.

1.3. Background

Many researchers have attempted to study the fracture process zone in concrete materials and its softening properties so that their influence on fracture characteristics of concrete structures can be modelled. The experimental investigations showed that the fracture process in concrete structures includes three different stages: crack initiation, stable crack propagation and unstable fracture (or failure) [38].

Usually, the whole region of crack initiation and stable crack propagation is called the fracture process zone (FPZ). In order to predict the crack propagation and to reflect the influence of the FPZ on fracture characteristics of concrete materials, several fracture models such as the fictitious crack model (FCM) by Hillerborg *et al.* [20], the crack band model (CBM) by Bažant and Oh [3], the two parameter fracture model (TPFM) by Jenq and Shah [23], the effective crack model (ECM) by Karihaloo and Nallathambi [24] and Swartz and Refai [34] and the size effect model (SEM) by Bažant, *et al.* [4] have been proposed. In the models, different material parameters are introduced to describe the cracking properties of concrete materials. Correspondingly, the test methods to determine the corresponding fracture parameters which are G_F defined in FCM, K^s_{Ic} and $CTOD_c$ in TPFM and G_F and c_f in SEM have been recommended by RILEM [29], [30] and [31].

The fictitious crack model and the crack band model are nonlinear fracture models that use the softening traction-separation law to model the fracture behaviour of the fracture process zone. The two models can be utilized to predict crack initiation, crack propagation and failure of a concrete structure using a finite element code. The softening traction-separation law is completely determined by three material parameters which are the fracture energy G_F, the tensile strength f_t , and the crack opening at zero stress w_c. According to the method recommended by RILEM [29], the fracture energy G_F, can be experimentally determined by three-point bending of notched beams. However, it was found by many researchers that the obtained values of the fracture energy G_F are size-dependent (Hillerborg [21], Wittmann *et al.*, [37], Xu and Zhao [39], Zhao et al., [40]). Moreover, if the softening traction-separation law (σ -w curve) is approximated by different types (linear, bilinear and exponential types etc.), the determined values of w_c are different even though the values of G_F and f_{tm} are constant.

The critical stress intensity factor K^s_{Ic}, and the critical crack tip opening displacement $CTOD_c$, are introduced as fracture parameters in the two-parameter fracture model [23]. For determining them, an unloading and reloading procedure in a test was performed so that an unloading compliance c_u can be used to evaluate the effective crack length a_c. The nonlinear behaviour of concrete fracture is reflected by the evaluation of the effective crack length a_c. Then, the measured value of the peak load P_{max}, and the evaluated value of the effective crack length a_c, are inserted into a formula of LEFM to determine the critical stress intensity factor K^s_{Ic}. This approach was recommended by RILEM [30]. It can underestimate the effective crack length a_c, because the inelastic part of $CMOD$ is not taken into account. It was found that the inelastic part, or the permanent deformation art of $CMOD$ has an important influence on crack propagation [4] and the nonlinear fracture characteristics of concrete is mainly associated with the fracture process zone or the stable crack propagation. In addition, the stable unloading procedure in tests requires a closed loop testing system. The advantage of this method is to use only a single size of three-point bend beam. All of K^s_{Ic}, $CTOD_c$, a_c and $CMOD_c$ in TPFM model can be directly measured. Therefore, it is possible that the properties of size-independence of K^s_{Ic} and $CTOD_c$ claimed by Jenq and Shah [23], could be further justified by the results that are directly measured.

Tang, Ouyang and Shah have pointed out that there is 'somehow restricted application of TPFM' [35]. Instead, the method of determining fracture parameters K^s_{Ic} and $CTOD_c$ as described by Jenq and Shah [23] and recommended by RILEM [30], they proposed a peak load method.

The effective crack model by Karihaloo and Nallathambi [24] and Swartz and Refai [34] attempts to determine the critical stress intensity factor K_{Ic}, by inserting the measured peak loads $P_{\max}$, and the evaluated effective crack length ac, into the formula of LEFM using a three-point bending test on a single size beam. The effective crack length a_c, is determined by the measured peak load $P_{\max}$, and the corresponding deflection δ_p.

Differing from the material parameters introduced in the two-parameter fracture model and the effective crack model, the critical energy release rate G_F and the critical effective crack extension c_f, for infinite specimen are introduced as the two material parameters into the size effect model by Bažant *et al.* [4]. A method to determine the values of G_F and c_f by testing several three-point bending notched beams of at least three different sizes and a similar geometry has been recommended by RILEM [31].

As a conclusion, in order to predict the crack initiation, crack propagation and failure of concrete material, the Fictitious Crack Model is one of the best models to be utilized. In the following sections, the concept of the FCM will be discussed and explained.

1.3.1. Dugdale model

Dugdale [15] was one of the first researchers who presented a crack model for an elastic-ideal plastic material. Even though the concrete is a brittle material, many approaches to crack analysis based on a single crack concept are based on the Dugdale model or the Barenblatt model [2].

For an elastic-ideal plastic material, the stress can never exceed the yield stress. In the model according to Dugdale, it is assumed that a narrow yield zone develops in front of the crack tip along the line of the crack, see Fig. 5. The stresses in the yield zone never exceed the yield stress and consequently load-case (a) in Fig. 5 equals the sum of the load-cases (b) and (c).

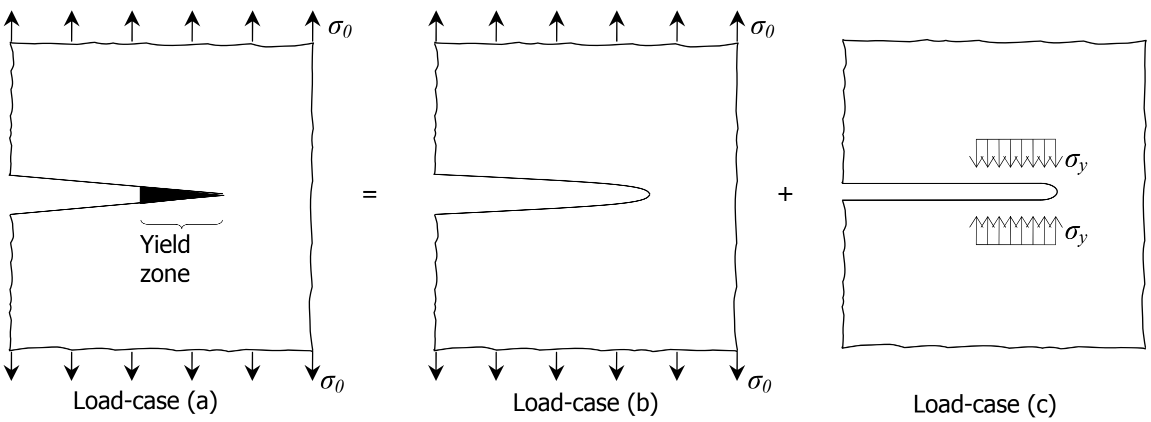


Fig. 5: Dugdale model of a single crack for elastic-ideal plastic material

1.3.2. Model of Hillerborg Modéer and Peterson

Hillerborg, Modéer and Petersson [20] developed the Fictitious Crack Model based on Dugdale [15] and Barnblatt [2] models. The basic idea of their model is demonstrated in Fig. 6a. When using the Finite Element Method (FEM), they modelled the fracture zone by 'nodal forces'. The closing stresses acting across the fracture zone (Fig. 6a) are replaced by nodal forces (Fig. 6b). The intensity of these forces of course depends on the width of the Fictitious Crack according to the σ-w curve of the material. When the tensile strength or another fracture criterion is reached in the top node (Fig. 6b), this node is "opened" and forces start acting on the crack at this point. In this way, it is possible to follow the crack growth through the material.

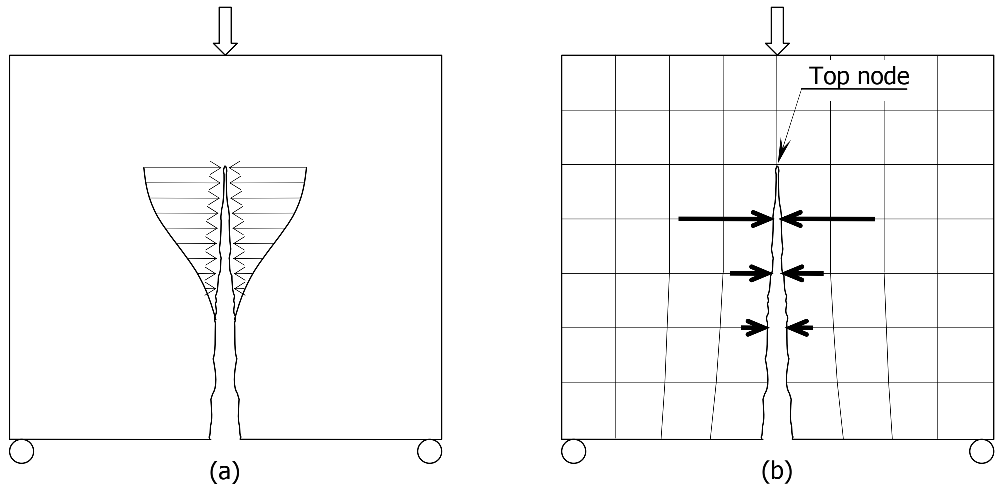


Fig. 6: When using FEM, the stresses acting across the Fictitious Crack (a) are replaced by nodal forces (b)

In Fig. 7, a schematic illustration of a deeply cracked structure subjected to the load is shown. This type of structure is used as the basis in the Hillerborg *et al.* method. The dots on the boundaries of the crack represent finite element nodes. The positions of the two nodes in each node pair (a node pair is two nodes on the opposite crack surfaces at the same distance from the crack tip) coincide when the structure is unloaded. The node pairs are numbered from 1 at the base of the crack to n+1 at the crack tip. The distance between two pairs of nodes i and i+1 is denoted a_i.

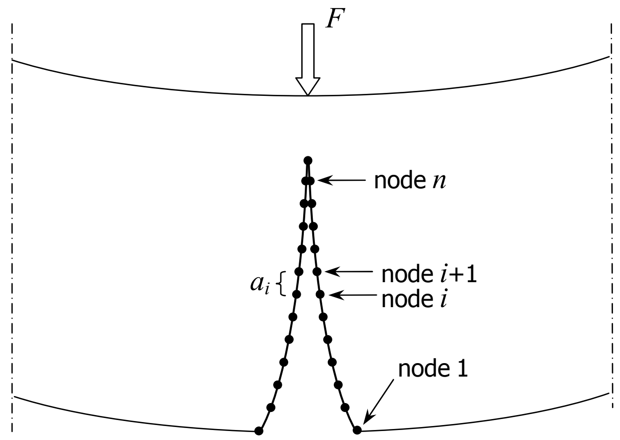


Fig. 7: Schematic illustration of the FE nodes along the crack boundaries in a deeply cracked specimen [20]

By introducing closing forces across the crack, it is possible to make the structure in Fig. 7 relevant for an arbitrary notch depth. According to Petersson [26], if the material is linear elastic and if the deformations are small, the widening of the crack at each node point from node 1 to node n can be expressed by n equations:

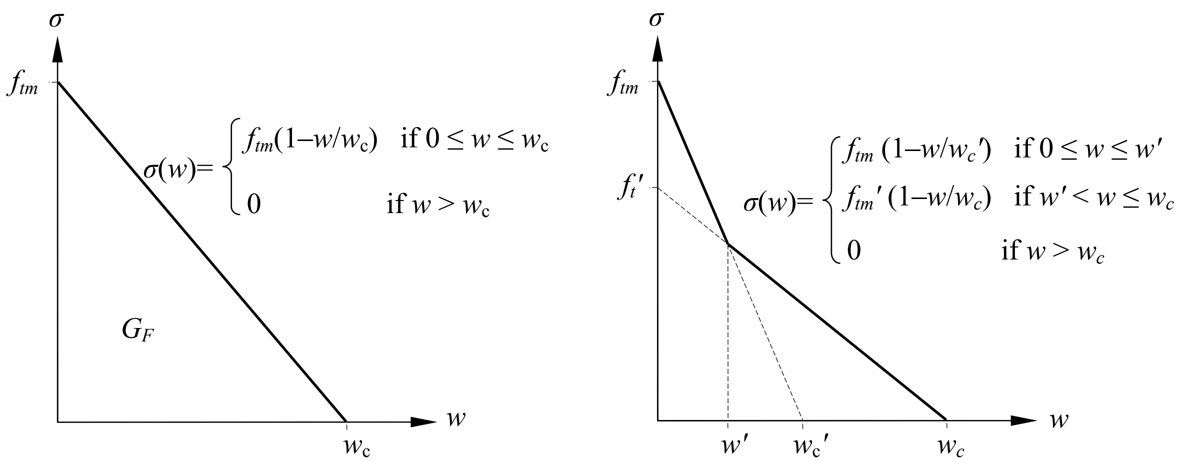
$$w_i = \sum_{j=1}^{n} K(i,j)P(j) + C(i)F \qquad (1)$$

where,
w_i is the width of the crack at node i,
F is the load applied to the structure,
$P(j)$ is the closing force acting at node j,
$K(i, j)$ is widening of the crack at node i of the structure in Fig. 7 when unity load is acting at node j,
$C(i)$ is the widening of the crack at node i of the structure when the applied load equals unity load.

If the crack propagation path is known in advance, then the values of constants $K(i, j)$, $C(i)$, $D(i)$ and D_F are known as well by means of finite element calculations. When determining the constants a number of different load cases are solved but the same global stiffness matrix can be used for all the load cases and consequently it is only necessary to carry out a single inversion of the stiffness matrix.

Sometimes it is impossible to predict the crack propagation path in advance, so a superposition principle should be used. Here, the first step is to apply the load F_1 to the linear elastic structure which gives the stress $\sigma(1, i)$ in each node i. The load F_1 is chosen so that the tensile strength is reached at the crack tip i.e. $\sigma(1, 1) = f_{tm}$. The second step is to "open" node 1 and to introduce opening forces across the crack at this mode. The intensity of the forces must depend on the width of the Fictitious Crack according to the σ-w curve and the area, which is represented by the forces. For the simple straight-lined σ-w curve in Fig. 8a, the intensity of the forces increases linearly from 0 to $a_1 \cdot b \cdot f_{\mathrm{tm}}/2$ when w increases from 0 to w_c. The forces are 0 when $w > w_c$. b is the width of the structure perpendicular to the plane and a_1 is the distance between nodes 1 and 2 (Fig. 7). The load F_2 is chosen so that $\sigma(1, 2)+ \sigma(2, 2)= f_{\mathrm{tm}}$ which means that, when load-case 1 and 2 are combined, the tensile strength is reached at node 2. The total load is then $F_1 + F_2$ and the stresses at the different nodes are given as $\sigma(1, i)+ \sigma(2, i)$. The stresses at node 1 due to load F_2 is negative (the forces at this node want to widen the crack) and consequently the total stress at node 1 decreases according to the σ-w curve.

By using this method, it is possible to choose the propagation direction of the fracture zone after each calculation step. Then the first principal stress is calculated at the tip of the fracture zone and propagation takes place along a path perpendicular to the first principal stress or, as the possible directions of propagation are limited to the directions of the element sides, along the element side, which deviates less from the theoretical propagation direction.



(a) (b)

Fig. 8: (a) The simplest approximation of the σ-w is a single, descending, straight line. (b) The σ-w curve approximated with two straight lines.

1.3.3. Model of Zhou and Hillerborg

Zhou and Hillerborg [41] proposed a time-dependent fracture model for concrete based on material tests. Under long-term loading, creep in the high stress zone around the fictitious crack tip may be high enough to reach the tensile strain capacity, so that crack formation can occur below the static tensile strength. Therefore, the criterion should be adjusted for a time effect. Zhou [42] used the static tensile strength-deformation curve as a criterion in all the models instead of using a stress-failure lifetime relation or stress-strain criterion.

Time-dependent problems are often solved in increments by dividing time into small steps. Under sustained loading it is usual to evaluate incremental creep strains from stresses at the beginning of the time step and structural responses in the time increment can be obtained by imposing a pseudo load from the creep strains. Since this approach cannot be used in the fracture zone, Zhou performed a series of deformation-controlled tests on the fracture zone. At the beginning of each time step, stress relaxations are computed instead, and consequently a pseudo load can be evaluated from the relaxation stresses. The time dependent σ-w relation is expressed in the following form:

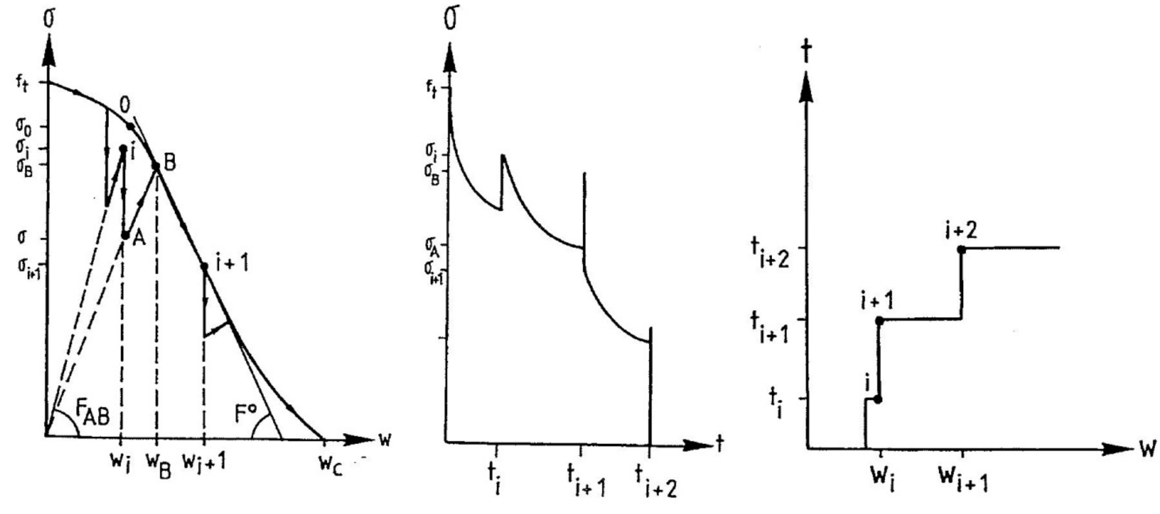
$$d\sigma = d\sigma^R + d\sigma^I \qquad (2)$$

where $d\sigma^R$ and $d\sigma^I$ are stress changes due to relaxation and the deformation increment dw respectively during the time increment dt.

Since it is quite difficult to perform relaxation tests during a long period of time, accurate stress-time functions in relaxation cannot be obtained from the tests. Therefore, simple functions based on experimental evidences are proposed in the model to illustrate the main features of the time effect in the fracture zone.

Fig. 9 illustrates the proposed model. During the time increment $dt = t_{i+1} - t_i$, the deformation is first held at w_i and the stress decrease $d\sigma^R$ due to relaxation is $\sigma_i - \sigma_A$. Then, when the deformation increases from

w_i to w_{i+1}, the stress can increase until it reaches the envelope of the static σ-w curve at point B along the path A-B and follow the curve until point i+1. The actual stress change $d\sigma^I$ is $\sigma_{i+1} - \sigma_A$. Of course, if the deformation increment dw is small, then point i+1 may not reach point B and will instead locate at a point somewhere between A and B. The relaxation function of a modified Maxwell model is chosen.



(a) Stress-deformation curve (b) Stress-time curve (c) Time-deformation curve

Fig. 9: Illustration of the model of Zhou [42].

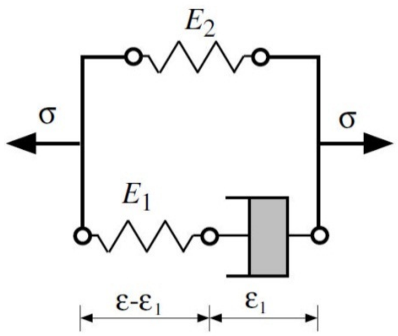


Fig. 10: Rheological model .

Zhou and Hillerborg used a simple rheological element to illustrate the main features of the problems concerned. Rheology is concerned with time-dependent deformation of solids. In the simplest rheological model of the linear standard viscoelastic solid (Fig. 10), the springs are characterized by linear stress–displacement relationships:

$$\begin{aligned} \sigma_1 &= E_1(\varepsilon - \varepsilon_1) \\ \sigma_2 &= E_2\varepsilon \end{aligned} \tag{3}$$

The stress relaxation within time increment dt is assumed to be given by:

$$\begin{aligned} d\sigma^R &= (\sigma_i - \alpha\sigma_0)\left(\exp(-\frac{dt}{\tau}) - 1\right) \quad && \sigma_i > \alpha\sigma_0 \\ d\sigma^R &= 0 && \sigma_i \le \alpha\sigma_0 \end{aligned} \tag{4}$$

where α is a constant, σ_0 is the stress corresponding to w_i in the static σ-w relation and τ is the relaxation time. Relaxation tests in tension show that stress relaxation seems to reach a limit value which is proposed to be equal to $\alpha\sigma_0$. Therefore, in equation (4) the term $\alpha\sigma_0$ has been introduced as a relaxation limit. Stress relaxation below the limit is assumed to be zero.

The stress change $d\sigma^I$ is proposed as:

$$\begin{aligned} d\sigma^I &= F*(w_{i+1}-w_i) & w_{i+1} \le w_B \\ d\sigma^I &= F*(w_B-w_i)+F^0*(w_{i+1}-w_B) & w_{i+1} > w_B \end{aligned} \tag{5}$$

where,

$$\begin{aligned} F &= F_{AB}(\exp(-dt/\tau)+1)/2 \\ F_{AB} &= \frac{\sigma_A}{w_i} = \frac{\sigma_i + d\sigma^R}{w_i} \\ F^0 &= \frac{\partial \sigma^0}{\partial w}(w_B) \end{aligned} \tag{6}$$

and $\sigma^0(w)$ represents the static σ-w curve.

The experimental loading-reloading curve is complicated (Fig. 11), thus in the model a linear stiffness is proposed to make a simple and proper description of the curve possible. Fracture energy reduces as deformation/loading rate is decreased.

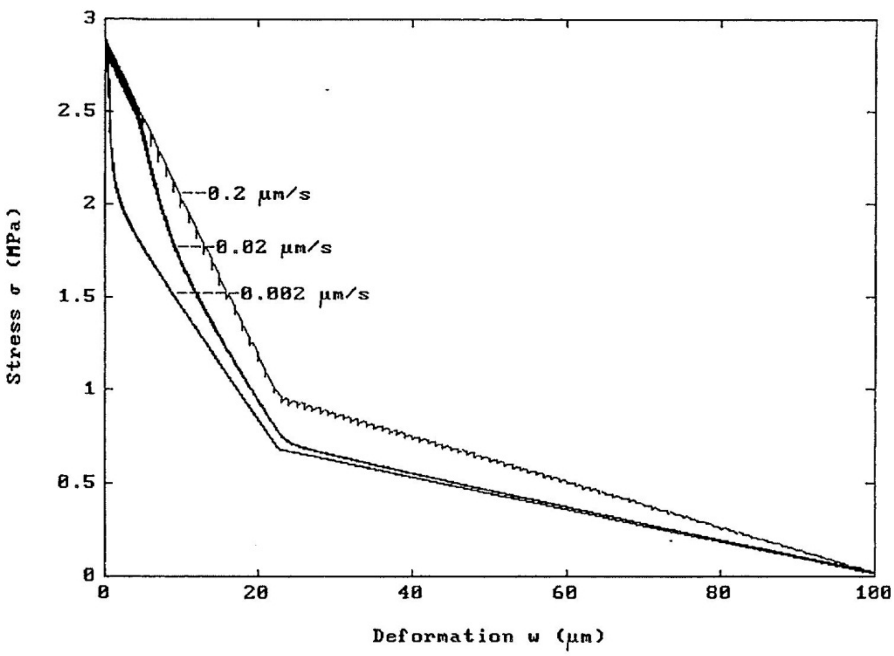


Fig. 11: Simulated tensile σ-w curves at different rates according to the model α=0.7, τ=25 s.

In Fig. 11 the model is applied to simulate stress-deformation curves at different deformation rates. If the rate is high (close to the static loading rate), the stress-deformation curve is near the static one. Meanwhile, the curve deviates more from the static one for slow rates, and the transmitting stress in the fracture zone becomes smaller than the static one for the same deformation.

2. Modelling of flexural and shear crack in plain concrete

2.1. Creep

Time dependent behaviour of quasi-brittle materials is usually described by means of creep or relaxation. In a uniaxial creep test, the stress history σ(t) is prescribed by:

$$\sigma(t) = \begin{cases} 0 & t < 0 \\ \sigma_0 & t \geq 0 \end{cases} \tag{7}$$

where σ_0 is a constant stress applied at time $t = 0$.

The creep strain $\varepsilon(t)$ is expressed as:

$$\varepsilon(t) = J(t)\ \sigma_0 \tag{8}$$

where, $J(t)$ is the creep compliance.

For creep simulation, creep may be divided into two main processes regarding the period of consideration; short-term bulk creep and long-term bulk creep. For duration of less than several months, short-term bulk creep can be calculated from Bažant and Baweja's Model B3 [6]:

$$\phi(t,t') = E(t')\ J(t,t') - 1 \tag{9}$$

$$J(t,t') = q_1 + C_0(t,t') + C_d(t,t',t_0) \tag{10}$$

where $E(t')$ is the modulus of elasticity at loading age t', q_1 is the instantaneous strain due to unit stress, $C_0(t,t')$ is a compliance function for basic creep and $C_d(t,t',t_0)$ is an additional compliance function due to simultaneous drying. A simplified compliance function based on a double power law was proposed earlier by Bažant and Chern [7]:

$$J(t,t') = 1/E_0\ [1+ \phi(t'^{-1/3}+0.05)(t-t')^{1/8}] \tag{11}$$

where, E_0 is 1.5~2.0E. In this study model B3 has been used for modeling together with the EC2 recommendation for creep [12].

For long-term periods exceeding several months, the creep coefficient $\phi(t)$ which is the ratio of the creep displacement to the elastic displacement at time t is expressed by an exponential growth function:

$$\phi(t) = \phi_\infty\ (1 - e^{-t/T}) \tag{12}$$

Where ϕ_∞ is the creep coefficient at time infinity and T is the retardation time at which 63% of the maximum value of ϕ is obtained.

2.2. Crack rate dependency

The rate process of the breakage of bond in the FPZ, which causes the softening law for the crack opening to be rate-dependent can be modelled by a cohesive crack growth model in a viscoelastic material [8]. This paper tries to develop an elastic-visco-plastic model with damage for the rate dependency of the crack strain. Typically, the viscoplastic constitutive equations are developed from a number of spring and dashpot elements arranged in series and parallel.

Two commonly used models are the generalized Maxwell chain and generalized Burger's model where in the former the same strain is shared across all the elements and the stress is additive and in generalized Burger's model the strains are additive and the stress is the same for each element. The generalized Burger's model will be adopted here because it shares the same framework as classical visco-plasticity models and allows non-linearities based on stress to be accommodated more easily [13]. It can be seen from Fig. 12 that the generalized Burger's model comprises an elastic element in series with a number of viscoelastic (Kelvin-Voigt) elements and a viscoplastic element. The stress transmitted through each element and strains are additive such that:

$$\varepsilon(t) = \varepsilon_{el}(t) + \varepsilon_{ve}(t) + \varepsilon_{vp}(t) \tag{13}$$

where ε, ε_{el}, ε_{ve}, ε_{vp} are the total elastic, viscoelastic and viscoplastic strain components at time t.

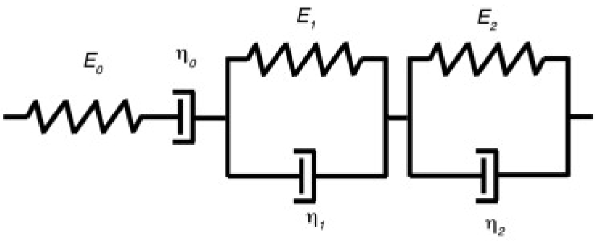


Fig. 12: Generalized Burger's model.

In the previous report of 'Experimental Investigation on Time-Dependent Flexural Crack Growth' [32], details of using a elasto-visco-plastic model with six Voigt elements are given which is fitted to the primary and secondary phase of the cracking strain. That model will be used in the following LEFM method.

[Some more details here….]

2.3. Linear elastic fracture model (LEFM)

Linear elastic fracture models have been widely used to predict the crack path development for quasi-brittle materials such as concrete, even for complex trajectories [18]. With step-wise linear increment, the initiation and the propagation of the crack can be simulated to a global response [41]. which can be reproduced by changing the material properties in every step.

Based on the fictitious crack model [20], [26] and using the finite element method, a linear elastic fracture model is employed to predict the time dependent behaviour of a flexural crack.

Under long-term loading, the strain due to the creep effect in the high stress zone around the fictitious crack tip may be large enough to reach the critical strain, so that crack formation can occur below static tensile strength. Therefore, the criterion should be adjusted to account for the time effect.

Time dependent problems are often solved by dividing time into small increments. Under sustained loading, it is usual to evaluate incremental creep strains from stresses at the beginning of the time step. In order to obtain the strain as a function of time, a bulk creep function should be used; either an existing recommended function for creep (e.g. the Eurocode 2 recommendation for creep [17] or Model B3 [6]) or an experiment-based function by means of a rheological model can be used. The Poisson ratio can be assumed to be time independent (being equal to 0.2) [9]. The structure is considered to be macroscopically homogeneous in the sense that the same $J(t,t')$ applies to every point of the homogenizing continuum throughout the bulk of the structure.

2.3.1. Finite element method

The finite element method with discrete approach has been chosen in modelling of FPZ. A special program was developed in MATLAB and used in all calculations. For the simplification in modelling of a flexural crack, the crack propagation path is assumed to be known in advance and is chosen to coincide with the boundary conditions. Material is modelled as linear elastic by using 4-node plane stress elements. Every single element has four individual nodes, which are connected to the nodes of the next element through a connection matrix. The connection of two/three/four neighbour nodes is lost when the crack passes through the elements. To solve the problem, an iterative method is required to update the stiffness matrix and find the resistant nodal forces on the crack face. Moreover, the stress redistribution due to crack opening in time is considered. The nonlinear behaviour is modelled by an interactive linear model that approximates the crack propagation into stepwise linear increments and regenerates the meshes in each segment.

In linear elastic finite element method, the tensile stress in front of the notch tip approach to infinity as the meshes become smaller. It should be mentioned that the size of the meshes has no influence in the results, because whether the meshes are large or very small, as soon as a node pair becomes open (crack initiation), the tensile forces due to softening behaviour will be applied to these node pairs, which reduces the stress in front of the crack tip. With this method, a finer mesh with small elements gives the best approximation of crack growth.

In this section, the static analysis of 2D beam in plane-stress will be presented. When the thickness of the beam is small compared to the other dimensions and/or the first two principal stresses are assumed to be constant in the normal direction, plane-stress analysis can be employed. In this way, the loads, displacements and boundary conditions are applied or computed in the plane of x-y (longitudinal and vertical directions, respectively) in the case of beam analysis. The stresses, forces and displacements regarding the normal direction to the x-y plane, are assumed to be negligible.

Displacements in x and y directions in a plane-stress problem are defined as:

$$\mathbf{u}(x,y) = \begin{Bmatrix} u(x,y) \\ v(x,y) \end{Bmatrix} \tag{14}$$

And strains can be obtained from derivation of displacements:

$$\boldsymbol{\varepsilon}(x,y) = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{Bmatrix} \tag{15}$$

The stresses in a linear elastic material can be calculated as:

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \mathbf{C} \cdot \boldsymbol{\varepsilon} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{(1-\nu)}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{16}$$

where, $\mathbf{C}$ is the stress-strain matrix that is constant in elastic problems, E is the modulus of elasticity and ν is the Poisson's ratio of the concrete.

In an ideal condition, if concrete is considered as a homogenous and isotropic material;

$$\sigma_z = \tau_{xz} = \tau_{yz} = 0 \tag{17}$$

$$\gamma_{xz} = \gamma_{yz} = 0 \tag{18}$$

The elastic potential energy of a deformed material in a domain Ω bounded by Γ is:

$$U = \frac{1}{2}\int_{\Omega} h\boldsymbol{\varepsilon}^T\boldsymbol{\sigma} d\Omega = \frac{1}{2}\int_{\Omega} h\boldsymbol{\varepsilon}^T\mathbf{C}\boldsymbol{\varepsilon} d\Omega \quad (19)$$

where $\boldsymbol{\varepsilon}^T$ is the transverse matrix of strain, and h is the thickness of the plate (beam).

For a single element with a displacement vector of $\mathbf{u}^e$, the elastic potential energy would be:

$$U^e = \frac{1}{2}\int_{\Omega^e} h\boldsymbol{\varepsilon}^T\boldsymbol{\sigma} d\Omega^e = \frac{1}{2}\int_{\Omega^e} h\boldsymbol{\varepsilon}^T\mathbf{C}\boldsymbol{\varepsilon} d\Omega^e \quad (20)$$

In conformance with the law of conservation of energy, the work done in the small movements of the external forces (W) must be equal to the potential energy (U) stored in the structure. The energy produced by the external forces can be defined as:

$$W = \int_{\Omega} h\mathbf{u}^T\mathbf{b} d\Omega + \int_{\Gamma_t} h\mathbf{u}^T\mathbf{t} d\Gamma \quad (21)$$

where, $\mathbf{u}^T$ is the transverse matrix of displacements, $\mathbf{b}_{1\times 2}$ is the matrix of body forces $\{b_x, b_y\}$, Γ_t is boundary force part, where the natural or force boundary conditions are applied on and $\mathbf{t}$ is the surface traction per unit area.

Once more, the energy by external forces for a single element can be expressed as:

$$W^e = \int_{\Omega^e} h\mathbf{u}^T\mathbf{b} d\Omega^e + \int_{\Gamma_t^e} h\mathbf{u}^T\mathbf{t} d\Gamma^e \quad (22)$$

For a single element with n nodes, the displacement vector defined by $2n$ degree of freedom may be expressed as:

$$\mathbf{u}^e = \{u_1 \quad v_1 \quad u_2 \quad v_2 \quad \cdots \quad u_n \quad v_n\}^T \quad (23)$$

The displacement vector $\mathbf{u}$ in each element is interpolated by nodal displacements:

$$u = \sum_{i=1}^{n} N_i^e u_i; \quad v = \sum_{i=1}^{n} N_i^e v_i \quad (24)$$

where, N_i^e is the element shape function. In a matrix form;

$$\mathbf{u} = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & \cdots & N_n^e & 0 \\ 0 & N_1^e & 0 & N_2^e & \cdots & 0 & N_n^e \end{bmatrix} \mathbf{u}^e = \mathbf{N}\mathbf{u}^e \quad (25)$$

where, $\mathbf{N}_{2\times 2n}$ is the shape function matrix.

With a four nodes ($n = 4$) linear quadrilateral element (Q4) as illustrated in Fig. 13, the four shape functions are:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_4 &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \quad (26)$$

where, ξ and η are natural coordinates of the element.

In quadrilateral element derivations, the Jacobian of two-dimensional transformations that connect the differentials of $\{x, y\}$ to those of $\{\xi, \eta\}$ can be obtained using the chain rule:

$$\begin{Bmatrix} dx \\ dy \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} d\xi \\ d\eta \end{Bmatrix} = \mathbf{J}^{T} \begin{Bmatrix} d\xi \\ d\eta \end{Bmatrix} \tag{27}$$

Here, $\mathbf{J}$ denotes the Jacobian operator, relating natural and global coordinates:

$$\mathbf{J} = \frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{28}$$

$$\mathbf{J}^{-1} = \frac{\partial(\xi,\eta)}{\partial(x,y)} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \tag{29}$$

where, $J = |\mathbf{J}| = \det(\mathbf{J}) = J_{11}J_{22} - J_{12}J_{21}$. Matrices $\mathbf{J}$ and $\mathbf{J}^{-1}$ are called simply the Jacobian and inverse Jacobian, respectively.

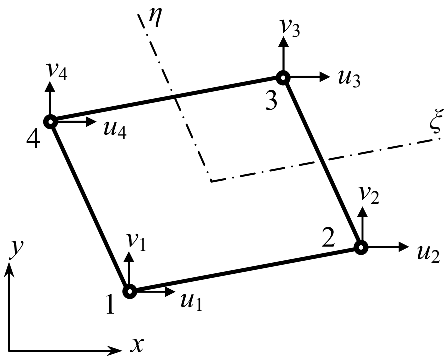


Fig. 13: Quadrilateral (Q4) element in natural ξ-η coordinates and global x-y coordinates

By introducing $\mathbf{B}_{3\times n}$ as the matrix of strain-displacement, Eq. 15 can be written as;

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \frac{\partial N_3^e}{\partial x} & 0 & \frac{\partial N_4^e}{\partial x} & 0 \\ 0 & \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial y} & 0 & \frac{\partial N_3^e}{\partial y} & 0 & \frac{\partial N_4^e}{\partial y} \\ \frac{\partial N_1^e}{\partial y} & \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial y} & \frac{\partial N_2^e}{\partial x} & \frac{\partial N_3^e}{\partial y} & \frac{\partial N_3^e}{\partial x} & \frac{\partial N_4^e}{\partial y} & \frac{\partial N_4^e}{\partial x} \end{bmatrix} \mathbf{u}^e = \mathbf{B}\mathbf{u}^e \tag{30}$$

The nonzero entries of $\mathbf{B}$ are partials of the shape functions with respect to x and y.

Now, Eqs. 20 and 22 will be written as follows:

$$U^e = \frac{1}{2}\int_{\Omega^e} h\boldsymbol{\varepsilon}^T \mathbf{C}\boldsymbol{\varepsilon}\, d\Omega^e = \frac{1}{2}\mathbf{u}^{eT}\mathbf{K}^e\mathbf{u}^e \tag{31}$$

$$W^e = \int_{\Omega^e} h\mathbf{u}^T\mathbf{b}d\Omega^e + \int_{\Gamma_t^e} h\mathbf{u}^T\mathbf{t}d\Gamma^e = \mathbf{u}^{eT}\mathbf{f}^e \tag{32}$$

where, $\mathbf{K}^e$ is the element stiffness matrix and $\mathbf{f}^e$ is the vector of nodal forces as follow;

$$\mathbf{K}^e = \int_{\Omega^e} h\mathbf{B}^T\mathbf{C}\mathbf{B}d\Omega^e \tag{33}$$

$$\mathbf{f}^e = \int_{\Omega^e} h\mathbf{N}^T\mathbf{b}d\Omega^e + \int_{\Gamma_t^e} h\mathbf{N}^T\mathbf{t}d\Gamma^e \tag{34}$$

In order to convert the integral on domain Ω^e to a canonical form, it is required to express $d\Omega^e$ in terms of differentials $d\xi$ and $d\eta$;

$$d\Omega^e = dx\,dy = \det(J)\,d\xi\,d\eta = J\,d\xi\,d\eta \tag{35}$$

Here, Eq. 33 can be written as:

$$\mathbf{K}^e = h\int_A \mathbf{B}^T\mathbf{C}\mathbf{B}\det(J)d\xi d\eta = h\int_{-1}^{1}\int_{-1}^{1}\mathbf{F}d\xi d\eta = \sum_{i,j}^{p,q}\mathbf{F}_{i,j}\,w_{i,j} \tag{36}$$

where, $\mathbf{F}_{i,j}$ is a matrix depends on the natural points (ξ_i, η_j). Integration points (ξ_i, η_j) and integration weights (w_i, w_j) depend on the type of integration, see Fig. 14. p and q are the number of integrating points in ξ and η directions, respectively.

Using a 2 by 2 Gauss points $(p = q = 2)$, the stiffness matrix can be obtained by:

$$\mathbf{K}^e = h\int_{\Omega^e}\mathbf{B}^T\mathbf{C}\mathbf{B}\,d\Omega^e = h\int_{-1}^{1}\int_{-1}^{1}\mathbf{F}d\xi d\eta = \sum_{i=1}^{2}\sum_{j=1}^{2}\mathbf{B}^T\mathbf{C}\mathbf{B}\det(J)\,w_i\,w_j \tag{37}$$

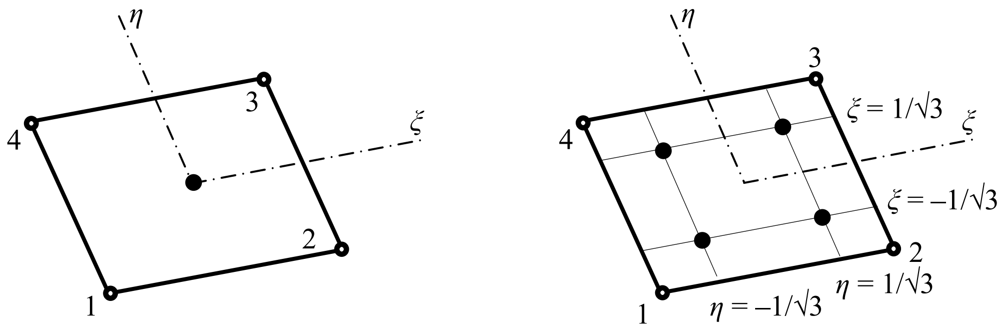


Fig. 14: Left: One Gauss point integration (ξ=0, η=0), Right: Two by two Gauss points integration (ξ=±1/√3, η=±1/√3).

2.4. Modelling of short-term 3-point bending test on a notched beam in Matlab

A concrete beam as illustrated in Fig. 15, which is assumed to have a constant stress in transverse direction and can be modelled as a plate, is loaded under displacement-controlled test until failure. Due to symmetry, only half of the beam is modelled.

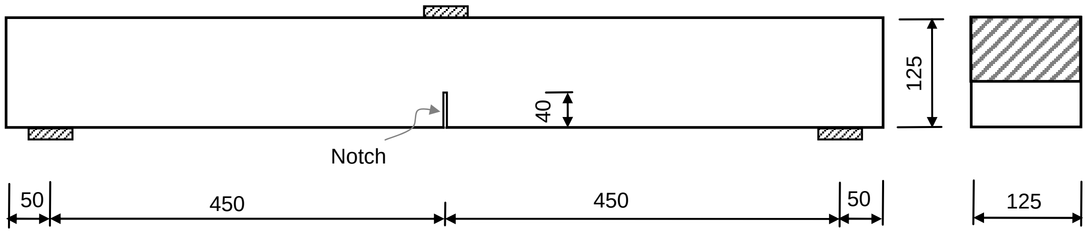


Fig. 15: Geometry of the specimens, (all dimensions are in mm)

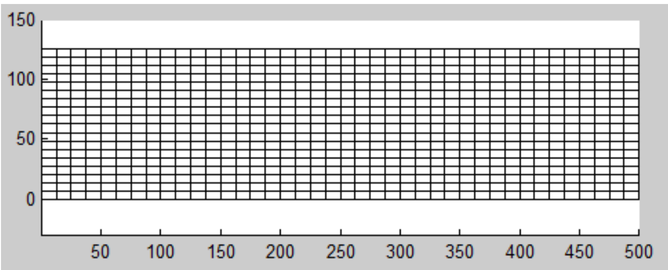


Fig. 16: FE mesh of the half of the specimen.

The first part of the FE code is the material properties, choosing between two codes of ACI or Eurocode2, calculating the softening behaviour according to [21], geometry of the specimen such as the position of the support, the notch depth, length and height and the mesh generation:

```
%..........................
% shorttermflexural.m
% ---------------------------------
% clear memory
clear all;colordef white;clf
totalsteps=12;
showstressgraph=0; %=0 graph off    =1 graph on
Code='ACI'; % EC2 or ACI

% ---------------------------------
% Given material properties
% ---------------------------------
fcube=52;
poisson = 0.20;
thickness=125;
rho=1;

% ---------------------------------
% Calculated material properties
% ---------------------------------
fcm = 0.785 * fcube;
E = 22000*(fcm/10)^0.3;
if strcmp(Code, 'EC2')==1
   fctm = 0.3*(fcm-8)^(2/3); % Eurocode2
else
   fctm=(145.0377*fcm)^0.5*6.7/145.0377; % based on ACI
end
```

```
% ---------------------------------
% Softening behaviour
% ---------------------------------
alphaF=7; alphaD=6;
GF=alphaD/1000*fcm^0.7;
w0=GF*alphaF/fctm;
ws=2*GF/fctm-0.15*w0;
sigmaS=0.15*fctm;
wc=ws/0.85;
ft=w0*sigmaS/(w0-ws);

% ---------------------------------
% Geometry (Support, boundary conditions, length, height)
% ---------------------------------
supportX=450;
Notchdepth=40;
Lx=500;
Ly=125;

% ---------------------------------
% matrix C
% ---------------------------------
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% ---------------------------------
% load
% ---------------------------------
P = -30;

% ---------------------------------
%Mesh generation
% ---------------------------------
numberElementsX=40;
numberElementsY=18;
numberElements=numberElementsX*numberElementsY;
```

In the second part of the code, the node coordinates and node numbers are defined. The elements and nodes are numbered from bottom left in a row, see Fig. 17-Left. Since in this model the crack follows the path between the elements, it is important to have discrete elements with individual node numbers. In order to connect the elements in the mesh, a 'connection matrix' is hired to connect the nodes in a neighbourhood. For instance, in Fig. 17-Left, five neighbourhoods wherein the nodes are connected to each other are: 2&3, 5&9, 6&7&10&11, 8&12 and 14&15. Later, in a modified matrix of element nodes, the number of nodes in each neighbourhood is reduced to the lowest number, see Fig. 17-Right.

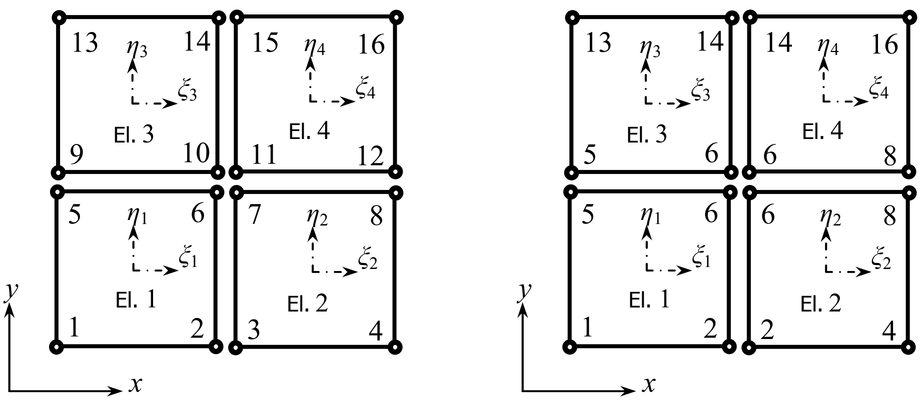


Fig. 17: Left: Numbering of elements and nodes. Right: Modified element nodes.

In case of crack initiation or growth between two elements, the connection between the nodes of two elements will be lost. For instance in case of Fig. 17, if the crack initiates between elements 1 and 2, the connection of nodes 2 and 3 would be lost and these two nodes are not neighbours anymore and if the crack grows between elements 2 and 4, the connection between nodes 6&7, 10&7 and 11&7 would be dropped.

```
% ----------------------------------------------------------------------
% Node coordinates
% ----------------------------------------------------------------------
nodeCoordinates=zeros((numberElementsX*2)*(numberElementsY*2),2);
for j=1:numberElementsY
    for i=1:numberElementsX
        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),1)= (i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),1)= i*Lx/numberElementsX;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),1)=(i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),1)=i*Lx/numberElementsX;

        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),2)=(j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),2)=(j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),2)=j*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),2)=j*Ly/numberElementsY;
    end;
end;

% ----------------------------------------------------------------------
% Element nodes
% ----------------------------------------------------------------------
elementNodes_primary=zeros(numberElements,4);
for j=0:numberElementsY-1
    for i=1:numberElementsX
        elementNodes_primary(i+j*(numberElementsX),1)= i*2-1+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),2)= i*2+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),3)= i*2+numberElementsX*(4*j+2);
        elementNodes_primary(i+j*(numberElementsX),4)= i*2-1+numberElementsX*(4*j+2);
    end;
end;

xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
numberNodes=size(xx,1);

% ----------------------------------------------------------------------
% Connection matrix
% ----------------------------------------------------------------------
connection=zeros(numberNodes);
for j=1:numberNodes
    for i=j+1:numberNodes
        if nodeCoordinates(i,:)==nodeCoordinates(j,:)
            connection(i,j)=1;
            connection(j,i)=1;
        end
    end;
end;
connectionTri=triu(connection);

% ----------------------------------------------------------------------
% Modification element nodes
% ----------------------------------------------------------------------
elementNodes=elementNodes_primary;
removednodesnr=0;
for j=1:numberElements;
    for i=1:4
        findconnection=find(connection(:,elementNodes(j,i))==1);
        if size(findconnection)>=1
            if min(findconnection)<elementNodes(j,i)
                removednodesnr=removednodesnr+1;
                removednodes(removednodesnr,1)=elementNodes(j,i);
            end
            elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
        end
    end;
end;
```

```
drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
```

In the next part of the code, the boundary condition is applied to the model. Due to symmetry, the nodes on the left bond, which are in the middle of the beam, are fixed in x direction except the nodes corresponding to the notch depth. The stiffness matrix is derived according to the method explained in the previous section. Moreover, the crack path is assumed to be along the notch tip (vertically).

```
% -------------------------------------------------------------------
% GDof: global number of degrees of freedom
% -------------------------------------------------------------------
GDof=2*numberNodes;

% -------------------------------------------------------------------
% Boundary conditions
% -------------------------------------------------------------------
fixedNodeX=find(nodeCoordinates(:,1)==0);  % fixed in XX
findnotchtip=round(Notchdepth/Ly*numberElementsY+0.5);
for i=1:findnotchtip*2-1
    fixedNodeX(1,:)=[];    % Removing nodes related to notch
end;

fixedNodeY=find(nodeCoordinates(:,1)==supportX & nodeCoordinates(:,2)==0);  % fixed in YY

% -------------------------------------------------------------------
% Force vector (point load applied at xx=0, yy=Ly)
% -------------------------------------------------------------------
force=zeros(GDof,1);
leftBord=find(nodeCoordinates(:,1)==0);
force(leftBord(end)+numberNodes)=P;

% -------------------------------------------------------------------
% Stiffness Matrix
% -------------------------------------------------------------------
stiffness=zeros(GDof);
        gaussLocations=[ -0.577350269189626 -0.577350269189626;
                          0.577350269189626 -0.577350269189626;
                          0.577350269189626  0.577350269189626;
                         -0.577350269189626  0.577350269189626];
        gaussWeights=[ 1;1;1;1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  ndof=length(indice);

  % cycle for Gauss point
   for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

% shape functions and derivatives
    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta); (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives = 1/4*[-(1-eta), -(1-xi);1-eta, -(1+xi); 1+eta, 1+xi;-(1+eta), 1-xi];

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives] = Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

%  B matrix
    B=zeros(3,2*ndof);
    B(1,1:ndof) = XYderivatives(:,1)';
    B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
    B(3,1:ndof) = XYderivatives(:,2)';
    B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

% stiffness matrix
    stiffness(elementDof,elementDof) = stiffness(elementDof,elementDof) + B'*C*thickness*B*gaussWeights(q)*det(Jacob);
```

```
  end;
end;

% ----------------------------------------------------------------------
% Crack along the notch
% ----------------------------------------------------------------------
for i=findnotchtip:numberElementsY-1
   crackpath(i-findnotchtip+1,:)=[i*numberElementsX*4+1,0,0,(i*2-1)* numberElementsX*2+1,2];
end;
```

Finally, in the last part, the stresses and the principal stresses are calculated, the nodes in front of the notch tip that is defined in the previous part, become open when the first principal stress exceeds the tensile strength f_{ctm}. If the principal stress is less than f_{ctm}, the load is increased linearly to increase the stresses and thus to let the crack grow.

There is a loop in this part that every time, checks the length of the crack and whenever the length of the crack was greater than zero the nodal forces on the opened nodes due to softening of the concrete are applied.

```
cracklength=0; stepnumber=0; savingnr=0; force_initial=force;

% ----------------------------------------------------------------------
% Beginning of analysis
% ----------------------------------------------------------------------
while stepnumber<totalsteps;
   stepnumber=stepnumber+1;
   iteration=1;
   iter=0;
   force_crack=zeros(GDof,1);
   upwardlift=0;

   if cracklength>0 % changes in connection matrix/boundary condition
      fixedNodeX(fixedNodeX==crackpath(cracklength,1),:)=[];
      fixedNodeX(fixedNodeX==crackpath(cracklength,4),:)=[];
   end
   prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
   activeDof=setdiff([1:GDof]', [prescribedDof]);
   activeDof=setdiff(activeDof,[removednodes; removednodes+numberNodes]);


   while iteration==1 %for iter=1:4

      savingnr=savingnr+1;
      force=force_initial*iter;

      % ------------------------------------------------------------
      % nodal forces
      % ------------------------------------------------------------
      if cracklength>0;
         crackforce(cracklength,stepnumber)=0;
         cracksigma(cracklength,stepnumber)=0;
         fixedNodeX(fixedNodeX==crackpath(cracklength,1),:)=[];
         fixedNodeX(fixedNodeX==crackpath(cracklength,4),:)=[];
         prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
         activeDof=setdiff([1:GDof]', [prescribedDof]);
         activeDof=setdiff(activeDof,[removednodes; removednodes+numberNodes]);

         % --------------------------------------------------------
         % Iterative method to find crack nodal forces
         % --------------------------------------------------------
         force_crack=zeros(GDof,1);

         for k=1:3 %stopiteration=1; while stopiteration==1
            U=zeros(GDof,1);
            U(activeDof)=stiffness(activeDof,activeDof) \  (force(activeDof)+force_crack(activeDof));
            displacements=U;

            % ----------------------------------------------------
            % Crack width, crack sigma, nodal forces
```

```
        % ------------------------------------------------------
        force_old=min(force_crack);
        for j=1:cracklength
            crackwidth(j,stepnumber)=2*displacements(((j+findnotchtip-1)*2-1)*numberElementsX*2+1);
            if crackwidth(j,stepnumber)>0
                if crackwidth(j,stepnumber)>w0
                    cracksigma(j,stepnumber)=0;
                else
                    if crackwidth(j,stepnumber)<ws
                        cracksigma(j,stepnumber)=fctm*(1-crackwidth(j,stepnumber)/wc);
                    else
                        cracksigma(j,stepnumber)=ft*(1-crackwidth(j,stepnumber)/w0);
                    end
                end
            else
                cracksigma(j,stepnumber)=fctm;
            end
            if j==1
                crackforce(j,stepnumber)=cracksigma(j,stepnumber)*thickness*Ly/numberElementsY/2;
            else
                crackforce(j,stepnumber)=cracksigma(j,stepnumber)*thickness*Ly/numberElementsY;
            end

            force_crack(((j+findnotchtip-1)*2-1)*numberElementsX*2+1)=-crackforce(j,stepnumber)/2;
            force_crack(((j+findnotchtip-1)*2)*numberElementsX*2+1)=-crackforce(j,stepnumber)/2;
        end;

        if abs((min(force_crack)-force_old)/force_old)<0.01
            stopiteration=0;
        end
    end;
end %if

% ---------------------------------------------------------------
% Solution
% ---------------------------------------------------------------
U=zeros(GDof,1);
U(activeDof)=stiffness(activeDof,activeDof)\(force(activeDof)+force_crack(activeDof));
displacements=U;
Reaction(savingnr)=-P*iter;
midspandeflection(savingnr)=U(1+numberNodes);
if iter==0
    upwardlift=midspandeflection(savingnr);
end
midspandeflection(savingnr)=U(1+numberNodes)-upwardlift;

% ---------------------------------------------------------------
% Stresses at nodes
% ---------------------------------------------------------------
stress=zeros(numberElements,size(elementNodes,2),3);
stressPoints=[-1 -1;1 -1;1 1;-1 1];

for e=1:numberElements
    indice=elementNodes(e,:);
    elementDof=[ indice indice+numberNodes ];
    nn=length(indice);
    for q=1:size(gaussWeights,1)
        pt=gaussLocations(q,:);
        wt=gaussWeights(q);
        xi=pt(1);
        eta=pt(2);
        % shape functions and derivatives
        shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);  (1+xi)*(1+eta);(1-xi)*(1+eta)];
        naturalDerivatives= 1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);   1+eta,      1+xi;-(1+eta),   1-xi];
        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives]= Jacobian(nodeCoordinates(indice,:),naturalDerivatives);
        %  B matrix
        B=zeros(3,2*nn);
        B(1,1:nn)       = XYderivatives(:,1)';
        B(2,nn+1:2*nn)  = XYderivatives(:,2)';
```

```
            B(3,1:nn)       = XYderivatives(:,2)';
            B(3,nn+1:2*nn)  = XYderivatives(:,1)';

            % element deformation
            strain=B*displacements(elementDof);
            stress(e,q,:)=C*strain;
         end
      end
      % Principal Stress
      stress_principal1=(stress(:,:,1)+stress(:,:,2))/2+ ((stress(:,:,1)-stress(:,:,2)).^2/4+stress(:,:,3).^2).^0.5;

      % -----------------------------------------------------------------
      % Stresses in nodes along the tip (the mean stress between 2 nodes)
      % -----------------------------------------------------------------
      for i=numberElementsY-1:-1:findnotchtip %+cracklength
         nodalstress(i-findnotchtip+1,cracklength+2)= (stress_principal1(i*numberElementsX+1,1)+...
            stress_principal1((i-1)*numberElementsX+1,4))/2;
         nodalstress(i-findnotchtip+1,1)=i;
      end;
      nodalstress((numberElementsY-findnotchtip)+1,cracklength+2)=...
         stress_principal1((numberElementsY-1)*numberElementsX+1,4);
      nodalstress((numberElementsY-findnotchtip)+1,1)=numberElementsY;

      stress_max = (nodalstress(cracklength+1,cracklength+2)+...
         nodalstress(cracklength+2,cracklength+2))/2;

      if stress_max>fctm
         cracklength=cracklength+1;
         iteration=0;
         Reaction(savingnr)=-P*iter*fctm/stress_max;
         midspandeflection(savingnr)=(U(1+numberNodes)-upwardlift)*fctm/stress_max;
         Results2(cracklength+1,:)=[midspandeflection(savingnr), Reaction(savingnr)];
      end

      Results(savingnr,:)=[midspandeflection(savingnr), Reaction(savingnr)];%,stepnumber,cracklength];

      iter=iter+1;
   end; %iteration

end; %stepnumber
plot(Results2);
```

The results of this model for the given data in Table 1, is show in Fig. 18. The load-deflection curves show from the models (ACI expressions and EC2) are both showing a conservative results comparing to the test results. The reason could be that the actual tensile strength was higher than the expressions from EC2 or ACI.

Table 1: General properties of the material and meshing

	f_{cm} [Mpa]	f_{ctm} [MPa]	Poisson's ratio	Nr. Elem. in x dir.	Nr. Elem. in y dir.
Matlab (EC2)	52	3,74	0,2	40	18
Matlab (ACI)	52	4,01	0,2	40	18
Test results (C2B1, C2B2, C2B3)	52	-	-	-	-

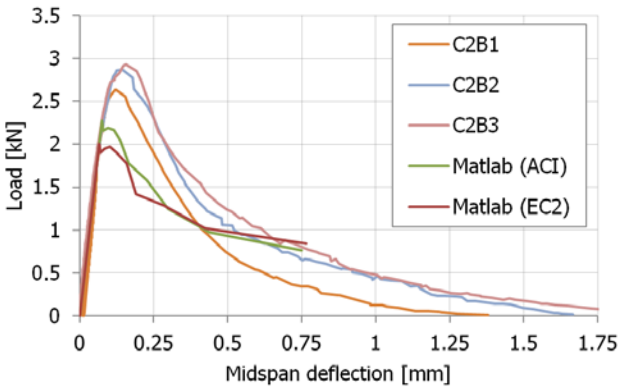


Fig. 18: load-deflection graph; FE model in Matlab and three test results

2.5. Modelling of long-term flexural test on a notched beam in Matlab

In order to model the long-term sustained loading, the beam illustrated in Fig. 15, is loaded under load-controlled test. Similar to the short-term loading, only half of the beam is modelled, due to symmetry.

In Table 2, a comparison of the predicted failure time between the test results and FE modelling is presented. The result of this model is very much dependent on the following terms:

- Creep function: Creep functions have different approach to the strain in time. Depending on the type of creep function, which is used in this mode, the failure occurs faster or slower.
- Initial strain: The strain at the beginning of the long-term loading has a big influence on the time of failure. The results of initial strain from the test results are used in this model.
- Fracture energy: The softening behaviour of the concrete is another important criterion that affects the time of failure. The fracture energy from [21] is calibrated with the test results.

Table 2: Comparison of failure time in test results and FE-Model. (The given times are in seconds)

Load ratio	60	67%	70%	75%	80%	85%	90%
Failure time-FE Model	9720000	3290000	2000000	783000	216000	25000	27
Failure time-Test results	-	2224000	1000000	200000	10000	2000	100

The first part of the code is almost similar to the short-term modelling, however the end time and the load ratio should be given, as well as the ultimate capacity according to the short-term modelling in section 0.

```
%..........................
% longtermflexural.m
% ---------------------------------
% clear memory
clear all;colordef white;clf
loadratio=0.6;
endtime=12000000;%*24*3600/time_interval; %(seconds of test)
time_interval=endtime/100;
showstressgraph=0; %=0 graph off    =1 graph on
Code='EC2'; % EC2 or B3

% ---------------------------------
% Given material property
% ---------------------------------
fcube=52;
poisson = 0.20;
thickness=125;
Lx=500; %Length of the beam
Ly=125; %Height
rho=1;
critical_strain=0.00015;
```

```
% ---------------------------------
% Calculated material property
% ---------------------------------
fcm = 0.785 * fcube;
E = 22000*(fcm/10)^0.3;
if strcmpi(Code,'EC2')==1
   fctm = 0.3*(fcm-8)^(2/3); % Eurocode2
else
   fctm=(145.0377*fcm)^0.5*6.7/145.0377; % fctm based on ACI
end

% ---------------------------------
% Softening behaviour
% ---------------------------------
alphaF=7; alphaD=6;
GF=alphaD/1000*fcm^0.7*2;%%%%
w0=GF*alphaF/fctm;
ws=2*GF/fctm-0.15*w0;
sigmaS=0.15*fctm;
wc=ws/0.85;
ft=w0*sigmaS/(w0-ws);

% ---------------------------------
% Creep function based on EC2 or B3 Model
% ---------------------------------
RH = 0.5; %Relative Humidity
if strcmpi(Code,'EC2')==1
   % Creep according to Eurocode
   alpha1=(35/fcm)^0.7; alpha2=(35/fcm)^0.2; alpha3=(35/fcm)^0.5;
   betafcm=16.8/fcm^0.5;
   h0=thickness*Ly/(thickness+Ly);
   ts=7; %curing time in days
   t0=28; % age in days at loading
   if fcm>35
      phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1)*alpha2;
      betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250*alpha3,1500*alpha3);
   else
      phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1);
      betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250,1500);
   end
   betat0=1/(0.1+t0^0.2);
   phi0=phiRH*betafcm*betat0;
else
   % Creep according to Model B3
   ts=7; %curing time in days
   t0=28; % age in days at loading
   Qf=(0.086*t0^(2/9)+1.21*t0^(4/9))^(-1);
   m=0.5; n= 0.1;
   rt=1.7*t0^0.12+8;
   cement=330; water=189; aggregate=1803;
   q1=0.6*1000000/E;
   q2=185.4*cement^0.5*fcm^(-0.9);
   q3=0.29*(water/cement)^4*q2;
   q4=20.3*(aggregate/cement)^(-0.7);

   ks=1.25;%(slab=1)(inf cyl=1.15)(inf sq prsm=1.25)(sphr=1.30)(cube=1.55)
   kt=8.5*ts^(-0.08)*fcm^(-0.25)/100;
   h0=thickness*Ly/(thickness+Ly);
   tao_sh=kt*(ks*h0)^2;
   alpha1=1.1; %(CEMENT type I=1.0)(CEM Type II=0.85)(CEM Type III=1.1)
   alpha2=1.0; %(Steam curing=0.75)(Normal Cur=1.2)(Curing in RH100%=1.0)
   Ht=1-(1-RH)*(tanh(((t0-ts)/tao_sh)^0.5));
   epsilon_infinit=-alpha1*alpha2*(1.9/100*(water^2.1)*(fcm^(-0.28))+270);
   q5=7.57*100000/fcm*(abs(epsilon_infinit)^(-0.6));
end

% ---------------------------------
% Support and boundary conditions
% ---------------------------------
supportX=450;
```

```
Notchdepth=40;

% ---------------------------------
% matrix C
% ---------------------------------
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% ---------------------------------
% load
% ---------------------------------
Pu= -941;
P = Pu*loadratio;
```

In the second part, the parameters of crack growth rate are given according to the test results []. Mesh generation, matrix of node coordinates and connection matrix are similar to section 0.

```
% ---------------------------------
% Parameters of crack growth rate
% ---------------------------------
tao1 = 5;  tao2 = 25; tao3 = 200; tao4 = 1000; tao5 = 10000; tao6 = 100000;
A1 = 13600-1000* log(tao1) ; B1 = 0.53*log(tao1)-8.75; E1 = A1*loadratio^B1; eta1 = tao1 * E1;
A2 = 13600-1000* log(tao2) ; B2 = 0.53*log(tao2)-8.75; E2 = A2*loadratio^B2; eta2 = tao2 * E2;
A3 = 13600-1000* log(tao3) ; B3 = 0.53*log(tao3)-8.75; E3 = A3*loadratio^B3; eta3 = tao3 * E3;
A4 = 13600-1000* log(tao4) ; B4 = 0.53*log(tao4)-8.75; E4 = A4*loadratio^B4; eta4 = tao4 * E4;
A5 = 13600-1000* log(tao5) ; B5 = 0.53*log(tao5)-8.75; E5 = A5*loadratio^B5; eta5 = tao5 * E5;
A6 = 13600-1000* log(tao6) ; B6 = 0.53*log(tao6)-8.75; E6 = A6*loadratio^B6; eta6 = tao6 * E6;
E0 = 1646.1*loadratio^(-2.516); eta0 = 90000000*loadratio^(-7.4);

% ---------------------------------
% Mesh generation
% ---------------------------------
numberElementsX=40;
numberElementsY=18;
numberElements=numberElementsX*numberElementsY;

% ------------------------------------------------------------------
% node coordinates
% ------------------------------------------------------------------
nodeCoordinates=zeros((numberElementsX*2)*(numberElementsY*2),2);
for j=1:numberElementsY
   for i=1:numberElementsX
      nodeCoordinates(i*2-1+numberElementsX*4*(j-1),1)=...
         (i-1)*Lx/numberElementsX;
      nodeCoordinates(i*2+numberElementsX*4*(j-1),1)=...
         i*Lx/numberElementsX;
      nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),1)=...
         (i-1)*Lx/numberElementsX;
      nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),1)=...
         i*Lx/numberElementsX;

      nodeCoordinates(i*2-1+numberElementsX*4*(j-1),2)=...
         (j-1)*Ly/numberElementsY;
      nodeCoordinates(i*2+numberElementsX*4*(j-1),2)=...
         (j-1)*Ly/numberElementsY;
      nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),2)=...
         j*Ly/numberElementsY;
      nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),2)=...
         j*Ly/numberElementsY;
   end;
end;

% ------------------------------------------------------------------
% element nodes
% ------------------------------------------------------------------
elementNodes_primary=zeros(numberElements,4);
for j=0:numberElementsY-1
   for i=1:numberElementsX
      elementNodes_primary(i+j*(numberElementsX),1)=...
```

```
            i*2-1+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),2)=...
            i*2+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),3)=...
            i*2+numberElementsX*(4*j+2);
        elementNodes_primary(i+j*(numberElementsX),4)=...
            i*2-1+numberElementsX*(4*j+2);
    end;
end;

xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
numberNodes=size(xx,1);

% -----------------------------------------------------------------------
% Connection matrix
% -----------------------------------------------------------------------
connection=zeros(numberNodes);
for j=1:numberNodes
    for i=j+1:numberNodes
        if nodeCoordinates(i,:)==nodeCoordinates(j,:)
            connection(i,j)=1;
            connection(j,i)=1;
        end
    end;
end;
connectionTri=triu(connection);

% -----------------------------------------------------------------------
% Modification element nodes
% -----------------------------------------------------------------------
elementNodes=elementNodes_primary;
removednodesnr=0;
for j=1:numberElements;
    for i=1:4
        findconnection=find(connection(:,elementNodes(j,i))==1);
        if size(findconnection)>=1
            if min(findconnection)<elementNodes(j,i)
                removednodesnr=removednodesnr+1;
                removednodes(removednodesnr,1)=elementNodes(j,i);
            end
            elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
        end
    end;
end;

drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
```

Additionally, in the third part, boundary conditions, force vector and stiffness matrix are derived.

```
% -----------------------------------------------------------------------
% GDof: global number of degrees of freedom
% -----------------------------------------------------------------------
GDof=2*numberNodes;

% -----------------------------------------------------------------------
% boundary conditions
% -----------------------------------------------------------------------
fixedNodeX=find(nodeCoordinates(:,1)==0);  % fixed in XX
findnotchtip=round(Notchdepth/Ly*numberElementsY+0.5);
for i=1:findnotchtip*2-1
    fixedNodeX(1,:)=[];   % removing nodes related to notch
end;

fixedNodeY=find(nodeCoordinates(:,1)==supportX & ...
    nodeCoordinates(:,2)==0);  % fixed in YY

% -----------------------------------------------------------------------
% force vector (point load applied at xx=0, yy=Ly)
% -----------------------------------------------------------------------
```

```
force_primary=zeros(GDof,1);
leftBord=find(nodeCoordinates(:,1)==0);
force_primary(leftBord(end)+numberNodes)=P;
force=force_primary;

% --------------------------------------------------------------------
% Stiffness Matrix
% --------------------------------------------------------------------
stiffness=zeros(GDof);
       gaussLocations=...
         [ -0.577350269189626 -0.577350269189626;
            0.577350269189626 -0.577350269189626;
            0.577350269189626  0.577350269189626;
           -0.577350269189626  0.577350269189626];
       gaussWeights=[ 1;1;1;1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  ndof=length(indice);

  % cycle for Gauss point
   for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

% shape functions and derivatives
    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
           (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives=...
          1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                1+eta,      1+xi;-(1+eta),   1-xi];

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=...
       Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

%  B matrix
    B=zeros(3,2*ndof);
    B(1,1:ndof)        = XYderivatives(:,1)';
    B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
    B(3,1:ndof)        = XYderivatives(:,2)';
    B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

% stiffness matrix
    stiffness(elementDof,elementDof)=...
       stiffness(elementDof,elementDof)+...
       B'*C*thickness*B*gaussWeights(q)*det(Jacob);

   end;

end;

% --------------------------------------------------------------------
% Crack along the notch
% --------------------------------------------------------------------
for i=findnotchtip:numberElementsY-1
    crackpath(i-findnotchtip+1,:)=[i*numberElementsX*4+1,0,0,...
       (i*2-1)*numberElementsX*2+1,2];
end;
```

Finally, in the last part, which is analysis of the beam in long-term, the strains due to creep and elastic deformation are measured in each time interval and if exceeds the critical strain, the node in front of the crack tip opens and crack length increases. When crack initiates, the cracking strain rate is also derived. The opening of the crack in time causes extra strain in front of the crack tip and along that.

```
% ---------------------------------------------------------------------
% ---------------------------------------------------------------------
% Beginning of analysis
% ---------------------------------------------------------------------
% ---------------------------------------------------------------------
cracklength=0; stepnumber=0; savingnr=0;
time=1;
CMOD='positive';
longterm=0;
force_crack=zeros(GDof,1);
strainCR=zeros(100,numberElementsY+1-findnotchtip);
strainEL=zeros(100,numberElementsY+1-findnotchtip);
strainRA=zeros(100,numberElementsY+1-findnotchtip);
str1(1)=0;str2(1)=0;str3(1)=0;str4(1)=0;str5(1)=0;str6(1)=0;

while cracklength<numberElementsY-findnotchtip-2 && time<=endtime
    stepnumber=stepnumber+1;

    force_crack=zeros(GDof,1);
    upwardlift=0;
    iteration=1;

    if cracklength>0 % changes in connection matrix/boundary condition
        fixedNodeX(fixedNodeX==crackpath(cracklength,1),:)=[];
        fixedNodeX(fixedNodeX==crackpath(cracklength,4),:)=[];
    end
    prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
    activeDof=setdiff([1:GDof]', [prescribedDof]);
    activeDof=setdiff(activeDof,[removednodes; removednodes+numberNodes]);

    while iteration==1 && time<=endtime

        savingnr=savingnr+1;
        force=force_primary;

        % ---------------------------------------------------------------
        % nodal forces
        % ---------------------------------------------------------------
        if cracklength>0;
            crackforce(cracklength,stepnumber)=0;
            cracksigma(cracklength,stepnumber)=0;
            fixedNodeX(fixedNodeX==crackpath(cracklength,1),:)=[];
            fixedNodeX(fixedNodeX==crackpath(cracklength,4),:)=[];
            prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
            activeDof=setdiff([1:GDof]', [prescribedDof]);
            activeDof=setdiff(activeDof,...
                [removednodes; removednodes+numberNodes]);

            % ---------------------------------------------------------
            % Iterative method to find crack nodal forces
            % ---------------------------------------------------------
            force_crack=zeros(GDof,1);

            for k=1:3 %stopiteration=1; while stopiteration==1
                U=zeros(GDof,1);
                U(activeDof)=stiffness(activeDof,activeDof)\ (force(activeDof)+force_crack(activeDof));
                displacements=U;

                % -----------------------------------------------------
                % Crack width, crack sigma, nodal forces
                % -----------------------------------------------------
                force_old=min(force_crack);
                for j=1:cracklength
                    cw_el=2*displacements(((j+findnotchtip-1)*2-1)* numberElementsX*2+1);
```

```
        crackwidth(j,stepnumber)=cw_el;
        if crackwidth(j,stepnumber)>0
            if crackwidth(j,stepnumber)>w0
                cracksigma(j,stepnumber)=0;
            else
                if crackwidth(j,stepnumber)<ws
                    cracksigma(j,stepnumber)= fctm*(1-crackwidth(j,stepnumber)/wc);
                else
                    cracksigma(j,stepnumber)= ft*(1-crackwidth(j,stepnumber)/w0);
                end
            end
        else
            cracksigma(j,stepnumber)=fctm;
        end
        if j==1
            crackforce(j,stepnumber)=cracksigma(j,stepnumber)*thickness*Ly/numberElementsY/2;
        else
            crackforce(j,stepnumber)= cracksigma(j,stepnumber)*thickness*Ly/numberElementsY;
        end

        force_crack(((j+findnotchtip-1)*2-1)* numberElementsX*2+1)=-crackforce(j,stepnumber)/2;
        force_crack(((j+findnotchtip-1)*2)* numberElementsX*2+1)=-crackforce(j,stepnumber)/2;
    end;

    if abs((min(force_crack)-force_old)/force_old)<0.1
        stopiteration=0;
    end

  end;

end %if

% ---------------------------------------------------------------
% Solution
% ---------------------------------------------------------------
force=force_primary+force_crack;
U=zeros(GDof,1);
U(activeDof)=stiffness(activeDof,activeDof)\force(activeDof);
displacements=U;
Reaction(savingnr)=-P;
midspandeflection(savingnr)=U(1+numberNodes);

% ---------------------------------------------------------------
% Stresses at nodes
% ---------------------------------------------------------------
stress=zeros(numberElements,size(elementNodes,2),3);
stressPoints=[-1 -1;1 -1;1 1;-1 1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  nn=length(indice);
  for q=1:size(gaussWeights,1)
    pt=gaussLocations(q,:);
    wt=gaussWeights(q);
    xi=pt(1);
    eta=pt(2);
    % shape functions and derivatives
    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
        (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives=...
        1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
        1+eta,      1+xi;-(1+eta),   1-xi];
    % Jacobian matrix, inverse of Jacobian,
    % derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);
    %  B matrix
    B=zeros(3,2*nn);
    B(1,1:nn)       = XYderivatives(:,1)';
```

```
            B(2,nn+1:2*nn)  = XYderivatives(:,2)';
            B(3,1:nn)       = XYderivatives(:,2)';
            B(3,nn+1:2*nn)  = XYderivatives(:,1)';

            % element deformation
            strain=B*displacements(elementDof);
            stress(e,q,:)=C*strain;
        end
    end
    % Principal Stress
    stress_principal1=(stress(:,:,1)+stress(:,:,2))/2+...
        ((stress(:,:,1)-stress(:,:,2)).^2/4+stress(:,:,3).^2).^0.5;

    % ------------------------------------------------------------
    % Stresses in nodes along the tip (the mean stress between 2 nodes)
    % ------------------------------------------------------------
    for i=numberElementsY-1:-1:findnotchtip
        nodalstress(i-findnotchtip+1,cracklength+2)=...
            (stress_principal1(i*numberElementsX+1,1)+stress_principal1((i-1)*numberElementsX+1,4))/2;
        nodalstress(i-findnotchtip+1,1)=i;
    end;
    nodalstress((numberElementsY-findnotchtip)+1,cracklength+2)=...
        stress_principal1((numberElementsY-1)*numberElementsX+1,4);
    nodalstress((numberElementsY-findnotchtip)+1,1)=numberElementsY;

    stress_max = (nodalstress(cracklength+1,cracklength+2)+nodalstress(cracklength+2,cracklength+2))/2;

    % ------------------------------------------------------------
    % finding max stress and check with fctm
    % ------------------------------------------------------------
    str_el=loadratio/E0*0.35;
    for checkstress=1:numberElementsY-findnotchtip
        if (nodalstress(checkstress,size(nodalstress',1))+nodalstress(checkstress+1,size(nodalstress',1)))/2>fctm
            strainEL((time-1)/time_interval+1,checkstress)=critical_strain;
        else
            if (nodalstress(checkstress,size(nodalstress',1))+nodalstress(checkstress+1,size(nodalstress',1)))/2>fctm*0.9
                strainEL((time-1)/time_interval+1,checkstress)=0.9*fctm/E+ (critical_strain-0.9*fctm/E)/(0.1*fctm)*...
                    ((nodalstress(checkstress,size(nodalstress',1))+nodalstress(checkstress+1,size(nodalstress',1)))/2-0.9*fctm);
            else
                strainEL((time-
1)/time_interval+1,checkstress)=(nodalstress(checkstress,size(nodalstress',1))+nodalstress(checkstress+1,size(nodalstress',1)))
/2/E;
            end
        end
        strainEL((time-1)/time_interval+1,checkstress)=strainEL((time-1)/time_interval+1,checkstress)*str_el/strainEL((time-
1)/time_interval+1,1);
    end;

    straintotal((time-1)/time_interval+1,:)=(strainRA((time-1)/time_interval+1,:)+strainCR((time-
1)/time_interval+1,:))*longterm+strainEL((time-1)/time_interval+1,:);

    if max(straintotal((time-1)/time_interval+1,:))>=critical_strain
        cracklength=cracklength+1;
        iteration=0;
        Reaction(savingnr)=-P*fctm/stress_max;
        midspandeflection(savingnr)=(U(1+numberNodes)-upwardlift)*fctm/stress_max;
        Results2(cracklength+1,:)=[midspandeflection(savingnr),Reaction(savingnr)];
        longterm=0;

    else
        longterm=1;
        time=time+time_interval;
        timet=time/24/3600;
        if strcmpi(Code,'EC2')==1
            % Creep according to EC2
            betaC=(timet/(betaH+timet))^0.3;
            creep=betaC*phi0;
        else
            % Creep according to Model B3
            Zt=(t0^(0-m))*log(1+(timet)^n);
            Qt=Qf*(1+(Qf/Zt)^rt)^(0-1/rt);
```

```
            C0=q2*Qt+q3*log(1+(timet)^n)+q4*log((timet+t0)/t0);
            St=tanh(((timet+t0-ts)/tao_sh)^0.5);
            Htt=1-(1-RH)*St;
            Cd=q5*(exp(-8*Htt)-exp(-8*Ht))^0.5;
            Jt=q1+C0+Cd;
            creep=((Jt*E)/1000000-1);
        end


        for k=2:(time-1)/time_interval+1
            delt_str1=str1(k-1)*(exp(-time_interval/(eta1/E1))-1)+(time_interval/eta1)*exp(-
time_interval/(2*eta1/E1))*loadratio;
            str1(k)=str1(k-1)+delt_str1;
            delt_str2=str2(k-1)*(exp(-time_interval/(eta2/E2))-1)+(time_interval/eta2)*exp(-
time_interval/(2*eta2/E2))*loadratio;
            str2(k)=str2(k-1)+delt_str2;
            delt_str3=str3(k-1)*(exp(-time_interval/(eta3/E3))-1)+(time_interval/eta3)*exp(-
time_interval/(2*eta3/E3))*loadratio;
            str3(k)=str3(k-1)+delt_str3;
            delt_str4=str4(k-1)*(exp(-time_interval/(eta4/E4))-1)+(time_interval/eta4)*exp(-
time_interval/(2*eta4/E4))*loadratio;
            str4(k)=str4(k-1)+delt_str4;
            delt_str5=str5(k-1)*(exp(-time_interval/(eta5/E5))-1)+(time_interval/eta5)*exp(-
time_interval/(2*eta5/E5))*loadratio;
            str5(k)=str5(k-1)+delt_str5;
            delt_str6=str6(k-1)*(exp(-time_interval/(eta6/E6))-1)+(time_interval/eta6)*exp(-
time_interval/(2*eta6/E6))*loadratio;
            str6(k)=str6(k-1)+delt_str6;
        end
        str_visc=k/eta0*loadratio;
        straincrackrate=str_visc+str1(k)+str2(k)+str3(k)+str4(k)+str5(k)+str6(k);

        strainCR((time-1)/time_interval+1,:)=nodalstress(:,size(nodalstress',1))'*creep/E;
        for k=1:cracklength
            strainRA((time-1)/time_interval+1,k)=straincrackrate*crackwidth(k,stepnumber)/crackwidth(1,stepnumber);
        end

    end

    Results(savingnr,:)=[midspandeflection(savingnr),Reaction(savingnr)];

  end %interation

end %stepnumber

plot(straintotal);
```

2.6. *Modelling of long-term shear test on a notched beam in Matlab*

A concrete beam as illustrated in Fig. 19, is loaded under sustained loading, until failure. The principal idea was the same as the previous modelling in sections 0 and 2.5. In this model, there are only two items included; Fracture Energy Mode II and the free crack path, which means the crack is allowed to go vertically or horizontally, perpendicular to the maximum principal stress.

The FE Code of the long-term shear test in Matlab is given in Appendix A.

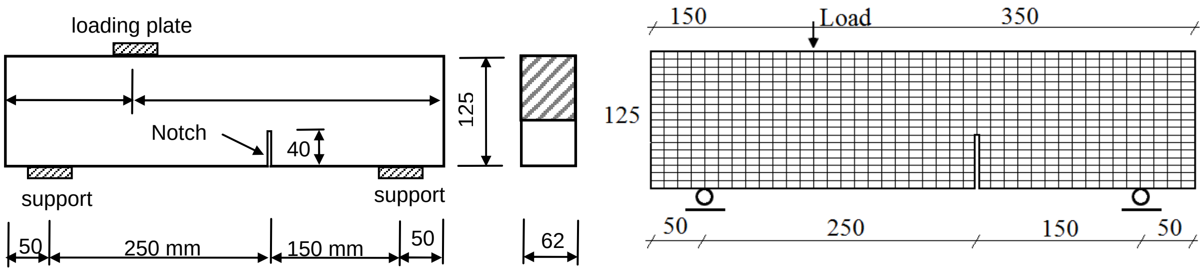


Fig. 19: Left: Geometry of the specimen with shear notch. Right: FE model of the beam with notch

2.6.1. Effect of compliance function

The estimated fracture time in this model depends on the compliance function that is used. Considering that the tensile strength in American codes (e.g. ACI) is larger than in the European code (EC2), it gives a higher capacity of the specimen for short-term loading. However, the compliance function of Model B3 [Bazant] gives a slower rate of crack propagation in comparison with the EC2 creep coefficient.

2.6.2. Effect of cracking strain rate

Excluding the cracking strain rate from the model, gives a very slow propagation of the crack. The specimen under 80% load ratio, would never fail if the cracking strain rate were neglected. The effect of cracking strain rate is well-presented by Di Luzio [16].

2.6.3. Effect of initial strain and critical strain

The initial strain is very important in this model as a higher initial strain gives a faster failure of the beam. The initial strains and the expression that derived from the experimental tests are used in the FE model.

The value of critical strain is chosen to be 0,00015 in this model. However, if the magnitude of the critical strain is changed to a lower value, e.g. 0.00014, a faster fracture is expected, but then the load ratio with the same load value would be also higher as the ultimate capacity is decreased. Hence, the effect of critical strain on time of failure is insignificant.

2.6.4. Effect of fracture energy mode I, $^{I}G_F$

The magnitude of the fracture energy mode I has an important influence on the resistant stress in FPZ, accordingly the stress in front of the crack depends on $^{I}G_F$; a lower fracture energy leads to a higher stress at the crack tip and faster development of the crack.

2.6.5. Effect of fracture energy mode II, $^{II}G_F$

The effect of fracture energy mode II on short-term shear failure is presented in Fig. 20. This result is obtained based on $^{I}G_F = {}^{II}G_F$. Obviously, when considering only fracture energy mode I, the fracture mode is not shear failure and the beam fails in flexural mode, while considering the mixed mode, the crack propagation is more likely to be the shear crack. However, test results show that in this type loading on plain concrete beam, the failure is very similar to Fig. 20(Right).

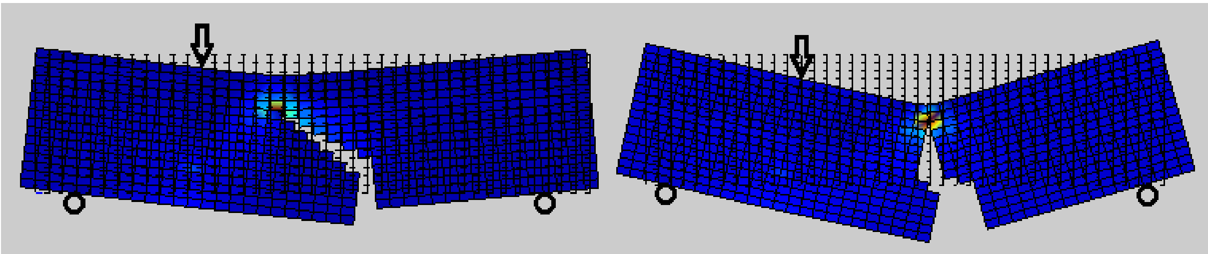


Fig. 20: Shear crack path in case of fracture energy mode I+II (Left) and only mode I (Right)

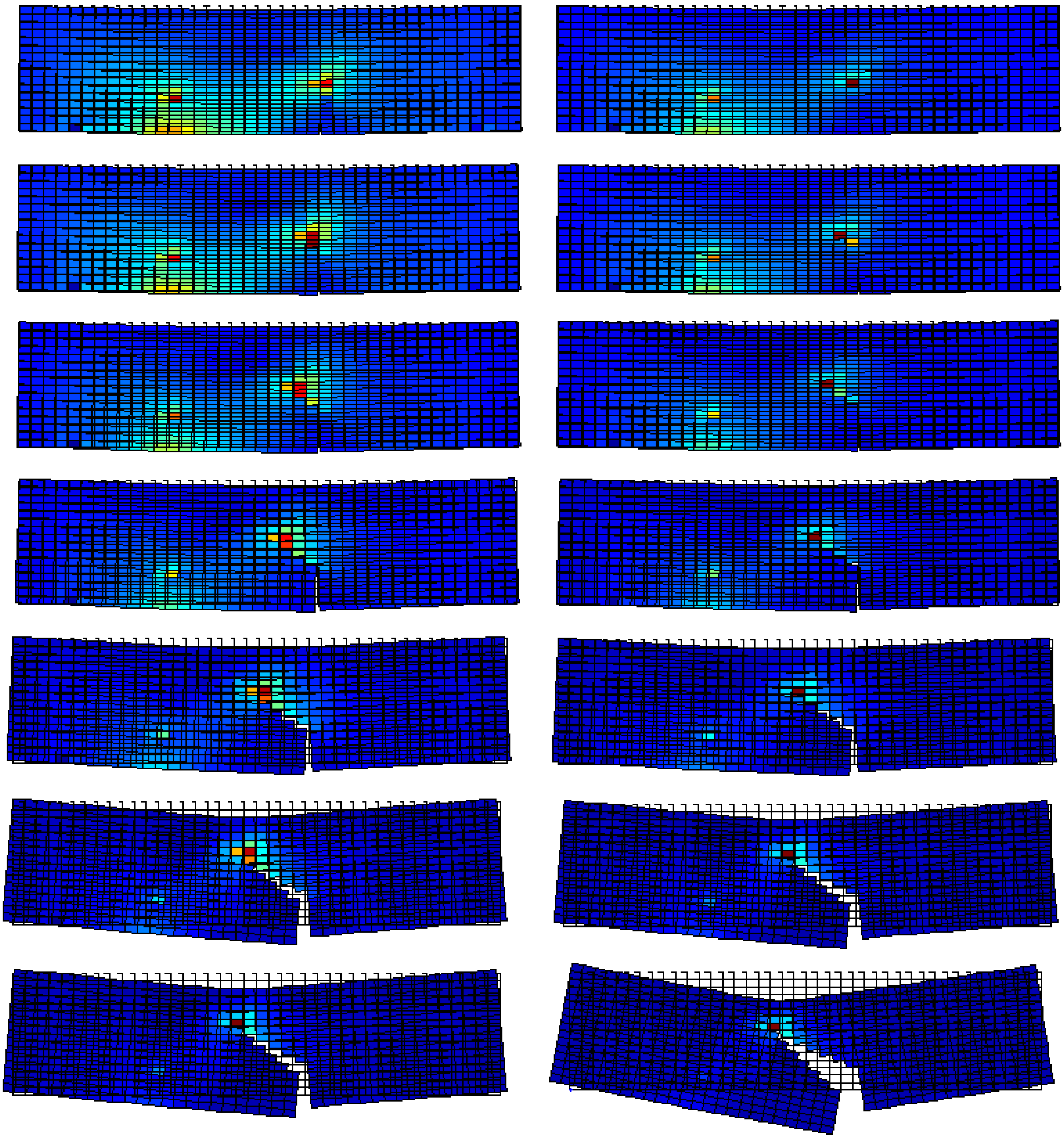


Fig. 21: Different steps of a shear crack growth. Fracture energy mode II is activated.

3. Modelling of shear crack in reinforced concrete

In order to explain the results of long-term tests on full-scale reinforced concrete beams [33], time-dependent behaviour of a single shear crack in reinforced concrete beam is going to be modelled in this chapter by means of the analytical model, which is described in Chapter 2.

3.1. Model definition

A concrete beam that is illustrated in Fig. 22 is modelled as described in Chapter 2. The material properties are given in Table 1. In order to simplify the model, a shear crack is implemented into the model and the corresponding elements were discrete as illustrated in Fig. 23.

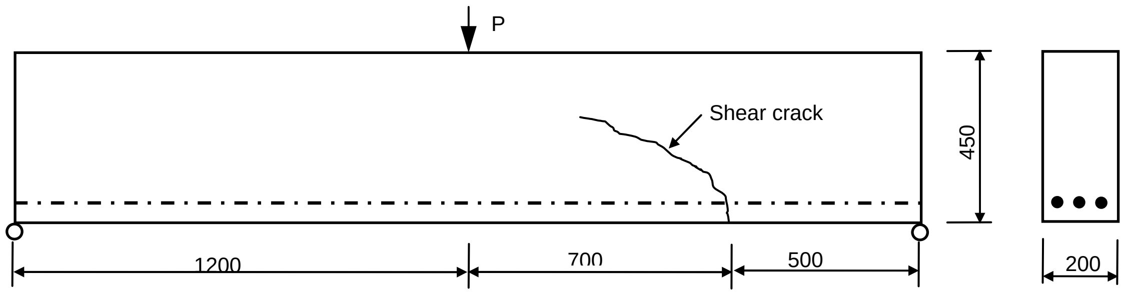


Fig. 22: Geometry of the specimen with shear crack (all dimensions are in mm)

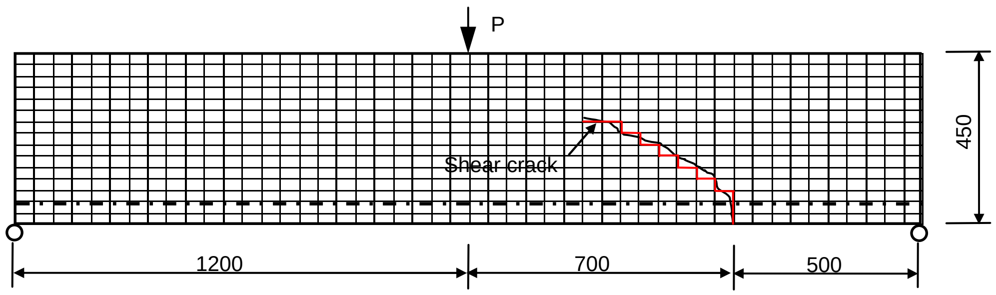


Fig. 23: Meshing of the specimen with shear crack. The red line represents the discretion of the elements

In order to simplify the model, the reinforcing bars were not modelled in the FE analysis, but the reinforcement forces and the dowel forces across the crack are applied into the model and to do that, a fracture model is used.

The formulation of the model is based on the fundamental relation of linear elastic fracture mechanics, $\delta G = \frac{1}{2} \delta W_{ext}$. The mechanism producing external work is the rotation, under constant load, about the tip of the diagonal crack. In order to calculate the energy release, the rotational stiffness of the beam need to be determined. To that aim, the bulk of uncracked concrete and the embedded reinforcement are considered to behave as a rigid body except for the concrete connection subjected to compression. The rotational stiffness depends on the axial and the dowel stiffness of the longitudinal reinforcement, itself depending on the extent of splitting releasing the reinforcing bar [19], [28]. The stiffness is worked out considering the free body diagram, see Fig. 24.

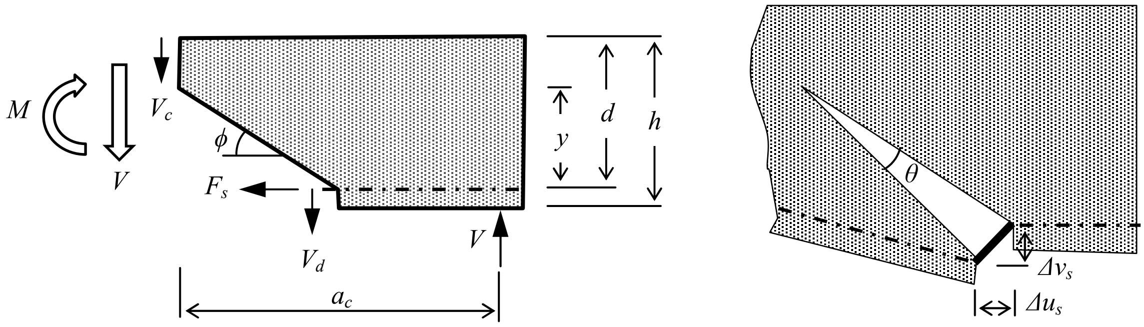


Fig. 24: Free body diagram of a shear crack in reinforced concrete

The axial and shear (dowel) force in the steel bar crossing the diagonal crack can be linked to the angle of rotation θ using the elastic properties of the bar and the geometry of the deformation mechanism.

$$F_S = \frac{E_S A_S}{\delta_S} \Delta u_S = \frac{E_S A_S}{\delta_S} y\theta \tag{38}$$

$$F_S = \frac{G_S \Sigma_S}{\delta_S} \Delta v_S = \frac{9}{26} \frac{E_S A_S}{\delta_S} \frac{y\theta}{\tan\phi} \tag{39}$$

where G_S is the shear modulus of steel, Σ_S is the reduced cross section of the bear, δ_S is unbounded length of reinforcement and can be calculated as follow:

$$\Sigma_S = 0.9 A_S \tag{40}$$

$$G_S = \frac{E_S}{2(1+\nu_S)} = \frac{1}{2.6} E_S \tag{41}$$

$$\delta_S = 0.7 y \tag{42}$$

3.2. Results of short-term analysis

According to the results of FE analysis and using the EC2 criteria, an ultimate capacity of 184,64 kN was acquired. At this load level, the principal stress at the crack tip exceeds f_{ctm} and the crack propagates to the next element joint and does not stop until fracture of the beam. The first four steps of crack propagation are shown in Fig. 25.

A bilinear stress-strain relation is assumed for reinforcing steel with a modulus of elasticity of E_S = 210 GPa and a yield strength of f_y = 420 MPa. The maximum load carried by the reinforcement is then assumed to be $A_S \cdot f_y$.

Table 3: Crack propagation at ultimate capacity of the beam

	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
Midspan deflection [mm]	-0,99	-1,18	-1,08			
Δu_s [mm]	0,27	0,28	0,28			
Δv_s [mm]	0,20	0,16	0,17			
Stress at crack tip [MPa]	3,2	6,63	3,4			
Rebar Tensile force [kN]	304,9	302,2	302,2			
Dowel force [kN]	95,9	103,4	106,6			

* Reinforcement Yields

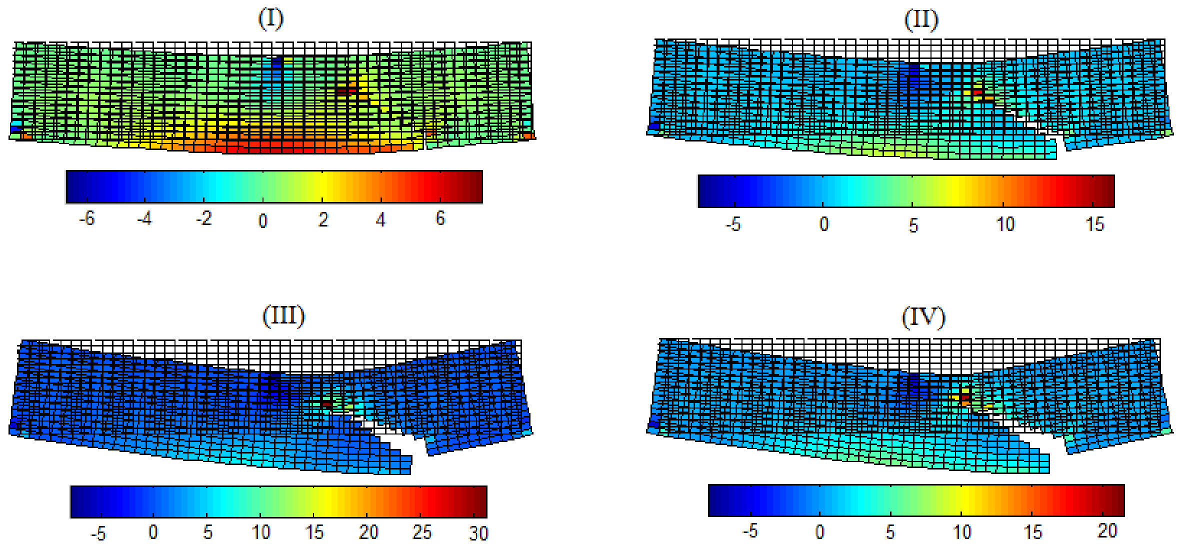


Fig. 25: Crack propagation in the reinforced beam with shear crack.

3.3. Results of long-term analysis

With a known value the ultimate capacity of 184,64 kN whereat the crack propagates and opens until yielding of the reinforcement, the behaviour of the beam under different load ratios can be determined. As shown in Table 4, the calculated principal stress at the crack tip reduces in time. During long-term loading, because the stress at the crack tip due to the applied load is less than concrete tensile strength, remains constant. However, during creep loading, the modulus of elasticity is supposed to reduce in time, thus Table 5 is showing the test results with reduced E modulus.

Table 4: Results of loading under 95% of the ultimate capacity when the modulus of elasticity remains constant

Time [sec]	Stress at tip [MPa]	Δu_s [mm]	Rebar Tensile force [kN]	Midspan deflection [mm]
0	1,76	0,265	289,21	-0,88
100	1,75	0,264	289,23	-0,88
1000	1,75	0,264	289,26	-0,88
10000	1,75	0,264	289,25	-0,88
100000	1,75	0,264	289,28	-0,88
1000000	1,76	0,264	289,21	-0,88
10000000	1,76	0,264	289,20	-0,88
100000000	1,76	0,264	289,20	-0,88

Table 5: Results of loading under 95% of the ultimate capacity with a reduced modulus of elasticity $E = E_0/(1+\varphi)$

Time [sec]	Stress at tip [MPa]	Δu_s [mm]	Rebar Tensile force [kN]	Midspan deflection [mm]
0	1,76	0,265	289,21	-0,88
100	1,75	0,25	272,37	-1,02
1000	1,75	0,24	264,78	-1,03
10000	1,70	0,25	267,68	-1,03
100000	-1,21	0,27	296,49	-1,03
1000000	-1,29	0,28	302,86	-1,35
10000000	-0,95	0,28	303,01	-1,88
100000000	-0.30	0.28	303,49	-2,68

Table 6: Results of loading under 95% of the ultimate capacity with a modified modulus of elasticity
$E = \beta_{cc}(t)E_0/(1+\varphi)$, $\beta_{cc}(t) = \exp\{s[1-(28/t)^{0,5}]\}$

Time [sec]	Stress at tip [MPa]	Δu_s [mm]	Rebar Tensile force [kN]	Midspan deflection [mm]
0	1,76	0,265	289,21	-0,88
100	2,70	0,257	287,71	-0,88
1000	2,24	0,262	281,43	-0,90
10000	2,14	0,262	281,63	-0,87
100000	1,73	0.263	285,14	-1,04
1000000	1,44	0,264	287,89	-1,25
10000000	0,99	0,265	282,13	-1,30
100000000	0,86	0,272	296,11	-2,28

4. References

[1] Barpi F. and Valente S. (2004), A fractional order rate approach for modeling concrete structures subjected to creep and fracture, *International Journal of Solids and Structures*, Vol. 41, pp. 2607–2621.

[2] Barenblatt G. (1962), The mathematical theory of equilibrium crack in the brittle fracture. *Advanced in Applied Mechanics*, Vol. 7, pp. 55–125

[3] Bažant , Z.P. and Oh, B.H. (1983). Crack band theory for fracture of concrete. *RILEM, Materials and Structures* 16(93), 155–177.

[4] Bažant, Z.P., Kim, Jenn-Keun, and Pfeiffer, P. A. (1986). "Nonlinear fracture properties from size effect tests." J. of Structural Engrg., ASCE 112, ST2, 289–307.

[5] Bažant, Z.P. and Planas, J., "Fracture and Size Effect in Concrete and Other Quasi-brittle Materials", CRC Press, New York, 1998.

[6] Bažant , Z. and Baweja, S. (2000), Creep and shrinkage prediction model for analysis and design of concrete structures: Model B3, in Adam Neville Symposium: Creep and Shrinkage – *Structural Design Effects, ACI SP-194,* A. Al-Manaseer Ed., Am. Concrete Institute, Farmington Hills, Michigan, 1–83.

[7] Bažant, Z.P. and Chern, J.C. (1985), Strain-softening with creep and exponential algorithm. *J. of Engrg. Mechanics*, ASCE 111, pp.391–415.

[8] Bažant, Z.P. and Li, Y.N., (1997), Cohesive crack with rate-dependent opening and viscoelasticity: I. mathematical model and scaling. *Int. J. of Fracture*, 86 (3), pp.247–265.

[9] Bažant, Z.P., and Prat, P.C., 1988, Effect of temperature and humidity on fracture energy of concrete. ACI Materials Jour. 84, July 1988, pp.262–271.

[10] Broek, D. (1986), "Elementary Engineering Fracture Mechanics", Martinus Nijhoff Publishers, Dordrecht, 374–380.

[11] Carol, I., Prat, P., López, M., (1997) Normal/shear cracking model: Application to discrete crack analysis, *J. Eng. Mech.* ASCE 123, 765– 773.

[12] Chaimoon, K., Attard, M.M. and Tin-Loi F. (2008), Crack Propagation Due To Time-Dependent Creep in Quasi-Brittle Materials under Sustained Loading. *Comput. Methods Appl. Mech. Engrg*. 197, pp.1938-1952

[13] Collop, A.C, Scarpas, A., Kasbergen, C. and de Bondt, A. (2003), Development and FE implementation of a stress dependent Elasto-Visco-Plastic constitutive model with Damage for asphalt, *82nd TRB Annual meeting*, Washington DC.

[14] Comité Euro-International du Béton: CEB-FIP Model Code 1990 (CEB_FIP MC90). Bulletin D'Information No. 213/214, Lausanne, May 1993

[15] Dugdale D.S. (1960), Yielding of steel sheets containing slits. *J. of Mech. Phys. Solids*, Vol. 8, pp. 100-104

[16] Di Luzio, G. (2009), Numerical Model for Time-Dependent Fracturing of Concrete, *Journal of Engineering Mechanics, ASCE*, Vol 135, No. 7, July 1, pp. 632-640.

[17] Eurocode 2 - Design of concrete structure. EN 1992-1-1 Part 1-1: General rules and rules for buildings

[18] Gálvez, J.C., Červenka, J., Cendón, D.A. and Saouma, V. (2002), A discrete crack approach to normal/shear cracking of concrete, *Cement and Concrete Research*, Vol. 32, 1567–1585.

[19] Gastebled, O.J, and May, I.M. (2001), Fracture Mechanics Model Applied to Shear Failure of Reinforced Concrete Beams Without Stirrups, ACI Structural Journal, Vol. 98, Issue 2, pp. 184-190

[20] Hillerborg, A., Modéer, M., Petersson, P.E., (1976), Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements, *Cem and Concrete Research* 6(6), pp. 773-782.

[21] Hillerborg, A. (1985). The theoretical basis of a method to determine the fracture energy G_F of concrete. *Mater. Struct.* , 18 (4), pp. 291-296.

[22] Hilsdorf, H.K. and Brameshuber, W. (1991), Code-type formulation offracture mechanics concepts for concrete, *Intl. Jl. of Fracture*, Vol.51, No.1, pp 61-72.

[23] Jenq, Y.S. and Shah S.P. (1985). Two parameter fracture model for concrete. *Journal of Engineering Mechanics*, ASCE, 111(10), 1227–1241.

[24] Karihaloo, B.L., Nallathambi, P. (1990), Effective crack model for the determination of fracture toughness of concrete. *Engineering Fracture Mechanics*, 35, pp. 637-645.

[25] MacGregor, J. G., Rizkalla, S. H., and Simmonds, S. H. (1980), "Cracking of Reinforced and Prestressed Concrete Wall Segments", Structural Engineering Report. No. 82, Department of Civil Engineering, University of Alberta, Edmonton

[26] Petersson, P.E., (1981), Crack Growth and Development of Fracture Zones in Plain Concrete and Similar Materials, Division of Building Materials, PhD Thesis, Lund Institute of Technology, Sweden, Rep. TVBM-1006, Lund, October 1981

[27] Petersson, P. E. (1981), Fracture energy of concrete: method of determination. *Cement and Concrete Research*, Vol. 10, No. 1. pp. 78-89. (Ref. 7 of original paper.)

[28] Reineck, K.H. (1991), Ultimate Shear Force of Structural Concrete Members Without Transverse Reinforcement Derived From a Mechanical Model, *ACI Structural Journal*, Vol. 88, Issue 5, pp. 592-602

[29] RILEM Technical Committee 50-FMC (1985). Determination of the fracture energy of mortar and concrete by means of three-point bend tests of notched beams, proposed RILEM draft recommendations. RILEM, *Materials and Structures* 18(106), pp. 285-296

[30] RILEM Technical Committee 89-FMT (1990a). Determination of fracture parameters .KsIc and CTODc/ of plain concrete using three-point bend tests, *proposed RILEM draft recommendations*. Ibid. 23(138), 457–460.

[31] RILEM Technical Committee 89-FMT (1990b). Size-effect method for determining fracture energy and process zone size of concrete, *proposed RILEM draft recommendations*. Ibid. 23(138), 461–465.

[32] Sarkhosh, R., den Uijl, J. A., Braam, C. R. and Walraven J. C. (2013), Experimental Investigation on Time-Dependent Flexural Crack Growth; Experimental tests and Analytical Model, Stevinreport 25.5.13-01, Delft University of Technology, the Netherlands.

[33] Sarkhosh, R., den Uijl, J. A., Braam, C. R. and Walraven J. C. (2012), Shear Capacity of Concrete Beams without Shear Reinforcement under Sustained Loads; Experimental tests, Stevinreport 25.5-12.07, Delft University of Technology, the Netherlands.

[34] Swartz, S.E. and Refai, T.M.E. (1987). Influence of Size on Opening Mode Fracture Parameters for Precracked Concrete Beams in Bending. Proceedings of SEM-RILEM International Conference on Fracture of Concrete and Rock (Edited by S.P. Shah and S.E. Swartz), Houston, Texas, 242–254.

[35] Tang, T., Ouyang, C. and Shah, S.P. (1996), A simple method for determining material fracture parameters from peak loads, ACI Mater. J. 93 (2), pp. 147-157.

[36] Tada, H., Paris, P.C. and Irwin, G.R. (1985). "The Stress Analysis of Cracks Handbook". Paris Productions Incorporated, St. Louis, Missouri, USA.

[37] Wittmann, F.H., Mihashi, H. and Nomura, N. (1990). Size effect on fracture energy of concrete. Engineering Fracture Mechanics 35(1–3), 107–115

[38] Xu, S. and Reinhardt, H. W. (1999), Determination of double-Determination of double-K criterion for crack propagation in quasi-brittle fracture Part I: experimental investigation of crack propagation, *International Journal of Fracture*, Volume 98, Issue 2, pp. 111-149.

[39] Xu S. and Zhao G., (1989), The Determination of the Fracture Toughness and the Fracture Energy of Concrete, *Fracture Toughness and Fracture Energy-Test Methods for Concrete and Rock, E*dited by H.Mihashi *et al.*, A.A.Balkema Publishers, The Netherlands, pp. 145-154.

[40] Zhao, G., Jiao, H., and Xu, S. (1991), Study of Fracture Toughness and Fracture Energy by Means of Wedge Splitting Test Specimen. *Proceeding of the Third International Symposium on Brittle Matrix Composites*. Warsaw, Poland: Elsevier Science Pub Ltd. pp. 62-71.

[41] Zhou, F.P., Hillerborg, A. (1992), Time-dependent fracture of concrete: testing and modelling. In: Bažant , Z.P. (Ed.), *Fracture Mechanics of Concrete Structures*. Elsevier Applied Science, The Netherlands, pp. 906–911.

[42] Zhou, F.P. (1992), Time-dependent crack growth and fracture in concrete. PhD thesis, Report TVBM-1011, Lund University of Technology, Sweden.

Appendix A: FE Model of a shear crack in plain concrete

```
%..........................
% longtermshear.m
% ---------------------------------
% clear memory
clear all;colordef white;clf
crackpropagation=14;
loadratio=0.6;
endtime=120000; %(seconds of test)
time_interval=endtime/100;
showstressgraph=1; %=0 graph off    =1 graph on
Code='EC2'; % EC2 or B3

% ---------------------------------
% Given material property
% ---------------------------------
fcube=52;
poisson = 0.20;
thickness=62;
Lx=500; %Length of the beam
Ly=125; %Height
rho=1;
critical_strain=0.00015;

% ---------------------------------
% Calculated material property
% ---------------------------------
fcm = fcube*0.785;
E = 22000*(fcm/10)^0.3;
if strcmpi(Code,'EC2')==1
   fctm = 0.3*(fcm-8)^(2/3); % Eurocode2
else
   fctm=(145.0377*fcm)^0.5*6.7/145.0377; % fctm based on ACI
end

% ---------------------------------
% Fracture Energy Mode I
% ---------------------------------
alphaF=7; alphaD=6;
GF=alphaD/1000*fcm^0.7;
w0=GF*alphaF/fctm;
ws=2*GF/fctm-0.15*w0;
sigmaS=0.15*fctm;
wc=ws/0.85;
ft=w0*sigmaS/(w0-ws);

% ---------------------------------
% Fracture Energy Mode II
% ---------------------------------
fccm=fctm; %cohesion
GF2=alphaD/1000*fcm^0.7/2;
w02=GF2*alphaF/fccm;
ws2=2*GF2/fccm-0.15*w02;
sigmaS2=0.15*fccm;
wc2=ws2/0.85;
ft2=w02*sigmaS2/(w02-ws2);

% ---------------------------------
% Creep function based on EC2 or B3 Model
% ---------------------------------
RH = 0.5; %Relative Humidity
if strcmpi(Code,'EC2')==1
   % Creep according to Eurocode
   alpha1=(35/fcm)^0.7; alpha2=(35/fcm)^0.2; alpha3=(35/fcm)^0.5;
   betafcm=16.8/fcm^0.5;
```

```
    h0=thickness*Ly/(thickness+Ly);
    ts=7; %curing time in days
    t0=28; % age in days at loading
    if fcm>35
        phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1)*alpha2;
        betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250*alpha3,1500*alpha3);
    else
        phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1);
        betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250,1500);
    end
    betat0=1/(0.1+t0^0.2);
    phi0=phiRH*betafcm*betat0;
else
    % Creep according to Model B3
    ts=7; %curing time in days
    t0=28; % age in days at loading
    Qf=(0.086*t0^(2/9)+1.21*t0^(4/9))^(-1);
    m=0.5; n= 0.1;
    rt=1.7*t0^0.12+8;
    cement=330; water=189; aggregate=1803;
    q1=0.6*1000000/E;
    q2=185.4*cement^0.5*fcm^(-0.9);
    q3=0.29*(water/cement)^4*q2;
    q4=20.3*(aggregate/cement)^(-0.7);

    ks=1.25;%(slab=1)(inf cyl=1.15)(inf sq prsm=1.25)(sphr=1.30)(cube=1.55)
    kt=8.5*ts^(-0.08)*fcm^(-0.25)/100;
    h0=thickness*Ly/(thickness+Ly);
    tao_sh=kt*(ks*h0)^2;
    alpha1=1.1; %(CEMENT type I=1.0)(CEM Type II=0.85)(CEM Type III=1.1)
    alpha2=1.0; %(Steam curing=0.75)(Normal Cur=1.2)(Curing in RH100%=1.0)
    Ht=1-(1-RH)*(tanh(((t0-ts)/tao_sh)^0.5));
    epsilon_infinit=-alpha1*alpha2*(1.9/100*(water^2.1)*(fcm^(-0.28))+270);
    q5=7.57*100000/fcm*(abs(epsilon_infinit)^(-0.6));
end

% ---------------------------------
% Support and boundary conditions
% ---------------------------------
supportX1=50;
supportX2=450;
loadingplateX=150;

% ---------------------------------
% matrix C
% ---------------------------------
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% ---------------------------------
% load
% ---------------------------------
Pu= -5500;
P = Pu*loadratio;

% ---------------------------------
% Parameters of crack growth rate
% ---------------------------------
tao1 = 5;  tao2 = 25; tao3 = 200; tao4 = 1000; tao5 = 10000; tao6 = 100000;
A1 = 13600-1000* log(tao1) ; B1 = 0.53*log(tao1)-8.75; E1 = A1*loadratio^B1; eta1 = tao1 * E1;
A2 = 13600-1000* log(tao2) ; B2 = 0.53*log(tao2)-8.75; E2 = A2*loadratio^B2; eta2 = tao2 * E2;
A3 = 13600-1000* log(tao3) ; B3 = 0.53*log(tao3)-8.75; E3 = A3*loadratio^B3; eta3 = tao3 * E3;
A4 = 13600-1000* log(tao4) ; B4 = 0.53*log(tao4)-8.75; E4 = A4*loadratio^B4; eta4 = tao4 * E4;
A5 = 13600-1000* log(tao5) ; B5 = 0.53*log(tao5)-8.75; E5 = A5*loadratio^B5; eta5 = tao5 * E5;
A6 = 13600-1000* log(tao6) ; B6 = 0.53*log(tao6)-8.75; E6 = A6*loadratio^B6; eta6 = tao6 * E6;
E0 = 1646.1*loadratio^(-2.516); eta0 = 90000000*loadratio^(-7.4);

% ---------------------------------
% Mesh generation
% ---------------------------------
```

```
numberElementsX=40;
numberElementsY=16;
numberElements=numberElementsX*numberElementsY;

% ---------------------------------------------------------------------
% node coordinates
% ---------------------------------------------------------------------
nodeCoordinates=zeros((numberElementsX*2)*(numberElementsY*2),2);
for j=1:numberElementsY
    for i=1:numberElementsX
        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),1)= (i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),1)= i*Lx/numberElementsX;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),1)=(i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),1)= i*Lx/numberElementsX;

        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),2)= (j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),2)=(j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),2)= j*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),2)= j*Ly/numberElementsY;
    end;
end;

% ---------------------------------------------------------------------
% element nodes
% ---------------------------------------------------------------------
elementNodes_primary=zeros(numberElements,4);
for j=0:numberElementsY-1
    for i=1:numberElementsX
        elementNodes_primary(i+j*(numberElementsX),1)= i*2-1+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),2)= i*2+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),3)= i*2+numberElementsX*(4*j+2);
        elementNodes_primary(i+j*(numberElementsX),4)= i*2-1+numberElementsX*(4*j+2);
    end;
end;

xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
numberNodes=size(xx,1);

% ---------------------------------------------------------------------
% Connection matrix
% ---------------------------------------------------------------------
connection=zeros(numberNodes);
for j=1:numberNodes
    for i=j+1:numberNodes
        if nodeCoordinates(i,:)==nodeCoordinates(j,:)
            connection(i,j)=1;
            connection(j,i)=1;
        end
    end;
end;
connectionTri=triu(connection);

% ---------------------------------------------------------------------
% Shear notch
% ---------------------------------------------------------------------
for i=1:numberElementsX*22
    if nodeCoordinates(i,1)==300 && nodeCoordinates(i+1,1)==300
        connection(i,i+1)=0;
        connection(i+1,i)=0;
        connection(i,i+numberElementsX*2+1)=0;
        connection(i+1,i+numberElementsX*2-1)=0;
    end
end;

% ---------------------------------------------------------------------
% Modification element nodes
% ---------------------------------------------------------------------
elementNodes=elementNodes_primary;
```

```
removednodesnr=0;
for j=1:numberElements;
    for i=1:4
        findconnection=find(connection(:,elementNodes(j,i))==1);
        if size(findconnection)>=1
            if min(findconnection)<elementNodes(j,i)
                removednodesnr=removednodesnr+1;
                removednodes(removednodesnr,1)=elementNodes(j,i);
            end
            elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
        end
    end;
end;

drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
scaleFactor=100;

% -----------------------------------------------------------------
% GDof: global number of degrees of freedom
% -----------------------------------------------------------------
GDof=2*numberNodes;

% -----------------------------------------------------------------
% boundary conditions
% -----------------------------------------------------------------
fixedNodeX=find(nodeCoordinates(:,1)==loadingplateX & nodeCoordinates(:,2)==Ly);  % fixed in XX

fixedNodeY=[find(nodeCoordinates(:,1)==supportX1 & nodeCoordinates(:,2)==0);find(nodeCoordinates(:,1)==supportX2 &
nodeCoordinates(:,2)==0)];  % fixed in YY

% -----------------------------------------------------------------
% force vector (point load applied at xx=0, yy=Ly)
% -----------------------------------------------------------------
force_primary=zeros(GDof,1);
leftBord=find(nodeCoordinates(:,1)==loadingplateX);
force_primary(leftBord(end-1)+numberNodes)=P;
force=force_primary;

% -----------------------------------------------------------------
% Stiffness Matrix
% -----------------------------------------------------------------
stiffness=zeros(GDof);
        gaussLocations=...
          [ -0.577350269189626 -0.577350269189626;
             0.577350269189626 -0.577350269189626;
             0.577350269189626  0.577350269189626;
            -0.577350269189626  0.577350269189626];
        gaussWeights=[ 1;1;1;1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  ndof=length(indice);

  % cycle for Gauss point
   for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

% shape functions and derivatives
    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
            (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives=1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);  1+eta,      1+xi;-(1+eta),   1-xi];

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=  Jacobian(nodeCoordinates(indice,:),naturalDerivatives);
```

```
%  B matrix
   B=zeros(3,2*ndof);
   B(1,1:ndof)       = XYderivatives(:,1)';
   B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
   B(3,1:ndof)       = XYderivatives(:,2)';
   B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

% stiffness matrix
   stiffness(elementDof,elementDof)=...
      stiffness(elementDof,elementDof)+...
      B'*C*thickness*B*gaussWeights(q)*det(Jacob);

  end;
end;

% -----------------------------------------------------------------------
% Beginning of analysis
% -----------------------------------------------------------------------
cracklength=0; stepnumber=0; savingnr=0; time=1;
CMOD='positive';
longterm=0; strainCR(1)=0;
crack_old=-1;
force_crack=zeros(GDof,1);
strainRA(1)=0;
direction(1)=3; %direction [1=downward,2=right,3=upward,4=left]
str1(1)=0;str2(1)=0;str3(1)=0;str4(1)=0;str5(1)=0;str6(1)=0;

while cracklength<crackpropagation && time<=endtime-1
   stepnumber=stepnumber+1;

   force_crack=zeros(GDof,1);
   upwardlift=0;
   iteration=1;

   if cracklength>0 % changes in connection matrix/boundary condition
      cracksigma(cracklength,4)=0;



   end
   prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
   activeDof=setdiff([1:GDof]', [prescribedDof]);
   activeDof=setdiff(activeDof,[removednodes; removednodes+numberNodes]);

   while iteration==1 && time<=endtime

      savingnr=savingnr+1;
      force=force_primary;

      % ---------------------------------------------------------------
      % nodal forces
      % ---------------------------------------------------------------
      if cracklength>0;
         prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
         activeDof=setdiff([1:GDof]', [prescribedDof]);
         activeDof=setdiff(activeDof,...
            [removednodes; removednodes+numberNodes]);

         % ---------------------------------------------------------
         % Iterative method to find crack nodal forces
         % ---------------------------------------------------------
         force_crack=zeros(GDof,1);

         stopiteration=1; width_old=0;
         while stopiteration==1
            U=zeros(GDof,1);
            U(activeDof)=stiffness(activeDof,activeDof)\...
```

```
        (force(activeDof)+force_crack(activeDof));
    displacements=U;

    % -----------------------------------------------------
    % Crack width, crack sigma, nodal forces
    % -----------------------------------------------------
    force_old=min(force_crack);
    for j=1:cracklength
        if connection(tip(j,4),tip(j,3))==1
            displacements(tip(j,4))=displacements(tip(j,3));
            displacements(tip(j,4)+numberNodes)=displacements(tip(j,3)+numberNodes);
        end
        if connection(tip(j,2),tip(j,3))==1
            displacements(tip(j,2))=displacements(tip(j,3));
            displacements(tip(j,2)+numberNodes)=displacements(tip(j,3)+numberNodes);
        end
        if connection(tip(j,1),tip(j,3))==1
            displacements(tip(j,1))=displacements(tip(j,3));
            displacements(tip(j,1)+numberNodes)=displacements(tip(j,3)+numberNodes);
        end
        if connection(tip(j,2),tip(j,4))==1
            displacements(tip(j,2))=displacements(tip(j,4));
            displacements(tip(j,2)+numberNodes)=displacements(tip(j,4)+numberNodes);
        end
        if connection(tip(j,1),tip(j,4))==1
            displacements(tip(j,1))=displacements(tip(j,4));
            displacements(tip(j,1)+numberNodes)=displacements(tip(j,4)+numberNodes);
        end
        if connection(tip(j,1),tip(j,2))==1
            displacements(tip(j,1))=displacements(tip(j,2));
            displacements(tip(j,1)+numberNodes)=displacements(tip(j,2)+numberNodes);
        end

        if connection(tip(j,4),tip(j,3))==0 crackwidth(j,1)=displacements(tip(j,4))-displacements(tip(j,3)); else
crackwidth(j,1)=0; end
        if connection(tip(j,4),tip(j,1))==0 crackwidth(j,2)=-
displacements(tip(j,4)+numberNodes)+displacements(tip(j,1)+numberNodes); else crackwidth(j,2)=0; end
        if connection(tip(j,1),tip(j,2))==0 crackwidth(j,3)=displacements(tip(j,1))-displacements(tip(j,2)); else
crackwidth(j,3)=0;  end
        if connection(tip(j,2),tip(j,3))==0 crackwidth(j,4)=-
displacements(tip(j,3)+numberNodes)+displacements(tip(j,2)+numberNodes); else crackwidth(j,4)=0; end
        slide(j,1)=displacements(tip(j,4)+numberNodes)-displacements(tip(j,3)+numberNodes);
        slide(j,2)=-displacements(tip(j,4))+displacements(tip(j,1));
        slide(j,3)=displacements(tip(j,1)+numberNodes)-displacements(tip(j,2)+numberNodes);
        slide(j,4)=-displacements(tip(j,3))+displacements(tip(j,2));

        for ii=1:4
            if crackwidth(j,ii)>0
                if crackwidth(j,ii)>w0
                    cracksigma(j,ii)=0;
                else
                    if crackwidth(j,ii)<ws
                        cracksigma(j,ii)=fctm*(1-crackwidth(j,ii)/wc);
                    else
                        cracksigma(j,ii)=ft*(1-crackwidth(j,ii)/w0);
                    end
                end
            else
                stopiteration=0; %when CMOD becomes negative
                CMOD='negative';
                cracksigma(j,ii)=0;
            end

            if slide(j,ii)>0
                if slide(j,ii)>w02
                    cracksigma2(j,ii)=0;
                else
                    if slide(j,ii)<ws2
                        cracksigma2(j,ii)=fccm* (1-slide(j,ii)/wc2);
                    else
```

```
                        cracksigma2(j,ii)=ft2* (1-slide(j,ii)/w02);
                    end
                end
            else
                cracksigma2(j,ii)=0;
            end

        end;

        %traction & cohesion
        force_crack(tip(j,1))=-cracksigma(j,3)*thickness*Ly/numberElementsY/2-
cracksigma2(j,2)*thickness*Lx/numberElementsX/2;
        force_crack(tip(j,2))=cracksigma(j,3)*thickness*Ly/numberElementsY/2-
cracksigma2(j,4)*thickness*Lx/numberElementsX/2;

force_crack(tip(j,3))=cracksigma(j,1)*thickness*Ly/numberElementsY/2+cracksigma2(j,2)*thickness*Lx/numberElementsX/2;
        force_crack(tip(j,4))=-
cracksigma(j,1)*thickness*Ly/numberElementsY/2+cracksigma2(j,4)*thickness*Lx/numberElementsX/2;
        force_crack(tip(j,1)+numberNodes)=-cracksigma(j,2)*thickness*Lx/numberElementsX/2-
cracksigma2(j,3)*thickness*Ly/numberElementsY/2;
        force_crack(tip(j,2)+numberNodes)=-
cracksigma(j,4)*thickness*Lx/numberElementsX/2+cracksigma2(j,3)*thickness*Ly/numberElementsY/2;

force_crack(tip(j,3)+numberNodes)=cracksigma(j,2)*thickness*Lx/numberElementsX/2+cracksigma2(j,1)*thickness*Ly/number
ElementsY/2;
        force_crack(tip(j,4)+numberNodes)=cracksigma(j,4)*thickness*Lx/numberElementsX/2-
cracksigma2(j,1)*thickness*Ly/numberElementsY/2;

      end;

      if abs(max(crackwidth(cracklength,:))-width_old)/max(crackwidth(cracklength,:))<0.01
        stopiteration=0;
      end
      width_old=max(crackwidth(cracklength,:));

    end;

  end %if

  % --------------------------------------------------------------------
  % Solution
  % --------------------------------------------------------------------
  force=force_primary+force_crack;
  U=zeros(GDof,1);
  U(activeDof)=stiffness(activeDof,activeDof)\force(activeDof);
  displacements=U;
  Reaction(savingnr)=-P;
  midspandeflection(savingnr)=U(1+numberNodes);

  UX=displacements(1:numberNodes);
  UY=displacements(numberNodes+1:GDof);

  % --------------------------------------------------------------------
  % Stresses at nodes
  % --------------------------------------------------------------------
  stress=zeros(numberElements,size(elementNodes,2),3);
  stressPoints=[-1 -1;1 -1;1 1;-1 1];

  for e=1:numberElements
    indice=elementNodes(e,:);
    elementDof=[ indice indice+numberNodes ];
    nn=length(indice);
    for q=1:size(gaussWeights,1)
      pt=gaussLocations(q,:);
      wt=gaussWeights(q);
      xi=pt(1);
      eta=pt(2);
      % shape functions and derivatives
```

```
            shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
               (1+xi)*(1+eta);(1-xi)*(1+eta)];
            naturalDerivatives= 1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);  1+eta,      1+xi;-(1+eta),   1-xi];
            % Jacobian matrix, inverse of Jacobian,
            % derivatives w.r.t. x,y
            [Jacob,invJacobian,XYderivatives]= Jacobian(nodeCoordinates(indice,:),naturalDerivatives);
            %  B matrix
            B=zeros(3,2*nn);
            B(1,1:nn)       = XYderivatives(:,1)';
            B(2,nn+1:2*nn)  = XYderivatives(:,2)';
            B(3,1:nn)       = XYderivatives(:,2)';
            B(3,nn+1:2*nn)  = XYderivatives(:,1)';

            % element deformation
            strain=B*displacements(elementDof);
            stress(e,q,:)=C*strain;
         end
      end
      % Principal Stress
      stress_principal1=(stress(:,:,1)+stress(:,:,2))/2+  ((stress(:,:,1)-stress(:,:,2)).^2/4+stress(:,:,3).^2).^0.5;

      % Nodal stress
      for k=1:numberElements
         nodalstress(floor(k/numberElementsX)*4*numberElementsX+(k-floor(k/numberElementsX)*numberElementsX)*2-
1)=stress_principal1(k,1);   nodalstress(floor(k/numberElementsX)*4*numberElementsX+(k-
floor(k/numberElementsX)*numberElementsX)*2)=stress_principal1(k,2);
nodalstress((floor(k/numberElementsX)*2+1)*2*numberElementsX+(k-
floor(k/numberElementsX)*numberElementsX)*2)=stress_principal1(k,3);
nodalstress((floor(k/numberElementsX)*2+1)*2*numberElementsX+(k-floor(k/numberElementsX)*numberElementsX)*2-
1)=stress_principal1(k,4);
      end;

      % --------------------------------------------------------------
      % Drawing stress fields on top of the deformed shape
      % --------------------------------------------------------------
      if showstressgraph==1 && cracklength>crack_old
         figure
         drawingField(nodeCoordinates+scaleFactor*[UX UY], elementNodes,'Q4',stress_principal1);%sigma princ1
         hold on
         drawingMesh(nodeCoordinates+scaleFactor*[UX UY], elementNodes,'Q4','k-');
         drawingMesh(nodeCoordinates,elementNodes,'Q4','k--');
         colorbar
         title('Sigma 1 principal stress (on deformed shape)')
         axis off
         crack_old=cracklength;
      end

      % --------------------------------------------------------------
      % Stresses in nodes along the tip (the mean stress between 2 nodes)
      % --------------------------------------------------------------
      if cracklength==0
         x1=find(stress_principal1(:,1)==max(max(stress_principal1)));
         x2=find(stress_principal1(:,2)==max(max(stress_principal1)));
         x3=find(stress_principal1(:,3)==max(max(stress_principal1)));
         x4=find(stress_principal1(:,4)==max(max(stress_principal1)));
            % x? is the edge of the element with max stress

         if size(x1,1)==1
            tipp=floor(x1/numberElementsX)*4*numberElementsX+(x1-floor(x1/numberElementsX)*numberElementsX)*2-1;
            tip(cracklength+1,:)=[tipp;tipp-1;tipp-numberElementsX*2-1;tipp-numberElementsX*2];
            spot=[max(stress_principal1(x1-1-numberElementsX,2),stress_principal1(x1-
numberElementsX,1));max(stress_principal1(x1-
numberElementsX,3),stress_principal1(x1,2));max(stress_principal1(x1,4),stress_principal1(x1-1,3));max(stress_principal1(x1-
1,1),stress_principal1(x1-1-numberElementsX,4))];
            stress_tip(cracklength+1)=stress_principal1(x1,1);
         end
         if size(x2,1)==1
            tipp=floor(x2/numberElementsX)*4*numberElementsX+(x2-floor(x2/numberElementsX)*numberElementsX)*2;
            tip(cracklength+1,:)=[tipp+1;tipp;tipp-numberElementsX*2;tipp-numberElementsX*2+1];
```

```
            spot=[max(stress_principal1(x2-numberElementsX,2),stress_principal1(x2-
numberElementsX+1,1));max(stress_principal1(x2-
numberElementsX+1,3),stress_principal1(x2+1,2));max(stress_principal1(x2,3),stress_principal1(x2+1,4));max(stress_principal
1(x2-numberElementsX,4),stress_principal1(x2,1))];
            stress_tip(cracklength+1)=stress_principal1(x2,2);
        end
        if size(x3,1)==1
            tipp=(floor(x3/numberElementsX)*2+1)*2*numberElementsX+(x3-
floor(x3/numberElementsX)*numberElementsX)*2;
            tip(cracklength+1,:)=[tipp+numberElementsX*2+1;tipp+numberElementsX*2;tipp;tipp+1];

spot=[max(stress_principal1(x3,2),stress_principal1(x3+1,1));max(stress_principal1(x3+1,3),stress_principal1(x3+numberElem
entsX+1,2));max(stress_principal1(x3+numberElementsX+1,4),stress_principal1(x3+numberElementsX,3));max(stress_principa
l1(x3+numberElementsX,1),stress_principal1(x3,4))];
            stress_tip(cracklength+1)=stress_principal1(x3,3);
        end
        if size(x4,1)==1
            tipp=(floor(x4/numberElementsX)*2+1)*2*numberElementsX+(x4-
floor(x4/numberElementsX)*numberElementsX)*2-1;
            tip(cracklength+1,:)=[tipp+numberElementsX*2;tipp+numberElementsX*2-1;tipp-1;tipp];
            spot=[max(stress_principal1(x4-1,2),stress_principal1(x4,1));max(stress_principal1(x4-
1,3),stress_principal1(x4+numberElementsX,2));max(stress_principal1(x4+numberElementsX,4),stress_principal1(x4+numberEl
ementsX-1,3));max(stress_principal1(x4-1,4),stress_principal1(x4+numberElementsX-1,1))];
            stress_tip(cracklength+1)=stress_principal1(x4,4);
        end
    end



    % --------------------------------------------------------------
    % finding max stress and check with fctm
    % --------------------------------------------------------------
    for checkstress=1:cracklength+1
        if stress_tip(checkstress)>fctm
            strainEL((time-1)/time_interval+1,:)=critical_strain;
        else
            if stress_tip(checkstress)>fctm*0.9
                strainEL((time-1)/time_interval+1,:)=0.9*fctm/E+ (critical_strain-0.9*fctm/E)/(0.1*fctm)*...
                    (stress_tip(checkstress)-0.9*fctm);
            else
                strainEL((time-1)/time_interval+1,:)=stress_tip(checkstress)/E;
            end
        end
    end;
    straintotal((time-1)/time_interval+1,:)=(strainRA((time-1)/time_interval+1,:)+strainCR((time-
1)/time_interval+1,:))*longterm+strainEL((time-1)/time_interval+1,:);

    if max(straintotal((time-1)/time_interval+1,:))>=critical_strain
        iteration=0;
        Reaction(savingnr)=-P;
        midspandeflection(savingnr)=(U(1+numberNodes)-upwardlift);
        Results2(cracklength+1,:)=[midspandeflection(savingnr), Reaction(savingnr)];
        longterm=0;

        % making zero the spot where the crack comes from
        if direction(cracklength+1)==2 spot(4,1)=0; end
        if direction(cracklength+1)==3 spot(1,1)=0; end
        if direction(cracklength+1)==4 spot(2,1)=0; end
        if direction(cracklength+1)==1 spot(3,1)=0; end
        spot(1,1)=0; spot(2,1)=0; % To make the crack go top left
        direction(cracklength+2)=find(spot(:,1)==max(spot));

        % changes in connection matrix
        if direction(cracklength+1)==3 && direction(cracklength+2)==4
            connection(tip(cracklength+1,2),tip(cracklength+1,3))=0;
            connection(tip(cracklength+1,3),tip(cracklength+1,2))=0;
            connection(tip(cracklength+1,3),tip(cracklength+1,1))=0;
            connection(tip(cracklength+1,1),tip(cracklength+1,3))=0;
            connection(tip(cracklength+1,3),tip(cracklength+1,4))=0;
            connection(tip(cracklength+1,4),tip(cracklength+1,3))=0;
        end
```

```
if direction(cracklength+1)==4 && direction(cracklength+2)==3
    connection(tip(cracklength+1,1),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,1))=0;
end
if direction(cracklength+1)==3 && direction(cracklength+2)==3
    connection(tip(cracklength+1,1),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,4))=0;
end
if direction(cracklength+1)==3 && direction(cracklength+2)==2
    connection(tip(cracklength+1,4),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,4))=0;
end
if direction(cracklength+1)==2 && direction(cracklength+2)==3
    connection(tip(cracklength+1,2),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,2))=0;
end
if direction(cracklength+1)==2 && direction(cracklength+2)==2
    connection(tip(cracklength+1,2),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,1))=0;
end
if direction(cracklength+1)==4 && direction(cracklength+2)==4
    connection(tip(cracklength+1,2),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,2),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,2))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,3))=0;
    connection(tip(cracklength+1,3),tip(cracklength+1,1))=0;
    connection(tip(cracklength+1,1),tip(cracklength+1,4))=0;
    connection(tip(cracklength+1,4),tip(cracklength+1,1))=0;
end

cracklength=cracklength+1;

% new tip
if direction(cracklength+1)==1
    tip(cracklength+1,:)=tip(cracklength,:)-4*numberElementsX;
end
if direction(cracklength+1)==2
    tip(cracklength+1,:)=tip(cracklength,:)+2;
end
if direction(cracklength+1)==3
    tip(cracklength+1,:)=tip(cracklength,:)+4*numberElementsX;
end
if direction(cracklength+1)==4
    tip(cracklength+1,:)=tip(cracklength,:)-2;
end
```

```
    % --------------------------------------------------------
    % Modification element nodes
    % --------------------------------------------------------
    elementNodes=elementNodes_primary;
    removednodesnr=0;
    for j=1:numberElements;
        for i=1:4
            findconnection=find(connection(:,elementNodes(j,i))==1);
            if size(findconnection)>=1
                if min(findconnection)<elementNodes(j,i)
                    removednodesnr=removednodesnr+1;
                    removednodes(removednodesnr,1)=elementNodes(j,i);
                end
                elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
            end
        end;
    end;

    % --------------------------------------------------------
    % New Stiffness Matrix
    % --------------------------------------------------------
    stiffness=zeros(GDof);
    for e=1:numberElements
        indice=elementNodes(e,:);
        elementDof=[ indice indice+numberNodes ];
        ndof=length(indice);

        % cycle for Gauss point
        for q=1:size(gaussWeights,1)
            GaussPoint=gaussLocations(q,:);
            xi=GaussPoint(1);
            eta=GaussPoint(2);

            % shape functions and derivatives
            shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
                (1+xi)*(1+eta);(1-xi)*(1+eta)];
            naturalDerivatives=...
                1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                1+eta,      1+xi;-(1+eta),   1-xi];

            % Jacobian matrix, inverse of Jacobian,
            % derivatives w.r.t. x,y
            [Jacob,invJacobian,XYderivatives]=Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

            %  B matrix
            B=zeros(3,2*ndof);
            B(1,1:ndof)       = XYderivatives(:,1)';
            B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
            B(3,1:ndof)       = XYderivatives(:,2)';
            B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

            % stiffness matrix
            stiffness(elementDof,elementDof)=  stiffness(elementDof,elementDof)+...
                B'*C*thickness*B*gaussWeights(q)*det(Jacob);

        end;

    end;


else
    longterm=1;
    time=time+time_interval;
    timet=time/24/3600;
    if strcmpi(Code,'EC2')==1
        % Creep according to EC2
        betaC=(timet/(betaH+timet))^0.3;
```

```
            creep=betaC*phi0;
        else
            % Creep according to Model B3
            Zt=(t0^(0-m))*log(1+(timet)^n);
            Qt=Qf*(1+(Qf/Zt)^rt)^(0-1/rt);
            C0=q2*Qt+q3*log(1+(timet)^n)+q4*log((timet+t0)/t0);
            St=tanh(((timet+t0-ts)/tao_sh)^0.5);
            Htt=1-(1-RH)*St;
            Cd=q5*(exp(-8*Htt)-exp(-8*Ht))^0.5;
            Jt=q1+C0+Cd;
            creep=((Jt*E)/1000000-1);
        end

        for k=2:(time-1)/time_interval+1
            delt_str1=str1(k-1)*(exp(-time_interval/(eta1/E1))-1)+(time_interval/eta1)*exp(-
time_interval/(2*eta1/E1))*loadratio;
            str1(k)=str1(k-1)+delt_str1;
            delt_str2=str2(k-1)*(exp(-time_interval/(eta2/E2))-1)+(time_interval/eta2)*exp(-
time_interval/(2*eta2/E2))*loadratio;
            str2(k)=str2(k-1)+delt_str2;
            delt_str3=str3(k-1)*(exp(-time_interval/(eta3/E3))-1)+(time_interval/eta3)*exp(-
time_interval/(2*eta3/E3))*loadratio;
            str3(k)=str3(k-1)+delt_str3;
            delt_str4=str4(k-1)*(exp(-time_interval/(eta4/E4))-1)+(time_interval/eta4)*exp(-
time_interval/(2*eta4/E4))*loadratio;
            str4(k)=str4(k-1)+delt_str4;
            delt_str5=str5(k-1)*(exp(-time_interval/(eta5/E5))-1)+(time_interval/eta5)*exp(-
time_interval/(2*eta5/E5))*loadratio;
            str5(k)=str5(k-1)+delt_str5;
            delt_str6=str6(k-1)*(exp(-time_interval/(eta6/E6))-1)+(time_interval/eta6)*exp(-
time_interval/(2*eta6/E6))*loadratio;
            str6(k)=str6(k-1)+delt_str6;
        end
        str_visc=k/eta0*loadratio;
        straincrackrate=str_visc+str1(k)+str2(k)+str3(k)+str4(k)+str5(k)+str6(k);

        strainCR((time-1)/time_interval+1,:)=nodalstress(:,size(nodalstress',1))'*creep/E;
        strainRA((time-1)/time_interval+1,1)=straincrackrate*strainEL((time-1)/time_interval,1)/strainEL((time-
1)/time_interval,1);

      end

      if cracklength>0
        spot=[max(nodalstress(tip(cracklength+1,3)-2*numberElementsX),nodalstress(tip(cracklength+1,4)-
2*numberElementsX));...
              max(nodalstress(tip(cracklength+1,4)+1),nodalstress(tip(cracklength+1,1)+1));...
max(nodalstress(tip(cracklength+1,1)+2*numberElementsX),nodalstress(tip(cracklength+1,2)+2*numberElementsX));...
               max(nodalstress(tip(cracklength+1,2)-1),nodalstress(tip(cracklength+1,3)-1))];

stress_tip(cracklength+1)=max(max(nodalstress(tip(cracklength+1,1)),nodalstress(tip(cracklength+1,2))),max(nodalstress(tip(c
racklength+1,3)),nodalstress(tip(cracklength+1,4))));
      end

      Results(savingnr,:)=[midspandeflection(savingnr), Reaction(savingnr)];%,stepnumber,cracklength];

    end %interation

end %stepnumber

plot(straintotal);
```

Appendix B: FE Model of a shear crack in reinforced concrete

```
%..........................
% Oneshearcrack.m
% ---------------------------------
% clear memory
clear all;colordef white;clf
crackpropagation=18;
endtime=10000; %(seconds of test)
time_interval=endtime/1;
showstressgraph=1; %=0 graph off    =1 graph on
Code='EC2'; % EC2 or B3

% ---------------------------------
% Given material property and geometry
% ---------------------------------
fcube=45;
poisson = 0.20;
critical_strain=0.00015;
thickness=200;
Lx=2400; %Length of the beam
Ly=450; %Height
d=Ly-40; %effective depth
As=3*10^2*3.14; %reinforcement
Es=210000;
fy=420;

% ---------------------------------
% Calculated material property
% ---------------------------------
fcm = fcube*0.785;
Einitial = 22000*(fcm/10)^0.3;
E=Einitial;
if strcmpi(Code,'EC2')==1
   fctm = 0.3*(fcm-8)^(2/3); % Eurocode2
else
   fctm=(145.0377*fcm)^0.5*6.7/145.0377; % fctm based on ACI
end

% ---------------------------------
% Fracture Energy Mode I
% ---------------------------------
alphaF=7; alphaD=6;
GF=alphaD/1000*fcm^0.7*2;
w0=GF*alphaF/fctm;
ws=2*GF/fctm-0.15*w0;
sigmaS=0.15*fctm;
wc=ws/0.85;
ft=w0*sigmaS/(w0-ws);

% ---------------------------------
% Fracture Energy Mode II
% ---------------------------------
fccm=fctm; %cohesion
GF2=alphaD/1000*fcm^0.7;
w02=GF2*alphaF/fccm;
ws2=2*GF2/fccm-0.15*w02;
sigmaS2=0.15*fccm;
wc2=ws2/0.85;
ft2=w02*sigmaS2/(w02-ws2);

% ---------------------------------
% Creep function based on EC2 or B3 Model
% ---------------------------------
RH = 0.5; %Relative Humidity
if strcmpi(Code,'EC2')==1
   % Creep according to Eurocode
   alpha1=(35/fcm)^0.7; alpha2=(35/fcm)^0.2; alpha3=(35/fcm)^0.5;
```

```
    betafcm=16.8/fcm^0.5;
    h0=thickness*Ly/(thickness+Ly);
    ts=7; %curing time in days
    t0=28; % age in days at loading
    if fcm>35
        phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1)*alpha2;
        betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250*alpha3,1500*alpha3);
    else
        phiRH=(1+(1-RH)/(0.1*h0^(1/3))*alpha1);
        betaH=min(1.5*(1+(0.012*RH*100)^18)*h0+250,1500);
    end
    betat0=1/(0.1+t0^0.2);
    phi0=phiRH*betafcm*betat0;
else
    % Creep according to Model B3
    ts=7; %curing time in days
    t0=28; % age in days at loading
    Qf=(0.086*t0^(2/9)+1.21*t0^(4/9))^(-1);
    m=0.5; n= 0.1;
    rt=1.7*t0^0.12+8;
    cement=330; water=189; aggregate=1803;
    q1=0.6*1000000/E;
    q2=185.4*cement^0.5*fcm^(-0.9);
    q3=0.29*(water/cement)^4*q2;
    q4=20.3*(aggregate/cement)^(-0.7);

    ks=1.25;%(slab=1)(inf cyl=1.15)(inf sq prsm=1.25)(sphr=1.30)(cube=1.55)
    kt=8.5*ts^(-0.08)*fcm^(-0.25)/100;
    h0=thickness*Ly/(thickness+Ly);
    tao_sh=kt*(ks*h0)^2;
    alpha1=1.1; %(CEMENT type I=1.0)(CEM Type II=0.85)(CEM Type III=1.1)
    alpha2=1.0; %(Steam curing=0.75)(Normal Cur=1.2)(Curing in RH100%=1.0)
    Ht=1-(1-RH)*(tanh(((t0-ts)/tao_sh)^0.5));
    epsilon_infinit=-alpha1*alpha2*(1.9/100*(water^2.1)*(fcm^(-0.28))+270);
    q5=7.57*100000/fcm*(abs(epsilon_infinit)^(-0.6));
end

% ---------------------------------
% Support and boundary conditions
% ---------------------------------
supportX1=0;
supportX2=2400;
loadingplateX=1200;
CrackX=1900;

% ---------------------------------
% matrix C
% ---------------------------------
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% ---------------------------------
% load
% ---------------------------------
Pu= -184640;     accuracy=0.015;
loadratio=.95;
P = Pu*loadratio;
Dowel_action=1000;

% ---------------------------------
% Parameters of crack growth rate
% ---------------------------------
tao1 = 5;  tao2 = 25; tao3 = 200; tao4 = 1000; tao5 = 10000; tao6 = 100000;
A1 = 13600-1000* log(tao1) ; B1 = 0.53*log(tao1)-8.75; E1 = A1*loadratio^B1; eta1 = tao1 * E1;
A2 = 13600-1000* log(tao2) ; B2 = 0.53*log(tao2)-8.75; E2 = A2*loadratio^B2; eta2 = tao2 * E2;
A3 = 13600-1000* log(tao3) ; B3 = 0.53*log(tao3)-8.75; E3 = A3*loadratio^B3; eta3 = tao3 * E3;
A4 = 13600-1000* log(tao4) ; B4 = 0.53*log(tao4)-8.75; E4 = A4*loadratio^B4; eta4 = tao4 * E4;
A5 = 13600-1000* log(tao5) ; B5 = 0.53*log(tao5)-8.75; E5 = A5*loadratio^B5; eta5 = tao5 * E5;
A6 = 13600-1000* log(tao6) ; B6 = 0.53*log(tao6)-8.75; E6 = A6*loadratio^B6; eta6 = tao6 * E6;
E0 = 1646.1*loadratio^(-2.516); eta0 = 90000000*loadratio^(-7.4);

% ---------------------------------
```

```
% Mesh generation
% ---------------------------------
numberElementsX=48;
numberElementsY=15;
numberElements=numberElementsX*numberElementsY;

% --------------------------------------------------------------------
% node coordinates
% --------------------------------------------------------------------
nodeCoordinates=zeros((numberElementsX*2)*(numberElementsY*2),2);
for j=1:numberElementsY
    for i=1:numberElementsX
        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),1)=...
            (i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),1)=...
            i*Lx/numberElementsX;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),1)=...
            (i-1)*Lx/numberElementsX;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),1)=...
            i*Lx/numberElementsX;

        nodeCoordinates(i*2-1+numberElementsX*4*(j-1),2)=...
            (j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*4*(j-1),2)=...
            (j-1)*Ly/numberElementsY;
        nodeCoordinates(i*2-1+numberElementsX*(4*(j-1)+2),2)=...
            j*Ly/numberElementsY;
        nodeCoordinates(i*2+numberElementsX*(4*(j-1)+2),2)=...
            j*Ly/numberElementsY;
    end;
end;

% --------------------------------------------------------------------
% element nodes
% --------------------------------------------------------------------
elementNodes_primary=zeros(numberElements,4);
for j=0:numberElementsY-1
    for i=1:numberElementsX
        elementNodes_primary(i+j*(numberElementsX),1)=...
            i*2-1+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),2)=...
            i*2+numberElementsX*4*j;
        elementNodes_primary(i+j*(numberElementsX),3)=...
            i*2+numberElementsX*(4*j+2);
        elementNodes_primary(i+j*(numberElementsX),4)=...
            i*2-1+numberElementsX*(4*j+2);
    end;
end;

xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
numberNodes=size(xx,1);

% --------------------------------------------------------------------
% Connection matrix
% --------------------------------------------------------------------
connection=zeros(numberNodes);
for j=1:numberNodes
    for i=j+1:numberNodes
        if nodeCoordinates(i,:)==nodeCoordinates(j,:)
            connection(i,j)=1;
            connection(j,i)=1;
        end
    end;
end;
connectionTri=triu(connection);

% --------------------------------------------------------------------
% Shear Crack
% --------------------------------------------------------------------
for i=1:numberElementsX*2
```

```
    if nodeCoordinates(i,1)==CrackX && nodeCoordinates(i+1,1)==CrackX
        connection(i,i+1)=0;
        connection(i+1,i)=0;
        %connection(i,i+numberElementsX*2+1)=0;
        %connection(i+1,i+numberElementsX*2-1)=0;
        tip(1,:)=[i+numberElementsX*4+1;i+numberElementsX*4;i+numberElementsX*2;i+numberElementsX*2+1];
    end
end;
direction=[3;3;3;4;3;4;3;4;3;4;3;4;3;4;3;4;4;]; %direction [1=downward,2=right,3=upward,4=left]
% changes in connection matrix
cracklength=17;
for i=1:cracklength-1
    if direction(i+1)==3

tip(i+1,:)=[tip(i,1)+numberElementsX*4;tip(i,2)+numberElementsX*4;tip(i,3)+numberElementsX*4;tip(i,4)+numberElementsX
*4];
    end
    if direction(i+1)==4
        tip(i+1,:)=[tip(i,1)-2;tip(i,2)-2;tip(i,3)-2;tip(i,4)-2];
    end
    if direction(i)==3 && direction(i+1)==4
        connection(tip(i,2),tip(i,3))=0;
        connection(tip(i,3),tip(i,2))=0;
        connection(tip(i,3),tip(i,1))=0;
        connection(tip(i,1),tip(i,3))=0;
        connection(tip(i,3),tip(i,4))=0;
        connection(tip(i,4),tip(i,3))=0;
    end
    if direction(i)==4 && direction(i+1)==3
        connection(tip(i,1),tip(i,2))=0;
        connection(tip(i,2),tip(i,1))=0;
        connection(tip(i,1),tip(i,3))=0;
        connection(tip(i,3),tip(i,1))=0;
        connection(tip(i,1),tip(i,4))=0;
        connection(tip(i,4),tip(i,1))=0;
    end
    if direction(i)==3 && direction(i+1)==3
        connection(tip(i,1),tip(i,2))=0;
        connection(tip(i,2),tip(i,1))=0;
        connection(tip(i,1),tip(i,3))=0;
        connection(tip(i,3),tip(i,1))=0;
        connection(tip(i,4),tip(i,2))=0;
        connection(tip(i,2),tip(i,4))=0;
        connection(tip(i,4),tip(i,3))=0;
        connection(tip(i,3),tip(i,4))=0;
    end
    if direction(i)==4 && direction(i+1)==4
        connection(tip(i,2),tip(i,3))=0;
        connection(tip(i,3),tip(i,2))=0;
        connection(tip(i,2),tip(i,4))=0;
        connection(tip(i,4),tip(i,2))=0;
        connection(tip(i,1),tip(i,3))=0;
        connection(tip(i,3),tip(i,1))=0;
        connection(tip(i,1),tip(i,4))=0;
        connection(tip(i,4),tip(i,1))=0;
    end
end

% ------------------------------------------------------------------
% Modification element nodes
% ------------------------------------------------------------------
elementNodes=elementNodes_primary;
removednodesnr=0;
for j=1:numberElements;
    for i=1:4
        findconnection=find(connection(:,elementNodes(j,i))==1);
        if size(findconnection)>=1
            if min(findconnection)<elementNodes(j,i)
                removednodesnr=removednodesnr+1;
                removednodes(removednodesnr,1)=elementNodes(j,i);
            end
```

```
            elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
        end
    end;
end;

%drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
scaleFactor=100;

% --------------------------------------------------------------------
% GDof: global number of degrees of freedom
% --------------------------------------------------------------------
GDof=2*numberNodes;

% --------------------------------------------------------------------
% boundary conditions
% --------------------------------------------------------------------
fixedNodeX=find(nodeCoordinates(:,1)==loadingplateX & nodeCoordinates(:,2)==Ly);  % fixed in XX
fixedNodeY=[find(nodeCoordinates(:,1)==supportX1 & nodeCoordinates(:,2)==0);find(nodeCoordinates(:,1)==supportX2 &
nodeCoordinates(:,2)==0)];  % fixed in YY

% --------------------------------------------------------------------
% force vector (point load applied at xx=loadingplateX, yy=Ly)
% --------------------------------------------------------------------
force_primary=zeros(GDof,1);
leftBord=find(nodeCoordinates(:,1)==loadingplateX);
force_primary(leftBord(end-1)+numberNodes)=P;
force=force_primary;

% --------------------------------------------------------------------
% Stiffness Matrix
% --------------------------------------------------------------------
stiffness=zeros(GDof);
        gaussLocations=...
          [ -0.577350269189626 -0.577350269189626;
             0.577350269189626 -0.577350269189626;
             0.577350269189626  0.577350269189626;
            -0.577350269189626  0.577350269189626];
        gaussWeights=[ 1;1;1;1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  ndof=length(indice);

  % cycle for Gauss point
   for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

% shape functions and derivatives
    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
            (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives=...
           1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                 1+eta,      1+xi;-(1+eta),   1-xi];

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

%  B matrix
    B=zeros(3,2*ndof);
    B(1,1:ndof)       = XYderivatives(:,1)';
    B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
    B(3,1:ndof)       = XYderivatives(:,2)';
    B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

% stiffness matrix
    stiffness(elementDof,elementDof)=...
```

```
        stiffness(elementDof,elementDof)+...
        B'*C*thickness*B*gaussWeights(q)*det(Jacob);

    end;

end;

% --------------------------------------------------------------------
% --------------------------------------------------------------------
% Beginning of analysis
% --------------------------------------------------------------------
% --------------------------------------------------------------------
stepnumber=0; savingnr=0; time=1;
CMOD='positive';
longterm=0; strainCR(1)=0;
crack_old=-1;
%strainEL=zeros(endtime,numberElementsY+1-findnotchtip);
strainRA(1)=0;
str1(1)=0;str2(1)=0;str3(1)=0;str4(1)=0;str5(1)=0;str6(1)=0;

reinf((time-1)/time_interval+1,1)=0; stop=0;

while cracklength<crackpropagation && time<=endtime+1
    stepnumber=stepnumber+1;


    iteration=1;
    cracksigma(cracklength,4)=0;

    prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];
    activeDof=setdiff([1:GDof]', [prescribedDof]);
    activeDof=setdiff(activeDof,[removednodes; removednodes+numberNodes]);

    while iteration==1 && time<=endtime+1

        savingnr=savingnr+1;

        % --------------------------------------------------------------
        % nodal forces
        % --------------------------------------------------------------

            % -------------------------------------------------------
            % Iterative method to find crack nodal forces
            % -------------------------------------------------------
            stopiteration=1; width_old((time-1)/time_interval+1,1)=0; ttt=1;
            %for bbb=1:60
            crackwidth=zeros(cracklength,4);
            force_reinf=zeros(GDof,1);
            force_crack=zeros(GDof,1); about=0; yield=0; lastiter=0;
            while stopiteration==1
                force=force_primary+force_reinf+force_crack;
                U=zeros(GDof,1);
                U(activeDof)=stiffness(activeDof,activeDof)\...
                    force(activeDof);
                displacements=U;

                % ---------------------------------------------------
                % Crack width, crack sigma, nodal forces
                % ---------------------------------------------------
                for j=1:cracklength
                    if connection(tip(j,4),tip(j,3))==1
                        displacements(tip(j,4))=displacements(tip(j,3));
                        displacements(tip(j,4)+numberNodes)=displacements(tip(j,3)+numberNodes);
                    end
                    if connection(tip(j,2),tip(j,3))==1
                        displacements(tip(j,2))=displacements(tip(j,3));
                        displacements(tip(j,2)+numberNodes)=displacements(tip(j,3)+numberNodes);
                    end
                    if connection(tip(j,1),tip(j,3))==1
                        displacements(tip(j,1))=displacements(tip(j,3));
                        displacements(tip(j,1)+numberNodes)=displacements(tip(j,3)+numberNodes);
```

```
                end
                if connection(tip(j,2),tip(j,4))==1
                    displacements(tip(j,2))=displacements(tip(j,4));
                    displacements(tip(j,2)+numberNodes)=displacements(tip(j,4)+numberNodes);
                end
                if connection(tip(j,1),tip(j,4))==1
                    displacements(tip(j,1))=displacements(tip(j,4));
                    displacements(tip(j,1)+numberNodes)=displacements(tip(j,4)+numberNodes);
                end
                if connection(tip(j,1),tip(j,2))==1
                    displacements(tip(j,1))=displacements(tip(j,2));
                    displacements(tip(j,1)+numberNodes)=displacements(tip(j,2)+numberNodes);
                end

                if connection(tip(j,4),tip(j,3))==0 crackwidth(j,1)=max((displacements(tip(j,4))-
displacements(tip(j,3)))*(1+strainRA((time-1)/time_interval+1,1)),0); else crackwidth(j,1)=0; end
                if connection(tip(j,4),tip(j,1))==0 crackwidth(j,2)=max((-
displacements(tip(j,4)+numberNodes)+displacements(tip(j,1)+numberNodes))*(1+strainRA((time-1)/time_interval+1,1)),0);
else crackwidth(j,2)=0; end
                if connection(tip(j,1),tip(j,2))==0 crackwidth(j,3)=max((displacements(tip(j,1))-
displacements(tip(j,2)))*(1+strainRA((time-1)/time_interval+1,1)),0); else crackwidth(j,3)=0;  end
                if connection(tip(j,2),tip(j,3))==0 crackwidth(j,4)=max((-
displacements(tip(j,3)+numberNodes)+displacements(tip(j,2)+numberNodes))*(1+strainRA((time-1)/time_interval+1,1)),0);
else crackwidth(j,4)=0; end
                slide(j,1)=displacements(tip(j,4)+numberNodes)-displacements(tip(j,3)+numberNodes);
                slide(j,2)=-displacements(tip(j,4))+displacements(tip(j,1));
                slide(j,3)=displacements(tip(j,1)+numberNodes)-displacements(tip(j,2)+numberNodes);
                slide(j,4)=-displacements(tip(j,3))+displacements(tip(j,2));

                for ii=1:4
                    if crackwidth(j,ii)>0
                        if crackwidth(j,ii)>w0
                            cracksigma(j,ii)=0;
                        else
                            if crackwidth(j,ii)<ws
                                cracksigma(j,ii)=fctm*(1-crackwidth(j,ii)/wc);
                            else
                                cracksigma(j,ii)=ft*(1-crackwidth(j,ii)/w0);
                            end
                        end
                    else
                        %when CMOD becomes negative
                        CMOD='negative';
                        cracksigma(j,ii)=0;
                    end

                    if slide(j,ii)>0
                        if slide(j,ii)>w02
                            cracksigma2(j,ii)=0;
                        else
                            if slide(j,ii)<ws2
                                cracksigma2(j,ii)=fccm*...
                                    (1-slide(j,ii)/wc2);
                            else
                                cracksigma2(j,ii)=ft2*...
                                    (1-slide(j,ii)/w02);
                            end
                        end
                    else
                        cracksigma2(j,ii)=0;
                    end

                end;
                %traction and cohesion
                force_crack(tip(j,1))=-cracksigma(j,3)*thickness*Ly/numberElementsY/2-
cracksigma2(j,2)*thickness*Lx/numberElementsX/2;
                force_crack(tip(j,2))=cracksigma(j,3)*thickness*Ly/numberElementsY/2-
cracksigma2(j,4)*thickness*Lx/numberElementsX/2;

force_crack(tip(j,3))=cracksigma(j,1)*thickness*Ly/numberElementsY/2+cracksigma2(j,2)*thickness*Lx/numberElementsX/2;
```

```
                force_crack(tip(j,4))=-
cracksigma(j,1)*thickness*Ly/numberElementsY/2+cracksigma2(j,4)*thickness*Lx/numberElementsX/2;
                force_crack(tip(j,1)+numberNodes)=-cracksigma(j,2)*thickness*Lx/numberElementsX/2-
cracksigma2(j,3)*thickness*Ly/numberElementsY/2;
                force_crack(tip(j,2)+numberNodes)=-
cracksigma(j,4)*thickness*Lx/numberElementsX/2+cracksigma2(j,3)*thickness*Ly/numberElementsY/2;

force_crack(tip(j,3)+numberNodes)=cracksigma(j,2)*thickness*Lx/numberElementsX/2+cracksigma2(j,1)*thickness*Ly/number
ElementsY/2;
                force_crack(tip(j,4)+numberNodes)=cracksigma(j,4)*thickness*Lx/numberElementsX/2-
cracksigma2(j,1)*thickness*Ly/numberElementsY/2;

            end;

            % --------------------------------------------------------
            % steel tension: strain=delta u/Scr ,(Scr=% 0,7(d-c)
            % --------------------------------------------------------
            ttt=ttt+1;
            width_old((time-1)/time_interval+1,ttt)=(1/3*crackwidth(1,1)+2/3*crackwidth(2,1));
            slide_old((time-1)/time_interval+1,ttt)=(1/3*slide(1,1)+2/3*slide(2,1));
            reinf_tension(ttt)=As*Es*width_old((time-1)/time_interval+1,ttt)/(0.7*(d-150));
            reinf((time-1)/time_interval+1,1)=reinf_tension(ttt);

            if yield == 0
                reinf((time-1)/time_interval+1,ttt)=reinf((time-1)/time_interval+1,ttt-1)*(1-1/ttt^1);
                Dowel_action=9/26*reinf((time-1)/time_interval+1,ttt);
            end
            if about==0 && ttt>5 && reinf_tension(ttt)>reinf((time-1)/time_interval+1,ttt)
               about=1;
            end
            if about==1 && yield == 0
                if reinf_tension(ttt)>reinf((time-1)/time_interval+1,ttt-1)
                    reinf((time-1)/time_interval+1,ttt)=reinf((time-1)/time_interval+1,ttt-2)/min((1-1/ttt^2),1-1/4000);
                else
                    reinf((time-1)/time_interval+1,ttt)=reinf((time-1)/time_interval+1,ttt-1)*min((1-1/ttt^3),1-1/4000);
                end
            end
            if yield==1 && ttt>10
                stopiteration=0;
                reinf((time-1)/time_interval+1,ttt)=As*fy;
                Dowel_action=9/26*reinf((time-1)/time_interval+1,ttt);
            end
            if ttt>10 && yield==0 && reinf_tension(ttt)>0
                if reinf((time-1)/time_interval+1,ttt)>As*fy
                    reinf((time-1)/time_interval+1,ttt)=As*fy;
                    Dowel_action=9/26*reinf((time-1)/time_interval+1,ttt);
                    yield = 1;
                end
            end

            if ttt>5 && yield==0
                if reinf_tension(ttt)>0 && abs(reinf_tension(ttt)-reinf((time-1)/time_interval+1,ttt-1))/reinf_tension(ttt)<accuracy
                    stopiteration=0;lastiter=lastiter+1;
                    reinf((time-1)/time_interval+1,ttt)=reinf((time-1)/time_interval+1,ttt-1);
                end
            end


            force_reinf(numberElementsX*2+1)=reinf((time-1)/time_interval+1,ttt)/6;
            force_reinf(numberElementsX*4)=-reinf((time-1)/time_interval+1,ttt)/6;
            force_reinf(numberElementsX*4+1)=reinf((time-1)/time_interval+1,ttt)/6;
            force_reinf(numberElementsX*6)=-reinf((time-1)/time_interval+1,ttt)/6;
            force_reinf(numberElementsX*6+1)=reinf((time-1)/time_interval+1,ttt)/3;
            force_reinf(numberElementsX*8)=-reinf((time-1)/time_interval+1,ttt)/3;
            force_reinf(numberElementsX*8+1)=reinf((time-1)/time_interval+1,ttt)/3;
            force_reinf(numberElementsX*10)=-reinf((time-1)/time_interval+1,ttt)/3;

            force_reinf(tip(2,1)+numberNodes)=-Dowel_action/3;
            force_reinf(tip(1,1)+numberNodes)=-Dowel_action/6;
            force_reinf(tip(2,2)+numberNodes)=Dowel_action/3;
            force_reinf(tip(1,2)+numberNodes)=Dowel_action/6;
```

```
            force_reinf(tip(2,3)+numberNodes)=Dowel_action/3;
            force_reinf(tip(1,3)+numberNodes)=Dowel_action/6;
            force_reinf(tip(2,4)+numberNodes)=-Dowel_action/3;
            force_reinf(tip(1,4)+numberNodes)=-Dowel_action/6;

        end;


    % ----------------------------------------------------------------
    % Solution
    % ----------------------------------------------------------------
    force=force_primary+force_reinf+force_crack;
    U(activeDof)=stiffness(activeDof,activeDof)\force(activeDof);
    displacements=U;
    Reaction(savingnr)=-P;
    midspandeflection(savingnr)=U(1+numberNodes);

    UX=displacements(1:numberNodes);
    UY=displacements(numberNodes+1:GDof);

    % ----------------------------------------------------------------
    % Stresses at nodes
    % ----------------------------------------------------------------
    stress=zeros(numberElements,size(elementNodes,2),3);
    stressPoints=[-1 -1;1 -1;1 1;-1 1];

    for e=1:numberElements
        indice=elementNodes(e,:);
        elementDof=[ indice indice+numberNodes ];
        nn=length(indice);
        for q=1:size(gaussWeights,1)
            pt=gaussLocations(q,:);
            wt=gaussWeights(q);
            xi=pt(1);
            eta=pt(2);
            % shape functions and derivatives
            shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
                (1+xi)*(1+eta);(1-xi)*(1+eta)];
            naturalDerivatives=...
                1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                1+eta,      1+xi;-(1+eta),    1-xi];
            % Jacobian matrix, inverse of Jacobian,
            % derivatives w.r.t. x,y
            [Jacob,invJacobian,XYderivatives]=...
                Jacobian(nodeCoordinates(indice,:),naturalDerivatives);
            %  B matrix
            B=zeros(3,2*nn);
            B(1,1:nn)       = XYderivatives(:,1)';
            B(2,nn+1:2*nn)  = XYderivatives(:,2)';
            B(3,1:nn)       = XYderivatives(:,2)';
            B(3,nn+1:2*nn)  = XYderivatives(:,1)';

            % element deformation
            strain=B*displacements(elementDof);
            stress(e,q,:)=C*strain;
        end
    end
    % Principal Stress
    stress_principal1=(stress(:,:,1)+stress(:,:,2))/2+...
        ((stress(:,:,1)-stress(:,:,2)).^2/4+stress(:,:,3).^2).^0.5;

    % Nodal stress
    for k=1:numberElements
        nodalstress(floor(k/numberElementsX)*4*numberElementsX+(k-floor(k/numberElementsX)*numberElementsX)*2-
1)=stress_principal1(k,1);
        nodalstress(floor(k/numberElementsX)*4*numberElementsX+(k-
floor(k/numberElementsX)*numberElementsX)*2)=stress_principal1(k,2);
        nodalstress((floor(k/numberElementsX)*2+1)*2*numberElementsX+(k-
floor(k/numberElementsX)*numberElementsX)*2)=stress_principal1(k,3);
        nodalstress((floor(k/numberElementsX)*2+1)*2*numberElementsX+(k-
floor(k/numberElementsX)*numberElementsX)*2-1)=stress_principal1(k,4);
```

```
    end;

    % -----------------------------------------------------------
    % Drawing stress fields on top of the deformed shape
    % -----------------------------------------------------------
    if showstressgraph==1 && cracklength>crack_old
        figure
        drawingField(nodeCoordinates+scaleFactor*[UX UY],...
            elementNodes,'Q4',stress_principal1);%sigma princ1
        hold on
        drawingMesh(nodeCoordinates+scaleFactor*[UX UY],...
            elementNodes,'Q4','k-');
        drawingMesh(nodeCoordinates,elementNodes,'Q4','k--');
        colorbar
        title('Sigma 1 principal stress (on deformed shape)')
        axis off
        crack_old=cracklength;
    end

    % -----------------------------------------------------------
    % finding max stress and check with fctm
    % -----------------------------------------------------------
    stress_tip((time-
1)/time_interval+1,cracklength)=(nodalstress(tip(cracklength,1))+nodalstress(tip(cracklength,2))+nodalstress(tip(cracklength,3
))+nodalstress(tip(cracklength,4)))/4;

    %for checkstress=2:cracklength+1
        if stress_tip((time-1)/time_interval+1,cracklength)>fctm
            strainEL((time-1)/time_interval+1,:)=critical_strain;
        else
            if stress_tip((time-1)/time_interval+1,cracklength)>fctm*0.9
                strainEL((time-1)/time_interval+1,:)=0.9*fctm/E+...
                    (critical_strain-0.9*fctm/E)/(0.1*fctm)*...
                    (stress_tip((time-1)/time_interval+1,cracklength)-0.9*fctm);
            else
                strainEL((time-1)/time_interval+1,:)=stress_tip((time-1)/time_interval+1,cracklength)/E;
            end
        end
    %end;
    straintotal((time-1)/time_interval+1,:)=(strainCR((time-1)/time_interval+1,:))*longterm+strainEL((time-
1)/time_interval+1,:);
    %straintotal((time-1)/time_interval+1,:)=(strainRA((time-1)/time_interval+1,:)+strainCR((time-
1)/time_interval+1,:))*longterm+strainEL((time-1)/time_interval+1,:);

    if max(straintotal((time-1)/time_interval+1,:))>=critical_strain
        iteration=0;
        Reaction(savingnr)=-P;
        midspandeflection(savingnr)=U(1+numberNodes);
        Results2(cracklength+1,:)=[midspandeflection(savingnr),...
            Reaction(savingnr)];
        longterm=0;

        spot=[max(nodalstress(tip(cracklength,3)-2*numberElementsX),nodalstress(tip(cracklength,4)-
2*numberElementsX));...
              max(nodalstress(tip(cracklength,4)+1),nodalstress(tip(cracklength,1)+1));...

max(nodalstress(tip(cracklength,1)+2*numberElementsX),nodalstress(tip(cracklength,2)+2*numberElementsX));...
              max(nodalstress(tip(cracklength,2)-1),nodalstress(tip(cracklength,3)-1))];

        % making zero the spot where the crack comes from
        if direction(cracklength)==2 spot(4,1)=0; end
        if direction(cracklength)==3 spot(1,1)=0; end
        if direction(cracklength)==4 spot(2,1)=0; end
        if direction(cracklength)==1 spot(3,1)=0; end
        spot(1,1)=0; spot(2,1)=0; % To make the crack go top left
        direction(cracklength+1)=find(spot(:,1)==max(spot));

        %if direction(cracklength+2)==1 direction(cracklength+2)=4;
        %tip(cracklength+1)=tip(cracklength-1);end

        % changes in connection matrix
```

```
if direction(cracklength)==3 && direction(cracklength+1)==4
    connection(tip(cracklength,2),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,2))=0;
    connection(tip(cracklength,3),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,3))=0;
end
if direction(cracklength)==4 && direction(cracklength+1)==3
    connection(tip(cracklength,1),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,1))=0;
end
if direction(cracklength)==3 && direction(cracklength+1)==3
    connection(tip(cracklength,1),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,1))=0;
    connection(tip(cracklength,4),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,4))=0;
end
if direction(cracklength)==3 && direction(cracklength+1)==2
    connection(tip(cracklength,4),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,4))=0;
end
if direction(cracklength)==2 && direction(cracklength+1)==3
    connection(tip(cracklength,2),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,2))=0;
end
if direction(cracklength)==2 && direction(cracklength+1)==2
    connection(tip(cracklength,2),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,2))=0;
    connection(tip(cracklength,1),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,1))=0;
end
if direction(cracklength)==4 && direction(cracklength+1)==4
    connection(tip(cracklength,2),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,2))=0;
    connection(tip(cracklength,2),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,2))=0;
    connection(tip(cracklength,1),tip(cracklength,3))=0;
    connection(tip(cracklength,3),tip(cracklength,1))=0;
    connection(tip(cracklength,1),tip(cracklength,4))=0;
    connection(tip(cracklength,4),tip(cracklength,1))=0;
end

cracklength=cracklength+1;

% new tip
if direction(cracklength)==1
    tip(cracklength,:)=tip(cracklength-1,:)-4*numberElementsX;
end
if direction(cracklength)==2
    tip(cracklength,:)=tip(cracklength-1,:)+2;
```

```
    end
    if direction(cracklength)==3
        tip(cracklength,:)=tip(cracklength-1,:)+4*numberElementsX;
    end
    if direction(cracklength)==4
        tip(cracklength,:)=tip(cracklength-1,:)-2;
    end

    % ---------------------------------------------------------
    % Modification element nodes
    % ---------------------------------------------------------
    elementNodes=elementNodes_primary;
    removednodesnr=0;
    for j=1:numberElements;
        for i=1:4
            findconnection=find(connection(:,elementNodes(j,i))==1);
            if size(findconnection)>=1
                if min(findconnection)<elementNodes(j,i)
                    removednodesnr=removednodesnr+1;
                    removednodes(removednodesnr,1)=elementNodes(j,i);
                end
                elementNodes(j,i)=min(min(findconnection),elementNodes(j,i));
            end
        end;
    end;

    % ---------------------------------------------------------
    % New Stiffness Matrix
    % ---------------------------------------------------------
    stiffness=zeros(GDof);
    for e=1:numberElements
        indice=elementNodes(e,:);
        elementDof=[ indice indice+numberNodes ];
        ndof=length(indice);

        % cycle for Gauss point
        for q=1:size(gaussWeights,1)
            GaussPoint=gaussLocations(q,:);
            xi=GaussPoint(1);
            eta=GaussPoint(2);

            % shape functions and derivatives
            shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
                (1+xi)*(1+eta);(1-xi)*(1+eta)];
            naturalDerivatives=...
                1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                1+eta,      1+xi;-(1+eta),   1-xi];

            % Jacobian matrix, inverse of Jacobian,
            % derivatives w.r.t. x,y
            [Jacob,invJacobian,XYderivatives]=Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

            %  B matrix
            B=zeros(3,2*ndof);
            B(1,1:ndof)       = XYderivatives(:,1)';
            B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
            B(3,1:ndof)       = XYderivatives(:,2)';
            B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

            % stiffness matrix
            stiffness(elementDof,elementDof)=...
                stiffness(elementDof,elementDof)+...
                B'*C*thickness*B*gaussWeights(q)*det(Jacob);

        end;

    end;


else
    longterm=1;
```

```
        time=time+time_interval;
        timet=time/24/3600;
        if strcmpi(Code,'EC2')==1
            % Creep according to EC2
            betaC=(timet/(betaH+timet))^0.3;
            creep=betaC*phi0;
        else
            % Creep according to Model B3
            Zt=(t0^(0-m))*log(1+(timet)^n);
            Qt=Qf*(1+(Qf/Zt)^rt)^(0-1/rt);
            C0=q2*Qt+q3*log(1+(timet)^n)+q4*log((timet+t0)/t0);
            St=tanh(((timet+t0-ts)/tao_sh)^0.5);
            Htt=1-(1-RH)*St;
            Cd=q5*(exp(-8*Htt)-exp(-8*Ht))^0.5;
            Jt=q1+C0+Cd;
            creep=((Jt*E)/1000000-1);
        end

        for k=2:(time-1)/time_interval+1
            delt_str1=str1(k-1)*(exp(-time_interval/(eta1/E1))-1)+(time_interval/eta1)*exp(-
time_interval/(2*eta1/E1))*loadratio;
            str1(k)=str1(k-1)+delt_str1;
            delt_str2=str2(k-1)*(exp(-time_interval/(eta2/E2))-1)+(time_interval/eta2)*exp(-
time_interval/(2*eta2/E2))*loadratio;
            str2(k)=str2(k-1)+delt_str2;
            delt_str3=str3(k-1)*(exp(-time_interval/(eta3/E3))-1)+(time_interval/eta3)*exp(-
time_interval/(2*eta3/E3))*loadratio;
            str3(k)=str3(k-1)+delt_str3;
            delt_str4=str4(k-1)*(exp(-time_interval/(eta4/E4))-1)+(time_interval/eta4)*exp(-
time_interval/(2*eta4/E4))*loadratio;
            str4(k)=str4(k-1)+delt_str4;
            delt_str5=str5(k-1)*(exp(-time_interval/(eta5/E5))-1)+(time_interval/eta5)*exp(-
time_interval/(2*eta5/E5))*loadratio;
            str5(k)=str5(k-1)+delt_str5;
            delt_str6=str6(k-1)*(exp(-time_interval/(eta6/E6))-1)+(time_interval/eta6)*exp(-
time_interval/(2*eta6/E6))*loadratio;
            str6(k)=str6(k-1)+delt_str6;
        end
        str_visc=k/eta0*loadratio;
        straincrackrate=str_visc+str1(k)+str2(k)+str3(k)+str4(k)+str5(k)+str6(k);

        strainCR((time-1)/time_interval+1,:)=nodalstress(:,size(nodalstress',1))'*creep/E;
        strainRA((time-1)/time_interval+1,1)=straincrackrate;

        betacc=exp(0.25*(1-(28/time/24/3600)^.5));
        E=Einitial/(1+creep);%*betacc;
        C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

        % New Stiffness Matrix
        % ----------------------------------------------------------
        stiffness=zeros(GDof);
        for e=1:numberElements
            indice=elementNodes(e,:);
            elementDof=[ indice indice+numberNodes ];
            ndof=length(indice);

            % cycle for Gauss point
            for q=1:size(gaussWeights,1)
                GaussPoint=gaussLocations(q,:);
                xi=GaussPoint(1);
                eta=GaussPoint(2);

                % shape functions and derivatives
                shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
                    (1+xi)*(1+eta);(1-xi)*(1+eta)];
                naturalDerivatives=...
                    1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
                    1+eta,      1+xi;-(1+eta),   1-xi];

                % Jacobian matrix, inverse of Jacobian,
                % derivatives w.r.t. x,y
```

```
                    [Jacob,invJacobian,XYderivatives]=Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

                    %  B matrix
                    B=zeros(3,2*ndof);
                    B(1,1:ndof)       = XYderivatives(:,1)';
                    B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
                    B(3,1:ndof)       = XYderivatives(:,2)';
                    B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

                    % stiffness matrix
                    stiffness(elementDof,elementDof)=...
                        stiffness(elementDof,elementDof)+...
                        B'*C*thickness*B*gaussWeights(q)*det(Jacob);

                end;

            end;


        end

        Results(savingnr,:)=[midspandeflection(savingnr),...
            Reaction(savingnr)];%,stepnumber,cracklength];


    end %interation

end %stepnumber

plot(straintotal);
```