Elastoviscoplastic Finite Element analysis in 100 lines of Matlab

C. Carstensen * and R. Klose †

Received 3 July, 2002

Abstract — This paper provides a short Matlab implementation with documentation of the $P_1$ finite element method for the numerical solution of viscoplastic and elastoplastic evolution problems in 2D and 3D for von-Mises yield functions and Prandtl-Reuß flow rules. The material behaviour includes perfect plasticity as well as isotropic and kinematic hardening with or without a viscoplastic penalisation in a dual model, i.e. with displacements and the stresses as the main variables. The numerical realisation, however, eliminates the internal variables and becomes displacement-oriented in the end. Any adaption from the given three time-depending examples to more complex applications can easily be performed because of the shortness of the program and the given documentation. In the numerical 2D and 3D examples an efficient error estimator is realized to monitor the stress error.

Keywords: finite element method, viscoplasticity, elastoplasticity, Matlab

1. INTRODUCTION

Elastoplastic time-evolution problems usually require universal, complex, commercial computer codes running on workstations or even super computers [11,15]. The argument of keeping commercial secrets hidden inside a black-box, leaves the typical user without any idea what exactly is going on behind the program’s user-friendly surface. The difference between a mathematical and a numerical model is fuzzy. Often, users do not care about nasty details such as quadrature rules or the exact material laws realised. But sometimes it does matter whether a regularised or penalised discrete model is solved, how the termination of an iterative process is steered, and what post-processing led to both, equilibrium and admissibility of the approximate stress field. In particular, if the solution process is part of guaranteed error control, it is quite important to know if discrete equilibrium is fulfilled exactly or not. Finally, the use of more than one black-box in a chain is likely to be less efficient. In summary, to aim efficiency (e.g. by well-adapting automatic mesh-refinements) and reliability we have to prevent, at least confess, all

*Institute for Applied Mathematics and Numerical Analysis, Vienna University of Technology, Wiedner Hauptstraße 8-10, A-1040 Vienna, Austria
†Mathematisches Seminar, Christian-Albrechts-Universität zu Kiel, Ludewig-Meyn-Str. 4, D-24098 Kiel, Germany
the numerical crimes. Those range from modelling errors (e.g. data errors, coarse geometrical resolution, wrong mathematical model, etc.) to variational crimes (e.g. material laws fulfilled at discrete points only or just in some approximate sense). This work aims a clear presentation of clean algorithms to compute a few model examples. Their implementation in Matlab (as one possible higher scientific computing tool) is the content of this paper. The provided software may serve as both, reference material for future numerical examples and as scientifically proved public domain source for modifications in other applications for research and educational purposes.

This paper continues our efforts on the Laplace and Navier–Lamé problem [2,3] towards more complicated material behaviour to advertise and to open the field of computational plasticity to a larger community of applied mathematicians and engineers. Within a small strain framework, the rheological models of this work include perfect plasticity, elastoplastic evolution with kinematic or isotropic hardening in the setting of von-Mises yield conditions and a Prandl-Reuß flow rule [11,15]. All variants are available for a viscoplastic regularisation with a Yosida-regularisation, known as viscoplasticity due to Perzyna.

The time discretisation yields a minimisation problem under severe constraints for each time step. The discrete problem is solved with a Newton-Raphson scheme and Matlab’s standard direct solver. The main principle of the presented method is, that, in each time step, the stress tensor is expressed explicitly and applied in Cauchy’s first law of motion.

We expect that the reader is familiar with Matlab as well as with linear elasticity and otherwise refer, e.g. to [3,13].

The paper is organised as follows. In Section 2 we introduce the viscoplastic and elastoplastic problem and their mathematical model. In Section 3 we perform a time discretisation and calculate the explicit expression of the stress tensor for the different models of hardening. In Section 4 the discretisation in space together with the data representation of the triangulation, the Dirichlet and Neumann boundary is considered. Furthermore the iterative solution of the discrete problem is explained. Section 5 concerns the calculation of the stiffness matrix. Section 6 discusses post processing routines, one for previewing the numerical solution and one for an a posteriori error estimation. Numerical examples in Section 7 illustrate the usage of the provided tools for four benchmark examples in two and three space dimensions.

The programs are written for Matlab 6.1 but adaption for previous versions is possible. For a numerical calculation it is necessary to run the program main.m with (user-specified) files coordinates.dat, elements.dat, dirichlet.dat, neumann.dat, as well as the subroutines f.m, g.m, and uJ.m. Their meaning and usage is explained throughout the paper and examples are provided in the later sections. The graphical representation is performed with the function show.m and the error is estimated in aposteriori.m. All files with the code and the data for all examples can be downloaded from www.math.tuwien.ac.at/~carsten/.
2. RHEOLOGICAL MODEL

2.1. Material models

The material body occupies a bounded Lipschitz domain \( \Omega \) in \( \mathbb{R}^d \) and deforms any time \( t, 0 \leq t \leq T \). Given a displacement field \( u = u(t) \in H^1(\Omega)^d \) (i.e., the functional matrix \( Du \) exists almost everywhere in \( \Omega \), is measurable, and satisfies \( \int_\Omega |Du|^2 \, dx < \infty \)) the total strain reads

\[
\varepsilon(u) = (Du + Du^T)/2, \quad (\varepsilon(u))_{jk} = (u_{j,k} + u_{k,j})/2, \quad j,k = 1, \ldots, d.
\]

In the engineering literature, \( \varepsilon(u) \) is known as the linear Green strain tensor. The default model for small-strain plasticity involves an additive split

\[
\varepsilon = \mathbf{e}(\sigma) + \mathbf{p}(\xi)
\]

of the total strain in an elastic part, \( \mathbf{e}(\sigma) = C^{-1}\sigma \), and an irreversible part \( \mathbf{p}(\xi) \). The plastic strain \( \mathbf{p}(\xi) \) depends on further internal variables \( \xi \). The stress field \( \sigma = \sigma(t) \in L^2(\Omega;\mathbb{R}^{d \times d}_{\text{sym}}) \) of the Cauchy stress tensor \( \sigma(x,t) \) is linked to the elastic, reversible strain. The linear relation is provided by \( C^{-1} \), the compliance tensor, which is the inverse of the fourth order elasticity tensor \( C \). In the numerical examples, a two-parameter model (\( \lambda \) and \( \mu \) are the Lamé constants) in linear isotropic, homogeneous elasticity, \( \mathbf{C} = \lambda \mathbf{tr}(\mathbf{e})\mathbf{I} + 2\mu \mathbf{e} \), is employed.

The evolution law for the plastic strain \( \mathbf{p} \) is more involved and requires the concept of admissible stresses, a yield function, and an associated flow rule.

The kinematic variables \( \mathbf{p} \) and \( \xi \) form the generalised strain \( \mathbf{P} = (\mathbf{p}, \xi) \). The corresponding generalised stress reads \( \Sigma = (\sigma, \alpha) \), where \( \alpha \in \mathbb{R}^{m \times m}_{\text{sym}} \) describes internal stresses.

We denote by \( K \) the set of admissible stresses, which is a closed, convex set, containing 0, and is defined by

\[
K = \{ \Sigma : \Phi(\Sigma) \leq 0 \}.
\]

The yield function \( \Phi \) describes admissible stresses by \( \Phi(\Sigma) \leq 0 \) and models either perfect plasticity or hardening. The dissipation functional \( \varphi(\Sigma) \), defined by

\[
\varphi(\sigma, \alpha) = \frac{1}{2\nu} \inf \{ |(\sigma - \tau, \alpha - \beta)|^2 : (\tau, \beta) \in \mathbb{R}^{d \times d}_{\text{sym}} \times \mathbb{R}^{m \times m}_{\text{sym}}, \quad \Phi(\tau, \beta) \leq 0 \}
\]

approaches the indicator function of the set \( K \), namely

\[
\varphi(\Sigma) = \begin{cases} 0, & \Phi(\Sigma) \leq 0 \\ \infty, & \Phi(\Sigma) > 0 \end{cases}
\]

as \( \nu \to 0 \). The time derivative of \( \mathbf{P} \) is given by the flow rule

\[
\mathbf{P} \in \partial \varphi(\Sigma) = \{(\sigma^*, \alpha^*) \in \mathbb{R}^{d \times d}_{\text{sym}} \times \mathbb{R}^{m \times m}_{\text{sym}} : \forall (\tau, \beta) \in \mathbb{R}^{d \times d}_{\text{sym}} \times \mathbb{R}^{m \times m}_{\text{sym}}, \quad \varphi(\sigma, \alpha) + \sigma^* : (\tau - \sigma) + \alpha^* : (\beta - \alpha) \leq \varphi(\tau, \beta) \}.
\]
Here, $A : B = \sum_{j,k=1}^n A_{jk} B_{jk}$ denotes the scalar products of (symmetric) matrices $A, B \in \mathbb{R}^{m \times m}$. For $\nu > 0$ the flow rule results in

$$\dot{\mathbf{p}} = (\dot{p}, \dot{\xi}) = 1/\nu (\sigma - \tau, \alpha - \beta)$$

(2.4)

for $(\tau, \beta)$ uniquely determined as the projection of $(\sigma, \alpha)$ onto $K$, i.e., $(\tau, \beta) \in K$ with $(\tau, \beta) := \Pi(\sigma, \alpha)$ uniquely determined by

$$\text{dist}((\sigma, \alpha), K) = \| (\sigma - \tau, \alpha - \beta) \| = \| (\sigma, \alpha) - \Pi(\sigma, \alpha) \|$$

$$\leq \inf \{ \| (\sigma - \tau', \alpha - \beta') \| : (\tau', \beta') \in \mathbb{R}^{d \times d} \times \mathbb{R}^{m \times m} \}, \quad \Phi(\tau', \beta') \leq 0 \}.$$  

(2.5)

The following examples involve the von-Mises yield function in various modifications.

**Example 2.1 (Viscoplasticity).** In the case of perfect viscoplasticity, there is no hardening and the internal variables $\xi$ are absent. The yield function is given by

$$\Phi(\sigma) = \| \text{dev}(\sigma) \| - \sigma_y.$$  

(2.6)

With (2.6) in (2.4) and $(s)_+ := \max \{ 0, s \}, s \in \mathbb{R}$ we obtain

$$\dot{p} = 1/\nu (1 - \sigma_y/\| \text{dev}(\sigma) \|)_+ \text{dev}(\sigma).$$  

(2.7)

**Example 2.2 (Viscoplasticity with isotropic hardening).** Isotropic hardening is characterised by the scalar hardening parameter $\alpha \geq 0$ and the constant $H > 0$, the modulus of hardening. The yield function is

$$\Phi(\sigma, \alpha) = \| \text{dev}(\sigma) \| - \sigma_y (1 + H \alpha).$$  

(2.8)

With (2.8) in (2.4) we obtain

$$\left( \frac{\dot{p}}{\dot{\xi}} \right) = \frac{1}{\nu 1 + H^2 \sigma_y^2} \left[ \left( 1 - \frac{(1 + \alpha H) \sigma_y}{\| \text{dev}(\sigma) \|} \right)_+ \left( \frac{\text{dev}(\sigma)}{\sigma_y \| \text{dev}(\sigma) \|} \right) \right].$$  

(2.9)

In our model the plastic part of the free energy is given by $\psi^p = \frac{1}{2} H_1 \xi^2$. The internal force is defined by $\alpha = -\partial \psi^p / \partial \xi$, which means $\alpha = -H_1 \xi$, where $H_1$ is a positive hardening parameter.

**Example 2.3 (Viscoplasticity with linear kinematic hardening).** In kinematic hardening the yield function is given by

$$\Phi(\sigma, \alpha) = \| \text{dev}(\sigma) \| - \| \text{dev}(\sigma - \alpha) \|- \sigma_y.$$  

(2.10)

With (2.10) in (2.4) we obtain

$$\left( \frac{\dot{p}}{\dot{\xi}} \right) = \frac{1}{2\nu} \left[ \left( 1 - \frac{\sigma_y}{\| \text{dev}(\sigma - \alpha) \|} \right)_+ \left( \frac{\text{dev}(\sigma - \alpha)}{\sigma_y \| \text{dev}(\sigma - \alpha) \|} \right) \right].$$  

(2.11)
Elastoviscoplastic FE analysis

For linear kinematic hardening, the plastic part of the free energy has the form $\psi^p = \frac{1}{2}k_1|\xi|^2$. The internal force is defined by $\alpha = -\partial \psi^p / \partial \xi$, which means $\alpha = -k_1\xi$ with a positive parameter $k_1$.

**Remark 2.1 (Elastoplasticity for $\nu \to 0$).** In the present situation the positive number $\nu$ may be seen as a viscose penalty which leads to equalities (such as (2.7), (2.9), (2.11)). The elastoplastic limit for $\nu \to 0$ leads in (2.3) to a variational inequality [11,15]. This model will be included subsequently as one may consider $\nu \to 0$ in the formulae below.

### 2.2. Equilibrium equations

The stress field $\sigma \in L^2(\Omega; \mathbb{R}^{d \times d})$ and the volume force $f \in L^2(\Omega; \mathbb{R}^d)$ are related by the local quasi-static balance of forces,

$$\text{div} \sigma + f = 0.$$  \hspace{1cm} (2.12)

On some closed subset $\Gamma_D$ of the boundary with positive surface measure, we assume Dirichlet conditions while we have Neumann boundary conditions on the (possible empty) part $\Gamma_N$. The two components of the displacement $u$ need not satisfy simultaneously Dirichlet or Neumann conditions, i.e., $\Gamma_D$ and $\Gamma_N$ need not be disjoint. With $\mathbf{M} \in L^\infty(\Gamma_D)^{d \times d}$, $\mathbf{w} \in H^1(\Omega)^d$, and a surface force $g \in L^2(\Gamma_N)$ we have

$$\mathbf{M} \mathbf{u} = \mathbf{w} \quad \text{on} \ \Gamma_D, \quad \mathbf{\sigma} \mathbf{n} = g \quad \text{on} \ \Gamma_N.$$  \hspace{1cm} (2.13)

Let $\Pi$ denote the projection onto the set of admissible stresses $K$. The plastic problem is then determined by the weak formulation: Seek $\mathbf{u} \in H^1(\Omega)^d$ that satisfies $\mathbf{M} \mathbf{u} = \mathbf{w}$ on $\Gamma_D$ that, for all $\mathbf{v} \in H^1(\Omega)^d := \{ \mathbf{v} \in H^1(\Omega)^d : \mathbf{M} \mathbf{v} = 0 \text{ on } \Gamma_D\}$,

$$\int_\Omega \mathbf{\sigma}(\mathbf{u}) : \mathbf{\varepsilon}(\mathbf{v}) \, dx = \int_\Omega \mathbf{f} \cdot \mathbf{v} \, dx + \int_{\Gamma_N} \mathbf{g} \cdot \mathbf{v} \, ds$$

$$\left[ \mathbf{\varepsilon}(\mathbf{u}) - C^{-1} \mathbf{\sigma} \right]_\mathbf{\xi} (\mathbf{\alpha}) = \frac{1}{\nu} \left[ \mathbf{\sigma} - \Pi \mathbf{\sigma} \right]_\mathbf{\alpha} - \Pi \mathbf{\alpha} \quad \text{a.e. in } \Omega.$$  \hspace{1cm} (2.14)

### 3. TIME DISCRETISATION AND ANALYTIC EXPRESSION OF THE STRESS TENSOR

#### 3.1. Time discretisation scheme

A generalised midpoint rule serves as a time-discretisation. In each time step there is a spatial problem with given variables $(u(t), \sigma(t), \alpha(t))$ at time $t_0$ denoted as $(u_0, \sigma_0, \alpha_0)$ and unknowns at time $t_1 = t_0 + k$ denoted as $(u_1, \sigma_1, \alpha_1)$. Time derivatives are replaced by backward difference quotients. The time discrete problem reads: Seek $\mathbf{u}_\Delta \in H^1(\Omega)^d$ that satisfies $\mathbf{M} \mathbf{u}_\Delta = \mathbf{w}$ on $\Gamma_D$ that, for all
In this subsection we derive explicit expressions for the stress tensor \( \sigma_{\theta} \) and \( \sigma_{1} \). According to (3.4a) the stress tensor \( \sigma_{\theta} \) in terms of \( \text{tr} \sigma_{\theta} \) and \( \text{dev} \sigma_{\theta} \) we note that \( \sigma_{\theta} = \lambda \text{tr} e_{\theta} I + 2\mu e_{\theta} \). There are constants \( \alpha, \beta, \gamma \) and \( \delta \) such that
\[
\sigma_{\theta} = \gamma \text{tr} e_{\theta} I + \delta \text{dev} e_{\theta}, \quad e_{\theta} = \alpha \text{tr} \sigma_{\theta} I + \beta \text{dev} \sigma_{\theta}.
\]
Inserting of (3.4a) in (3.4b) we have
\[
e_{\theta} = d\alpha \gamma \text{tr} e_{\theta} I + \beta \delta \text{dev} e_{\theta}, \quad \alpha = 1/(d^{2}\gamma), \quad \beta = 1/\delta.
\]
According to (3.4a) the stress tensor \( \sigma_{\theta} \) reads
\[
\sigma_{\theta} = \gamma \text{tr} e_{\theta} I + \delta (e_{\theta} - 1/d \text{tr} e_{\theta} I) = \left( \frac{\gamma - \delta}{d} \right) \text{tr} e_{\theta} I + \delta e = \lambda \text{tr} e_{\theta} I + 2\mu e_{\theta}.
\]
This shows \( \delta = 2\mu \) and \( \gamma = \lambda + 2\mu /d \). Inserting this in (3.4a) we deduce
\[
\text{C}^{-1} \sigma_{\theta} = \frac{1}{d^{2}\lambda + 2d\mu} \text{tr} \sigma_{\theta} I + \frac{1}{2\mu} \text{dev} \sigma_{\theta}.
\]

### 3.2. Analytic expression of the stress tensor

In this subsection we derive explicit expressions for the stress tensor \( \sigma_{\theta} \) for the different cases of hardening. With the connection \( \sigma_{\theta} = \sigma_{\theta}(A(u_{\theta})) \) between \( \sigma_{\theta} \) and \( u_{\theta} \) the elastoplastic problem is determined by a nonlinear variational problem: Seek \( u_{\theta} \in H^{1}(\Omega)^{d} \) that satisfies \( M u_{\theta} = w \) on \( D_{0} \) and, for all \( v \in H^{1}_{D}(\Omega)^{d} := \{ v \in H^{1}(\Omega)^{d} : M v = 0 \text{ on } D_{0} \} \),
\[
\int_{\Omega} \sigma : \varepsilon (u_{\theta} - u_{0}) + \text{C}^{-1} \sigma_{0} : \varepsilon (v) \, dx = \int_{\Omega} f_{\theta} \cdot v \, dx + \int_{D_{0}} g_{\theta} \cdot v \, ds.
\]
We solve (3.3) for \( \sigma_{\theta} \) in the flow laws of Examples 2.1–2.3.
Theorem 3.1 (Viscoplasticity and plasticity without hardening). For perfect viscoplasticity and perfect plasticity there exist constants $C_1$, $C_2$, and $C_3$ such that the stress tensor reads
\[
\sigma = C_1 \text{tr}(\dot{\vartheta} k A) I + [C_2 + C_3/|\text{dev} \vartheta k A|] \text{dev} \vartheta k A. \tag{3.9}
\]
For perfect viscoplasticity these constants are
\[
C_1 := \lambda + 2\mu / d, \quad C_2 := \nu / (\beta \nu + \dot{\vartheta}), \quad C_3 := \vartheta k \sigma_y / (\beta \nu + \dot{\vartheta}) \tag{3.10}
\]
and in the limit $\nu \to 0$ for perfect plasticity these constants read
\[
C_1 := \lambda + 2\mu / d, \quad C_2 := 0, \quad C_3 := \sigma_y. \tag{3.11}
\]
The plastic phase occurs for
\[
|\text{dev} \vartheta k A| > \frac{\sigma_y}{2\mu}. \tag{3.12}
\]
In the elastic phase the stress tensor reads
\[
\sigma = C_1 \text{tr}(\dot{\vartheta} k A) I + 2\mu \text{dev} \vartheta k A. \tag{3.13}
\]

Proof. The discretised version of the flow law reads
\[
A = C - \frac{1}{\dot{\vartheta} k} \left( \frac{1 - \sigma_y}{|\text{dev} \sigma_\vartheta|} \right) \text{dev} \sigma_\vartheta. \tag{3.14}
\]
We consider the plastic phase $(1 - \sigma_y / |\text{dev} \sigma_\vartheta|) > 0$. With $C^{-1} \sigma_\vartheta = \alpha \text{tr} \sigma_\vartheta I + \beta \text{dev} \sigma_\vartheta$ we obtain
\[
\dot{\vartheta} k \text{dev} A = \left( \beta + \frac{\dot{\vartheta} k}{\nu} \left( 1 - \frac{\sigma_y}{|\text{dev} \sigma_\vartheta|} \right) \right) \text{dev} \sigma_\vartheta, \quad |\text{dev} \sigma_\vartheta| = \frac{\vartheta k (\nu |\text{dev} A| + \sigma_y)}{\dot{\vartheta} k + \beta \nu}. \tag{3.15}
\]
Inserting (3.15b) in (3.15a) shows
\[
\text{dev} \sigma_\vartheta = (C_2 + C_3/|\text{dev} \vartheta k A|) \text{dev} \vartheta k A. \tag{3.16}
\]
From (3.14) we obtain $\text{tr} \sigma_\vartheta = C_1 d \text{tr} \vartheta k A$. The expression for $\text{dev} \sigma_\vartheta$ in the elastic phase is obtained from (3.14) with $1 - \sigma_y / |\text{dev} \sigma_\vartheta| \leq 0$.

The plastic phase occurs for $(1 - \sigma_y / |\text{dev} \sigma_\vartheta|) > 0$. With (3.15b) this is equivalent to $|\text{dev} \vartheta k A| > \sigma_y / (2\mu)$. \qed

Theorem 3.2 (Viscoplasticity and plasticity with isotropic hardening). For viscoplasticity and plasticity with isotropic hardening there exist constants $C_1$, $C_2$, $C_3$, and $C_4$ such that the stress tensor reads
\[
\sigma = C_1 \text{tr}(\dot{\vartheta} k A) I + (C_3/(C_2 |\text{dev} \vartheta k A|) + C_4/C_2) \text{dev} \vartheta k A. \tag{3.17}
\]
For viscoplasticity with isotropic hardening these constants are

\[ \begin{align*}
C_1 &:= \lambda + 2\mu/d, & C_2 &:= \beta \nu (1 + H^2 \sigma^2) + \vartheta k (1 + \beta H^2 \sigma^2) \\
C_3 &:= \vartheta k \sigma_y (1 + \alpha_0 H), & C_4 &:= H_1 H^2 \vartheta k \sigma^2 + \nu (1 + H^2 \sigma^2)
\end{align*} \]

(3.18)

and in the limit \( \nu \to 0 \) for plasticity with isotropic hardening these constants read

\[ \begin{align*}
C_1 &:= \lambda + 2\mu/d, & C_2 &:= \vartheta k (1 + \beta H^2 \sigma^2) \\
C_3 &:= \vartheta k \sigma_y (1 + \alpha_0 H), & C_4 &:= H_1 H^2 \vartheta k \sigma^2.
\end{align*} \]

(3.19)

The plastic phase occurs for

\[ |\text{dev} \vartheta k \mathbf{A}| > \beta (1 + \alpha_0 H) \sigma_y. \]

(3.20)

In the elastic phase the stress tensor reads

\[ \sigma_\vartheta = C_1 \text{tr}(\vartheta k \mathbf{A}) \mathbf{I} + 2\mu \text{dev} \vartheta k \mathbf{A}. \]

(3.21)

**Proof.** The discretised version of the flow law is

\[ \mathbf{A} - \mathbb{C}^{-1} \frac{\partial \sigma_\vartheta}{\partial k} = \frac{1}{1 + H^2 \sigma^2} \left( 1 - \frac{(1 + \alpha_0 H) \sigma_y}{|\text{dev} \sigma_\vartheta|} \right) \text{dev} \sigma_\vartheta \]

(3.22)

\[ -H_1 \frac{(1 + \alpha_0 H) \sigma_y}{\vartheta k} = \frac{1}{\nu} \left( 1 - \frac{(1 + \alpha_0 H) \sigma_y}{|\text{dev} \sigma_\vartheta|} \right) H \sigma_y |\text{dev} \sigma_\vartheta|. \]

(3.23)

We consider the plastic phase \( 1 - (1 + \alpha_0 H) \sigma_y/|\text{dev} \sigma_\vartheta| > 0 \). With \( \mathbb{C}^{-1} \sigma_\vartheta = \alpha \text{tr} \sigma_\vartheta \mathbf{I} + \beta \text{dev} \sigma_\vartheta \) in (3.22) we infer

\[ \vartheta k \text{dev} \mathbf{A} = \left( \beta + \frac{\vartheta k}{\nu} \frac{1}{1 + H^2 \sigma^2} \left( 1 - \frac{(1 + \alpha_0 H) \sigma_y}{|\text{dev} \sigma_\vartheta|} \right) \right) \text{dev} \sigma_\vartheta \]

(3.24)

and so

\[ |\text{dev} \vartheta k \mathbf{A}| = \left( \beta + \frac{\vartheta k}{\nu} \frac{1}{1 + H^2 \sigma^2} \left( 1 - \frac{(1 + \alpha_0 H) \sigma_y}{|\text{dev} \sigma_\vartheta|} \right) \right) |\text{dev} \sigma_\vartheta|. \]

(3.25)

The second component of the flow rule, (3.23), can be solved for \( \alpha_0 \). Inserting of \( \alpha_0 \) in (3.25), solving for \( |\text{dev} \sigma_\vartheta| \) and inserting the resulting expression in (3.24) yields an expression for \( \text{dev} \sigma_\vartheta \). We obtain

\[ \text{dev} \sigma_\vartheta = \left( \frac{C_3}{C_2 |\text{dev} \vartheta k \mathbf{A}|} + \frac{C_4}{C_2} \right) \text{dev} \vartheta k \mathbf{A}. \]

(3.26)

From the first component of the flow rule we conclude \( \text{tr} \sigma_\vartheta = C_1 \text{tr} \vartheta k \mathbf{A} \). With \( |\text{dev} \sigma_\vartheta| \leq (1 + \alpha_0 H) \sigma_y \) the expression for \( \text{dev} \sigma_\vartheta \) in the elastic results from (3.22).

The plastic phase occurs for \( |\text{dev} \sigma_\vartheta| > (1 + \alpha_0 H) \sigma_y \). With (3.26) this is equivalent to \( 1 - \beta (1 + \alpha_0 H) \sigma_y/|\text{dev} \vartheta k \mathbf{A}| > 0 \).
Viscoplasticity and plasticity with kinematic hardening there exit constants \( C_1, C_2, \) and \( C_3 \) such that the stress tensor reads

\[
\sigma_\phi = C_1 \mathrm{tr}(\vartheta k \lambda) \mathbf{I} + (C_2 + C_3 / |\mathrm{dev}(\vartheta k \lambda - \beta \alpha_0)|) \mathrm{dev}(\vartheta k \lambda - \beta \alpha_0) + \mathrm{dev} \alpha_0.
\]  

(3.27)

For viscoplasticity with kinematic hardening these constants are

\[
C_1 := \lambda + 2 \mu / d, \quad C_2 := \frac{\vartheta k k_1 + 2 \nu}{\vartheta k + \vartheta k k_1 / (2 \mu) + \nu / \mu}, \quad C_3 := \frac{\vartheta k \sigma_y}{\vartheta k + \vartheta k k_1 / (2 \mu) + \nu / \mu},
\]  

(3.28)

and in the limit \( \nu \to 0 \) for plasticity with kinematic hardening these constants read

\[
C_1 := \lambda + 2 \mu / d, \quad C_2 := \frac{\vartheta k k_1}{\vartheta k + \vartheta k k_1 / (2 \mu)}, \quad C_3 := \frac{\vartheta k \sigma_y}{\vartheta k + \vartheta k k_1 / (2 \mu)}.
\]  

(3.29)

The plastic phase occurs for

\[
|\mathrm{dev}(\vartheta k \lambda - \beta \alpha_0)| > \beta \sigma_y.
\]  

(3.30)

In the elastic phase the stress tensor reads

\[
\sigma_\phi = C_1 \mathrm{tr}(\vartheta k \lambda) \mathbf{I} + 2 \mu \, \mathrm{dev} \vartheta k \lambda.
\]  

(3.31)

Proof. The discretised version of the flow law reads

\[
\mathbf{A} - C^{-1} \frac{\partial \sigma_\phi}{\partial \vartheta} = \frac{1}{2 \nu} \left(1 - \sigma_y / |\mathrm{dev}(\sigma_\phi - \alpha_\phi)|\right) \mathrm{dev}(\sigma_\phi - \alpha_\phi) = \frac{1}{k_1} \frac{\alpha_\phi - \alpha_0}{\vartheta k}.
\]  

(3.32)

We consider the plastic phase \( \nu \to 0 \), \( \sigma_y / |\mathrm{dev}(\sigma_\phi - \alpha_\phi)| > 0 \). With \( C^{-1} \sigma_\phi = \alpha \mathrm{tr} \sigma_\phi \mathbf{I} + \beta \mathrm{dev} \sigma_\phi \),

\[
\alpha_\phi = \alpha_0 + k_1 (\vartheta k \lambda - \alpha \mathrm{tr} \sigma_\phi \mathbf{I} - \beta \mathrm{dev} \sigma_\phi).
\]  

(3.33)

This yields in the flow rule that

\[
\mathrm{dev}(\vartheta k \lambda - \beta \sigma_\phi) = \frac{\vartheta k}{2 \nu} \left(1 - \frac{\sigma_y}{|\mathrm{dev}(\sigma_\phi - \alpha_0 - k_1 \vartheta k \lambda + k_1 \beta \sigma_\phi)|}\right) \times \mathrm{dev}(\sigma_\phi - \alpha_0 - k_1 \vartheta k \lambda + k_1 \beta \sigma_\phi).
\]  

The absolute value on both sides results in an equation for \( |\mathrm{dev}(\sigma_\phi - \alpha_0 + k_1 \vartheta k \lambda - k_1 \beta \sigma_\phi)| \). We solve for this modulus and substitute it in the above identity. This shows

\[
\mathrm{dev}(\vartheta k \lambda - \beta \sigma_\phi) = \frac{\vartheta k}{2 \nu} \left(1 - \frac{\sigma_y}{2 \nu / (\vartheta k)}|\mathrm{dev}(\vartheta k \lambda - \beta \sigma_\phi)| + \sigma_y\right) \times \mathrm{dev}(\sigma_\phi - \alpha_0 - k_1 \vartheta k \lambda + k_1 \beta \sigma_\phi).
\]  

(3.34)
Multiplying both sides of (3.34) with \((2v)/(\partial k)\)|dev(\(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta\))| + \(\sigma_y\) together with some basic transformations shows

\[
\left(\frac{2v}{\partial k} + k_1\right)|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta)| + \sigma_y
\]

\[\times \text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta)| = |\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta)|\text{dev}|\mathbf{\sigma}_\theta - \mathbf{\alpha}_0|.
\]

(3.35)

On both sides of (3.35) we build the absolute value and obtain

\[
|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta)| = \frac{|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| - \sigma_y}{k_1 + 2v/\partial k}
\]

which can be inserted in (3.35),

\[
|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\sigma}_\theta) = \frac{|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| - \sigma_y}{k_1 + 2v/\partial k}\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0).
\]

(3.37)

Adding the term |\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)|\(\mathbf{\beta}\text{dev}\mathbf{\alpha}_0\) on both sides of (3.37) yields with some basic transformations

\[
|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\alpha}_0) = \left(\mathbf{\beta}\right|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)\right) + \frac{|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| - \sigma_y}{k_1 + 2v/\partial k}
\]

\[\times \text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0).
\]

(3.38)

Because of \(2v/\partial k + k_1 > 0\) and with |\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| - \(\sigma_y\) > 0 from (3.36) we can calculate the absolute value of \text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0) as

\[
|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| = \frac{|\partial k + 2v/\partial k|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\alpha}_0)| + \partial k\sigma_y}{\partial k + \mathbf{\beta}\partial k + 2\mathbf{\beta} v}
\]

(3.39)

We insert (3.39) in (3.38) to obtain

\[
\text{dev}\mathbf{\sigma}_\theta = (C_2 + C_3/|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\alpha}_0)|)\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\alpha}_0) + \text{dev}\mathbf{\alpha}_0.
\]

(3.40)

From the first component of the flow rule we have \(\text{tr}\mathbf{\sigma}_\theta = C_1\text{tr}(\partial k\mathbf{A})\). The expression for \text{dev}\mathbf{\sigma}_\theta in the elastic phase is obtained from (3.32) with \(1 - \sigma_y/|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)|\leq 0\).

The plastic phase occurs for \(1 - \sigma_y/|\text{dev}(\mathbf{\sigma}_\theta - \mathbf{\alpha}_0)| > 0\). With (3.36), (3.37) and (3.39) this is equivalent to \(|\text{dev}(\partial k\mathbf{A} - \mathbf{\beta}\mathbf{\alpha}_0)| > \mathbf{\beta}\sigma_y\).

\[\Box\]

4. NUMERICAL SOLUTION OF THE TIME-DISCRETE PROBLEM

This section is devoted to the framework of the spatial discretisation of one timestep (3.1) with data structures in Subsection 4.1 and the iteration in Subsection 4.2. The heart of the matter is the assembling of the tangential local stiffness matrix explained in Section 5.
4.1. Discretisation in space

Suppose the domain $\Omega$ has a polygonal boundary $\Gamma$, we can cover $\bar{\Omega}$ by a regular triangulation $\mathcal{T}$, i.e., $\bar{\Omega} = \bigcup_{T \in \mathcal{T}} T$, where the elements of $\mathcal{T}$ are triangles for $d = 2$ and tetrahedrons for $d = 3$. Regular triangulation in the sense of Ciarlet [10] means that the nodes $\mathcal{N}$ of the mesh lie on the vertices of the elements, the elements of the triangulation do not overlap, no node lies on an edge of an element, and each edge $E \subset \Gamma$ of an element $T \in \mathcal{T}$ belongs either to $\Gamma_N$ or to $\Gamma_D$.

Matlab supports reading data from files given in ASCII format by *.dat files. Figure 1 shows the initial mesh of a two dimensional example and Figure 2 shows the corresponding data files. The file coordinates.dat contains the coordinates of each node. The two real numbers per row are the $x$- and $y$-coordinates of each node. The file elements.dat contains for each element the node numbers of the vertices, numbered anti-clockwise.

The files neumann.dat and dirichlet.dat contain the two node numbers which bound the corresponding edge on the boundary.

In the discrete version of (2.14), $H^1(\Omega)$ and $H^1_D(\Omega)$ are replaced by finite dimensional subspaces $\mathcal{S}$ and $\mathcal{S}_D = \{ V \in \mathcal{S} : V = 0 \text{ on } \Gamma_D \}$. The discrete problem
### Data files

<table>
<thead>
<tr>
<th>coordinates.dat</th>
<th>elements.dat</th>
<th>dirichlet.dat</th>
<th>neumann.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2.0000</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2.0000</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2.0000</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1.0000</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0000</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8660</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3989</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3748</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0241</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.** Data files `coordinates.dat`, `elements.dat`, `dirichlet.dat`, and `neumann.dat` for the initial triangulation in the example of Subsection 7.1. The files `u_b.m`, `f.m`, and `g.m` are listed in Section 7.

### 4.2. Iterative solution

This subsection describes the iterative solution of (4.2) by a Newton–Raphson scheme which is realised in the Matlab program `fem.m`, listed at the end of this section.
Elastoviscoplastic FE analysis

One step of the Newton iteration reads

\[ DF(U_k^\theta)U_k^{\theta+1} = DF(U_k^\theta)U_k^{\theta} - F(U_k^\theta). \tag{4.5} \]

The discrete displacement vector \( U_k^\theta \) is expressed in the nodal basis as

\[ U_k^\theta = \sum_{i=1}^{N} U_{\theta,i}(\Phi_i, (U_{\theta,1}, \ldots, U_{\theta,dn})) \]

are the components of \( U_k^\theta \). The matrix \( DF \) is defined as

\[ (DF(U_k^{\theta,1}, \ldots, U_k^{\theta,dn}))_{pq} := \partial F_p(U_k^{\theta,1}, \ldots, U_k^{\theta,dn})/\partial U_k^{\theta,q}. \tag{4.6} \]

At the end of this section the Matlab program \texttt{fem.m}, a finite element program for the two dimensional case is listed. In the following line 1

1 function ut=FEM(coordinates,elements,dirichlet,neumann,ut,u0,t0,t1,th,N)

shows the function call with input coordinates, elements, dirichlet and neumann together with the vectors ut and u0, which are the start-vector \( u_0 \) of the Newton iteration and the displacement \( u_0 \) at time \( t_0 \). The last three variables in the parameter list are the times \( t_0, t_1 \), and the variable \( \theta \). Lines 2–7,

\%Initialisation
2 DF=sparse(2*N,2*N);q=zeros(2*N,1);P=zeros(2*N,1);
3 list=2*elements(:,[1,1,2,2,3,3])-ones(size(elements,1),1)*[1,0,1,0,0,0];
\%Assembly
4 for j=1:size(elements,1)
5 \[ \text{DF}(L,L)(j,:)=stima(\text{DF}(L,L),\text{U}(L),\text{coordinates}(\text{elements}(j,:)),\text{ut}(L),\text{u0}(L),j); \]
7 end

show the assembling of the stiffness matrix \( DF \) and the stiffness vector \( Q \). Line 2 initialises \( DF, Q, \) and \( P \), line 3 defines a vector \text{list} which contains the numbers of degrees of freedom for each element. Lines 4–7 perform the assembling of \( DF \) and \( Q \) with a call of subroutine \text{stima.m}. The element with number \( j \) contributes the degrees of freedom \( \text{L} = \text{list}(j,:) \). The local stiffness matrix and the local stiffness vector of element number \( j \) have to be added at the positions \( (L,L) \) of the global stiffness matrix and \( (L) \) of the global stiffness vector.

In the following lines 8–22

\%Volume forces
8 for j=1:size(elements,1)
9 \[ \text{coordinates}=coordinates(\text{elements}(j,:));\text{area}=\text{det}([1,1,1;\text{coordinates}])/2; \]
10 \[ \text{b}=\text{area}^2*(1-th)*f(\text{sum}(coordinates,1)/3,t0)/9\text{th}+f(\text{sum}(coordinates,1)/3,t1)/3; \]
11 P(list(j,:))=P(list(j,:))+repmat(b,3,1);
12 end
\%Neumann conditions
13 if ~isempty(neumann)
14 \[ \text{Wlist}=\text{zeros}([],[1,1,2,2]);\text{one}(\text{size}(\text{neumann}(:,1),1))*\text{repmat([1,0,1,2],1,2);} \]
15 \[ \text{EV}=\text{coordinates}(\text{neumann}(::2,:))-\text{coordinates}(\text{neumann}(::1,:)); \]
16 \[ \text{Lg}=\text{sqrt}((\text{EV}+\text{EV},2));\text{EV}+	ext{EV}./[\text{Lg},\text{Lg}];\text{NV}=(\text{EV}+[0,1,1,0]); \]
17 for j=1:size(neumann,1)
18 \[ \text{gs}=(1-th)*g(\text{sum}(coordinates(\text{neumann}(j,:,:),1))/2,\text{NV}(j,:),t0)/2 \]
19 \[ +\text{th}*(\text{sum}(coordinates(\text{neumann}(j,:,:),1))/2,\text{NV}(j,:),t1)/2); \]
20 P(Wlist(j,:))=P(Wlist(j,:))+repmat(gs,2,1);
21 end

Bereitgestellt von | Humboldt-Universität zu Berlin
Angemeldet | 141.20.50.190
Heruntergeladen am | 25.07.13 14:29
the vector \( F = Q - P \) is determined. \( P \) contains the work of the volume forces \( f \) and the surface forces \( g \) at time \( t_0 \). The values of \( f \) are provided by the subroutine \( f, m \) and evaluated in the centre of gravity \((x_S, y_S)\) of \( T \), to approximate the integral
\[
\int_T f \cdot \varphi_j \, dx \approx \frac{1}{3} |T|f_k(x_S, y_S), \quad k := \mod(r - 1, 2) + 1. \tag{4.7}
\]
The integral \( \int_E g \cdot \varphi_k \, ds \) in (4.2) that involves the Neumann conditions is approximated with the value of \( g \) in the centre \((x_M, y_M)\) of the edge \( E \) with length \( |E| \),
\[
\int_E g \cdot \varphi_k \, ds \approx \frac{1}{2} |E|g_k(x_M, y_M), \quad k := \mod(r - 1, 2) + 1. \tag{4.8}
\]

The following lines 24–33

\begin{verbatim}
22 end  
23 F=F-P;

\end{verbatim}

take into account Dirichlet conditions whereas gliding boundary conditions are allowed as well. Gliding boundary conditions such as those along symmetry axes, are implemented different from Dirichlet conditions (for all components). The displacement is fixed merely in one specified direction and possibly free in others. A general approach to this type of condition reads, for each node on the Dirichlet boundary,
\[
\begin{pmatrix} m_1 \\ \\ \vdots \\ m_d \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_d \end{pmatrix} = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} \tag{4.9}
\]

where \( m_j \in \mathbb{R}^{1 \times d} \), \( w_j \in \mathbb{R} \), and \((U_1, \ldots, U_d)\) are the degrees of freedom for the considered node on the Dirichlet boundary. With the normal vector \( n \) along \( \Gamma \) and the canonical basis \( e_j \), \( j = 1, \ldots, d \), in \( \mathbb{R}^d \), gliding and classical Dirichlet conditions and the lack of Dirichlet-type conditions on certain parts of \( \Gamma \) can all be included by
\[
\begin{pmatrix} n \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_d \end{pmatrix} = 0, \quad \begin{pmatrix} e_f^1 \\ \vdots \\ e_f^d \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_d \end{pmatrix} = u_d, \quad \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_d \end{pmatrix} = 0 \tag{4.10}
\]
respectively. Line 24 determines all nodes on the Dirichlet boundary and stores them in the variable `nodes`. Let there be \( n \) Dirichlet nodes. Line 26 creates a matrix \( H \) of size \( 2n \times 2 \) and a vector \( w \) of size \( 2n \times 1 \). The rows \( 2(j-1)+1 \) up to \( 2j \) for \( j = 1, \ldots, n \) of \( H \) and \( w \) contain for each Dirichlet node the matrix \( (m_1, \ldots, m_d)^T \) and the vector \((w_1, w_2)\). For the discrete problem, the generalised Dirichlet conditions lead to the linear system

\[
BU = w
\]

where each row of \( B \in \mathbb{R}^{2n \times 2N} \) contains the value of some \( m_j \) at a node on the boundary and \( w \in \mathbb{R}^{2n} \) contains the corresponding values \( w_j \) at this node. The matrix \( B \) is calculated in lines 27–31 from the entries of \( C \). A row \( m_j \) is not included in \( B \) and \( W \), which is realised in line 33.

The Newton iteration takes place. Coupling the set of boundary conditions through Lagrange parameters with \((4.5)\) leads to the extended system

\[
\begin{pmatrix}
DF(U_0^{k+1}) & B^T \\
B & 0
\end{pmatrix}
\begin{pmatrix}
U_0^{k+1} \\
\lambda
\end{pmatrix}
= \begin{pmatrix}
b \\
w
\end{pmatrix}
\]

with

\[
b := DF(U_0^k)U_0^k - f(U_0^k). \tag{4.13}
\]

This system of equations is solved in line 37 by the Matlab solver. Matlab makes use of the properties of a sparse, symmetric matrix for solving the system of equations efficiently. Line 37 determines the parts \( U_0^{k+1} \) and \( \lambda \) of the solution. Line 38 to 43 assemble the local stiffness matrix and stiffness vector. Line 32 calculates the set \( \mathcal{K} \) of free nodes. There are \( m \) free nodes \((z_1, \ldots, z_m) \in \mathcal{K}\). The iteration process only continues in line 34, if the relative residual \( F(U_0^k, z_1, \ldots, U_0^k, 0) \) is greater than a given tolerance and if the maximum number of iterations is less than 100.

4.3. 2D Finite Element program

```
function ut=FEM(coordinates,elements,dirichlet,newmann,ut,uo,t0,t1,th,N)
%initialisation
```

Bereitgestellt von | Humboldt-Universität zu Berlin
Angemeldet | 141.20.50.190
Heruntergeladen am | 25.07.13 14:29
4.3 Finite Element program

1 function ut = FEINS_d(coordinates, elements, dirichlet, neumann, ut, u0, t0, t1, N, MJ)
2 \%initialisation
3 D = sparse(3*N, 3*N); \$zeros(3*N, 1); \$zeros(3*N, 1);
4 list = elements([1, 1, 2, 2, 3, 3, 4, 4]) - ones(NJ, 1); ut(NJ, 1) = [2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0]; \%Assembly
5 for j = 1:size(elements, 1)
6 L = list(j, :);
7 [DF(L), \$L(L)] = stima(DF(L), \$L(L), coordinates(elements(j, :), ut(L), u0(L), j));
8 end
9 \%Neumann conditions
10 if isempty(neumann)
11 Nlist = 2*neumann(:, [1, 1, 2, 2]) - ones(size(neumann, 1), 1); neumann(:, 1, 2) = 0.0;
12 EV = cooordinates(neumann(:, 2)); \%coordinates(neumann(:, 1));
13 Lg = sqrt(sum(EV, \%EV)); EV = [Lg, Lg]; \%EV = [0, 1, 0];
14 for j = 1:size(neumann, 1)
15 g = g(j); ((1-th) + (sum(cordinate, 1) / S, t0) / S) + (sum(cordinate(1, 1), t0) / S);
16 P(list(j, :)) = P(list(j, :)) + repmat(g, 1, 1, 1);
17 end
18 end
19 end
20\%Dirichlet conditions
21 dnodes = unique(dirichlet);
22 [W0, W1] = u_D(dnodes, .), t0); \%W1 M1 = u_D(dnodes, .), t1);
23 W = (1-th) + W0 + W1; \%M = (1-th) + M0 + M1; B = sparse(size(W, 1), 2*N);
24 for k = 0:1
25 for j = 1:W
26 B(1:j + 2, size(M, 1), 2*dnodes - 1 + k) = diag(M(1:j + 2, size(M, 1), 1 + k));
27 end
28 end
29 mask = find(sum(abs(B)));
30 freeNodes = find(sum(abs(B));)
31 B = B(mask, :); W = W(mask, :); normO = norm(P(freeNodes)); nit = 0;
32 while ((norm(P(freeNodes)) > 10^(-10) + 10^(-6)) \& nit < 100)
33 nit = nit + 1
34 \%Assembly
35 D = sparse(3*N, 3*N); \$zeros(3*N, 1);
36 for j = 1:size(elements, 1)
37 L = list(j, :);
38 [DF(L), \$L(L)] = stima(DF(L), \$L(L), coordinates(elements(j, :), ut(L), u0(L), j));
39 end
40 end
41 end
42 P = q-f;
43 end
44 if nit > 100 disp('Fails to converge within 100 iteration steps'); end

4.4. 3D Finite Element program

1 function ut = FEINS_d(coordinates, elements, dirichlet, neumann, ut, u0, t0, t1, N, MJ)
2 \%initialisation
3 D = sparse(3*N, 3*N); \$zeros(3*N, 1); \$zeros(3*N, 1);
4 list = elements([1, 1, 2, 2, 3, 3, 4, 4]) - ones(NJ, 1); ut(NJ, 1) = [2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0]; \%Assembly
5 for j = 1:size(elements, 1)
6 L = list(j, :);
7 [DF(L), \$L(L)] = stima(DF(L), \$L(L), coordinates(elements(j, :), ut(L), u0(L), j));
8 end
\%Volume forces
9 for j=1:size(elements,1)
10   lcoordinates=coordinates(elements(j,:,:));vol=det([l,1,1,1;coordinates'])/6;
11   b=vol*(1-th)+f(sum(lcoordinates,1)/3,t0)/4+th*f(sum(lcoordinates,1)/3,t1)/4;
12   F(list(j,:))=F(list(j,:))+repmat(b,4,1);
13 end
\%Neumann conditions
14 if isempty(neumann)
15   Nlist=neumann(:,[1,1,2,2,2,3,3,3],ones(size(neumann,1),1))*repmat([2,1,0],1,3);
16 for j=1:size(neumann,1)
17   NV=coordinates(neumann(j,:,:))
18   coordinates(neumann(j,:,:),1,:),...
19   coordinates(neumann(j,:,:),1,:)));
20   area=norm(NV(2,:))/2;NV=NV/norm(NV);
21   ge=area*((1-th)*g(sum(coordinates(neumann(j,:,:)))/2,NV,t0)/2...
22   +th*g(sum(coordinates(neumann(j,:,:)))/2,NV,t1)/2);
23   F(Nlist(j,:))=F(Nlist(j,:))+repmat(ge,3,1);
24 end
25 F=P-Q-P;
26 \%Dirichlet conditions
27 dnodes=unique(dircilhet);
28 [x0,M]=u.D(coordinates(dnodes,:),t0);[x1,M]=u.D(coordinates(dnodes,:),t1);
29 for k=0:2
30 for j=0:2
31 B(1+j,3*size(M,1),3*dnodes+2*k)=diag([1+j,3:size(M,1),1+k]);
32 end
33 end
34 mask=find(sum(abs(B'),1));freeNodes=find(sum(abs(B)));)
35 B=B(mask,:);W=W(mask,:);normO=norm(F(freeNodes));nit=0;
36 while (norm(P(F(freeNodes))))>10^(-10)+10^(-6)*normO & (nit<100)
37 nit=nit+1
38 \% Calculating the solution
39 DF=DF(ut-F,W);DF=DB,B,sparse(size(B,1),size(B,1));
40 \% Assembly
41 for j=1:size(elements,1)
42 L=list(j,:);
43 DF(L,L),U(L)=stims(DF(L,L),U(L),coordinates(elements(j,:,:),ut(L),ud(L),j);
44 end
45 F=P-Q-P;
46 end
47 if (nit=100) disp('Fails to converge within 100 iteration steps!'); end

5. TANGENTIAL LINEAR SYSTEM OF EQUATIONS

The tangential stiffness matrix $DF$ in (4.6) as well as the vector $F$ in (4.2) are expressed as a sum over all elements $T$ in $\mathcal{T}$. Each part of the sum defines the local stiffness matrix $M$ and the vector of the right-hand side $R$ for the corresponding element. Let $k_1$ through $k_K$ be the numbers of the nodes of an element $T$. There are $dK$ basis functions with support on $T$, namely, $\phi_{\tau,T,l} = \phi_{k_1}e_1, \ldots, \phi_{\tau,T,d} = \phi_{k_d}e_d$, where $e_j$ is the $j$-th unit vector. The function $\phi_{k_j}$ is the local scalar hat function of node $k_j$. The function $\pi$ maps the indices $1, \ldots, dK$ of the local numeration with respect to $T$ to the global numeration. In the sequel we want to compute the local stiffness matrix $M$ and a local vector $R$ for element $T$. For simplicity we write $\phi_1, \ldots, \phi_{dK}$ instead of $\phi_{\tau,T,l}, \ldots, \phi_{\tau,T,dK}$.
The local vector $R$ is defined as
\[
R_r := \int_r \sigma (\epsilon (U_0 - U_0) + \mathbb{C}^{-1} \sigma_0) : \epsilon(\varphi_r) \, dx, \quad r = 1, \ldots, dK.
\]
(5.1)

The local stiffness matrix for element $T$ is defined, for $r,s = 1, \ldots, dK$, by
\[
M_{rs} = \frac{\partial R_r}{\partial U^k_{s,r}} = \frac{1}{\partial U^k_{s,r}} \left( \int_T \sigma \left( \sum_{r=1}^K U_{s,r} \epsilon(\varphi_r - U_0) + \mathbb{C}^{-1} \sigma_0 \right) : \epsilon(\varphi_r) \, dx \right).
\]
(5.2)

### 5.1. Notation for triangular elements

The Matlab implementation of $M$ and $R$ can be done in a compact way by defining a matrix epsilon by
\[
epsilon = \left[ \epsilon_{11}(\varphi) \epsilon_{12}(\varphi) \epsilon_{13}(\varphi) \epsilon_{14}(\varphi) \right]_{j=1}^6 = \frac{1}{2} (\text{Det}1 + \text{Det}2)
\]
\[
= \frac{1}{2} \left[ (\text{Det}1_{11}, \text{Det}1_{12}, \text{Det}1_{13}, \text{Det}1_{14}) + (\text{Det}2_{11}, \text{Det}2_{12}, \text{Det}2_{13}, \text{Det}2_{14}) \right]_{j=1}^6
\]
(5.3)

with
\[
\text{Det}1 = \begin{bmatrix}
\frac{\partial \varphi_1}{\partial x} & \frac{\partial \varphi_1}{\partial y} & 0 & 0 \\
0 & \frac{\partial \varphi_2}{\partial x} & \frac{\partial \varphi_2}{\partial y} & 0 \\
0 & 0 & \frac{\partial \varphi_3}{\partial x} & \frac{\partial \varphi_3}{\partial y} \\
0 & 0 & 0 & \frac{\partial \varphi_4}{\partial x} \\
\end{bmatrix}, \quad \text{Det}2 = \begin{bmatrix}
0 & \frac{\partial \varphi_1}{\partial y} & \frac{\partial \varphi_2}{\partial y} & \frac{\partial \varphi_3}{\partial y} & \frac{\partial \varphi_4}{\partial y} \\
\frac{\partial \varphi_1}{\partial x} & 0 & \frac{\partial \varphi_2}{\partial x} & \frac{\partial \varphi_3}{\partial x} & \frac{\partial \varphi_4}{\partial x} \\
0 & \frac{\partial \varphi_1}{\partial y} & \frac{\partial \varphi_2}{\partial y} & \frac{\partial \varphi_3}{\partial y} & \frac{\partial \varphi_4}{\partial y} \\
0 & 0 & \frac{\partial \varphi_1}{\partial x} & \frac{\partial \varphi_2}{\partial x} & \frac{\partial \varphi_3}{\partial x} \\
\end{bmatrix}.
\]
(5.4)

With the coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ of the vertices of the element $T$, the entries of Det1 and Det2 are stored in the matrix Dphi,
\[
\text{Dphi} = \begin{bmatrix}
\frac{\partial \varphi_1}{\partial x} & \frac{\partial \varphi_1}{\partial y} & 0 \\
0 & \frac{\partial \varphi_2}{\partial x} & \frac{\partial \varphi_2}{\partial y} \\
\frac{\partial \varphi_3}{\partial x} & \frac{\partial \varphi_3}{\partial y} & 0 \\
\frac{\partial \varphi_4}{\partial x} & \frac{\partial \varphi_4}{\partial y} & 0 \\
\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix}x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix}y_1 & y_2 & y_3 \end{bmatrix}^{-1} \begin{bmatrix}0 & 0 \\
1 & 0 \\
0 & 1 \\
\end{bmatrix}.
\]
(5.5)

With Dphi from (5.5) the following Matlab lines calculate the matrices Det1 and Det2:

```
Det1=zeros(6,4); Det1(1:2:5,1:2)=Dphi; Det1(2:4,1:2)=Dphi;
Det2=zeros(6,4); Det2(1:2:5,1:3)=Dphi; Det2(2:4,1:3)=Dphi;
```
5.2. Notation for tetrahedral elements

In the three-dimensional case the matrix \( \epsilon \) is defined by

\[
\epsilon := [e_{11}(\phi_j) e_{12}(\phi_j) e_{13}(\phi_j) e_{21}(\phi_j) e_{22}(\phi_j) e_{23}(\phi_j) e_{31}(\phi_j) e_{32}(\phi_j) e_{33}(\phi_j)]_{j=1}^{12}
\]

\[
= \frac{1}{2} (\text{Det}1 + \text{Det}2)
\]  

with

\[
\text{Det}1 = [(D\phi_j)_{11}, (D\phi_j)_{12}, (D\phi_j)_{13}, (D\phi_j)_{21}, (D\phi_j)_{22}, (D\phi_j)_{23}, (D\phi_j)_{31}, (D\phi_j)_{32}, (D\phi_j)_{33}]_{j=1}^{12}
\]

\[
= \begin{bmatrix}
\frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} \\
\end{bmatrix}
\]  

(5.7)

and

\[
\text{Det}2 = [(D\phi_j)_{11}, (D\phi_j)_{12}, (D\phi_j)_{13}, (D\phi_j)_{21}, (D\phi_j)_{22}, (D\phi_j)_{23}, (D\phi_j)_{31}, (D\phi_j)_{32}, (D\phi_j)_{33}]_{j=1}^{12}
\]

\[
= \begin{bmatrix}
\frac{\partial \phi_i}{\partial x} & 0 & 0 & \frac{\partial \phi_i}{\partial x} & 0 & 0 & \frac{\partial \phi_i}{\partial x} & 0 \\
0 & \frac{\partial \phi_i}{\partial y} & 0 & 0 & \frac{\partial \phi_i}{\partial y} & 0 & 0 & \frac{\partial \phi_i}{\partial y} \\
0 & 0 & \frac{\partial \phi_i}{\partial z} & 0 & 0 & \frac{\partial \phi_i}{\partial z} & 0 & \frac{\partial \phi_i}{\partial z} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \frac{\partial \phi_i}{\partial y} & 0 & 0 & \frac{\partial \phi_i}{\partial y} & 0 & 0 & \frac{\partial \phi_i}{\partial y} \\
0 & 0 & \frac{\partial \phi_i}{\partial z} & 0 & 0 & \frac{\partial \phi_i}{\partial z} & 0 & \frac{\partial \phi_i}{\partial z} \\
\end{bmatrix}
\]  

(5.8)

The entries of \( \text{Det}1 \) and \( \text{Det}2 \) are stored in the matrix \( \text{Dphi} \),

\[
\text{Dphi} = \begin{bmatrix}
\frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} \\
\frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} \\
\frac{\partial \phi_i}{\partial x} & \frac{\partial \phi_i}{\partial y} & \frac{\partial \phi_i}{\partial z} \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 & 1 & 1 & 1 \\
x_1 & x_2 & x_3 & x_4 \\
y_1 & y_2 & y_3 & y_4 \\
z_1 & z_2 & z_3 & z_4 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}^{-1}
\]

(5.9)

where \( (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4) \) are the coordinates of the vertices of the element \( T \). The following Matlab lines calculate the matrices \( \text{Det}1 \) and \( \text{Det}2 \).
5.3. Local stiffness matrix $M$ and local vector $R$

The subsequent algorithms perform a full differentiation between the elastic and the plastic phase. In the plastic phase the local stiffness matrix $M$ and the local vector $R$ for element $T$ are obtained by evaluating (5.2) and (5.1) for the different cases of hardening. In the elastic phase $M$ and $R$ are well known, c.f. [3]. The Matlab routine stima.m calculates the local stiffness matrix $M$ according to (5.2) and the first summand of (5.1). With $M$ and $R$ a step of the assembling procedure is performed in stima.m.

Example 5.1 (Perfect viscoplasticity for 2D and 3D). The Matlab program calculating the local stiffness matrix and the vector $R$ depending on the element $T$ is listed at the end of this example for the 2D and 3D case, together with a special implementation of the deviator and trace function, called dev2 and tr2 in the 2D case and dev3 and tr3 in the 3D case. The 2D program is described below. In line 2 global variables are transferred to the routine,

$C_1 := \lambda + 2\mu / d, \quad C_2 := v / (\beta v + \vartheta k), \quad C_3 := \vartheta k \sigma_3 / (\beta v + \vartheta k), \quad C_4 := 2\mu$.

In lines 4–7 the matrix epsilon is calculated as described in the previous section. In line 7 we define the matrix $v := \varepsilon (U_0 - U_0) + C^{-1} \sigma_0$ and store it row wise. In line 8 the area of element $T$ is determined. In line 9 a fall differentiation is performed. If $|\text{dev}(v)| - \sigma_j / (2\mu) > 0$ the plastic phase occurs, otherwise the elastic phase. Depending on the result of this distinction, a constant $C_5$ and a vector $C_6$ are defined by

$$C_5 := \begin{cases} C_2 + C_3 / |\text{dev}(v)| & \text{if } |\text{dev}(v)| - \sigma_j / (2\mu) > 0 \\ 2\mu & \text{else} \end{cases} \quad (5.10)$$

$$C_6 := \begin{cases} C_3 / |\text{dev}(v)|^3 & \text{if } |\text{dev}(v)| - \sigma_j / (2\mu) > 0 \\ 0 & \text{else} \end{cases} \quad (5.11)$$

In lines 14 and 15 an update of the local stiffness matrix and the first summand of the vector $R$ is performed. The components of the local stiffness matrix read

$$M_{jk} = |T| (C_1 \text{tr}(\varepsilon(\varphi_j)) \text{tr}(\varepsilon(\varphi_k)) + C_5 \text{dev}(\varepsilon(\varphi_j)) : \varepsilon(\varphi_k) - \langle C_6 \rangle \text{dev}(v) : \varepsilon(\varphi_k)).$$

The components of the first summand of the local vector $R$ read

$$R_j = |T| (C_1 \text{tr}(v) \text{tr}(\varepsilon(\varphi_j)) + C_5 \text{dev}(v) : \varepsilon(\varphi_j)). \quad (5.13)$$

The Matlab routines stima.m, tr2.m and dev2.m for the 2D case are listed below.
The Matlab routines \texttt{stima.m}, \texttt{tr3.m} and \texttt{dev3.m} for the 3D case are listed below.

\begin{verbatim}
1 function [M,F]=stima(N0,FO,loccoordinates,ut,u0,j);
2 global lambda mu sigma_y C1 C2 C3 e0
3 Dphi=inv([1,1,1;loccoordinates'])*[0,0,eye(2)];
4 eps=epsilon(1,4,9);eps1=epsilon12(1,4,9);eps12=epsilon12(2,4,9);
5 Det11=Det22=Det33=det(epsilon13);eps13=epsilon13(3,4,9);
6 eps14=epsilon14(4,4,9);eps24=epsilon24(4,4,9);eps34=epsilon34(4,4,9);
7 eps15=epsilon15(5,4,9);eps25=epsilon25(5,4,9);eps35=epsilon35(5,4,9);
8 Det12=Det13=Det23=det(epsilon(1,2,3));eps123=epsilon123(1,2,3,9);
9 if norm(dev2(v))>sigma_y(2*e0)/2;
10 C0=C2+C3norm(dev2(v))/6;C1=C3norm(dev2(v))^3*(dev2(e0)+dev2(v))';
11 else
12 C5=2*e0;eps=epsilon(6,1,1);
13 end
14 M=FO*T(C1+2*eps)+eps2*det(epsilon13)+eps3*det(epsilon123);
15 F=FO*T(C1+2*eps)+eps2*det(epsilon13)+eps3*det(epsilon123);

function tr2=tr2(A);
  tr2=A(:,1)+A(:,4);
function dev2=dev2(A);
  dev2=[A(:,1)-A(:,4)]/2,A(:,2),A(:,3),(A(:,4)-A(:,1))/2];
\end{verbatim}

Example 5.2 (Isotropic hardening in 2D). The Matlab program calculating the local stiffness matrix and the local vector $R$ is listed at the end of this example. In line 2 global variables $C_1$ up to $C_6$ are transfered to the routine,

\begin{align*}
C_1 &:= \lambda + \mu, \quad C_2 := \frac{\nu}{2\mu}(1 + H^2 \sigma_y^2) + \frac{1}{2\mu}H_1H^2\sigma_y^2, \\
C_3 &:= \partial k \sigma_y (1 + \alpha_0 H), \quad C_4 := H_1H^2 \partial k \sigma_y^2 + \nu(1 + H^2 \sigma_y^2), \\
C_5 &:= 2\mu, \quad C_6 := \frac{1}{2\mu}(1 + \alpha_0 H)\sigma_y.
\end{align*}

Lines 3–8 are analog to the last example and perform the calculation of \texttt{epsilon},
define the vector $v$ and determine the element area $T$. In line 9 the fall differentiation is performed. If $|\text{dev}(v)| - C_6 > 0$ the plastic phase occurs, otherwise the elastic phase. Depending on the result of this distinction, constants $C_7$ and a vector $C_8$ are defined by

$$C_7 := \begin{cases} C_3/(C_2|\text{dev}(v)|) + C_4/C_2 & \text{if } |\text{dev}(v)| - C_6 > 0 \\ C_5 & \text{else} \end{cases}$$ (5.14)

$$(C_8)_j := \begin{cases} C_3/(C_2|\text{dev}(v)|^3)\text{dev}2(\epsilon(\varphi_j)) : \text{dev}2(v) & \text{if } |\text{dev}(v)| - C_6 > 0 \\ [0 \ 0 \ 0 \ 0 \ 0]^T & \text{else}. \end{cases}$$ (5.15)

In lines 14 and 15 an update of the local stiffness matrix and the first summand of the local vector $R$ is performed. The components of the local stiffness matrix read

$$M_{jk} = |T|(C_1 \text{tr}(\epsilon(\varphi_j)) \text{tr}(\epsilon(\varphi_k)) + C_7 \text{dev}(\epsilon(\varphi_j)) : \epsilon(\varphi_k) - (C_8)_j \text{dev}(v) : \epsilon(\varphi_k)).$$ (5.16)

The components of the first summand of the vector $R$ read

$$R_j = |T|(C_1 \text{tr}(\epsilon(\varphi_j)) + C_7 \text{dev}(v) : \epsilon(\varphi_j)).$$ (5.17)

The Matlab routine stima.m is listed below.

```matlab
1 function [N,r]=stima(N0,R0,locationcoordinates,u0,j);
2 global C1 C2 C3 C4 C5 C6 C7 C8
3 Dphi=inv([1,1,1;locationcoordinates])*[0,0,eye(2)];
4 eps=zeros(6,4);eps2=zeros(6,4);
5 Det1=t1/2;Det2=t2/6;Det=Det1*Det2;
6 eps=eps(1:4,1:2)+Dphi;eps2=eps(1:4,1:2)+Dphi;
7 eps=eps(1:4,1:2);eps2=eps2(1:4,1:2);
8 T=det([1,1,1;locationcoordinates])^2/2;
9 if norm(eps2(v))<0.001
10 C7=C7(j)/(C2+norm(eps2(v)));C8=C8(j)/(C2+norm(eps2(v))+3)*dev2(eps)*dev2(v);
11 else
12 C7=C7;C8=zeros(6,1);
13 end
14 N=0.001*T*(C1*tr2(eps)+tr2(eps)^2)+C7*dev2(eps)+C7*dev2(v);
15 R=0.001*(C1+tr2(eps)+tr2(eps)^2)+C7*eps+dev2(v);
```

**Example 5.3 (Kinematic hardening in 2D).** The Matlab program calculating the local stiffness matrix and the local vector $R$ is listed at the end of this example. In line 2 global variables $C_1$ up to $C_4$ are transfered to the routine,

$$C_1 := \lambda + \mu, \quad C_2 := \frac{\partial k k_1 + 2\nu}{\partial k + \partial k k_1/(2\mu) + \nu/\mu}, \quad C_3 := \frac{\partial k \sigma_y}{\partial k + \partial k k_1/(2\mu) + \nu/\mu}, \quad C_4 := 2\mu.$$ 

Lines 3–8 are analog to the last example. The strain tensor epsilon is calculated, and matrices $v$ and $v1$ are defined, with $v := \epsilon(U_0 - U_0) + C^{-1}\sigma_0, v1 :=$
\[- 1/(2\mu |\alpha_0|). In line 9 the fall differentiation is performed. If \(|\text{dev}(v)| - \sigma_j/(2\mu) > 0\) the plastic phase occurs, otherwise the elastic phase. Depending on the result of this distinction, a constant \(C_5\), a vector \(C_6\) and a Kronecker delta \(d\) are defined by

\[
C_5 := \begin{cases} 
\frac{C_3}{\text{dev}(v)} + C_2 & \text{if } |\text{dev}(v)| - \sigma_j/(2\mu) > 0 \\
\frac{C_4}{\text{dev}(v)} & \text{else}
\end{cases}
\]

(5.18)

\[
(C_6)_j := \begin{cases} 
\frac{C_3}{\text{dev}(v)} & \text{if } |\text{dev}(v)| - \sigma_j/(2\mu) > 0 \\
[0 0 0 0 0]^\top & \text{else}
\end{cases}
\]

(5.19)

\[
d := \begin{cases} 
1 & \text{if } |\text{dev}(v)| - \sigma_j/(2\mu) > 0 \\
0 & \text{else}.
\end{cases}
\]

(5.20)

In lines 14 and 15 an update of the local stiffness matrix and the first summand of the local vector \(R\) is performed. The components of the local stiffness matrix read

\[
M_{jk} = |T| (C_1 \text{tr}(\epsilon(\phi_j)) |\text{tr}(\epsilon(\phi_k)) + C_5 \text{dev}(\epsilon(\phi_j)) : \epsilon(\phi_k) - (C_6)_j \text{dev}(v) : \epsilon(\phi_j)).
\]

(5.21)

The components of the first summand of the vector \(R\) read

\[
R_j = |T| (C_1 \text{tr}(\epsilon(\phi_j)) |\text{tr}(\epsilon(\phi_j)) + C_5 \text{dev}(v) : \epsilon(\phi_j) + d \text{dev}(\alpha_0) : \epsilon(\phi_j)).
\]

(5.22)

The Matlab routine \texttt{stim.m} is listed below.

```matlab
1 function [M,R]=stim(90,RO,loccoordinates,u1,u0,j);
2 global musigma y C1 C2 CS O4 e0 a10
3 Dphi=inv([1.1,1;loccoordinates])*[0,0;eye(12)];
4 eps=zeros(6,4);Det1=zeros(6,4);Det2=zeros(6,4);
5 Det1(1:2;5:1:2)=Dphi;Det2(2:2;6:3:4)=Dphi;
6 Det2(1:2;6,1:3)=Dphi;Det2(2;2:6,1:3)=Dphi;
7 eps=(Det1*Det2)/2;v0=(u1-u0);v0(1:2,j,:)=v0(1:2,j,:)+a10(1:2,j,:);v=v0+v1;
8 T=det([1.1,1;loccoordinates])^2;
9 if norm(dev2(v))<sigma_y/(2*mu)
10   C5=C5+C5*norm(dev2(v))\0; OB=(C5/norm(dev2(v)))^3*dev2(eps)*dev2(v)';d=d+1;
11   else
12   C5=C5;OB=zeros(6,1);d=d;
13 end
14 N=(N(0)+T(1)*tr2(eps))';OB=C5*dev2(eps)*eps'-OB*dev2(v)*eps';
15 R=R0+(N(1)*tr2(eps))'+C5*eps*dev2(v*dv1)+d*eps*dev2(a10(1:2,j,:))';
```

6. POSTPROCESSING

6.1. Displaying the solution

Our two dimensional problems model the plain stress condition. In that case, the complete stress tensor \( \sigma \in \mathbb{R}^{3x3}_{\text{sym}} \) has the form

\[
\sigma = \begin{pmatrix} 
\sigma_{11} & \sigma_{12} & 0 \\
\sigma_{12} & \sigma_{22} & 0 \\
0 & 0 & 0 
\end{pmatrix}
\]
with \( \sigma_{33} = \frac{1}{2} \lambda / (\mu + \lambda) (\sigma_{11} + \sigma_{22}) \). It then follows for \( |\text{dev} \sigma|^2 \), where \( A := A - \frac{1}{4} \text{tr} \text{Ad}_{\Omega} \) and \( |A| := (\sum_{j,k=1}^{n} A_{jk}^2)^{1/2} \) is the Frobenius norm, that

\[
|\text{dev} \sigma|^2 = \left( \frac{\mu^2}{6(\mu + \lambda)} + \frac{1}{2} \right) (\sigma_{11} + \sigma_{22})^2 + 2(\sigma_{12}^2 - \sigma_{11} \sigma_{22}).
\]

The function `show.m`, listed at the end of this subsection, calculates the variable \( \text{AvS} \) in lines 2–9, which stores the nodal values of \( \sigma_h^2 \). Here \( \sigma_h \) is the stress calculated by the finite element method and \( \sigma_h^2 \) is a smoother approximation of the discrete stress \( \sigma_h \), which may be used for error control [8]. Line 10 determines the shear energy density \( |\text{dev} \sigma_h|^2 / (4\mu) \) and stores it in the variable \( \text{AvC} \). In line 13 the Matlab routine `trisurf` is used to draw the deformed mesh with a magnification of the displacement, which is fixed in line 11. Grey tones are attached to the displayed meshes which are proportional to the elastic shear energy density \( |\text{dev} \sigma_h|^2 / (4\mu) \).

```matlab
function show(coordinates, elements, Sigma, u, lambda, mu)
    AreaLine = zeros(size(coordinates,1),1);
    for j=1:size(elements,1)
        area = det([1 1 1;coordinates(j,:);coordinates(j+1,:)-coordinates(j,:)]/2);
        AreaLine = [AreaLine area];
    end;
    AvS = AreaLine./AreaLine(1:1:1:1);
    AvC = (mu/(2*(mu+lambda)-1))*(AvS(:,:)) + 2./(2+mu)*((AvS(:,:));... + AvS(:,:));
    trisurf(elements, magnify*u(1:2;size(u,1))*coordinates(1,:),... magnify*u(1:2;size(u,1))*coordinates(2,:),... zeros(size(coordinates,1),1),AvC,'facecolor','interp');
    view(0,90)
    colorbar('vert')
```

The Matlab routine `trisurf(elements,x,y,z,AvC,’facecolor’,’interp’)` is used to draw triangulations for equal types of elements. Every row of the matrix `elements` determines one polygon where the \( x, y \)- and \( z \)-coordinate of each corner of this polygon is given by the corresponding entry in \( x, y \) and \( z \). For a 2D-problem, we use for the \( z \)-coordinate the same value (zero) in all mesh points. The values \( AvC \) together with the options ‘facecolor’, ‘interp’ determine the grey tone of the area. The grey tone of each node is determined by \( AvC \), that of the rest of the area is linearly interpolated.

### 6.2. A posteriori error indication

The averaged stress field \( \sigma_h^2 \), allows an a posteriori error estimation by comparing it to the discrete (discontinuous) stress \( \sigma_h \). For one time step, the computable quantity

\[
\eta_h = \| \sigma_h - \sigma_h^2 \|_{L^2(\Omega)}
\]  

(6.1)
defines the error estimate [8] (for pure Dirichlet conditions – Neumann boundary conditions require modifications [9])

\[ \| \sigma - \sigma_h \|_{L^2(\Omega)} \leq C \eta_h. \]  

(6.2)

Here, \( \sigma_h^t \) may be any piecewise affine and globally continuous stress approximation. It is emphasized that the error bound (6.2) holds only if the time-discretization error is neglected. It is generally believed that the accumulation error (in time) cannot be captured by averaging (in space). However the quantity \( \eta_h \) may monitor the local spatial approximation error and a large size of \( \eta_h \) indicates a larger spatial error and motivates refinements.

**Theorem 6.1 (Efficency).** Assume that the stress field \( \sigma \) at time \( t \) is smooth, i.e. \( \sigma \in H^1(\Omega; \mathbb{R}^{d \times d}) \), then \( \eta_h \) is an efficient error estimator up to higher order terms in the sense that

\[ \eta_h \leq 4 \| \sigma - \sigma_h \|_{L^2(\Omega)} + h.o.t. \]  

(6.3)

where generically h.o.t. = \( O(\| h^2 D\sigma \|_{L^2(\Omega)}) \) \( \ll \| \sigma - \sigma_h \|_{L^2(\Omega)} \).

**Proof.** The triangle inequality provides, for all \( \tau_h \) which are globally continous and \( \mathcal{T} \)-piecewise affine (like \( \sigma_h^t \)), that

\[ \| \sigma_h - \tau_h \|_{L^2(\Omega)} \leq \| \sigma - \sigma_h \|_{L^2(\Omega)} + \| \sigma - \tau_h \|_{L^2(\Omega)} \leq \| \sigma - \sigma_h \|_{L^2(\Omega)} + h.o.t. \]  

(6.4)

for the nodal interpolation \( \tau_h = \| \sigma \) of \( \sigma \) [5]. One can prove that (with \( \tau_h \) globally continous and \( \mathcal{T} \)-piecewise affine but arbitrary otherwise)

\[ \eta_M := \min_{\tau_h} \| \sigma_h - \tau_h \|_{L^2(\Omega)} \leq \| \sigma_h - \sigma_h^t \|_{L^2(\Omega)} \leq 4 \eta_M. \]

This is shown for arbitrary constant in [6,14,16] and for the constant 4 and quite general boundary conditions in [7]. The main argument is a (local) inverse estimate, caused by equivalence of norms on finite dimensional vector spaces.

The combination of the two inequalities leads to

\[ \eta_h = \| \sigma - \sigma_h^t \|_{L^2(\Omega)} \leq 4 \eta_M \leq 4 \| \sigma - \sigma_h^t \|_{L^2(\Omega)} + 4 \| \sigma - \sigma_h \|_{L^2(\Omega)} \]

for any \( \tau_h \), in particular for \( \tau_h = \| \sigma \).

It is emphasized once more that the reliability (6.2) is problematic and holds only for one time step and for large hardening and large viscosity [8].

The Matlab routine at the end of this subsection calculates \texttt{eta1} at time \( t_t \). Lines 3–4 initialise \([\alpha_k] \), the patch of the hat function at node \( z \), which is stored in \texttt{AreaOmega} and \( \sigma_h^t \), which is stored in \( \sigmaVS \). Line 5 determines weights and barycentric coordinates by a two dimensional quadrature rule. In our algorithm we have \( \sigma_h^t(z) = \int_{\omega_k} \sigma_h dx / |\omega_k| \). Lines 6–10 calculate the area of the elements (line 7), the
patch size $|\omega|$ (line 8) and $\int_{\omega} \sigma_h \, dx$ (line 9) by a loop over all elements. In line 11, $\sigma^t_h$ is obtained by dividing $\int_{\omega} \sigma_h \, dx$ through the patch size. Lines 12–16 compute $\sigma^t_h$, where for $\sigma^t_h$ four interpolation points are used on each element $T$.

7. NUMERICAL EXAMPLES

7.1. On a plate with circular hole

A two dimensional squared plate with a hole, $\Omega = (-2, 2)^2 \setminus B(0, 1)$ is submitted to time dependent surface forces $g(t) = (600t)n$ at the top ($y = 2$) and the bottom ($y = -2$), where $n$ denotes the outer normal to $\partial \Omega$. The rest of the boundary is traction free. As the problem is symmetric only a quarter of $\Omega$ is discretised. The boundary conditions are specified in files $\text{u.txt}$, $\text{f.txt}$, and $\text{g.txt}$:

```matlab
function [W,M]=u_D(x,t)
M=zeros(2*size(x,1),2);
W=zeros(2*size(x,1),1);
% symmetry condition on the x-axis
tmp=find(x(:,1)>0 & x(:,2)==0);
M(2*tmp-1,1)=ones(size(tmp,1),1)*[0 1];
W(2*tmp-1,1)=zeros(size(tmp,1),1);
% symmetry condition on the y-axis
M(2*tmp-1,2)=ones(size(tmp,1),1)*[1 0];
W(2*tmp-1,2)=zeros(size(tmp,1),1);

function volforce = f(x);
volforce=zeros(size(x,1),2);

function sforce=g(x,n,t)
sforce=zeros(size(x,1),2);
if n(2)==1
sforce(:,2)=600*t;
end
```
This numerical example from [17] models perfect viscoplasticity with Young’s modulus $E = 206900$, Poisson’s ratio $\nu = 0.29$, yield stress $\sigma_y = 450$ and vanishing initial data for the stress tensor $\sigma_0$.

The solution is calculated with 3474 degrees of freedom in the time interval from $t = 0$ to $t = 0.4$ by the implicit Euler method in 8 uniform steps of length $k = 1/20$. The material remains elastic between $t = 0$ and $t = 0.2$. Figure 3 shows the deformed mesh at $t = 0.2$, Table 1 shows the number of iterations in Newton’s method and the estimated error for each time step.

Table 1.
Number of iterations and indicated error for membrane with hole in the example of Subsection 7.1 for various time steps.

| step | iterations | $\eta = ||\sigma_0 - \sigma_n||_2/||\sigma_0||_2$ |
|------|------------|----------------------------------|
| 1    | 1          | 0.0432                           |
| 2    | 1          | 0.0432                           |
| 3    | 1          | 0.0432                           |
| 4    | 5          | 0.0436                           |
| 5    | 5          | 0.0462                           |
| 6    | 5          | 0.0505                           |
| 7    | 6          | 0.0565                           |
| 8    | 6          | 0.0642                           |
7.2. Numerical examples for Cook's membrane problem

In the second numerical example we solve the viscoplastic problem with isotropic hardening for a two-dimensional model of a tapered panel of plexiglass described by \( \Omega = \text{conv} \{(0,0), (48,44), (48,60), (0,44)\} \). The panel is clamped at one end \((x = 0)\) and subjected to a shearing load \( g = (0,t) \) on the opposite end \((x = 48)\) with vanishing volume force \( f \). The boundary conditions are specified in files \( \text{m}, \text{m}, \text{m}, \text{m}, \text{m}, \text{m} \):

```matlab
function [W,M]=u_d(x,t)
M= zeros(2*size(x,1),2);
W= zeros(2*size(x,1),1);
M(1:2:2*size(x,1)-1,1)=ones(size(x,1),1);
M(2:2:2*size(x,1),2)=ones(size(x,1),1);
value=zeros(size(x,1),1);
W(1:2:2*size(u,1)-1,1)=value(:,:,1);
W(2:2:2*size(u,1),1)=value(:,:,2);

function volforce=f(x);
volforce= zeros(size(x,1),2);

function sforce=g(u,n,t)
sforce=zeros(size(x,1),2);
sforce(find(n(:,:,2)==1),2)=exp(2*t);
```

For plexiglass we set \( E = 2900 \) and \( \nu = 0.4 \). This example is often referred to as the Cook's membrane problem and constitutes a standard test for bending dominated response. We calculate the solution with 8450 degrees of freedom in the time interval \([0,0.1]\) with vanishing initial data for the displacement \( u_0 \), the stress tensor \( \sigma_0 \) and the hardening parameter \( \alpha_0 \). The time step in the implicit Euler method is \( k = 0.01 \). The plastic parameters are \( H = 1000, H_1 = 1 \) and \( \sigma_y = 0.1 \). The problem becomes plastic at \( t = 0.02 \). Figure 4 shows the deformed mesh at \( t = 0.1 \) for 2178 degrees of freedom, while Table 2 shows the number of iterations in Newton’s method and the estimated error for each time step.

**Table 2.**
Number of iterations and indicated error for Cook’s membrane problem in the example of Subsection 7.2 for various time steps.

| step | iterations | \( \eta = ||\sigma_{\eta} - \sigma_{\eta}^*||_2/||\sigma_{\eta}||_2 \) |
|------|------------|--------------------------------------------------|
| 1    | 1          | 0.0674                                           |
| 2    | 7          | 0.0674                                           |
| 3    | 11         | 0.0675                                           |
| 4    | 12         | 0.0675                                           |
| 5    | 13         | 0.0680                                           |
| 6    | 14         | 0.0684                                           |
| 7    | 14         | 0.0695                                           |
| 8    | 14         | 0.0706                                           |
| 9    | 14         | 0.0723                                           |
| 10   | 14         | 0.0737                                           |
7.3. Numerical example for axisymmetric ring

The third numerical example is taken from [4]. We solve a viscoplastic model with kinematic hardening. The geometry is a two-dimensional section of a long tube with inner radius 1 and outer radius 2 (Fig. 6). We have no volume force, \( f = 0 \) but time depending surface forces \( g_1(r, \phi, t) = te_r \) and \( g_2(r, \phi, t) = -t/4e_r \). The system is required to keep centered at the origin and rotation is prohibited. The boundary conditions are specified in files \( u_{\text{d}}.m \), \( f_{\text{m}} \), and \( g_{\text{m}} \):

```matlab
function [W,N] = u_D(x,t)
M = zeros(2*size(x,1),2);
W = zeros(2*size(x,1),1);
% symmetry condition on the x-axis
tmp = find(x(:,1)>0 & x(:,2)==0);
M(2*tmp-1,1:2) = ones(size(tmp,1),1)*[1 0];
W(2*tmp-1,1) = zeros(size(tmp,1),1);
% symmetry condition on the y-axis
tmp = find(x(:,2)>0 & x(:,1)==0);
M(2*tmp-1,1:2) = ones(size(tmp,1),1)*[0 1];
W(2*tmp-1,1) = zeros(size(tmp,1),1);

function volforce=f(x,t)
volforce=zeros(size(x,1),2);
```

Figure 4. Deformed mesh for Cooks' membrane problem in the example of Subsection 7.2 calculated with 2178 degrees of freedom. The grey tone visualises \( |\text{dev} \sigma|^2/(4\mu) \) as described in Subsection 6.1.
function sforce = g(x,n,t)
    [phi,r]=cart2pol(x(:,1),x(:,2));
    phiNeg=find(phi<0);
    phi(phiNeg)=phi(phiNeg)...
+2*pi*ones(size(phiNeg));
    for i=1:size(x,1)
        if r(i)>.5 & r(i)<1.5
            k(i,:)=t*[cos(phi(i)),sin(phi(i))];
        elseif r(i)>1.5 & r(i)<2.5
            k(i,:)=-t/4*[cos(phi(i)),sin(phi(i))];
        end
        if x(i,1)==0 | x(i,2)==0
            k(i,:)=[0 0];
        end
    end
end

The exact solution is

\[ u(r, \phi, t) = u_r(r, t)e_r \]

\[ \sigma(r, \phi, t) = \sigma_r(r, t)e_r \otimes e_r + \sigma_\phi e_\phi \otimes e_\phi \]

\[ p(r, \phi, t) = P_t(r, t)(e_r \otimes e_r - e_\phi \otimes e_\phi) \]

with \( e_r = (\cos \phi, \sin \phi) \), \( e_\phi = (-\sin \phi, \cos \phi) \) and with \( a = \mu + \lambda, \, \alpha = 2\mu/(2\mu + \lambda) \).

\[ u_r(r, t) = \begin{cases} \frac{t}{2\mu r} - \frac{1}{3} zI(R(t)) \left( r + \frac{4a}{\mu r} \right), & r \geq R(t) \\ \frac{t}{2\mu r} - \frac{1}{3} zI(R(t)) \left( 4r + \frac{4a}{\mu r} \right) + zI(r)r, & r < R(t) \end{cases} \]

(7.2)

\[ \sigma_r(r, t) = \begin{cases} -\frac{t}{r^2} - \frac{8}{3} a\alpha I(R(t)) \left( 1 - \frac{1}{r^2} \right), & r \geq R(t) \\ -\frac{t}{r^2} - \frac{8}{3} a\alpha I(R(t)) \left( \frac{1}{4} - \frac{1}{r^2} \right) + 2a\alpha I(r), & r < R(t) \end{cases} \]

(7.3)

\[ \sigma_\phi(r, t) = \frac{\partial (r \sigma_r)}{\partial r} \]

(7.4)

\[ P_t(r, t) = \begin{cases} 0, & r \geq R(t) \\ \frac{\sigma_\eta}{\sqrt{2}(a\alpha + k_1)}(1 - R^2/r^2), & r < R(t) \end{cases} \]

(7.5)

\[ I(r) = \frac{\sigma_\eta}{\sqrt{2}(a\alpha + k_1)}(\ln r + 1/2(R^2/r^2 - R^2)). \]

(7.6)

With \( \alpha = 4a\alpha/(3(a\alpha + k_1)) \), the radius of the plastic boundary \( R(t) \) is the positive root of

\[ f(R) = -2\alpha \ln R + (\alpha - 1)R^2 - \alpha + \frac{\sqrt{2}}{\sigma_\eta}t. \]

(7.7)
Table 3.
Number of iterations, indicated error, and exact error for axisymmetric ring in the example of Subsection 7.3. This numerical example validates our claim that $\eta$ can be an accurate error guess.

<table>
<thead>
<tr>
<th>step</th>
<th>iterations</th>
<th>$e = |\sigma - \sigma_0|_2/|\sigma_0|_2$</th>
<th>$\eta = |\sigma_h - \sigma_0|_2/|\sigma_0|_2$</th>
<th>$e/\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.0512</td>
<td>0.0510</td>
<td>1.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.0512</td>
<td>0.0510</td>
<td>1.00</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>0.0521</td>
<td>0.0515</td>
<td>1.01</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>0.0533</td>
<td>0.0529</td>
<td>1.01</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>0.0549</td>
<td>0.0543</td>
<td>1.01</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>0.0565</td>
<td>0.0560</td>
<td>1.01</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>0.0584</td>
<td>0.0579</td>
<td>1.01</td>
</tr>
</tbody>
</table>

The material parameters are $E = 70000$ for Young’s modulus, $\nu = 0.33$ for Poisson’s ratio and the yield stress is $\sigma_0 = 0.2$.

The solution is calculated in the time interval $[0,0.2]$ with 3202 degrees of freedom and vanishing initial data for the displacement $u_0$, the stress tensor $\sigma_0$ and the hardening parameter $\alpha_0$. The time step in the implicit Euler method is $k = 0.01$. The hardening parameter is $k_1 = 1$. The material becomes plastic at $t = 0.16$. Table 3 shows the error and the estimated error for the different time steps together with the number of iterations in Newton’s method. Figure 5 shows the deformed mesh at $t = 0.18$, displacements magnified by the factor 10.

7.4. Numerical example in three dimensions

As a three-dimensional example we solve a perfect viscoplastic problem on the cube $[0,1]^3 \setminus [0,0.5]^3$. The face with $x = 1$ is Dirichlet boundary, the rest is Neumann boundary. The volume force $f$ is zero while the surface force is $g = m$, if the normal vector $n$ points in y-direction. The boundary conditions are specified in files `u.d.m, f.m, and g.m`

```matlab
function [W,N] = u_d(x,t)
W= zeros(3*size(x,1),2);
N= zeros(3*size(x,1),1);
M(1:3:3*size(x,1)-2,1)= zeros(size(x,1),1);
M(2:3:3*size(x,1)-1,2)= zeros(size(x,1),1);
M(3:3:3*size(x,1),3)=zeros(size(x,1),1);
value=zeros(size(x,1),3);
W(1:3:3*size(x,1)-2,1)=value(:,1);
W(2:3:3*size(x,1)-1,2)=value(:,2);
W(3:3:3*size(x,1),1)=value(:,3);

function volforce = f(x);
volforce= zeros(size(x,1),2);

function sforce = g(u,n,t)
sforce= zeros(size(x,1),2);
sforce(find(n(:,2)==1),2)=t;
```
Figure 5. Deformed mesh for axisymmetric ring in the example of Subsection 7.3 calculated with 3202 degrees of freedom. The grey tone visualises $|\text{dev}\sigma|^2/(4\mu)$ as described in Subsection 6.1.

Figure 6. Axisymmetric ring in the example of Subsection 7.3 with initial triangulation of a quarter of the ring endowed with symmetric boundary conditions.

\begin{align*}
E &= 70000 \\
\nu &= 0.33 \\
\sigma_y &= 0.2 \\
k_1 &= 1 \\
g_1 &= t \varepsilon_r \\
g_2 &= -t/4\varepsilon_r
\end{align*}
The solution is calculated in the time interval [0,1] with stepsize 0.05 and vanishing initial data for the displacement $u_0$ and the stress tensor $\sigma_0$ and for material parameters $E = 206900$, $\nu = 0.29$, and yield stress $\sigma_y = 450$. Up to $t = 0.2$ s the material is entirely elastic. The deformed mesh for 10359 degrees of freedom at $t = 0.15$ s is shown in Fig. 7.

**Acknowledgments.** It is our pleasure to thank Dr. Jochen Alberty and Andre Geilenkothen for discussions. The second author (R. K.) thankfully acknowledges the partial support by the German research foundation at the Graduiertenkolleg 357 ‘Effiziente Algorithmen und Mehrskalenmethoden’. Parts of this work was done at the Institute for Applied Mathematics and Numerical Analysis, Vienna University of Technology whose hospitality is gratefully acknowledged.

**REFERENCES**


7. C. Carstensen, First-order averaging techniques for \textit{a posteriori} Finite Element error control on unstructured grids are efficient and reliable, 2002 (submitted).


8. APPENDIX

8.1. Main program in Example 1

1 global lambda mu sigma_y C1 C2 C3 O e0
2 th=1.0; time=0:0.05:0.4; nu=0;
3 %Initialization
4 load coordinates.dat; load elements.dat; load dirichlet.dat; load neumann.dat;
5 N=size(coordinates,1);initvector=zeros(2*N,1);
6 sigma0=zeros(size(elements,1),4);w0=zeros(2*N,1);
7 %material_parameters
8 lambda1=1.07438169066076e+05;mu=6.018379949661290e+04;C1=lambda1+mu;sigma_y=450;
9 for step=2:length(time)
10 t=itemp(step-1);ti=itemp(step);dt=t1-t0;
11 C2=mu/(mu+C2*mu)+w*dt;C3=dt+sigma_y/(mu/(2*mu)+w*dt);
12 e0=1/(4*(lambda1+mu))+(t2(sigma0)^[1,0,0,1]+1/(2*mu)*dev2(sigma0);
13 u_th=pem(coordinates,elements,dirichlet,neumann,initvector,w0,t0,ti,th,4);
14 sigma_th=tension(coordinates,elements,u_th,w0,sigma0);
8.2. Calculation of the stress in Example 1

```matlab
1 function sigma = tension(coordinates, elements, u_1, u_0, sigma0);
2 global lambda mu sigma_y C1 C2 C3 e0
3 sigma = sigma0;
4 for j=1:size(elements,1)
5 1coordinates = coordinates(elements(j,:,:));
6 1phi = inv([1,1,1;coordinates'])*[0,0;eye(2)];
7 Det1 = zeros(6,4); Det2 = zeros(6,4); u0 = zeros(6,1);
8 Deltai(1:2:5,1:2)=Dphi; Deltai(2:2:6,3:4)=Dphi;
9 Det2(1:2:5, [1,3]) = Dphi; Det2(2,2:6, [2,4]) = Dphi;
10 1u(1:2:5) = u(1:2*elements(j,:,:));
11 epsilon = Det2(1:2:5, 1:2)*u0(Deltai(1:2:5, 1:2));
12 eps = Det2(1:2:5, 1:2); u0(1:2*elements(j,:,:));
13 if norm(dev2(v)) - sigma_y/(2*mu) > 0
14 1sigma(1,:) = Ct + 2(v)*[1,0,0,1] + (C2*C3/norm(dev2(v)))*dev2(v);
15 else
16 1sigma(1,:) = Ct + 2(v)*[1,0,0,1]/2*mu*dev2(v);
17 end
18 end
```

8.3. Main program in Example 4

```matlab
1 global lambda mu sigma_y C1 C2 C3 e0
2 th=1:0.01:time=[0:0.01:1.1]; mu=0;
3 Initialisation
4 load coordinates.dat; load elements.dat; load dirichlet.dat; load neumann.dat;
5 load(coordinates,1); initvector=zeros(3*N,1); NJ=size (elements,1);
6 sigma = zeros (NJ,9); u0 = zeros (3*N,1);
7 Material parameters
8 lambda = 1.07432160196996e+05; mu = 0.19337984961240e+04; C1 = lambda +2*mu/3; sigma_y = 450;
9 for step=2:length(time)
10 t0=time(step-1); t1=time(step); dt = t1-t0;
11 C2 = mu/(2*mu)*t0*Dt; C3 = t0*Dt*sigma_y/(mu/(2*mu)+t0*Dt);
12 epsilon = zeros(12,9); Det1 = zeros(12,9); Det2 = zeros(12,9);
13 Det2 = 1:10,1:3*Dphi; Det2(1:2:3, 1:4:6) = Dphi; Det2(3:3:12, 7:9) = Dphi;
14 Det2(1:3:10, [1,4,7]) = Dphi; Det2(2:3:11, [2,5,8]) = Dphi; Det2(3:3:12, [3,6,9]) = Dphi;
```

8.4. Calculation of the stress in Example 4

```matlab
1 function sigma = tension(coordinates, elements, u_1, u_0, sigma0);
2 global mu sigma_y C1 C2 C3 e0
3 sigma = sigma0;
4 for j=1:size(elements,1)
5 1coordinates = coordinates(elements(j,:,:));
6 1phi = inv([1,1,1;coordinates'])*[zeros(1,3);eye(3)];
7 epsilon = zeros(12,9); Det1 = zeros(12,9); Det2 = zeros(12,9);
8 Det1(1:3,1:3) = Dphi; Det2(2:3, 1:4:6) = Dphi; Det2(3:3:12, 7:9) = Dphi;
9 Det2(1:3:10, [1,4,7]) = Dphi; Det2(2:3:11, [2,5,8]) = Dphi; Det2(3:3:12, [3,6,9]) = Dphi;
```
\begin{verbatim}
10 u1 = u_1(3*elements(j.reshape(repmat([1:4],3,1),1,1,12)) - repmat([2,1,0],1,1,1));
11 u0 = u_0(3*elements(j.reshape(repmat([1:4],3,1),1,1,12)) - repmat([2,1,0],1,1,1));
12 eps = (Delta1+Delta2)/2; v = (u1-u0)*eps+e0(j,:);
13 if norm(dev3(v)) - sigma_y/(2*mu) > 0
14   signal(j,:) = C1*tr3(v)*[1,0,0,0,1,0,0,0,1] + (C2*CS/norm(dev3(v)))*dev3(v);
15 else
16   signal(j,:) = C1*tr3(v)*[1,0,0,0,1,0,0,0,1] + 2*mu*dev3(v);
17 end
18 end
\end{verbatim}