Implementation of a constitutive model for masonry shells as a stand-alone subroutine

Paulo B. Lourenço, Késio Palácio, Francisco Prieto

Report 02-DEC/E-13

The present research has been carried out under
contract GROW-1999-70420 "ISO-BRICK" from the European Commission
Task 1.1

Date: November 2002
No. of Pages: 50

Keywords: Fortran code, orthotropic model, validation, masonry shells

Escola de
Engenharia

Departamento de
Engenharia Civil

Universidade
do Minho

Azurém, 4800-085 Guimarães - Tel. +351 253 510 200 - Fax +351 253 510 217 - E-mail secG@civil.uminho.pt

Contents

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

1 Introduction

This report deals with the implementation of a macro-model for the out-of-plane behavior of masonry structures. The composite plasticity model proposed in Lourenço (2000) combines anisotropic elastic behavior with anisotropic plastic behavior and is formulated in modern algorithmic plasticity concepts. These include implicit Euler backward return mapping schemes and consistent tangent operators for all regimes of the model. The yield criterion combines the advantages of modern plasticity concepts with a powerful representation of anisotropic material behavior, which incorporates different hardening/softening behavior along each material axis. The model is thus capable of reproducing independent (in the sense of completely diverse) elastic and inelastic behavior along a prescribed set of material axes. The energy-based regularization technique, which is employed to obtain objective results with respect to mesh refinement, resorts then to four different fracture energies.

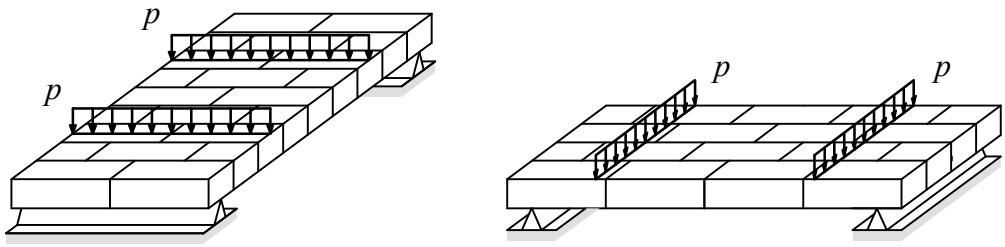
The numerical implementation of the model is evaluated by means of a comparison between numerical results and experimental results for the case of masonry panels with out-of-plane loading. The performance of the model is evaluated by means of a comparison with experimental results in reinforced masonry shells.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

2 Masonry Out-of-Plane Behavior

2.1 Uniaxial Behavior

The flexural strength of masonry has been mostly investigated in relation to the resistance of wall panels to wind loads. The flexural strength is, of course, different for bending perpendicular or parallel to the bed joints, see Figure 2.1, being normally several times larger when bending leads to failure in a plane perpendicular to the bed joints. In general, experimental results have been concerned solely with measurements of the flexural strength and mostly for the case when the plane of failure occurs parallel to the bed joints, while elastic and inelastic properties have usually been ignored.



(a) (b)

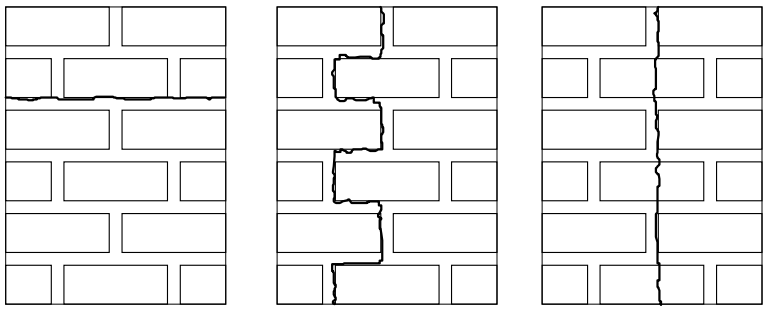
Figure 2.1 - Four-point bending test in two different directions: (a) plane of failure parallel to the bed joints – vertical bending or bending normal to the bed joints – and (b) plane of failure perpendicular to the bed joints – horizontal bending or bending parallel to the bed joints

In the case of bending leading to failure in a plane parallel to the bed joints ("vertical bending"), failure is generally caused by the relatively low tensile bond strength between the bed joints and the unit, see Figure 2.2a. In masonry with low strength units and greater tensile bond strength between the bed joints and the unit, e.g. high-strength mortar and units with numerous small perforations, which produce a dowel effect, failure may occur as a result of stresses exceeding the unit tensile strength. In any case, the typical moment-curvature relation is linear up to 70%-85% of the failure load, Lawrence (1983)

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

and van der Pluijm et al (1995), see Figure 2.3a. Beyond this level, micro-cracking starts to occur, associated with unloading ("softening") of the extreme tensile fibers.

For bending leading to failure in a plane perpendicular to the bed joints ("horizontal bending"), two different types of failure are common, depending on the relative strength of joints and units, see Figure 2.2b. In the first type of failure cracks zigzag through head and bed joints. The post-peak response of the specimen is governed by the fracture energy of the head joints and the friction behavior of bed joints. In the second type of failure cracks run almost vertically through the units and head joints. The post-peak response is governed by the fracture energy of the units and head joints. In both types of failure, the typical moment-curvature relation indicates a sudden decrease of stiffness before non-linear behavior starts to occur, see Figure 2.3b. This kind of behavior was originally noticed by Ryder (1963). Lawrence (1983) attributed this behavior to the gradual decrease of stiffness of the head joints, i.e. the softening behavior, which has been confirmed by van der Pluijm et al (1995). It is noted that the above holds true only in the general case of units with a tensile strength substantially larger than the tensile strength of the head joints.



(a) (b1) (b2)

Figure 2.2 - Possible failure modes for masonry subjected to bending along the material axes. Failure in a plane parallel to the bed joints: (a) "debonding". Failure in a plane perpendicular to the bed joints: (b1) "toothed" and (b2) "splitting"

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

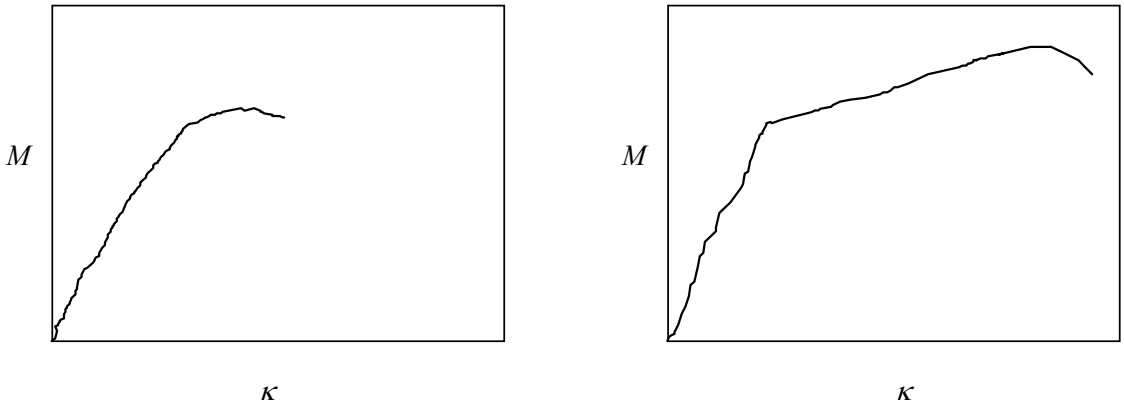
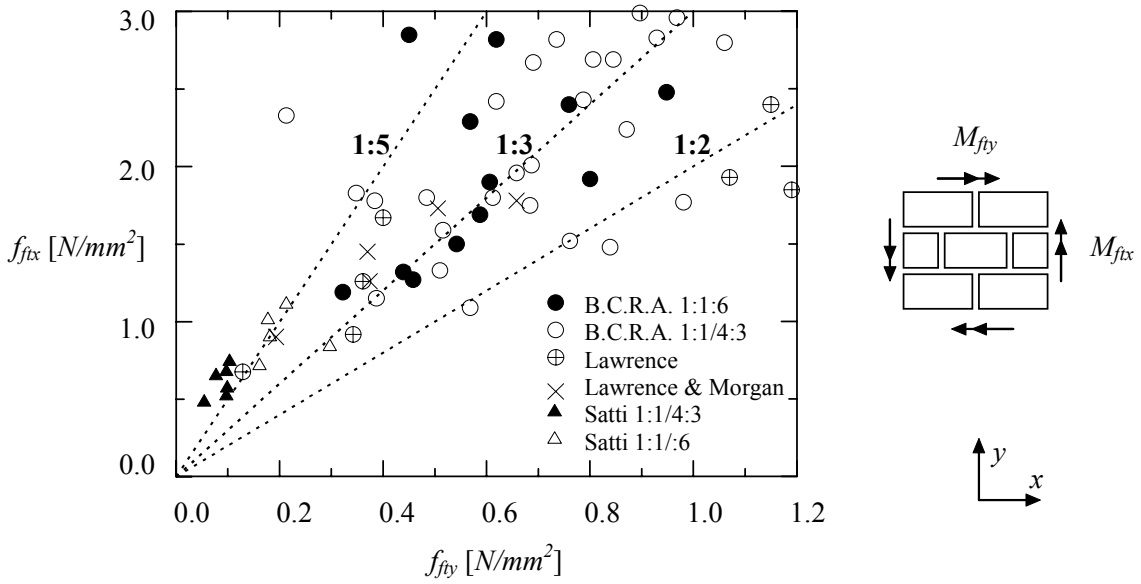


Figure 2.3 - Typical moment-curvature (M-k) diagrams for masonry in bending such that failure occurs in a plane (a) parallel and (b) perpendicular to the bed joints

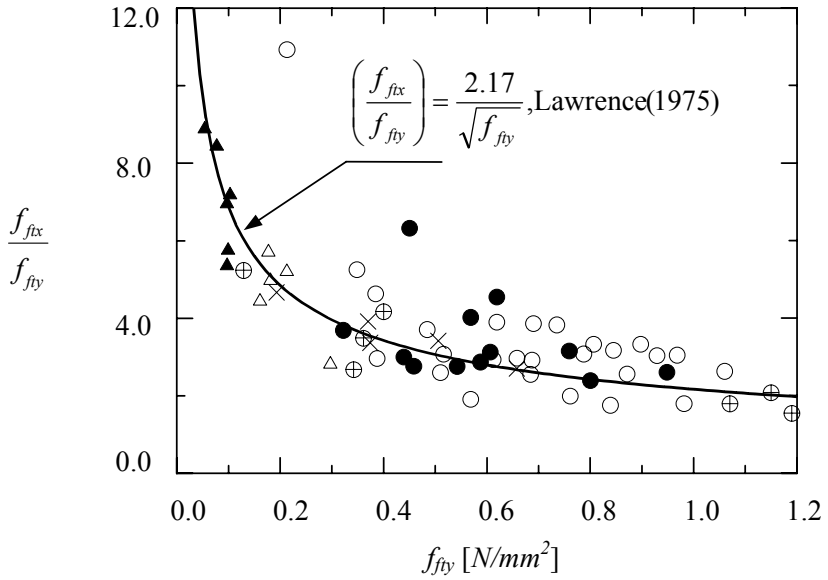
The ratio of flexural strength along the two material axes varies significantly as can be seen in Figure 2.4a. Nevertheless, a definite trend seems to be distinguished in the orthogonal strength ratio, which decreases markedly with an increasing flexural bond strength, see Figure 2.4b and also Baker (1979). It is noted that, a large scatter occurs and such a simplified assumption is hardly acceptable. The scatter is partly caused by:

- Different specimen and load configurations used to determine flexural strengths;
- The variability of the masonry constituents and workmanship. Ideally, each point should be established by averaging a convenient number of tests;
- Different flexural strengths of the units used. Since failure in bending parallel to the bed joints often occurs by unit breaking after degradation of the head joints strength, this parameter is important. This cannot be distinguished by such an empirical curve but a theoretical relationship that takes into account the flexural strength of the units can be found in Baker (1979).

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217



(a)



(b)

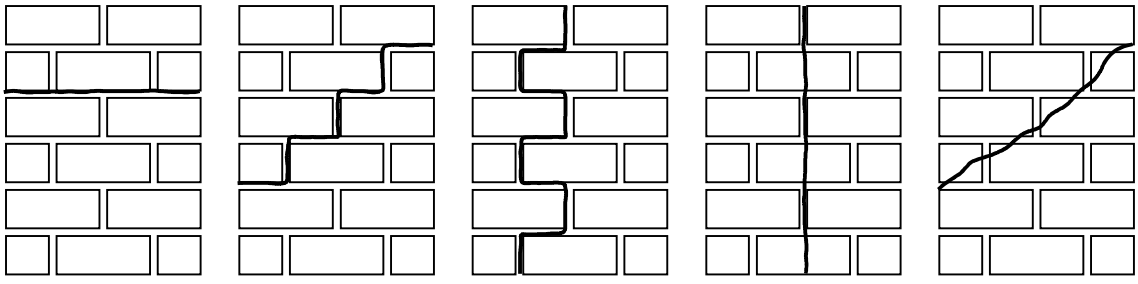
Figure 2.4 - Flexural tensile strength along the two material axes, Hendry (1990): (a) strength along x-axis vs. strength along y-axis; (b) strength ratio vs. strength along y-axis

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

2.2 Biaxial Behavior

There is relatively little knowledge about the flexural strength at different angles to the bed joints or about the interaction relationships for biaxial flexural tension. Losberg and Johansson (1969) present, probably, the first paper about four-point bending tests with masonry beams featuring different orientations with respect to the bed joints. This is a very complete article and a remarkable work for such early period. Satti and Hendry (1973) reported that the flexural strength at 45 degrees to the bed joints was approximately one half to a third of the strength in horizontal bending. In contrast, Cajdert and Losberg (1973) found that the diagonal flexural strength was higher than both the vertical and horizontal strengths. However, these early works were based on very few tests and there was a wide scatter of results.

Recently researchers have tried to gather the necessary experimental insight in biaxial bending, Gazzola and Drysdale (1986), Guggisberg and Thürlimann (1990) and van der Pluijm et al (1995). Gazzola and Drysdale (1986) and van der Pluijm et al (1995) performed similar tests, resorting to four-point bending masonry beams loaded in different orientations with respect to the material axes, but only van der Pluijm et al (1995) was able to measure and report moment-curvature diagrams. Guggisberg and Thürlimann (1990) resorted to a more complex test set-up capable of applying a general combination of bending moments. Figure 2.5 illustrates possible flexural tensile failure modes of masonry subject to out-of-plane loading.



(a) debonding (b) stepped (c) toothed (d) splitting (e) straight

Figure 2.5 - Possible flexure failure modes

Being masonry an anisotropic material, its complete out-of-plane behavior must be established in terms of, either, the three moment-components (M_{xx}, M_{yy} and M_{xy}), or, the

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

principal moments and one angle θ, which measures the orientation of the principal axes with respect to the material axes. It is noted that, in the general case of shell analysis, the material behavior must be described by five stress components (the usual assumption of thin structures assumes the out-of-plane normal stress to remain zero) and a yield criterion established solely in terms of principal moments is inadequate.

The experimental failure patterns of Gazzola and Drysdale (1986), in hollow concrete block masonry, are summarized in Figure 2.6. Five tests were carried out, for each orientation of the principal bending moment θ. Failure occurred, generally, in a combination of tension and shear, except for some cracking through the units observed for bending parallel to the bed joints. The average and standard deviation of the five tests, for each orientation of loading, are represented in Figure 2.7. It is observed that, in this particular testing program, the flexural strength for bending parallel to the bed joints is 2.5 times larger than the flexural strength for bending normal to the bed joints.

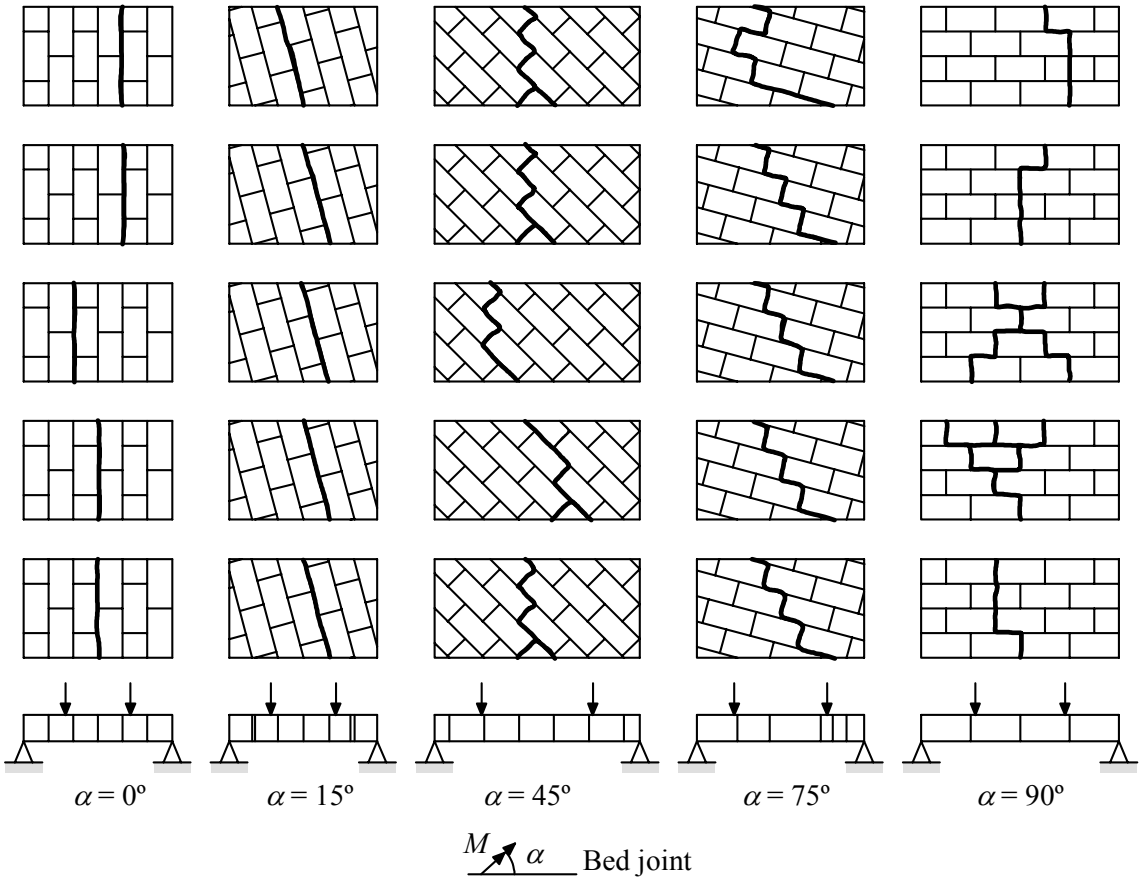


Figure 2.6 – Failure patterns in masonry wallettes, Gazzola and Drysdale (1986)

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

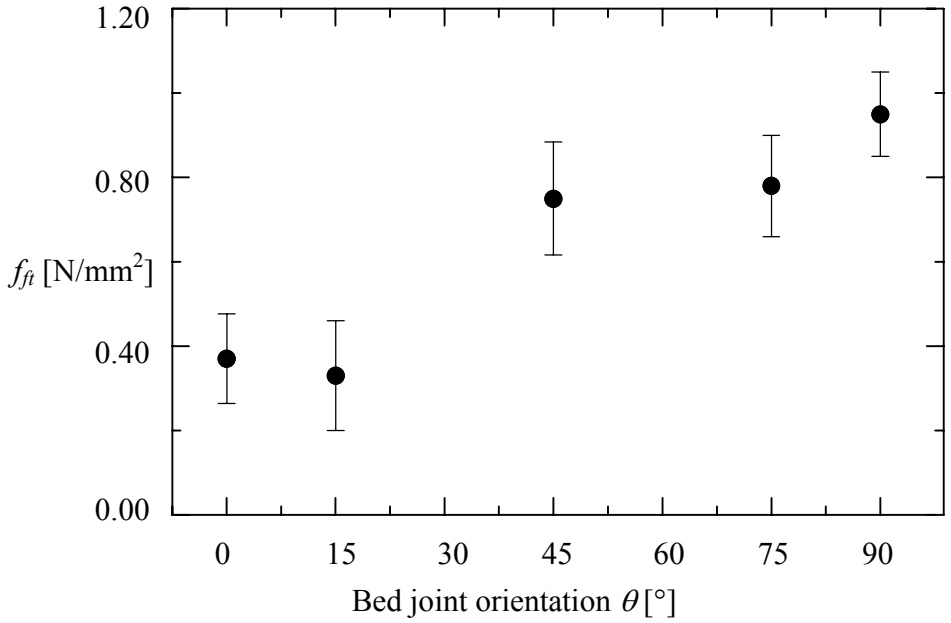


Figure 2.7 - Flexural strength for different orientations of loading, Gazzola and Drysdale (1986)

The results from van der Pluijm et al (1995), in solid brick clay masonry and calcium silicate block masonry, are summarized in Figure 2.8 and Figure 2.9 by the average and standard deviation of the tests, for each orientation of loading. Twelve tests were carried out, for each orientation of the principal bending moment θ. Failure occurred, generally, in a combination of tension and shear in the head and bed joints, except for loading making an angle of 70 degrees with the bed joints, where cracking through the units was also observed. It can be seen that, in this particular testing program, the flexural strength for bending parallel to the bed joints is approximately 4.0 times larger than the flexural strength for bending normal to the bed joints.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

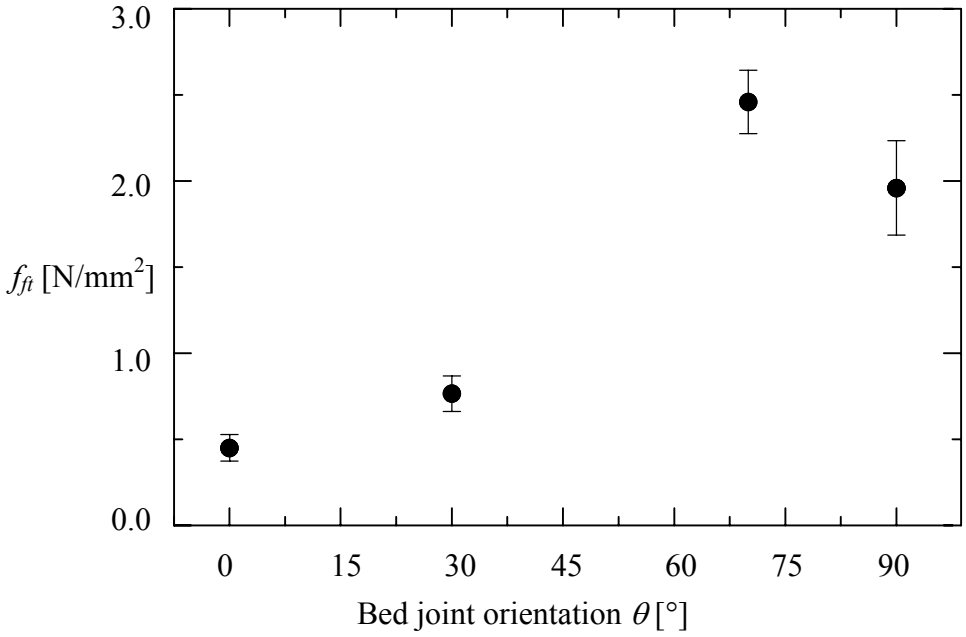


Figure 2.8 - Flexural strength for different orientations of loading in solid brick clay masonry, van der Pluijm et al (1995)

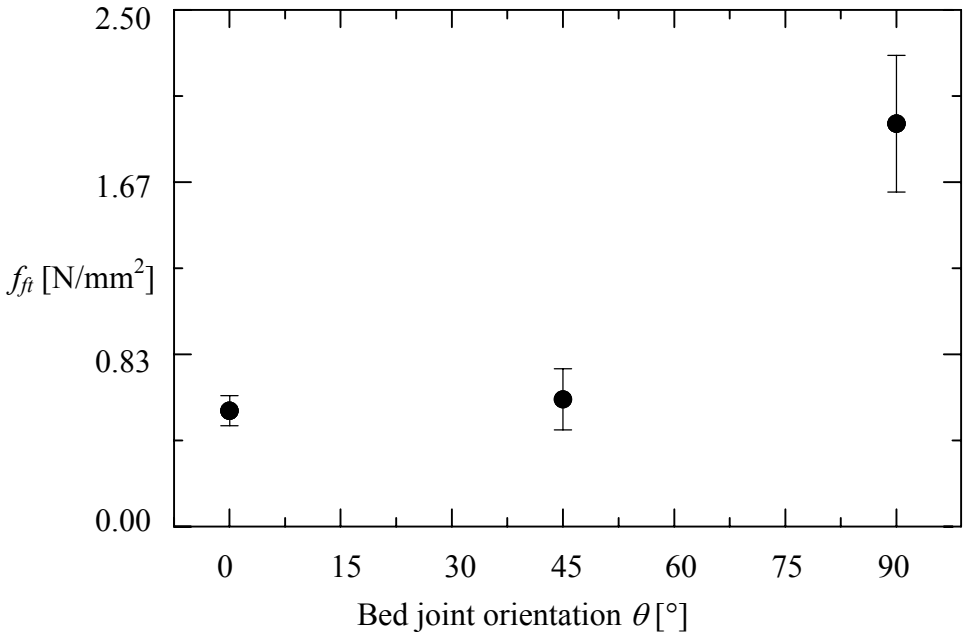


Figure 2.9 - Flexural strength for different orientations of loading in solid block calcium silicate masonry, van der Pluijm et al (1995)

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

3 Description of the Anisotropic Continuum Model

In this study, the problem of modeling laminar masonry structures (plates and shells with one dimension substantially smaller than the other two dimensions) is addressed. A typical hypothesis in this type of elements is "zero normal stress". This hypothesis states that the normal stress component perpendicular to the plane of the structure equals zero, and simplifies material modeling to a great extent.

3.1 Plates, shells and finite elements

Plates and shells are but a particular form of a three-dimensional solid, the treatment of which presents no theoretical difficulties. Owing to the symmetry of the stress and strain tensors, it is normal to collect the tensors components, see Figure 3.1, in vectors as

$$\begin{cases} \{\sigma\} = \{\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz}, \tau_{xz}\}^T \\ \{\varepsilon\} = \{\varepsilon_x, \varepsilon_y, \varepsilon_z, \gamma_{xy}, \gamma_{yz}, \gamma_{xz}\}^T \end{cases} \tag{3.1}$$

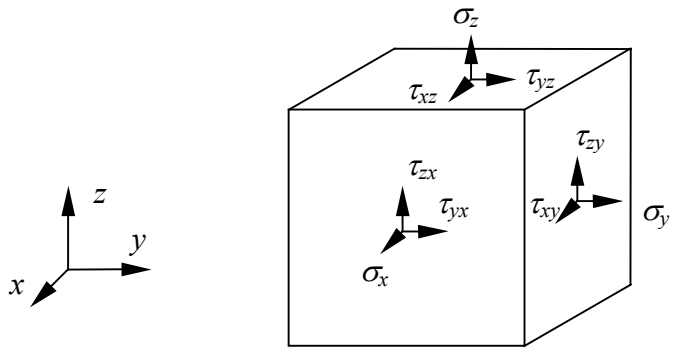


Figure 3.1 - Stress components in a three-dimensional body

However, the thickness of plates and shells (denoted in the following as *t*) is small when compared with other dimensions, and complete three-dimensional numerical treatment would be, in general, not only costly but in addition could lead to serious equation conditioning problems. To ease the solution several classical assumptions

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

regarding the behavior of such structures were introduced. Clearly such assumptions resulted in a series of approximations, which led, basically, to a thin plate theory and a thick plate theory. Here, no discussion is given regarding the approximations involved or the range of validity of the theories. For a comprehensive review of plate and shell bending approximations and the finite element implementation the reader is referred to Zienkiewicz and Taylor (1991).

Here, the finite element adopted to reduce the degrees-of-freedom in a complete three-dimensional analysis, is the curved shell element degenerated from a three-dimensional formulation. This element, originally proposed by Ahmad et al (1970) for the linear analysis of moderately thick shells, has been extensively used for the geometrical and non-linear analysis of shell structures. Typical characteristics of this element are the two hypotheses on which the degeneration is based: "straight normals" and "zero normal stress". The first hypothesis assumes that the normals to the mid-plane of the element remain straight after deformation, but not necessarily perpendicular to the mid-plane. The second hypothesis, already introduced above, states that the normal stress component perpendicular to the mid-plane equals zero, and the element formulation has been obtained ignoring the strain energy resulting from this stress. Assuming that the local z-axis represents the normal to the mid-plane, the five stress components left are σ_x, σ_y, τ_{xy}, τ_{yz} and τ_{xz}, see Figure 3.2.

Five degrees of freedom are defined for each element node: three translations and two rotations, see Figure 3.3. The definition of the independent translations and rotations includes the influence of shear deformation. The rotations are not coupled to the gradient of the mid-plane. In this study, two by two Gauss integration in the plane and seven-point Simpson integration in the thickness direction are used.

As the behavior of plates and shells is, typically, two-dimensional, it is possible to adopt a yield criterion developed for plane stress conditions enhanced with two new stress components. Therefore, the adopted yield criterion is based on the plane stress anisotropic yield criterion of Lourenço (1996), which includes a Hill type criterion for compression and a Rankine type criterion for tension, see Figure 3.4. The word type is used here because the yield criteria adopted are close to the original yield criteria, even if they represent solely a fit of experimental results.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

In the present implementation, there is no difference between the formulation for a complete three-dimensional stress-state and the formulation for shells, which is quite convenient for general purpose finite element programs. Therefore, in the present study, all the formulation will be given in the full six components stress-strain space. It suffices to use a Young's modulus E_z equal to zero, to retrieve the traditional five component formulation. The full model implementation is given in Annex A in the format of a Fortran file.

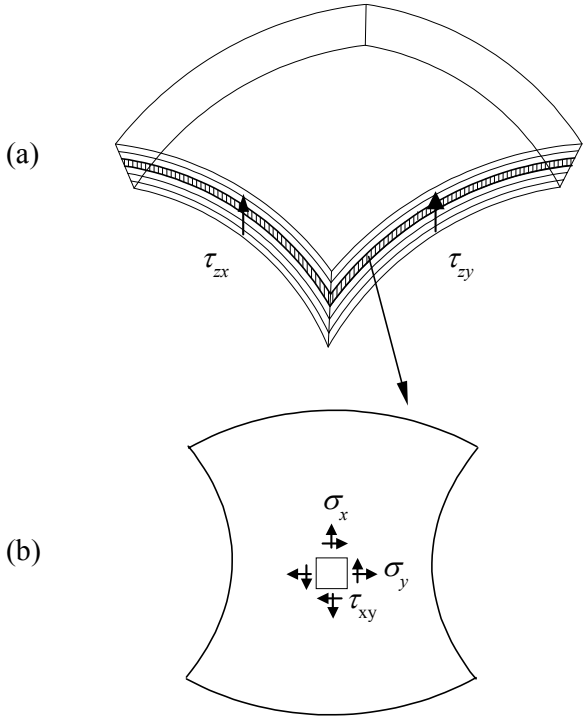


Figure 3.2 - Thin shells: (a) layered shell with five stress components; (b) layer, essentially, in plane stress conditions

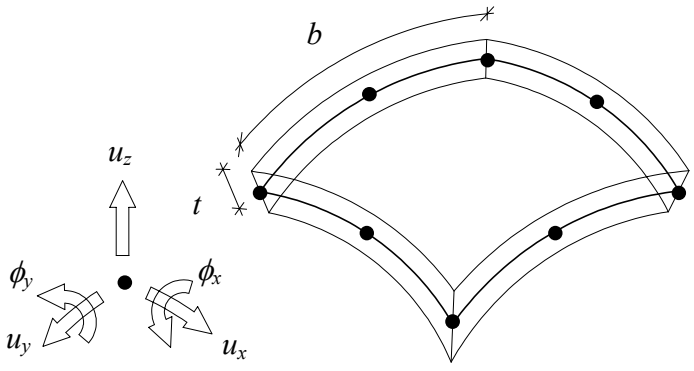


Figure 3.3 - Curved shell element (applicable for thin shells with t << b)

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

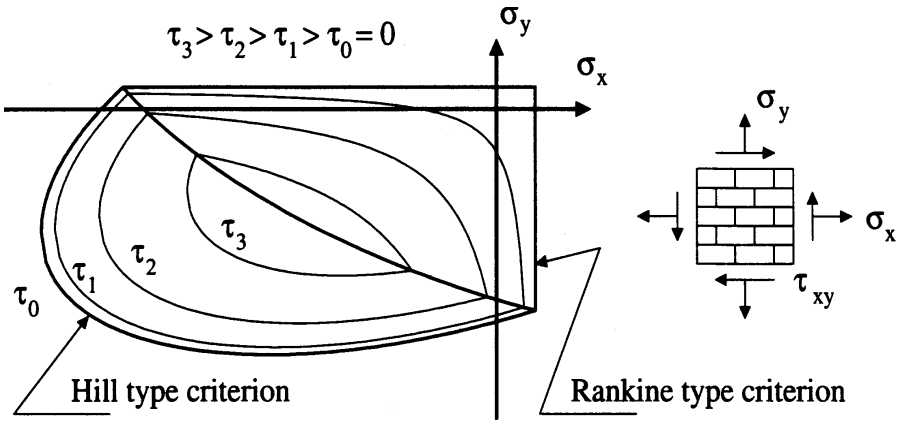


Figure 3.4 - Adopted plane stress composite yield criterion with iso-shear stress lines, Lourenço (1996)

3.2 The Rankine Type Criterion

An adequate formulation of the Rankine criterion is given by a single function, which is governed by the first principal stress and one yield value $\overline{\sigma}_t$ that describes the softening behaviour of the material as, see Feenstra and de Borst (1995),

$$f_1 = \frac{\sigma_x + \sigma_y}{2} + \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} - \overline{\sigma}_t(\kappa_t) \tag{3.2}$$

where the scalar κ_t controls the amount of softening. This expression can be rewritten as

$$f_1 = \frac{(\sigma_x - \overline{\sigma}_t(\kappa_t)) + (\sigma_y - \overline{\sigma}_t(\kappa_t))}{2} + \sqrt{\left(\frac{(\sigma_x - \overline{\sigma}_t(\kappa_t)) - (\sigma_y - \overline{\sigma}_t(\kappa_t))}{2}\right)^2 + \tau_{xy}^2} \tag{3.3}$$

where coupling exists between the stress components and the yield value. Setting forth a Rankine type criterion for an anisotropic material, with different tensile strengths along the x, y directions, is now straightforward if eq. (3.3) is modified to

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

$$f_1 = \frac{(\sigma_x - \overline{\sigma}_{tx}(\kappa_t)) + (\sigma_y - \overline{\sigma}_{ty}(\kappa_t))}{2} + \sqrt{\left(\frac{(\sigma_x - \overline{\sigma}_{tx}(\kappa_t)) - (\sigma_y - \overline{\sigma}_{ty}(\kappa_t))}{2}\right)^2 + \alpha\tau_{xy}^2} \quad (3.4)$$

where the parameter α, which controls the shear stress contribution to failure, reads

$$\alpha = \frac{f_{tx}\ f_{ty}}{\tau_{u,t}^2} \quad (3.5)$$

Here, f_{tx}, f_{ty} and $\tau_{u,t}$ are, respectively, the uniaxial tensile strengths in the x, y directions and the pure shear strength. Note that the material axes are now fixed with respect to a specific frame of reference. Thus, it shall be assumed that all stresses and strains for the elastoplastic algorithm are given in the material reference axes, see also Section 3.4.

Eq. (3.4) can be recast in a matrix form as

$$f_1 = (\tfrac{1}{2}\{\xi\}^T[\mathrm{P}_t]\{\xi\})^{\frac{1}{2}} + \tfrac{1}{2}\{\pi\}^T\{\xi\} \quad (3.6)$$

where the projection matrix $[\mathrm{P}_t]$ reads

$$[\mathrm{P}_t] = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.7)$$

the projection vector $\{\pi\}$ reads

$$\{\pi\} = \{1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}^T \quad (3.8)$$

the reduced stress vector $\{\xi\}$ reads

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

$$\{\xi\} = \{\sigma\} - \{\eta\} \tag{3.9}$$

the stress vector $\{\sigma\}$, as given in eq. (3.1), reads

$$\{\sigma\} = \{\sigma_x \quad \sigma_y \quad \sigma_z \quad \tau_{xy} \quad \tau_{yz} \quad \tau_{xz}\}^T \tag{3.10}$$

and the back stress vector $\{\eta\}$ reads

$$\{\eta\} = \{\sigma_{tx}(\kappa_t) \quad \sigma_{ty}(\kappa_t) \quad 0 \quad 0 \quad 0 \quad 0\}^T \tag{3.11}$$

Exponential tensile softening is considered for both equivalent stress-equivalent strain diagrams, with different fracture energies (G_{fx} and G_{fy}) for each yield value, which read

$$\overline{\sigma}_{tx} = f_{tx} \exp(-\frac{h f_{tx}}{G_{fx}} \kappa_t) \quad \text{and} \quad \overline{\sigma}_{ty} = f_{ty} \exp(-\frac{h f_{ty}}{G_{fy}} \kappa_t) \tag{3.12}$$

where the standard equivalent length h is related to the element size, Bazant and Oh (1983).

A non-associated plastic potential g_1

$$g_1 = (\tfrac{1}{2}\{\xi\}^T [\mathrm{P}_g]\{\xi\})^{\frac{1}{2}} + \tfrac{1}{2}\{\pi\}^T \{\xi\} \tag{3.13}$$

is considered, where the projection matrix $[\mathrm{P}_g]$ represents the original Rankine plastic flow, i.e. $\alpha = 1$ in eq. (3.7). The inelastic behaviour is described by a strain softening hypothesis given by the maximum principal plastic strain $\dot{\varepsilon}_t^p$ as

$$\dot{\kappa}_t = \dot{\varepsilon}_1^p = \frac{\dot{\varepsilon}_x^p + \dot{\varepsilon}_y^p}{2} + \tfrac{1}{2}\sqrt{(\dot{\varepsilon}_x^p - \dot{\varepsilon}_y^p)^2 + (\dot{\gamma}_{xy}^p)^2} \tag{3.14}$$

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

which reduces to the particularly simple expression

$$\dot{\kappa}_t = \dot{\lambda}_t \tag{3.15}$$

3.3 The Hill Type Criterion

The simplest yield criterion that features different compressive strengths along the two material axes is a rotated centred ellipsoid in the full stress space. The expression for such a quadric can be written as

$$f_2 = \frac{\overline{\sigma}_{cy}(\kappa_c)}{\overline{\sigma}_{cx}(\kappa_c)}\sigma_x^2 + \beta\sigma_x\sigma_y + \frac{\overline{\sigma}_{cx}(\kappa_c)}{\overline{\sigma}_{cy}(\kappa_c)}\sigma_y^2 + \gamma(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{xz}^2) - \overline{\sigma}_{cx}(\kappa_c)\,\overline{\sigma}_{cy}(\kappa_c) = 0 \tag{3.16}$$

where $\overline{\sigma}_{cx}(\kappa_c)$ and $\overline{\sigma}_{cy}(\kappa_c)$ are, respectively, the yield values along the material axes *x* and *y*. β and γ values are additional material parameters that determine the shape of the yield criterion. The parameter β controls the coupling between the normal stress values, i.e. rotates the yield criterion around the shear stress axis, and must be obtained from one additional experimental test, e.g. biaxial compression with a unit ratio between principal stresses. The parameter γ, which controls the shear stresses contributions to failure, can be obtained from

$$\gamma = \frac{f_{cx}\, f_{cy}}{\tau_{u,c}^2} \tag{3.17}$$

where f_{cx} , f_{cy} and $\tau_{u,c}$ are, respectively, the uniaxial compressive strengths in the *x*, *y* directions and a fictitious pure shear in compression.

For the purpose of numerical implementation, it is convenient to recast this yield criterion in a matrix form as

$$f_2 = (\tfrac{1}{2}\{\sigma\}^T[\mathrm{P}_c]\{\sigma\})^{\frac{1}{2}} - \overline{\sigma}_c(\kappa_c) \tag{3.18}$$

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

where the projection matrix $[P_c]$ reads

$$[P_c] = \begin{bmatrix} 2\dfrac{\overline{\sigma}_{cy}(\kappa_c)}{\overline{\sigma}_{cx}(\kappa_c)} & \beta & 0 & 0 & 0 & 0 \\ \beta & 2\dfrac{\overline{\sigma}_{cx}(\kappa_c)}{\overline{\sigma}_{cy}(\kappa_c)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\gamma & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\gamma & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\gamma \end{bmatrix} \tag{3.19}$$

the yield value $\overline{\sigma}_c$ is given by

$$\overline{\sigma}_c(\kappa_c) = \sqrt{\overline{\sigma}_{cx}(\kappa_c)\ \overline{\sigma}_{cy}(\kappa_c)} \tag{3.20}$$

and the scalar κ_c controls the amount of hardening and softening.

The inelastic law adopted comprehends parabolic hardening followed by parabolic/exponential softening for both equivalent stress-equivalent strain diagrams, with different compressive fracture energies (G_{fcx} and G_{fcy}) along the material axes, see Lourenço (1996). The problem of mesh objectivity of the analyses with strain softening materials is a well debated issue, at least for tensile behaviour, and the stress-strain diagram must be adjusted according to an equivalent length h to provide an objective energy dissipation, see Feenstra and de Borst (1996).

An associated flow rule and a work-like hardening/softening hypothesis are considered. This yields

$$\dot{\kappa}_c = \frac{1}{\overline{\sigma}_c}\{\sigma\}^T\{\dot{\varepsilon}^p\} = \dot{\lambda}_c \tag{3.21}$$

3.4 Orientation of the Material Axes

For the sake of simplicity, the formulation of the plasticity model was presented based on the assumption that the principal axes of anisotropy coincided with the frame of

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

reference (local or global) for stresses and strains in finite element computations. Since this is not necessarily the case, such non-alignment effects must be taken into account.

Two different approaches can be followed. In the first approach, with each call to the plasticity model, stresses, strains and, finally, consistent tangent stiffness matrices must be rotated into and out of the material frame of reference, respectively, as pre- and post-processing. In the second approach, before the analysis begins, the elastic stiffness matrix [D], the projection matrices [Pt], [Pg] and [Pc], and the projection vectors {p} and {h} must be rotated from the material frame of reference into the global frame of reference at each quadrature point, eliminating the need of all subsequent rotation operations. The drawback of the latter approach is that the matrices then lose their sparse nature, resulting in less clear algorithms. For this reason, the plasticity model is implemented employing the former option.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

4 Validation

4.1 McMaster University hollow concrete block panels

The first series of panels analysed consists of hollow concrete block masonry. The tests were carried out by Gazzola *et al* (1985) and are denoted by W. The inelastic properties of the composite material are obtained from Gazolla *et al* (1985), see Table 4.1 and Table 4.2. Only tensile inelastic behaviour is considered in the analysis and the fracture energy values indicated include already the size h of the element, $g_f = G_f / h$. The elastic properties were not available and had to be estimated. The Young's modulus in the vertical direction has been calculated from the masonry compressive strength according to Hendry (1990), the elastic orthotropy is assumed to equal two and the adopted in plane Poisson's ratio is 0.2. In order to remove the effect of the z-axis, the following approximately zero values were assumed: E_z = 1.d-3, ν_{xz} = 1.d-7 and ν_{yz} = 1.d-7.

Table 4.1 - Material properties for McMaster University panels

E_x	E_y	E_z	ν_{xy}	ν_{yz}	ν_{xz}	G_{xy}	G_{yz}	G_{xz}
5000	10000	1.d-3	0.2	1.d-7	1.d-7	3125	3125	3125
N/mm^2	N/mm^2	N/mm^2	-	-	-	N/mm^2	N/mm^2	N/mm^2

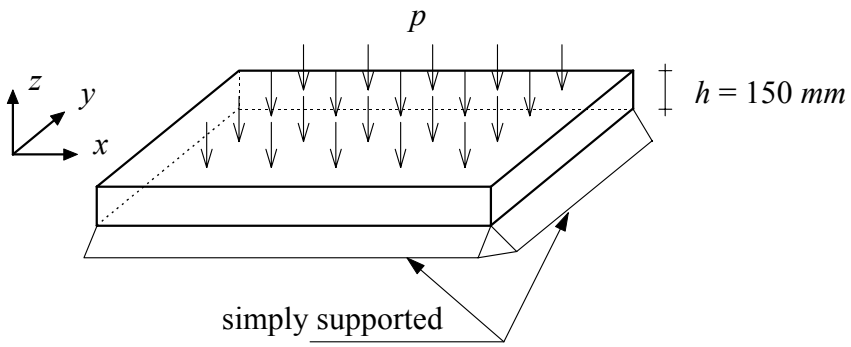
Table 4.2 - Material properties for McMaster University panels

Tension regime				
f_{tx}	f_{ty}	α	g_{fx}	g_{fy}
0.94	0.35	0.4	0.0012	0.00008
N/mm^2	N/mm^2	-	N/mm^2	N/mm^2

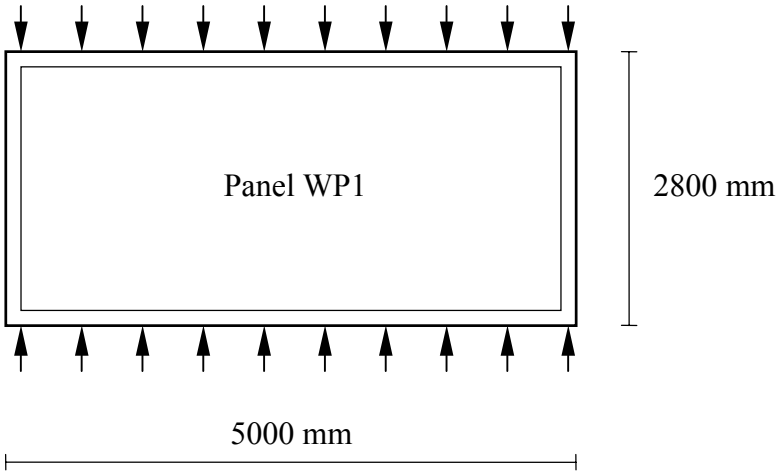
Only panel WP1, see Figure 4.1, was analysed with the composite plasticity model. The panel was loaded until failure with increasing out-of-plane uniform pressure p. This panel was loaded, previously to the application of the out-of-plane loading, with an in-plane confining pressure of 0.2 N/mm^2, which was kept constant until failure of the

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

specimen due to the pressure p. For each configuration, three different tests have been carried out and the results reported in Gazzola *et al* (1985) represent the average of the tests.



(a)



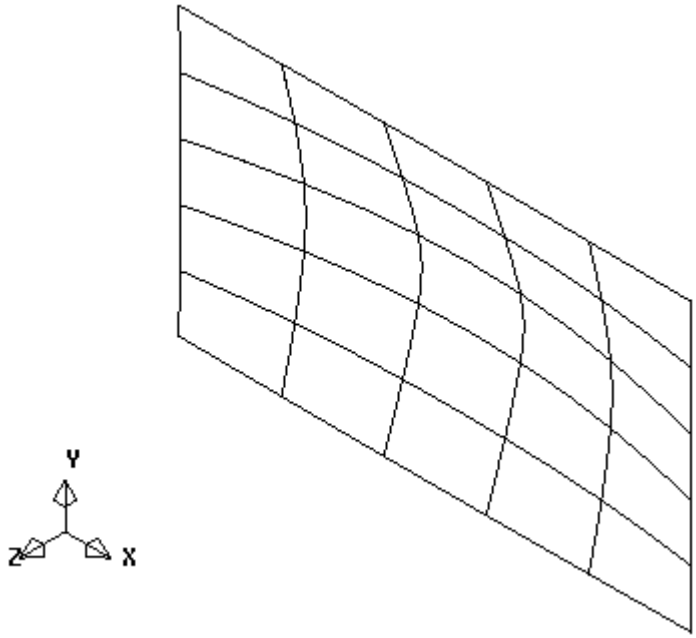
(b)

Figure 4.1 - Panels with constant out-of-plane pressure p, Gazzola *et al* (1985): (a) out-of-plane loading; (b) geometry and in-plane loading.

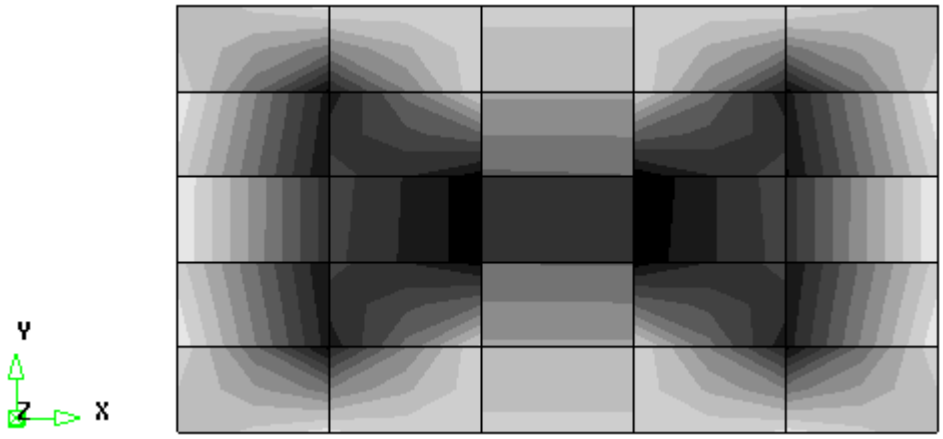
For the numerical analyses eight-noded degenerated shell elements with two by two in-plane Gauss integration and seven-point Simpson through thickness integration are used. Due to the double symmetry of the structure only one quarter (or one half) of the structure needs to be modelled. Nevertheless, for easiness in the post-processing and to obtain more clear pictures, the full structure is modelled. For this purpose, a regular mesh of five by five rectangular elements is used in all the analyses. The analyses are carried out with arc-length control enhanced with line searches, aiming at a globally convergent algorithm.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

The panel analysed, denoted by WP1, has dimensions 5000 × 2800 × 150 mm^3. The panel is simply supported in the four edges. Figure 4.2 and Figure 4.3 illustrate the behaviour of the panel at peak load and ultimate load. The development of typical failure lines, similar to the yield line theory of slabs is clearly visible in Figure 4.3. At peak, the failure lines are not fully formed and cannot be obtained.



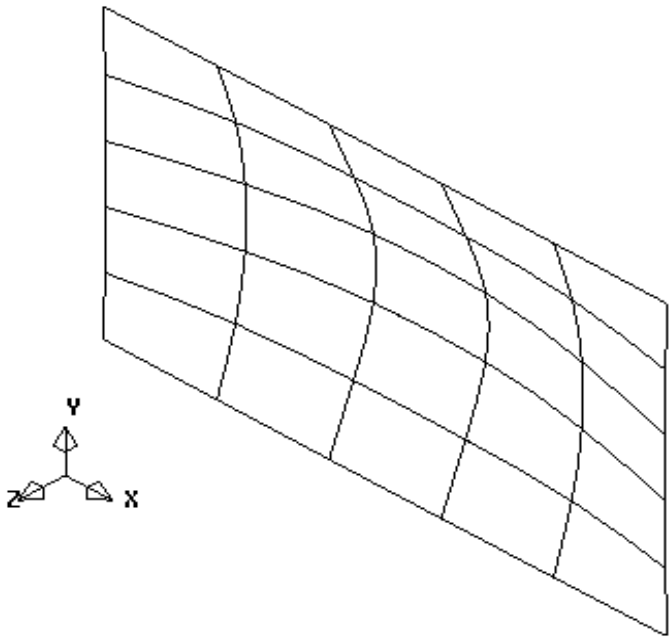
(a)



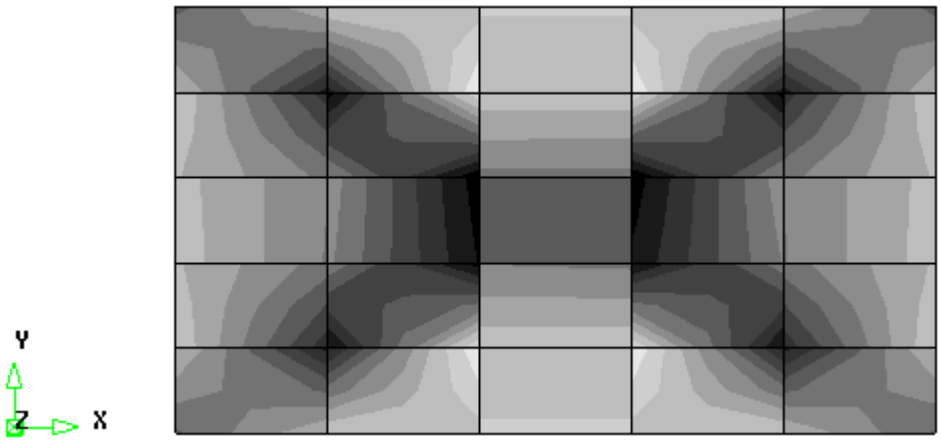
(b)

Figure 4.2 - Panel WP1. Results of the analysis at a pressure *p* equal to 9.45 kPa (peak): (a) total deformed mesh; (b) maximum principal strain

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217



(a)



(b)

Figure 4.3 - Panel WP1. Results of the analysis at a pressure *p* equal to 8.20 kPa (87% post-peak): (a) total deformed mesh; (b) maximum principal strain

The calculated load-displacement diagram for a displacement *d* at the centre of the specimen and the experimental collapse load are given in Figure 4.4. An extremely ductile behaviour is observed and reasonable agreement is found with the experimental value.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

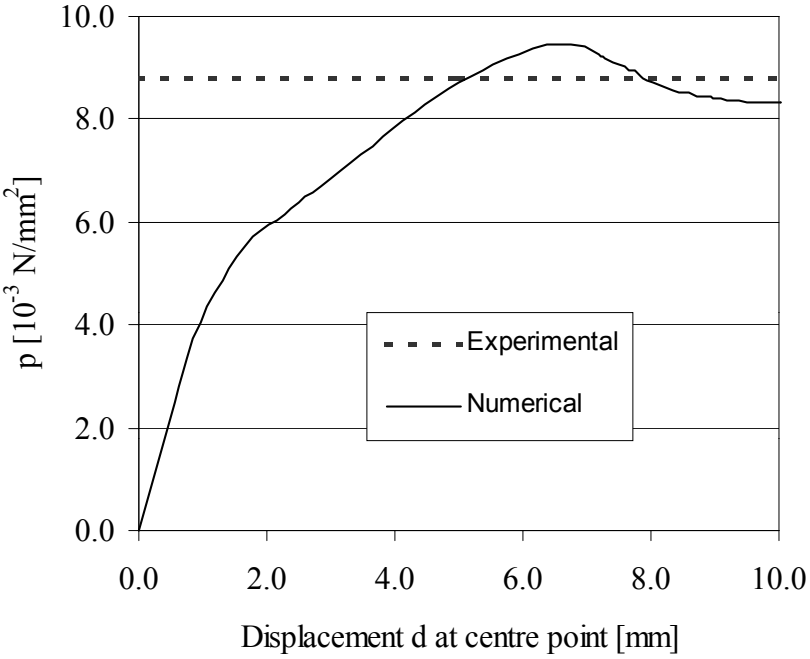


Figure 4.4 - Panels WP1. Load-displacement diagram and experimental failure load (average of three tests).

4.2 University of Plymouth solid brick clay masonry panels

The second series considered here consists of a set of solid clay brick masonry panels submitted to out-of-plane loading. The tests were carried out by Chong *et al* (1995) and are denoted by SB. The elastic and inelastic properties of the composite material are given in Table 4.3 and Table 4.4. Only tensile inelastic behaviour is considered and, following Chong *et al* (1995), isotropic material properties are assumed (this can be accepted because the panel is built with solid brick masonry). The adopted Poisson's ratio is 0.2 and the Young's modulus has been obtained by inverse fitting. Please note that, in order to remove the effect of the *z*-axis, orthotropic material data must be introduced with the following approximately zero values were assumed: E_z = 1.d-3, ν_{xz} = 1.d-7 and ν_{yz} = 1.d-7. The uniaxial tensile strengths and the fracture energies along the material axes g_{fx} and g_{fy} are then calculated so that the correct flexural strength is obtained in a pure bending test, see Lourenço (2000). The parameter α has been obtained by inverse fitting.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

Table 4.3 - Elastic material properties for University of Plymouth panels

E_x	E_y	E_z	ν_{xy}	ν_{yz}	ν_{xz}	G_{xy}	G_{yz}	G_{xz}
14000	14000	1.d-3	0.2	1.d-7	1.d-7	5833	5833	5833
N/mm^2	N/mm^2	N/mm^2	-	-	-	N/mm^2	N/mm^2	N/mm^2

Table 4.4 - Inelastic material properties for University of Plymouth panels

Tension regime				
f_{tx}	f_{ty}	α	g_{fx}	g_{fy}
1.37	0.58	1.1	0.00589	0.00071
N/mm^2	N/mm^2	-	N/mm^2	N/mm^2

The panel analysed here, denoted by SB02, yields an additional assessment of the ability of the composite plasticity model to reproduce the behaviour of a panel with an opening of $2260 \times 1125\ mm^2$. Its dimensions are $5600 \times 2475 \times 100\ mm^3$, built in stretcher bond between two stiff abutments with the vertical edges simply supported (allowance for in-plane displacements was provided) and the top edge free. A fixed support was provided at the base because of practical difficulties in providing a simple support. The opening size and dimensions used in the test were chosen to be representative of those used in practice, see Figure 4.5. The panel was loaded by air-bags until failure with increasing out-of-plane uniform pressure p. The air pressure and the displacement d for the middle point of the free edge were monitored during testing.

For the numerical analyses eight-noded degenerated shell elements with two by two in-plane Gauss integration and seven-point through thickness integration are used. Due to the symmetry of the structure only one half or one quarter of the panel needs to be modelled. Nevertheless, for easiness in the post-processing and to obtain more clear pictures, the full structure is modelled. The analysis is carried out with arc-length control enhanced with line searches, aiming at a globally convergent algorithm.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

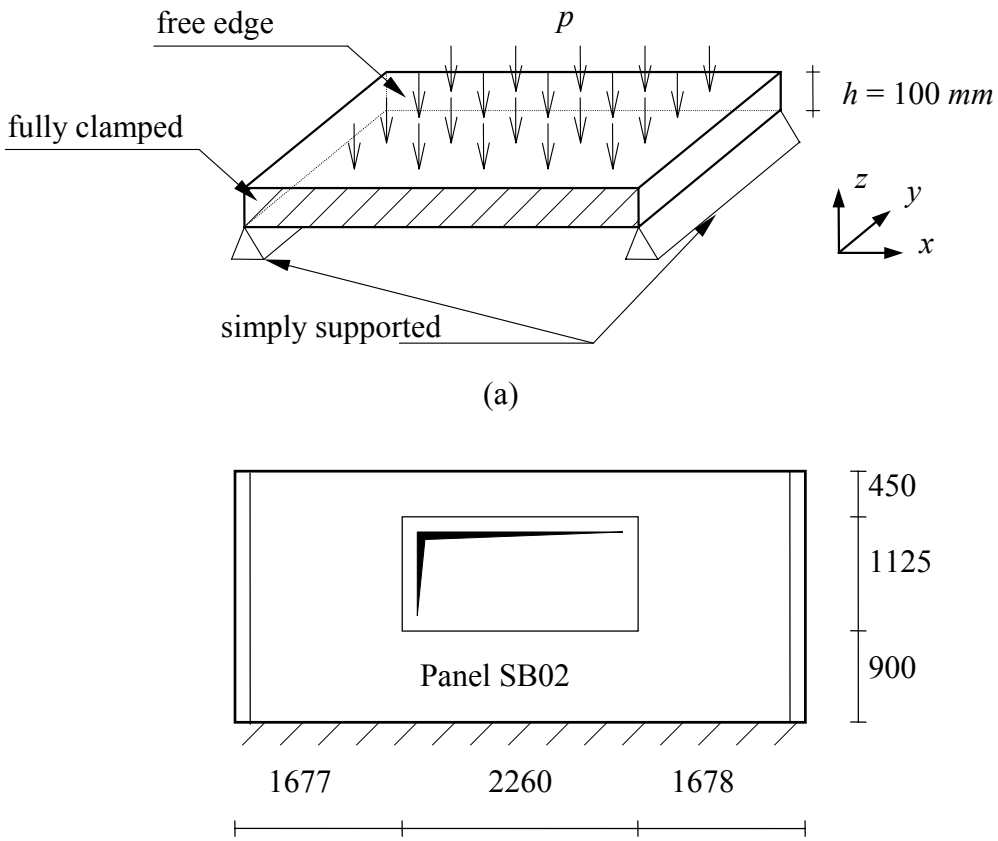
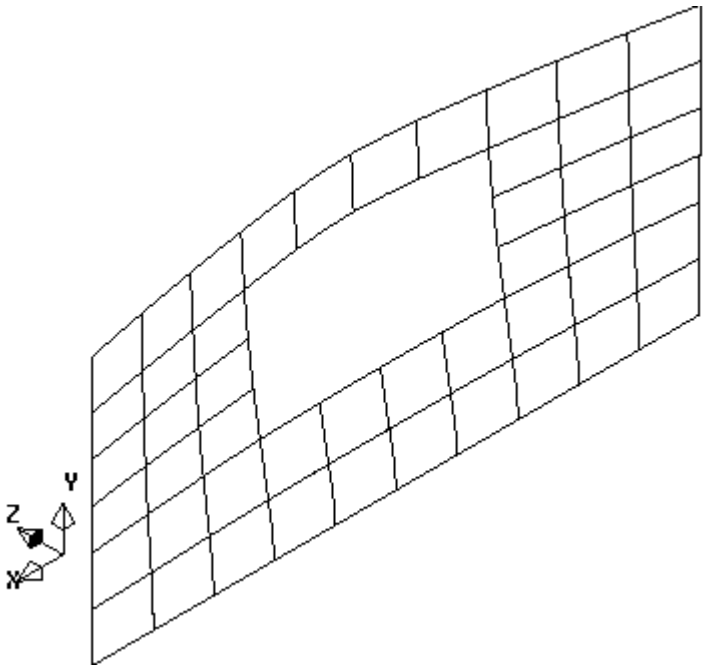


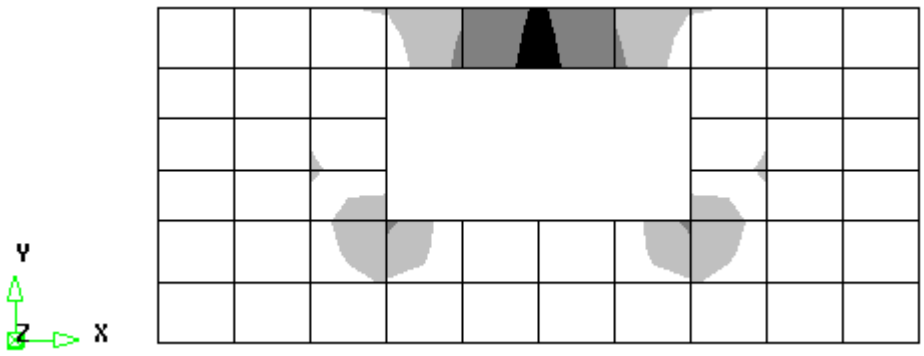
Figure 4.5 - Panels SB02, Chong *et al* (1995): (a) out-of-plane loading; (b) geometry, with dimensions in [mm].

Figure 4.6 and Figure 4.7 illustrate the behaviour of the panel at peak load and ultimate load. Cracking is initiated in the middle of the lintel above the central opening, with propagation to the bottom corners. These results are in agreement with the failure mode found experimentally, see Figure 4.8.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217



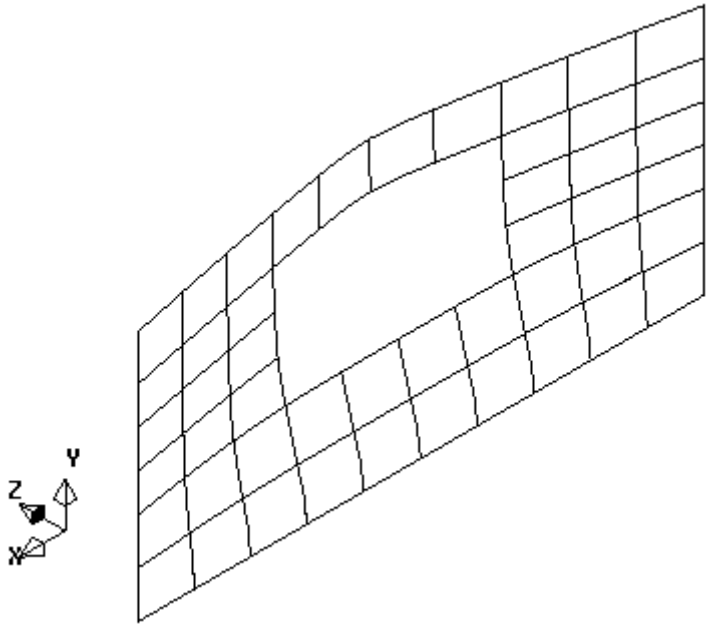
(a)



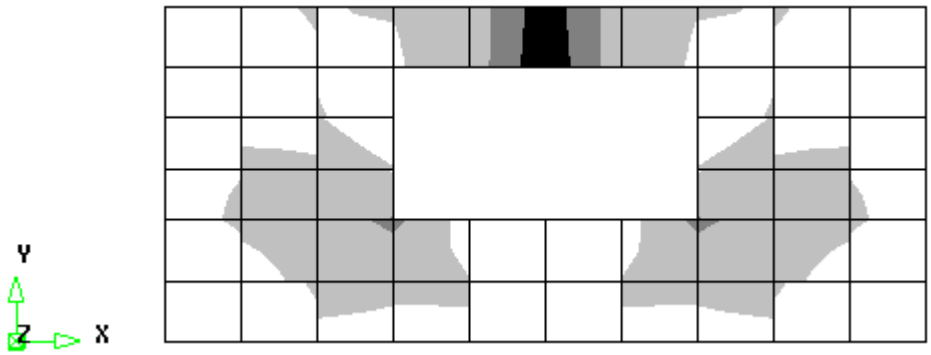
(b)

Figure 4.6 - Panel SB02. Results of the analysis at a pressure *p* equal to 2.15 kPa (peak): (a) total deformed mesh; (b) maximum principal strain

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217



(a)



(b)

Figure 4.7 - Panel SB02. Results of the analysis at a pressure *p* equal to 1.56 kPa (ultimate): (a) total deformed mesh; (b) maximum principal strain

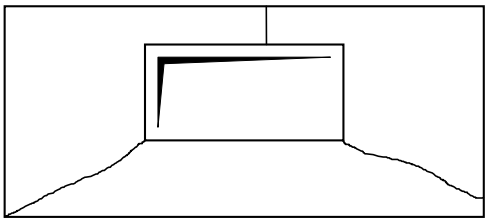


Figure 4.8 – Experimental failure mode.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

The calculated and numerical load-displacement diagrams for the displacement *d* at the middle of the top (free) edge of the specimen are given in Figure 4.9. Reasonable agreement is found.

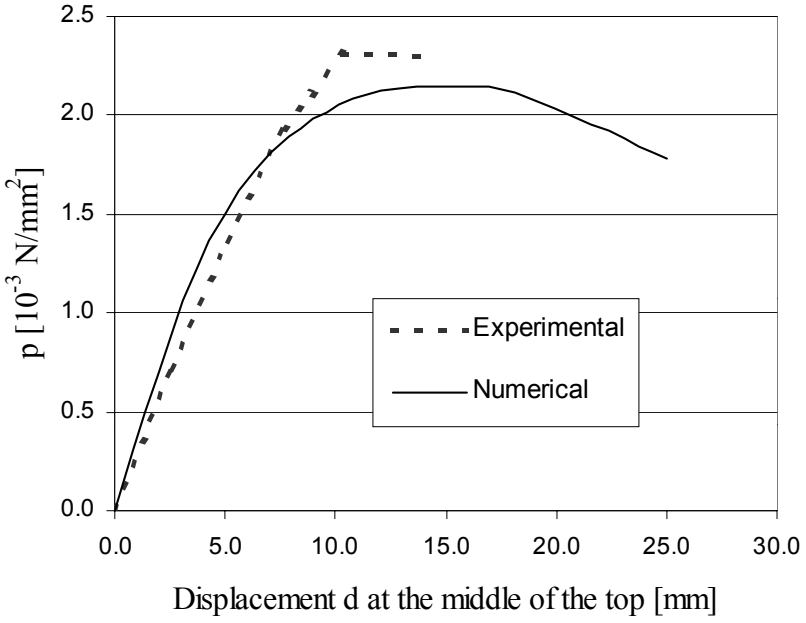


Figure 4.9 - Panel SB02. Load-displacement diagrams.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

5 Conclusions

An anisotropic model for masonry plates and shells was implemented and assessed. The model can be utilized both with shell elements and three-dimensional elements.

The numerical calculations were compared with results available in the literature for tests in masonry panels subjected to out-of-plane loading. Reasonable agreement was found in all cases. The tests were made of different masonry types and were subjected to different loading conditions. It is, therefore, possible to conclude that the proposed model is of general application for modeling masonry plates and shells.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

6 References

AHMAD, S., IRONS, B.M. and ZIENKIEWICZ, O.C. (1970) - Analysis of thick and thin shell structures by curved finite elements, Int. J. Numer. Methods Engrg., 2, p. 419-451.

BAKER, L.R. (1979) - A failure criterion for brickwork in biaxial bending, in: Proc. Fifth Int. Brick Mas. Conf., Washington, USA, p. 71-78.

BAZANT, Z.P. and OH, B.H. (1983) - Crack band theory for fracture of concrete, Materials and Structures, RILEM, 93(16), p. 155-177.

CAJDERT, A. and LOSBERG, A. (1973) - Laterally loaded light expanded clay block masonry: The effect of reinforcement in horizontal joints, in: Proc. Third Int. Brick Mas. Conf., Essen, Germany, p. 245-251.

CHONG, V.L., SOUTHCOMBE, C. and MAY, I.M. (1995) - The behaviour of laterally loaded masonry panels with openings, in: Proc. Fourth. Int. Mas. Conf., London, UK, p. 178-182.

FEENSTRA, P.H. and de BORST, R. (1995) - A plasticity model and algorithm for mode-I cracking in concrete, Int. J. Numer. Methods Engrg., 38, p. 2509-2529.

FEENSTRA, P.H. and de BORST, R. (1996) - A composite plasticity model for concrete, Int. J. Solids Structures, 33(5), p. 707-730.

GAZZOLA, E.A., DRYSDALE, R.G. and ESSAWY, A.S. (1985) - Bending of concrete masonry walls at different angles to the bed joints, in: Proc. Third North Amer. Mas. Conf., Arlington, Texas, USA, Paper 27.

GAZZOLA, E.A. and DRYSDALE, R.G. (1986) - A component failure criterion for blockwork in flexure, in: Structures ‘86, ASCE, edited by S.C. Anand, New Orleans, Louisiana, USA, p. 134-153.

GUGGISBERG, R. and THÜRLIMANN, B. (1990) - Failure criterion for laterally loaded masonry walls, in: Proc. Fifth North Amer. Mas. Conf., Urbana-Champaign, Illinois, USA, p. 949-958.

HENDRY, A.W. (1990) - Structural masonry, MacMillan Education, London, UK.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

LAWRENCE, S.J. (1983) - Behaviour of brick masonry walls under lateral loading, Dissertation, School of Civil Engineering, University of New South Wales, Australia.

LOSBERG, A. and JOHANSSON, S. (1969) - Sideway pressure on masonry walls of brickwork, in: Proc. Int. Symp. Bearing Walls, Warsaw, Poland, Paper 29.

LOURENÇO, P.B. (1996) - Computational strategies for masonry structures, Dissertation, Delft University of Technology, Delft, The Netherlands.

LOURENÇO, P.B. (2000) - Anisotropic softening model for masonry plates and shells, J. Struct. Engrg., ASCE, 126(9), p.1008-1016.

VAN DER PLUIJM, R., RUTTEN, H.S. and SCHIEBROEK, C.S. (1995) - Flexural behaviour of masonry in different directions, in: Proc. Fourth. Int. Mas. Conf., London, UK, p. 117-123.

RYDER, J.F. (1963) - The use of small brickwork panels for testing mortars, Trans. Brit. Ceram. Soc., 62.

SATTI, K.M.H. and HENDRY, A.W. (1973) - The modulus of rupture of brickwork, in: Proc. Third Int. Brick Mas. Conf., Essen, Germany, p. 155-160.

ZIENKIEWICZ, O.C. and TAYLOR, R.L. (1991) - The finite element method - Volume 2: Solid and fluid mechanics; Dynamics and non-linearity, McGraw-Hill, Berkshire, England, U.K.

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

ANNEX A

Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c
c     TESTING UNIAXIAL TENSION
c
      PROGRAM TEST
      INTEGER          nstr, NUV, NUS, ITER
      parameter        (      nstr = 4, nuv = 14, nus = 2)
      DOUBLE PRECISION DEPS(nstr), se(nstr,nstr), USRVAL(NUV),
     $                 USRSTA(NUS), tsig(nstr), yt(nstr*nstr),
     $                 youn(2)
c
      iter = 0
c
      deps(1) =  2.d-5
      deps(2) = -4.d-6
      deps(3) = 0.
      deps(4) = 0.
c
c     Exx = 10000.;  Eyy = 5000.;  Ezz = 1.d-3
c     nu_xy = 0.2 ;  nu_yz = 1.d-7;  nu_xz = 1-d-7
c     Gxy = 3000.
      se(1,1) = 10200.
      se(1,2) = 1020.
      se(1,3) = 1.224d-10
      se(1,4) = 0.
      se(2,1) = se(1,2)
      se(2,2) = 5102.
      se(2,3) = 1.122d-10
      se(2,4) = 0.
      se(3,1) = se(1,3)
      se(3,2) = se(2,3)
      se(3,3) = 1.d-3
      se(3,4) = 0.
      se(4,1) = se(1,4)
      se(4,2) = se(2,4)
      se(4,3) = se(3,4)
      se(4,4) = 3.d3
c
      tsig(1) = 1.d0
      tsig(2) = 8.d-6
      tsig(3) = 1.d-14
      tsig(4) = 0.
c
      usrval(1) = 1.0
      usrval(2) = 0.0002
      usrval(3) = 0.5
      usrval(4) = 0.00006
      usrval(5) = 1.0
      usrval(6) = 1.0
      usrval(7) = 1.87d3
      usrval(8) = 5.0d3
      usrval(9) = 7.61d3
      usrval(10) = 10.d3
      usrval(11) = -1.05
      usrval(12) = 1.2
      usrval(13) = 0.0008
      usrval(14) = 1.
c
      youn(1) = 10000.
      youn(2) = 5000.
c
      call USRMAT( DEPS, nstr, 1, 1, se, ITER, USRVAL,
     $                   NUV, USRSTA, NUS, tsig, yt, youn )
c
      call privec( tsig, nstr, 'TSIG  ' )
      call primat( se, nstr, nstr, 'YT    ' )
c
c       Quatro iterações para obter:
c
c          tsig(1) =  8.370e-01
c          tsig(2) = -3.629e-02
c          tsig(3) =  7.645e-15
c          tsig(4) =  0.
c
c          se(1,1) = -7.097e03
c          se(1,2) = -7.097e02
c          se(2,2) =  4.929e+03
c          se(2,3) =  9.144e-11
c          se(4,4) =  1.970e+03
      end
c
c
c
      SUBROUTINE USRMAT( DEPS, nstr,
     $                   il, ip, se, ITER, USRVAL,
     $                   NUV, USRSTA, NUS, tsig, yt, young )
C
C...    USER-SUPPLIED MATERIAL MODEL
C...    RETURN UPDATED STRESS AND TANGENTIAL STIFFNESS MATRIX
C
      INTEGER          nstr, NUV, NUS, il, ip, ITER
      DOUBLE PRECISION DEPS(nstr),
     $                 se(nstr,nstr), USRVAL(NUV),
     $                 USRSTA(NUS), tsig(nstr), yt(nstr*nstr),
young(2)
C
C-------------------------------------------------------------
----------------------------
C...
C...   ARGUMENTS:
C...     deps   In     D(nstr)       - Strain increment
C...     nstr   In     I             - Dimension of stress
space (must be 4 or 6)
C...     il     In     I             - Number of element
C...     ip     In     I             - Number of integartion
point
C...     se     In     D(nstr,nstr)  - Linear elastic stress
strain relation
C...     iter   Out    I             - Number of internal
iterations for stress update
C...     usrval In     D(nuv)        - Parameters for yield
surface
C...     nuv    In     I             - Number of parameters for
yield surface (equal to 14)
C...     usrsta InOut  D(nus)        - State variables
C...     nus    In     I             - Number of state
variables (equal to 2)
C...     tsig   InOut  D(nstr)           - Stress at the
beginning of the step / Stress update
C...     yt     Out    D(nstr,nstr)  - Tangent stress strain
relation
C...     youn   In     D(2)          - Young modulus Exx and
Eyy - OUT-OF-PLANE FAILURE IGNORED!!!
C...
C-------------------------------------------------------------
----------------------------
C...
C...   usrval  D(14)  - Yield surface parameters
C...     (1)    : ftx
C...     (2)    : Gftx
C...     (3)    : fty
C...     (4)    : Gfty
C...     (5)    : alpha
C...     (6)    : alpha sub g
C...     (7)    : fmx
C...     (8)    : Gfcx
C...     (9)    : fmy
C...    (10)    : Gfcy
C...    (11)    : beta
C...    (12)    : gamma
C...    (13)    : kappa_peak_compression
C...    (14)    : h (length scale)
C...
C-----------------------------------------------------
C...
C...   usrsta  D(2)   - State variables
C...     (1)    : equivalent plastic strain tension
C...     (2)    : equivalent plastic strain compression
C...
C-----------------------------------------------------
C...
C...   yldpr  D(12)   - Yield properties
C...     (1)    : ftx(kappa)
C...     (2)    : fty(kappa)
C...     (3)    : alpha
C...     (4)    : alpha sub g
C...     (5)    : fmx(kappa)
C...     (6)    : fmy(kappa)
C...     (7)    : beta
C...     (8)    : gamma
C...     (9)    : d(sigq)/d(epsq)tx
C...    (10)    : d(sigq)/d(epsq)ty
C...    (11)    : d(sigq)/d(epsq)mx
C...    (12)    : d(sigq)/d(epsq)my
C...
C.....................................................
........
C
C...     externals
      double precision dasum
c
c...     common definition
      logical sw
      common /SWITCH/ sw(6)
c
C
      double precision youn
      common /YOUNG/ youn(2)
c
c...     local variables
      logical          lapex
      integer          mstr, miter
      double precision tolyfu
      parameter        ( mstr = 6, miter = 30, tolyfu = 1.d-5 )
c
      double precision yldpr(14), dsig(mstr), depse(mstr),
     .                 depsp(mstr), depeq(2), epeqyf(2)
c
      double precision jacob(10*10), invmat(10,10), dl(2),
fx(10),
     $                 x(10), a(mstr,mstr), b(mstr,mstr),
     $                 s(10), gradf(mstr), f, fold, gradnt,
     $                 tsign(mstr), tsigt(mstr), yf(2),
w(mstr*mstr),
     $                 hess(10,10), norm1, search
c
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
      integer          iiter, ierr, rp(10), nx, iapex, iyld,
ix, jx,
     $                 i, j
c
      double precision delold
c
      data rp / 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 /
c
      sw(3) = .true.
c
      youn(1) = young(1)
      youn(2) = young(2)
c
c
c...     Preliminary Checks
c
      if ( nstr .ne. 4 .and. nstr.ne. 6 ) THEN
         print *, 'SUBROUTINE ONLID VALID FOR PLANE STRAIN AND
3D'
         print *, 'PLEASE CHANGE SUBROUTINE'
         stop
      end if

c
      if ( nuv .ne. 14 ) THEN
         print *, 'WRONG NUMBER OF MATERIAL PARAMETERS <> 14'
         print *, 'PLEASE PROVIDE AS INPUT:'
         print *,
     $     'ftx  Gfx  fty  Gfy  alpha  alpha_g  fmx  Gfcx  fmy
Gfcy
     .      beta  gamma  eps_peak h'
         stop
      end if
c
      if ( nus .ne. 2 ) THEN
         print *, 'WRONG NUMBER OF STATE VARIABLES <> 2'
         print *, 'PLEASE PROVIDE AS INPUT: 0.0  0.0'
         stop
      end if
c=====------------------------------------------------------------
----------------
c...     Material Parameters - Definition of the yeldpr vector
c=====------------------------------------------------------------
----------------
      yldpr(1) = usrval(1)
      yldpr(2) = usrval(3)
      yldpr(3) = usrval(5)
      yldpr(4) = usrval(6)
      yldpr(5) = usrval(7)
      yldpr(6) = usrval(9)
      yldpr(7) = usrval(11)
      yldpr(8) = usrval(12)
c
      iyld = 3
      epeqyf(1) = usrsta(1)
      epeqyf(2) = usrsta(2)
      call rset( 0.d0, depeq, 2 )
c=====------------------------------------------------------------
----------------
c...      set up total test stress
c=====------------------------------------------------------------
----------------
      call rab( se, nstr, nstr, deps, 1, dsig)
      call uvpw( tsig, dsig, nstr, tsigt )
      if ( sw(3) ) call privec( tsigt, nstr, 'TSIGT' )
c
  600 continue
      linac = .false.
      nx = nstr + 2
      call rset( 0.d0, x, nx )
      call rmove( tsigt, x, nstr )
      call rmove( tsigt, tsign, nstr )
      if ( sw(3) ) call privec( x, nx, 'X' )
      call ypropx( usrval, epeqyf, depeq, yldpr )
      if ( sw(3) ) call privec( yldpr, 12, 'YLDPR' )
c
      if ( ( yldpr(5) .lt. 5.d-3) .or. ( yldpr(6) .lt. 5.d-3 )
) then
         call rset( 0.d0, tsign, nstr )
         iyld = 4
         iapex = 1
         goto 300
      end if
c
c=====------------------------------------------------------------
----------------
c...     First value of iyld
c=====------------------------------------------------------------
----------------
      call masyf( yldpr, tsigt, yf, nstr, lapex, usrval, epeqyf
)
      if ( yf(1) .le. 1.d-12 .and. yf(2) .le. 1.d-12) then
c        Behaviour remains elastic
         call rmove( tsigt, tsig, nstr )
         call rmove( se, yt, nstr*nstr)
         return
      end if
      if ( yf(2) .gt. 0.d0 .and. iyld .eq. 3 ) then
         iyld = 2
         nx = nstr + 1
      else if(yf(1) .gt. 0.d0 .and. (iyld .eq. 3 .or. iyld .eq.
2))then
         iyld = 1
         nx = nstr + 1
      else
         iyld = 3
         nx = nstr + 2
      end if

c
      iapex = 0
      lapex = .false.
      call rset( 0.d0, dl, 2 )
c
c=====------------------------------------------------------------
----------------
c      Plastic predictor-corrector
c=====------------------------------------------------------------
----------------
      do 100, iiter = 1, miter
         if ( sw(3) ) call priivl( iiter, 'ITER  ' )
  999    continue
         call mafx( fx, x, se, tsigt, nstr, iyld, lapex,
usrval, epeqyf,
     $              yldpr)
         if ( sw(3) ) call privec( fx, nx, 'fx-ok' )
         if ( sw(3) ) call priivl( iyld, 'yld   ' )
         if ( lapex ) goto 500
         if ( iyld .eq. 1 ) then
            call jaco12( usrval, epeqyf, jacob, se, nstr, x,
nx, iyld,
     $                   yldpr )
            if ( sw(3) ) call primat( jacob, nx, nx, 'jacob' )
         end if
         if ( iyld .eq. 2 ) then
            call jaco22( usrval, epeqyf, jacob, se, nstr, x,
nx, iyld,
     $                   yldpr )
            if ( sw(3) ) call primat( jacob, nx, nx, 'jacob' )
         end if
         if ( iyld .eq. 3 ) then
            call jaco32( usrval, epeqyf, jacob, se, nstr, x,
nx, iyld,
     $                   yldpr )
            if ( sw(3) ) call primat( jacob, nx, nx, 'jacob' )
         end if
         call invpgb( jacob, nx, rp, invmat, 1.d-10, ierr )
         call rab( jacob, nx, nx, fx, 1, s )
         if ( iyld .ne. 1 ) then
            call uvmw( x, s, nx, x )
            search = 1.d0
         else
            call lnsea2( fx, x, s, nx, se, tsigt, nstr, iyld,
search,
     $                   lapex, usrval, epeqyf, yldpr )
         end if
c=====------------------------------------------------------------
----------------
cc...                                              -1
cc...       IF THE LINE SEARCH  -> 0 THEN ASSUME [J]
SINGULAR.
cc...        PERTURB JACOBIAN AND RESTART
c=====------------------------------------------------------------
----------------
         if ( search .lt. 1.d-3 ) then
            call invpgb( jacob, nx, rp, invmat, 1.d-10, ierr )
            call rabt( jacob, nx, nx, jacob, nx, hess )
            if ( sw(3) ) call primat( hess, nx, nx, 'hess' )
c=====------------------------------------------------------------
----------------
c           NORM 1 of HESSIAN
c=====------------------------------------------------------------
----------------
            call rset( 0.d0, w, nx )
            norm1 = 0.d0
            do 101 ix = 1, nx
               do 102 jx = 1, nx
                  w(ix) = w(ix) + abs( hess( jx, ix ) )
  102          continue
               if ( w(ix) .gt. norm1 ) norm1 = w(ix)
  101       continue
            call unimax( invmat, nx )
            call uvpws( hess, invmat, nx*nx, norm1 * 5.d-8,
hess )
            if ( sw(3) ) call primat( hess, nx, nx, 'hess' )
            call invpgb( hess, nx, rp, invmat, 1.d-10, ierr )
            call rabt( hess, nx, nx, jacob, nx, invmat )
            call rab( invmat, nx, nx, fx, 1, s )
            if ( sw(3) ) call primat( invmat, nx, nx, 'new-ja'
)
            call lnsea2( fx, x, s, nx, se, tsigt, nstr, iyld,
search,
     $                   lapex, usrval, epeqyf, yldpr )
         end if
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
         if ( x(nstr+1) .lt. 0.d0 ) x(nstr+1) =0.d0
         if ( x(nstr+2) .lt. 0.d0 ) x(nstr+2) =0.d0
c=====------------------------------------------------------------
----------------
c        Check for convergence
c=====------------------------------------------------------------
----------------
         if ( dasum( nx, fx, 1 ) .le. ( tolyfu * nx ) ) goto
300
         call exma( tsign, dl, depeq, x, iyld, nstr )
         if( sw(3) ) then
            call privec( fx, nx, 'F(X)' )
            call privec( tsign, nstr, 'TSIGN ' )
            call privec( x, nx, 'X     ' )
            call prival( tolyfu, 'TOLYFU' )
            call privec( dl, 2, 'dl' )
            call prival( epeqyf(1)+depeq(1), 'EPEQ1' )
            call prival( epeqyf(2)+depeq(2), 'EPEQ2' )
         end if
         call fxma( tsign, depeq, x, iyld, nstr )
c
  100 continue
      if ((tsigt(1).lt.yldpr(1)) .or. (tsigt(2).lt.yldpr(2)))
then
         linac = .true.
         call privec( tsigt, nstr, 'SIGT' )
         call privec( tsign, nstr, 'SIGN' )
         call exma( tsign, dl, depeq, x, iyld, nstr )
         goto 300
      end if
  500 continue
c=====------------------------------------------------------------
----------------
c      Handle apex
c=====------------------------------------------------------------
----------------
      call rset( 0.d0, depeq, 2 )
      dl(2) = 0.d0
      call ypropx( usrval, epeqyf, depeq, yldpr )
      if ( ( yldpr(1) .lt. 1.d-4) .and. ( yldpr(2) .lt. 1.d-4 )
) then
         call rset( 0.d0, tsign, 4 )
         iyld = 4
         iapex = 1
         goto 300
      end if
      iyld = 1
      write( *, 4 ) il, ip
      call rmove( se, invmat, nstr*nstr )
      call invsym( invmat, nstr )
      call filma( +1, invmat, nstr )
c=====------------------------------------------------------------
----------------
c...     Calculate plastic corrector
c=====------------------------------------------------------------
----------------
      call fabra( a, b, nstr, se )
      do 400, iiter = 1, miter
         if( sw(3) )call priivl( iiter, 'ITER  ' )
c
      call ypropx( usrval, epeqyf, depeq, yldpr )
      call rset( 0.d0, w, nstr )
      w(1) = yldpr(1)
      w(2) = yldpr(2)
      call rab( a, nstr, nstr, w, 1, tsign )
      call rab( b, nstr, nstr, tsigt, 1, w )
      call uvpw( tsign, w, nstr, tsign )
c=====------------------------------------------------------------
----------------
c      Update plastic strains
c=====------------------------------------------------------------
----------------
      call uvmw( tsign, tsig, nstr, w )
      call rab( invmat, nstr, nstr, w, 1, depsp )
      call uvmw( deps, depsp, nstr, depsp )
c=====------------------------------------------------------------
----------------
c...       Determine elastic strain_increment and
stress_increment
c=====------------------------------------------------------------
----------------
      call uvmw( deps, depsp, nstr, depse )
      if( sw(3) ) then
         call privec( tsign, nstr, 'TSIGN ' )
         call privec( depsp, nstr, 'DEPSP ' )
         call privec( depse, nstr, 'DEPSE ' )
         call prival( epeqyf(1)+depeq(1), 'EPEQ1' )
       end if
c=====------------------------------------------------------------
----------------
c...     Calculate new depeq - secant method
c=====------------------------------------------------------------
----------------
      if ( iiter. eq. 1 ) then
         de1old = 0.d0
         fold = 0.d0 - (depsp(1)+depsp(2)) / 2.d0 - 0.5d0 *
sqrt(abs((
     $     depsp(1) - depsp(2) ) ** 2 + depsp(4) ** 2 /
yldpr(4) ) )
         depeq(1) = (depsp(1)+depsp(2)) / 2.d0 - 0.5d0 *
sqrt(abs((
     $     depsp(1) - depsp(2) ) ** 2 + depsp(4) ** 2 /
yldpr(4) ) )
       else
         f = depeq(1) - (depsp(1)+depsp(2)) / 2.d0 - 0.5d0 *
sqrt(abs((
     $     depsp(1) - depsp(2) ) ** 2 + depsp(4) ** 2 /
yldpr(4) ) )
         gradnt = ( f - fold ) / ( depeq(1) - de1old )
         de1old = depeq(1)
         fold = f
         depeq(1) = depeq(1) - ( f / gradnt )
         if( sw(3) ) call prival( f, 'f' )
       end if
c
      if( sw(3) ) then
         call prival( depeq(1), 'dEPEQ1' )
       end if
c=====------------------------------------------------------------
----------------
c         Check for convergence
c=====------------------------------------------------------------
----------------
         if ((abs(de1old-depeq(1))) .le. tolyfu)
     $      goto 450
  400 continue
      linac = .true.
      print*, 'Bug in the apex return mapping!'
      stop
c
  450 continue
      dl(1) = depeq(1)
      call uvs( depsp, nstr, 1.d0 / dl(1), gradf )
c=====------------------------------------------------------------
----------------
c      Store apex data
c=====------------------------------------------------------------
----------------

      call ypropx( usrval, epeqyf, depeq, yldpr )
      call rset( 0.d0, w, nstr )
      w(1) = yldpr(1)
      w(2) = yldpr(2)
      call rab( a, nstr, nstr, w, 1, tsign )
      call rab( b, nstr, nstr, tsigt, 1, w )
      call uvpw( tsign, w, nstr, tsign )
c=====------------------------------------------------------------
----------------
c      Update plastic strains
c=====------------------------------------------------------------
----------------
      call uvmw( tsign, tsig, nstr, w )
      call rab( invmat, nstr, nstr, w, 1, depsp )
      call uvmw( deps, depsp, nstr, depsp )
      iapex = 1
      iyld = 1
      if( sw(3) ) then
         call prival( dl(1), 'dl' )
         call privec( gradf, nstr, 'GRADF' )
      end if
  300 continue
      if ( iapex .eq. 0 ) call exma( tsign, dl, depeq, x, iyld,
nstr )
c=====------------------------------------------------------------
----------------
c     Check if another yield surface becomes active
c=====------------------------------------------------------------
----------------
      if ( iyld .ne. 3 ) then
      call ypropx( usrval, epeqyf, depeq, yldpr )
         call masyf( yldpr, tsign, yf, nstr, lapex, usrval,
epeqyf  )
         if ( iyld .eq. 2 .and. yf(1) .gt. 0.d0 ) then
c            print*, "MOVE FROM COMPRESSION TO TENSION"
            goto 600
         end if
         if ( iyld .eq. 1 .and. yf(2) .gt. 0.d0 ) then
c            print*, "MOVE FROM TENSION TO CORNER"
            goto 600
         end if
      end if
c=====------------------------------------------------------------
----------------
c     Check if the wrong side of the yield surface was selected
c=====------------------------------------------------------------
----------------
      if ( ( iyld .eq. 1 ) .and. ( iapex .eq. 0 ) )
     $   call findap( tsigt, nstr, x, yldpr, lapex, usrval,
epeqyf )

c...      update plastic strains
      call uvmw( tsigt, tsign, nstr, w )
      call rmove( se, invmat, nstr*nstr )
      call invsym( invmat, nstr )
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
      call filma( +1, invmat, nstr )
      call rab( invmat, nstr, nstr, w, 1, depsp )
c=====------------------------------------------------------------
----------------
c       determine elastic strain_increment and stress_increment
c=====------------------------------------------------------------
----------------
      call uvmw( deps, depsp, nstr, depse )
c
      call uvmw( tsign, tsig, nstr, dsig )
c=====------------------------------------------------------------
----------------
c...     Show yield values - Only for debug
c=====------------------------------------------------------------
----------------
      if (sw(3)) then
      call ypropx( usrval, epeqyf, depeq, yldpr )
         call PRIVEC( YLDPR, 8, 'YLDPR' )
      end if
c=====------------------------------------------------------------
----------------
c..      Calculate equivalent plastic strains and update
stresses
c=====------------------------------------------------------------
----------------
      usrsta(1) = epeqyf(1) + depeq(1)
      usrsta(2) = epeqyf(2) + depeq(2)
      call rmove( tsign, tsig, nstr )
c      print*, "yldpr",epeqyf(1), YLDPR(5)/YLDPR(6)
c      print*, "yldpr",epeqyf(2), YLDPR(1)/YLDPR(2)
c
      if ( sw(3) ) write( *, 5 ) il, ip, iiter
c-------
      if ( tsigt(4) * tsign(4) .lt. 0.d0 ) then
         print*, "Change of sign in shear stress. El: Ip:", il,
ip
         print*, "sigt: sign:", tsigt(4), tsign(4)
      end if
c-------
      if ( linac ) then
         if ( iyld .eq. 1 ) write( *, 1 ) il, ip, fx(nstr+1)
         if ( iyld .eq. 2 ) write( *, 2 ) il, ip, fx(nstr+1)
         if ( iyld .eq. 3 ) write( *, 3 ) il, ip, fx(nstr+1),
     $      fx(nstr+2)
      end if
c=====------------------------------------------------------------
----------------
c     Calculate consistent tangent operator
c=====------------------------------------------------------------
----------------
      if ( iapex .eq. 0 ) then
         do 80 j = 1, nstr
            do 80 i = 1, nstr
               yt( i+(j-1)*nstr ) = jacob(i+(j-1)*nx )
   80    continue
      else
         call drapex( yt, se, nstr, yldpr, tsign, depsp )
c          call rset( 0.d0, yt, nstr*nstr )
c          yt(1) = 10.d0
c          yt(nstr+2) = 10.d0
c          yt(2*nstr+3) = 10.d0
      end if
c
      if ( sw(3) ) call primat( yt, nstr, nstr, 'YT' )
c
    1 FORMAT ( ' ELEMENT:', I4, '   IP:', I4,
     $         ' IS INACCURATE - FTENS=', E11.3, '    TENSION
REGIME' )
    2 FORMAT ( ' ELEMENT:', I4, '   IP:', I4,
     $         ' IS INACCURATE - FCOMP=', E11.3,
     $         '    COMPRESSION REGIME' )
    3 FORMAT ( ' ELEMENT:', I4, '   IP:', I4,
     $         ' IS INACCURATE - FTENS=', E11.3, '    FCOMP=',
E11.3,
     $         '    CORNER REGIME' )
    4 FORMAT ( ' ELEMENT:', I4, '   IP:', I4, ' IS IN THE
APEX')
    5 FORMAT ( ' ELEMENT:', I4, '   IP:', I4, ' NO. OF INT.
ITER: ',I2)
      end
c
c
      SUBROUTINE YPROPx( usrval, epeqyf, DEPEQ, YLDPR )
C
C.....------------------------------------------
C...
C...  PURPOSE:
C...     UPDATING OF THE YIELD-CRITERION PROPERTIES
C...     TAKING INTO ACCOUNT: -HARDENING
C...
C.....------------------------------------------------------------
.......
C
C
      DOUBLE PRECISION usrval(*), epeqyf(*), DEPEQ(*), YLDPR(*)
C
      LOGICAL          SW
      COMMON /SWITCH/  SW(6)
C
C
      DOUBLE PRECISION h, xtop, HCT, HCS, HFT, KAPPA, DKAPPA,
sigeq
c
      h = usrval(14)
      xtop = usrval(13)
c
      kappa  = epeqyf(1)
      dkappa = depeq(1)
c
         call fnsigx( 3, usrval(1), 'HARVLX', h, xtop,
usrval(2), kappa,
     $               dkappa, sigeq, hct, hcs )
         yldpr(1) = sigeq
         yldpr(9) = hct
c
         call fnsigx( 3, usrval(3), 'HARVLY', h, xtop,
usrval(4), kappa,
     $               dkappa, sigeq, hct, hcs )
         yldpr(2) = sigeq
         yldpr(10) = hct
C
         kappa  = epeqyf(2)
         dkappa = depeq(2)
C
         call fnsigx( 14, usrval(7), 'CMPVLX', h, xtop,
usrval(8), kappa,
     $               dkappa, sigeq, hct, hcs )
         yldpr(5) = sigeq
         yldpr(11) = hct
c
         call fnsigx( 14, usrval(9), 'CMPVLY', h, xtop,
usrval(10), kappa,
     $               dkappa, sigeq, hct, hcs )
         yldpr(6) = sigeq
         yldpr(12) = hct
C
      IF ( SW(3) ) CALL PRIVEC( YLDPR, 12, 'YLDPR ' )
c
      RETURN
      END
c
c
      SUBROUTINE FNSIGx( IHARFN, sigq0, NAME, h, xtop, gf,
KAPPA,
     $                   DKAPPA, SIGEQ, HCT, HCS )
C
C.....------------------------------------------
C.....------------------------------------------------------------
.........
C
c...     common definition
      logical sw
      common /SWITCH/ sw(6)
C
      double precision youn
      common /YOUNG/ youn(2)
C
      INTEGER          EXPONE, CURVE3
      PARAMETER ( EXPONE=3, CURVE3=14 )
C
      INTEGER          IHARFN
      CHARACTER*6      NAME
      DOUBLE PRECISION xmed, sigq0, h, xtop, gf, KAPPA, DKAPPA,
SIGEQ,
     $                 HCT,HCS, hmax, xult, xval, xmdmax, young
C
      INTEGER          NHARVL, IDUM
C
C...       USER-SUPPLIED EQUIVALENT STRESS
C
C...     DETERMINE INITIAL EQUIVALENT STRESS
C
      IF ( IHARFN .EQ. EXPONE ) THEN
         if ( name .eq. 'HARVLX' ) then
            young = youn(1)
         end if
         if ( name .eq. 'HARVLY' ) then
            young = youn(2)
         end if
C
         HMAX = GF * YOUNG / ( SIGQ0 * SIGQ0 )
c
         IF ( H .GT. HMAX ) THEN
            SIGQ0 = SQRT( GF * YOUNG / H )
         END IF
         XULT = GF / ( H * SIGQ0 )
c
         XVAL = KAPPA + DKAPPA
         CALL FNCEXP( SIGQ0, XULT, XVAL, SIGEQ, HCT )
         sigeq = max( sigeq, 1.d-03*sigq0 )
c
      else if ( iharfn .eq. curve3 ) then
         if ( name .eq. 'CMPVLX' ) then
            young = youn(1)
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
         else
            young = youn(2)
         end if
c
         xmed   = 75.d0 / 67.d0 * gf / ( h * sigq0 ) + xtop
         xmdmax = sigq0 / young + xtop
         if ( xmed .lt. xmdmax ) then
            sigq0 = sqrt(  75.d0 / 67.d0 * gf * young / h )
            xmed  = xmdmax
         end if
c
         call fncr3( sigq0, xtop, xmed, kappa+dkappa, sigeq,
hct )
         sigeq = max( sigeq, 1.d-03*sigq0 )
c
      ELSE
         print*, 'Hardening / softening law mot yet
implemented'
         stop
      END IF
C
      RETURN
      END

c
c
      subroutine fncr3( fval0, xtop, xmed, x, fval, grad )
C
C.....................................................
C      Return yield value and df/dk for parabolic + exponencial
curve
C.....................................................................
.........
C
      double precision fval0, xtop, xmed, x, fval, grad
      double precision fval1, fval2, fval3, fval4, m
C
      fval1 = 0.3333333d0 * fval0
      fval2 = fval0
      fval3 = 0.5d0 * fval0
      fval4 = 0.1d0 * fval0
c
c...     Special case of kappa equal to zero : grad = +inf
      if ( x .eq. 0.D0 ) then
         fval = fval1
         grad = 1.d20
         return
      end if
c
      if ( x .le. xmed ) then
         if ( x .le. xtop ) then
            m    = sqrt( abs( ( 2.D0 * x / xtop - x * x /
     $                   ( xtop * xtop ) ) ) )
            fval = fval1 + ( fval2 - fval1 ) * m
            grad = ( fval2 - fval1 ) * ( 2.D0 / xtop - 2.D0 *
     $               x / ( xtop * xtop ) ) / 2.D0 / m
         else
            fval = ( fval3 - fval2 ) / ( xmed - xtop ) ** 2 *
     $             ( x - xtop ) ** 2 + fval2
            grad = 2.D0 * ( fval3 - fval2 ) / ( xmed - xtop )
     $               ** 2 * ( x - xtop )
         end if
       else
         m    = 2.D0 * ( fval3 - fval2 ) / ( xmed - xtop )
         fval = fval3 + ( fval4 - fval3 ) * ( 1.d0 -
     $           exp( m / ( fval3 - fval4 ) * ( x - xmed ) ) )
         grad = m * exp( m / ( fval3 - fval4 ) * ( x - xmed ) )
       end if
c
       return
c
       end
c
c
c
       subroutine masyf( yldpr, sig, yf, nstr, lapex, usrval,
     $                         epeqyf )
C
C.................................................
C...
C...  purpose:
C...     to determine the function value of the masonry
C...     yield criterion
C...
C...  input:
C...     arg.list:  yldpr  -yieldproperties
C...                sig    -stress values
C...                nstr   -dimension of stress space
C...
C...  output:
C...     arg.list:  yf     -function value of yield criteria.
C...                  (1)  -orthotropic rankine
C...                  (2)  -hill type
C...
C.....................................................................
.........
C
C...     arguments
      integer          nstr
      double precision sig(*), yf(*), yldpr(*), usrval(*),
epeqyf(*)
c
c...     common definition
      logical sw
      common /switch/ sw(6)
c
c...     local variables
      integer          mstr
      parameter ( mstr = 6 )
      double precision p(mstr,mstr) , tri(1) , w(mstr),
     .                 eta(mstr),
     .                 pi(mstr), uv
c
c...     orthotropic rankine
c
c                          T           (1/2)           T
c...     yf = ( 0.5 <eta> [p] <eta> )      + 1/2 <pi> <eta>
c
      call fpraor( p, nstr, yldpr(3) )
      call rset( 0.d0, pi, nstr )
      pi(1) = 1.d0
      pi(2) = 1.d0
      call feta( sig, yldpr, eta, nstr )
c      if( sw(4) ) call privec( ETA, nstr, 'ETA   ' )
c      if( sw(4) ) call primat( P, nstr, nstr, 'P     ' )
      call ratba( eta, nstr, 1, p, tri(1), w, 1  )
      yf(1) = sqrt( 0.5d0 * tri(1) ) + 0.5d0 * uv( pi, eta,
nstr )
c
c...     hill type
c                         T          1/2
c...     yf = ( 0.5 <sig> [p] <sig> )   - sigma
c
      call fpma( p, nstr, yldpr )
c      if( sw(4) ) call primat( p, nstr, nstr, 'P     ' )
      call ratba( sig, nstr, 1, p, tri(1), w, 1  )
      yf(2) = sqrt( 0.5d0 * tri(1) ) - sqrt( yldpr(5)*yldpr(6)
)
c
      end
c
c
      subroutine mafx( fx, x, se, sigt, nstr, iyld, lapex,
usrval,
     $           epeqyf, yldpr )
C
C.................................................
C...
C...
C...  PURPOSE:
C...     To setup and fill the vector of unknowns for the
return mapping
c...     of the composite yield surface for masonry
C...
C...  OUTPUT:
C...     ARG.LIST:  f(x) -vector of unknowns
C...
C.....................................................................
.........
C
C
C...     common definition
      logical sw
      common /switch/ sw(6)
c
      double precision rpla, uv, depeq(2), usrval(*), epeqyf(*)
      logical          lapex
c
      integer          mstr, nstr
      parameter        ( mstr = 6 )
c
      double precision se(nstr,*), yldpr(*), sigt(*), fx(*),
x(*)
c
      integer          ix, iyld
c
      double precision sig(mstr), gradg(mstr), gradfc(mstr),
     $                 w(mstr*mstr), yf(2), psig(mstr),
     $                 p(mstr,mstr), pc(mstr,mstr)
c
      double precision dl(2)
c
      ix = nstr + 1
c=====------------------------------------------------------------
----------------
c     Update softening parameters
c=====------------------------------------------------------------
----------------
      call exma( sig, dl, depeq, x, iyld, nstr )
      call ypropx( usrval, epeqyf, depeq, yldpr )
c=====------------------------------------------------------------
----------------
c     Calculate gradients
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c=====------------------------------------------------------------
----------------
      call fpraor( p, nstr, yldpr(4) )
      call ragror( p, sig, gradg, yldpr, nstr, lapex )
      if ( lapex ) return
      call fpma( pc, nstr, yldpr )
      call rab( pc, nstr, nstr, sig, 1, psig )
      call uvs( psig, nstr, 0.5d0 / sqrt( 0.5d0 *
     $          abs( uv( sig, psig, nstr) ) ), gradfc )
c
c      IF( SW(3) )CALL PRIVEC( GRADFC, NSTR, 'GRADFC' )
c      IF( SW(3) )CALL PRIVEC( GRADF, NSTR, 'GRADF' )
c
c...     Fill equations relative to sigma unknowns
c...     <FX> = <SIG> - <SIGT> + dl(1) [D]  <GRADG> + dl(2) [D]
<GRADFC>
      call uvmw( sig, sigt, nstr, fx )
      call invsym( se, nstr )
      call filma( +1, se, nstr )
      call rab( se, nstr, nstr, fx, 1, w )
      call invsym( se, nstr )
      call filma( +1, se, nstr )
      call uvpws( w, gradg, nstr, dl(1), fx )
      call uvpws( fx, gradfc, nstr, dl(2), fx )
c...
      call masyf( yldpr, sig, yf, nstr, lapex, usrval, epeqyf )
      if ( iyld .eq. 1 .or. iyld. eq. 3 ) then
c
c...        Tension
c...        Fill equation relative to scalar (dl(1))
c...        FX = F
         fx(ix) = yf(1)
         ix = ix + 1
      end if
      if ( iyld .eq. 2 .or. iyld. eq. 3 ) then
c
c...        Compression
c...        Fill equations relative to scalar unknown (dl(2))
c...        FX = F
         fx(ix) = yf(2)
         ix = ix + 1
      end if
c
      end
c
c
      subroutine exma( sig, dl, depeq, x, iyld, nstr )
C
C...................................................
C...
C...  PURPOSE:
C...     to extract sig and dl from the vector of unknowns
C...
C...  INPUT:
C...     ARG.LIST:  x    -vector of unknowns
C...                iyld -which yield functions are active
C...
C...  OUTPUT:
C...     ARG.LIST:  sig  -stress
C...                dl   -plastic multiplier
C...                depeq-equivalent plastic strain
C...
C.............................................................
.........
C
C...     arguments
      integer          iyld, nstr, i
      double precision sig(*), x(*), dl(*), depeq(*)
c
      call rmove( x, sig, nstr )
      do 10 i = 1, nstr
         sig(i) = x(i)
   10 continue
      if ( iyld .eq. 1 ) then
         dl(1)    = x(nstr+1)
         depeq(1) = x(nstr+1)
         depeq(2) = 0.d0
         dl(2)    = 0.d0
      else if ( iyld .eq. 2 ) then
         dl(1)    = 0.d0
         depeq(1) = 0.d0
         dl(2)    = x(nstr+1)
         depeq(2) = x(nstr+1)
      else
         dl(1)    = x(nstr+1)
         depeq(1) = x(nstr+1)
         depeq(2) = x(nstr+2)
         dl(2)    = x(nstr+2)
      end if
c
      end
c
c
      subroutine fpraor( p, nstr, tau )
C
C...................................................
C...
C...  purpose:
C...     to setup and fill the projection-matrix on the
"orthotropic" rankine
C...     yield condition
C...
C...  input:
C...     arg.list:
C...
C...  output:
C...     arg.list:  p    -projection-matrix
C...
C.............................................................
.........
C
c...     arguments
      integer          nstr
      double precision p(nstr,*), tau
c
c...     local variables
      integer          i
      double precision r1
c
      call rset( 0.0d0 , p , nstr*nstr )
c
      r1 = 0.5D0
c
      p(1,1) =   r1
      p(1,2) = - r1
      p(2,1) = - r1
      p(2,2) =   r1
      do 10 i = 4, 4
         p(i,i) = 2.0d0 * tau
   10 continue
c
      end
c
c
      subroutine ragror( p, sig, gradf, yldpr, nstr, lapex )
C
C...................................................
C...
C...  purpose:
C...     to determine gradient vector  < gradf > for the
C...     "orthotropic" rankine yield criterion
C...
C...  input:
C...     arg.list:  sig    -stress values
C...                nstr   -dimension of stress space
C...
C...     /SWITCH/:  sw     -logical flags for test output.
C...
C...  output:
C...     arg.list:  gradf  -gradient vector.
C...
C...  workspace:
C...      local:    grad   -gradient vector
C...                p(6,6) -projection matrix
C...                r      -constant
C...                tri    -triple product <SIG>[P]<SIG>
C...
C.............................................................
....
C
c...     externals
      double precision  uv
c
c...     arguments
      logical          lapex
      integer          nstr
      double precision sig(*), gradf(*), yldpr(*), p(*)
c
c...     common definition
      logical sw
      common /SWITCH/ sw(6)
c
c...     local variables
      integer          mstr
      parameter ( mstr = 6 )
      double precision  grad(mstr), r, tri, pi(mstr), eta(mstr)
c
      lapex = .false.
      call feta( sig, yldpr, eta, nstr )
      call rset( 0.d0 , pi, nstr )
      pi(1) = 1.d0
      pi(2) = 1.d0
c
c...     <GRAD> = [P] * <ETA>
      call rab( p, nstr, nstr, eta, 1, grad )
c                  T
c...     TRI = <eta> * [P] * <eta>
      tri = uv( eta, grad, nstr )
c
      if ( abs( tri ) .lt. 1.d-14 ) then
         if ( sw(3) ) call prival( tri, 'TRI')
         lapex = .true.
         return
      end if
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c
      r   = sqrt( 2.0d0 / abs(tri) ) / 2.d0
c
c...                          [P] <eta>
c...     <GRADF> = ----------------------------   + 1/2 <pi>
c...                2 sqrt {1/2 <eta> [P] <eta> }
c
      call uvs( grad, nstr, r, grad )
      call uvpws( grad, pi, nstr, 0.5D0, gradf )
c
      end
c
c
      subroutine fpma( p, nstr, yldpr )
C
C.....................................................
C...
C...  purpose:
C...     to setup and fill the projection-matrix of the
compressive part
C...     of the masonry yield criteria
C...
C...  output:
C...     arg.list:  p    -projection-matrix
C...
C...  programmed by P.B.Lourenco
C.....................................................................
.........
C
C...     arguments
      integer          nstr, i
      double precision p(nstr,*), yldpr(*)
c
      call rset( 0.0d0 , p , nstr*nstr )
c
      p(1,1) = 2.d0 * yldpr(6) / yldpr(5)
      p(1,2) = yldpr(7)
      p(2,1) = yldpr(7)
      p(2,2) = 2.d0 * yldpr(5) / yldpr(6)
      do 10 i = 4, 4
         p(i,i) = 2.d0 * yldpr(8)
   10 continue
c
      end


c
c
      subroutine feta( sig, yldpr, eta, nstr )
C
C.....................................................
C...
C...  purpose:
C...     to calculate the stress in the reduced space for the
orthotropic
C...     rankine yield criterion
C...
C...  input:
C...     arg.list:  sig    -stresses in the original space
c...                yldpr  -yield properties
C...
C...  output:
C...     arg.list:  eta    -stresses in the reduced space
C...
C.....................................................................
.........
C
C...     arguments
      integer          nstr
      double precision sig(*), yldpr(*), eta(nstr)
c
      call rmove( sig, eta, nstr )
c
      eta(1) = eta(1) - yldpr(1)
      eta(2) = eta(2) - yldpr(2)
c
      end
c
c=====-----------------------------------------------------------
----------------
c     This subroutine calaculates the Jacobian for Rankine type
yield criterion
c=====-----------------------------------------------------------
----------------
c
      subroutine jaco12( usrval, epeqyf, a, stiff, nstr, x, nx,
iyld,
     $                   yldpr )
C
C.....................................................
C...
C...  purpose:
C...     to calculate the Jacobian for Rankine type yield
criterion
C...
C...  input:
C...     arg.list:  stiff -elastic stiffness matrix
C...                x     -vector of unknowns
C...                nx    -number of unknowns
C...
C...  output:
C...     arg.list:  a    -Jacobian
C...
C.....................................................................
.........
C
C...     externals
      double precision uv
c
c...     parameters
      integer          mstr
      parameter        ( mstr=6 )
c
c...     arguments
      integer          nstr, nx, iyld
c
      double precision yldpr(*), stiff(nstr,*), x(*), a(nx,*),
     $                 usrval (*), epeqyf(*)
c
c...     common definition
      logical sw
      common /switch/ sw(6)
c
c...     local variables
      double precision dl(2), sig(mstr)
      double precision p(mstr,mstr), depeq(2), eta(mstr),
peta(mstr),
     $                 d2gds2(mstr,mstr), gradf(mstr),
w(10*10),
     $                 dnedk(mstr), d2gdk(mstr), dfdk, tri,
     $                 gradg(mstr)
      integer          i, j, k
      logical          ldum
c
      call rset( 0.d0, dnedk, nstr )
      call rset( 0.d0, a, 5*5 )
c
c...     Update softening parameters
      call exma( sig, dl, depeq, x, iyld, nstr )
      call ypropx( usrval, epeqyf, depeq, yldpr )
c
      if ( sw(3) ) then
         call prival( yldpr(9),  'HC1' )
         call prival( yldpr(10), 'HC2' )
         call prival( dl(1),     'DL ' )
         call privec( sig, nstr, 'SIG' )
      end if
c
      call fpraor( p, nstr, yldpr(3) )
      call feta( sig, yldpr, eta, nstr )
c
c...     Calculate gradients
      call ragror( p, sig, gradf,  yldpr, nstr, ldum )
      call fpraor( P, nstr, yldpr(4) )
      call ragror( p, sig, gradg,  yldpr, nstr, ldum )
      call rab( P, nstr, nstr, eta, 1, peta )
      tri = sqrt( 0.5d0 * uv( eta, peta, nstr ) )
      if ( sw(3) ) then
         call privec( gradf, nstr, 'GRADF' )
         call privec( gradg, nstr, 'GRADG' )
         call privec( eta, nstr,   'ETA  ' )
      end if
C
C...     calculate d2g/dsigma2 = d2gds2
      call rabt( peta, nstr, 1, peta, nstr, w )
      call uvs( w, nstr*nstr, -0.25d0 / tri**3, w )
      call uvpws( w, p, nstr*nstr, 0.5D0 / tri, d2gds2 )
c
c...     calculate d2f/dkdsigma = d2gdk
      dnedk(1) = -yldpr(9)
      dnedk(2) = -yldpr(10)
      call rab( d2gds2, nstr, nstr, dnedk, 1, d2gdk )
      if ( sw(3) ) then
         call primat( d2gds2, nstr, nstr, 'D2GDS2' )
         call primat( stiff, nstr, nstr,  'STIFF ' )
         call privec( d2gdk, nstr,        'D2GDK ' )
      end if
c-------------------------------------------------------------
c...     Fill matrix [A] with sigma related lines
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      call uvpws( stiff, d2gds2, nstr*nstr, dl(1), w )
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      k = 0
      do 10 j = 1, nstr
         do 10 i = 1, nstr
            k = k + 1
            a(i,j) = w(k)
   10 continue
c
      call uvpws( gradg, d2gdk, nstr, dl(1), w )
      do 20 i = 1, nstr
         a(i,nstr+1) = w(i)
   20 continue
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c
c-------------------------------------------------------
c
c...     Calculate df/dk = dfdk
      dfdk = uv( gradf, dnedk, nstr )
      if ( sw(3) ) then
         call prival( dfdk, 'DFDK')
      end if
c-------------------------------------------------------
c...     Fill matrix [A] with yield function related line
      do 50 i = 1, nstr
         a(nstr+1,i) = gradf(i)
   50 continue
c
      a(nstr+1,nstr+1) = dfdk
c
c-------------------------------------------------------
      end
c
c=====-------------------------------------------------------
----------------
c     This subroutine fills the vector of unknowns for the
return mapping
c=====-------------------------------------------------------
----------------
      subroutine fxma( sig, depeq, x, iyld, nstr )
C
C.............................................
C...
C...  purpose:
C...     to fill the vector of unknowns for the return mapping
C...
C...  input:
C...     arg.list:  sig  -stress
C...
C...  output:
C...     arg.list:  x    -vector of unknowns
C...
C.............................................................
.........
C
C...     arguments
      integer          iyld, nstr
      double precision sig(*), x(*), depeq(*)
c
      call rmove( sig, x, nstr )
      if ( iyld .eq. 1 ) then
         x(nstr+1) = depeq(1)
      else if ( iyld .eq. 2 ) then
         x(nstr+1) = depeq(2)
      else
         x(nstr+1)  = depeq(1)
         x(nstr+2)  = depeq(2)
      end if
c
      end
c=====-------------------------------------------------------
----------------
c     This subroutine calculate the inverse of the matrix A by
a Gauss-Jordan
c      algorithm
c=====-------------------------------------------------------
----------------
      SUBROUTINE INVPGB( A, NA, RP, RA, EPS, IERR )
C
C.................................................
C...
C...     PURPOSE:
C...        MATRIX INVERSION WITH PARTIAL PIVOTING( GAUSS
JORDAN).
C...        THE COMPLETE INVERSE OF A IS PROJECTED ON THE
ORIGINAL MATRIX
C...        [ A ].
C...        < RP > AND < RA > ARE USED FOR ROW INTERCHANGING.
C
C...     INPUT:
C...        ARG.LIST:  A      -R(NA*NA): MATRIX TO BE INVERTED.
C...                   NA     -I(1):     ORDER OF [ A ]
C...                   EPS    -R(1):     TOLERANCE FOR PIVOT
C...     OUTPUT:
C...        ARG.LIST:  A      -R(NA*NA): INVERSE OF [ A ]
C...                   IERR   -I(1):     ERROR CODE:
C...                                   = 0 INVERSION SUCCESFULL;
C...                                   =-1 SINGULARITY
ENCOUNTERED;
C...
C...     WORKSPACE:
C...        ARG.LIST:  RP     -I(NA):    ROW INDICES OF [ A ]
C...                   RA     -R(NA):    WORKSPACE FOR ROW OF [
A ]
C...
C.............................................................
.........
C
C...ARGUMENTS:
      INTEGER          NA, RP(*), IERR
      DOUBLE PRECISION A(NA,*), RA(*), EPS
C
C...LOCAL
      INTEGER          I, IR, J, K, N
      DOUBLE PRECISION APIV, AR, EPIV, HPIV
C
      IERR = 0
      N = NA
      EPIV = EPS
C
C...     INITIALIZE ROW-INDICES:
      DO 10, I = 1, N
         RP(I) = I
   10 CONTINUE
C
C...     ROW LOOP:
      DO 200, I = 1, N
C
C...        FIND PIVOT
         HPIV = ABS( A(I,I) )
         IR = I
         IF( I .LT. N ) THEN
            DO 50, K = (I+1), N
               APIV = ABS( A(K,I) )
               IF( APIV .GT. HPIV ) THEN
                 HPIV = APIV
                 IR = K
               END IF
   50       CONTINUE
         END IF
         IF( HPIV .LT. EPIV ) THEN
            IERR = -RP(IR)
            print*, 'INVERSION FAILED'
            RETURN
         END IF
C
         IF( IR .GT. I ) THEN
C...        INTERCHANGE ROWS:
            DO 60, J = 1, N
               AR      = A(I, J)
               A(I, J) = A(IR,J)
               A(IR,J) = AR
   60       CONTINUE
            K      = RP(I)
            RP(I)  = RP(IR)
            RP(IR) = K
         END IF
C
         APIV = 1.D0/A(I,I)
         CALL UVS( A(1,I), N, APIV, A(1,I) )
         A(I,I) = APIV
C
C...        COLUMN LOOP:
         DO 100, J = 1, N
            IF( J .NE. I ) THEN
               APIV = -A(I,J)
               IF( APIV .NE. 0.D0 ) THEN
                  CALL UVPWS( A(1,J), A(1,I), N, APIV, A(1,J) )
                  A(I,J) = APIV * A(I,I)
               ELSE
                  A(I,J) = 0.D0
               END IF
            END IF
  100    CONTINUE
C
  200 CONTINUE
C
C...     INTERCHANGE COLUMNS:
      DO 250, I = 1, N
C
         DO 210, J = 1, N
               K  = RP(J)
            RA(K) = A(I,J)
  210    CONTINUE
C
         DO 220, J = 1, N
            A(I,J) = RA(J)
  220    CONTINUE
C
  250 CONTINUE
      RETURN
      END
      subroutine lnsea2( fx, x, s, nx, se, sigt, nstr, iyld,
dl,
     $                   lapex, usrval, epeqyf, yldpr )
c
c.............................................
c...
c...     Line search subroutine for the X update
c...     ARGUMENTS:
c...                x(nx)  - vector of unknonwns
c...                fx(nx) - vector of residuals
c...                s(nx)  - x update vector
c...                                  -1
c...                          <s> = [J(x)] . <fx>
c...     OUTPUT:
c...                x(nx)  - updated vector of unknonwns
c...
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c...     SEE DENNIS AND SCHNABEL, pp. 147-152
c...     PARAMETERS:
c...                  ALFA controls the reduction on the initial
fx value
c...                  TOLER overrides the linesearch if fx is
already small
c...                                                  T
c...                       N.B: Toler = 0.5 * <fx> <fx>
c...
C.....................................................................
.........
C
c
c...     parameters
      integer          mx, msear
      double precision alfa, toler
      logical          lapex
      parameter        ( mx = 10, msear = 20, alfa = 1.d-4,
     $                   toler = 1.d-8 )
c...     externals
      double precision uv
c
c...     common definition
      logical sw
      common /SWITCH/ sw(6)
c
c...     arguments
      integer          nx, nstr, iyld
      double precision fx(*), x(*), s(*), se(nstr,*), sigt(*),
     $                 usrval(*), epeqyf(*), yldpr(*)
c
c...     local variables
      integer          isear
      double precision xnew(mx), fx1(mx), fx2(mx), dl, dlprev,
fxx,
     $                 fx11, fx22, fxfx, mat11, mat12, mat21,
mat22,
     $                 fval1, fval2, det, a, b
c
      if ( nx .gt. mx ) then
         print*, "Error in LNSEAR.F - Please correct array
boundaries"
         stop
       end if
c
      dl = 1.d0
      fxfx = - uv( fx, fx, nx )
c
      call uvmw( x, s, nx, xnew )
      if ( sw(3) ) call privec( xnew, nx, 'XTRIAL' )
      call mafx( fx1, xnew, se, sigt, nstr, iyld,
     $           lapex, usrval, epeqyf, yldpr )
      fxx = uv( fx, fx, nx ) / 2.d0
      fx11 = uv( fx1, fx1, nx ) / 2.d0
      if ( sw(3) ) then
         call prival( fx11, 'FX-NEW' )
         call prival( fxx + alfa * dl * fxfx, 'FX-INI' )
      end if
      if ( ( fx11 .lt. 2.d0 * ( fxx + alfa * dl * fxfx ) )
     $    .or. ( fx11 .lt. toler ) ) then
         call rmove( xnew, x, nx )
         return
      end if
c
c...     Line search correction - Quadratic backtrack
      dlprev = dl
      dl = -0.5d0 * fxfx / ( fx11 - fxx - fxfx )
      call chekdl( dlprev, dl )
      if ( sw(3) ) call prival( dl, 'LNSEAR' )
      call uvpws( x, s, nx, -dl, xnew )
c
      do 100, isear = 1, msear
c
         call mafx( fx2, xnew, se, sigt, nstr, iyld,
     $              lapex, usrval, epeqyf, yldpr )
         fx22 = uv( fx2, fx2, nx ) / 2.d0
         if ( sw(3) ) then
            call prival( fx22, 'FX-NEW' )
            call prival( fxx + alfa * dl * fxfx, 'FX-INI' )
         end if
         if ( fx22 .lt. ( fxx + alfa * dl * fxfx ) ) then
            call rmove( xnew, x, nx )
            return
         end if
c...        Line search correction - Cubic backtrack
         mat11 =  1.d0 / dl / dl
         mat12 = -1.d0 / dlprev / dlprev
         mat21 = -dlprev / dl / dl
         mat22 = dl / dlprev / dlprev
c
         fval1 = fx22 - fxx - fxfx * dl
         fval2 = fx11 - fxx - fxfx * dlprev
c
         det = dl - dlprev
c
         a = ( mat11 * fval1 + mat12 * fval2 ) / det
         b = ( mat21 * fval1 + mat22 * fval2 ) / det
c
         dlprev = dl
         dl = ( - b + sqrt( b * b - 3.d0 * a * fxfx ) ) / 3.d0
/ a
         call chekdl( dlprev, dl )
         if ( sw(3) ) call prival( dl, 'LNSEAR' )
         fx11 = fx22
         call uvpws( x, s, nx, -dl, xnew )
c
  100 continue
c
      if ( sw(3) )
     $   print*, "Line search did not converge after 10
iterations"
      call rmove( xnew, x, nx )
      return
c
      end

      subroutine chekdl( dlprev, dl )
c
c...     PARAMETERS
      double precision mintol, maxtol
      parameter        ( mintol = 0.1d0, maxtol = 0.5d0 )
C
C...     ARGUMENTS
      double precision dlprev, dl
c
      dl = min( dl, maxtol * dlprev )
      dl = max( dl, mintol * dlprev )
c
      return
c
      end

c
c...  Returns the identity matrix
c
      subroutine unimax( matrix, l )
c
      integer          i,l
      double precision matrix(l,l)

c
      call rset(0.d0, matrix, l*l )
      do 10, i = 1, l
   10    matrix(i,i) = 1.d0
c
      return
c
      end
c
      subroutine jaco22( usrval, epeqyf, a, stiff, nstr, x, nx,
iyld,
     $                   yldpr )
C
C.................................................
C...
C...  purpose:
C...     to calculate the Jacobian for Hill type yield
criterion
C...
C...  input:
C...     arg.list:  stiff -elastic stiffness matrix
C...                x     -vector of unknowns
C...                nx    -number of unknowns
C...
C...  output:
C...     arg.list:  a    -Jacobian
C...
C.....................................................................
.........
c
c...     externals
      double precision uv
c
c...     parameters
      integer          mstr
      parameter        ( mstr=6 )
c
c...     arguments
      integer          nstr, nx, iyld
c
      double precision yldpr(*), stiff(nstr,*), x(*), a(nx,*),
     $                 usrval (*), epeqyf(*)
c
c...     common definition
      logical sw
      common /switch/ sw(6)
c
c...     local variables
      double precision dl(2), sig(mstr)
      double precision p(mstr,mstr), depeq(2), gradf(mstr),
w(10*10),
     $                 dfdk, tri, dpdk(mstr*mstr), psig( mstr),
     $                 yield, d2fds2(mstr,mstr), dfdkds(mstr)
      integer          i, j, k
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
C
      call rset( 0.d0, a, nx*nx )
      call rset( 0.d0, dpdk, nstr*nstr )
c
c...     Update softening parameters
      call exma( sig, dl, depeq, x, iyld, nstr )
      call ypropx( usrval, epeqyf, depeq, yldpr )
      call fpma( p, nstr, yldpr )
c
      if ( sw(3) ) then
         call prival( yldpr(11),     'HC1C' )
         call prival( yldpr(12),     'HC2C' )
         call prival( dl(2),         'DL  ' )
         call privec( sig, nstr,     'SIG ' )
         call primat( p, nstr, nstr, 'P' )
      end if
c
c...     Calculate gradient
      call rab( p, nstr, nstr, sig, 1, psig )
      yield = sqrt ( 0.5d0 * abs ( uv( sig, psig, nstr ) ) )
      call uvs( psig, nstr, 0.5d0 / yield, gradf )
c
c...     Calculate d[P]/dk
      dpdk(1) = 2.d0 * ( yldpr(12) * yldpr(5) - yldpr(11) *
yldpr(6) )
     $          / yldpr(5) / yldpr(5)
      dpdk(nstr+2) = 2.d0 * ( yldpr(11) * yldpr(6) - yldpr(12)
*
     $               yldpr(5) ) / yldpr(6) / yldpr(6)
c
      if ( sw(3) ) call primat( dpdk, nstr, nstr, 'DPDK' )
c                              T
c...     Calculate dfdk = <SIG> d[P]/dk <SIG> / ( 4 yield ) -
d(yield)/dk
      call rab( dpdk, nstr, nstr, sig, 1, w )
      tri = uv( sig, w, nstr )
      dfdk = 0.25d0 * tri / yield - 0.5d0 *
     $       ( yldpr(12) * YLDPR(5) + yldpr(11) * YLDPR(6) ) /
     $       sqrt( yldpr(5) * yldpr(6) )
      if ( sw(3) ) call prival( dfdk, 'dfdk' )
c
c                                                          T
3
c...     Calculate d2fds2 = [P] / ( 2 yield ) - [P]<SIG>
<SIG>[P] / ( 4 yield  )
      call rab( psig, nstr, 1, psig, nstr, w )
      call uvs( w, nstr*nstr, -0.25d0 / yield ** 3, w )
      call uvpws( w, p, nstr*nstr, 0.5d0 / yield, d2fds2 )
      if ( sw(3) ) call primat( d2fds2, nstr, nstr, 'D2FDS2' )
c
c...     Calculate d2fdkds = d[P]/dk <SIG> /  ( 2 yield ) -
c...                                 T
3
c...                          ( <SIG> d[P]/dk <SIG> ) [P] <SIG>
/ ( 8 yield  )
      call rab( dpdk, nstr, nstr, sig, 1, w )
      call uvs( w, nstr*nstr, 0.5d0 / yield, w )
      call uvpws( w, psig, nstr, - tri / 8.d0 / yield ** 3,
dfdkds )
      if ( sw(3) ) call privec( dfdkds, nstr, 'DFDKDS' )
c-------------------------------------------------------------
c...     FILL MATRIX A WITH SIGMA RELATED LINES
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      call uvpws( stiff, d2fds2, nstr*nstr, dl(2), w )
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      k = 0
      do 10 j = 1, nstr
         do 10 i = 1, nstr
            k = k + 1
            a(i,j) = w(k)
   10 continue
c
      call uvpws( gradf, dfdkds, nstr, dl(2), w )
      do 20 i = 1, nstr
         a(i,nstr+1) = w(i)
   20 continue
c
c-------------------------------------------------------------
c...     FILL MATRIX A WITH YIELD FUNCTION RELATED LINE
      do 50 i = 1, nstr
         a(nstr+1,i) = gradf(i)
   50 continue
c
      a(nstr+1,nstr+1) = dfdk
c
c-------------------------------------------------------------
      if ( sw(3) ) call primat( a, 5, 5, 'A' )
c
      end

      subroutine fabra( a, b, nstr, stiff )
C
C.................................................
C...
C...  purpose:
C...     to setup and fill the return mapping matrices of the
c...     "orthotropic" Rankine type yield condition in the apex
C...
C...  input:
C...     arg.list:
C...
C...  output:
C...     arg.list:  a,b    -projection-matrices
C...
C...  programmed by P.B.Lourenco
C.............................................................
.........
C
c...     arguments
      integer          nstr
      double precision a(nstr,*), b(nstr,*), stiff(nstr,*)
c
      call rset( 0.0d0 , a , nstr*nstr )
      call rset( 0.0d0 , b , nstr*nstr )
c
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
c
      call unimax( a, nstr )
      a(3,1) = - stiff(3,1) / stiff(3,3)
      a(3,2) = - stiff(3,2) / stiff(3,3)
c
      b(3,1) = stiff(3,1) / stiff(3,3)
      b(3,2) = stiff(3,2) / stiff(3,3)
      b(3,3) = 1.d0
      if ( nstr .eq. 6 ) then
         b(5,5) = 1.d0
         b(6,6) = 1.d0
      end if
c
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
c
      end
c
      subroutine findap( tsigt, nstr, x, yldpr, lapex, usrval,
epeqyf )
C
C.................................................
C...
C...  purpose:
C...     check if the wrong side of the rankine yield surface
was selected
C...
C.............................................................
.........
C
c...     externals
      double precision uv
c
c...     parameters
      integer          mstr
      parameter        ( mstr=6 )
c
c...     arguments
      integer          nstr
      logical          lapex
c
      double precision yldpr(*), tsigt(*), x(*), usrval(*),
epeqyf(*)
c
c...     common definition
      logical sw
      common /SWITCH/ sw(6)
c...     local variables
      double precision dl(2), sig(mstr)
      double precision p(mstr,mstr), depeq(2), eta(MSTR),
pi(mstr),
     $                 dsig(mstr), pdsig(mstr), rdum, pidsig,
k, yf(2)
c
      call rset( 0.d0, pi, nstr )
      k = 0.d0
c
c...     Update softening parameters
      call exma( sig, dl, depeq, x, 1, nstr )
      call ypropx( usrval, epeqyf, depeq, yldpr )
c
      call uvmw( tsigt, sig, nstr, dsig )
      if ( sw(3) ) call privec( tsigt, nstr, 'tsigt' )
      if ( sw(3) ) call privec( sig, nstr, 'tsign' )
c
      call fpraor( p, nstr, yldpr(3) )
      pi(1) = 1.d0
      pi(2) = 1.d0
      call feta( sig, yldpr, eta, nstr )
      if ( sw(3) ) call privec( eta, nstr, 'ETA   ' )
c
      call rab( p, nstr, nstr, dsig, 1, pdsig )
      pidsig = uv( pi, dsig, nstr )
      rdum = uv( dsig, pdsig, nstr ) - 0.5d0 * pidsig * pidsig
      if ( abs( rdum ) .lt. 1.d-8 ) return
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
      k = ( uv( pi, eta, nstr ) * pidsig -
     $    2.d0 * uv( eta, pdsig, nstr ) ) / rdum
      if ( sw(3) ) call prival( k, 'K' )
      if ( k .gt. 0.d0 .and. k. lt. 1.d0 ) then
c
c...     Other solution might exist
         call uvpws( sig, dsig, nstr, k, sig )
         call masyf( yldpr, sig, yf, nstr, lapex, usrval,
epeqyf )
         IF ( SW(3) ) call prival( yf(1), 'yf-new' )
         if ( abs ( yf(1) ) .lt. 1.d-6 ) then
c...        other solution exists
            print*, "Other solution exists for apex
solution..."
         end if
      end if
c
      end

      subroutine jaco32( usrval, epeqyf, a, stiff, nstr, x, nx,
iyld,
     $                   yldpr )
C
C.....................................................
C...
C...  purpose:
C...     to calculate the Jacobian for the corner of the
C...     composite masonry criterion
C...
C...  input:
C...     arg.list:  stiff -elastic stiffness matrix
C...                x     -vector of unknowns
C...                nx    -number of unknowns
C...
C...  output:
C...     arg.list:  a     -Jacobian
C...
C.....................................................
.........
c
c...     externals
      double precision uv
c
c...     parameters
      integer          mstr
      parameter        ( mstr=6 )
c
c...     arguments
      integer          nstr, nx, iyld
c
      double precision yldpr(*), stiff(NSTR,*), x(*), a(nx,*),
     $                 usrval (*), epeqyf(*)
c
c...     common definition
      logical sw
      common /switch/ sw(6)
c
c...     local variables
      double precision dl(2), sig(mstr)
      double precision p(mstr,mstr), depeq(2), eta(mstr),
peta(mstr),
     $                 d2gds2(mstr,mstr), gradf(mstr),
w(10*10),
     $                 dnedk(mstr), d2gdk(mstr), df1dk, tri,
     $                 gradg(mstr), pc(mstr,mstr),
gradfc(mstr), tric,
     $                 psig(mstr), dfdk, yield,
d2fds2(mstr,mstr),
     $                 dfdkds(mstr), dpdk(mstr*mstr),
gamma(mstr)
      integer          i, j, k
      logical          ldum
c
      call rset( 0.d0, dnedk, nstr )
      call rset( 0.d0, a, nx*nx )
      call rset( 0.d0, dpdk, nstr*nstr )
c
c...     Update softening parameters
      call exma( sig, dl, depeq, x, iyld, nstr )
      call ypropx( usrval, epeqyf, depeq, yldpr )
c
      if ( sw(3) ) then
         call prival( yldpr( 9), 'HC1' )
         call prival( yldpr(10), 'HC2' )
         call prival( yldpr(11), 'HC1C' )
         call prival( yldpr(12), 'HC2C' )
         call prival( dl(1),     'DL1' )
         call prival( dl(2),     'DL2' )
         call privec( sig, nstr, 'SIG' )
      end if
c
c...    Compresssion data
      call fpma( pc, nstr, yldpr )
c
c...     Calculate gradient
      call rab( pc, nstr, nstr, sig, 1, psig )
      yield = sqrt ( 0.5d0 * abs ( uv( sig, psig, nstr ) ) )
      call uvs( psig, nstr, 0.5d0 / yield, gradfc )
c
c...     Calculate d[P]/dk
      dpdk(1) = 2.d0 * ( yldpr(12) * yldpr(5) - yldpr(11) *
yldpr(6) )
     $          / yldpr(5) / yldpr(5)
      dpdk(nstr+2) = 2.d0 * ( yldpr(11) * yldpr(6) - yldpr(12)
*
     $               yldpr(5) ) / yldpr(6) / yldpr(6)
c
c                              T
c...     Calculate dfdk = <SIG> d[P]/dk <SIG> / ( 4 yield ) -
d(yield)/dk
      call rab( dpdk, nstr, nstr, sig, 1, w )
      tric = uv( sig, w, nstr )
      dfdk = 0.25d0 * tric / yield - 0.5d0 *
     $       ( yldpr(12) * yldpr(5) + yldpr(11) * yldpr(6) ) /
     $       sqrt( yldpr(5) * yldpr(6) )
c
c                                                    T
3
c...     Calculate d2fds2 = [P] / ( 2 yield ) - [P]<SIG>
<SIG>[P] / ( 4 yield )
      call rab( psig, nstr, 1, psig, nstr, w )
      call uvs( w, nstr*nstr, -0.25d0 / yield ** 3, w )
      call uvpws( w, pc, nstr*nstr, 0.5d0 / yield, d2fds2 )
c
c...     Calculate d2fdkds = d[P]/dk <SIG> /  ( 2 yield ) -
c...                                  T
3
c...                          ( <SIG> d[P]/dk <SIG> ) [P] <SIG>
/ ( 8 yield  )
      call rab( dpdk, nstr, nstr, sig, 1, w )
      call uvs( w, nstr*nstr, 0.5d0 / yield, w )
      call uvpws( w, psig, nstr, - tric / 8.d0 / yield ** 3,
dfdkds )
c
      call uvpws( gradfc, dfdkds, nstr, dl(2), gamma )
c
c
      if ( sw(3) ) then
         call primat( d2fds2, nstr, nstr, 'D2FDS2' )
         call primat( dpdk, nstr, nstr,   'DPDK'   )
         call privec( dfdkds, nstr,       'DFDKDS' )
         call privec( gamma, nstr,        'GAMMA'  )
         call prival( dfdk,               'DFDK'   )
      end if
c
c...     End compression
c
c...     Begin tension
c
      call fpraor( p, nstr, yldpr(3) )
      call feta( sig, yldpr, eta, nstr )
c
c...     Calculate gradients
      call ragror( p, sig, gradf,  yldpr, nstr, ldum )
      call fpraor( p, nstr, yldpr(4) )
      call rab( p, nstr, nstr, eta, 1, peta )
      tri = sqrt( 0.5d0 * uv( eta, peta, nstr ) )
      call ragror( p, sig, gradg,  yldpr, nstr, ldum )
      if ( sw(3) ) then
         call privec( gradf, nstr, 'GRADF' )
         call privec( gradg, nstr, 'GRADG' )
         call privec( eta, nstr,   'ETA  ' )
      end if
C
C...     calculate d2g/dsigma2 = d2gds2
      call rabt( peta, nstr, 1, peta, nstr, w )
      call uvs( w, nstr*nstr, -0.25d0 / tri**3, w )
      call uvpws( w, p, nstr*nstr, 0.5D0 / tri, d2gds2)
c
c...     calculate d2f/dkdsigma = d2gdk
      dnedk(1) = -yldpr(9)
      dnedk(2) = -yldpr(10)
      call rab( d2gds2, nstr, nstr, dnedk, 1, d2gdk )
      if ( sw(3) ) then
         call primat( d2gds2, nstr, nstr, 'D2GDS2' )
         call primat( stiff, nstr, nstr,  'STIFF'  )
         call privec( d2gdk, nstr,        'D2GDK'  )
      end if
c-------------------------------------------------------------
c...     Fill matrix [A] with sigma related lines
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      call uvpws( stiff, d2gds2, nstr*nstr, dl(1), w )
      call uvpws( w, d2fds2, nstr*nstr, dl(2), w )
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      k = 0
      do 10 j = 1, nstr
         do 10 i = 1, nstr
            k = k + 1
            a(i,j) = w(k)
   10 continue
c
      call uvpws( gradg, d2gdk, nstr, dl(1), w )
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
c
      do 20 i = 1, nstr
         a(i,nstr+1) = w(i)
   20 continue
c
      do 41 i = 1, nstr
         a(i,nstr+2) = gamma(i)
   41 continue
c-------------------------------------------------------------
c...     Calculate df1/dk = df1dk
      df1dk = uv( gradf, dnedk, nstr )
      if ( sw(3) ) then
         call prival( df1dk, 'DF1DK')
      end if
c-------------------------------------------------------------
c...     Fill matrix [A] with yield function 1 related line
      do 50 i = 1, nstr
         a(nstr+1,i) = gradf(i)
   50 continue
c
      a(nstr+1,nstr+1) = df1dk
c-------------------------------------------------------------
c...     Fill matrix [A] with yield function 2 related line
      do 55 i = 1, nstr
         a(nstr+2,i) = gradfc(i)
   55 continue
c
      a(nstr+2,nstr+2) = dfdk
c-------------------------------------------------------------
c
      if ( sw(3) ) call primat( a, nx, nx, 'A' )
c
      end
c
      DOUBLE PRECISION FUNCTION DASUM(N,DX,INCX)
C.....................................
C
C     TAKES THE SUM OF THE ABSOLUTE VALUES.
C
      DOUBLE PRECISION DX(1),DTEMP
      INTEGER I,INCX,M,MP1,N,NINCX
C
      DASUM = 0.0D0
      DTEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DTEMP = DTEMP + DABS(DX(I))
   10 CONTINUE
      DASUM = DTEMP
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,6)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DTEMP = DTEMP + DABS(DX(I))
   30 CONTINUE
      IF( N .LT. 6 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,6
        DTEMP = DTEMP + DABS(DX(I)) + DABS(DX(I + 1)) +
DABS(DX(I + 2))
     *  + DABS(DX(I + 3)) + DABS(DX(I + 4)) + DABS(DX(I + 5))
   50 CONTINUE
   60 DASUM = DTEMP
      RETURN
      END
C
      SUBROUTINE FILMA( KOD, A, N )
C
C.................................................
C
C...     TO FILL THE NON-FILLED HALF OF A SYMMETRIC MATRIX
C
C...       KOD.GT.0  - LOWER HALF IS FILLED
C
C...       KOD.LT.0    - UPPER HALF IS FILLED
C
C.............................................................
.........
C
      INTEGER          KOD, N
      DOUBLE PRECISION A(N,N)
      INTEGER          I, J
C
      IF (      KOD .GT. 0 ) THEN
C...        FILL LOWER HALF
         DO 15 I=1,N
            DO 10 J=I,N
               A(J,I)=A(I,J)
   10       CONTINUE
   15    CONTINUE
      ELSE IF ( KOD .LT. 0 ) THEN
C...        FILL UPPER HALF
         DO 25 I=1,N
            DO 20 J=I,N
               A(I,J)=A(J,I)
   20       CONTINUE
   25    CONTINUE
      END IF
      END
C
      SUBROUTINE FNCEXP( FVAL0, XULT, X, FVAL, GRAD )
C
C.................................................
C.............................................................
.........
C
      DOUBLE PRECISION FVAL0, XULT, X, FVAL, GRAD
C
      FVAL = FVAL0 * EXP( -X / XULT )
      GRAD = -FVAL / XULT
C
      RETURN
      END
C
      SUBROUTINE INVSYM( A, NA )
C
C.................................................
C...
C... INVERSE OF A(NA,NA) . A'S LOWER TRIANGLE IS FILLED.
C...  IN THE UPPER TRIANGLE WILL BE THE INVERSE.
C...  THE LOWER TRIANGLE IS DESTROYED.
C...
C.............................................................
.........
C
      DOUBLE PRECISION EPS
      PARAMETER        ( EPS=1.D-10 )
C
      INTEGER          NA
      DOUBLE PRECISION A(1), D, F, DM
      LOGICAL          ROWONE
C
      INTEGER          IHD, N, JRLBEG, IRY, JBEG, IHJ, IHJUP,
     $                 JRLEND, IRLB, JRY, IRL, JRL, K, IRLE,
IRH, JE,
     $                 JRH, JH, KE, NM1
C
C...     CHECK IF THIS IS A CORRECT CALL ....
      IF ( NA .LT. 1 ) RETURN
      IF ( NA .EQ. 1 ) THEN
         A(1) = 1.0D0/A(1)
         RETURN
      END IF
C
      DM = 0.D0
      IHD=0
      N=NA
      NM1=N-1
      JRLBEG=-NM1
      ROWONE=.TRUE.
      DO 60 IRY=1,NM1
         IHD=IHD+IRY
         D = A(IHD)
         DM = MAX( ABS( D ), DM )
         IF( ABS( D ) .LT. EPS*DM ) then
            print*, 'Cannot invert an ill-conditioned matrix'
            stop
         end if
         D=1.D0/D
         A(IHD)=D
C
         JBEG=IHD+1
         IHJ=IHD+N
         IHJUP=IRY
         IHD=JBEG+NM1-IRY
         JRLBEG=JRLBEG+N
         JRLEND=JRLBEG+IRY-2
         IRLB=JRLBEG
         DO 40 JRY=JBEG,IHD
            F=A(JRY)*D
            IRLB=IRLB+N
            IRL=IRLB
            IF( ROWONE ) GOTO 15
            DO 10 JRL=JRLBEG,JRLEND
               A(IRL)=A(IRL)-F*A(JRL)
               IRL=IRL+1
   10       CONTINUE
   15       CONTINUE
            A(IRL)=-F
            DO 20 K=JRY,IHD
              IHJ=IHJ+1
              A(IHJ)=A(IHJ)-F*A(K)
   20       CONTINUE
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
            A(JRY)=F
            IHJUP=IHJUP+1
            IHJ=IHJ+IHJUP
   40     CONTINUE
          IF ( ROWONE ) GOTO 50
          DO 45 JRL=JRLBEG,JRLEND
             A(JRL)=A(JRL)*D
   45     CONTINUE
          GOTO 60
   50     CONTINUE
          ROWONE=.FALSE.
   60 CONTINUE
      D = A(IHD+N)
      DM = MAX( ABS( D ), DM )
      IF( ABS( D ) .LT. EPS*DM ) then
         print*, 'Cannot invert ill-conditioned matrix'
         stop
      end if
      D = 1.D0/D
      A(IHD+N)=D
      IRLE=IHD+NM1
      DO 70 IRL=IRLB,IRLE
         A(IRL)=D*A(IRL)
   70 CONTINUE
C...
C BACK SUBSTITUTION STARTS NOW.
C...
      IRH=IHD-NM1
      DO 140 IRY=1,NM1
         JE=IHD-1
         KE=IHD+IRY-1
         DO 120 JRH=IRH,JE
            JH=JRH
            D=0.0D0
            DO 100 K=IHD,KE
              JH=JH+N
              D=D+A(K)*A(JH)
  100       CONTINUE
            A(JRH)=A(JRH)-D
  120    CONTINUE
         IHD=JE-N
         IRH=IRH-N
  140 CONTINUE
      RETURN
      END
c
      SUBROUTINE PRIIVL( IVAL, LABEL )
C
C.....................................................................
.........
C...
C...  PRINT INTEGER VALUE.
C...
C...  METHOD:
C...     THE VALUE IS IDENTIFIED BY A 1-6 CHARACTER LABEL WHICH
IS
C...     PRINTED IN FRONT.
C...
C...  INPUT:
C...     ARG.LIST:  IVAL   -VALUE TO BE PRINTED.
C...                LABEL  -ONE TO SIX CHARACTERS IN A6-FORMAT.
C...
C.....................................................................
.........
C
      INTEGER          IVAL
      CHARACTER*(*)    LABEL
C
C...     PRINT LABEL AND VALUE.
      WRITE ( *, 1 ) LABEL, IVAL
      RETURN
C
    1 FORMAT ( 1X, A, '=', I6 )
      END
c
      SUBROUTINE PRIMAT( RMAT, N, M, LABEL )
C
C...............................................
C...
C...  PURPOSE:
C...     TO PRINT AN REAL MATRIX ON THE LOGICAL OUTPUT UNIT.
C...
C...  METHOD:
C...     THE MATRIX IS IDENTIFIED BY A 1-6 CHARACTER LABEL
WHICH IS
C...     PRINTED ON TOP.
C...     THE MATRIX IS PRINTED ROW BY ROW, IN LINES OF TEN
VALUES,
C...     THE VALUES ARE FORMATTED IN A 'E11.3' FIELD.
C...     COLUMN- AND ROW-INDICES ARE PRINTED ABOVE AND AT THE
LEFT SIDE.
C...     A BLANK LINE IS INSERTED BETWEEN EACH ROW.
C...
C...  INPUT:
C...     ARG.LIST:  RMAT   -MATRIX TO BE PRINTED.
C...                N      -NUMBER OF ROWS.
C...                M      -NUMBER OF COLUMNS.
C...                LABEL  -ONE TO SIX CHARACTERS IN A6-FORMAT.
C.....................................................................
.........
C
C
      INTEGER          M, N
      DOUBLE PRECISION RMAT(N,M)
      CHARACTER*(*)    LABEL
C
      INTEGER          I, J
C
C...     PRINT LABEL AND COLUMN-INDICES.
      WRITE(*,1) LABEL, (J,J=1,M)
C
C...     PRINT ROWS, PRECEDED BY ROW-INDEX.
      DO 10 I = 1, N
         WRITE(LOUT,2) I, (RMAT(I,J),J=1,M)
   10 CONTINUE
      RETURN
C
    1 FORMAT(1X,A,':',10(I8,4X)/(8X,10(I8,4X)))
    2 FORMAT(1X,I7,1P,10E12.3/(8X,10E12.3))
      END
C
      SUBROUTINE PRIVAL( RVAL, LABEL )
C
C...............................................
C...
C...  PRINT REAL VALUE.
C...
C...  METHOD:
C...     THE VALUE IS IDENTIFIED BY A 1-6 CHARACTER LABEL WHICH
IS
C...     PRINTED IN FRONT.
C...
C...  INPUT:
C...     ARG.LIST:  RVAL   -VALUE TO BE PRINTED.
C...                LABEL  -ONE TO SIX CHARACTERS IN A6-FORMAT.
C...     /INOUT /:  LOUT   -OUTPUT LOGICAL UNITNUMBER.
C...
C.....................................................................
.........
C
      DOUBLE PRECISION RVAL
      CHARACTER*(*) LABEL
C
C...     PRINT LABEL AND VALUE.
      WRITE ( *, 1 ) LABEL, RVAL
      RETURN
C
    1 FORMAT ( 1X, A, '=', 1P, E12.3 )
      END
C
      SUBROUTINE PRIVEC( RVEC, N, LABEL )
C
C...............................................
C...
C...  PRINT REAL VECTOR (ARRAY).
C...
C...  METHOD:
C...     THE VECTOR IS IDENTIFIED BY A 1-6 CHARACTER LABEL
WHICH IS
C...     PRINTED IN FRONT.
C...
C...  INPUT:
C...     ARG.LIST:  RVEC   -VECTOR TO BE PRINTED.
C...                N      -NUMBER OF VALUES IN VECTOR.
C...                LABEL  -ONE TO SIX CHARACTERS IN A6-FORMAT.
C...
C.....................................................................
.........
C
C
      INTEGER LIN, LOUT, N
      DOUBLE PRECISION RVEC(N)
      CHARACTER*(*) LABEL
C
C...     PRINT LABEL AND VALUES.
      WRITE(*,1) LABEL, RVEC
      RETURN
C
    1 FORMAT(1X,A,':',1P,10E12.3/(8X,10E12.3))
      END
C
      SUBROUTINE RAB(A,N,M,B,L,R)
C
C...............................................
C...
.
C...  PURPOSE :  R = A * B
.
C...
.
C...  METHOD  :  USE FUNCTION VINPRO TO COMPUTE THE INNER-
PRODUCT OF
C...             A ROW OF A AND A COLUMN OF B.           .
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
C...
.
C...  REAL DIMENSIONS : A(N,M) B(M,L), R(N,L)
.
C...
.
C.....................................................
.........
C
      INTEGER          N, M, L
      DOUBLE PRECISION A(1), B(1), R(1), VINPRO
      INTEGER          I, IR, J, ML
C
      ML=M*L
      DO 130 I=1,N
         IR=I
         DO 110 J=1,ML,M
            R(IR) = VINPRO( A(I), N, B(J), 1, M )
            IR=IR+N
  110    CONTINUE
  130 CONTINUE
      RETURN
      END
C
      SUBROUTINE RABT( A, N, M, B, L, R )
C
C.............................................
C...
.
C...                    T
.
C...  PURPOSE :  R = A*B
.
C...
.
C...  METHOD  :  CALL FUNCTION VINPRO TO COMPUTE INNER-PRODUCT
OF
C...             ROW OF A AND ROW OF B.                  .
C...
.
C...  REAL DIMENSIONS : A(N,M), B(L,M), R(N,L)
.
C...
.
C.....................................................
.........
C
      INTEGER          N, M, L
      DOUBLE PRECISION A(1), B(1), R(1), VINPRO
      INTEGER          IR, J, I
C
      IR=0
      DO 120 J=1,L
         DO 110 I=1,N
            IR = IR + 1
            R(IR) = VINPRO( A(I), N, B(J), L, M )
  110    CONTINUE
  120 CONTINUE
      RETURN
      END
C
      SUBROUTINE RATBA( A, N, M, B, R, H, K )
C
C.............................................
C...
C...                                        T
C...  PURPOSE :  TRIPLE MATRIX PRODUCT R = A  * B * A
C...
C...  METHOD : USE FUNCTIONS VINPRO AND UV TO COMPUTE
C...
C...  REAL DIMENSIONS : A(N,M), B(N,N), R(K,M), H(N)
C...
C.....................................................
.........
C
      INTEGER          N, M, K
      DOUBLE PRECISION A(1), B(1), H(1), R(1), VINPRO, UV
      INTEGER          MN, IR, J, L, I
C
      CALL RSET( 0.D0, R, K*M )
C
      MN = M*N
      IR = 1
      DO 100 J = 1, MN, N
         DO 20 L = 1, N
            H(L) = VINPRO( B(L), N, A(J), 1, N )
   20    CONTINUE
         DO 40 I = 1, MN, N
            R(IR) = R(IR) + UV( H(1), A(I), N )
            IR = IR + 1
   40    CONTINUE
         IR = IR + K - M
  100 CONTINUE
      RETURN
      END
C
      SUBROUTINE RMOVE( RFROM, RTO, N )
C
C.............................................
C...
C...  PURPOSE:
C...     MOVE N REALS FROM  < RFROM >  TO < RTO > :
C...
C...                N               N
C...         < RTO >   :=  < RFROM >
C...                1               1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  RFROM(1:N), N
C...
C...  OUTPUT:
C...     ARG.LIST:  RTO(1:N)
C...
C.....................................................
.........
C
C
      INTEGER I, N
      DOUBLE PRECISION RFROM(N), RTO(N)
C
      DO 10 I = 1, N
         RTO(I) = RFROM(I)
   10 CONTINUE
      RETURN
      END
C
      SUBROUTINE RSET( RVAL, RAR, N )
C
C.............................................
C...
C...  PURPOSE:
C...     INITIALIZE REAL ARRAY:
C...
C...               N
C...         < RAR >   :=  RVAL
C...               1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  RVAL, N.
C...
C...  OUTPUT:
C...     ARG.LIST:  RAR(1:N)
C...
C.....................................................
.........
C
C
      INTEGER I, N
      DOUBLE PRECISION RAR(N), RV, RVAL
C
      RV = RVAL
      DO 10 I = 1, N
         RAR(I) = RV
   10 CONTINUE
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION UV( U, V, N )
C
C.............................................
C...
C...  INNER PRODUCT:
C...
C...                     N
C...         UV  :=  SIGMA (  < U >  *  < V >  )
C...                     1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  U(1:N), V(1:N), N
C...
C...  OUTPUT:
C...     FUNC.VAL:  UV
C...
C.....................................................
.........
C
      INTEGER          N
      DOUBLE PRECISION R, U(N), V(N)
      INTEGER          I
C
      R = 0.D0
      DO 10 I = 1, N
         R = R  +  U(I) * V(I)
   10 CONTINUE
      UV = R
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
      RETURN
      END
C
      SUBROUTINE UVMW( V, W, N, U )
C
C.....................................................
C...
C...  PURPOSE:
C...     SUBTRACT TWO VECTORS:
C...
C...              N           N          N
C...         < U >   :=  < V >   -  < W >
C...              1           1          1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  V(1:N), W(1:N), N
C...
C...  OUTPUT:
C...     ARG.LIST:  U(1:N)
C...
C.....................................................................
.........
C
      INTEGER          N
      DOUBLE PRECISION U(N), V(N), W(N)
      INTEGER          I
C
      DO 10 I = 1, N
         U(I) = V(I) - W(I)
   10 CONTINUE
      RETURN
      END
C
      SUBROUTINE UVPW( V, W, N, U )
C
C.....................................................
C...
C...  PURPOSE:
C...     ADD TWO VECTORS:
C...
C...              N           N          N
C...         < U >   :=  < V >   +  < W >
C...              1           1          1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  V(1:N), W(1:N), N
C...
C...  OUTPUT:
C...     ARG.LIST:  U(1:N)
C...
C.....................................................................
.........
C
      INTEGER          N
      DOUBLE PRECISION U(N), V(N), W(N)
      INTEGER          I
C
      DO 10 I = 1, N
         U(I) = V(I) + W(I)
   10 CONTINUE
      END
C
      SUBROUTINE UVS( V, N, S, U )
C
C.....................................................
C...
C...  PRODUCT OF VECTOR AND SCALAR:
C...
C...              N           N
C...         < U >   :=  < V >   *  S
C...              1           1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  V(1:N), N, S
C...
C...  OUTPUT:
C...     ARG.LIST:  U(1:N)
C...
C.....................................................................
.........
C
      INTEGER          N
      DOUBLE PRECISION S, U(N), V(N)
      INTEGER          I
C
      DO 10 I = 1, N
         U(I) = V(I) * S
   10 CONTINUE
      END
c
      SUBROUTINE Drapex( SE, stiff, NSTR, yldpr, sig, depsp )
C
C.....................................................
C...
C...  PURPOSE:
C...     TO CALCULATE THE CONSISTENT TANGENT OF THE
'ORTHOTROPIC RANKINE' MODEL
c...     IN THE APEX - Version 2
C...
C...
C.....................................................................
.........
C
C...     EXTERNALS
      DOUBLE PRECISION UV
C
C...     PARAMETERS
      DOUBLE PRECISION TOLER
      PARAMETER        ( TOLER = 1.D-12 )
      INTEGER          MSTR
      PARAMETER        ( MSTR=6 )
C
C...     COMMON DEFINITION
      LOGICAL SW
      COMMON /SWITCH/ SW(6)
C
C...     ARGUMENTS
      INTEGER          NSTR
      DOUBLE PRECISION SE(NSTR,*), stiff(NSTR,*)
C
C...     LOCAL VARIABLES
      DOUBLE PRECISION SIG(*)
      DOUBLE PRECISION w((mstr+1)*(mstr+1)), dnedk(mstr),
yldpr(*),
     $                 a(mstr+1,mstr+1), uni(mstr*mstr),
q(mstr*mstr),
     $                 DEPSP(*), dkdep(mstr), pi(mstr),
     $                 u(mstr), aa(MSTR,MSTR), bb(MSTR,MSTR),
     $                 qdepsp(mstr)
      INTEGER          IERR, IDUM(mstr+1), i, j, k
      DATA IDUM / 1, 2, 3, 4, 5, 6, 7 /
C
      call rset( 0.d0, dnedk, nstr )
      call rset( 0.d0, pi, nstr )
      call rset( 0.d0, a, (mstr+1)*(mstr+1) )
c
      if ( sw(3) ) CALL PRIVEC( DEPSP, nstr, 'DEPSP ' )
      if ( sw(3) ) CALL PRIVEC( SIG, nstr, 'SIG   ' )
c
c...     Fill unimax
      call unimax( uni, nstr )
      IF ( SW(3) ) then
         CALL PRIMAT( stiff, nstr, nstr, 'stiff')
      end if
c-------------------------------------------------------------
c...     FILL MATRIX A WITH SIGMA RELATED LINES
      k = 0
      do 10 j = 1, nstr
         do 10 i = 1, nstr
            k = k + 1
            a(i,j) = uni(k)
   10 continue
c
      dnedk(1) = yldpr( 9)
      dnedk(2) = yldpr(10)
      call fabra( aa, bb, nstr, stiff )
      call rab( aa, nstr, nstr, dnedk, 1, w )
      do 30 i = 1, nstr
         a(i,nstr+1) = -w(i)
   30 continue
c
c-------------------------------------------------------------
c
c...     Calculate d(deltak)/depsp = dkdep
      CALL fpraor( q, NSTR, 0.d0 )
      do 40 i = 4, nstr
         q((i-1)*nstr+i) = 0.5d0 / yldpr(4)
   40 continue
      pi(1) = 1.d0
      pi(2) = 1.d0
      call rab( q, nstr, nstr, DEPSP, 1, qdepsp )
      if ( abs( uv( DEPSP, qdepsp, nstr) ) .lt. 1.d-16 ) then
         call uvs( pi, nstr, 0.5d0, dkdep )
      else
         call uvs( qdepsp, nstr, 0.5d0 /
     $     sqrt( 0.5 * abs( uv ( DEPSP, qdepsp, nstr) ) ),
dkdep )
         call uvpws( dkdep, pi, nstr, 0.5d0, dkdep )
      end if
c
      IF ( SW(3) ) then
         CALL PRIVEC( dkdep, nstr, 'dkdep' )
      end if
c-------------------------------------------------------------
c...     FILL MATRIX A WITH SOFTENING RELATED LINE
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217

```
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
      call rab( stiff, nstr, nstr, dkdep, 1, w )
      do 60 i = 1, nstr
         a(nstr+1,i) = w(i)
   60 continue
c
      a(nstr+1,nstr+1) = 1.d0
c
c-------------------------------------------------------
c
      IF ( SW(3) ) CALL PRIMAT( A, nstr+1, nstr+1, 'A' )
C
      CALL INVPGB( A, nstr+1, IDUM, W, TOLER, IERR )
      IF ( IERR .LT. 0 ) then
         print*, 'INVERSION SYSTEM "A" FAILED'
         stop
      end if
      IF ( SW(3) ) THEN
         CALL PRIMAT( A, nstr+1, nstr+1, 'A-1' )
      END IF
c
      do 90 i = 1, nstr
         u(i) = dkdep(i)
   90 continue
c
      k = 0
      do 80 i = nstr+1, nstr+1
         do 80 j = 1, nstr
            k = k + 1
            w(k) = a(j,i)
   80 continue
c
      IF ( SW(3) ) CALL PRIMAT( w, nstr, 1, 'A-1m*1' )
c
      call rab( w, nstr, 1, u, nstr, se )
      IF ( SW(3) ) CALL PRIMAT( SE, NSTR, NSTR, 'SE-TAN' )
c
      call invsym( stiff, nstr )
      call filma( +1, stiff, nstr )
c
      k = 0
      do 85 i = 1, nstr
         do 85 j = 1, nstr
            k = k + 1
            w(k) = a(j,i)
   85 continue
c
      IF ( SW(3) ) CALL PRIMAT( w, nstr, nstr, 'A-1m*m' )
c
      call rab( w, nstr, nstr, bb, nstr, aa )
      call rab( aa, nstr, nstr, stiff, nstr, w )
      call uvpw( se, w, nstr*nstr, se )
c
c...    Avoid a zero stiffness matriz in the z direction
      if ( se(3,3) .eq. 0.d0 ) se(3,3) = 1.d-7
c
      IF ( SW(3) ) CALL PRIMAT( SE, NSTR, NSTR, 'SE-TAN' )
C
      END
C
      SUBROUTINE UVPWS( V, W, N, S, U )
C
C...............................................
C...
C...          N           N          N
C...     < U >   :=  < V >   +  < W >  * S
C...          1           1          1
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  V(1:N), W(1:N), N, S
C...
C...  OUTPUT:
C...     ARG.LIST:  U(1:N)
C...
C.............................................................
.........
C
      INTEGER          N
      DOUBLE PRECISION S, U(1), V(1), W(1)
      INTEGER          I
C
      IF ( N .LE. 0 ) RETURN
      DO 10 I = 1, N
         U(I)  =  V(I)  +  W(I) * S
   10 CONTINUE
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION VINPRO( U, IU, V, IV, N )
C
C...............................................
C...
C...  PURPOSE:
C...     INNER PRODUCT OF TWO VECTORS:
C...
C...                                N,IU          N,IV
C...    VINPRO  :=  SIGMA (  < U >      *  < V >     )
C...                                1             1
C...
C...     IU, AND IV ARE INDEX INCREMENTS FOR <U> AND <V>, IF
BOTH
C...     IU AND IV ARE 1, BETTER USE FUNCTION UV.
C...
C...  METHOD:
C...     FORTRAN DO-LOOP
C...
C...  INPUT:
C...     ARG.LIST:  U(.), IU, V(.), IV, N
C...
C...  OUTPUT:
C...     FUNC.VAL:  VINPRO
C...
C.............................................................
.........
C
      INTEGER          IU, IV, N
      DOUBLE PRECISION R, U(1), V(1)
      INTEGER          NU, JV, I
C
      R  = 0.D0
      NU = N*IU
      JV = 1
      DO 10 I = 1, NU, IU
         R  = R  + U(I) * V(JV)
         JV = JV + IV
   10 CONTINUE
      VINPRO = R
      END
```


Azurém, P – 4800-058 Guimarães Tel. +351 253 510200 • Fax +351 253 510217