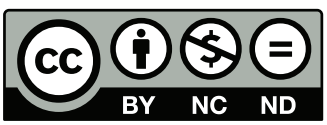
Constitutive Modelling Cookbook

Dr. Theodore Chang

April 25, 2024



Contents

List of Figures

List of Tables

1. Preface

Most books/papers present constitutive models analytically, leaving the implementation to readers to address.

This book aims to provide practical guidance to numerical implementation of plasticity models.

Readers are expected to have undergraduate level background of linear algebra, numerical analysis, elasticity and some programming knowledge.

1.1. Overview

The book presents a collection of constitutive models covering both uniaxial and triaxial and a wide range of common materials used in engineering. The implementation details are derived for each model.

The ultimate target is to provide an easy–to–follow, error and confusion free reference for readers who are interested in implementing constitutive models under the modern plasticity framework.

However, a wide range of different constitutive models are covered in this book, not only formulations but also implementations, some of which are very lengthy. There are a huge amount of symbols and formulae.

1.2. Implementation

For scientific computation that involves a significant amount of linear algebra operations, in my opinion, operator overloading is essential. Lengthy implementation is almost inevitable if the language does not support operator overloading. In this sense, account for performance, C++ is perhaps the first choice. However, Rust may be another candidate in the future.

Python and MATLAB, in the author's opinion, are ideal for experimenting. Due to performance related issues, they are not ready for production development.

For models only involve scalar operations, such as most uniaxial models, C/Fortran can be equivalently chosen. All variables can be grouped into structs and a typical state determination interface in C may look like this.

```c
struct State
{
    double strain;
    double stress;
    double stiffness;
    // more history variables
    double elastic_modulus;
    // more model constants and parameters
};

int update_state(const struct State *const current, struct State *const trial){
    // update trial state
    trial->stress = 2.;

    return 0;
}

int main()
{
    struct State current, trial;

    // set trial strain
    trial.strain = 1.;
    // call state determination
    int err = update_state(&current, &trial);
    // handle error
}
```

In all the state determination algorithms presented in this book, by default it is assumed all local iteraions eventually converge. Thus the handling of failure of convergence is not presented for brevity. One shall always bear in mind that not all algorithms converge, and for robust, practical algorithms, more numerical processes are required.

1.3. Digest

As the title indicates, this book is drafted as a reference book on constitutive modelling, it shall be self–complete in terms of theoretical part. For practical implementation, algorithms in pseudo code are given for most models. Core CPP implementations are also provided as references/comparisons for readers who are familiar with CPP and scientific computation with relevant tools. The code snippets are taken from the implementations from `suanPan` [1]. In this regard, this book can also be used as the program manual.

Part I.

Prerequisites

2. Tensor Basics

In engineering, often tensors are defined in Euclidean spaces and belong to Cartesian tensor (type). Transformation between covariant and contravariant bases, thus raising/lowering indices, is somehow not emphasised (but implied). The most likely reason is that, often orthonormal bases are used so that covariant and contravariant bases lead to the same coordinates. Thus, only subscripts (i, j, k, l, etc.) are used to represent tensor indices, implying that contravariant bases are used. Furthermore, scalars are denoted by normal symbols such as A while vectors and tensors are not distinguished, all denoted by boldface symbols such as $\boldsymbol{A}$.

2.1. Notations

Let $\boldsymbol{A} \in \mathbb{R}^2 \times \mathbb{R}^2$ denote a second order tensor with contravariant bases $\boldsymbol{e}^1$ and $\boldsymbol{e}^2$. With that, $\boldsymbol{A}$ has four components that can be arranged in the following matrix.

$$\boldsymbol{A} = A_{ij}\boldsymbol{e}^i \otimes \boldsymbol{e}^j = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}. \tag{2.1}$$

The tensor notation of $\boldsymbol{A}$ is the notation with indices, which refers to coordinates A_{ij}. The matrix representation refers to the 2D matrix of size two. The operator $\otimes$ stands for tensor product which will be introduced later. In some literature, it is omitted for simplicity, resulting in

$$\boldsymbol{A} = A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j. \tag{2.2}$$

This notation will be used in this book. In this case, $\boldsymbol{e}^i\boldsymbol{e}^j$ does not represent the dot product of two vectors. Rather, the operator $\cdot$ shall be explicitly shown as $\boldsymbol{e}^i \cdot \boldsymbol{e}^j$ to avoid potential confusion.

The Einstein summation convention is adopted, that is, in each single term, if the same index appears exactly twice, the summation of that term over all values of that index. In this example, $i, j = \{1, 2\}$. Thus,

$$\boldsymbol{A} = A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \sum_{i,j=1}^{2} A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = A_{11}\boldsymbol{e}^1\boldsymbol{e}^1 + A_{12}\boldsymbol{e}^1\boldsymbol{e}^2 + A_{21}\boldsymbol{e}^2\boldsymbol{e}^1 + A_{22}\boldsymbol{e}^2\boldsymbol{e}^2. \tag{2.3}$$

Since indices only act as placeholders, it does not matter which symbol is used. The following

expressions are equivalent.

$$\boldsymbol{A} = A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = A_{ik}\boldsymbol{e}^i\boldsymbol{e}^k = A_{ji}\boldsymbol{e}^j\boldsymbol{e}^i = A_{jk}\boldsymbol{e}^j\boldsymbol{e}^k = A_{ki}\boldsymbol{e}^k\boldsymbol{e}^i = A_{kj}\boldsymbol{e}^k\boldsymbol{e}^j. \tag{2.4}$$

But it is not equivalent to

$$\boldsymbol{A} \neq A_{ii}\boldsymbol{e}^i\boldsymbol{e}^i = A_{jj}\boldsymbol{e}^j\boldsymbol{e}^j = A_{kk}\boldsymbol{e}^k\boldsymbol{e}^k. \tag{2.5}$$

If $\boldsymbol{A}$ represents a symmetric stress tensor $\boldsymbol{\sigma}$, it can be expressed in both matrix and vector representations such as

$$\boldsymbol{\sigma} = \sigma_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \begin{bmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{21} & \sigma_{22}\end{bmatrix} = \begin{bmatrix}\sigma_x & \tau_{xy}\\ \tau_{xy} & \sigma_y\end{bmatrix} \quad \text{matrix representation,} \tag{2.6}$$

$$\boldsymbol{\sigma} = \begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix} = \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12} = \sigma_{21}\end{bmatrix} \quad \text{Voigt notation,} \tag{2.7}$$

$$\boldsymbol{\sigma} = \begin{bmatrix}\sigma_x\\ \sigma_y\\ \sqrt{2}\tau_{xy}\end{bmatrix} = \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sqrt{2}\sigma_{12} = \sqrt{2}\sigma_{21}\end{bmatrix} \quad \text{Mandel notation.} \tag{2.8}$$

Readers shall be familiar with the Voigt notation as it is widely used due to simplicity. The Mandel notation is an alternative that provides convenience when it comes to some tensor algebra operations. Examples will be shown later. It shall be noted that different vector/matrix representations of a tensor may have different components.

If $\boldsymbol{A}$ represents a symmetric strain tensor $\boldsymbol{\varepsilon}$, then its matrix and vector representations are

$$\boldsymbol{\varepsilon} = \varepsilon_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \begin{bmatrix}\varepsilon_{11} & \varepsilon_{12}\\ \varepsilon_{21} & \varepsilon_{22}\end{bmatrix} = \begin{bmatrix}\varepsilon_x & \frac{1}{2}\gamma_{xy}\\ \frac{1}{2}\gamma_{xy} & \varepsilon_y\end{bmatrix} \quad \text{matrix representation,} \tag{2.9}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{bmatrix} = \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ 2\varepsilon_{12} = 2\varepsilon_{21}\end{bmatrix} \quad \text{Voigt notation,} \tag{2.10}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \frac{\sqrt{2}}{2}\gamma_{xy}\end{bmatrix} = \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \sqrt{2}\varepsilon_{12} = \sqrt{2}\varepsilon_{21}\end{bmatrix} \quad \text{Mandel notation.} \tag{2.11}$$

In above, $\gamma_{xy} = 2\varepsilon_{12} = 2\varepsilon_{21}$ is commonly known as the engineering shear strain.

[2] presents a great discussion on compressed matrix representation covering both second order and fourth order tensors.

2.2. Tensor Operations

2.2.1. Tensor Product

Definition

The tensor product is also called dyadic product, which is an operation to construct high order tensor from low order tensors. Let $\boldsymbol{A} = A_i \boldsymbol{e}^i$ and $\boldsymbol{B} = B_i \boldsymbol{e}^i$ be two first order tensors (vectors), then tensor product of $\boldsymbol{A}$ and $\boldsymbol{B}$ gives a second order tensor $\boldsymbol{C}$

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = A_i \boldsymbol{e}^i \otimes B_j \boldsymbol{e}^j = A_i B_j \boldsymbol{e}^i \otimes \boldsymbol{e}^j = C_{ij} \boldsymbol{e}^i \otimes \boldsymbol{e}^j. \tag{2.12}$$

The simplified notation can also be adopted

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} = A_i \boldsymbol{e}^i B_j \boldsymbol{e}^j = A_i B_j \boldsymbol{e}^i \boldsymbol{e}^j = C_{ij} \boldsymbol{e}^i \boldsymbol{e}^j. \tag{2.13}$$

The components of $\boldsymbol{C}$ can be expressed as

$$C_{ij} = A_i B_j. \tag{2.14}$$

If $\boldsymbol{A} = A_{ij} \boldsymbol{e}^i \boldsymbol{e}^j$ and $\boldsymbol{B} = B_{ij} \boldsymbol{e}^i \boldsymbol{e}^j$ are two second order tensors, then the result is a fourth order tensor

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} = A_{ij} \boldsymbol{e}^i \boldsymbol{e}^j B_{kl} \boldsymbol{e}^k \boldsymbol{e}^l = A_{ij} B_{kl} \boldsymbol{e}^i \boldsymbol{e}^j \boldsymbol{e}^k \boldsymbol{e}^l = C_{ijkl} \boldsymbol{e}^i \boldsymbol{e}^j \boldsymbol{e}^k \boldsymbol{e}^l \tag{2.15}$$

with components $C_{ijkl} = A_{ij} B_{kl}$. If both $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric tensors, then $\boldsymbol{C}$ possesses major symmetry

$$C_{ijkl} = C_{klij} \tag{2.16}$$

and minor symmetry

$$C_{ijkl} = C_{jikl} = C_{ijlk}. \tag{2.17}$$

Vector/Matrix Representation

Let $\boldsymbol{A} = A_i \boldsymbol{e}^i \in \mathbb{R}^2$ and $\boldsymbol{B} = B_i \boldsymbol{e}^i \in \mathbb{R}^2$ be two first order tensors. Their column vector representations can be expressed as

$$\boldsymbol{A} = A_i \boldsymbol{e}^i = \begin{bmatrix} A_1 & A_2 \end{bmatrix}^{\mathrm{T}}, \qquad \boldsymbol{B} = B_i \boldsymbol{e}^i = \begin{bmatrix} B_1 & B_2 \end{bmatrix}^{\mathrm{T}}. \tag{2.18}$$

The tensor product Eq. (2.13) gives a second order tensor $\boldsymbol{C} \in \mathbb{R}^2 \times \mathbb{R}^2$ that can be represented by a matrix as shown in Eq. (2.1), which is

$$\boldsymbol{C} = C_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_1B_1 & A_1B_2 \\ A_2B_1 & A_2B_2 \end{bmatrix}. \tag{2.19}$$

Now if tensor $\boldsymbol{C}$ is treated as a matrix while tensors $\boldsymbol{A}$ and $\boldsymbol{B}$ are treated as column vectors, the tensor product is exactly the outer product between $\boldsymbol{A}$ and $\boldsymbol{B}$. That is,

$$\underbrace{\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{A}\boldsymbol{B}}_{\text{tensor product between tensors}} \qquad \longleftrightarrow \qquad \underbrace{\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}^{\mathrm{T}}}_{\text{vector/matrix representation}}. \tag{2.20}$$

Now let $\boldsymbol{A} = A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j \in \mathbb{R}^3 \times \mathbb{R}^3$ and $\boldsymbol{B} = B_{ij}\boldsymbol{e}^i\boldsymbol{e}^j \in \mathbb{R}^3 \times \mathbb{R}^3$ be two symmetric second order **stress** tensors, adopting the Voigt notation, their column vector representation can be shown as

$$\boldsymbol{A} = A_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \begin{bmatrix} A_{11} & A_{22} & A_{33} & A_{12} & A_{23} & A_{31} \end{bmatrix}^{\mathrm{T}}, \tag{2.21}$$

$$\boldsymbol{B} = B_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \begin{bmatrix} B_{11} & B_{22} & B_{33} & B_{12} & B_{23} & B_{31} \end{bmatrix}^{\mathrm{T}}. \tag{2.22}$$

The tensor product Eq. (2.15) between $\boldsymbol{A}$ and $\boldsymbol{B}$ gives the fourth order tensor $\boldsymbol{C} \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$ which can be arranged in a 2D matrix accounting for both major and minor symmetries.

$$\boldsymbol{C} = C_{ijkl}\boldsymbol{e}^i\boldsymbol{e}^j\boldsymbol{e}^k\boldsymbol{e}^l = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & C_{1112} & C_{1123} & C_{1131} \\ C_{2211} & C_{2222} & C_{2233} & C_{2212} & C_{2223} & C_{2231} \\ C_{3311} & C_{3322} & C_{3333} & C_{3312} & C_{3323} & C_{3331} \\ C_{1211} & C_{1222} & C_{1233} & C_{1212} & C_{1223} & C_{1231} \\ C_{2311} & C_{2322} & C_{2333} & C_{2312} & C_{2323} & C_{2331} \\ C_{3111} & C_{3122} & C_{3133} & C_{3112} & C_{3123} & C_{3131} \end{bmatrix}. \tag{2.23}$$

It can be observed Eq. (2.20) still applies since $C_{ijkl} = A_{ij}B_{kl}$. It is thus convenient to adopt the Voigt notation to perform tensor product between stress tensors.

However, if $\boldsymbol{A}$ and $\boldsymbol{B}$ are **strain** tensors, the Voigt notation leads to the following result of the outer product

$$\hat{\boldsymbol{C}} = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & 2C_{1112} & 2C_{1123} & 2C_{1131} \\ C_{2211} & C_{2222} & C_{2233} & 2C_{2212} & 2C_{2223} & 2C_{2231} \\ C_{3311} & C_{3322} & C_{3333} & 2C_{3312} & 2C_{3323} & 2C_{3331} \\ 2C_{1211} & 2C_{1222} & 2C_{1233} & 4C_{1212} & 4C_{1223} & 4C_{1231} \\ 2C_{2311} & 2C_{2322} & 2C_{2333} & 4C_{2312} & 4C_{2323} & 4C_{2331} \\ 2C_{3111} & 2C_{3122} & 2C_{3133} & 4C_{3112} & 4C_{3123} & 4C_{3131} \end{bmatrix}. \tag{2.24}$$

Fundamentally, different base tensors are used for stress tensors (contravariant) and strain tensors (covariant).

With the compressed matrix representations, one must be clear about such difference and apply proper scaling vectors/matrices when necessary.

2.2.2. Double Contraction

Definition

The double contraction, also known as the double dot product, is a tensor operation to construct low order tensors from high order tensors. Let $\boldsymbol{A} = A_i \boldsymbol{e}^i$ and $\boldsymbol{B} = B_i \boldsymbol{e}^i$ be two first order tensors (vectors), then dot product of $\boldsymbol{A}$ and $\boldsymbol{B}$ gives a zeroth order tensor (scalar) C as

$$C = \boldsymbol{A} \cdot \boldsymbol{B} = A_i \boldsymbol{e}^i \cdot B_j \boldsymbol{e}^j = A_i B_j \boldsymbol{e}^i \cdot \boldsymbol{e}^j = \delta_{ij} A_i B_j = A_i B_i, \tag{2.25}$$

where δ_{ij} is the Kronecker delta which equals 1 if $i = j$ or 0 otherwise.

If $\boldsymbol{A} = A_{ij} \boldsymbol{e}^i \boldsymbol{e}^j$ and $\boldsymbol{B} = B_{ij} \boldsymbol{e}^i \boldsymbol{e}^j$ are two second order tensors, then the double contraction performs dot product twice on different indices, resulting in a zeroth order tensor (scalar) as

$$\begin{aligned} C = \boldsymbol{A} : \boldsymbol{B} &= A_{ij} \boldsymbol{e}^i \boldsymbol{e}^j : B_{kl} \boldsymbol{e}^k \boldsymbol{e}^l = A_{ij} B_{kl} (\boldsymbol{e}^i \cdot \boldsymbol{e}^k)(\boldsymbol{e}^j \cdot \boldsymbol{e}^l) \\ &= \delta_{ik} \delta_{jl} A_{ij} B_{kl} = A_{ij} B_{ij}. \end{aligned} \tag{2.26}$$

Vector/Matrix Representation

Let $\boldsymbol{A} = A_{ij} \boldsymbol{e}^i \boldsymbol{e}^j \in \mathbb{R}^3 \times \mathbb{R}^3$ and $\boldsymbol{B} = B_{ij} \boldsymbol{e}^i \boldsymbol{e}^j \in \mathbb{R}^3 \times \mathbb{R}^3$ be two symmetric second order **stress** tensors. According to Eq. (2.26), the double contraction of two gives

$$\begin{aligned} C = \boldsymbol{A} : \boldsymbol{B} = A_{ij} B_{ij} &= A_{11} B_{11} + A_{22} B_{22} + A_{33} B_{33} \\ &+ A_{12} B_{12} + A_{13} B_{13} + A_{21} B_{21} + A_{23} B_{23} + A_{31} B_{31} + A_{32} B_{32}. \end{aligned} \tag{2.27}$$

It can be expressed via column vector representations (in the Voigt notation) of $\boldsymbol{A}$ and $\boldsymbol{B}$ as

$$C = \boldsymbol{A}^{\mathrm{T}} \boldsymbol{S} \boldsymbol{B} = \begin{bmatrix} A_{11} & A_{22} & A_{33} & A_{12} & A_{23} & A_{31} \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2 \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{22} \\ B_{33} \\ B_{12} \\ B_{23} \\ B_{31} \end{bmatrix}. \tag{2.28}$$

Its derivative reads

$$\frac{\partial C}{\partial \boldsymbol{A}} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{S}, \qquad \frac{\partial C}{\partial \boldsymbol{B}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{S}. \tag{2.29}$$

However, if the Mandel notation is adopted, it is simply

$$C = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{B} = \begin{bmatrix} A_{11} & A_{22} & A_{33} & \sqrt{2}A_{12} & \sqrt{2}A_{23} & \sqrt{2}A_{31} \end{bmatrix} \begin{bmatrix} B_{11} \\ B_{22} \\ B_{33} \\ \sqrt{2}B_{12} \\ \sqrt{2}B_{23} \\ \sqrt{2}B_{31} \end{bmatrix}. \tag{2.30}$$

If $\boldsymbol{A}$ and $\boldsymbol{B}$ represent **strain** tensors, a similar expression can be obtained with a different scaling matrix $\boldsymbol{S}$ using the Voigt notation.

$$\boldsymbol{S} = \mathrm{diag}\begin{pmatrix} 1 & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}. \tag{2.31}$$

With the Mandel notation, again no additional scaling matrix is required, $C = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$.

The Mandel notation is constructed on top of orthonormal basis of second order tensors. The advantage in obvious: there is no need to handle covariant and contravariant representations. However, it is not widely used as its physical meaning is not that obvious. In this book, the Voigt notation is used by default.

2.3. Stress Tensor Norm

The double contraction between a second order tensor $\boldsymbol{A}$ and itself results in a scalar that can be used to characterise the norm of $\boldsymbol{A}$. In this sense, double contraction of second order tensors can be deemed as an equivalent version of dot product of vectors.

Let $\boldsymbol{\sigma} \in \mathbb{R}^3 \times \mathbb{R}^3$ denote the symmetric stress tensor, define its Euclidean norm as

$$\begin{aligned} \|\boldsymbol{\sigma}\| &= \sqrt{\boldsymbol{\sigma} : \boldsymbol{\sigma}} \\ &= \sqrt{\sigma_{11}^2 + \sigma_{22}^2 + \sigma_{33}^2 + 2\sigma_{12}^2 + 2\sigma_{23}^2 + 2\sigma_{31}^2}. \end{aligned} \tag{2.32}$$

Accordingly, its normalised version

$$\boldsymbol{n} = \frac{\boldsymbol{\sigma}}{\|\boldsymbol{\sigma}\|}. \tag{2.33}$$

The derivative of $\|\boldsymbol{\sigma}\|$ can be computed accordingly via the chain rule.

$$\frac{\mathrm{d}\|\boldsymbol{\sigma}\|}{\mathrm{d}\boldsymbol{\sigma}} = \frac{1}{2}\frac{2\boldsymbol{\sigma} : \mathbb{I}}{\|\boldsymbol{\sigma}\|} = \boldsymbol{n} \qquad \text{in tensor notation,} \tag{2.34}$$

where $\mathbb{I}$ is the fourth order identity tensor. It shall be noted Eq. (2.34) is shown in tensor notation. For column vector representation in the Voigt notation, it shall be expressed as

$$\begin{aligned}\frac{\mathrm{d}\|\boldsymbol{\sigma}\|}{\mathrm{d}\boldsymbol{\sigma}} &= \frac{1}{2}\frac{1}{\|\boldsymbol{\sigma}\|}\begin{bmatrix}2\sigma_{11} & 2\sigma_{22} & 2\sigma_{33} & 4\sigma_{12} & 4\sigma_{23} & 4\sigma_{31}\end{bmatrix} \\ &= \left(\frac{\boldsymbol{\sigma}}{\|\boldsymbol{\sigma}\|}\right)^{\mathrm{T}} \mathrm{diag}\left(\begin{pmatrix}1 & 1 & 1 & 2 & 2 & 2\end{pmatrix}\right) \\ &= \frac{1}{\|\boldsymbol{\sigma}\|}\begin{bmatrix}\sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{12} & \sigma_{23} & \sigma_{31}\end{bmatrix}\begin{bmatrix}1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2\end{bmatrix}.\end{aligned} \tag{2.35}$$

It is clear that with the Voigt notation, the vector/matrix representation differs from the corresponding tensor representation. However, with the Mandel notation,

$$\frac{\mathrm{d}\|\boldsymbol{\sigma}\|}{\mathrm{d}\boldsymbol{\sigma}} = \frac{1}{\|\boldsymbol{\sigma}\|}\begin{bmatrix}\sigma_{11} & \sigma_{22} & \sigma_{33} & \sqrt{2}\sigma_{12} & \sqrt{2}\sigma_{23} & \sqrt{2}\sigma_{31}\end{bmatrix}, \tag{2.36}$$

which matches the tensor notation.

Now we proceed to compute the derivative of $\boldsymbol{n}$ with respect to $\boldsymbol{\sigma}$.

$$\frac{\mathrm{d}\boldsymbol{n}}{\mathrm{d}\boldsymbol{\sigma}} = \frac{\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\boldsymbol{\sigma}}\|\boldsymbol{\sigma}\| - \boldsymbol{\sigma}\frac{\mathrm{d}\|\boldsymbol{\sigma}\|}{\mathrm{d}\boldsymbol{\sigma}}}{\|\boldsymbol{\sigma}\|^2} = \frac{1}{\|\boldsymbol{\sigma}\|}\left(\mathbb{I} - \boldsymbol{n}\otimes\boldsymbol{n}\right) \qquad \text{in tensor notation.} \tag{2.37}$$

With the Voigt notation, it can be computed as follows.

$$\frac{\mathrm{d}\boldsymbol{n}}{\mathrm{d}\boldsymbol{\sigma}} = \frac{1}{\|\boldsymbol{\sigma}\|}\mathrm{diag}\begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \frac{1}{\|\boldsymbol{\sigma}\|^3}\begin{bmatrix}\sigma_{11}^2 & \sigma_{11}\sigma_{22} & \sigma_{11}\sigma_{33} & 2\sigma_{11}\sigma_{12} & 2\sigma_{11}\sigma_{23} & 2\sigma_{11}\sigma_{31}\\ \sigma_{11}\sigma_{22} & \sigma_{22}^2 & \sigma_{22}\sigma_{33} & 2\sigma_{12}\sigma_{22} & 2\sigma_{22}\sigma_{23} & 2\sigma_{22}\sigma_{31}\\ \sigma_{11}\sigma_{33} & \sigma_{22}\sigma_{33} & \sigma_{33}^2 & 2\sigma_{12}\sigma_{33} & 2\sigma_{23}\sigma_{33} & 2\sigma_{31}\sigma_{33}\\ \sigma_{11}\sigma_{12} & \sigma_{12}\sigma_{22} & \sigma_{12}\sigma_{33} & 2\sigma_{12}^2 & 2\sigma_{12}\sigma_{23} & 2\sigma_{12}\sigma_{31}\\ \sigma_{11}\sigma_{23} & \sigma_{22}\sigma_{23} & \sigma_{23}\sigma_{33} & 2\sigma_{12}\sigma_{23} & 2\sigma_{23}^2 & 2\sigma_{23}\sigma_{31}\\ \sigma_{11}\sigma_{31} & \sigma_{22}\sigma_{31} & \sigma_{31}\sigma_{33} & 2\sigma_{12}\sigma_{31} & 2\sigma_{23}\sigma_{31} & 2\sigma_{31}^2\end{bmatrix}. \tag{2.38}$$

It is essentially

$$\frac{\mathrm{d}\boldsymbol{n}}{\mathrm{d}\boldsymbol{\sigma}} = \frac{1}{\|\boldsymbol{\sigma}\|}\left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\begin{bmatrix}1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2\end{bmatrix}\right), \tag{2.39}$$

where $\boldsymbol{I}$ is the identity matrix of size 6.

2.4. Tensor Function of Stress Tensors

Some tensor–valued functions of stress tensors are frequently used in the analysis of plasticity. Let $\boldsymbol{\beta} = f(\boldsymbol{\sigma}, \boldsymbol{\alpha})$ denote a tensor-valued function of the stress tensor $\boldsymbol{\sigma}$ and some other tensors denoted by $\boldsymbol{\alpha}$. Let the tensor–valued function $\boldsymbol{\gamma} = g(\boldsymbol{\beta})$ be the normalised version of $\boldsymbol{\beta}$, that is

$$\boldsymbol{\gamma} = \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|} = \frac{\boldsymbol{\beta}}{\sqrt{\boldsymbol{\beta} : \boldsymbol{\beta}}}. \tag{2.40}$$

The partial derivative can be expressed as

$$\frac{\partial \boldsymbol{\gamma}}{\partial} = \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{\beta}} : \frac{\partial \boldsymbol{\beta}}{\partial} = \frac{1}{\|\boldsymbol{\beta}\|}(\mathbb{I} - \boldsymbol{\gamma} \otimes \boldsymbol{\gamma}) : \frac{\partial \boldsymbol{\beta}}{\partial} \qquad \text{in tensor notation.} \tag{2.41}$$

Depending on the form of $\boldsymbol{\beta}$, the compressed vector/matrix representation would differ. This will be dealt in specific context.

3. Plasticity Basics

Here we present a general framework of plasticity.

The idealisation of a typical elasto-plastic model can be represented by a frictional device as shown in Fig. 3.1. The device consists of an elastic spring element that deforms in an elastic manner, the deformation of which can be fully recovered when applied external force becomes zero, and a friction element, which deforms when deformation exceeds a certain limit and its deformation cannot be recovered even when applied external force becomes zero.

Most plastic models are formulated in **stress space**, or equivalently, **strain driven**. The task is to determine stress response based on strain input.

Figure 3.1.: idealisation of a typical elasto-plastic model

3.1. Decomposition of Strain

With the above model, it is clear that total strain $\boldsymbol{\varepsilon}$ can be decomposed into two parts, namely recoverable elastic strain $\boldsymbol{\varepsilon}^e$ and unrecoverable plastic strain $\boldsymbol{\varepsilon}^p$.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^e + \boldsymbol{\varepsilon}^p. \tag{3.1}$$

The elastic part obeys Hooke's law,

$$\boldsymbol{\sigma} = \boldsymbol{D} : \boldsymbol{\varepsilon}^e = \boldsymbol{D} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p)\,, \tag{3.2}$$

where $\boldsymbol{D}$ is the elastic stiffness moduli.

3.2. Yield Function

To be able to determine whether a given stress state is admissible, the yield function is introduced as the criterion. It is often a scaler–valued function of stress $\boldsymbol{\sigma}$ and some additional internal variables $\boldsymbol{q}$, viz.,

$$f = f\left(\boldsymbol{\sigma},\ \boldsymbol{q}\right). \tag{3.3}$$

By convention, it is defined so that all admissible stress states would lead to $f \leqslant 0$ while $f > 0$ is not allowed by any means. If f is deemed as a surface in stress space, all admissible stress states shall fall in the volume bounded by f. Thus yield function is also called yield surface in some literature, it may further evolve (change of size, location, etc.) with the development of plasticity. Sometimes it is further classified into two types: initial yield surface (when plasticity has not occurred) and subsequent yield surface (when plasticity has developed and is developing).

The internal variables $\boldsymbol{q}$ are called history variables.

3.3. Flow Rule

As plastic strain may evolve during the loading/unloading process, it is necessary to know how it evolves, the flow rule is defined as follows.

$$\dot{\boldsymbol{\varepsilon}^p} = \gamma \boldsymbol{r}\left(\boldsymbol{\sigma},\ \boldsymbol{q}\right), \tag{3.4}$$

where γ is a non-negative scalar called the consistency parameter, $\boldsymbol{r} = \boldsymbol{r}\left(\boldsymbol{\sigma},\ \boldsymbol{q}\right)$ is a tensor–valued function that indicates the direction of plastic flow.

In the context of plasticity, since time t is often not explicitly involved, the dot symbol $\dot{(\cdot)}$ above a quantity denotes its increment $\dot{(\cdot)} = \dfrac{\partial(\cdot)}{\partial t}\Delta t$ for brevity.

3.4. Hardening Law

In order to allow yield function to evolve, internal variables shall be governed by some function, which is defined as hardening law.

$$\dot{\boldsymbol{q}} = \gamma \boldsymbol{h}\left(\boldsymbol{\sigma},\ \boldsymbol{q}\right), \tag{3.5}$$

where $\boldsymbol{h} = \boldsymbol{h}\left(\boldsymbol{\sigma},\ \boldsymbol{q}\right)$ is another tensor–valued function that indicates the type of hardening.

3.5. Consistency Conditions

Since $f \not> 0$ and $\gamma \not< 0$, the number of admissible situations is limited:

- $f < 0$ — elastic loading/unloading
- $f = 0$ and $\dot{f} < 0$ — elastic unloading from yield surface
- $f = 0$ and $\dot{f} = 0$ and $\gamma > 0$ — plastic loading
- $f = 0$ and $\dot{f} = 0$ and $\gamma = 0$ — neutral loading on current yield surface

The above lengthy list can be elegantly characterised by the consistency conditions:

$$\gamma f = 0 \qquad \text{and} \qquad \gamma \dot{f} = 0. \tag{3.6}$$

3.6. State Determination

The yield function, flow rule and hardening law are three elements that complete most plastic models.

For a given strain input $\Delta\varepsilon$ and initial conditions, a general algorithm would possess the following structure.

Algorithm 1 general state determination algorithm of plastic models

```
freeze plasticity (assume elastic loading/unloading)
compute yield function f
if f ⩾ 0 then
    plasticity must develop
    compute plastic response
    update internal variables
else
    elastic loading/unloading
    compute elastic response
    no need to update internal variables
end if
```

3.6.1. Local Iteration

If plasticity must develop, noting that $f = 0$, with the assist of Eq. (3.2), $\boldsymbol{\sigma}$ can be equivalently represented by $\boldsymbol{\varepsilon}^p$, there are three unknown variables γ, $\boldsymbol{\varepsilon}^p$ and $\boldsymbol{q}$ to be solved subjected to the

following three equations.

$$\begin{cases} 0 = f\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right), \\ \dot{\boldsymbol{\varepsilon}^p} = \gamma \boldsymbol{r}\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right), \\ \dot{\boldsymbol{q}} = \gamma \boldsymbol{h}\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right). \end{cases} \tag{3.7}$$

Often the above system is nonlinear by construction and needs to be solved with an iterative method. It can be rewritten in a more nonlinear–flavoured style. Let $\boldsymbol{x} = \begin{bmatrix} \gamma & \boldsymbol{\varepsilon}^p & \boldsymbol{q} \end{bmatrix}$ denote the local variable vector, and $\boldsymbol{R}(\gamma,\ \boldsymbol{\varepsilon}^p,\ \boldsymbol{q})$ denote the residual of the nonlinear system

$$\boldsymbol{R} = \begin{cases} f\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right), \\ \dot{\boldsymbol{\varepsilon}^p} - \gamma \boldsymbol{r}\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right), \\ \dot{\boldsymbol{q}} - \gamma \boldsymbol{h}\left(\boldsymbol{\varepsilon}^p,\ \boldsymbol{q}\right). \end{cases} \tag{3.8}$$

The target is to find the solution $\boldsymbol{x}$ to $\boldsymbol{R} = \boldsymbol{0}$ for a prescribed total strain $\boldsymbol{\varepsilon}$. The Newton–Raphson method can be adopted with the help of Jacobian $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$. It shall be noted that partial derivatives are used here since fundamentally $\boldsymbol{\varepsilon}$ enters the system. It is also a variable which is held constant only at local iteration level.

3.6.2. Consistent Tangent Stiffness

To compute the consistent tangent stiffness, one can start from Eq. (3.2),

$$\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} = \boldsymbol{D} - \boldsymbol{D} : \frac{\partial \boldsymbol{\varepsilon}^p}{\partial \boldsymbol{\varepsilon}}. \tag{3.9}$$

Noting that at equilibrium, local residual shall be zero, viz., $\boldsymbol{R} = \boldsymbol{0}$. Taking full differentiation leads to

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}} \equiv \boldsymbol{0}, \tag{3.10}$$

thus

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}}. \tag{3.11}$$

The term $\dfrac{\partial \boldsymbol{\varepsilon}^p}{\partial \boldsymbol{\varepsilon}}$ can be extracted from $\dfrac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}}$.

Although not in the closed–form, the above procedure provides a universal approach to compute the consistent tangent stiffness in a highly efficient manner which can be applied to a wide range of plastic models.

Readers who are interested in plasticity theory please refer to [3] for more formal mathematical

formulations.

3.7. Some Tensor Quantities

Here we present some frequently used tensor quantities defined in 3D space.

3.7.1. Spherical Stress

The spherical stress of a stress tensor $\boldsymbol{\sigma}$ refers to the portion corresponds to an isotropic hydrostatic pressure p,

$$p = \frac{1}{3}\text{trace}\left(\boldsymbol{\sigma}\right) = \frac{1}{3}\sigma_{ii} = \frac{1}{3}\left(\sigma_{11} + \sigma_{22} + \sigma_{33}\right). \tag{3.12}$$

The corresponding spherical stress in the Voigt notation is then

$$p\mathbf{1} = \begin{bmatrix} p & p & p & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}. \tag{3.13}$$

3.7.2. Deviatoric Stress

The remaining portion is often known the deviatoric stress, denoted by $\boldsymbol{s}$.

$$\boldsymbol{s} = \text{dev}\left(\boldsymbol{\sigma}\right) = \boldsymbol{\sigma} - p\mathbf{1} = \begin{bmatrix} \sigma_{11} - p \\ \sigma_{22} - p \\ \sigma_{33} - p \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{bmatrix}. \tag{3.14}$$

In which, $\mathbf{1}$ is the unit second order tensor. In tensor notation,

$$\boldsymbol{s} = \mathbb{I}^{\text{dev}} : \boldsymbol{\sigma}. \tag{3.15}$$

By simple comparison, one can immediately find

$$\mathbb{I}^{\text{dev}} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix} \tag{3.16}$$

is the matrix representation of the fourth order deviatoric operator $\mathbb{I}^{\text{dev}}$. It must be noted that due to the adoption of the Voigt notation, the matrix representation of $\mathbb{I}^{\text{dev}}$ is **not** unique.

3.7.3. Volumetric Strain

It is possible to define the volumetric strain ε^v to be

$$\varepsilon^v = \text{trace}\,(\boldsymbol{\varepsilon}) = \varepsilon_{ii} = \varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}. \tag{3.17}$$

In tensor notation, it is

$$\varepsilon^v = \mathbf{1} : \boldsymbol{\varepsilon}. \tag{3.18}$$

3.7.4. Deviatoric Strain

Similarly, the remaining portion of strain is called the deviatoric strain $\boldsymbol{\varepsilon}^d$.

$$\boldsymbol{e} = \boldsymbol{\varepsilon}^d = \text{dev}\,(\boldsymbol{\varepsilon}) = \boldsymbol{\varepsilon} - \frac{\varepsilon^v}{3}\mathbf{1} = \begin{bmatrix} \varepsilon_{11} - \frac{\varepsilon^v}{3} \\ \varepsilon_{22} - \frac{\varepsilon^v}{3} \\ \varepsilon_{33} - \frac{\varepsilon^v}{3} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \gamma_{12} \\ \gamma_{23} \\ \gamma_{31} \end{bmatrix}. \tag{3.19}$$

In tensor notation,

$$\boldsymbol{e} = \boldsymbol{\varepsilon}^d = \mathbb{I}^{\text{dev}} : \boldsymbol{\varepsilon}. \tag{3.20}$$

3.7.5. Hooke's Law

The spherical part and deviatoric part are governed by two material parameters: bulk modulus K and shear modulus G, in tensor notation,

$$p = K\varepsilon^v, \qquad \boldsymbol{s} = 2G\boldsymbol{\varepsilon}^d. \tag{3.21}$$

Please note the latter in component form reads

$$\underbrace{\begin{bmatrix} s_{11} & s_{12} & s_{31} \\ s_{12} & s_{22} & s_{23} \\ s_{31} & s_{23} & s_{33} \end{bmatrix}}_{\boldsymbol{s}} = 2G \underbrace{\begin{bmatrix} \varepsilon^d_{11} & \varepsilon^d_{12} & \varepsilon^d_{31} \\ \varepsilon^d_{12} & \varepsilon^d_{22} & \varepsilon^d_{23} \\ \varepsilon^d_{31} & \varepsilon^d_{23} & \varepsilon^d_{33} \end{bmatrix}}_{\boldsymbol{\varepsilon}^d}. \tag{3.22}$$

The corresponding compressed matrix representation shall be expressed as

$$\underbrace{\begin{bmatrix} s_{11} \\ s_{22} \\ s_{33} \\ s_{12} \\ s_{23} \\ s_{31} \end{bmatrix}}_{\boldsymbol{s}} = 2G \underbrace{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \frac{1}{2} & & \\ & & & & \frac{1}{2} & \\ & & & & & \frac{1}{2} \end{bmatrix}}_{\mathbb{I}} \underbrace{\begin{bmatrix} \varepsilon^d_{11} \\ \varepsilon^d_{22} \\ \varepsilon^d_{33} \\ 2\varepsilon^d_{12} \\ 2\varepsilon^d_{23} \\ 2\varepsilon^d_{31} \end{bmatrix}}_{\boldsymbol{\varepsilon}^d} = 2G \underbrace{\begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & & & \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & & & \\ & & & \frac{1}{2} & & \\ & & & & \frac{1}{2} & \\ & & & & & \frac{1}{2} \end{bmatrix}}_{\mathbb{I}^{\mathrm{dev}}} \underbrace{\begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{bmatrix}}_{\boldsymbol{\varepsilon}}. \tag{3.23}$$

In this case, the matrix representation of $\mathbb{I}$ changes its form since it links strain with stress.

After some manipulations, total stress can be expressed by total strain as follows.

$$\begin{aligned} \boldsymbol{\sigma} &= \boldsymbol{s} + p\mathbf{1} \\ &= 2G\boldsymbol{\varepsilon}^d + K\varepsilon^v\mathbf{1} \\ &= 2G\mathbb{I}^{\mathrm{dev}} : \boldsymbol{\varepsilon} + K\mathbf{1}\otimes\mathbf{1} : \boldsymbol{\varepsilon} \\ &= \left(2G\mathbb{I}^{\mathrm{dev}} + K\mathbf{1}\otimes\mathbf{1}\right) : \boldsymbol{\varepsilon}. \end{aligned} \tag{3.24}$$

Thus,

$$\boldsymbol{D} = 2G\mathbb{I}^{\mathrm{dev}} + K\mathbf{1}\otimes\mathbf{1} = 2G\mathbb{I} - 2G\frac{1}{3}\mathbf{1}\otimes\mathbf{1} + K\mathbf{1}\otimes\mathbf{1}. \tag{3.25}$$

In matrix representation,

$$\boldsymbol{D} = 2G \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & \cdot & \cdot & \cdot \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \cdot & \cdot & \cdot \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \frac{1}{2} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \frac{1}{2} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \frac{1}{2} \end{bmatrix} + K \begin{bmatrix} 1 & 1 & 1 & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \tag{3.26}$$

Here one may observe $\mathbb{I}^{\mathrm{dev}}$ appears in a different form. This inconsistency is caused by the Voigt notation, as here the operand is strain tensor while the result is stress tensor. One can again refer to [2] for detailed explanation.

Using Lamé's constant $\lambda = K - \frac{2}{3}G$, $\boldsymbol{D}$ can be alternatively expressed as

$$\boldsymbol{D} = \begin{bmatrix} \lambda+2G & \lambda & \lambda & \cdot & \cdot & \cdot \\ \lambda & \lambda+2G & \lambda & \cdot & \cdot & \cdot \\ \lambda & \lambda & \lambda+2G & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & G & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & G & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & G \end{bmatrix}. \tag{3.27}$$

3.7.6. Lode Angle

For some material models, the Haigh–Westergaard stress space may be used. The lode angle is often used to define some parameter that alters the yield surface which may have a special shape on the π-plane. The lode angle θ can be defined as

$$\cos(3\theta) = \frac{J_3}{2}\left(\frac{3}{J_2}\right)^{1.5}, \tag{3.28}$$

where

$$J_2 = \frac{1}{2}\mathrm{trace}\left(\boldsymbol{s}^2\right) = \frac{1}{2}\|\boldsymbol{s}\|^2, \qquad J_3 = \frac{1}{3}\mathrm{trace}\left(\boldsymbol{s}^3\right) = \det \boldsymbol{s} \tag{3.29}$$

are the invariants of the deviatoric stress tensor $\boldsymbol{s}$.

Eq. (3.28) can be equivalently expressed as

$$\cos(3\theta) = \frac{6^{1.5}}{2}\frac{J_3}{\|\boldsymbol{s}\|^3} = \sqrt{54}\frac{\det \boldsymbol{s}}{\|\boldsymbol{s}\|^3} = \sqrt{54}\det\frac{\boldsymbol{s}}{\|\boldsymbol{s}\|}. \tag{3.30}$$

By the chain rule, the derivative of Eq. (3.28) can be expressed as

$$\frac{\mathrm{d}\cos(3\theta)}{\mathrm{d}\boldsymbol{s}} = \frac{1}{2}\left(\frac{3}{J_2}\right)^{1.5}\frac{\mathrm{d}J_3}{\mathrm{d}\boldsymbol{s}} - \frac{3^{2.5}}{4}\frac{J_3}{J_2^{2.5}}\frac{\mathrm{d}J_2}{\mathrm{d}\boldsymbol{s}}. \tag{3.31}$$

With the assist of

$$\frac{\mathrm{d}J_2}{\mathrm{d}\boldsymbol{s}} = \boldsymbol{s}, \qquad \frac{\mathrm{d}J_3}{\mathrm{d}\boldsymbol{s}} = \boldsymbol{s}\cdot\boldsymbol{s} - \frac{2}{3}J_2\mathbf{1}, \tag{3.32}$$

Eq. (3.31) becomes

$$\begin{aligned} \frac{\mathrm{d}\cos(3\theta)}{\mathrm{d}\boldsymbol{s}} &= \frac{1}{2}\left(\frac{3}{J_2}\right)^{1.5}\left(\boldsymbol{s}\cdot\boldsymbol{s} - \frac{2}{3}J_2\mathbf{1}\right) - \frac{3^{2.5}}{4}\frac{J_3}{J_2^{2.5}}\boldsymbol{s} \\ &= \sqrt{2}\left(\frac{3}{2J_2}\right)^{1.5}\left(\boldsymbol{s}\cdot\boldsymbol{s} - \frac{2J_2}{3}\mathbf{1} - \frac{3}{2J_2}J_3\boldsymbol{s}\right). \end{aligned} \tag{3.33}$$

Denoting $s = \dfrac{2J_2}{3}$, then

$$\frac{\mathrm{d}\cos(3\theta)}{\mathrm{d}\boldsymbol{s}} = \sqrt{2}s^{-1.5}\left(\boldsymbol{s}\cdot\boldsymbol{s} - s\mathbf{1} - \frac{J_3}{s}\boldsymbol{s}\right). \tag{3.34}$$

Part II.

Uniaxial Models

4. Uniaxial Metal Models

In this chapter, several models suitable for modelling metals are presented. We shall start from simple von Mises criterion based models, to more advanced uniaxial models suitable for modelling buckling restrained bracing members.

4.1. Linear Isotropic Hardening Model

4.1.1. Theory

For uniaxial models, the constitutive equation Eq. (3.2) simplifies to

$$\sigma = E\left(\varepsilon - \varepsilon^p\right) \tag{4.1}$$

where E is Young's modulus.

Yield Function

The (probably) most simplest yield function is

$$f = |\sigma| - \sigma^y \tag{4.2}$$

where $\sigma^y = \sigma^y\left(\sigma,\ q\right)$ is the yield stress which is a function of internal variable q thus would evolve.

For example, a linear function can be chosen so that

$$\sigma^y = \sigma^i + Kq, \tag{4.3}$$

where σ^i is the initial yield stress (non-negative) and K is the isotropic hardening modulus (either positive for hardening or negative for softening).

Flow Rule

It is reasonable to define that the direction of plastic flow coincides with the direction of stress, that is

$$\dot{\varepsilon^p} = \gamma \text{ sign}(\sigma). \tag{4.4}$$

Physically, it simply states plastic strain evolves towards the direction of stress. If stress is positive, plastic strain increases and vice versa.

Noting that $\dfrac{\partial f}{\partial \sigma} = \text{sign}(\sigma)$, the flow rule can also be expressed as

$$\dot{\varepsilon^p} = \gamma \frac{\partial f}{\partial \sigma}. \tag{4.5}$$

In 3D space, such a flow rule is often called the **associative** rule.

Hardening Law

For the internal hardening variable q, the simplest case would be

$$\dot{q} = |\dot{\varepsilon^p}| = \gamma. \tag{4.6}$$

Thus q characterises the accumulated magnitude of plastic strain. Given that $\gamma \geqslant 0$, it is clear that q is a non-decreasing (strictly increasing) function.

We are ready to formulate and implement our first plastic model.

4.1.2. Formulation

In this book, subscript $(\cdot)_n$ is adopted to indicate initial conditions (current state or converged state) and subscript $(\cdot)_{n+1}$ is adopted to indicate solution (new state or trial state). Sometimes $(\cdot)_{n+1}$ is omitted for simplicity.

The summation of this simple isotropic hardening model is listed as follows.

Constitutive Law	$\sigma = E(\varepsilon - \varepsilon^p)$
Yield Function	$f = \lvert\sigma\rvert - (\sigma^i + Kq)$
Flow Rule	$\dot{\varepsilon^p} = \gamma \text{ sign}(\sigma)$
Hardening Law	$\dot{q} = \gamma$

For this example, the initial conditions are stress σ_n, total strain ε_n, plastic strain ε_n^p and

hardening variable q_n. For a given ε_{n+1}, the model shall compute new σ_{n+1}, ε^p_{n+1} and q_{n+1}.

Elastic Loading/Unloading

By freezing plasticity, one can first check if the new state is elastic. Since it may not be the final new state, we denote the computed stress and trial stress. According to

$$\sigma^{\text{trial}}_{n+1} = E\left(\varepsilon_{n+1} - \varepsilon^p_n\right), \tag{4.7}$$

or equivalently,

$$\sigma^{\text{trial}}_{n+1} = \sigma_n + E\left(\varepsilon_{n+1} - \varepsilon_n\right), \tag{4.8}$$

the yield function becomes

$$f^{\text{trial}} = \left|\sigma^{\text{trial}}\right| - \sigma^y = E|\varepsilon_{n+1} - \varepsilon^p_n| - \left(\sigma^i + Kq_n\right), \tag{4.9}$$

$$f^{\text{trial}} = \left|\sigma^{\text{trial}}\right| - \sigma^y = |\sigma_n + E\left(\varepsilon_{n+1} - \varepsilon_n\right)| - \left(\sigma^i + Kq_n\right). \tag{4.10}$$

If $f^{\text{trial}} < 0$, indicating elastic loading/unloading, the new state is simply $\varepsilon^p_{n+1} = \varepsilon^p_n$, $q_{n+1} = q_n$ and $\sigma_{n+1} = \sigma^{\text{trial}}_{n+1}$.

Plastic Evolution

Otherwise the new state consists of new evolution of plasticity. In this case,

$$\sigma_{n+1} = E\left(\varepsilon_{n+1} - \varepsilon^p_{n+1}\right). \tag{4.11}$$

The yield function is

$$f = |\sigma| - \sigma^y = E|\varepsilon_{n+1} - \varepsilon^p_{n+1}| - \left(\sigma^i + Kq_{n+1}\right) = 0. \tag{4.12}$$

The flow rule and hardening law shall be expressed as

$$\varepsilon^p_{n+1} = \varepsilon^p_n + \dot{\varepsilon^p} = \varepsilon^p_n + \gamma\ \text{sign}\left(\sigma_{n+1}\right), \tag{4.13}$$

$$q_{n+1} = q_n + \dot{q} = q_n + \gamma. \tag{4.14}$$

Noting that

$$\begin{aligned} \sigma_{n+1} = \sigma^{\text{trial}}_{n+1} - E\dot{\varepsilon^p} \quad &\longrightarrow \quad \sigma_{n+1} + E\gamma\ \text{sign}\left(\sigma_{n+1}\right) = \sigma^{\text{trial}}_{n+1} \\ &\longrightarrow \quad \left(|\sigma_{n+1}| + E\gamma\right)\ \text{sign}\left(\sigma_{n+1}\right) = \left|\sigma^{\text{trial}}_{n+1}\right|\ \text{sign}\left(\sigma^{\text{trial}}_{n+1}\right), \end{aligned} \tag{4.15}$$

since both E and γ are non-negative, $\text{sign}\left(\sigma_{n+1}\right) = \text{sign}\left(\sigma_{n+1}^{\text{trial}}\right)$ and $|\sigma_{n+1}| + E\gamma = \left|\sigma_{n+1}^{\text{trial}}\right|$. Thus inserting the above expressions into the yield function, one can obtain

$$\left|\sigma_{n+1}^{\text{trial}} - E\gamma \ \text{sign}\left(\sigma_{n+1}^{\text{trial}}\right)\right| - \left(\sigma^i + K\left(q_n + \gamma\right)\right) = 0. \tag{4.16}$$

After some manipulations, it is

$$\left|\sigma_{n+1}^{\text{trial}}\right| - E\gamma - \left(\sigma^i + K\left(q_n + \gamma\right)\right) = 0. \tag{4.17}$$

The above expression holds since $\left|\sigma_{n+1}^{\text{trial}}\right| - E\gamma = |\sigma_{n+1}|$ is non-negative.

The consistency parameter can be solved as

$$\gamma = \frac{\left|\sigma_{n+1}^{\text{trial}}\right| - \left(\sigma^i + Kq_n\right)}{E + K} = \frac{f^{\text{trial}}}{E + K}. \tag{4.18}$$

It is easy to further compute

$$\frac{\partial \gamma}{\partial \varepsilon_{n+1}} = \frac{E}{E + K}. \tag{4.19}$$

4.1.3. Implementation

The implementation is quite straightforward for such a simple model. Algorithm 2 summarised the state determination algorithm for the above isotropic hardening model.

Algorithm 2 state determination of uniaxial isotropic hardening model

Parameter: E, K
Input: ε_{n+1}, ε_n, ε_n^p, σ_n, q_n
Output: E_{n+1}, ε_{n+1}^p, σ_{n+1}, q_{n+1}
compute σ^{trial} and f^{trial}
if $f^{\text{trial}} \geqslant 0$ **then**
    $\gamma = \dfrac{f^{\text{trial}}}{E + K}$
    $\varepsilon_{n+1}^p = \varepsilon_n^p + \gamma \ \text{sign}\left(\sigma_{n+1}^{\text{trial}}\right)$
    $q_{n+1} = q_n + \gamma$
    $\sigma_{n+1} = \sigma_{n+1}^{\text{trial}} - E\gamma \ \text{sign}\left(\sigma_{n+1}^{\text{trial}}\right)$
    $E_{n+1} = E - \dfrac{E^2}{E + K} = \dfrac{EK}{E + K}$
else
    $\varepsilon_{n+1}^p = \varepsilon_n^p$
    $q_{n+1} = q_n$
    $\sigma_{n+1} = \sigma_{n+1}^{\text{trial}}$
    $E_{n+1} = E$

16: **end if**

4.2. Combined Isotropic/Kinematic Hardening Model

Isotropic hardening controls the size of yield surface without changing its location. Kinematic hardening, on contrary, changes yield surface location but does not touch its size. Combining isotropic hardening and kinematic hardening allows flexible response to be modelled.

4.2.1. Theory

Yield Function

Since σ^y characterises the size of yield surface, a natural approach to allow yield surface to move around is to introduce the explicit location of its centre.

$$f = |\eta| - \sigma^y, \tag{4.20}$$

with $\eta = \sigma - \alpha$ is defined to be the shifted stress with α denoting the back stress. The additional stress quantity α characterises the centre of yield surface.

Flow Rule

Assuming associative rule, the flow rule shall be updated as

$$\dot{\varepsilon^p} = \gamma \frac{\partial f}{\partial \sigma} = \gamma \ \text{sign}\,(\eta)\,. \tag{4.21}$$

Hardening Law

The additional internal variable α shall evolve as well. It can take a similar form as follows.

$$\dot{\alpha} = H\dot{\varepsilon^p} = \gamma H \ \text{sign}\,(\eta)\,. \tag{4.22}$$

in which H denotes the kinematic hardening modulus. The existing hardening law for q does not altered.

$$\dot{q} = |\dot{\varepsilon^p}| = \gamma. \tag{4.23}$$

4.2.2. Formulation

The summation of this combined isotropic/kinematic hardening model is listed as follows.

Constitutive Law	$\sigma = E\left(\varepsilon - \varepsilon^p\right)$
Yield Function	$f = \lvert\sigma - \alpha\rvert - \left(\sigma^i + Kq\right)$
Flow Rule	$\dot{\varepsilon^p} = \gamma \text{ sign}\left(\sigma - \alpha\right)$
Hardening Law	$\dot{q} = \gamma$
	$\dot{\alpha} = \gamma H \text{ sign}\left(\sigma - \alpha\right)$

The algorithm aims to compute new σ_{n+1}, ε^p_{n+1}, α_{n+1} and q_{n+1} based on current σ_n, ε_n, ε^p_n, α_n, q_n and new ε_{n+1}.

Elastic Loading/Unloading

The trial stress can be computed following Eq. (4.8). Then by denoting $\eta^{\text{trial}} = \sigma^{\text{trial}} - \alpha_n$, the yield function becomes

$$f^{\text{trial}} = \left|\eta^{\text{trial}}\right| - \left(\sigma^i + Kq_n\right). \tag{4.24}$$

Plastic Evolution

With new state variables, let $\eta_{n+1} = \sigma_{n+1} - \alpha_{n+1}$, compute the yield function as follows.

$$\begin{aligned} f &= \left|\sigma_{n+1} - \alpha_{n+1}\right| - \left(\sigma^i + Kq_{n+1}\right) \\ &= \left|\sigma^{\text{trial}}_{n+1} - E\gamma \text{ sign}\left(\eta_{n+1}\right) - \alpha_n - H\gamma \text{ sign}\left(\eta_{n+1}\right)\right| - \left(\sigma^i + Kq_n + K\gamma\right) \\ &= \left|\eta^{\text{trial}}_{n+1} - \left(E + H\right)\gamma \text{ sign}\left(\eta_{n+1}\right)\right| - \left(\sigma^i + Kq_n + K\gamma\right) = 0. \end{aligned} \tag{4.25}$$

Similar to Eq. (4.15), it can be derived that

$$\text{sign}\left(\eta^{\text{trial}}_{n+1}\right) = \text{sign}\left(\eta_{n+1}\right), \qquad \left|\eta^{\text{trial}}_{n+1}\right| = \left|\eta_{n+1}\right| + \left(E + H\right)\gamma. \tag{4.26}$$

Thus,

$$f = \left|\eta^{\text{trial}}_{n+1}\right| - \left(E + H\right)\gamma - \left(\sigma^i + Kq_n + K\gamma\right) = 0. \tag{4.27}$$

From which η can be solved.

$$\eta = \frac{\left|\eta^{\text{trial}}_{n+1}\right| - \left(\sigma^i + Kq_n\right)}{E + H + K} = \frac{f^{\text{trial}}}{E + H + K}. \tag{4.28}$$

4.2.3. Implementation

The state determination algorithm resembles the previous one for isotropic hardening model.

Algorithm 3 state determination of uniaxial combined isotropic/kinematic hardening model

Parameter: E, H, K
Input: ε_{n+1}, ε_n, ε^p_n, σ_n, α_n, q_n
Output: E_{n+1}, ε^p_{n+1}, σ_{n+1}, α_{n+1}, q_{n+1}
compute σ^{trial}, η^{trial} and f^{trial}
if $f^{\text{trial}} \geqslant 0$ **then**
    $\gamma = \dfrac{f^{\text{trial}}}{E + H + K}$
    $\varepsilon^p_{n+1} = \varepsilon^p_n + \gamma \operatorname{sign}\left(\eta^{\text{trial}}_{n+1}\right)$
    $q_{n+1} = q_n + \gamma$
    $\alpha_{n+1} = \alpha_n + H\gamma \operatorname{sign}\left(\eta^{\text{trial}}_{n+1}\right)$
    $\sigma_{n+1} = \sigma^{\text{trial}}_{n+1} - E\gamma \operatorname{sign}\left(\eta^{\text{trial}}_{n+1}\right)$
    $E_{n+1} = E - \dfrac{E^2}{E + H + K} = \dfrac{E\,(H + K)}{E + H + K}$
else
    $\varepsilon^p_{n+1} = \varepsilon^p_n$
    $q_{n+1} = q_n$
    $\alpha_{n+1} = \alpha_n$
    $\sigma_{n+1} = \sigma^{\text{trial}}_{n+1}$
    $E_{n+1} = E$
end if

4.3. Armstrong–Fredrick Hardening Model

So far, two simple models has been introduced. The linear isotropic/kinematic hardening law is adopted so that the local residual is a linear function which can be solved within one step. However, linear hardening has limited applications.

To allow more versatile applications, in this section, a metal model incorporating Armstrong–Fredrick type kinematic hardening [4] and Voce type isotropic hardening [5] is introduced. Both hardening types are nonlinear.

4.3.1. Theory

Yield Function and Flow Rule

The same yield function and flow rule used in the previous combined isotropic/kinematic hardening model are adopted.

$$f = |\eta| - \sigma^y, \tag{4.29}$$

$$\dot{\varepsilon^p} = \gamma \frac{\partial f}{\partial \sigma} = \gamma n = \gamma \ \text{sign}\left(\eta\right), \tag{4.30}$$

in which $\eta = \sigma - \alpha$ is the shifted stress, $\alpha = \alpha\left(q\right)$ is the back stress, $\sigma^y = \sigma^y\left(q\right)$ is the isotropic hardening stress and $n = \text{sign}\left(\eta\right)$.

Hardening Law

The same hardening law for q is adopted as well.

$$\dot{q} = |\dot{\varepsilon^p}| = \gamma. \tag{4.31}$$

Isotropic Hardening Instead of using a simple linear function, a Voce type function is adopted for σ^y.

$$\sigma^y = \sigma^i + Kq + \sigma^s\left(1 - e^{-mp}\right), \tag{4.32}$$

in which K denotes the linear hardening modulus, σ^s is the satuated stress that denotes the size of additional yield stress caused by exponential hardening and m is a model parameter that controls the rate of exponential part of hardening.

If $K = 0$, it can be seen that

$$\lim_{p\to\infty} \sigma^y = \sigma^i + \sigma^s, \qquad \lim_{p\to 0} \sigma^y = \sigma^i. \tag{4.33}$$

Kinematic Hardening A multiplicative formulation [6] for back stress is adopted with the Armstrong–Fredrick type hardening rule. The back stress is defined to be the summation of several back stresses that evolve independently with different rates. That is,

$$\alpha = \sum_{i=1}^{n} \alpha_i \tag{4.34}$$

with

$$\dot{\alpha_i} = a_i \dot{\varepsilon^p} - b_i \alpha_i \dot{q} \tag{4.35}$$

where a_i and b_i are two sets of model parameters. In terms of γ, it is

$$\dot{\alpha}_i = a_i \gamma n - b_i \alpha_i \gamma. \tag{4.36}$$

4.3.2. Formulation

The summation of this AF model is listed as follows.

Constitutive Law	$\sigma = E\left(\varepsilon - \varepsilon^p\right)$
Yield Function	$f = \lvert\sigma - \alpha\rvert - \left(\sigma^i + Kp + \sigma^s\left(1 - e^{-mp}\right)\right)$
Flow Rule	$\dot{\varepsilon^p} = \gamma \operatorname{sign}\left(\sigma - \alpha\right)$
Hardening Law	$\dot{q} = \gamma$
	$\alpha = \sum_{i=1}^{n} \alpha_i$
	$\dot{\alpha}_i = a_i \gamma n - b_i \alpha_i \gamma$

Elastic Loading/Unloading

The trial stress can be computed following Eq. (4.8). Then by denoting $\eta^{\text{trial}} = \sigma^{\text{trial}} - \alpha_n$, the yield function becomes

$$f^{\text{trial}} = \left|\eta^{\text{trial}}\right| - \left(\sigma^i + Kp_n + \sigma^s\left(1 - e^{-mp_n}\right)\right). \tag{4.37}$$

Plastic Evolution

It is now clear that nonlinearity is introduced since $\dot{\alpha}_i = g\left(\alpha_i, \cdots\right)$ is a function of α_i and other variables. Various first order numerical methods can be applied. For example,

- explicit/forward Euler method

$$\alpha_{i,n+1} = \alpha_{i,n} + g\left(\alpha_{i,n}, \cdots\right), \tag{4.38}$$

- implicit/backward Euler method

$$\alpha_{i,n+1} = \alpha_{i,n} + g\left(\alpha_{i,n+1}, \cdots\right), \tag{4.39}$$

- mid-point method

$$\alpha_{i,n+1} = \alpha_{i,n} + \frac{g\left(\alpha_{i,n}, \cdots\right) + g\left(\alpha_{i,n+1}, \cdots\right)}{2}. \tag{4.40}$$

We mainly use the implicit Euler method in our plastic models, although it is significantly more complex than the explicit one.

Then $\alpha_{i,n+1}$ can be expressed as

$$\alpha_{i,n+1} = \alpha_{i,n} + \dot{\alpha_i} = \alpha_{i,n} + a_i \gamma n - b_i \alpha_{i,n+1} \gamma. \tag{4.41}$$

Thus,

$$\alpha_{i,n+1} = \frac{\alpha_{i,n} + a_i \gamma n}{1 + b_i \gamma}. \tag{4.42}$$

Then,

$$\alpha_{n+1} = \sum \frac{\alpha_{i,n} + a_i \gamma n}{1 + b_i \gamma}. \tag{4.43}$$

The shifted stress can be then computed as

$$\eta_{n+1} = \sigma_{n+1} - \alpha_{n+1} = \sigma^{\text{trial}} - E\gamma n - \sum \frac{\alpha_{i,n} + a_i \gamma n}{1 + b_i \gamma}. \tag{4.44}$$

Rearranging leads to

$$\eta_{n+1} + E\gamma n + \sum \frac{a_i \gamma n}{1 + b_i \gamma} = \sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma}. \tag{4.45}$$

Given that $n = \operatorname{sign}(\eta_{n+1})$ and $\gamma \geqslant 0$,

$$\left(|\eta_{n+1}| + E\gamma + \sum \frac{a_i \gamma}{1 + b_i \gamma} \right) n = \left| \sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma} \right| \operatorname{sign} \left(\sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma} \right). \tag{4.46}$$

Thus,

$$n = \operatorname{sign}(\eta_{n+1}) = \operatorname{sign} \left(\sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma} \right), \tag{4.47}$$

$$|\eta_{n+1}| + E\gamma + \sum \frac{a_i \gamma}{1 + b_i \gamma} = \left| \sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma} \right|. \tag{4.48}$$

With the above expressions, the yield function can be evaluated as

$$\begin{aligned} f &= |\eta_{n+1}| - \left(\sigma^i + K q_{n+1} + \sigma^s \left(1 - e^{-m q_{n+1}} \right) \right) \\ &= \left| \sigma^{\text{trial}} - \sum \frac{\alpha_{i,n}}{1 + b_i \gamma} \right| - E\gamma - \sum \frac{a_i \gamma}{1 + b_i \gamma} - \sigma^i - K q_n - K\gamma - \sigma^s \left(1 - e^{-m(q_n + \gamma)} \right) \\ &= 0. \end{aligned} \tag{4.49}$$

The Newton–Raphson method shall be used to solve this nonlinear equation. The Jacobian

reads

$$\frac{\partial f}{\partial \gamma} = \sum \frac{b_i \alpha_{i,n} n - a_i}{(1 + b_i \gamma)^2} - E - K - m\sigma^s e^{-m(q_n + \gamma)}. \tag{4.50}$$

The tangent modulus shall be computed via the chain rule.

$$\frac{\partial \sigma_{n+1}}{\partial \varepsilon_{n+1}} = \frac{\partial \sigma^{\text{trial}}}{\partial \varepsilon_{n+1}} + En \frac{\partial \gamma}{\partial \varepsilon_{n+1}} = E + En \frac{\partial \gamma}{\partial \varepsilon_{n+1}}. \tag{4.51}$$

Following the general procedure Eq. (3.10), at equilibrium, the full differentiation of yield function is

$$\frac{\partial f}{\partial \varepsilon_{n+1}} + \frac{\partial f}{\partial \gamma} \frac{\partial \gamma}{\partial \varepsilon_{n+1}} = 0. \tag{4.52}$$

Thus,

$$\frac{\partial \gamma}{\partial \varepsilon_{n+1}} = -\left(\frac{\partial f}{\partial \gamma}\right)^{-1} \frac{\partial f}{\partial \varepsilon_{n+1}}. \tag{4.53}$$

In which,

$$\frac{\partial f}{\partial \varepsilon_{n+1}} = nE, \tag{4.54}$$

and $\dfrac{\partial f}{\partial \gamma}$ should have been computed when local iteration converges.

Finally, the tangent stiffness is

$$\frac{\partial \sigma_{n+1}}{\partial \varepsilon_{n+1}} = E + En \frac{\partial \gamma}{\partial \varepsilon_{n+1}} = E + \left(\frac{\partial f}{\partial \gamma}\right)^{-1} E^2. \tag{4.55}$$

4.3.3. Implementation

The state determination algorithm of this AF model is given in Algorithm 4.

Algorithm 4 state determination of uniaxial AF steel model

Parameter: E, K, σ^s, m, a_i, b_i
Input: ε_{n+1}, ε_n, ε_n^p, σ_n, $\alpha_{i,n}$, q_n
Output: E_{n+1}, ε_{n+1}^p, σ_{n+1}, $\alpha_{i,n+1}$, q_{n+1}
compute σ^{trial}, η^{trial} and f^{trial} $n = \text{sign}\left(\sigma^{\text{trial}} - \sum \dfrac{\alpha_{i,n}}{1 + b_i \gamma}\right)$
if $f^{\text{trial}} \geqslant 0$ **then**
    $\gamma = 0$
    while true **do**
        compute f and $\dfrac{\partial f}{\partial \gamma}$ ▷ Eq. (4.49) and Eq. (4.50)

$\Delta\gamma = \left(\frac{\partial f}{\partial \gamma}\right)^{-1} f$

if $|\Delta\gamma| <$ tolerance **then**
break
end if
$\gamma \leftarrow \gamma - \Delta\gamma$
end while
$\varepsilon^p_{n+1} = \varepsilon^p_n + \gamma n$
$q_{n+1} = q_n + \gamma$
$\alpha_{i,n+1} = \frac{\alpha_{i,n} + a_i \gamma n}{1 + b_i \gamma}$
$\sigma_{n+1} = \sigma^{\text{trial}}_{n+1} - E\gamma n$
$E_{n+1} = E + \left(\frac{\partial f}{\partial \gamma}\right)^{-1} E^2$
else
$\varepsilon^p_{n+1} = \varepsilon^p_n$
$q_{n+1} = q_n$
$\alpha_{i,n+1} = \alpha_{i,n}$
$\sigma_{n+1} = \sigma^{\text{trial}}_{n+1}$
$E_{n+1} = E$
end if

So far it is clear that for both simple and complex models, the structure of state determination algorithm remains more or less the same. The core formulation only differs due to different yield function, flow rule and hardening law. Nevertheless, some simplifications are often possible.

```
int ArmstrongFrederick1D::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * incre_strain;

    trial_history = current_history;
    auto& p = trial_history(size);

    auto yield_func = fabs(trial_stress(0) - accu(trial_history.head(size))) - std::max(0.,
    ↪  yield + hardening * p + saturated * (1. - exp(-m * p)));

    if(yield_func < 0.) return SUANPAN_SUCCESS;

    auto gamma = 0.;
    double xi, jacobian;

    unsigned counter = 0;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("ArmstrongFrederick1D cannot converge in %u iterations.\n",
            ↪  max_iteration);
```

```
            return SUANPAN_FAIL;
        }

        const auto exp_term = saturated * exp(-m * p);

        auto k = yield + saturated + hardening * p - exp_term;
        auto dk = hardening + m * exp_term;
        if(k < 0.) k = dk = 0.;

        auto sum_a = 0., sum_b = 0.;
        for(unsigned I = 0; I < size; ++I) {
            const auto denom = 1. + b(I) * gamma;
            sum_a += trial_history(I) / denom;
            sum_b += a(I) / denom;
        }

        yield_func = fabs(xi = trial_stress(0) - sum_a) - (elastic_modulus + sum_b) * gamma -
        ↪  k;

        jacobian = -elastic_modulus - dk;

        if(xi > 0.) for(unsigned I = 0; I < size; ++I) jacobian += (b(I) * trial_history(I) -
        ↪  a(I)) * pow(1. + b(I) * gamma, -2.);
        else for(unsigned I = 0; I < size; ++I) jacobian -= (b(I) * trial_history(I) + a(I))
        ↪  * pow(1. + b(I) * gamma, -2.);

        const auto incre = yield_func / jacobian;
        suanpan_extra_debug("ArmstrongFrederick1D local iterative loop error: %.5E.\n",
        ↪  fabs(incre));
        if(fabs(incre) <= tolerance) break;

        gamma -= incre;
        p -= incre;
    }

    if(xi > 0.) {
        for(unsigned I = 0; I < size; ++I) trial_history(I) = (trial_history(I) + a(I) *
        ↪  gamma) / (1. + b(I) * gamma);

        trial_stress -= elastic_modulus * gamma;
    }
    else {
        for(unsigned I = 0; I < size; ++I) trial_history(I) = (trial_history(I) - a(I) *
        ↪  gamma) / (1. + b(I) * gamma);

        trial_stress += elastic_modulus * gamma;
    }

    trial_stiffness += elastic_modulus / jacobian * elastic_modulus;

    return SUANPAN_SUCCESS;
}
```

4.4. Uniaxial Model for BRB Steel

The previous models show symmetric behaviour regardless under tension or compression. For some application such as buckling restrained braces, buckling under compression would result in lower strength. In this section, a model [7] suitable for BRB steel is presented with a neat implementation.

4.4.1. Theory

To distinguish different responses in tension and compression, one set of flow rule and hardening law is not sufficient. Instead, tension and compression shall be treated separately. Here superscripts $(\cdot)^+$ and $(\cdot)^-$ are used to denote tension and compression governing equations.

Plasticity Activation

The model adopts a different approach. The activation of plasticity is decided based on the product $\sigma\dot{\varepsilon}$. When loading towards tension (compression), as long as σ is in tension (compression), plasticity evolves. In mathematical language,

$$\begin{aligned} &\dot{\varepsilon}\sigma > 0, \quad \text{plasticity develops} \\ &\dot{\varepsilon}\sigma \leqslant 0, \quad \text{elastic unloading} \end{aligned} \tag{4.56}$$

In other words, loading is always plastic while only unloading can be elastic.

Flow Rule

The increment of plastic strain is defined to be a portion of the increment of total strain,

$$\dot{\varepsilon^p} = \begin{cases} \left|\dfrac{\sigma - K\varepsilon^p}{\sigma_y^+}\right|^{\alpha^+} \dot{\varepsilon}, & \text{tension evolution} \\ \left|\dfrac{\sigma - K\varepsilon^p}{\sigma_y^-}\right|^{\alpha^-} \dot{\varepsilon}. & \text{compression evolution} \end{cases} \tag{4.57}$$

The term $K\varepsilon^p$ defines the linear hardening, which serves as a baseline, given than yield stress σ_y^+ (σ_y^-) defines the boundary, the fraction is always smaller than unity.

Hardening Law

The yield stress follows a Voce type rule.

$$\begin{aligned}\sigma_y^+ &= \sigma_0 + \left(\sigma_s^+ - \sigma_0\right)\left(1 - \exp\left(-\frac{q}{n^+}\right)\right) \\ &= \sigma_s^+ - \left(\sigma_s^+ - \sigma_0\right)\exp\left(-\frac{q}{n^+}\right),\end{aligned} \tag{4.58}$$

$$\begin{aligned}\sigma_y^- &= \sigma_0 + \left(\sigma_s^- - \sigma_0\right)\left(1 - \exp\left(-\frac{q}{n^-}\right)\right) \\ &= \sigma_s^- - \left(\sigma_s^- - \sigma_0\right)\exp\left(-\frac{q}{n^-}\right).\end{aligned} \tag{4.59}$$

In which n^+ and n^- are two factors controlling the speed of evolution of yield stress while σ_s^+ and σ_s^- are two saturated stresses. By setting different values, tension response can differ from compression response.

4.4.2. Formulation

The governing equations are summarised as follows.

Constitutive Law	$\sigma = E\left(\varepsilon - \varepsilon^p\right)$
Flow Rule	$\dot{\varepsilon^p} = \begin{cases} \left\lvert\dfrac{\sigma - K\varepsilon^p}{\sigma_y^+}\right\rvert^{\alpha^+}\dot{\varepsilon}, & \text{tension evolution,} \\ \left\lvert\dfrac{\sigma - K\varepsilon^p}{\sigma_y^-}\right\rvert^{\alpha^-}\dot{\varepsilon}, & \text{compression evolution.} \end{cases}$
Hardening Law	$\dot{q} = \lvert\dot{\varepsilon^p}\rvert$
	$\sigma_y^+ = \sigma_s^+ - \left(\sigma_s^+ - \sigma_0\right)\exp\left(-\dfrac{q}{n^+}\right)$
	$\sigma_y^- = \sigma_s^- - \left(\sigma_s^- - \sigma_0\right)\exp\left(-\dfrac{q}{n^-}\right)$

The state determination is based on the flow rule. Rearranging it gives

$$R = \dot{\varepsilon^p} - \left\lvert\frac{\sigma - K\varepsilon^p}{\sigma_y^\pm}\right\rvert^{\alpha^\pm}\dot{\varepsilon} \tag{4.60}$$

with superscript $(\cdot)^\pm$ covering both tension and compression, whichever suits.

It can be further expanded as

$$R = \dot{\varepsilon^p} - \left\lvert\frac{E\left(\varepsilon_{n+1} - \varepsilon_n^p - \dot{\varepsilon^p}\right) - K\varepsilon_n^p - K\dot{\varepsilon^p}}{\sigma_y^\pm}\right\rvert^{\alpha^\pm}\dot{\varepsilon}. \tag{4.61}$$

Denoting $\sigma^{\text{trial}} = E\left(\varepsilon_{n+1} - \varepsilon_n^p\right) - k\varepsilon_n^p$, it is

$$R = \dot{\varepsilon^p} - \left|\frac{\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}}{\sigma_y^{\pm}}\right|^{\alpha^{\pm}} \dot{\varepsilon}. \tag{4.62}$$

The corresponding derivative is

$$\frac{\partial R}{\partial \dot{\varepsilon^p}} = 1 + \alpha^{\pm}\dot{\varepsilon}\left|\frac{\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}}{\sigma_y^{\pm}}\right|^{\alpha^{\pm}} \frac{\left(E+K\right)\sigma_y^{\pm} + \left(\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}\right)\dfrac{\mathrm{d}\sigma_y^{\pm}}{\mathrm{d}\dot{\varepsilon^p}}}{\sigma_y^{\pm}\left(\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}\right)}. \tag{4.63}$$

It can be validated the above derivative holds for both positive and negative fraction.

The derivative of yield function can be computed as

$$\frac{\mathrm{d}\sigma_y^{\pm}}{\mathrm{d}\dot{\varepsilon^p}} = \frac{\sigma_s^{\pm} - \sigma_0}{n^{\pm}} \exp\left(-\frac{q}{n^{\pm}}\right) \operatorname{sign}\left(\dot{\varepsilon^p}\right). \tag{4.64}$$

For tangent stiffness, the following expression will be used.

$$\frac{\partial R}{\partial \varepsilon_{n+1}} = -\left|\frac{\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}}{\sigma_y^{\pm}}\right|^{\alpha^{\pm}} \left(1 + \frac{\alpha^{\pm}\dot{\varepsilon}E}{\sigma^{\text{trial}} - E\dot{\varepsilon^p} - K\dot{\varepsilon^p}}\right). \tag{4.65}$$

4.4.3. Implementation

A clear and concise implementation is presented as follows. Compared with other existing bloated implementations, the following one is significantly simpler.

Algorithm 5 state determination of uniaxial BRB steel model

Parameter: E, K, σ_0, σ_s^+, σ_s^-, n^+, n^-, α^+, α^-
Input: ε_{n+1}, ε_n, ε_n^p, σ_n, q_n
Output: E_{n+1}, ε_{n+1}^p, σ_{n+1}, q_{n+1}
$\dot{\varepsilon} = \varepsilon_{n+1} - \varepsilon_n$
assuming elastic response $\sigma_{n+1} = \sigma_n + E\left(\varepsilon_{n+1} - \varepsilon_n\right)$
if $\sigma_{n+1}\dot{\varepsilon} \leqslant 0$. **then** ▷ elastic
    $\varepsilon_{n+1}^p = \varepsilon_n^p$
    $q_{n+1} = q_n$
    $E_{n+1} = E$
else ▷ plastic
    determine tension/compression plastic loading
    $\dot{\varepsilon^p} = \dfrac{1}{2}\dot{\varepsilon}$
    while true **do**
        $\varepsilon_{n+1}^p = \varepsilon_n^p + \dot{\varepsilon^p}$

$q_{n+1} = q_n + |\dot{\varepsilon^p}|$
$\sigma_{n+1} = E\left(\varepsilon_{n+1} - \varepsilon_{n+1}^p\right)$
compute $\sigma_y^{\pm}$ and $\dfrac{\mathrm{d}\sigma_y^{\pm}}{\mathrm{d}\dot{\varepsilon^p}}$ using the proper function
compute R and $\dfrac{\partial R}{\partial \dot{\varepsilon^p}}$ ▷ Eq. (4.62) and Eq. (4.63)
$\Delta = -\left(\dfrac{\partial R}{\partial \dot{\varepsilon^p}}\right)^{-1} R$
if $|\Delta| <$ tolerance **then**
break
end if
$\dot{\varepsilon^p} \leftarrow \dot{\varepsilon^p} + \Delta$
end while
$E_{n+1} = E + E\left(\dfrac{\partial R}{\partial \dot{\varepsilon^p}}\right)^{-1} \dfrac{\partial R}{\partial \varepsilon_{n+1}}$
end if

The idea of the above steel BRB model is fairly straightforward. Given that the target is to generate different response under tension/compression, one can simply adopt two sets of flow rule and hardening law to fulfil this task. For each case, the overall structure of algorithm remains the same, one can switch between two sets of model parameters.

The CPP implementation of the state determination algorithm is shown as follows.

```cpp
int SteelBRB::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = elastic_modulus) * incre_strain;

    trial_history = current_history;
    const auto& current_accumulated_strain = current_history(0); // u
    const auto& current_plastic_strain = current_history(1);     // \delta_1
    auto& accumulated_strain = trial_history(0);                 // u
    auto& plastic_strain = trial_history(1);                     // \delta_1

    if(trial_stress(0) / incre_strain(0) < 0.) return SUANPAN_SUCCESS;

    const auto tension_flag = incre_strain(0) >= 0.;
    const auto& exponent = tension_flag ? t_exponent : c_exponent;
    const auto compute_stress = tension_flag ? std::mem_fn(&SteelBRB::compute_t_yield_stress)
    ↪   : std::mem_fn(&SteelBRB::compute_c_yield_stress);

    auto incre = .5 * incre_strain(0), incre_plastic_strain = 0.;
    auto counter = 0;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("SteelBRB cannot converge within %u iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

```

```
28          incre_plastic_strain += incre;
29
30          plastic_strain = current_plastic_strain + incre_plastic_strain;
31          trial_stress = elastic_modulus * (trial_strain - plastic_strain);
32
33          const auto sigma_y = compute_stress(this, accumulated_strain =
            ↪  current_accumulated_strain + fabs(incre_plastic_strain));
34          const auto numerator = trial_stress(0) - plastic_modulus * plastic_strain;
35          const auto fraction = numerator / sigma_y(0);
36          const auto pow_term = pow(fabs(fraction), exponent);
37          auto residual = -incre_strain(0) * pow_term;
38          const auto jacobian = 1. + exponent / numerator * residual * (s_modulus - fraction *
            ↪  (incre_plastic_strain >= 0. ? sigma_y(1) : -sigma_y(1)));
39          residual += incre_plastic_strain;
40
41          const auto error = fabs(incre = -residual / jacobian);
42
43          suanpan_debug("SteelBRB local iteration error: %.5E.\n", error);
44
45          if(error <= tolerance) {
46              trial_stiffness *= 1. - (pow_term + incre_strain(0) * elastic_modulus * exponent
                ↪  * pow_term / numerator) / jacobian;
47
48              return SUANPAN_SUCCESS;
49          }
50      }
51  }
```

4.5. VAFCRP1D

Finally, an extension of Armstrong–Fredrick model is presented to close this chapter. A 3D version will be introduced later.

4.5.1. Theory

Yield Function

A von Mises yielding function is used.

$$f = \|\eta\| - k, \tag{4.66}$$

in which $\eta = \sigma - \beta$ is the shifted stress, β is the back stress and $k = k\left(q\right)$ is the isotropic hardening stress.

Flow Rule

The associated plasticity flow is adopted. The plastic strain rate is then

$$\dot{\varepsilon^p} = \gamma \frac{\partial f}{\partial \sigma} = \gamma n = \gamma \text{ sign}(\eta), \tag{4.67}$$

where $n = \dfrac{\eta}{\|\eta\|} = \text{sign}(\eta)$. The corresponding accumulated plastic strain rate is

$$\dot{p} = \|\dot{\varepsilon^p}\| = \gamma. \tag{4.68}$$

Hardening Law

An exponential function with a linear component is used for isotropic hardening stress.

$$k = \sigma_y + k_l p + k_s - k_s e^{-mp}. \tag{4.69}$$

The corresponding derivative is

$$\frac{\mathrm{d}k}{\mathrm{d}\gamma} = k_l + k_s m e^{-mp}. \tag{4.70}$$

The rate form of back stress $\beta = \sum \beta^i$ is defined as

$$\dot{\beta}^i = a^i \dot{\varepsilon^p} - b^i \beta^i \dot{p}.$$

In terms of γ, it is $\dot{\beta}^i = a^i \gamma n - b^i \beta^i \gamma$. The incremental form is thus

$$\beta^i = \beta_n^i + a^i \gamma n - b^i \beta^i \gamma, \qquad \beta^i = \frac{\beta_n^i + a^i \gamma n}{1 + b^i \gamma}. \tag{4.71}$$

Plastic Multiplier

The rate of plastic multiplier is defined as

$$\frac{\gamma}{\Delta t} = \dot{\gamma} = \frac{1}{\mu} \left(\left(\frac{\|\eta\|}{k} \right)^{\frac{1}{\epsilon}} - 1 \right), \tag{4.72}$$

in which μ and ϵ are two material constants. Equivalently, it is

$$\|\eta\| \left(\frac{\Delta t}{\Delta t + \mu \gamma} \right)^{\epsilon} - k = 0. \tag{4.73}$$

4.5.2. Formulation

Incremental Form

The shifted stress can be computed as

$$\eta = \sigma - \beta = E\left(\varepsilon^{\text{trial}} - \varepsilon_n^p - \gamma n\right) - \beta = \sigma^{\text{trial}} - nE\gamma - \sum \frac{\beta_n^i + na^i\gamma}{1 + b^i\gamma} \tag{4.74}$$

with $\sigma^{\text{trial}} = E\left(\varepsilon^{\text{trial}} - \varepsilon_n^p\right)$. Knowing that γ is positive, the following can be obtained by splitting the summation into two parts,

$$\left(\|\eta\| + E\gamma + \sum \frac{a^i\gamma}{1 + b^i\gamma}\right) n = \left\|\sigma^{\text{trial}} - \sum \frac{\beta_n^i}{1 + b^i\gamma}\right\| u,$$

where $u = \dfrac{\sigma^{\text{trial}} - \sum \dfrac{\beta_n^i}{1 + b^i\gamma}}{\left\|\sigma^{\text{trial}} - \sum \dfrac{\beta_n^i}{1 + b^i\gamma}\right\|}$. This expression is equivalent to $n = u$ and

$$\eta = \left(\left\|\sigma^{\text{trial}} - \sum \frac{\beta_n^i}{1 + b^i\gamma}\right\| - E\gamma - \sum \frac{a^i\gamma}{1 + b^i\gamma}\right) u. \tag{4.75}$$

The reason to find such an identity is that u is only a function of γ, the derivative of which can be easily computed. Similar derivations can also be seen in the 3D version, leading to the conclusion that the direction of back stress is aligned with some direction.

The corresponding derivatives are

$$\frac{\partial\|\eta\|}{\partial\varepsilon^{\text{trial}}} = uE, \qquad \frac{\partial\|\eta\|}{\partial\gamma} = \sum \frac{b^i\beta_n^i u - a^i}{(1 + b^i\gamma)^2} - E.$$

Scalar Equation Iteration

With the above expression, it is possible to establish the local residual based on the creep rule, which is

$$R = \left(\left\|\sigma^{\text{trial}} - \sum \frac{\beta_n^i}{1 + b^i\gamma}\right\| - E\gamma - \sum \frac{a^i\gamma}{1 + b^i\gamma}\right)\left(\frac{\Delta t}{\Delta t + \mu\gamma}\right)^{\epsilon} - \left(\sigma_y + k_l\left(p_n + \gamma\right) + k_s\left(1 - e^{-m(p_n+\gamma)}\right)\right). \tag{4.76}$$

The corresponding derivatives are then

$$\frac{\partial R}{\partial \varepsilon^{\text{trial}}} = uE\left(\frac{\Delta t}{\Delta t + \mu\gamma}\right)^{\epsilon}, \tag{4.77}$$

$$\frac{\partial R}{\partial \gamma} = \left(\sum \frac{b^i \beta_n^i u - a^i}{(1+b^i\gamma)^2} - E - \frac{\epsilon\mu\|\eta\|}{\Delta t + \mu\gamma}\right)\left(\frac{\Delta t}{\Delta t + \mu\gamma}\right)^{\epsilon} - \frac{\mathrm{d}k}{\mathrm{d}\gamma}. \tag{4.78}$$

Consistent Tangent Stiffness

For stiffness, $\varepsilon^{\text{trial}}$ is now varying, then

$$\frac{\partial R}{\partial \varepsilon^{\text{trial}}} + \frac{\partial R}{\partial \gamma}\frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon^{\text{trial}}} = 0, \qquad \frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon^{\text{trial}}} = -\left(\frac{\partial R}{\partial \gamma}\right)^{-1}\frac{\partial R}{\partial \varepsilon^{\text{trial}}}. \tag{4.79}$$

Since the stress can be written as

$$\sigma = E(\varepsilon^{\text{trial}} - \varepsilon^p) = E(\varepsilon^{\text{trial}} - \varepsilon_n^p - \Delta\varepsilon^p) = E(\varepsilon^{\text{trial}} - \varepsilon_n^p) - E\gamma u. \tag{4.80}$$

The derivative is

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\varepsilon^{\text{trial}}} = E - Eu\frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon^{\text{trial}}} = E + E^2\left(\frac{\partial R}{\partial \gamma}\right)^{-1}\left(\frac{\Delta t}{\Delta t + \mu\gamma}\right)^{\epsilon}. \tag{4.81}$$

4.5.3. Implementation

```
int VAFCRP1D::update_trial_status(const vec& t_strain) {
    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * (incre_strain =
    ↪  (trial_strain = t_strain) - current_strain);

    trial_history = current_history;
    auto& p = trial_history(size);

    if(fabs(trial_stress(0) - accu(trial_history.head(size))) < std::max(0., yield +
    ↪  hardening * p + saturated * (1. - exp(-m * p)))) return SUANPAN_SUCCESS;

    auto gamma = 0.;
    double xi, jacobian, exp_gamma;

    unsigned counter = 0;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("VAFCRP1D cannot converge in %u iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        const auto exp_term = saturated * exp(-m * p);

        auto k = yield + saturated + hardening * p - exp_term;
```

```
        auto dk = hardening + m * exp_term;
        if(k < 0.) k = dk = 0.;

        auto sum_a = 0., sum_b = 0.;
        for(unsigned I = 0; I < size; ++I) {
            const auto denom = 1. + b(I) * gamma;
            sum_a += trial_history(I) / denom;
            sum_b += a(I) / denom;
        }

        const auto q = fabs(xi = trial_stress(0) - sum_a) - (elastic_modulus + sum_b) *
        ↪  gamma;

        exp_gamma = pow(*incre_time / (*incre_time + mu * gamma), epsilon);

        jacobian = -elastic_modulus - epsilon * mu * q / (*incre_time + mu * gamma);

        if(xi > 0.) for(unsigned I = 0; I < size; ++I) jacobian += (b(I) * trial_history(I) -
        ↪  a(I)) * pow(1. + b(I) * gamma, -2.);
        else for(unsigned I = 0; I < size; ++I) jacobian -= (b(I) * trial_history(I) + a(I))
        ↪  * pow(1. + b(I) * gamma, -2.);

        const auto incre = (q * exp_gamma - k) / ((jacobian *= exp_gamma) -= dk);
        suanpan_extra_debug("VAFCRP1D local iterative loop error: %.5E.\n", fabs(incre));
        if(fabs(incre) <= tolerance) break;

        gamma -= incre;
        p -= incre;
    }

    if(xi > 0.) {
        for(unsigned I = 0; I < size; ++I) trial_history(I) = (trial_history(I) + a(I) *
        ↪  gamma) / (1. + b(I) * gamma);

        trial_stress -= elastic_modulus * gamma;
    }
    else {
        for(unsigned I = 0; I < size; ++I) trial_history(I) = (trial_history(I) - a(I) *
        ↪  gamma) / (1. + b(I) * gamma);

        trial_stress += elastic_modulus * gamma;
    }

    trial_stiffness += elastic_modulus / jacobian * elastic_modulus * exp_gamma;

    return SUANPAN_SUCCESS;
}
```

5. Uniaxial Phenomenological Models

In this chapter, we present a number of uniaxial phenomenological models. Due to their phenomenological nature, some models may violate Ilyushin's postulate.

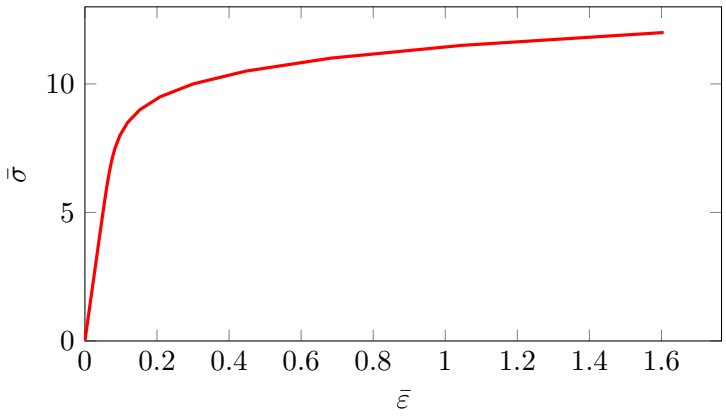
5.1. Ramberg–Osgood Model

5.1.1. Theory

The Ramberg–Osgood relation is nonlinear function that defines

$$\bar{\varepsilon} = \frac{\bar{\sigma}}{E} + \alpha \frac{\bar{\sigma}}{E} \left(\frac{\bar{\sigma}}{\bar{\sigma}^y} \right)^{n-1}, \tag{5.1}$$

where α and n are two model constants. To account for cyclic response, strain and stress are not directly used in the above expression.



5.1.2. Formulation

The residual equation is formulated as

$$R = \bar{\sigma} + \alpha \bar{\sigma} \left(\frac{\bar{\sigma}}{\bar{\sigma}^y} \right)^{n-1} - E\bar{\varepsilon}, \tag{5.2}$$

its derivative with regard to $\bar{\sigma}$ is

$$\frac{\partial R}{\partial \bar{\sigma}} = 1 + n\alpha \left(\frac{\bar{\sigma}}{\bar{\sigma}^y} \right)^{n-1}. \tag{5.3}$$

Once $\bar{\varepsilon}$ and $\bar{\sigma}^y$ are determined, $\bar{\sigma}$ can be iteratively computed.

For cyclic loads, it is necessary to record the location of reversing point, denoted by ε^r and σ^r. Then,

$$\bar{\varepsilon} = |\varepsilon - \varepsilon^r|, \qquad \bar{\sigma} = |\sigma - \sigma^r|. \tag{5.4}$$

It simply means $\bar{\varepsilon}$ and $\bar{\sigma}$ are the absolute strain and stress measured from the reversing point.

The yield stress is taken as

$$\bar{\sigma}^y = \sigma^y + |\sigma^r|. \tag{5.5}$$

where σ^y is the initial yield stress. This definition is fine for large cycle loads, but not sufficient for small cycle loads. For which, the previous reversing point shall be used, let ε^{pr} and σ^{pr} denote its strain and stress, then for small cycles, $|\sigma^{pr}| > |\sigma^r|$,

$$\bar{\sigma}^y = |\sigma^{pr} - \sigma^r|. \tag{5.6}$$

To wrap up,

$$\bar{\sigma}^y = \begin{cases} \max\left(\sigma^y + |\sigma^r|, |\sigma^{pr} - \sigma^r|\right), & \text{if } \sigma^{pr} = 0 \text{ or } |\sigma^{pr}| < |\sigma^r|, \\ |\sigma^{pr} - \sigma^r|, & \text{otherwise.} \end{cases} \tag{5.7}$$

5.1.3. Implementation

The CPP implementation can be found as follows.

```
int RambergOsgood::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;

    trial_history = current_history;
    auto& load_sign = trial_history(0);
    auto& reverse_strain = trial_history(1);
    auto& reverse_stress = trial_history(2);
    auto& p_reverse_strain = trial_history(3);
    auto& p_reverse_stress = trial_history(4);

    if(const auto trial_load_sign = suanpan::sign(incre_strain(0));
    ↪  !suanpan::approx_equal(trial_load_sign, load_sign)) {
```

```
        if(!suanpan::approx_equal(load_sign, 0.)) {
            p_reverse_strain = reverse_strain;
            p_reverse_stress = reverse_stress;
            reverse_strain = current_strain(0);
            reverse_stress = current_stress(0);
        }
        load_sign = trial_load_sign;
    }

    const auto elastic_predictor = elastic_modulus * fabs(trial_strain(0) - reverse_strain);

    const auto norm_yield_stress = std::max(datum::eps, 0. == p_reverse_stress ||
    ↪ fabs(p_reverse_stress) < fabs(reverse_stress) ? std::max(fabs(reverse_stress -
    ↪ p_reverse_stress), yield_stress + fabs(reverse_stress)) : fabs(reverse_stress -
    ↪ p_reverse_stress));

    const auto pow_a = pow(norm_yield_stress, nm);

    auto norm_stress = fabs(current_stress(0) - reverse_stress);

    unsigned counter = 0;
    while(true) {
        const auto pow_b = offset * pow(norm_stress, nm);
        const auto jacobian = pow_a + n * pow_b;
        const auto incre = (norm_stress * (pow_a + pow_b) - elastic_predictor * pow_a) /
        ↪ jacobian;
        const auto error = fabs(incre) / yield_stress;
        suanpan_debug("RambergOsgood local iteraton error: %.5E.\n", error);
        if(error <= tolerance || max_iteration == ++counter) {
            trial_stress = load_sign * norm_stress + reverse_stress;
            trial_stiffness = elastic_modulus * pow_a / jacobian;
            return SUANPAN_SUCCESS;
        }
        norm_stress -= incre;
    }
}
```

5.2. MPF Steel Model

5.2.1. Theory

5.2.2. Formulation

5.2.3. Implementation

The CPP implementation can be found as follows.

```cpp
int MPF::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;

    trial_history = current_history;
    auto& reverse_stress = trial_history(0);
    auto& reverse_strain = trial_history(1);
    auto& inter_stress = trial_history(2);
    auto& inter_strain = trial_history(3);
    auto& pre_inter_strain = trial_history(4);
    auto& max_strain = trial_history(5);
    auto& load_sign = trial_history(6);

    auto shift_stress = 0.;
    if(isotropic_hardening) {
        shift_stress = std::max(0., A3 * yield_stress * (max_strain / yield_strain - A4));
        max_strain = std::max(max_strain, fabs(trial_strain(0)));
    }

    if(const auto trial_load_sign = suanpan::sign(incre_strain(0));
    ↪  !suanpan::approx_equal(trial_load_sign, load_sign)) {
        if(!suanpan::approx_equal(load_sign, 0.)) {
            reverse_stress = current_stress(0);
            reverse_strain = current_strain(0);
            pre_inter_strain = inter_strain;
            inter_strain = yield_stress * hardening_ratio - yield_stress - shift_stress;
            if(trial_load_sign > 0.) inter_strain = -inter_strain;
            inter_strain = (inter_strain + elastic_modulus * reverse_strain - reverse_stress)
            ↪  / (elastic_modulus - hardening_ratio * elastic_modulus);
            inter_stress = elastic_modulus * (inter_strain - reverse_strain) +
            ↪  reverse_stress;
        }
        else if(trial_load_sign > 0.) {
            inter_stress = yield_stress;
            inter_strain = yield_strain;
        }
        else {
            inter_stress = -yield_stress;
            inter_strain = -yield_strain;
        }
        load_sign = trial_load_sign;
    }

    auto radius = R0;
    if(!constant_radius) {
        // update radius
        const auto xi = fabs(reverse_strain - pre_inter_strain) / yield_strain;
        radius -= A1 * xi / (A2 + xi);
    }

    const auto gap_strain = inter_strain - reverse_strain;
    const auto gap_stress = inter_stress - reverse_stress;
    const auto normal_strain = std::max(datum::eps, (trial_strain(0) - reverse_strain) /
    ↪  gap_strain);
    const auto factor_a = 1. + pow(normal_strain, radius);
    const auto factor_b = (1. - hardening_ratio) * pow(factor_a, -1. / radius);
```

```
54
55      trial_stress = (hardening_ratio + factor_b) * normal_strain * gap_stress +
        ↪  reverse_stress;
56      trial_stiffness = gap_stress / gap_strain * (hardening_ratio + factor_b / factor_a);
57
58      return SUANPAN_SUCCESS;
59  }
```

5.3. Bouc–Wen Model

5.3.1. Theory

5.3.2. Formulation

5.3.3. Implementation

```
1   int BoucWen::update_trial_status(const vec& t_strain) {
2       incre_strain = (trial_strain = t_strain) - current_strain;
3
4       if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;
5
6       const auto n_strain = incre_strain(0) / yield_strain;
7
8       trial_history = current_history;
9       const auto& current_z = current_history(0); // z
10      auto& z = trial_history(0);                 // z
11
12      auto incre = .5 * n_strain;
13      unsigned counter = 0;
14      while(true) {
15          if(max_iteration == ++counter) {
16              suanpan_error("BoucWen cannot converge within %u iterations.\n", max_iteration);
17              return SUANPAN_FAIL;
18          }
19
20          z += incre;
21
22          const auto p_term = (gamma + (z * n_strain >= 0. ? beta : -beta)) *
            ↪  pow(std::max(datum::eps, fabs(z)), n);
23          const auto t_term = n_strain * p_term;
24
25          const auto residual = z - current_z + t_term - n_strain;
26          const auto jacobian = z + n * t_term;
27
28          const auto error = fabs(incre = -residual * z / jacobian);
29
```

```
30          suanpan_debug("BoucWen local iteration error: %.5E.\n", error);
31
32          if(error <= tolerance) {
33              trial_stress = modulus_a * trial_strain + modulus_b * z;
34              trial_stiffness = modulus_a + modulus_b / yield_strain * (1. - p_term) * z /
                ↪  jacobian;
35
36              return SUANPAN_SUCCESS;
37          }
38      }
39  }
```

5.4. General Framework for Hysteresis Models

5.4.1. Theory

The general framework consists of three main elements:

1. backbone curve — compression/tension envelope, blue curves in Fig. 5.1
2. unloading curve — unload from backbone to the corresponding residual point (with zero stress), green curves in Fig. 5.1
3. reloading curve — loading from residual point to backbone curve on the opposite side, purple curves in Fig. 5.1

Each of those three curves can be either linear/nonlinear or piece-wise linear/nonlinear and may depend on other internal variables.

Accordingly, at least four points are essential to control the response of the model, namely, the tension unloading point, the tension residual point, the compression unloading point, the compression residual point. A schematic illustration is given in Fig. 5.1. Some complex models may consist of more control points, such as reloading points that connect reloading curves with backbones.

With the above definition, it is clear that any given current state ε_n and σ_n shall on one of three curves. With prescribed strain increment $\Delta\varepsilon$, the determination of stress σ_{n+1} is equivalent to computing the new point on one of three curves. As the current state $(\varepsilon_n, \sigma_n)$ has to be on one of three curves, it is easy to determine whether $\varepsilon_{n+1} = \varepsilon_n + \Delta\varepsilon$ is on unloading/reloading/backbone curve by simply comparing the magnitudes of ε_{n+1} and that of unloading/residual strain, which are stored as history variables.

The state determination algorithm can be cast in a branching programming style. In the

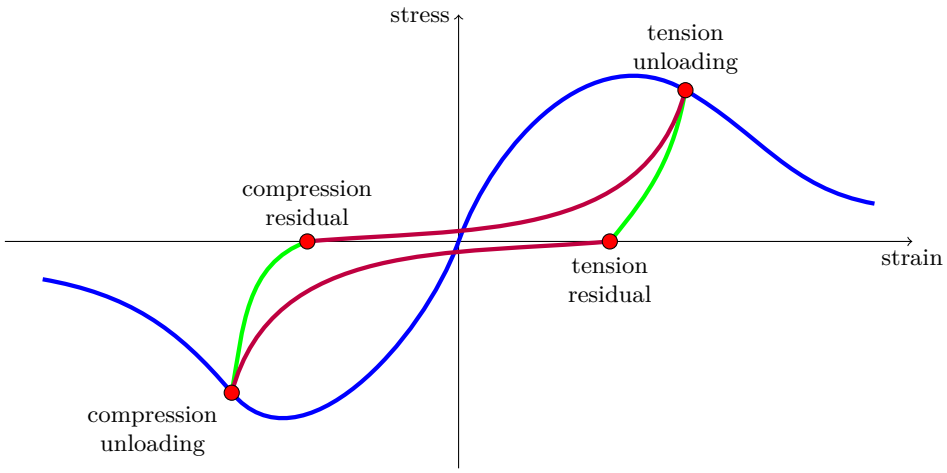


Figure 5.1.: schematic illustration a generalised hysteresis model

following procedure, ε_{cr} is compression residual strain, ε_{cu} is compression unloading strain, ε_{tr} is tension residual strain, ε_{tu} is tension unloading strain.

Algorithm 6 state determination of general hysteresis model

1: **Parameter**: necessary model parameters
2: **Input**: ε_{n+1}, ε_n, σ_n and other relevant history variables
3: **Output**: E_{n+1}, σ_{n+1} and other relevant history variables
4: get strain values of four control points
5: $\varepsilon_{n+1} = \varepsilon_n + \Delta\varepsilon$
6: determine which curve the new state is on based on the curve the current state is on and the magnitudes of ε_{n+1}, ε_{cr}, ε_{cu}, ε_{tr}, ε_{tu}
7: call corresponding methods to compute the new state $(\varepsilon_{n+1}, \sigma_{n+1})$ based on relevant two control points and history variables, if any

5.4.2. Implementation

By using two flags, it is easy to track which curve each point is on.

```
1    enum class Status { NONE, CBACKBONE, TBACKBONE, CINNER, TINNER };
2
3    Status trial_flag = Status::NONE, current_flag = Status::NONE;
```

The implementation presented uses a nested structure to simply the procedure. The top level determines whether the new state is on the backbone, if not, the second level determines whether the new state is on the corresponding unloading curve or reloading curve and computes the response accordingly.

For each curve, a universal interface can be provided such that each takes ε_{n+1} as the input and returns σ_{n+1} and E_{n+1} in an array.

```
    [[nodiscard]] virtual podarray<double> compute_compression_backbone(double) const = 0;
    [[nodiscard]] virtual podarray<double> compute_tension_backbone(double) const = 0;
    [[nodiscard]] podarray<double> compute_compression_inner(double) const;
    [[nodiscard]] podarray<double> compute_tension_inner(double) const;
```

The control points can be updated on the demand with some auxiliary methods.

```
    [[nodiscard]] virtual podarray<double> compute_compression_initial_reverse() const = 0;
    [[nodiscard]] virtual podarray<double> compute_tension_initial_reverse() const = 0;
    [[nodiscard]] virtual double compute_compression_residual(double, double) const = 0;
    [[nodiscard]] virtual double compute_tension_residual(double, double) const = 0;
```

The following CPP code snippet shows a working implementation of such a state determination procedure.

```
int SimpleHysteresis::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;

    if(current_history.is_empty() || !any(current_history)) {
        current_history.zeros(8);
        auto point = compute_compression_initial_reverse();
        current_history(2) = point(0);
        current_history(3) = point(1);
        point = compute_tension_initial_reverse();
        current_history(4) = point(0);
        current_history(5) = point(1);
        initial_history = current_history;
    }
    else if(current_history.size() != 8) {
        current_history.resize(8);
        initial_history = current_history;
    }

    trial_history = current_history;
    auto& max_c_strain = trial_history(0);      // maximum compression strain recorded
    auto& max_t_strain = trial_history(1);      // maximum tension strain recorded
    auto& reverse_c_strain = trial_history(2);  // unloading point strain compression side
    auto& reverse_c_stress = trial_history(3);  // unloading point stress compression side
    auto& reverse_t_strain = trial_history(4);  // unloading point strain tension side
    auto& reverse_t_stress = trial_history(5);  // unloading point stress tension side
    auto& residual_c_strain = trial_history(6); // residual strain in compression unloading
    ↪ path
    auto& residual_t_strain = trial_history(7); // residual strain in compression unloading
    ↪ path

    podarray<double> response;
```

```
32
33      if(Status::NONE == current_flag) {
34          if(incre_strain(0) > 0.) {
35              trial_flag = Status::TBACKBONE;
36              response = compute_tension_backbone(max_t_strain = trial_strain(0));
37          }
38          else {
39              trial_flag = Status::CBACKBONE;
40              response = compute_compression_backbone(max_c_strain = trial_strain(0));
41          }
42      }
43      else if(Status::CBACKBONE == current_flag) {
44          if(incre_strain(0) > 0.) {
45              residual_c_strain = compute_compression_residual(reverse_c_strain =
                ↪  current_strain(0), reverse_c_stress = current_stress(0));
46              if(trial_strain(0) > reverse_t_strain) {
47                  trial_flag = Status::TBACKBONE;
48                  response = compute_tension_backbone(max_t_strain = trial_strain(0));
49              }
50              else {
51                  trial_flag = Status::CINNER;
52                  response = compute_compression_inner(trial_strain(0));
53              }
54          }
55          else {
56              trial_flag = Status::CBACKBONE;
57              response = compute_compression_backbone(max_c_strain = trial_strain(0));
58          }
59      }
60      else if(Status::TBACKBONE == current_flag) {
61          if(incre_strain(0) < 0.) {
62              residual_t_strain = compute_tension_residual(reverse_t_strain =
                ↪  current_strain(0), reverse_t_stress = current_stress(0));
63              if(trial_strain(0) < reverse_c_strain) {
64                  trial_flag = Status::CBACKBONE;
65                  response = compute_compression_backbone(max_c_strain = trial_strain(0));
66              }
67              else {
68                  trial_flag = Status::TINNER;
69                  response = compute_tension_inner(trial_strain(0));
70              }
71          }
72          else {
73              trial_flag = Status::TBACKBONE;
74              response = compute_tension_backbone(max_t_strain = trial_strain(0));
75          }
76      }
77      else if(Status::CINNER == current_flag) {
78          if(trial_strain(0) > reverse_t_strain) {
79              trial_flag = Status::TBACKBONE;
80              response = compute_tension_backbone(max_t_strain = trial_strain(0));
81          }
82          else if(trial_strain(0) < reverse_c_strain) {
83              trial_flag = Status::CBACKBONE;
84              response = compute_compression_backbone(max_c_strain = trial_strain(0));
85          }
86          else {
```

```
            trial_flag = Status::CINNER;
            response = compute_compression_inner(trial_strain(0));
        }
    }
    else if(Status::TINNER == current_flag) {
        if(trial_strain(0) > reverse_t_strain) {
            trial_flag = Status::TBACKBONE;
            response = compute_tension_backbone(max_t_strain = trial_strain(0));
        }
        else if(trial_strain(0) < reverse_c_strain) {
            trial_flag = Status::CBACKBONE;
            response = compute_compression_backbone(max_c_strain = trial_strain(0));
        }
        else {
            trial_flag = Status::TINNER;
            response = compute_tension_inner(trial_strain(0));
        }
    }

    trial_stress = response(0);
    trial_stiffness = response(1);

    suanpan_debug([&]() { if(!trial_stress.is_finite() || !trial_stiffness.is_finite()) throw
    ↪  invalid_argument("infinite number detected.\n"); });

    return SUANPAN_SUCCESS;
}
```

Remarks

A lot of hysteresis models can be reformulated in the above framework. For example, by fixing residual points to origin, all unloading paths (either linear or nonlinear) converge to origin, which correspond to a class of models that are often called origin–oriented. Alternatively, the reloading curve can be defined in a way so that it always loads back to the peak point on the backbone curve on the opposite side. This is often known as peak–oriented model.

Some complex models suitable for concrete can be formulated accordingly, in which the curves are nonlinear and depend on other internal variables as well. Given that tension response and compression response are independent from each other, some asymmetric models can also be defined.

Noting that the top level branching does not care about history variables, the updating of them are solely handled in the corresponding methods. The presented framework provides a flexible container so that a wide range of models can be defined.

6. Uniaxial Plasticity Models (Other Materials)

6.1. K4 Concrete

In this section, we introduce a uniaxial model for concrete that is formulated within the classic plasticity framework. The model is named as K4 model in this book as the four authors' surnames all start with letter 'K'. However, the model presented in the original paper adopts a linear degradation, leading to inevitable sudden zero stiffness after exceeding certain strain limit. Numerical models do not prefer exact zeros, thus, it is revised to adopt an exponentially decaying function instead. It shall be pointed out that the original formulation [8] contains some mistakes, in the event of any discrepancy or inconsistency, we prefer the formulation discussed in this section.

There are quite a number of phenomenological models for concrete. Since they are phenomenological, it is relatively flexible to define different loading/unloading response. Such kind of flexibility makes the corresponding model easier for engineers to understand and extrapolate, however, they may violate thermodynamic principles.

6.1.1. Theory

The classic plasticity framework does not support definition of stiffness degradation, the elastic loading/unloading paths always follow the gradient of initial stiffness. It is a common practice to combine plasticity and damage [9] together so that the apparent stiffness can gradually degrade.

The apparent stress σ is conventionally expressed as

$$\sigma = (1 - d)\,\bar{\sigma}, \tag{6.1}$$

where d is the damage factor ranging from zero to unity and $\bar{\sigma}$ is the effective stress. With such a formulation, damage and plasticity apply to d and $\bar{\sigma}$ respectively.

Just like other plasticity models, the additive decomposition applies such that

$$\bar{\sigma} = E\,(\varepsilon - \varepsilon^p)\,. \tag{6.2}$$

Yield Function

Unlike metals, there is no need to model ratcheting in concrete, as a result, conventionally there is only isotropic hardening, but no kinematic hardening. Thus, there is also no need to introduce back stress into the model. The yield function F can be simply expressed as

$$F = |\bar{\sigma}| - \bar{\sigma}_y. \tag{6.3}$$

It states that plasticity evolution will be triggered whenever the magnitude of effective stress $\bar{\sigma}$ exceeds the backbone defined by $\bar{\sigma}_y$, which is a function of some internal hardening variable.

It becomes apparent that if only one $\bar{\sigma}_y$ is used, there is no way to differentiate tensile behaviour from compressive one. This issue can be addressed by adopting two sets of rules, one for tension and the other for compression.

$$F = \begin{cases} F_t, & \bar{\sigma} > 0, \\ F_c, & \text{otherwise}, \end{cases} \tag{6.4}$$

with

$$F_t = \bar{\sigma} - \bar{\sigma}_{y,t}, \qquad F_c = -\bar{\sigma} - \bar{\sigma}_{y,c}. \tag{6.5}$$

Flow Rule

For uniaxial models, the flow direction is simply the loading direction,

$$\dot{\varepsilon^p} = \gamma \text{ sign}\left(\bar{\sigma}\right) = \gamma \frac{\partial F}{\partial \bar{\sigma}}. \tag{6.6}$$

One would find this coincide with the associative plasticity (the second equality).

Hardening Law

Any functions can be chosen as the hardening law, meaning that

$$\bar{\sigma}_{y,t} = H_t\left(k_t\right), \qquad \bar{\sigma}_{y,c} = H_c\left(k_c\right), \tag{6.7}$$

where k_t and k_c are internal hardening variables.

Given that the damage theory is adopted to account for stiffness and strength degradation, often monotonically increasing functions are chosen for hardening functions.

The original formulation adopts the following explicit forms.

$$\bar{\sigma}_{y,t} = f_t + h_t k_t, \qquad \bar{\sigma}_{y,c} = \begin{cases} f_y + h_c k_c, & k_c \leqslant k_0, \\ f_y + h_c k_0 + h_d \left(k_c - k_0\right), & \text{otherwise.} \end{cases} \tag{6.8}$$

In the above expression, f_t and f_y are crack strength (for tension, in positive) and yield strength (for compression, in positive), h_t, h_c and h_d are three positive moduli, and k_0 denotes the value of k_c at compressive crush strength. Essentially, $\bar{\sigma}_{y,t}$ is a linear hardening function while $\bar{\sigma}_{y,c}$ is a bilinear hardening function. The above expression can be further simplified if we let f_c denote $f_y + h_c k_0$.

Since stiffness and strength degradation is taken care of by the damage part, here no softening is considered such that

$$h_t, h_c, h_d \geqslant 0. \tag{6.9}$$

The evolutions of k_t and k_c are tied to the plastic strain.

$$\dot{k}_t = |\dot{\varepsilon^p}| = \gamma \qquad \text{for tension loading,} \tag{6.10}$$

$$\dot{k}_c = |\dot{\varepsilon^p}| = \gamma \qquad \text{for compression loading.} \tag{6.11}$$

Although the same expression is used for both, it shall be noted that for one single loading step, only one of two yield functions will be activated, hence the plastic strain increment γ (of that particular step) can only contribute to either k_t or k_c, not both.

Degradation

With the above yield function, flow rule and hardening law, we effectively have defined a very simple plasticity model with only isotropic hardening which may have different behaviour in tension and compression. It bears some resemblance compared with the model introduced in § 4.4, which adopts the same concept to differentiate tension and compression but is more complex in terms of the specific functions chosen.

Similarly, it is desired to control degradation rate independently for tension and compression, to this end,

$$d = \begin{cases} d_t, & \bar{\sigma} > 0, \\ d_c, & \text{otherwise,} \end{cases} \tag{6.12}$$

The specific forms of tensile damage d_t and compressive damage d_c could vary, for example, the original formulation [8] uses the linear degradation [10]. Here, we use a simpler exponential function.

$$d_t = 1 - \exp\left(-\frac{k_t}{e_{r,t}}\right), \qquad d_c = 1 - \exp\left(-\frac{k_c}{e_{r,c}}\right). \tag{6.13}$$

The parameters $e_{r,t}$ and $e_{r,c}$ are introduced to control the degradation rate.

For compression, if the original bilinear hardening function Eq. (6.8) is used, it is desired to achieve the crush strength, thus, the damage evolution shall be delayed. The following function

$$d_c = 1 - \exp\left(-\frac{\max\left(k_c - k_0, 0\right)}{e_{r,c}}\right)$$

can be used such that d_c stays at zero until k_c becomes larger than k_0.

Total Energy If linear hardening is used, say for example,

$$\bar{\sigma}_y = ak + b. \tag{6.14}$$

Here we do not distinguish between tension and compression, and use general parameters a and b. Combining $\bar{\sigma}_y$ with damage factor gives the final stress backbone as

$$\sigma_y = (1 - d)\,\bar{\sigma}_y = \exp\left(-\frac{k}{c}\right)(ak + b)\,. \tag{6.15}$$

Integration leads to

$$\int_0^\infty \sigma_y \,\mathrm{d}k = ac^2 + bc. \tag{6.16}$$

The initial slope is

$$\left.\frac{\mathrm{d}\sigma_y}{\mathrm{d}k}\right|_{k=0} = a - \frac{b}{c}. \tag{6.17}$$

For it to be smaller than zero (no initial hardening, softening only), it is required that

$$ac < b. \tag{6.18}$$

If $a = 0$, then no matter what value c takes, there is no initial hardening since $b > 0$. If $ac \geqslant b$, implying $a > 0$, the backbone has a peak value at $k = c - \dfrac{b}{a}$.

It is possible to associate c with b such that $c = \dfrac{b}{\zeta E}$ where ζ is a positive multiplier, then the total energy becomes

$$a\frac{b^2}{\zeta^2 E^2} + b\frac{b}{\zeta E} = \left(\frac{a}{\zeta^2 E^2} + \frac{1}{\zeta E}\right) b^2. \tag{6.19}$$

The peak value for $\zeta E < a$ is

$$\frac{ab}{\zeta E}\exp\left(\frac{\zeta E - a}{a}\right). \tag{6.20}$$

If, for the reference characteristic length l_r, the reference ζ_r is given, then for any characteristic length l, if the total energy needs to stay unchanged, then

$$\left(\frac{a}{\zeta^2 E^2} + \frac{1}{\zeta E}\right) b^2 l = \left(\frac{a}{\zeta_r^2 E^2} + \frac{1}{\zeta_r E}\right) b^2 l_r, \tag{6.21}$$

$$\frac{a}{\zeta^2 E^2} + \frac{1}{\zeta E} = \frac{l_r}{l}\left(\frac{a}{\zeta_r^2 E^2} + \frac{1}{\zeta_r E}\right), \tag{6.22}$$

$$\frac{1}{\zeta E} = \frac{\sqrt{1 + 4\frac{l_r}{l}\left(\frac{a^2}{\zeta_r^2 E^2} + \frac{a}{\zeta_r E}\right)} - 1}{2a}. \tag{6.23}$$

Denoting $r_a = \dfrac{a}{E}$,

$$\zeta = \frac{2r_a}{\sqrt{1 + 4\frac{l_r}{l}\left(\frac{r_a^2}{\zeta_r^2} + \frac{r_a}{\zeta_r}\right)} - 1}. \tag{6.24}$$

Note here we only enforce the total energy to be constant. It is not equivalent to objective results under arbitrary magnitudes of loading.

Crack/Gap Closing

According to Eq. (6.10), k_t is the accumulated tensile plastic strain, physically, it represents the crack opening. When reloading towards compression, this opening can be gradually closed.

To account for this crack closing mechanism, one more history variable k_k is introduced to allow additional plasticity to develop. k_k tracks crack closing.

For this inner plasticity model, we adopt a simple linear model such that a fixed fraction of strain increment contributes to plastic strain, that is

$$\Delta\varepsilon^p = \Delta\varepsilon\frac{E}{E + h_k}. \tag{6.25}$$

Knowing that this is only activated during compressive loading ($\bar{\sigma} < 0$ and $\Delta\varepsilon < 0$), then

$\Delta\varepsilon^p < 0$ and $|\Delta\varepsilon^p| = -\Delta\varepsilon^p$.

$$\Delta k_k = -\Delta\varepsilon^p, \tag{6.26}$$
$$\Delta\bar{\sigma} = -E\Delta\varepsilon^p. \tag{6.27}$$

Such a mechanism states that, for reloading towards compression ($\bar{\sigma} < 0$ and $\Delta\varepsilon < 0$), whenever $k_k < k_t$, viz., crack closing is smaller than crack opening, there is net crack opening needs to be closed. Under such a condition, the inner plasticity is activated such that part of strain increment is converted to plastic strain (cracking closing). The following two implementation details need to be considered to ensure a correct model.

Entering It shall be pointed out that the exact transition step from tensile stress to compressive stress needs special treatment. The tensile stress needs to fully unload to zero, this part involves no plasticity, the remaining is compressive loading which involves plasticity. Thus $\Delta\varepsilon^p$ should be limited such that

$$\Delta\varepsilon^p = \left(\Delta\varepsilon + \frac{\max\left(\bar{\sigma}_n, 0\right)}{E}\right)\frac{E}{E+h_k} = \Delta\varepsilon\frac{E}{E+h_k} + \frac{\max\left(\bar{\sigma}_n, 0\right)}{E+h_k}. \tag{6.28}$$

The above expression states that if the current effective stress $\bar{\sigma}_n$ is positive (tensile), then we remove the corresponding tensile strain ($\frac{\max\left(\bar{\sigma}_n, 0\right)}{E}$) from the total strain increment $\Delta\varepsilon$. The resulting $\Delta\varepsilon^p$ would be fully induced by compressive part.

Exiting Similarly, the exact transition step from not yet closed to fully closed also needs additional attention. Noting that k_k shall never exceed k_t, and during the evolution of k_k, k_t does not change, then

$$k_k = k_{k,n} + \Delta k_k = k_{k,n} - \Delta\varepsilon^p \leqslant k_t, \tag{6.29}$$

equivalently,

$$k_{k,n} - k_t \leqslant \Delta\varepsilon^p. \tag{6.30}$$

If the above condition is not satisfied by the plastic strain increment computed from Eq. (6.28), then one shall manually set $\Delta\varepsilon^p = k_{k,n} - k_t$ and update k_k and $\bar{\sigma}$ accordingly.

Remark For crack closing, the original model [8] formulates it as a plasticity model but does not use a returning mapping algorithm in the corresponding implementation. Characterising such a mechanism using a yield function, such as

$$F_k = |\bar{\sigma}| - \bar{\sigma}_k$$

with $\bar{\sigma}_k = h_k k_k$, does not properly define the behaviour. Especially under cyclic loading, the elastic region grows as $\bar{\sigma}_k$ would grow.

To correct the response, it, at least, has to be defined as

$$\bar{\sigma}_k = h_k \left(k_k - k_k^i \right), \tag{6.31}$$

where k_k^i is the initial k_k at the beginning of current loading path, and needs to be updated whenever load reversal occurs.

We avoid such a complex presentation as it may cause unnecessary confusions. At its core, the crack closing mechanism tries to close the unclosed crack $(k_t - k_k)$ by accumulating a portion of strain increment to k_k ($\Delta k_k = |\Delta\varepsilon| \frac{E}{E + h_k}$), and this mechanism is conditionally activated.

6.1.2. Formulation

Tangent Stiffness

By the chain rule, differentiating the expression of stress

$$\sigma = (1 - d)\, \bar{\sigma} \tag{6.32}$$

leads to

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\varepsilon} = (1 - d) \frac{\mathrm{d}\bar{\sigma}}{\mathrm{d}\varepsilon} - \bar{\sigma} \frac{\mathrm{d}d}{\mathrm{d}k} \frac{\mathrm{d}k}{\mathrm{d}\varepsilon}. \tag{6.33}$$

Here we use an 'anisotropic' damage concept, that is

$$d = \begin{cases} d_t, & \bar{\sigma} > 0, \\ d_c, & \text{else}, \end{cases} \qquad k = \begin{cases} k_t, & \bar{\sigma} > 0, \\ k_c, & \text{else}. \end{cases} \tag{6.34}$$

Knowing that $\frac{\mathrm{d}k}{\mathrm{d}\varepsilon}$ is effectively $\frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon}$ as $\dot{k} = \gamma$, from the local residual (yield function) at equilibrium,

$$\frac{\partial F}{\partial \varepsilon} = \frac{\partial F}{\partial \gamma} \frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon} + \frac{\partial F}{\partial \varepsilon} = 0, \tag{6.35}$$

one could derive

$$\frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon} = -\left(\frac{\partial F}{\partial \gamma} \right)^{-1} \frac{\partial F}{\partial \varepsilon}. \tag{6.36}$$

The yield function can be expressed as

$$\begin{aligned} F &= |\bar{\sigma}| - \bar{\sigma}_y \\ &= \left|\bar{\sigma}^{\text{trial}} - \gamma E \text{sign}\left(\bar{\sigma}\right)\right| - \bar{\sigma}_y, \end{aligned} \tag{6.37}$$

then

$$\frac{\partial F}{\partial \gamma} = -\text{sign}\left(\bar{\sigma}\right) E \text{sign}\left(\bar{\sigma}\right) - \frac{\mathrm{d}\bar{\sigma}_y}{\mathrm{d}k} = -E - \frac{\mathrm{d}\bar{\sigma}_y}{\mathrm{d}k}, \tag{6.38}$$

$$\frac{\partial F}{\partial \varepsilon} = \text{sign}\left(\bar{\sigma}\right) E. \tag{6.39}$$

In the above, $J = \dfrac{\partial F}{\partial \gamma}$ is in fact the Jacobian of the local system. Thus,

$$\frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon} = -\text{sign}\left(\bar{\sigma}\right) \frac{E}{J}. \tag{6.40}$$

For the effective stress,

$$\frac{\mathrm{d}\bar{\sigma}}{\mathrm{d}\varepsilon} = E - \text{sign}\left(\bar{\sigma}\right) E \frac{\mathrm{d}\gamma}{\mathrm{d}\varepsilon} = E + \frac{E^2}{J}. \tag{6.41}$$

The stiffness can be expressed as

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\varepsilon} = (1-d)\left(E + \frac{E^2}{J}\right) + |\bar{\sigma}| \frac{\mathrm{d}d}{\mathrm{d}k} \frac{E}{J}. \tag{6.42}$$

The explicit form of $\dfrac{\mathrm{d}d}{\mathrm{d}k}$ shall be determined by the damage evolution.

6.1.3. Implementation

Algorithm 7 state determination of K4 concrete model

```
compute trial stress
if crack closing is activated then
    apply crack closing plasticity
end if
if tension then
    apply tensile plasticity
    apply tensile damage
else
    apply compressive plasticity
    apply compressive damage
end if
```

It is straightforward to translate the above pseudo code to implementation.

```
int NonlinearK4::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(fabs(incre_strain(0)) <= datum::eps) return SUANPAN_SUCCESS;

    trial_history = current_history;
    const auto& plastic_strain = trial_history(0);
    // auto& kt = trial_history(1);
    // auto& kc = trial_history(2);
    const auto& current_kt = current_history(1);
    const auto& current_kk = current_history(3);

    trial_stress = (trial_stiffness = elastic_modulus) * (trial_strain - plastic_strain);

    if(apply_crack_closing && trial_stress(0) < 0. && incre_strain(0) < 0. && current_kt >
    ↪  current_kk) compute_crack_close_branch();

    return compute_plasticity();
}
```

The crack closing accounts for the aforementioned entering/exiting details.

```
void NonlinearK4::compute_crack_close_branch() {
    auto& plastic_strain = trial_history(0);
    const auto& kt = trial_history(1);
    auto& kk = trial_history(3);

    const auto jacobian = elastic_modulus + hardening_k;

    // account for entering
    const auto net_strain = fabs(incre_strain(0)) - std::max(0., current_stress(0)) /
    ↪  elastic_modulus;
    const auto dgamma = elastic_modulus / jacobian;
    auto incre = net_strain * dgamma;

    // physically, the tension plastic strain is the crack opening, closing the crack should
    ↪  not exceed the opening
    // ensure the crack plastic strain is bounded by the tension plastic strain
    if(incre > kt - kk) incre = kt - kk;
    else trial_stiffness -= dgamma * elastic_modulus; // otherwise, the stiffness is degraded
    ↪  during the closing phase

    const auto incre_ep = incre * suanpan::sign(trial_stress(0));

    kk += incre;
    plastic_strain += incre_ep;
    trial_stress -= elastic_modulus * incre_ep;
}
```

The tension and compression return mapping algorithms are almost identical except that they update different history variables and call different functions to compute backbone and damage. One can unify them and use one implementation for both.

```
int NonlinearK4::compute_plasticity() {
    auto& plastic_strain = trial_history(0);

    const auto sign_sigma = suanpan::sign(trial_stress(0));

    auto& k = sign_sigma > 0. ? trial_history(1) : trial_history(2);

    const auto backbone_handle = sign_sigma > 0. ?
    ↪  std::mem_fn(&NonlinearK4::compute_tension_backbone) :
    ↪  std::mem_fn(&NonlinearK4::compute_compression_backbone);
    const auto damage_handle = sign_sigma > 0. ?
    ↪  std::mem_fn(&NonlinearK4::compute_tension_damage) :
    ↪  std::mem_fn(&NonlinearK4::compute_compression_damage);

    auto counter = 0u;
    auto ref_error = 1.;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("Cannot converge within {} iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        const auto backbone = backbone_handle(this, k);
        const auto residual = fabs(trial_stress(0)) - backbone(0);

        if(1u == counter && residual <= 0.) {
            if(apply_damage) {
                const auto damage = damage_handle(this, k);
                const auto damage_factor = 1. - damage(0);

                trial_stress *= damage_factor;
                trial_stiffness *= damage_factor;
            }

            return SUANPAN_SUCCESS;
        }

        const auto jacobian = elastic_modulus + backbone(1);
        const auto incre = residual / jacobian;
        const auto error = fabs(incre);
        if(1u == counter) ref_error = error;
        suanpan_debug("Local plasticity iteration error: {:.5E}.\n", error);
        if(error < tolerance * ref_error || (fabs(residual) < tolerance && counter > 5u)) {
            const auto dgamma = elastic_modulus / jacobian;
            trial_stiffness -= dgamma * elastic_modulus;

            if(apply_damage) {
                const auto damage = damage_handle(this, k);
                const auto damage_factor = 1. - damage(0);

                trial_stiffness *= damage_factor;
                trial_stiffness -= trial_stress * damage(1) * sign_sigma * dgamma;
```

```

                trial_stress *= damage_factor;
            }

            return SUANPAN_SUCCESS;
        }

        const auto incre_ep = incre * sign_sigma;

        k += incre;
        plastic_strain += incre_ep;
        trial_stress -= elastic_modulus * incre_ep;
    }
}
```

In the computation of damage variables, it is possible to adjust ζ according to the previous discussion.

```
double NonlinearK4::objective_scale(const double a, const double zeta) const {
    if(!objective_damage) return zeta;

    const auto ratio = a / zeta;
    return 2. * a / (std::sqrt(1. + 4. / get_characteristic_length() * (ratio * ratio +
    ↪  ratio)) - 1.);
}

vec2 ConcreteK4::compute_tension_damage(const double k) const {
    const auto e_t = f_t / objective_scale(hardening_t, zeta_t);
    const auto factor = exp(-k / e_t);
    return vec2{1. - factor, factor / e_t};
}
```

Part III.

2D/3D Models

7. Metal

In this chapter, several frameworks suitable for developing metal models are introduced. The basic one is the von Mises framework, which is also called J2 model in some literature as it adopts the second invariant of the deviatoric stress to characterise yield function. The intermediate one is the VAFCRP model, in which a Voce type nonlinear isotropic hardening, a multiplicative Armstrong–Fredrick type kinematic hardening and a Peric type viscosity are implemented. Thus, this model can account for dynamic effects. The third model is a general framework developed based on the Hoffman criterion, it is suitable for orthotropic materials.

7.1. von Mises Framework

Here the uniaxial combined isotropic/kinematic model introduced in § 4.2 is reformulated in 3D space. Some difference will be observed, but the final local residual is a scalar equation.

7.1.1. Theory

Yield Function

The von Mises yield criterion is adopted,

$$f = \|\boldsymbol{\eta}\| - \sqrt{\frac{2}{3}}\sigma^y, \tag{7.1}$$

with $\boldsymbol{\eta} = \boldsymbol{s} - \boldsymbol{\alpha}$ is the shifted stress with $\boldsymbol{\alpha}$ denoting the back stress and $\boldsymbol{s} = \operatorname{dev}(\boldsymbol{\sigma})$ denoting the deviatoric stress. The only purpose of the constant $\sqrt{\frac{2}{3}}$ is to produce consistent response under uniaxial loading compared to the uniaxial version with the same set of model parameters. By definition, the back stress $\boldsymbol{\alpha}$ is also a deviatoric stress, thus $\operatorname{trace}(\boldsymbol{\alpha}) = 0$.

Flow Rule

Assuming associative rule, the flow rule is

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \frac{\partial f}{\partial \boldsymbol{\sigma}} = \gamma \frac{1}{2} \frac{\mathbb{I}^{\text{dev}} : \boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} = \gamma \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} = \gamma \boldsymbol{n}. \tag{7.2}$$

All analytical formulations are based on tensor notation. However, compressed matrix representation is used in implementation. One shall note the difference due to the Voigt notation.

$$\underbrace{\begin{bmatrix} \dot{\varepsilon}^p{}_{11} & \dot{\varepsilon}^p{}_{12} & \dot{\varepsilon}^p{}_{31} \\ \dot{\varepsilon}^p{}_{12} & \dot{\varepsilon}^p{}_{22} & \dot{\varepsilon}^p{}_{23} \\ \dot{\varepsilon}^p{}_{31} & \dot{\varepsilon}^p{}_{23} & \dot{\varepsilon}^p{}_{33} \end{bmatrix}}_{\dot{\boldsymbol{\varepsilon}}^p} = \gamma \underbrace{\begin{bmatrix} n_{11} & n_{12} & n_{31} \\ n_{12} & n_{22} & n_{23} \\ n_{31} & n_{23} & n_{33} \end{bmatrix}}_{\boldsymbol{n}}, \qquad \text{expressed in components} \tag{7.3}$$

We define the scaling vector $\boldsymbol{c} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix}^{\mathrm{T}}$ and let $\circ$ be the Hadamard (element–wise) product operator, then the above expression is equivalent to

$$\underbrace{\begin{bmatrix} \dot{\varepsilon}^p{}_{11} \\ \dot{\varepsilon}^p{}_{22} \\ \dot{\varepsilon}^p{}_{33} \\ 2\dot{\varepsilon}^p{}_{12} \\ 2\dot{\varepsilon}^p{}_{23} \\ 2\dot{\varepsilon}^p{}_{31} \end{bmatrix}}_{\dot{\boldsymbol{\varepsilon}}^p} = \gamma \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2 \end{bmatrix} \underbrace{\begin{bmatrix} n_{11} \\ n_{22} \\ n_{33} \\ n_{12} \\ n_{23} \\ n_{31} \end{bmatrix}}_{\boldsymbol{n}} = \gamma\, \boldsymbol{c} \circ \boldsymbol{n}. \qquad \text{expressed in the Voigt notation} \tag{7.4}$$

This agrees with Eq. (2.35).

It can be observed that $\dot{\boldsymbol{\varepsilon}}^p$ has a magnitude of γ while $\boldsymbol{n}$ is a unit tensor in $\mathbb{R}^3 \times \mathbb{R}^3$ hyper space. Thus $\boldsymbol{n}$ serves as a direction indicator, $\boldsymbol{\varepsilon}^p$ evolves towards $\boldsymbol{n}$ by the amount of γ. Since $\boldsymbol{n}$ is deviatoric, $\boldsymbol{\varepsilon}^p$ is also deviatoric.

Hardening Law

For internal variable q, the hardening law takes the accumulated magnitude of $\boldsymbol{\varepsilon}^p$.

$$\dot{q} = \sqrt{\frac{2}{3}} \|\dot{\boldsymbol{\varepsilon}}^p\| = \sqrt{\frac{2}{3}} \gamma. \tag{7.5}$$

For isotropic hardening, σ^y is defined as a general function of q,

$$\sigma^y = \sigma^y(q). \tag{7.6}$$

For kinematic hardening, the evolution law of back stress $\boldsymbol{\alpha}$ is defined to be

$$\dot{\boldsymbol{\alpha}} = \sqrt{\frac{2}{3}}\dot{H}\boldsymbol{n}. \tag{7.7}$$

in which $H = H(q)$ is now a scalar–valued function of q that controls the development of $\|\boldsymbol{\alpha}\|$, $\boldsymbol{\alpha}$ always evolves towards $\boldsymbol{n}$ by the amount $\dot{H} = H(q_{n+1}) - H(q_n)$ characterising the increment of H. The fraction $\sqrt{\frac{2}{3}}$ is introduced for consistent response as stated early. Since $\dot{\boldsymbol{\alpha}}$ and $\boldsymbol{n}$ are coaxial, $\boldsymbol{\alpha}$ stays deviatoric but may not be coaxial with all $\boldsymbol{n}$ through the loading process.

7.1.2. Formulation

The summation of the von Mises model is listed as follows.

Constitutive Law	$\boldsymbol{\sigma} = \boldsymbol{D} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p)$
Yield Function	$f = \|\boldsymbol{\eta}\| - \sqrt{\frac{2}{3}}\sigma^y$
Flow Rule	$\dot{\boldsymbol{\varepsilon}^p} = \gamma\boldsymbol{n}$
Hardening Law	$\dot{q} = \sqrt{\frac{2}{3}}\gamma$
	$\dot{\boldsymbol{\alpha}} = \sqrt{\frac{2}{3}}\gamma\dot{H}\boldsymbol{n}$

Elastic Loading/Unloading

The trial stress can be computed such as

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p) = \boldsymbol{\sigma}_n + \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n)\,. \tag{7.8}$$

In matrix representation, it is

$$\underbrace{\begin{bmatrix}\sigma_{11}^{\text{trial}}\\ \sigma_{22}^{\text{trial}}\\ \sigma_{33}^{\text{trial}}\\ \sigma_{12}^{\text{trial}}\\ \sigma_{23}^{\text{trial}}\\ \sigma_{31}^{\text{trial}}\end{bmatrix}}_{\boldsymbol{\sigma}^{\text{trial}}} = \underbrace{\begin{bmatrix}\lambda+2G & \lambda & \lambda & \cdot & \cdot & \cdot\\ \lambda & \lambda+2G & \lambda & \cdot & \cdot & \cdot\\ \lambda & \lambda & \lambda+2G & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & G & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & G & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & G\end{bmatrix}}_{\boldsymbol{D}} \underbrace{\left(\begin{bmatrix}\varepsilon_{11,n+1}\\ \varepsilon_{22,n+1}\\ \varepsilon_{33,n+1}\\ \gamma_{12,n+1}\\ \gamma_{23,n+1}\\ \gamma_{31,n+1}\end{bmatrix} - \begin{bmatrix}\varepsilon_{11,n}^p\\ \varepsilon_{22,n}^p\\ \varepsilon_{33,n}^p\\ \gamma_{12,n}^p\\ \gamma_{23,n}^p\\ \gamma_{31,n}^p\end{bmatrix}\right)}_{\dot{\boldsymbol{\varepsilon}}_{n+1} - \dot{\boldsymbol{\varepsilon}}_n^p}. \tag{7.9}$$

Then $\boldsymbol{\eta}^{\text{trial}} = \text{dev}\left(\boldsymbol{\sigma}^{\text{trial}}\right) - \boldsymbol{\alpha}_n$, which gives trial yield function

$$f^{\text{trial}} = \left\|\boldsymbol{\eta}^{\text{trial}}\right\| - \sqrt{\frac{2}{3}}\sigma_n^y \tag{7.10}$$

with $\sigma_n^y = \sigma^y\left(q_n\right)$ evaluated with current q_n.

Plastic Evolution

By default, we present the formulation with the implicit Euler method.

The yield function evaluated at the new state reads

$$\begin{aligned} f &= \left\|\text{dev}\left(\boldsymbol{\sigma}_{n+1}\right) - \boldsymbol{\alpha}_{n+1}\right\| - \sqrt{\frac{2}{3}}\sigma_{n+1}^y \\ &= \left\|\text{dev}\left(\boldsymbol{\sigma}^{\text{trial}}\right) - \gamma 2G\boldsymbol{n} - \boldsymbol{\alpha}_n - \sqrt{\frac{2}{3}}\left(H_{n+1} - H_n\right)\boldsymbol{n}\right\| - \sqrt{\frac{2}{3}}\sigma_{n+1}^y \\ &= \left\|\boldsymbol{\eta}^{\text{trial}} - \left(2G\gamma + \sqrt{\frac{2}{3}}\left(H_{n+1} - H_n\right)\right)\boldsymbol{n}\right\| - \sqrt{\frac{2}{3}}\sigma_{n+1}^y \\ &= \left\|\boldsymbol{\eta}^{\text{trial}}\right\| - \left(2G\gamma + \sqrt{\frac{2}{3}}\left(H_{n+1} - H_n\right)\right) - \sqrt{\frac{2}{3}}\sigma_{n+1}^y, \end{aligned} \tag{7.11}$$

given that it can be proved that $\boldsymbol{\eta}^{\text{trial}}$ and $\boldsymbol{\eta}$ are coaxial, following a similar derivation as shown previously.

The Jacobian reads

$$\frac{\partial f}{\partial \gamma} = -2G - \sqrt{\frac{2}{3}}\frac{\partial H_{n+1}}{\partial q_{n+1}}\frac{\partial q_{n+1}}{\partial \gamma} - \sqrt{\frac{2}{3}}\frac{\partial \sigma_{n+1}^y}{\partial q_{n+1}}\frac{\partial q_{n+1}}{\partial \gamma}. \tag{7.12}$$

Since $q_{n+1} = q_n + \sqrt{\frac{2}{3}}\gamma$, $\frac{\partial q_{n+1}}{\partial \gamma} = \sqrt{\frac{2}{3}}$. Hence,

$$\frac{\partial f}{\partial \gamma} = -2G - \frac{2}{3}\frac{\partial H_{n+1}}{\partial q_{n+1}} - \frac{2}{3}\frac{\partial \sigma_{n+1}^y}{\partial q_{n+1}}. \tag{7.13}$$

Consistent Tangent Stiffness

From $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}} - \gamma 2G\boldsymbol{n}$, as $\boldsymbol{n} = \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} = \frac{\boldsymbol{\eta}^{\text{trial}}}{\|\boldsymbol{\eta}^{\text{trial}}\|}$, the consistent tangent stiffness can be computed via the chain rule as

$$\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \frac{\partial \boldsymbol{\sigma}_{n+1}^{\text{trial}}}{\partial \boldsymbol{\varepsilon}_{n+1}} - 2G\boldsymbol{n} \otimes \frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} = \boldsymbol{D} - 2G\left(\boldsymbol{n} \otimes \frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} + \gamma\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\varepsilon}_{n+1}}\right). \tag{7.14}$$

In which, according to Eq. (2.41),

$$\begin{aligned}\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\varepsilon}_{n+1}} &= \frac{1}{\|\boldsymbol{\eta}^{\text{trial}}\|}\left(\mathbb{I}-\boldsymbol{n}\otimes\boldsymbol{n}\right):\frac{\partial \boldsymbol{\eta}^{\text{trial}}}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ &= \frac{2G}{\|\boldsymbol{\eta}^{\text{trial}}\|}\left(\mathbb{I}-\boldsymbol{n}\otimes\boldsymbol{n}\right):\mathbb{I}^{\text{dev}} \\ &= \frac{2G}{\|\boldsymbol{\eta}^{\text{trial}}\|}\left(\mathbb{I}^{\text{dev}}-\boldsymbol{n}\otimes\boldsymbol{n}\right).\end{aligned} \tag{7.15}$$

From converged local residual (yield function),

$$\frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} = -\left(\frac{\partial f}{\partial \gamma}\right)^{-1}\frac{\partial f}{\partial \boldsymbol{\varepsilon}_{n+1}} = -\left(\frac{\partial f}{\partial \gamma}\right)^{-1} 2G\boldsymbol{n}. \tag{7.16}$$

Thus the final expression for consistent tangent stiffness can be assembled as

$$\begin{aligned}\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} &= \boldsymbol{D}-2G\left(-2G\left(\frac{\partial f}{\partial \gamma}\right)^{-1}\boldsymbol{n}\otimes\boldsymbol{n}+\gamma\frac{2G}{\|\boldsymbol{\eta}^{\text{trial}}\|}\left(\mathbb{I}^{\text{dev}}-\boldsymbol{n}\otimes\boldsymbol{n}\right)\right) \\ &= \boldsymbol{D}+4G^2\left(\frac{\partial f}{\partial \gamma}\right)^{-1}\boldsymbol{n}\otimes\boldsymbol{n}+\frac{4G^2\gamma}{\|\boldsymbol{\eta}^{\text{trial}}\|}\left(\boldsymbol{n}\otimes\boldsymbol{n}-\mathbb{I}^{\text{dev}}\right) \\ &= \boldsymbol{D}+4G^2\left(\left(\frac{\partial f}{\partial \gamma}\right)^{-1}+\frac{\gamma}{\|\boldsymbol{\eta}^{\text{trial}}\|}\right)\boldsymbol{n}\otimes\boldsymbol{n}-\frac{4G^2\gamma}{\|\boldsymbol{\eta}^{\text{trial}}\|}\mathbb{I}^{\text{dev}} \\ &= \boldsymbol{D}+4G^2\left(\frac{\gamma}{\|\boldsymbol{\eta}^{\text{trial}}\|}-\frac{1}{2G+\frac{2}{3}\frac{\partial H_{n+1}}{\partial q_{n+1}}+\frac{2}{3}\frac{\partial \sigma^y_{n+1}}{\partial q_{n+1}}}\right)\boldsymbol{n}\otimes\boldsymbol{n}-\frac{4G^2\gamma}{\|\boldsymbol{\eta}^{\text{trial}}\|}\mathbb{I}^{\text{dev}}.\end{aligned} \tag{7.17}$$

It shall be noted that $\mathbb{I}^{\text{dev}}$ takes the form as presented in Eq. (3.23). Readers are strongly suggested to derive it via both tensor notation and compression matrix representation as a practice. Both leads to identical results.

Since the local iteration is a scalar function, the closed–form of consistent tangent stiffness is relatively easy to compute. It will be seen in more complex models that closed–forms do not always possess simple forms.

As a general framework, the above formulation does not require explicit forms of $H(q)$ and $\sigma^y(q)$. Thus, various types of scalar–valued functions can be adopted.

7.1.3. Implementation

The state determination algorithm of this general model incorporating von Mises criterion is given in Algorithm 8.

Algorithm 8 state determination of general von Mises model

Parameter: λ, G
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\varepsilon}^p_n$, $\boldsymbol{\sigma}_n$, $\boldsymbol{\alpha}_n$, q_n
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\varepsilon}^p_{n+1}$, $\boldsymbol{\sigma}_{n+1}$, $\boldsymbol{\alpha}_{n+1}$, q_{n+1}
compute $\boldsymbol{\sigma}^{\text{trial}}$, $\boldsymbol{\eta}^{\text{trial}}$, $\boldsymbol{n}$ and f^{trial} ▷ Eq. (7.8) and Eq. (7.10)
if $f^{\text{trial}} \geqslant 0$ **then**
  $\gamma = 0$
  while true **do**
    compute f and $\dfrac{\partial f}{\partial \gamma}$ ▷ Eq. (7.11) and Eq. (7.13)
    update $\Delta H = H\left(q_{n+1}\right) - H\left(q_n\right)$
    $\Delta\gamma = \left(\dfrac{\partial f}{\partial \gamma}\right)^{-1} f$
    if $|\Delta\gamma| <$ tolerance **then**
      break
    end if
    $\gamma \leftarrow \gamma - \Delta\gamma$
    $q_{n+1} = q_n + \sqrt{\dfrac{2}{3}}\gamma$
  end while
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}} - \gamma 2G\boldsymbol{n}$
  $\boldsymbol{\varepsilon}^p_{n+1} = \boldsymbol{\varepsilon}^p_n + \gamma\boldsymbol{n}$
  $\boldsymbol{\alpha}^p_{n+1} = \boldsymbol{\alpha}^p_n + \sqrt{\dfrac{2}{3}}\Delta H\boldsymbol{n}$
  compute $\boldsymbol{D}_{n+1}$ ▷ Eq. (7.17)
else
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}$
  $\boldsymbol{\varepsilon}^p_{n+1} = \boldsymbol{\varepsilon}^p_n$
  $\boldsymbol{\alpha}^p_{n+1} = \boldsymbol{\alpha}^p_n$
  $q_{n+1} = q_n$
  $\boldsymbol{D}_{n+1} = \boldsymbol{D}$
end if

It shall be noted that the algorithm does not implement $H\left(q\right)$ and $\sigma^y\left(q\right)$. It is assumed those two functions are defined somewhere else and are able to provide derivatives.

7.1.4. Closing Remarks

As the first 3D material model introduced, the von Mises framework allows beginners to get familiar with multiaxial constitutive modelling with a relatively smooth learning curve. The formulation is expressed in tensor notation. Readers are strongly encouraged to derive the formulation from three governing equations independently in both tensor and compressed matrix notions separately. It is a good practice to get each tiny detail correct.

```cpp
int NonlinearJ2::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * incre_strain;

    trial_history = current_history;
    auto& plastic_strain = trial_history(0);
    vec back_stress(&trial_history(1), 6, false, true);

    const vec rel_stress = tensor::dev(trial_stress) - back_stress;
    const auto norm_rel_stress = tensor::stress::norm(rel_stress);

    auto yield_func = norm_rel_stress - root_two_third * std::max(0.,
    ↪  compute_k(plastic_strain));

    if(yield_func < 0.) return SUANPAN_SUCCESS;

    const auto current_h = compute_h(plastic_strain);
    auto gamma = 0., incre_h = 0., denom = 0.;
    unsigned counter = 0;
    while(++counter < max_iteration) {
        denom = double_shear + two_third * (compute_dk(plastic_strain) +
        ↪  compute_dh(plastic_strain));
        const auto incre_gamma = yield_func / denom;
        const auto abs_error = fabs(incre_gamma);
        suanpan_extra_debug("NonlinearJ2 local iteration error: %.5E.\n", abs_error);
        if(abs_error <= tolerance) break;
        incre_h = compute_h(plastic_strain = current_history(0) + root_two_third * (gamma +=
        ↪  incre_gamma)) - current_h;
        yield_func = norm_rel_stress - double_shear * gamma - root_two_third * (std::max(0.,
        ↪  compute_k(plastic_strain)) + incre_h);
    }

    if(max_iteration == counter) {
        suanpan_error("NonlinearJ2 cannot converge within %u iterations.\n", max_iteration);
        return SUANPAN_FAIL;
    }

    back_stress += root_two_third * incre_h / norm_rel_stress * rel_stress;

    auto t_factor = double_shear * gamma / norm_rel_stress;
    trial_stress -= t_factor * rel_stress;

    t_factor *= double_shear;
    trial_stiffness += (t_factor - square_double_shear / denom) / norm_rel_stress /
    ↪  norm_rel_stress * rel_stress * rel_stress.t() - t_factor * unit_dev_tensor;

    return SUANPAN_SUCCESS;
}
```

7.2. Hoffman J2 Model

Here we introduce an anisotropic model that adopts the Hoffman yielding criterion. This framework resembles the isotropic von Mises model. It can be used to model orthtropic materials such sheet steel and timber.

7.2.1. Theory

Yield Function

The yield function adopts the Hoffman criterion.

$$f = \frac{1}{2}\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\sigma} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{\sigma} - \sigma_y^2, \tag{7.18}$$

where $\boldsymbol{P} = \boldsymbol{P}^{\mathrm{T}}$ and $\boldsymbol{q}$ are constant scaling matrix/vector of various forms [11] that depend on material constants.

For example, the Hoffman criterion can be expressed as

$$\boldsymbol{P} = \begin{bmatrix} T_1 & -\frac{T_1+T_2-T_3}{2} & -\frac{T_3+T_1-T_2}{2} & & & \\ -\frac{T_1+T_2-T_3}{2} & T_2 & -\frac{T_2+T_3-T_1}{2} & & & \\ -\frac{T_3+T_1-T_2}{2} & -\frac{T_2+T_3-T_1}{2} & T_3 & & & \\ & & & \frac{1}{f_{12}^2} & & \\ & & & & \frac{1}{f_{23}^2} & \\ & & & & & \frac{1}{f_{13}^2} \end{bmatrix}, \quad \boldsymbol{q} = \begin{bmatrix} \left(f_{11}^c - f_{11}^t\right) T_1 \\ \left(f_{22}^c - f_{22}^t\right) T_2 \\ \left(f_{33}^c - f_{33}^t\right) T_3 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{7.19}$$

in which,

$$T_1 = \frac{1}{f_{11}^t f_{11}^c}, \qquad T_2 = \frac{1}{f_{22}^t f_{22}^c}, \qquad T_3 = \frac{1}{f_{33}^t f_{33}^c}, \tag{7.20}$$

with $f_{ij}^{\aleph}$ representing the yielding stress along different directions.

Flow Rule

The associated plasticity is assumed such that the plastic potential g is identical to f. The plastic flow direction is then

$$\boldsymbol{n} = \frac{\partial g}{\partial \boldsymbol{\sigma}} = \frac{\partial f}{\partial \boldsymbol{\sigma}} = \boldsymbol{P}\boldsymbol{\sigma} + \boldsymbol{q}. \tag{7.21}$$

The flow rule can be defined as

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \boldsymbol{n}. \tag{7.22}$$

Hardening Law

The reference yield stress σ_y is defined as a function of the accumulated equivalent plastic strain ε_p.

$$\sigma_y = H\left(\varepsilon^p\right). \tag{7.23}$$

The evolution of ε^p is driven by the norm of $\dot{\boldsymbol{\varepsilon}}^p$.

$$\dot{\varepsilon}^p = \|\dot{\boldsymbol{\varepsilon}}^p\| = \gamma \|\boldsymbol{n}\|, \tag{7.24}$$

where $\|\boldsymbol{n}\|$, in matrix form, can be expressed as

$$\|\boldsymbol{n}\| = \sqrt{\frac{2}{3}\boldsymbol{n}^{\mathrm{T}}\boldsymbol{T}\boldsymbol{n}}, \tag{7.25}$$

with $\boldsymbol{T} = \operatorname{diag}\begin{pmatrix} 1 & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$.

7.2.2. Formulation

To some extend, the model is even simpler than the von Mises model as there is no back stress to support kinematic hardening. Furthermore, the yield function involves only matrix–vector operations, the corresponding derivatives are relatively easy to compute.

Constitutive Law	$\boldsymbol{\sigma} = \boldsymbol{D} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p)$
Yield Function	$f = \frac{1}{2}\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\sigma} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{\sigma} - \sigma_y^2$
Flow Rule	$\dot{\boldsymbol{\varepsilon}}^p = \gamma \boldsymbol{n}$
Hardening Law	$\dot{\varepsilon}^p = \gamma \|\boldsymbol{n}\|$

Elastic Loading/Unloading

Compared with the von Mises framework, there is no essential difference in terms of elastic loading/unloading, the plasticity is frozen at the beginning of each substep, allowing one to compute the trial stress such that

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p) = \boldsymbol{\sigma}_n + \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n). \tag{7.26}$$

One shall note that the second form is not used here. In most cases, both forms are equivalent. However, some models may further apply damage mechanics to the result of plasticity, making the second form incorrect (as $\boldsymbol{\sigma}_n$ may contain damage reductions).

The trial yield function can then be computed using the unchanged plastic strain.

$$f^{\text{trial}} = \frac{1}{2}\boldsymbol{\sigma}^{\text{trial,T}}\boldsymbol{P}\boldsymbol{\sigma}^{\text{trial}} + \boldsymbol{q}^{\text{T}}\boldsymbol{\sigma}^{\text{trial}} - \sigma_{y,n}^2. \tag{7.27}$$

Plastic Evolution

Since it is an anisotropic model, the local iteration may have difficulties in convergence, especially when a high anisotropy is defined. Some implementations [12, 13] adopt line search, which does mitigate the local convergence issue but does not address it at the global level.

In this work, we choose another approach to alleviate the problem. Instead of the first order accurate backward Euler method for numerical integration, the higher order accurate method is used. In specific, the discretised evolution of plastic strain is written as

$$\boldsymbol{\varepsilon}_{n+1}^p = \boldsymbol{\varepsilon}_n^p + \Delta\boldsymbol{\varepsilon}^p = \boldsymbol{\varepsilon}_n^p + \gamma\boldsymbol{n}_m, \tag{7.28}$$

where $\boldsymbol{n}_m$ is $\boldsymbol{n}$ evaluated at the middle of the substep. Since it is a linear function of σ,

$$\boldsymbol{n}_m = \frac{\boldsymbol{\sigma}_n + \boldsymbol{\sigma}_{n+1}}{2}. \tag{7.29}$$

One shall note that replacing $\boldsymbol{n}_m$ by $\boldsymbol{n}_{n+1} = \boldsymbol{P}\boldsymbol{\sigma}_{n+1} + \boldsymbol{q}$ gives the evolution formula of the implicit Euler method.

Local Residual

The residual is chosen as follows. For brevity, all subscripts $(\cdot)_{n+1}$ are dropped.

$$\boldsymbol{R} = \begin{cases} \frac{1}{2}\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\sigma} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{\sigma} - \sigma_y^2, \\ \boldsymbol{\sigma} + \gamma \boldsymbol{E}\boldsymbol{n}_m - \boldsymbol{\sigma}^{\mathrm{trial}}. \end{cases} \tag{7.30}$$

By choosing $\boldsymbol{x} = \begin{bmatrix} \gamma & \boldsymbol{\sigma} \end{bmatrix}^{\mathrm{T}}$ as the independent variables, the Jacobian can be then computed as

$$\boldsymbol{J} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \begin{bmatrix} -2\sigma_y \dfrac{\mathrm{d}\sigma_y}{\mathrm{d}\varepsilon_p}\|\boldsymbol{n}_m\| & \boldsymbol{n}^{\mathrm{T}} - \sigma_y \dfrac{\mathrm{d}\sigma_y}{\mathrm{d}\varepsilon_p}\gamma \dfrac{\mathrm{d}\|\boldsymbol{n}_m\|}{\mathrm{d}\boldsymbol{n}_m}\boldsymbol{P} \\ \boldsymbol{E}\boldsymbol{n}_m & \boldsymbol{I} + \dfrac{\gamma}{2}\boldsymbol{E}\boldsymbol{P} \end{bmatrix}. \tag{7.31}$$

In the above expression,

$$\frac{\mathrm{d}\|\boldsymbol{n}_m\|}{\mathrm{d}\boldsymbol{n}_m} = \frac{2}{3}\frac{\boldsymbol{n}_m^{\mathrm{T}}\boldsymbol{T}}{\|\boldsymbol{n}_m\|}. \tag{7.32}$$

Some references would further derive a scalar local residual at the cost of complicating gradient. Here we choose to increase the size of local system in order to express the Jacobian in a simpler form. Performance wise, a scalar local residual does not necessarily leads to faster state determination.

Consistent Tangent Stiffness

The consistent tangent stiffness can be directly computed from the local residual, given that $\boldsymbol{\sigma}_{n+1}$ is chosen as the variable. Differentiating $\boldsymbol{R}$ at equilibrium $\boldsymbol{R} = \boldsymbol{0}$ gives

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \boldsymbol{0}, \tag{7.33}$$

rearranging which gives

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = -\boldsymbol{J}^{-1}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{7.34}$$

The right hand side $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ can be computed as

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} 0 \\ -\boldsymbol{E} \end{bmatrix}, \tag{7.35}$$

while the left hand side $\dfrac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ contains

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} \dfrac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} \end{bmatrix}. \tag{7.36}$$

Thus,

$$\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \left(\boldsymbol{J}^{-1} \begin{bmatrix} 0 \\ \boldsymbol{E} \end{bmatrix} \right)^{\langle 2-7 \rangle}, \tag{7.37}$$

where $(\cdot)^{\langle 2-7 \rangle}$ denotes the second to the seventh row of target quantity $(\cdot)$.

Unlike the von Mises framework, in which the analytical expression for the consistent tangent stiffness matrix is derived. Here we take advantage of the fact that when the local equilibrium is achieved, $\boldsymbol{R} = \boldsymbol{0}$ or at least $\boldsymbol{R} \approx \boldsymbol{0}$, allowing one to take full differentiation to obtain some useful quantities that otherwise may be difficult to compute. If $\boldsymbol{\sigma}_{n+1}$ is directly involved as one of the independent variables in local iteration, the consistent stiffness can be directly obtained. Otherwise, often additional simple steps (chain rule) shall be applied to the stress update formula to compute $\dfrac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}}$.

This method avoid the computation of lengthy, cumbersome analytical expressions of consistent tangent. In most cases, it is also very simple to implement as $\boldsymbol{J}$ is already available when the local iteration converges, and $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ often is very easy to compute. Readers shall try to grasp the beauty of Eq. (7.33), as this method will be frequently used in the models introduced later in this book.

7.2.3. Implementation

```
int NonlinearHoffman::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    trial_history = current_history;
    auto& eqv_strain = trial_history(0);
```

```
    const auto& current_eqv_strain = current_history(0);
    vec plastic_strain(&trial_history(1), 6, false, true);

    const vec predictor = (trial_stiffness = initial_stiffness) * (trial_strain -
    ↪ plastic_strain);
    trial_stress = predictor;
    const vec c_stress = .5 * proj_a * initial_stiffness * (current_strain - plastic_strain);

    auto gamma = 0., ref_error = 1.;

    vec incre, residual(7);
    mat jacobian(7, 7);

    auto counter = 0u;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("Cannot converge within {} iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        const vec factor_a = proj_a * trial_stress;
        const vec factor_b = .5 * factor_a + proj_b;
        const vec n_mid = c_stress + factor_b;
        const auto norm_n_mid = root_two_third * tensor::strain::norm(n_mid);
        const auto k = compute_k(eqv_strain = current_eqv_strain + gamma * norm_n_mid);
        const auto f = dot(trial_stress, factor_b) - k * k;

        if(1u == counter && f < 0.) return SUANPAN_SUCCESS;

        const rowvec dn = two_third / norm_n_mid * (n_mid % tensor::strain::norm_weight).t();
        const auto factor_c = k * compute_dk(eqv_strain);

        residual(sa) = f;
        residual(sb) = trial_stress + gamma * initial_stiffness * n_mid - predictor;

        jacobian(sa, sa) = -2. * factor_c * norm_n_mid;
        jacobian(sa, sb) = factor_a.t() + proj_b.t() - factor_c * gamma * dn * proj_a;
        jacobian(sb, sa) = initial_stiffness * n_mid;
        jacobian(sb, sb) = eye(6, 6) + .5 * gamma * elastic_a;

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        const auto error = norm(incre);
        if(1u == counter && error > ref_error) ref_error = error;
        suanpan_debug("Local plasticity iteration error: {:.5E}.\n", error / ref_error);

        if(error <= tolerance * std::max(1., ref_error)) {
            plastic_strain += gamma * n_mid;

            mat left, right(7, 6, fill::zeros);
            right.rows(sb) = initial_stiffness;

            if(!solve(left, jacobian, right)) return SUANPAN_FAIL;

            trial_stiffness = left.rows(sb);

            return SUANPAN_SUCCESS;
```

```
        }

        gamma -= incre(sa);
        trial_stress -= incre(sb);
    }
}
```

8. Timber

8.1. TimberPD

Here we present a 3D model based on the Hoffman yielding criterion and damage mechanics. It is suitable for modelling timber. The model is based on the split of the effective stress $\bar{\boldsymbol{\sigma}}$ such that

$$\bar{\boldsymbol{\sigma}} = \bar{\boldsymbol{\sigma}}_t + \bar{\boldsymbol{\sigma}}_c, \tag{8.1}$$

with

$$\bar{\boldsymbol{\sigma}}_t = \sum_{i=1}^{3} \langle \hat{\sigma}_i \rangle \, \boldsymbol{p}_i \otimes \boldsymbol{p}_i, \qquad \bar{\boldsymbol{\sigma}}_c = \sum_{i=1}^{3} \left(\hat{\sigma}_i - \langle \hat{\sigma}_i \rangle \right) \boldsymbol{p}_i \otimes \boldsymbol{p}_i, \tag{8.2}$$

where $\boldsymbol{p}_i$ and $\hat{\sigma}_i$ are eigenvectors and eigenvalues of the second order tensor $\bar{\boldsymbol{\sigma}}$. The above expression shall be interpreted with the tensor notation.

The effective stress $\bar{\boldsymbol{\sigma}}$ obeys the conventional hardening model using the Hoffman yielding criterion, see § 7.2 for details. The final stress can be obtained after applying damage factors on tensile and compressive part of $\bar{\boldsymbol{\sigma}}$.

$$\boldsymbol{\sigma} = (1 - \omega_t) \, \bar{\boldsymbol{\sigma}}_t + (1 - \omega_c) \, \bar{\boldsymbol{\sigma}}_c. \tag{8.3}$$

8.1.1. Damage

The damage part follows the one proposed in [14]. The damage evolution is governed by the equivalent stress $\bar{\tau}_{\aleph}$.

$$\bar{\tau}_{\aleph} = \sqrt{\frac{1}{2} \bar{\boldsymbol{\sigma}}_{\aleph}^{\mathrm{T}} \boldsymbol{H}_{\aleph} \bar{\boldsymbol{\sigma}}_{\aleph}}, \tag{8.4}$$

that covers both tensile and compressive cases. The matrix $\boldsymbol{H}_{\aleph}$ is the projection matrix of the Hill criterion, which is a special case of Eq. (7.19). For $\aleph = t$, set $f_{ii}^{c} = f_{ii}^{t}$. For $\aleph = c$, set $f_{ii}^{t} = f_{ii}^{c}$. It could be noted that in either case, $\boldsymbol{q} = \boldsymbol{0}$, thus, Eq. (8.4) does not contain a second term.

The damage variables are updated based on the maximum history of $\bar{\tau}_{\aleph}$, that is

$$r_{\aleph} = \max_t \bar{\tau}_{\aleph}, \tag{8.5}$$

and

$$\omega_t = 1 - \frac{r_{t,0}}{r_t}\left(1 - n + n\exp\left(b\left(r_{t,0} - r_t\right)\right)\right), \tag{8.6}$$

$$\omega_c = \beta\left(1 - \frac{r_{c,0}}{r_c}\right)^m. \tag{8.7}$$

The final stress is

$$\boldsymbol{\sigma} = \left(1 - \omega_t\right)\bar{\boldsymbol{\sigma}}_t + \left(1 - \omega_c\right)\bar{\boldsymbol{\sigma}}_c. \tag{8.8}$$

8.1.2. Consistent Tangent Stiffness

In the case of activation of damage evolution,

$$\frac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}} = \left(\left(\left(1-\omega_t\right)\boldsymbol{I} - \bar{\boldsymbol{\sigma}}_t\frac{\mathrm{d}\omega_t}{\mathrm{d}r_t}\frac{\mathrm{d}r_t}{\mathrm{d}\bar{\boldsymbol{\sigma}}_t}\right)\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_t}{\mathrm{d}\bar{\boldsymbol{\sigma}}} + \left(\left(1-\omega_c\right)\boldsymbol{I} - \bar{\boldsymbol{\sigma}}_c\frac{\mathrm{d}\omega_c}{\mathrm{d}r_c}\frac{\mathrm{d}r_c}{\mathrm{d}\bar{\boldsymbol{\sigma}}_c}\right)\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_c}{\mathrm{d}\bar{\boldsymbol{\sigma}}}\right)\frac{\partial\bar{\boldsymbol{\sigma}}}{\partial\boldsymbol{\varepsilon}}. \tag{8.9}$$

8.1.3. Implementation

```
int TimberPD::update_trial_status(const vec& t_strain) {
    if(SUANPAN_SUCCESS != BilinearHoffman::update_trial_status(t_strain)) return SUANPAN_FAIL;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    vec principal_stress;    // 3
    mat principal_direction; // 3x3
    if(!eig_sym(principal_stress, principal_direction, tensor::stress::to_tensor(trial_stress), "std")) return SUANPAN_FAIL;

    mat stiffness_t = transform::eigen_to_tensile_derivative(principal_stress, principal_direction);
    mat stiffness_c = eye(6, 6) - stiffness_t;

    const vec sigma_t = transform::eigen_to_tensile_stress(principal_stress, principal_direction);
    const vec sigma_c = trial_stress - sigma_t;

    const auto omega_t = update_damage_t(sigma_t, stiffness_t);
    const auto omega_c = update_damage_c(sigma_c, stiffness_c);

    trial_stress = (1. - omega_t) * sigma_t + (1. - omega_c) * sigma_c;
```

```
    trial_stiffness = (stiffness_t + stiffness_c) * trial_stiffness;

    return SUANPAN_SUCCESS;
}

double TimberPD::update_damage_t(const vec& sigma_t, mat& stiffness_t) {
    auto& r_t = trial_history(7);

    bool new_damage_t = false;
    if(const auto eqv_stress_t = sqrt(.5 * dot(hill_t * sigma_t, sigma_t)); eqv_stress_t > r_t) {
        new_damage_t = true;
        r_t = eqv_stress_t;
    }

    const auto omega_t = compute_damage_t(r_t);
    if(new_damage_t) {
        const auto domega_t = ini_r_t / r_t / r_t * ((b_t * n_t * r_t + n_t) * exp(b_t * (ini_r_t - r_t)) - n_t + 1.);
        stiffness_t = ((1. - omega_t) * eye(6, 6) - sigma_t * domega_t * .5 / r_t * sigma_t.t() * hill_t) * stiffness_t;
    }
    else stiffness_t *= 1. - omega_t;

    return omega_t;
}

double TimberPD::update_damage_c(const vec& sigma_c, mat& stiffness_c) {
    auto& r_c = trial_history(8);

    bool new_damage_c = false;
    if(const auto eqv_stress_c = sqrt(.5 * dot(hill_c * sigma_c, sigma_c)); eqv_stress_c > r_c) {
        new_damage_c = true;
        r_c = eqv_stress_c;
    }

    const auto omega_c = compute_damage_c(r_c);
    if(new_damage_c) {
        const auto domega_c = m_c * ini_r_c / r_c * omega_c / (r_c - ini_r_c);
        stiffness_c = ((1. - omega_c) * eye(6, 6) - sigma_c * domega_c * .5 / r_c * sigma_c.t() * hill_c) * stiffness_c;
    }
    else stiffness_c *= 1. - omega_c;

    return omega_c;
}

double TimberPD::compute_damage_t(const double r_t) const { return 1. - ini_r_t / r_t * (1. - n_t + n_t * exp(b_t * (ini_r_t - r_t))); }

double TimberPD::compute_damage_c(const double r_c) const { return b_c * pow(std::max(datum::eps, 1. - ini_r_c / r_c), m_c); }
```

9. Concrete

9.1. Concrete Damage Plasticity Model

In this section, the concrete damage plasticity model proposed by [15] is presented. A slight different version (with Lode angle dependency and others) is implemented in ABAQUS.

The CDP model follows Lemaitre's damage theory [9] and is developed under the assumption of isotropic damage. Accordingly, the final stress $\boldsymbol{\sigma}$ can be expressed as the product of the effective stress $\bar{\boldsymbol{\sigma}}$ and some function of damage measure.

$$\boldsymbol{\sigma} = h\left(d_t, d_c\right) \bar{\boldsymbol{\sigma}}, \tag{9.1}$$

where $h\left(d_t, d_c\right)$ is a function of two damage variables d_t and d_c, which depend on some internal history variables.

The effective part $\bar{\boldsymbol{\sigma}}$ fully resembles the conventional plasticity. Thus $h\left(d_t, d_c\right)$ and $\bar{\boldsymbol{\sigma}}$ can be handled in a relatively independent manner.

9.1.1. Plasticity Theory

Yield Function

The yield function is defined as

$$f = \alpha \bar{I}_1 + \sqrt{\frac{3}{2}} \|\bar{\boldsymbol{s}}\| + \beta \left\langle \hat{\sigma}_1 \right\rangle - (1-\alpha)\, c_c, \tag{9.2}$$

with $\bar{I}_1 = \operatorname{trace}\left(\bar{\boldsymbol{\sigma}}\right)$ is the first invariant of effective stress tensor $\bar{\boldsymbol{\sigma}}$, $\hat{\sigma}_1$ is the major effective principal stress, $c_c = -\bar{f}_c$ denotes cohesion and $\beta = \dfrac{\bar{f}_c}{\bar{f}_t}(\alpha - 1) - (\alpha + 1)$. The effective backbone stresses (both positive) $\bar{f}_c$ and $\bar{f}_t$ will be defined later.

Flow Rule

The flow potential g is chosen to be

$$g = \sqrt{2\bar{J}_2} + \alpha_p \bar{I}_1 = \|\bar{\boldsymbol{s}}\| + \operatorname{trace}(\alpha_p \bar{\boldsymbol{\sigma}}). \tag{9.3}$$

The flow rule is accordingly defined as

$$\dot{\boldsymbol{\varepsilon}^p} = \gamma \frac{\partial g}{\partial \bar{\boldsymbol{\sigma}}} = \gamma \left(\frac{\bar{\boldsymbol{s}}}{\|\bar{\boldsymbol{s}}\|} + \alpha_p \mathbf{1} \right) = \gamma (\boldsymbol{n} + \alpha_p \mathbf{1}). \tag{9.4}$$

In deviatoric and spherical components,

$$\dot{\boldsymbol{\varepsilon}^{d,p}} = \gamma \boldsymbol{n}, \quad \dot{\varepsilon^{v,p}} = \gamma 3 \alpha_p. \tag{9.5}$$

Noting that

$$\bar{\boldsymbol{s}} = \bar{\boldsymbol{s}}^{\text{trial}} - 2G\dot{\boldsymbol{\varepsilon}^{d,p}} = \bar{\boldsymbol{s}}^{\text{trial}} - \gamma 2G\boldsymbol{n}, \tag{9.6}$$

$$\bar{p} = \bar{p}^{\text{trial}} - K\dot{\varepsilon^{v,p}} = \bar{p}^{\text{trial}} - \gamma 3K\alpha_p, \tag{9.7}$$

equivalently,

$$\|\bar{\boldsymbol{s}}\| + \gamma 2G = \left\|\bar{\boldsymbol{s}}^{\text{trial}}\right\|, \tag{9.8}$$

$$\bar{I}_1 + \gamma 9K\alpha_p = \bar{I}_1^{\text{trial}}. \tag{9.9}$$

Furthermore, $\boldsymbol{s}$ and $\boldsymbol{s}^{\text{trial}}$ are coaxial, thus,

$$\frac{\bar{\boldsymbol{s}}}{\|\bar{\boldsymbol{s}}\|} = \frac{\bar{\boldsymbol{s}}^{\text{trial}}}{\|\bar{\boldsymbol{s}}^{\text{trial}}\|} \equiv \boldsymbol{n}. \tag{9.10}$$

It simply means the flow direction is fixed for all iterations in each sub-step. And due to the coaxiality, $\bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{s}}^{\text{trial}}$ share the same eigen space. More importantly, the eigenvectors remain constant for each iteration. Thus,

$$\frac{\bar{\boldsymbol{s}}}{\|\bar{\boldsymbol{s}}\|} = \frac{\bar{\boldsymbol{s}}^{\text{trial}}}{\|\bar{\boldsymbol{s}}^{\text{trial}}\|}, \qquad \frac{\hat{\boldsymbol{s}}}{\|\hat{\boldsymbol{s}}\|} = \frac{\hat{\boldsymbol{s}}^{\text{trial}}}{\|\hat{\boldsymbol{s}}^{\text{trial}}\|}, \tag{9.11}$$

where $\hat{\boldsymbol{s}}$ denotes the principal stress tensor of the deviatoric stress tensor $\bar{\boldsymbol{s}}$. As the yield function f can be equivalently expressed with the principal stresses, in the following derivation, $\hat{\boldsymbol{n}}$ is used to represent the unit principal deviatoric stress which has three components. The transformation matrix $\boldsymbol{T}$ is defined as

$$\underbrace{\hat{\boldsymbol{\sigma}}}_{3\times 1} = \underbrace{\boldsymbol{T}}_{3\times 6}\underbrace{\bar{\boldsymbol{\sigma}}}_{6\times 1}, \qquad \underbrace{\hat{\boldsymbol{s}}}_{3\times 1} = \underbrace{\boldsymbol{T}}_{3\times 6}\underbrace{\bar{\boldsymbol{s}}}_{6\times 1}, \tag{9.12}$$

and can be formulated from eigenanalysis.

Hardening Law

Since concrete shows different behaviour under compression and tension, the subscript $(\cdot)_{\aleph}$ is used to denoted either tension $(\cdot)_t$ or compression $(\cdot)_c$.

Internal hardening parameters $\kappa_{\aleph}$ shall satisfy the following expression,

$$\dot{\kappa_{\aleph}} = \gamma H_{\aleph}, \tag{9.13}$$

where $H_{\aleph}$ defines the hardening law. Different $H_{\aleph}$ shall be used for compression/tension.

$$H_t = r\frac{f_t}{g_t}\left(\hat{n}_1 + \alpha_p\right), \tag{9.14}$$

$$H_c = (1-r)\frac{f_c}{g_c}\left(\hat{n}_3 + \alpha_p\right). \tag{9.15}$$

In which $\hat{n}_1$ and $\hat{n}_3$ denotes the maximum and minimum components in $\hat{\boldsymbol{n}}$ and r is a scalar valued function of the effective principal stress. In the original model, it is defined as

$$r\left(\hat{\boldsymbol{\sigma}}\right) = \frac{\langle\hat{\sigma}_1\rangle + \langle\hat{\sigma}_2\rangle + \langle\hat{\sigma}_3\rangle}{|\hat{\sigma}_1| + |\hat{\sigma}_2| + |\hat{\sigma}_3|}. \tag{9.16}$$

The purpose of r is to characterise the proportion of tension in a multiaxial loading case.

Backbone Curve

The backbone curve $f_{\aleph}$ is related to the internal parameter $\kappa_{\aleph}$.

$$f_{\aleph} = f_{\aleph,0}\sqrt{\phi_{\aleph}}\Phi_{\aleph},$$

with

$$\phi_{\aleph} = 1 + a_{\aleph}\left(2 + a_{\aleph}\right)\kappa_{\aleph}, \qquad \Phi_{\aleph} = \frac{1 + a_{\aleph} - \sqrt{\phi_{\aleph}}}{a_{\aleph}}.$$

The effective counterpart $\bar{f}_{\aleph}$ is defined as

$$\bar{f}_{\aleph} = \frac{f_{\aleph}}{1 - d_{\aleph}} = f_{\aleph,0}\sqrt{\phi_{\aleph}}\Phi_{\aleph}^{1-c_{\aleph}/b_{\aleph}},$$

with

$$d_{\aleph} = 1 - \Phi_{\aleph}^{c_{\aleph}/b_{\aleph}}.$$

In general, the backbone curve can be customised. The main algorithm has no interested in how the backbone curve is computed, only $f_{\aleph}$, $\bar{f}_{\aleph}$ and $d_{\aleph}$ and their derivatives with regard to

$\kappa_{\aleph}$ need to be provided. The exponential form adopted in the original model may encounter some numerical difficulties, which in the author's opinion is not ideal. One can always choose another form, such as providing those quantities in a tabulated fashion. This is also what ABAQUS offers.

9.1.2. Damage Theory

The damage measure takes the form

$$h\left(d_t, d_c\right) = \left(1 - d_c\right)\left(1 - s d_t\right) \tag{9.17}$$

with $s = s_0 + \left(1 - s_0\right) r$ is the recovery factor.

9.1.3. Plasticity Formulation

The CDP model is driven by three quantities κ_t, κ_c and $\boldsymbol{\varepsilon}^p$. The governing equations are listed as follows.

Yield Function	$f = \alpha \bar{I}_1 + \sqrt{\frac{3}{2}} \|\bar{\boldsymbol{s}}\| + \beta \left\langle \hat{\sigma}_1 \right\rangle - \left(1 - \alpha\right) c_c$
Flow Rule	$\dot{\boldsymbol{\varepsilon}^p} = \gamma \left(\frac{\bar{\boldsymbol{s}}}{\|\bar{\boldsymbol{s}}\|} + \alpha_p \mathbf{1} \right)$
Hardening Law	$\dot{\kappa_{\aleph}} = \gamma H_{\aleph}$

Elastic Loading/Unloading

Assuming elastic loading/unloading, the trial state of the effective part can be computed as done in other plasticity models.

$$\bar{\boldsymbol{\sigma}}^{\text{trial}} = \boldsymbol{D} : \left(\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p\right). \tag{9.18}$$

Then by performing eigenanalysis, $\hat{\boldsymbol{\sigma}}^{\text{trial}}$ can be computed. The trial yield function is

$$f^{\text{trial}} = \alpha \bar{I}_1^{\text{trial}} + \sqrt{\frac{3}{2}} \left\| \bar{\boldsymbol{s}}^{\text{trial}} \right\| + \beta \left\langle \hat{\sigma}_1^{\text{trial}} \right\rangle - \left(1 - \alpha\right) c_c, \tag{9.19}$$

with β and c_c computed by using $\kappa_{t,n}$ and $\kappa_{c,n}$. If $f^{\text{trial}} < 0$, indicating elastic loading/unloading, then $\bar{\boldsymbol{\sigma}}_{n+1} = \bar{\boldsymbol{\sigma}}^{\text{trial}}$, the final stress is simply

$$\boldsymbol{\sigma}_{n+1} = \left(1 - d_{c,n}\right)\left(1 - s d_{t,n}\right) \bar{\boldsymbol{\sigma}}^{\text{trial}}. \tag{9.20}$$

The corresponding tangent stiffness is then

$$\begin{aligned}\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} &= \frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \bar{\boldsymbol{\sigma}}_{n+1}} : \frac{\partial \bar{\boldsymbol{\sigma}}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ &= (1-d_{c,n})\, d_{t,n}\, (s_0-1)\, \bar{\boldsymbol{\sigma}}^{\mathrm{trial}} \otimes \frac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}} : \boldsymbol{D} + (1-d_{c,n})\,(1-s d_{t,n})\, \boldsymbol{D} \\ &= (1-d_{c,n}) \left(d_{t,n}\, (s_0-1)\, \bar{\boldsymbol{\sigma}}^{\mathrm{trial}} \otimes \frac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}} + (1-s d_{t,n})\, \mathbb{I} \right) : \boldsymbol{D}.\end{aligned} \tag{9.21}$$

Plasticity Evolution

The yield function and the damage evolutions are three local equations shall be satisfied.

$$\boldsymbol{R} = \begin{cases} \alpha \bar{I}_1 + \sqrt{\frac{3}{2}} \left\| \bar{\boldsymbol{s}}^{\mathrm{trial}} \right\| - \gamma\sqrt{6}G + \beta \left\langle \bar{\sigma}_1^{\mathrm{trial}} - \gamma\,(2G\hat{n}_1 + 3K\alpha_p) \right\rangle + (1-\alpha)\, \bar{f}_c, \\ \kappa_{t,n} + \gamma r \dfrac{f_t}{g_t}\,(\hat{n}_1 + \alpha_p) - \kappa_t, \\ \kappa_{c,n} + \gamma\,(1-r)\,\dfrac{f_c}{g_c}\,(\hat{n}_3 + \alpha_p) - \kappa_c. \end{cases} \tag{9.22}$$

By choosing $\boldsymbol{x} = \begin{bmatrix} \gamma & \kappa_t & \kappa_c \end{bmatrix}^{\mathrm{T}}$ as the independent variables and assuming $\hat{\boldsymbol{n}} = \dfrac{\hat{\boldsymbol{s}}^{\mathrm{trial}}}{\left\| \hat{\boldsymbol{s}}^{\mathrm{trial}} \right\|}$ that is a function of $\boldsymbol{\varepsilon}_{n+1}$ only thus does not contain γ, the Jacobian can be computed as

$$\boldsymbol{J} = \begin{bmatrix} -9K\alpha\alpha_p - \sqrt{6}G - \beta\,(2G\hat{n}_1 + 3K\alpha_p)\, H\,(\hat{\sigma}_1) & \langle \hat{\sigma}_1 \rangle \dfrac{\partial \beta}{\partial \kappa_t} & (1-\alpha)\, \bar{f}_c' + \langle \hat{\sigma}_1 \rangle \dfrac{\partial \beta}{\partial \kappa_c} \\ f_t \dfrac{\hat{n}_1 + \alpha_p}{g_t} \left(r + \gamma \dfrac{\partial r}{\partial \gamma} \right) & \gamma r \dfrac{\hat{n}_1 + \alpha_p}{g_t} f_t' - 1 & \cdot \\ f_c \dfrac{\hat{n}_3 + \alpha_p}{g_c} \left(1 - r - \gamma \dfrac{\partial r}{\partial \gamma} \right) & \cdot & \gamma\,(1-r)\, \dfrac{\hat{n}_3 + \alpha_p}{g_c} f_c' - 1 \end{bmatrix}, \tag{9.23}$$

where $H\,(\cdot)$ is the heaviside function and

$$\begin{aligned} \frac{\partial \beta}{\partial \kappa_t} &= (1-\alpha)\, \frac{\bar{f}_c}{\bar{f}_t^2} \bar{f}_t', \\ \frac{\partial \beta}{\partial \kappa_c} &= (\alpha - 1)\, \frac{1}{\bar{f}_t} \bar{f}_c'. \end{aligned}$$

In the explicit form, if $\hat{\sigma}_1 > 0$,

$$\boldsymbol{J} = \begin{bmatrix} -9K\alpha\alpha_p - \sqrt{6}G - \beta\left(2G\hat{n}_1 + 3K\alpha_p\right) & (1-\alpha)\dfrac{\bar{f}_c\hat{\sigma}_1}{\bar{f}_t^2}\bar{f}_t' & (1-\alpha)\left(1-\dfrac{\hat{\sigma}_1}{\bar{f}_t}\right)\bar{f}_c' \\ f_t\dfrac{\hat{n}_1+\alpha_p}{g_t}\left(r+\gamma\dfrac{\partial r}{\partial\gamma}\right) & r\gamma\dfrac{\hat{n}_1+\alpha_p}{g_t}f_t' - 1 & \cdot \\ f_c\dfrac{\hat{n}_3+\alpha_p}{g_c}\left(1-r-\gamma\dfrac{\partial r}{\partial\gamma}\right) & \cdot & (1-r)\,\gamma\dfrac{\hat{n}_3+\alpha_p}{g_c}f_c' - 1 \end{bmatrix}, \tag{9.24}$$

otherwise,

$$\boldsymbol{J} = \begin{bmatrix} -9K\alpha\alpha_p - \sqrt{6}G & \cdot & (1-\alpha)\,\bar{f}_c' \\ f_t\dfrac{\hat{n}_1+\alpha_p}{g_t}\left(r+\gamma\dfrac{\partial r}{\partial\gamma}\right) & r\gamma\dfrac{\hat{n}_1+\alpha_p}{g_t}f_t' - 1 & \cdot \\ f_c\dfrac{\hat{n}_3+\alpha_p}{g_c}\left(1-r-\gamma\dfrac{\partial r}{\partial\gamma}\right) & \cdot & (1-r)\,\gamma\dfrac{\hat{n}_3+\alpha_p}{g_c}f_c' - 1 \end{bmatrix}. \tag{9.25}$$

9.1.4. Damage Formulation

There is no local iteration required in the damage part. Once κ_t and κ_c are determined, the effective part $\bar{\boldsymbol{\sigma}}$ can be determined. Damage measures d_t and d_c can be computed accordingly.

9.1.5. Consistent Tangent Stiffness

In order to take derivatives with regard to trial strain, one can replace $\left\|\hat{\boldsymbol{s}}^{\text{trial}}\right\|$ and $\left\|\bar{\boldsymbol{s}}^{\text{trial}}\right\|$, which yields

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{cases} 3K\alpha\mathbf{1} + \sqrt{6}G\boldsymbol{n} + H\left(\hat{\sigma}_1\right)\beta\dfrac{\mathrm{d}\hat{\sigma}_1}{\mathrm{d}\bar{\boldsymbol{\sigma}}} : \dfrac{\partial\bar{\boldsymbol{\sigma}}}{\partial\boldsymbol{\varepsilon}_{n+1}}, \\ \gamma\dfrac{f_t}{g_t}\left(r\dfrac{\partial\hat{n}_1}{\partial\boldsymbol{\varepsilon}_{n+1}} + \left(\hat{n}_1+\alpha_p\right)\dfrac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}} : \dfrac{\partial\bar{\boldsymbol{\sigma}}}{\partial\boldsymbol{\varepsilon}_{n+1}}\right), \\ \gamma\dfrac{f_c}{g_c}\left(\left(1-r\right)\dfrac{\partial\hat{n}_3}{\partial\boldsymbol{\varepsilon}_{n+1}} - \left(\hat{n}_3+\alpha_p\right)\dfrac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}} : \dfrac{\partial\bar{\boldsymbol{\sigma}}}{\partial\boldsymbol{\varepsilon}_{n+1}}\right). \end{cases} \tag{9.26}$$

so that

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} \dfrac{\mathrm{d}\gamma}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\mathrm{d}\kappa_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\mathrm{d}\kappa_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \end{bmatrix} = -\left(\frac{\partial\boldsymbol{R}}{\partial\boldsymbol{x}}\right)^{-1}\frac{\partial\boldsymbol{R}}{\partial\boldsymbol{\varepsilon}_{n+1}}. \tag{9.27}$$

The effective stress $\bar{\boldsymbol{\sigma}}_{n+1}$ only depends on $\boldsymbol{\varepsilon}_{n+1}$ and γ. The partial derivative is

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{\sigma}}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} &= \frac{\partial}{\partial \boldsymbol{\varepsilon}_{n+1}}\left(\boldsymbol{s}^{\text{trial}} - 2G\gamma\frac{\boldsymbol{s}^{\text{trial}}}{\|\boldsymbol{s}^{\text{trial}}\|} + \left(p^{\text{trial}} - 3K\alpha_p\gamma\right)\mathbf{1}\right)\\ &= \boldsymbol{D} - \frac{4G^2\gamma}{\|\boldsymbol{s}^{\text{trial}}\|}\left(\mathbb{I}^{\text{dev}} - \boldsymbol{n}\otimes\boldsymbol{n}\right).\end{aligned} \tag{9.28}$$

The full derivative is

$$\begin{aligned}\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}\left(\boldsymbol{s}^{\text{trial}} - 2G\gamma\frac{\boldsymbol{s}^{\text{trial}}}{\|\boldsymbol{s}^{\text{trial}}\|} + \left(p^{\text{trial}} - 3K\alpha_p\gamma\right)\mathbf{1}\right)\\ &= \boldsymbol{D} - \frac{4G^2\gamma}{\|\boldsymbol{s}^{\text{trial}}\|}\left(\mathbb{I}^{\text{dev}} - \boldsymbol{n}\otimes\boldsymbol{n}\right) - (2G\boldsymbol{n} + 3K\alpha_p\mathbf{1})\otimes\frac{\partial\gamma}{\partial\boldsymbol{\varepsilon}_{n+1}}\\ &= \frac{\partial\bar{\boldsymbol{\sigma}}_{n+1}}{\partial\boldsymbol{\varepsilon}_{n+1}} - (2G\boldsymbol{n} + 3K\alpha_p\mathbf{1})\otimes\frac{\partial\gamma}{\partial\boldsymbol{\varepsilon}_{n+1}}.\end{aligned} \tag{9.29}$$

The derivative of the damage factor can be expressed as

$$\begin{aligned}\frac{\mathrm{d}h}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} &= (1-d_c)\frac{\mathrm{d}\left(1-sd_t\right)}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} + (1-sd_t)\frac{\mathrm{d}\left(1-d_c\right)}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}\\ &= (d_c-1)\left(s\frac{\mathrm{d}d_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} + d_t\frac{\mathrm{d}s}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}\right) + (sd_t-1)\frac{\mathrm{d}d_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}},\end{aligned} \tag{9.30}$$

with

$$\frac{\mathrm{d}d_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} = \frac{\mathrm{d}d_t}{\mathrm{d}\kappa_t}\frac{\mathrm{d}\kappa_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}, \tag{9.31}$$

$$\frac{\mathrm{d}d_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} = \frac{\mathrm{d}d_c}{\mathrm{d}\kappa_c}\frac{\mathrm{d}\kappa_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}, \tag{9.32}$$

$$\frac{\mathrm{d}s}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} = (1-s_0)\frac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}} : \frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}. \tag{9.33}$$

The following derivatives would be useful.

$$\frac{\mathrm{d}d_\aleph}{\mathrm{d}\kappa_\aleph} = \frac{c_\aleph}{b_\aleph}\frac{a_\aleph+2}{2\sqrt{\phi_\aleph}}\Phi_\aleph^{c_\aleph/b_\aleph-1}, \tag{9.34}$$

$$\frac{\mathrm{d}f_\aleph}{\mathrm{d}\kappa_\aleph} = f_{\aleph,0}\frac{a_\aleph+2}{2\sqrt{\phi_\aleph}}\left(a_\aleph - 2\sqrt{\phi_\aleph}+1\right), \tag{9.35}$$

$$\frac{\mathrm{d}\bar{f}_\aleph}{\mathrm{d}\kappa_\aleph} = f_{\aleph,0}\frac{a_\aleph+2}{2\sqrt{\phi_\aleph}}\frac{\left(a_\aleph+1+\left(\frac{c_\aleph}{b_\aleph}-2\right)\sqrt{\phi_\aleph}\right)}{\Phi_\aleph^{c_\aleph/b_\aleph}}. \tag{9.36}$$

Given that the stress update is computed as follows,

$$\boldsymbol{\sigma}_{n+1} = h\bar{\boldsymbol{\sigma}}_{n+1}, \tag{9.37}$$

the consistent tangent stiffness is then

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\sigma}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} &= \bar{\boldsymbol{\sigma}}_{n+1} \otimes \frac{\mathrm{d}h}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} + h\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \\ &= \bar{\boldsymbol{\sigma}}_{n+1} \otimes \left((d_c - 1) \left(s\frac{\mathrm{d}d_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} + d_t\frac{\mathrm{d}s}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \right) + (sd_t - 1)\frac{\mathrm{d}d_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \right) + h\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}},\end{aligned} \tag{9.38}$$

which is equivalently,

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\sigma}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} = \bar{\boldsymbol{\sigma}}_{n+1} \otimes \underbrace{\left(\begin{bmatrix} s\,(d_c - 1)\,\dfrac{\mathrm{d}d_t}{\mathrm{d}\kappa_t} & (sd_t - 1)\,\dfrac{\mathrm{d}d_c}{\mathrm{d}\kappa_c} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\kappa_t}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\mathrm{d}\kappa_c}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}} \end{bmatrix} \right)}_{\text{dot product in vector representation}} \\ + \left(d_t\,(d_c - 1)\,(1 - s_0)\,\bar{\boldsymbol{\sigma}}_{n+1} \otimes \frac{\mathrm{d}r}{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}} + h\mathbb{I} \right) \frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_{n+1}}{\mathrm{d}\boldsymbol{\varepsilon}_{n+1}}.\end{aligned} \tag{9.39}$$

The CDP model has no kinematic hardening, the effective part resembles the bounding surface concept. Due to the coaxiality, the model can be constructed in the eigen space, which brings some convenience in terms of implementation.

9.1.6. Implementation

The state determination of the CDP model is shown in Algorithm 9. Compared with the original implementation, the presented one is much more concise and is able to avoid lengthy computation of consistent tangent stiffness.

It is worth noting some quantities remain constant in the local iteration, for example,

$$2G\hat{n}_1 + 3K\alpha_p, \qquad \frac{\hat{n}_1 + \alpha_p}{g_t}, \qquad \frac{\hat{n}_3 + \alpha_p}{g_c}. \tag{9.40}$$

They can be computed as stored before entering local iteration.

Algorithm 9 state determination of the CDP model

Parameter: λ, G
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\varepsilon}_n^p$, $\boldsymbol{\sigma}_n$, $\kappa_{\aleph,n}$
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\varepsilon}_{n+1}^p$, $\boldsymbol{\sigma}_{n+1}$, $\kappa_{\aleph,n+1}$
$\bar{\boldsymbol{\sigma}}^{\text{trial}} = \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p)$
$\boldsymbol{s}^{\text{trial}} = \operatorname{dev}\left(\bar{\boldsymbol{\sigma}}^{\text{trial}}\right)$
perform eigenanalysis on $\bar{\boldsymbol{\sigma}}^{\text{trial}}$ and compute $\boldsymbol{T}$, $\bar{\boldsymbol{s}}^{\text{trial}}$
$\hat{\boldsymbol{n}} = \dfrac{\bar{\boldsymbol{s}}^{\text{trial}}}{\|\bar{\boldsymbol{s}}^{\text{trial}}\|}$
compute f^{trial} ▷ Eq. (9.19)
$\kappa_{\aleph,n+1} = \kappa_{\aleph,n}$

if $f^{\text{trial}} \geqslant 0$ **then** ▷ plasticity evolution
  compute $f_{\aleph}$, $\bar{f}_{\aleph}$ and $d_{\aleph}$ and their derivatives ▷ This can be an independent overridable method.
  while true **do**
    $\gamma = 0$
    compute $\boldsymbol{R}$ and $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$ ▷ Eq. (9.22) and Eq. (9.23)
    $\Delta = \left(\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \boldsymbol{R}$ ▷ $\Delta = \begin{bmatrix}\delta\gamma & \delta\kappa_t & \delta\kappa_c\end{bmatrix}$
    if $\|\Delta\| <$ tolerance **then**
      break
    end if
    $\gamma \leftarrow \gamma - \delta\gamma$
    $\kappa_{t,n+1} \leftarrow \kappa_{t,n+1} - \delta\kappa_t$
    $\kappa_{t,n+1} \leftarrow \kappa_{t,n+1} - \delta\kappa_c$
  end while
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}} - \gamma\left(2G\boldsymbol{n} + 3K\alpha_p\mathbf{1}\right)$
  $\boldsymbol{\varepsilon}^p_{n+1} = \boldsymbol{\varepsilon}^p_n + \gamma\left(\boldsymbol{n} + \alpha_p\mathbf{1}\right)$
  compute $\boldsymbol{D}_{n+1}$ ▷ Eq. (9.39)
else ▷ elastic loading/unloading
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}$
  $\boldsymbol{\varepsilon}^p_{n+1} = \boldsymbol{\varepsilon}^p_n$
  compute $\boldsymbol{D}_{n+1}$ ▷ Eq. (9.21)
end if

The CPP implementation of state determination can be found as follows.

```
int NonlinearCDP::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= datum::eps) return SUANPAN_SUCCESS;

    trial_history = current_history;
    auto& d_t = trial_history(0);
    auto& d_c = trial_history(1);
    auto& kappa_t = trial_history(2);
    auto& kappa_c = trial_history(3);
    vec plastic_strain(&trial_history(4), 6, false, true);

    const auto& current_kappa_t = current_history(2);
    const auto& current_kappa_c = current_history(3);

    trial_stress = (trial_stiffness = initial_stiffness) * (trial_strain - plastic_strain); // 6

    vec principal_stress;    // 3
    mat principal_direction; // 3x3
    if(!eig_sym(principal_stress, principal_direction, tensor::stress::to_tensor(trial_stress), "std")) return SUANPAN_FAIL;

    const auto trans = transform::compute_jacobian_nominal_to_principal(principal_direction);
```

```
23
24      const auto s = tensor::dev(trial_stress);    // 6
25      const auto norm_s = tensor::stress::norm(s); // 1
26      vec n = s / norm_s;                          // 6
27      if(!n.is_finite()) n.zeros();
28
29      const auto ps = tensor::dev(principal_stress); // 3
30      const vec pn = normalise(ps);                  // 3
31
32      const vec dsigmadlambda = -double_shear * pn - three_alpha_p_bulk; // 6
33
34      const auto dgdsigma_t = (pn(2) + alpha_p) / g_t;
35      const auto dgdsigma_c = (pn(0) + alpha_p) / g_c;
36
37      auto new_stress = principal_stress;     // converged principal stress
38      const auto& max_stress = new_stress(2); // algebraically maximum principal stress
39
40      const auto const_yield = alpha * accu(principal_stress) + root_three_two * norm_s;
41
42      vec residual(3), incre;
43      mat jacobian(3, 3, fill::zeros);
44      mat left(3, 6);
45
46      podarray<double> t_para, c_para;
47
48      auto lambda = 0., ref_error = 0.;
49      double r, beta;
50      vec dr;
51
52      unsigned counter = 0;
53      while(true) {
54          if(max_iteration == ++counter) {
55              suanpan_error("NonlinearCDP cannot converge within %u iterations.\n",
                ↪  max_iteration);
56              return SUANPAN_FAIL;
57          }
58
59          t_para = compute_tension_backbone(kappa_t);
60          c_para = compute_compression_backbone(kappa_c);
61
62          const auto tension_flag = max_stress > 0.;
63
64          beta = -one_minus_alpha * c_para(2) / t_para(2) - alpha - 1.;
65
66          residual(0) = const_yield + pfplambda * lambda + one_minus_alpha * c_para(2);
67
68          if(tension_flag) residual(0) += beta * max_stress;
69
70          r = compute_r(new_stress);
71
72          if(1 == counter && residual(0) < 0.) {
73              const auto damage_c = scale * d_c - 1.;
74              const auto damage_t = compute_s(r) * scale * d_t - 1.;
75              const auto damage = damage_c * damage_t;
76              trial_stiffness = (damage * eye(6, 6) + damage_c * scale * d_t * (1. - s0) *
                ↪  trial_stress * compute_dr(new_stress).t() * trans) * initial_stiffness;
77              trial_stress *= damage;
```

```
            return SUANPAN_SUCCESS;
        }

        const auto t_term = t_para(1) * dgdsigma_t;
        const auto c_term = c_para(1) * dgdsigma_c;

        residual(1) = r * t_term * lambda + current_kappa_t - kappa_t;
        residual(2) = (c_term - r * c_term) * lambda + current_kappa_c - kappa_c;

        if(tension_flag) {
            jacobian(0, 0) = pfplambda + beta * dsigmadlambda(2);
            const auto tmp_term = one_minus_alpha * max_stress / t_para(2);
            jacobian(0, 1) = tmp_term * c_para(2) / t_para(2) * t_para(5);
            jacobian(0, 2) = (one_minus_alpha - tmp_term) * c_para(5);
        }
        else {
            jacobian(0, 0) = pfplambda;
            jacobian(0, 1) = 0.;
            jacobian(0, 2) = one_minus_alpha * c_para(5);
        }

        const auto dlambda = r + lambda * dot(dr = compute_dr(new_stress), dsigmadlambda);
        jacobian(1, 0) = t_term * dlambda;
        jacobian(2, 0) = c_term - c_term * dlambda;
        jacobian(1, 1) = r * lambda * dgdsigma_t * t_para(4) - 1.;
        jacobian(2, 2) = (lambda - r * lambda) * dgdsigma_c * c_para(4) - 1.;

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        auto error = norm(residual);
        if(1 == counter) ref_error = std::max(1., error);
        suanpan_debug("NonlinearCDP local iteration error: %.5E.\n", error /= ref_error);
        if(error <= tolerance || norm(incre) <= tolerance) break;

        lambda -= incre(0);
        kappa_t -= incre(1);
        kappa_c -= incre(2);
        new_stress -= dsigmadlambda * incre(0);

        if(kappa_t > 1.) kappa_t = .999; // avoid overshoot
        if(kappa_c > 1.) kappa_c = .999; // avoid overshoot
    }

    // update damage indices
    d_t = t_para(0);
    d_c = c_para(0);
    // update plastic strain
    plastic_strain += lambda * (n % tensor::stress::norm_weight + unit_alpha_p);

    const auto recovery = compute_s(r);
    const auto damage_c = scale * d_c - 1.;
    const auto damage_t = recovery * scale * d_t - 1.;
    const auto damage = damage_c * damage_t;

    // update trial stress
    trial_stress = transform::compute_jacobian_principal_to_nominal(principal_direction) *
    ↪  new_stress;
```

```
134
135     const mat dnde = double_shear / norm_s * (unit_dev_tensor - n * n.t());
136
137     // \dfrac{\partial\bar{\sigma}}{\partial\varepsilon^{tr}}
138     trial_stiffness -= double_shear * lambda * dnde;
139
140     const rowvec drdsigma = dr.t() * trans;
141     const rowvec prpe = drdsigma * trial_stiffness;
142
143     // compute local derivatives
144     left.row(0) = 3. * alpha * bulk * tensor::unit_tensor2.t() + root_three_two *
        ↪  double_shear * n.t();
145     left.row(1) = t_para(1) * lambda * (r / g_t * trans.row(2) * dnde + dgdsigma_t * prpe);
146     left.row(2) = c_para(1) * lambda * ((1. - r) / g_c * trans.row(0) * dnde - dgdsigma_c *
        ↪  prpe);
147
148     if(max_stress > 0.) left.row(0) += beta * trans.row(2) * trial_stiffness;
149
150     const mat right = -solve(jacobian, left);
151     const auto& dlambdade = right.row(0);
152     const auto& dkappade = right.rows(1, 2);
153
154     // \dfrac{\mathrm{d}\bar{\sigma}}{\mathrm{d}\varepsilon^{tr}}
155     trial_stiffness -= (double_shear * n + three_alpha_p_bulk * tensor::unit_tensor2) *
        ↪  dlambdade;
156
157     trial_stiffness = (damage * eye(6, 6) + scale * d_t * damage_c * (1. - s0) * trial_stress
        ↪  * drdsigma) * trial_stiffness + trial_stress * scale * rowvec{recovery * damage_c *
        ↪  t_para(3), damage_t * c_para(3)} * dkappade;
158
159     trial_stress *= damage;
160
161     return SUANPAN_SUCCESS;
162 }
```

9.2. CDPM2 Model

The CDP adopts an isotropic damage, which leads to, for example, degradation of compressive/tensile strength due to tensile/compressive damage. In cyclic loading cases, it may not be ideal. The CDPM2 model uses a different approach that applies tensile damage to tensile part of stress and compressive damage to compressive part of stress.

Compared to the original formulation [10], here the dependency of the Lode angle is removed for brevity.

9.2.1. Plasticity

The model is driven by the norm of the deviatoric stress $s = \|\boldsymbol{s}\|$, the hydrostatic stress p and the internal hardening variable κ_p.

Yield Function

The yield function F is defined as

$$F = g_1^2 + m_0 q_{h1}^2 q_{h2} g_3 - q_{h1}^2 q_{h2}^2, \tag{9.41}$$

where the friction parameter m_0

$$m_0 = \frac{3f_c^2 - 3f_t^2}{f_c f_t} \frac{e}{1+e} \tag{9.42}$$

depends on material strengths f_c and f_t, e is the eccentricity constant which depends on the previous two strengths and equibiaxial compression strength f_{bc}. The hardening functions q_{h1} and q_{h2} will be introduced later. The helper functions g_1 and g_3 are also used to define the plastic potential G. If the Lode angle shall be considered, g_3 needs to be replaced by g_4 which is defined as

$$g_4 = \frac{s}{\sqrt{6} f_c} r + \frac{p}{f_c}, \tag{9.43}$$

with $r = r(\theta)$ being a function of the lode angle θ.

The partial derivatives of F are

$$\frac{\partial F}{\partial p} = 2g_1 \frac{\partial g_1}{\partial p} + m_0 q_{h1}^2 q_{h2} \frac{\partial g_3}{\partial p}, \tag{9.44}$$

$$\frac{\partial F}{\partial s} = 2g_1 \frac{\partial g_1}{\partial s} + m_0 q_{h1}^2 q_{h2} \frac{\partial g_3}{\partial s}, \tag{9.45}$$

$$\frac{\partial F}{\partial \kappa_p} = 2g_1 \frac{\partial g_1}{\partial \kappa_p} + 2m_0 q_{h1} q_{h2} g_3 \frac{\mathrm{d}q_{h1}}{\mathrm{d}\kappa_p} + m_0 q_{h1}^2 g_3 \frac{\mathrm{d}q_{h2}}{\mathrm{d}\kappa_p} - 2q_{h1} q_{h2} \left(q_{h2} \frac{\mathrm{d}q_{h1}}{\mathrm{d}\kappa_p} + q_{h1} \frac{\mathrm{d}q_{h2}}{\mathrm{d}\kappa_p} \right). \tag{9.46}$$

Flow Rule

The plastic potential G is defined as

$$G = g_1^2 + q_{h1}^2 g_2, \tag{9.47}$$

where

$$g_1 = (1 - q_{h1})\, g_3^2 + \sqrt{\frac{3}{2}}\frac{s}{f_c}, \qquad g_2 = \frac{m_0 s}{\sqrt{6} f_c} + \frac{m_g}{f_c}, \qquad g_3 = \frac{s}{\sqrt{6} f_c} + \frac{p}{f_c}, \tag{9.48}$$

$$m_g = A_g B_g f_c \exp\left(\frac{3p - q_{h2} f_t}{3 B_g f_c}\right), \tag{9.49}$$

$$A_g = \frac{3 f_t q_{h2}}{f_c} + \frac{m_0}{2}, \tag{9.50}$$

$$B_g = \frac{q_{h2}\,(1 + f_t/f_c)\,/3}{\ln A_g - \ln(2D_f - 1) - \ln(3q_{h2} + m_0/2) + \ln(D_f + 1)}. \tag{9.51}$$

With the derivatives of auxiliary functions expressed as

$$\frac{\partial g_3}{\partial p} = \frac{1}{f_c}, \qquad \frac{\partial g_3}{\partial s} = \frac{1}{\sqrt{6} f_c}, \qquad \frac{\partial g_2}{\partial p} = \frac{1}{f_c}\frac{\partial m_g}{\partial p}, \qquad \frac{\partial g_2}{\partial s} = \frac{m_0}{\sqrt{6} f_c}, \tag{9.52}$$

$$\frac{\partial g_1}{\partial p} = 2\,(1 - q_{h1})\, g_3 \frac{\partial g_3}{\partial p}, \qquad \frac{\partial g_1}{\partial s} = 2\,(1 - q_{h1})\, g_3 \frac{\partial g_3}{\partial s} + \sqrt{\frac{3}{2}}\frac{1}{f_c}. \tag{9.53}$$

The flow rule can be derived as

$$G_p = \frac{\partial G}{\partial p} = 2 g_1 \frac{\partial g_1}{\partial p} + q_{h1}^2 \frac{\partial g_2}{\partial p} = \frac{4\,(1 - q_{h1})\, g_1 g_3 + q_{h1}^2 A_g \exp\left(\dfrac{p - q_{h2} f_t/3}{B_g f_c}\right)}{f_c}, \tag{9.54}$$

$$G_s = \frac{\partial G}{\partial s} = 2 g_1 \frac{\partial g_1}{\partial s} + q_{h1}^2 \frac{\partial g_2}{\partial s} = \frac{4\,(1 - q_{h1})\, g_1 g_3 + 6 g_1 + m_0 q_{h1}^2}{\sqrt{6} f_c}. \tag{9.55}$$

Hardening Law

The variables q_{h1} and q_{h2} are functions of the hardening variable κ_p.

$$q_{h1} = \begin{cases} q_{h0} + (1 - q_{h0})\left(\kappa_p^3 - 3\kappa_p^2 + 3\kappa_p\right) - H_p\left(\kappa_p^3 - 3\kappa_p^2 + 2\kappa_p\right) & \text{for } \kappa_p < 1, \\ 1 & \text{for } \kappa_p \geqslant 1. \end{cases} \tag{9.56}$$

$$q_{h2} = \begin{cases} 1 & \text{for } \kappa_p < 1, \\ 1 + H_p\,(\kappa_p - 1) & \text{for } \kappa_p \geqslant 1. \end{cases} \tag{9.57}$$

The evolution of κ_p is defined as

$$\dot{\kappa_p} = \frac{\|\dot{\varepsilon_p}\|}{x_h} 4\cos^2(\theta)\,. \tag{9.58}$$

Here the dependency on the Lode angle θ is removed such that

$$x_h \dot{\kappa_p} = \|\dot{\varepsilon_p}\| = \gamma G_\kappa, \tag{9.59}$$

with $G_\kappa = \sqrt{G_s^2 + G_p^2/3}$. This effectively means $\theta = 2\pi/3$ is a constant.

Plasticity Residual

Collecting the yield function, evolutions of s, p and κ_p, the local residual of the plasticity part can be expressed as

$$\boldsymbol{R} = \begin{cases} g_1^2 + m_0 q_{h1}^2 q_{h2} g_3 - q_{h1}^2 q_{h2}^2, \\ s + 2G\gamma G_s - s^{\text{trial}}, \\ p + K\gamma G_p - p^{\text{trial}}, \\ x_h \kappa_p^n + \gamma G_\kappa - x_h \kappa_p. \end{cases} \tag{9.60}$$

The local system consists of four scalar equations.

The Jacobian thus has a size of 4 and reads

$$\boldsymbol{J} = \begin{bmatrix} . & \dfrac{\partial F}{\partial s} & \dfrac{\partial F}{\partial p} & \dfrac{\partial F}{\partial \kappa_p} \\ 2GG_s & 1 + 2G\gamma \dfrac{\partial G_s}{\partial s} & 2G\gamma \dfrac{\partial G_s}{\partial p} & 2G\gamma \dfrac{\partial G_s}{\partial \kappa_p} \\ KG_p & K\gamma \dfrac{\partial G_p}{\partial s} & 1 + K\gamma \dfrac{\partial G_p}{\partial p} & K\gamma \dfrac{\partial G_p}{\partial \kappa_p} \\ G_\kappa & \gamma \dfrac{\partial G_\kappa}{\partial s} & \left(\kappa_p^n - \kappa_p\right) \dfrac{\mathrm{d}x_h}{\mathrm{d}p} + \gamma \dfrac{\partial G_\kappa}{\partial p} & \gamma \dfrac{\partial G_\kappa}{\partial \kappa_p} - x_h \end{bmatrix}. \tag{9.61}$$

9.2.2. Damage

Equivalent Strain

The equivalent strain is defined as

$$\tilde{\varepsilon} = g_5 + \sqrt{g_5^2 + g_6^2}, \tag{9.62}$$

with

$$g_5 = \frac{\varepsilon_0 m_0}{2} g_3, \qquad g_6 = \sqrt{\frac{3}{2}} \frac{s}{f_c}. \tag{9.63}$$

Again, if the Lode angle needs to be considered, replace g_3 with g_4. The equivalent strain is further split into tensile and compressive parts.

$$\tilde{\varepsilon}_t = \alpha_t \tilde{\varepsilon}, \qquad \tilde{\varepsilon}_c = \alpha_c \tilde{\varepsilon}, \tag{9.64}$$

where $\alpha_t + \alpha_c = 1$ are two parameters that characterise the tensile/compressive portion of the current loading step.

The damage history variables κ_{dt} and κ_{dc} track the maximum values of $\tilde{\varepsilon}_t$ and $\tilde{\varepsilon}_c$, respectively.

Tension

The inelastic strain is expressed as a function of damage factors.

$$\varepsilon_i = \kappa_{dt1} + \omega_t \kappa_{dt2}. \tag{9.65}$$

While the uniaxial stress response can be expressed as

$$\sigma = (1 - \omega_t) E \kappa_{dt}. \tag{9.66}$$

Assume an exponential degradation curve,

$$\sigma = f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{ft}}\right) \tag{9.67}$$

The local residual is

$$R = f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{ft}}\right) - (1 - \omega_t) E \kappa_{dt}. \tag{9.68}$$

For a given set of κ_{dt}, κ_{dt1} and κ_{dt2}, the above residual is a function of unknown ω_t. It can be solved by, for example, Newton's method. The corresponding derivatives are

$$\frac{\partial R}{\partial \omega_t} = E \kappa_{dt} - \frac{\kappa_{dt2}}{\varepsilon_{fc}} f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{ft}}\right), \tag{9.69}$$

$$\frac{\partial R}{\partial \kappa_{dt}} = -(1 - \omega_t) E, \tag{9.70}$$

$$\frac{\partial R}{\partial \kappa_{dt1}} = -\frac{1}{\varepsilon_{ft}} f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{ft}}\right), \tag{9.71}$$

$$\frac{\partial R}{\partial \kappa_{dt2}} = -\frac{\omega_t}{\varepsilon_{ft}} f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{ft}}\right). \tag{9.72}$$

Compression

For compression, the definition fully resembles its tensile counterpart. The inelastic strain is driven by two internal parameters κ_{dc1} and κ_{dc2},

$$\varepsilon_i = \kappa_{dc1} + \omega_c \kappa_{dc2}. \tag{9.73}$$

The uniaxial response is

$$\sigma = (1 - \omega_c)\, E \kappa_{dc}. \tag{9.74}$$

With the exponential backbone, the local residual is

$$R = f_t \exp\left(-\frac{\varepsilon_i}{\varepsilon_{fc}}\right) - (1 - \omega_c)\, E \kappa_{dc}. \tag{9.75}$$

Stress

The final stress can be expressed as

$$\boldsymbol{\sigma} = (1 - \omega_t)\,(1 - \omega_c)\,\bar{\boldsymbol{\sigma}}. \tag{9.76}$$

This form is identical to the one used in the CDP model. It represents the isotropic damage. Alternatively, it can also be expressed as

$$\boldsymbol{\sigma} = (1 - \omega_t)\,\bar{\boldsymbol{\sigma}}_t + (1 - \omega_c)\,\bar{\boldsymbol{\sigma}}_c, \tag{9.77}$$

which stands for the anisotropic damage. The tensile and compressive part of the stress tensor $\bar{\boldsymbol{\sigma}}_t$ and $\bar{\boldsymbol{\sigma}}_c$ are obtained by performing an eigenanalysis such that

$$\bar{\boldsymbol{\sigma}}_t = \sum_{i=1}^{3} \langle \hat{\sigma}_i \rangle\, \boldsymbol{p}_i \otimes \boldsymbol{p}_i, \qquad \bar{\boldsymbol{\sigma}}_c = \bar{\boldsymbol{\sigma}} - \bar{\boldsymbol{\sigma}}_t, \tag{9.78}$$

where $\hat{\sigma}_i$ is the eigenvalue of $\bar{\boldsymbol{\sigma}}$ and $\boldsymbol{p}_i$ is the associated eigenvector.

Eq. (9.77) can be equivalently expressed as

$$\begin{aligned} \boldsymbol{\sigma} &= (1 - \omega_t)\,\bar{\boldsymbol{\sigma}}_t + (1 - \omega_c)\,(\bar{\boldsymbol{\sigma}} - \bar{\boldsymbol{\sigma}}_t) \\ &= (1 - \omega_c)\,\bar{\boldsymbol{\sigma}} + (\omega_c - \omega_t)\,\bar{\boldsymbol{\sigma}}_t. \end{aligned} \tag{9.79}$$

With the above expression, the tangent stiffness can be expressed as

$$\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\boldsymbol{\varepsilon}} = (1 - \omega_c)\,\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}}{\mathrm{d}\boldsymbol{\varepsilon}} + (\omega_c - \omega_t)\,\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}_t}{\mathrm{d}\bar{\boldsymbol{\sigma}}}\frac{\mathrm{d}\bar{\boldsymbol{\sigma}}}{\mathrm{d}\boldsymbol{\varepsilon}} - \bar{\boldsymbol{\sigma}} \otimes \frac{\mathrm{d}\omega_c}{\mathrm{d}\boldsymbol{\varepsilon}} + \bar{\boldsymbol{\sigma}}_t \otimes \left(\frac{\mathrm{d}\omega_c}{\mathrm{d}\boldsymbol{\varepsilon}} - \frac{\mathrm{d}\omega_t}{\mathrm{d}\boldsymbol{\varepsilon}}\right). \tag{9.80}$$

9.2.3. Formulation

9.2.4. Implementation

The main body of state determination consists of two main tasks: 1) compute the plasticity part and 2) compute the damage part. The implementation can be found as follows.

```
int CDPM2::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    trial_history = current_history;
    const auto& current_kp = current_history(0);
    auto& kp = trial_history(0);
    vec plastic_strain(&trial_history(1), 6, false, true);

    trial_stress = (trial_stiffness = initial_stiffness) * (trial_strain - plastic_strain);

    //
    // plasticity part
    //

    const auto dev_stress = tensor::dev(trial_stress);
    const auto hydro_stress = tensor::mean3(trial_stress);
    const auto trial_s = tensor::stress::norm(dev_stress);
    const auto trial_p = hydro_stress;
    const vec n = dev_stress / trial_s;

    auto gamma = 0., s = trial_s, p = trial_p;

    mat jacobian(4, 4, fill::none), left(4, 6, fill::zeros);
    jacobian(0, 0) = 0.;

    vec residual(4), incre;

    podarray<double> data(15);
    const auto& f = data(0);
    const auto& pfps = data(1);
    const auto& pfpp = data(2);
    const auto& pfpkp = data(3);
    const auto& gs = data(4);
    const auto& gp = data(5);
    const auto& gg = data(6);
    const auto& pgsps = data(7);
    const auto& pgspp = data(8);
    const auto& pgspkp = data(9);
    const auto& pgpps = data(10);
    const auto& pgppp = data(11);
    const auto& pgppkp = data(12);
    const auto& xh = data(13);
    const auto& dxhdp = data(14);

```

```
47      auto counter = 0u;
48
49      while(true) {
50          if(max_iteration == ++counter) return SUANPAN_FAIL;
51
52          compute_plasticity(s, p, kp, data);
53
54          if(1 == counter && f < 0.) break;
55
56          residual(0) = f;
57          residual(1) = s + double_shear * gamma * gs - trial_s;
58          residual(2) = p + bulk * gamma * gp - trial_p;
59          residual(3) = xh * (current_kp - kp) + gamma * gg;
60
61          jacobian(0, 1) = pfps;
62          jacobian(0, 2) = pfpp;
63          jacobian(0, 3) = pfpkp;
64
65          jacobian(1, 0) = double_shear * gs;
66          jacobian(1, 1) = double_shear * gamma * pgsps + 1.;
67          jacobian(1, 2) = double_shear * gamma * pgspp;
68          jacobian(1, 3) = double_shear * gamma * pgspkp;
69
70          jacobian(2, 0) = bulk * gp;
71          jacobian(2, 1) = bulk * gamma * pgpps;
72          jacobian(2, 2) = bulk * gamma * pgppp + 1.;
73          jacobian(2, 3) = bulk * gamma * pgppkp;
74
75          jacobian(3, 0) = gg;
76          jacobian(3, 1) = gamma / gg * (gs * pgsps + gp / 3. * pgpps);
77          jacobian(3, 2) = gamma / gg * (gs * pgspp + gp / 3. * pgppp) + (current_kp - kp) *
            ↪ dxhdp;
78          jacobian(3, 3) = gamma / gg * (gs * pgspkp + gp / 3. * pgppkp) - xh;
79
80          if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;
81
82          const auto error = norm(residual);
83          suanpan_debug("CDPM2 local plasticity iteration error: %.5E.\n", error);
84
85          if(error <= tolerance) {
86              const vec unit_n = n % tensor::stress::norm_weight;
87
88              plastic_strain += gamma * gs * unit_n + gamma * gp / 3. * tensor::unit_tensor2;
89
90              trial_stress = s * n + p * tensor::unit_tensor2;
91
92              mat right(4, 6, fill::none);
93
94              right.row(0).zeros();
95              right.row(1) = double_shear * unit_n.t() * unit_dev_tensor;
96              right.row(2) = bulk * tensor::unit_tensor2.t();
97              right.row(3).zeros();
98
99              if(!solve(left, jacobian, right)) return SUANPAN_FAIL;
100
101             trial_stiffness = n * (left.row(1) - s / trial_s * right.row(1)) + s / trial_s *
                ↪ double_shear * unit_dev_tensor;
```

```
            trial_stiffness.row(0) += left.row(2);
            trial_stiffness.row(1) += left.row(2);
            trial_stiffness.row(2) += left.row(2);

            break;
        }

        gamma -= incre(0);
        s -= incre(1);
        p -= incre(2);
        kp -= incre(3);
    }

    //
    // damage part
    //

    vec principal_stress;    // 3
    mat principal_direction; // 3x3
    if(!eig_sym(principal_stress, principal_direction,
    ↪  tensor::stress::to_tensor(trial_stress), "std")) return SUANPAN_FAIL;

    vector<uword> tp, cp;
    tp.reserve(3);
    cp.reserve(3);
    for(auto I = 0llu; I < 3llu; ++I)
        if(principal_stress(I) > 0.) tp.emplace_back(I);
        else cp.emplace_back(I);

    const uvec t_pattern(tp), c_pattern(cp);

    const auto aca = accu(square(principal_stress(c_pattern)));
    const auto acb = accu(square(principal_stress));
    const auto ac = aca / acb;
    rowvec daca = 2. * principal_stress.t();
    daca(t_pattern).fill(0.);
    const rowvec dac = (daca - 2. * ac * principal_stress.t()) / acb;

    if(SUANPAN_SUCCESS != compute_damage(gamma, s, p, kp, ac, data)) return SUANPAN_FAIL;

    if(DamageType::NODAMAGE == damage_type) return SUANPAN_SUCCESS;

    const auto& omegat = trial_history(16);
    const auto& omegac = trial_history(17);
    const rowvec pot(&data(0), 4, false, true);
    const rowvec poc(&data(4), 4, false, true);
    const auto& pocpac = data(8);

    const rowvec potpe = pot * left;
    const rowvec pocpe = poc * left + pocpac * dac *
    ↪  transform::compute_jacobian_nominal_to_principal(principal_direction) *
    ↪  trial_stiffness;

    if(DamageType::ISOTROPIC == damage_type) {
        trial_stiffness *= (1. - omegat) * (1. - omegac);
        trial_stiffness -= trial_stress * ((1. - omegat) * pocpe + (1. - omegac) * potpe);

```

```
156         trial_stress *= (1. - omegat) * (1. - omegac);
157     }
158     else if(DamageType::ANISOTROPIC == damage_type) {
159         const auto compute_fraction = [&](const double a, const double b) { return
            ↪ suanpan::approx_equal(a, b, 4) ? a + b <= 0. ? 0. : 2. : 2. * (suanpan::ramp(a) -
            ↪ suanpan::ramp(b)) / (a - b); };
160         const auto get_fraction = [&](const vec& p_stress) { return
            ↪ vec{compute_fraction(p_stress(0), p_stress(1)), compute_fraction(p_stress(1),
            ↪ p_stress(2)), compute_fraction(p_stress(2), p_stress(0))}; };
161 
162         const mat pnn = [](const mat& eig_vec) {
163             const mat n12 = eig_vec.col(0) * eig_vec.col(1).t();
164             const mat n23 = eig_vec.col(1) * eig_vec.col(2).t();
165             const mat n31 = eig_vec.col(2) * eig_vec.col(0).t();
166 
167             mat pij(6, 6);
168 
169             pij.col(0) = tensor::stress::to_voigt(eig_vec.col(0) * eig_vec.col(0).t());
170             pij.col(1) = tensor::stress::to_voigt(eig_vec.col(1) * eig_vec.col(1).t());
171             pij.col(2) = tensor::stress::to_voigt(eig_vec.col(2) * eig_vec.col(2).t());
172             pij.col(3) = tensor::stress::to_voigt(.5 * (n12 + n12.t()));
173             pij.col(4) = tensor::stress::to_voigt(.5 * (n23 + n23.t()));
174             pij.col(5) = tensor::stress::to_voigt(.5 * (n31 + n31.t()));
175 
176             return pij;
177         }(principal_direction);
178 
179         mat tension_projector = pnn.cols(t_pattern) * pnn.cols(t_pattern).t();
180         mat tension_derivative = tension_projector + pnn.tail_cols(3) *
            ↪ diagmat(get_fraction(principal_stress)) * pnn.tail_cols(3).t();
181 
182         tension_projector.tail_cols(3) *= 2.;
183         tension_derivative.tail_cols(3) *= 2.;
184 
185         const vec tension_stress = tension_projector * trial_stress;
186 
187         trial_stiffness = (1. - omegac) * trial_stiffness - trial_stress * pocpe + (omegac -
            ↪ omegat) * tension_derivative * trial_stiffness + tension_stress * (pocpe -
            ↪ potpe);
188 
189         trial_stress *= 1. - omegac;
190         trial_stress += (omegac - omegat) * tension_stress;
191     }
192 
193     return SUANPAN_SUCCESS;
194 }
```

Here, all relevant quantities (mainly derivatives) are stored in array `data` that is passed to the corresponding functions. For example, in order to compute plasticity, the following method is used. It is lengthy and tedious, mainly computing derivatives via the chain rule.

```
void CDPM2::compute_plasticity(const double s, const double p, const double kp,
↪  podarray<double>& data) const {
    auto& f = data(0);
    auto& pfps = data(1);
    auto& pfpp = data(2);
    auto& pfpkp = data(3);
    auto& gs = data(4);
    auto& gp = data(5);
    auto& gg = data(6);
    auto& pgsps = data(7);
    auto& pgspp = data(8);
    auto& pgspkp = data(9);
    auto& pgpps = data(10);
    auto& pgppp = data(11);
    auto& pgppkp = data(12);
    auto& xh = data(13);
    auto& dxhdp = data(14);

    auto qh1 = 1., qh2 = 1.;
    auto dqh1dkp = 0., dqh2dkp = 0.;

    if(kp < 1.) {
        qh1 = qh0 + (1. - qh0) * kp * (kp * (kp - 3.) + 3.) - hp * kp * (kp - 1.) * (kp -
        ↪  2.);
        dqh1dkp = (3. - 3. * qh0) * pow(kp - 1., 2.) - hp * (3. * kp * (kp - 2.) + 2.);
    }
    else {
        qh2 = 1. + hp * (kp - 1.);
        dqh2dkp = hp;
    }

    const auto ag = 3. * ftfc * qh2 + .5 * m0;
    const auto dagdkp = 3. * ftfc * dqh2dkp;
    const auto cg = qh2 * (1. + ftfc) / 3.;
    const auto dg = log(ag) - log(3. * qh2 + .5 * m0) + lndf;
    const auto dcgdkp = dqh2dkp * (1. + ftfc) / 3.;
    const auto ddgdkp = dagdkp / ag - dqh2dkp / (qh2 + m0 / 6.);
    const auto bg = cg / dg;
    const auto dbgdkp = (dcgdkp - bg * ddgdkp) / dg;

    const auto eg = (p / fc - qh2 * ftfc / 3.) / bg;
    const auto pegpkp = (-ftfc / 3. * dqh2dkp - eg * dbgdkp) / bg;
    const auto pmgpp = ag * exp(eg);

    const auto g3 = (s / sqrt_six + p) / fc;
    const auto g1 = (1. - qh1) * g3 * g3 + sqrt_three_two * s / fc;

    const auto pg3pp = 1. / fc;
    const auto pg3ps = pg3pp / sqrt_six;

    const auto pg2pp = pmgpp / fc;
    const auto pg2ps = m0 * pg3ps;

    const auto pg1pp = (2. - 2. * qh1) * g3 * pg3pp;
    const auto pg1ps = (2. - 2. * qh1) * g3 * pg3ps + sqrt_three_two / fc;
    const auto pg1pkp = -dqh1dkp * g3 * g3;

```

```
    f = g1 * g1 + m0 * qh1 * qh1 * qh2 * g3 - qh1 * qh1 * qh2 * qh2;

    pfpp = 2. * g1 * pg1pp + m0 * qh1 * qh1 * qh2 * pg3pp;
    pfps = 2. * g1 * pg1ps + m0 * qh1 * qh1 * qh2 * pg3ps;
    pfpkp = 2. * g1 * pg1pkp + 2. * qh1 * qh2 * (m0 * g3 * dqh1dkp - qh1 * dqh2dkp - qh2 *
    ↪  dqh1dkp) + m0 * qh1 * qh1 * g3 * dqh2dkp;

    gp = 2. * g1 * pg1pp + qh1 * qh1 * pg2pp;
    gs = 2. * g1 * pg1ps + qh1 * qh1 * pg2ps;

    gg = sqrt(gs * gs + gp * gp / 3.);

    pgppp = (4. - 4. * qh1) * (pg1pp * g3 + g1 * pg3pp) + qh1 * qh1 * ag * exp(eg) / bg / fc;
    pgpps = (4. - 4. * qh1) * (pg1ps * g3 + g1 * pg3ps);
    pgppkp = 4. * g3 * (pg1pkp - qh1 * pg1pkp - dqh1dkp * g1) + (2. * dqh1dkp * ag + qh1 *
    ↪  (dagdkp + ag * pegpkp)) * qh1 * exp(eg);

    pgppp /= fc;
    pgpps /= fc;
    pgppkp /= fc;

    pgspp = (4. - 4. * qh1) * (pg1pp * g3 + g1 * pg3pp) + 6. * pg1pp;
    pgsps = (4. - 4. * qh1) * (pg1ps * g3 + g1 * pg3ps) + 6. * pg1ps;
    pgspkp = 4. * g3 * (pg1pkp - qh1 * pg1pkp - dqh1dkp * g1) + 6. * pg1pkp + 2. * m0 * qh1 *
    ↪  dqh1dkp;

    pgspp /= sqrt_six * fc;
    pgsps /= sqrt_six * fc;
    pgspkp /= sqrt_six * fc;

    if(const auto rh = -p / fc - 1. / 3.; rh >= 0.) {
        xh = (bh - ah) * exp(-rh / ch);
        dxhdp = xh / ch / fc;
        xh += ah;
    }
    else {
        xh = eh * exp(rh / fh);
        dxhdp = -xh / fh / fc;
        xh += dh;
    }
}
```

The converged `data` is then passed to the method to compute damage factors. It is similar to the plasticity part.

```
int CDPM2::compute_damage(const double gamma, const double s, const double p, const double
↪  kp, const double ac, podarray<double>& data) {
    const auto& gs = data(4);
    const auto& gp = data(5);
    const auto& gg = data(6);
    const auto& pgsps = data(7);
    const auto& pgspp = data(8);
    const auto& pgspkp = data(9);
```

```
    const auto& pgpps = data(10);
    const auto& pgppp = data(11);
    const auto& pgppkp = data(12);

    const auto& current_ee = current_history(7);
    const auto& current_et = current_history(8);
    const auto& current_ec = current_history(9);
    const auto& current_kdt1 = current_history(12);
    const auto& current_kdc1 = current_history(13);
    const auto& current_kdt2 = current_history(14);
    const auto& current_kdc2 = current_history(15);
    auto& ee = trial_history(7);
    auto& et = trial_history(8);
    auto& ec = trial_history(9);
    auto& kdt = trial_history(10);
    auto& kdc = trial_history(11);
    auto& kdt1 = trial_history(12);
    auto& kdc1 = trial_history(13);
    auto& kdt2 = trial_history(14);
    auto& kdc2 = trial_history(15);
    auto& omegat = trial_history(16);
    auto& omegac = trial_history(17);

    // ee
    const auto ptapp = .5 * e0 * m0 / fc;
    const auto ptaps = ptapp / sqrt_six;
    const auto ptbps = sqrt_three_two * e0 / fc;
    const auto term_a = ptaps * s + ptapp * p;
    const auto term_b = ptbps * s;
    const auto term_c = sqrt(term_a * term_a + term_b * term_b);
    ee = term_a + term_c;
    const auto incre_ee = ee - current_ee;
    const auto peeps = (ptaps * ee + term_b * ptbps) / term_c;
    const auto peepp = ptapp * ee / term_c;

    // ep
    const auto ep = gamma * gg;
    const auto peppg = gg;
    const auto pepps = gamma / gg * (gs * pgsps + gp / 3. * pgpps);
    const auto peppp = gamma / gg * (gs * pgspp + gp / 3. * pgppp);
    const auto peppkp = gamma / gg * (gs * pgspkp + gp / 3. * pgppkp);

    // xs
    auto xs = 1., pxsps = 0., pxspp = 0.;
    if(p <= 0.) {
        pxspp = (sqrt_six - as * sqrt_six) / s;
        xs += pxspp * p;
        pxsps = -pxspp * p / s;
    }

    // kdt
    auto incre_kdt = 0., pkdtps = 0., pkdtpp = 0.;
    if((et = current_et + incre_ee) > kdt) {
        incre_kdt = et - kdt;
        kdt = et;
        pkdtps = peeps;
        pkdtpp = peepp;
```

```
    }

    // kdt1
    auto pkdt1pg = 0., pkdt1ps = 0., pkdt1pp = 0., pkdt1pkp = 0.;
    if(incre_kdt > 0. && kdt > e0) {
        const auto incre_kdt1 = ep / xs;
        kdt1 = current_kdt1 + incre_kdt1;
        pkdt1pg = peppg / xs;
        pkdt1ps = (pepps - incre_kdt1 * pxsps) / xs;
        pkdt1pp = (peppp - incre_kdt1 * pxspp) / xs;
        pkdt1pkp = peppkp / xs;
    }

    // kdt2
    const auto incre_kdt2 = incre_kdt / xs;
    kdt2 = current_kdt2 + incre_kdt2;
    const auto pkdt2ps = (pkdtps - incre_kdt2 * pxsps) / xs;
    const auto pkdt2pp = (pkdtpp - incre_kdt2 * pxspp) / xs;

    // kdc
    auto incre_kdc = 0., pkdcps = 0., pkdcpp = 0., pkdcpac = 0.;
    if((ec = current_ec + ac * incre_ee) > kdc) {
        incre_kdc = ec - kdc;
        kdc = ec;
        pkdcps = ac * peeps;
        pkdcpp = ac * peepp;
        pkdcpac = incre_ee;
    }

    // kdc1
    auto pkdc1pg = 0., pkdc1ps = 0., pkdc1pp = 0., pkdc1pkp = 0., pkdc1pac = 0.;
    if(incre_kdc > 0. && kdc > e0) {
        auto qh2 = 1., dqh2dkp = 0.;
        if(kp >= 1.) {
            qh2 += hp * kp - hp;
            dqh2dkp = hp;
        }

        const auto betac = sqrtdf * qh2 / s;
        const auto pbetacpkp = sqrtdf / s * dqh2dkp;
        const auto pbetacps = -betac / s;

        pkdc1pac = ep * betac / xs;
        const auto incre_kdc1 = pkdc1pac * ac;
        kdc1 = current_kdc1 + incre_kdc1;
        pkdc1pg = peppg * ac * betac / xs;
        pkdc1ps = ac / xs * (pepps * betac + ep * pbetacps - ep * betac / xs * pxsps);
        pkdc1pp = ac / xs * betac * (peppp - ep / xs * pxspp);
        pkdc1pkp = ac / xs * (peppkp * betac + ep * pbetacpkp);
    }

    // kdc2
    const auto incre_kdc2 = incre_kdc / xs;
    kdc2 = current_kdc2 + incre_kdc2;
    const auto pkdc2ps = (pkdcps - incre_kdc2 * pxsps) / xs;
    const auto pkdc2pp = (pkdcpp - incre_kdc2 * pxspp) / xs;
    const auto pkdc2pac = pkdcpac / xs;
```

```

    podarray<double> datad(3);

    if(SUANPAN_SUCCESS != compute_damage_factor(kdt, kdt1, kdt2, eft, omegat, datad)) return
    ↪ SUANPAN_FAIL;
    const auto& potpkdt = datad(0);
    const auto& potpkdt1 = datad(1);
    const auto& potpkdt2 = datad(2);

    auto& potpg = data(0);
    auto& potps = data(1);
    auto& potpq = data(2);
    auto& potpkp = data(3);

    potpg = potpkdt1 * pkdt1pg;
    potps = potpkdt * pkdtps + potpkdt1 * pkdt1ps + potpkdt2 * pkdt2ps;
    potpq = potpkdt * pkdtpp + potpkdt1 * pkdt1pp + potpkdt2 * pkdt2pp;
    potpkp = potpkdt1 * pkdt1pkp;

    if(SUANPAN_SUCCESS != compute_damage_factor(kdc, kdc1, kdc2, efc, omegac, datad)) return
    ↪ SUANPAN_FAIL;
    const auto& pocpkdc = datad(0);
    const auto& pocpkdc1 = datad(1);
    const auto& pocpkdc2 = datad(2);

    auto& pocpg = data(4);
    auto& pocps = data(5);
    auto& pocpq = data(6);
    auto& pocpkp = data(7);
    auto& pocpac = data(8);

    pocpg = pocpkdc1 * pkdc1pg;
    pocps = pocpkdc * pkdcps + pocpkdc1 * pkdc1ps + pocpkdc2 * pkdc2ps;
    pocpq = pocpkdc * pkdcpp + pocpkdc1 * pkdc1pp + pocpkdc2 * pkdc2pp;
    pocpkp = pocpkdc1 * pkdc1pkp;
    pocpac = pocpkdc * pkdcpac + pocpkdc1 * pkdc1pac + pocpkdc2 * pkdc2pac;

    return SUANPAN_SUCCESS;
}
```

Finally, the damage part involves a local iteration procedure.

```
int CDPM2::compute_damage_factor(const double kd, const double kd1, const double kd2, const
↪ double ef, double& omega, podarray<double>& data) const {
    auto& popkd = data(0);
    auto& popkd1 = data(1);
    auto& popkd2 = data(2);

    if(kd < e0) {
        popkd = 0.;
        popkd1 = 0.;
        popkd2 = 0.;
        return SUANPAN_SUCCESS;
    }
```

```
12
13      auto counter = 0u;
14      while(true) {
15          if(max_iteration == ++counter) return SUANPAN_FAIL;
16
17          const auto term_a = ft * exp(-(kd1 + omega * kd2) / ef);
18          const auto term_b = (1. - omega) * elastic_modulus;
19          const auto jacobian = elastic_modulus * kd - kd2 / ef * term_a;
20          const auto incre = (term_a - term_b * kd) / jacobian;
21
22          const auto error = fabs(incre);
23          suanpan_debug("Local damage iteration error: {:.5E}.\n", error);
24
25          if(error <= tolerance) {
26              popkd = term_b / jacobian;
27              popkd1 = term_a / ef / jacobian;
28              popkd2 = popkd1 * omega;
29              return SUANPAN_SUCCESS;
30          }
31
32          omega -= incre;
33      }
34  }
```

10. Rubber

10.1. Mooney–Rivlin Model

10.1.1. Theory

10.1.2. Formulation

10.1.3. Implementation

```
// takes green strain as input
int MooneyRivlin::update_trial_status(const vec& t_strain) {
    const vec G = weight % (trial_strain = t_strain) + tensor::unit_tensor2;

    const auto &C1 = G(0), &C2 = G(1), &C3 = G(2), &C4 = G(3), &C5 = G(4), &C6 = G(5);

    const auto I1 = C1 + C2 + C3;
    const auto I2 = C1 * C2 + C1 * C3 + C2 * C3 - C4 * C4 - C5 * C5 - C6 * C6;
    const auto I3 = std::max(datum::eps, C1 * C2 * C3 + 2. * C4 * C5 * C6 - C1 * C5 * C5 - C2
    ↪  * C6 * C6 - C3 * C4 * C4);

    const auto J3M1 = sqrt(I3) - 1.;

    const vec I2E{C2 + C3, C3 + C1, C1 + C2, -C4, -C5, -C6};
    const vec I3E{C2 * C3 - C5 * C5, C3 * C1 - C6 * C6, C1 * C2 - C4 * C4, C5 * C6 - C3 * C4,
    ↪  C6 * C4 - C1 * C5, C4 * C5 - C2 * C6};

    auto W1 = pow(I3, -one_three);
    auto W2 = two_three * I1 * pow(I3, -four_three);
    auto W3 = 2. * W1 * W1;
    auto W4 = four_three * I2 * pow(I3, -five_three);
    auto W5 = pow(I3, -.5);

    const vec J1E = W1 * I1E - W2 * I3E;
    const vec J2E = W3 * I2E - W4 * I3E;
    const vec J3E = W5 * I3E;

    trial_stress = A10 * J1E + A01 * J2E + K * J3M1 * J3E;

    mat I3EE(6, 6, fill::zeros);
    I3EE(1, 2) = I3EE(2, 1) = -2. * (I3EE(4, 4) = -2. * C1);
    I3EE(0, 2) = I3EE(2, 0) = -2. * (I3EE(5, 5) = -2. * C2);
```

```
    I3EE(0, 1) = I3EE(1, 0) = -2. * (I3EE(3, 3) = -2. * C3);
    I3EE(2, 3) = I3EE(3, 2) = -2. * (I3EE(4, 5) = I3EE(5, 4) = 2. * C4);
    I3EE(0, 4) = I3EE(4, 0) = -2. * (I3EE(3, 5) = I3EE(5, 3) = 2. * C5);
    I3EE(1, 5) = I3EE(5, 1) = -2. * (I3EE(3, 4) = I3EE(4, 3) = 2. * C6);

    const auto W8 = W5;
    const auto W6 = .5 * W3;
    W1 = two_three * W8;
    W2 *= four_three;
    W3 = .375 * W2;
    W5 = two_three * W4;
    W4 = four_three * W8;
    const auto W7 = .75 * W5;
    const auto W9 = .5 * W8;

    const mat TA = A10 * W1 * J1E + A01 * W4 * J2E;
    const mat TB = TA * J3E.t();

    trial_stiffness = (A10 * W2 + A01 * W5 + K - K * J3M1 * W8) * J3E * J3E.t() + (K * J3M1 *
    ↪  W9 - A10 * W3 - A01 * W7) * I3EE + A01 * W6 * I2EE - TB - TB.t();

    return SUANPAN_SUCCESS;
}
```

10.2. Blatz–Ko Model

10.2.1. Theory

10.2.2. Formulation

10.2.3. Implementation

```
// takes green strain as input
int BlatzKo::update_trial_status(const vec& t_strain) {
    trial_strain = t_strain;

    vec G = weight % t_strain + tensor::unit_tensor2;

    vec H(6);
    H(0) = G(1) * G(2) - G(4) * G(4);
    H(1) = G(2) * G(0) - G(5) * G(5);
    H(2) = G(0) * G(1) - G(3) * G(3);
    H(3) = G(4) * G(5) - G(2) * G(3);
    H(4) = G(5) * G(3) - G(0) * G(4);
    H(5) = G(3) * G(4) - G(1) * G(5);

```

```
    const auto I3 = G(0) * H(0) + G(3) * H(3) + G(5) * H(5);

    auto factor_a = pow(std::max(datum::eps, I3), -half_beta_two);

    trial_stress = shear_modulus * (tensor::unit_tensor2 - factor_a * H);

    G *= -(factor_a *= 2. * shear_modulus);

    trial_stiffness.zeros(6, 6);

    trial_stiffness(4, 4) = -.5 * (trial_stiffness(1, 2) = G(0));
    trial_stiffness(5, 5) = -.5 * (trial_stiffness(0, 2) = G(1));
    trial_stiffness(3, 3) = -.5 * (trial_stiffness(0, 1) = G(2));
    trial_stiffness(4, 5) = -.5 * (trial_stiffness(2, 3) = -G(3));
    trial_stiffness(3, 5) = -.5 * (trial_stiffness(0, 4) = -G(4));
    trial_stiffness(3, 4) = -.5 * (trial_stiffness(1, 5) = -G(5));

    factor_a *= half_beta_two / I3;

    for(auto I = 0; I < 6; ++I) {
        const auto factor_b = factor_a * H(I);
        trial_stiffness(I, I) += factor_b * H(I);
        for(auto J = I + 1; J < 6; ++J) trial_stiffness(J, I) = trial_stiffness(I, J) +=
        ↪  factor_b * H(J);
    }

    return SUANPAN_SUCCESS;
}
```

10.3. Yeoh Model

10.3.1. Theory

10.3.2. Formulation

10.3.3. Implementation

```
// takes green strain as input
int Yeoh::update_trial_status(const vec& t_strain) {
    const vec G = weight % (trial_strain = t_strain) + tensor::unit_tensor2;

    const auto &C1 = G(0), &C2 = G(1), &C3 = G(2), &C4 = G(3), &C5 = G(4), &C6 = G(5);

    const auto I1 = C1 + C2 + C3;
    const auto I3 = std::max(datum::eps, C1 * C2 * C3 + 2. * C4 * C5 * C6 - C1 * C5 * C5 - C2
    ↪  * C6 * C6 - C3 * C4 * C4);
```

```
9
10      const vec I3E = 2. * vec{C2 * C3 - C5 * C5, C3 * C1 - C6 * C6, C1 * C2 - C4 * C4, C5 * C6
        ↪  - C3 * C4, C6 * C4 - C1 * C5, C4 * C5 - C2 * C6};
11
12      const auto W1 = pow(I3, -one_three);
13      const auto W2 = one_three * I1 * pow(I3, -four_three);
14      const auto W5 = .5 * pow(I3, -.5);
15
16      const vec J1E = W1 * I1E - W2 * I3E;
17      const vec J3E = W5 * I3E;
18
19      const auto D = compute_derivative(I1 * W1 - 3., sqrt(I3) - 1.);
20
21      const auto &DWDJ1 = D(0), &DWDJ3 = D(1), &DDWDDJ1 = D(2), &DDWDDJ3 = D(3);
22
23      trial_stress = DWDJ1 * J1E + DWDJ3 * J3E;
24
25      mat I3EE(6, 6, fill::zeros);
26      I3EE(1, 2) = I3EE(2, 1) = -2. * (I3EE(4, 4) = -2. * C1);
27      I3EE(0, 2) = I3EE(2, 0) = -2. * (I3EE(5, 5) = -2. * C2);
28      I3EE(0, 1) = I3EE(1, 0) = -2. * (I3EE(3, 3) = -2. * C3);
29      I3EE(2, 3) = I3EE(3, 2) = -2. * (I3EE(4, 5) = I3EE(5, 4) = 2. * C4);
30      I3EE(0, 4) = I3EE(4, 0) = -2. * (I3EE(3, 5) = I3EE(5, 3) = 2. * C5);
31      I3EE(1, 5) = I3EE(5, 1) = -2. * (I3EE(3, 4) = I3EE(4, 3) = 2. * C6);
32
33      const auto P1 = 2. * four_three * W2 * DWDJ1;
34      const auto P2 = four_three * W5 * DWDJ1;
35      const auto P3 = W2 * DWDJ1;
36      const auto P4 = W5 * DWDJ3;
37
38      trial_stiffness = (P1 + DDWDDJ3 - 2. * P4) * J3E * J3E.t() + (P4 - P3) * I3EE + (DDWDDJ1
        ↪  * J1E - P2 * J3E) * J1E.t() - P2 * J1E * J3E.t();
39
40      return SUANPAN_SUCCESS;
41  }
```

11. Geomaterial

In geotechnical engineering, compression positive convention is used in existing literatures. This brings some convenience to geotechnical engineering specific applications but conflicts against the common convention adopted in continuum mechanics (tension positive).

To be consistent with other models, in this chapter the continuum mechanics convention is adopted. Thus the formulations presented here may differ from what readers can found in other literatures.

11.1. Drucker–Prager Model

The Drucker–Prager model is a simple model that resembles the J2 model and can be used in simple preliminary stability/capacity analysis.

11.1.1. Theory

Yield Function

$$f = \sqrt{J_2} + \eta p - \xi c, \tag{11.1}$$

where $J_2 = \frac{1}{2}\boldsymbol{s} : \boldsymbol{s}$ is the J2 invariant of the deviatoric stress tensor $\boldsymbol{s}$, $p = \frac{1}{3}\mathrm{trace}\,(\boldsymbol{\sigma})$ is the hydrostatic stress, η and ξ are two model parameters which can be adjusted to match different shapes, c is the cohesion.

Flow Rule

The plastic potential possesses a form that resembles the yield function.

$$g = \sqrt{J_2} + \bar{\eta} p. \tag{11.2}$$

When $\bar{\eta} = \eta$, an associative flow rule is implied. The corresponding rate form of plastic strain is then

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \frac{\partial g}{\partial \boldsymbol{\sigma}} = \gamma \left(\frac{\sqrt{2}}{2} \boldsymbol{n} + \frac{\bar{\eta}}{3} \mathbf{1} \right). \tag{11.3}$$

Hardening Law

The accumulated equivalent plastic strain α can be defined as

$$\dot{\alpha} = \gamma. \tag{11.4}$$

Then the cohesion can be defined as a function of α.

$$c = c(\alpha). \tag{11.5}$$

More complex laws to allow η and ξ to evolve with the development of plasticity are also possible. Here for simplicity, it is assumed that η and ξ are two constants.

11.1.2. Formulation

As the cohesion c is not explicitly defined in this model, it can be customised to produce various hardening responses.

The only local variable is the plasticity consistency parameter γ.

Elastic Loading/Unloading

The trial state can be computed as

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p). \tag{11.6}$$

The deviatoric stress and hydrostatic stress can be computed accordingly,

$$\boldsymbol{s}^{\text{trial}} = \text{dev}\left(\boldsymbol{\sigma}^{\text{trial}}\right), \tag{11.7}$$

$$p^{\text{trial}} = \frac{1}{3}\text{trace}\left(\boldsymbol{\sigma}^{\text{trial}}\right). \tag{11.8}$$

11.1.3. Implementation

```
int NonlinearDruckerPrager::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= datum::eps) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * incre_strain;

    trial_history = current_history;
    auto& plastic_strain = trial_history(0);

    const auto dev_stress = tensor::dev(trial_stress);
    const auto hydro_stress = tensor::mean3(trial_stress);
    const auto sqrt_j2 = sqrt(std::max(datum::eps, tensor::stress::invariant2(dev_stress)));

    const auto yield_const = sqrt_j2 + eta_yield * hydro_stress;

    if(yield_const <= xi * compute_c(plastic_strain)) return SUANPAN_SUCCESS;

    auto gamma = 0.;
    double denominator;

    auto counter = 0u;
    auto ref_error = 1.;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("Cannot converge within {} iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        const auto residual = yield_const - factor_a * gamma - xi *
        ↪  compute_c(plastic_strain);
        const auto incre_gamma = residual / (denominator = factor_a + xi * xi *
        ↪  compute_dc(plastic_strain));
        const auto error = fabs(incre_gamma);
        if(1u == counter) ref_error = error;
        suanpan_debug("Local iteration error: {:.5E}.\n", error);
        if(error < tolerance * ref_error || (fabs(residual) < tolerance && counter > 5u))
        ↪  break;
        plastic_strain = current_history(0) + xi * (gamma += incre_gamma);
    }

    if(sqrt_j2 >= shear * gamma) {
        const auto norm_s = tensor::stress::norm(dev_stress);

        const auto t_factor = shear / sqrt_j2 * gamma;

        trial_stress -= t_factor * dev_stress + bulk * eta_flow * gamma *
        ↪  tensor::unit_tensor2;

        trial_stiffness += double_shear * (t_factor - shear / denominator) / norm_s / norm_s
        ↪  * dev_stress * dev_stress.t() - double_shear * t_factor * unit_dev_tensor -
        ↪  factor_d / denominator * unit_x_unit;

```

```
48          const mat t_mat = eta_yield * factor_c / denominator / norm_s * dev_stress *
            ↪ tensor::unit_tensor2.t();
49
50          associated ? trial_stiffness -= t_mat + t_mat.t() : trial_stiffness -= t_mat +
            ↪ eta_flow / eta_yield * t_mat.t();
51      }
52      else {
53          // apex return
54          gamma = 0.; // volumetric strain reuse variable
55          plastic_strain = current_history(0);
56
57          counter = 0u;
58          while(true) {
59              if(max_iteration == ++counter) {
60                  suanpan_error("Cannot converge within {} iterations.\n", max_iteration);
61                  return SUANPAN_FAIL;
62              }
63
64              const auto residual = compute_c(plastic_strain) * xi / eta_flow - hydro_stress +
                ↪ bulk * gamma;
65              const auto incre_gamma = residual / (denominator = factor_b *
                ↪ compute_dc(plastic_strain) + bulk);
66              const auto error = fabs(incre_gamma);
67              if(1u == counter) ref_error = error;
68              suanpan_debug("Local iteration error: {:.5E}.\n", error);
69              if(error < tolerance * ref_error || (fabs(residual) < tolerance && counter > 5u))
                ↪ break;
70              plastic_strain = current_history(0) + xi / eta_yield * (gamma -= incre_gamma);
71          }
72
73          trial_stress = (hydro_stress - bulk * gamma) * tensor::unit_tensor2;
74
75          trial_stiffness = (bulk - bulk * bulk / denominator) * unit_x_unit;
76      }
77
78      return SUANPAN_SUCCESS;
79  }
```

11.2. Modified Cam Clay Model

This model is identical to the one documented in [13, see § 10.1] apart from the definition of hardening variable. In this section, we present an equivalent formulation.

11.2.1. Theory

Basics

The model adopts a two-scalar formulation that depends on hydrostatic stress p and von Mises equivalent stress q, which are defined as

$$p = \frac{\text{trace}\,(\boldsymbol{\sigma})}{3} = \frac{I_1}{3}, \qquad q = \sqrt{3J_2} = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}} = \sqrt{\frac{3}{2}}\|\boldsymbol{s}\|. \tag{11.9}$$

They are effectively two invariants I_1 and J_2.

Yield Function

The yield function is defined as

$$f = \frac{p_e^2}{b^2} + \frac{q^2}{M^2} - a^2, \tag{11.10}$$

where $p_t \geqslant 0$ is the tensile yield strength, $b = 1$ if $p_e \geqslant 0$ and $b = \beta$ if $p_e < 0$ with $p_e = p - p_t + a$ denotes relative stress to the centre of ellipse, β is a constant that controls the shape of negative-wards half of yielding ellipse, M is the ratio between two radii of yielding ellipse.

The above yield function essentially possesses the form of an ellipse, for example,

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1.$$

The centre of the ellipse is shifted to $p_t - a$ so that the majority of the ellipse is in the compression region. The radius b alters its value depending on the sign of p_e. The yield surface is effectively two half ellipses glued together.

Flow Rule

The associative plasticity is assumed so that

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \frac{\partial f}{\partial \boldsymbol{\sigma}} = \gamma \left(\frac{2p_e}{3b^2}\mathbf{1} + \frac{3}{M^2}\boldsymbol{s} \right). \tag{11.11}$$

The following relationship is used,

$$\frac{\mathrm{d}q^2}{\mathrm{d}\boldsymbol{\sigma}} = \frac{3}{2}\frac{\mathrm{d}\,(\boldsymbol{s} : \boldsymbol{s})}{\mathrm{d}\boldsymbol{\sigma}} = \frac{3}{2}\left(\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\sigma}} : \boldsymbol{s} + \boldsymbol{s} : \frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\sigma}} \right) = 3\boldsymbol{s},$$

since $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}\boldsymbol{\sigma}} : \boldsymbol{s} = \mathbb{I}^{\mathrm{dev}} : \boldsymbol{s} = \boldsymbol{s}$. Accordingly,

$$\dot{\varepsilon_v^p} = \gamma \frac{2p_e}{b^2}, \qquad \dot{\boldsymbol{\varepsilon}_d^p} = \gamma \frac{3}{M^2}\boldsymbol{s}, \tag{11.12}$$

where $\dot{\varepsilon_v^p} = \mathrm{trace}\left(\dot{\boldsymbol{\varepsilon}^p}\right)$ is the increment of the volumetric strain scalar and $\dot{\boldsymbol{\varepsilon}_d^p}$ is the increment of the deviatoric strain tensor.

Hardening Rule

The hardening variable α is defined as the volumetric strain ε_v^p so that

$$\alpha = \varepsilon_v^p. \tag{11.13}$$

The corresponding incremental form is then

$$\alpha = \alpha_n + \gamma \frac{2p_e}{b^2}. \tag{11.14}$$

The hardening rule is then defined as a function of α,

$$a = a\left(\alpha\right) \geqslant 0. \tag{11.15}$$

Note here we simple take $\alpha = \varepsilon_v^p$ instead of $\alpha = -\varepsilon_v^p$ in the original literature [13].

11.2.2. Formulation

Local Residual

By using the elastic relationship,

$$\boldsymbol{s} = 2G\boldsymbol{\varepsilon}_d^e = 2G\left(\boldsymbol{\varepsilon}_d^{\mathrm{trial}} - \Delta\boldsymbol{\varepsilon}_d^p\right) = \boldsymbol{s}^{\mathrm{trial}} - \gamma \frac{6G}{M^2}\boldsymbol{s}, \tag{11.16}$$

one can rearrange and obtain the following,

$$\boldsymbol{s} = \frac{M^2}{M^2 + 6G\Delta\gamma}\boldsymbol{s}^{\mathrm{trial}}, \qquad q = \frac{M^2}{M^2 + 6G\Delta\gamma}q^{\mathrm{trial}}. \tag{11.17}$$

The governing residual equations for independent variables $\boldsymbol{x} = \begin{bmatrix}\gamma & \alpha\end{bmatrix}^{\mathrm{T}}$ can be expressed

as

$$\boldsymbol{R} = \begin{cases} \dfrac{p_e^2}{b^2} + \dfrac{q^2}{M^2} - a^2 = 0, \\ \alpha - \alpha_n - \gamma \dfrac{2}{b^2} p_e = 0. \end{cases} \tag{11.18}$$

where $p_e = p^{\text{trial}} - K\alpha + K\alpha_n - p_t + a$ since

$$p = K\varepsilon_v^e = K\left(\varepsilon_v^{\text{trial}} - \Delta\varepsilon_v^p\right) = p^{\text{trial}} - K\left(\alpha - \alpha_n\right), \tag{11.19}$$

and $q = \dfrac{M^2}{M^2 + 6G\Delta\gamma} q^{\text{trial}}$.

Local Iteration

The Jacobian can be formed accordingly.

$$\boldsymbol{J} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{-12GM^2 q^{\text{trial},2}}{(M^2 + 6G\gamma)^3} & \dfrac{2}{b^2} p_e (a' - K) - 2aa' \\ -\dfrac{2}{b^2} p_e & 1 - \gamma \dfrac{2}{b^2}(a' - K) \end{bmatrix}. \tag{11.20}$$

Tangent Stiffness

At local iteration, $\boldsymbol{\varepsilon}^{\text{trial}}$ (or $\boldsymbol{\varepsilon}^{n+1}$, or just $\boldsymbol{\varepsilon}$ for brevity) is fixed and $\boldsymbol{R}$ is iterated out. Noting that in the global iteration, $\boldsymbol{\varepsilon}^{\text{trial}}$ is also a variable that changes. If local iteration is converged, then $\boldsymbol{R} = \boldsymbol{0}$, so

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}\boldsymbol{\varepsilon}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{\varepsilon}} = \boldsymbol{0}, \tag{11.21}$$

consequently,

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{\varepsilon}} = \begin{bmatrix} \dfrac{\mathrm{d}\gamma}{\mathrm{d}\boldsymbol{\varepsilon}} \\ \dfrac{\mathrm{d}\alpha}{\mathrm{d}\boldsymbol{\varepsilon}} \end{bmatrix} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}} = -\boldsymbol{J}^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}}. \tag{11.22}$$

Taking derivatives about $\boldsymbol{\varepsilon}$ gives

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}} = \begin{bmatrix} \dfrac{2p_e K}{b^2}\boldsymbol{1} + \dfrac{6G}{M^2 + 6G\gamma}\boldsymbol{s} \\ \dfrac{-2\gamma K}{b^2}\boldsymbol{1} \end{bmatrix}. \tag{11.23}$$

The stress can be expressed as

$$\boldsymbol{\sigma} = \boldsymbol{s} + p\mathbf{1} = \frac{M^2}{M^2 + 6G\Delta\gamma}\boldsymbol{s}^{\text{trial}} + \left(p^{\text{trial}} - K\left(\alpha - \alpha_n\right)\right)\mathbf{1}. \tag{11.24}$$

Direct differentiation gives

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\boldsymbol{\varepsilon}} &= \frac{M^2}{M^2 + 6G\gamma}\frac{\mathrm{d}\boldsymbol{s}^{\text{trial}}}{\mathrm{d}\boldsymbol{\varepsilon}} + \boldsymbol{s}^{\text{trial}} \otimes \frac{\mathrm{d}\dfrac{M^2}{M^2 + 6G\gamma}}{\mathrm{d}\boldsymbol{\varepsilon}} + \mathbf{1} \otimes \frac{\mathrm{d}\left(p^{\text{trial}} - K\left(\alpha - \alpha_n\right)\right)}{\mathrm{d}\boldsymbol{\varepsilon}} \\ &= \frac{2GM^2}{M^2 + 6G\gamma}\mathbb{I}^{\text{dev}} + K\mathbf{1} \otimes \mathbf{1} - K\mathbf{1} \otimes \frac{\mathrm{d}\alpha}{\mathrm{d}\boldsymbol{\varepsilon}} - \frac{6GM^2}{\left(M^2 + 6G\gamma\right)^2}\boldsymbol{s}^{\text{trial}} \otimes \frac{\mathrm{d}\gamma}{\mathrm{d}\boldsymbol{\varepsilon}}, \\ &= \boldsymbol{D} - \frac{12G^2\gamma}{M^2 + 6G\gamma}\mathbb{I}^{\text{dev}} - K\mathbf{1} \otimes \frac{\mathrm{d}\alpha}{\mathrm{d}\boldsymbol{\varepsilon}} - \frac{6G}{M^2 + 6G\gamma}s \otimes \frac{\mathrm{d}\gamma}{\mathrm{d}\boldsymbol{\varepsilon}}.\end{aligned} \tag{11.25}$$

In which $\dfrac{\mathrm{d}\alpha}{\mathrm{d}\boldsymbol{\varepsilon}}$ and $\dfrac{\mathrm{d}\gamma}{\mathrm{d}\boldsymbol{\varepsilon}}$ are obtained from Eq. (11.22). The above derivation also takes advantage of the expression

$$\boldsymbol{D} = 2G\mathbb{I}^{\text{dev}} + K\mathbf{1} \otimes \mathbf{1}. \tag{11.26}$$

11.2.3. Implementation

```
int NonlinearCamClay::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= datum::eps) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * incre_strain;

    trial_history = current_history;
    auto& alpha = trial_history(0);
    const auto& current_alpha = current_history(0);

    auto trial_s = tensor::dev(trial_stress);
    const auto trial_q = sqrt_three_two * tensor::stress::norm(trial_s);
    const auto p = tensor::mean3(trial_stress);

    auto gamma = 0.;

    vec residual(2), incre;
    mat jacobian(2, 2);

    auto counter = 0u;
    auto rel_error = 1.;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("Cannot converge within {} iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }
```

```
28
29          const auto a = compute_a(alpha);
30          const auto da = compute_da(alpha);
31          const auto incre_alpha = alpha - current_alpha;
32          const auto trial_p = p - bulk * incre_alpha;
33          const auto rel_p = trial_p - pt + a;
34          const auto square_b = rel_p >= 0. ? 1. : square_beta;
35          const auto denom = square_m + six_shear * gamma;
36          const auto square_qm = pow(m * trial_q / denom, 2.);
37
38          residual(0) = rel_p * rel_p / square_b + square_qm - a * a;
39
40          if(1u == counter && residual(0) < 0.) return SUANPAN_SUCCESS;
41
42          residual(1) = incre_alpha - 2. * gamma / square_b * rel_p;
43
44          jacobian(0, 0) = -2. * six_shear / denom * square_qm;
45          jacobian(1, 0) = -2. * rel_p / square_b;
46          jacobian(0, 1) = jacobian(1, 0) * (bulk - da) - 2. * a * da;
47          jacobian(1, 1) = 1. - 2. * gamma / square_b * (da - bulk);
48
49          if(!solve(incre, jacobian, residual, solve_opts::equilibrate)) return SUANPAN_FAIL;
50
51          const auto error = inf_norm(incre);
52          if(1u == counter) rel_error = error;
53          suanpan_debug("Local iteration error: {:.5E}.\n", error);
54          if(error < tolerance * rel_error || (inf_norm(residual) < tolerance && counter > 5u))
            ↪ {
55              mat left(6, 2);
56
57              rel_error = 2. * bulk / square_b; // reuse variable
58              left.col(0) = rel_error * rel_p * tensor::unit_tensor2 + six_shear / denom *
                ↪ (trial_s *= square_m / denom);
59              left.col(1) = -rel_error * gamma * tensor::unit_tensor2;
60
61              trial_stress = trial_s + trial_p * tensor::unit_tensor2;
62
63              rel_error = six_shear / denom; // reuse variable
64              trial_stiffness -= 2. * shear * gamma * rel_error * unit_dev_tensor -
                ↪ join_rows(rel_error * trial_s, bulk * tensor::unit_tensor2) * solve(jacobian,
                ↪ left.t());
65
66              return SUANPAN_SUCCESS;
67          }
68
69          gamma -= incre(0);
70          alpha -= incre(1);
71      }
72  }
```

11.3. Simple Sand Model

This sand model is a hybrid of several existing models with significant simplifications. Readers should not expect this model to give accurate results. Instead, this model only captures the characteristics of sand material. It is presented to craft a basic framework from which more complex sand models can be developed.

Compared to other models for sand, this model is more like a preliminary model just like the bi-linear hardening von Mises model compared to other models for metals.

A distinctive feature of sand models is that, often a non-associative plastic flow is used and the hardening law is significantly more complex, involving heavy tensor operations, than aforementioned models. Often the hardening laws are tensor–valued functions, and cannot be simplified to scalar–valued functions. The numerical complexity is higher.

11.3.1. Theory

Elastic Constitutive Relation

Often for sand, the elastic response is nonlinear (to be specific, hyper-elastic). The shear modulus and bulk modulus often are functions of hydrostatic pressure. In this model, for simplicity, an linear elastic relation is assumed. Readers shall be warned this does no reflect the real behaviour of sand.

The decomposition of total strain into elastic/plastic strain components is still valid. Thus the trial elastic state can be computed via the same expression.

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p) = \boldsymbol{\sigma}_n + \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n)\,. \tag{11.27}$$

Furthermore,

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{s}^{\text{trial}} + p^{\text{trial}}\mathbf{1}. \tag{11.28}$$

Yield Function

A wedge–like function is chosen to be the yield surface.

$$f = \|\boldsymbol{s} + p\boldsymbol{\alpha}\| + mp = \|\boldsymbol{\eta}\| + mp,$$

where $\boldsymbol{s} = \operatorname{dev}(\boldsymbol{\sigma})$ is the deviatoric stress, $p = \dfrac{1}{3}\operatorname{trace}(\boldsymbol{\sigma})$ is the hydrostatic stress, $\boldsymbol{\alpha}$ is the so called back stress and m characterises the size of the wedge. For simplicity, m is assumed to be a constant in this model.

Flow Rule

A non-associated plastic flow is used, the corresponding flow rule is defined as follows.

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \left(\boldsymbol{n} + \frac{1}{3} A \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) \mathbf{1} \right),$$

where $D = A \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right)$ is the dilatancy parameter, $\boldsymbol{n} = \dfrac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|}$ is the unit directional tensor, A is a model constant, it can also be defined as a function of current state. Note since the sign convention is changed, a positive A leads to dilatant behaviour.

Since the linear elasticity is applied, the following expressions hold.

$$\boldsymbol{s} = \boldsymbol{s}^{\text{trial}} - 2G\gamma\boldsymbol{n}, \qquad p = p^{\text{trial}} - KA\gamma \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right).$$

Hardening Law

The evolution rate of the back stress $\boldsymbol{\alpha}$ is defined in terms of a proper distance measure from the bounding surface, Here, such a distance measure is chosen to be $h \left(\left(\alpha^b - m \right) \boldsymbol{n} - \boldsymbol{\alpha} \right)$, where h is a model constant. Thus,

$$\dot{\boldsymbol{\alpha}} = \gamma h \left(\left(\alpha^b - m \right) \boldsymbol{n} - \boldsymbol{\alpha} \right).$$

Essentially, this is similar to the Armstrong–Frederick type kinematic hardening rule. It shall be noted that $\boldsymbol{n}$ is a deviatoric stress, and $\boldsymbol{\alpha}$ stays deviatoric but may not be coaxial with $\boldsymbol{n}$. Thus in tensor notion,

$$\boldsymbol{n} = \mathbb{I} : \boldsymbol{n} = \mathbb{I}^{\text{dev}} : \boldsymbol{n}, \qquad \boldsymbol{\alpha} = \mathbb{I} : \boldsymbol{\alpha} = \mathbb{I}^{\text{dev}} : \boldsymbol{\alpha}. \tag{11.29}$$

Critical State

The state parameter is defined to be

$$\psi = v_0 \left(1 + \varepsilon^{v,\text{trial}} \right) - v_c + \lambda_c \ln \left(\frac{p}{p_c} \right),$$

where v_0 is the initial specific volume, v_c is the corresponding specific volume on the critical line at p_c and λ_c is absolute value of the slope of the critical line in v–$\ln(-p)$ space.

The Lode angle dependence is not considered for simple derivations of the corresponding terms. Hence both the dilatancy surface and bounding surface will be circular cones in the principal stress space.

The dilatancy surface is defined as

$$\alpha^d = \alpha^c \exp\left(n^d \psi\right).$$

The corresponding derivatives are

$$\frac{\partial \alpha^d}{\partial \boldsymbol{\varepsilon}^{\text{trial}}} = \alpha^d n^d v_0 \mathbf{1}^{\mathrm{T}}, \qquad \frac{\partial \alpha^d}{\partial p} = \alpha^d n^d \frac{\lambda_c}{p}.$$

The bounding surface is defined as

$$\alpha^b = \alpha^c \exp\left(-n^b \psi\right).$$

The corresponding derivatives are

$$\frac{\partial \alpha^b}{\partial \boldsymbol{\varepsilon}^{\text{trial}}} = -\alpha^b n^b v_0 \mathbf{1}^{\mathrm{T}}, \qquad \frac{\partial \alpha^b}{\partial p} = -\alpha^b n^b \frac{\lambda_c}{p}.$$

The symbol $\mathbf{1}$ denotes the unit second order tensor. In explicit compressed matrix representation, it is $\mathbf{1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$.

11.3.2. Formulation

Some tensorial derivatives are presented first. They will be used in both formulation and implementation. According to Eq. (2.41), the following expressions can be derived.

$$\frac{\partial \|\boldsymbol{\eta}\|}{\partial} = \boldsymbol{n} : \frac{\partial \boldsymbol{\eta}}{\partial}, \qquad \frac{\partial \boldsymbol{n}}{\partial} = \frac{1}{\|\boldsymbol{\eta}\|} \left(\mathbb{I} - \boldsymbol{n} \otimes \boldsymbol{n}\right) : \frac{\partial \boldsymbol{\eta}}{\partial}.$$

Hence, in compressed matrix representation with the scaling vector $\boldsymbol{c} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix}^{\mathrm{T}}$,

$$\begin{aligned}
\frac{\partial \|\boldsymbol{\eta}\|}{\partial p} &= \boldsymbol{n}^{\mathrm{T}} \mathrm{diag}\left(\boldsymbol{c}\right) \boldsymbol{\alpha}, & \frac{\partial \boldsymbol{n}}{\partial p} &= \frac{1}{\|\boldsymbol{\eta}\|} \left(\boldsymbol{\alpha} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}} \mathrm{diag}\left(\boldsymbol{c}\right) \boldsymbol{\alpha}\right), \\
\frac{\partial \|\boldsymbol{\eta}\|}{\partial \boldsymbol{s}} &= \boldsymbol{n} \circ \boldsymbol{c}, & \frac{\partial \boldsymbol{n}}{\partial \boldsymbol{s}} &= \frac{1}{\|\boldsymbol{\eta}\|} \left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}} \mathrm{diag}\left(\boldsymbol{c}\right)\right), \\
\frac{\partial \|\boldsymbol{\eta}\|}{\partial \boldsymbol{\alpha}} &= p\ \boldsymbol{n} \circ \boldsymbol{c}, & \frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\alpha}} &= \frac{p}{\|\boldsymbol{\eta}\|} \left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}} \mathrm{diag}\left(\boldsymbol{c}\right)\right).
\end{aligned}$$

The governing equations of this model are

Constitutive Law	$\boldsymbol{\sigma} = \boldsymbol{D} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^p)$
Yield Function	$f = \|\boldsymbol{s} + p\boldsymbol{\alpha}\| + mp = \|\boldsymbol{\eta}\| + mp$
Flow Rule	$\dot{\boldsymbol{\varepsilon}^p} = \gamma \left(\boldsymbol{n} + \frac{1}{3} A \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) \mathbf{1} \right)$
Hardening Law	$\dot{\boldsymbol{\alpha}} = \gamma h \left(\left(\alpha^b - m \right) \boldsymbol{n} - \boldsymbol{\alpha} \right)$
Critical State	$\psi = v_0 \left(1 + \varepsilon^{v,\text{trial}} \right) - v_c + \lambda_c \ln \left(\frac{p}{p_c} \right)$
	$\alpha^d = \alpha^c \exp \left(n^d \psi \right)$
	$\alpha^b = \alpha^c \exp \left(-n^b \psi \right)$

Elastic Loading/Unloading

The trial yield function can be computed from trial stress as

$$f^{\text{trial}} = \left\| \boldsymbol{s}^{\text{trial}} + p^{\text{trial}} \boldsymbol{\alpha} \right\| + m p^{\text{trial}} = \left\| \boldsymbol{\eta}^{\text{trial}} \right\| + m p^{\text{trial}}. \tag{11.30}$$

Plastic Evolution

In summary, there are four residual equations.

$$\boldsymbol{R} = \begin{cases} \|\boldsymbol{\eta}_{n+1}\| + m p_{n+1}, \\ p_{n+1} - p^{\text{trial}} + KA\gamma \left(\alpha^d_{n+1} - m - \boldsymbol{\alpha}_{n+1} : \boldsymbol{n}_{n+1} \right), \\ \boldsymbol{s}_{n+1} - \boldsymbol{s}^{\text{trial}} + 2G\gamma \boldsymbol{n}_{n+1}, \\ \boldsymbol{\alpha}_n + \gamma h \left(\alpha^b_{n+1} - m \right) \boldsymbol{n}_{n+1} - (\gamma h + 1) \boldsymbol{\alpha}_{n+1}. \end{cases} \tag{11.31}$$

To avoid complex computation of derivatives, the above four result equations are not further combined. By defined $\boldsymbol{x} = \begin{bmatrix} \gamma & p_{n+1} & \boldsymbol{s}_{n+1} & \boldsymbol{\alpha}_{n+1} \end{bmatrix}^{\text{T}}$, the local Jacobian can be derived and expressed in the compressed matrix form. It shall be noted for simplicity the subscript $(\cdot)_{n+1}$ is dropped.

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial \boldsymbol{R}}{\partial \gamma} & \frac{\partial \boldsymbol{R}}{\partial p} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{s}} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\alpha}} \end{bmatrix}, \tag{11.32}$$

with

$$\frac{\partial \boldsymbol{R}}{\partial \gamma} = \begin{bmatrix} 0 \\ KA\left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n}\right) \\ 2G\boldsymbol{n} \\ h\left(\alpha^b - m\right)\boldsymbol{n} - h\boldsymbol{\alpha} \end{bmatrix}, \tag{11.33}$$

$$\frac{\partial \boldsymbol{R}}{\partial p} = \begin{bmatrix} \boldsymbol{\alpha} : \boldsymbol{n} + m \\ 1 + KA\gamma\left(\alpha^d n^d \dfrac{\lambda_c}{p} - \dfrac{\boldsymbol{\alpha} : \boldsymbol{\alpha} - (\boldsymbol{\alpha} : \boldsymbol{n})^2}{\|\boldsymbol{\eta}\|}\right) \\ \dfrac{2G\gamma}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{\alpha} - (\boldsymbol{\alpha} : \boldsymbol{n})\,\boldsymbol{n}\right) \\ \dfrac{\gamma h\left(\alpha^b_{n+1} - m\right)}{\|\boldsymbol{\eta}\|}\boldsymbol{\alpha} - \left(\gamma h \alpha^b n^b \dfrac{\lambda_c}{p} + \dfrac{\gamma h\left(\alpha^b_{n+1} - m\right)}{\|\boldsymbol{\eta}\|}(\boldsymbol{\alpha} : \boldsymbol{n})\right)\boldsymbol{n} \end{bmatrix}, \tag{11.34}$$

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{s}} = \begin{bmatrix} \boldsymbol{n}^{\mathrm{T}} \circ \boldsymbol{c}^{\mathrm{T}} \\ \dfrac{KA\gamma}{\|\boldsymbol{\eta}\|}\left((\boldsymbol{\alpha} : \boldsymbol{n})\,\boldsymbol{n} - \boldsymbol{\alpha}\right)^{\mathrm{T}} \circ \boldsymbol{c}^{\mathrm{T}} \\ \boldsymbol{I} + \dfrac{2G\gamma}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{c})\right) \\ \dfrac{\gamma h\left(\alpha^b_{n+1} - m\right)}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{c})\right) \end{bmatrix}, \tag{11.35}$$

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\alpha}} = \begin{bmatrix} p\;\boldsymbol{n}^{\mathrm{T}} \circ \boldsymbol{c}^{\mathrm{T}} \\ KA\gamma\left(\left(1 + \dfrac{p}{\|\boldsymbol{\eta}\|}(\boldsymbol{\alpha} : \boldsymbol{n})\right)\boldsymbol{n} - \dfrac{p}{\|\boldsymbol{\eta}\|}\boldsymbol{\alpha}\right)^{\mathrm{T}} \circ \boldsymbol{c}^{\mathrm{T}} \\ 2G\gamma\dfrac{p}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{c})\right) \\ \dfrac{\gamma h\left(\alpha^b_{n+1} - m\right)p}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{c})\right) - (\gamma h + 1)\,\boldsymbol{I} \end{bmatrix}. \tag{11.36}$$

For simplicity, the double contraction is not explicitly expressed, one shall always use $\boldsymbol{A}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{c})\,\boldsymbol{B}$ (in compressed matrix representation) for $\boldsymbol{A} : \boldsymbol{B}$. The Jacobian has a size of $1 + 1 + 6 + 6 =$

14.

Consistent Tangent Stiffness

The consistent tangent stiffness can be derived from the decomposition

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{s}_{n+1} + p_{n+1}\mathbf{1}. \tag{11.37}$$

From which,

$$\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \frac{\partial \boldsymbol{s}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} + \mathbf{1} \otimes \frac{\partial p_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{11.38}$$

Since $\boldsymbol{s}$ and p are local independent variables, from local residual,

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \mathbf{0}, \tag{11.39}$$

then

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} \dfrac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\partial p_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\partial \boldsymbol{s}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ \dfrac{\partial \boldsymbol{\alpha}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} \end{bmatrix} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{11.40}$$

From the above equation, $\dfrac{\partial \boldsymbol{s}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ and $\dfrac{\partial p_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ can be extracted. The term $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}$ can be computed as

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} 0 \\ \left(KA\gamma\alpha^d n^d v_0 - K\right)\mathbf{1}^{\mathrm{T}} \\ -2G\mathbb{I}^{\mathrm{dev}} \\ -\gamma h \alpha^b n^b v_0 \mathbf{1}^{\mathrm{T}} \end{bmatrix} \tag{11.41}$$

11.3.3. Implementation

The implementation of this simple sand model is presented in Algorithm 10. Please note since the local iteration uses $\boldsymbol{s}$ and p as independent variables, the plastic strain history does not need

to be explicitly stored. It can always be recovered via $\boldsymbol{\varepsilon}^p = \boldsymbol{\varepsilon} - \boldsymbol{D}^{-1} : \sigma$.

Algorithm 10 state determination of simple sand model

Parameter: K, G, A, m, h, n^d, n^b, α_c, v_0, v_c, λ_c, p_c
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\sigma}_n$, $\boldsymbol{\alpha}_n$
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\sigma}_{n+1}$, $\boldsymbol{\alpha}_{n+1}$
$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{\sigma}_n + \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n)$
$\boldsymbol{s}^{\text{trial}} = \text{dev}\left(\boldsymbol{\sigma}^{\text{trial}}\right)$
$p^{\text{trial}} = \frac{1}{3}\text{trace}\left(\boldsymbol{\sigma}^{\text{trial}}\right)$
$\boldsymbol{\eta}^{\text{trial}} = \boldsymbol{s}^{\text{trial}} + p^{\text{trial}}\boldsymbol{\alpha}_n$
compute f^{trial} ▷ Eq. (11.30)
if $f^{\text{trial}} \geqslant 0$ **then**
  $\gamma = 0$
  $p_{n+1} = p^{\text{trial}}$
  $\boldsymbol{s}_{n+1} = \boldsymbol{s}^{\text{trial}}$
  $\boldsymbol{\alpha}_{n+1} = \boldsymbol{\alpha}^{\text{trial}}$
  while true **do**
    $\boldsymbol{n} = \frac{\boldsymbol{\eta}_{n+1}}{\|\boldsymbol{\eta}_{n+1}\|}$
    compute $\boldsymbol{R}$ and $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$ ▷ Eq. (11.31) and Eq. (11.32)
    $\Delta \boldsymbol{x} = \left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \boldsymbol{R}$
    if $\|\Delta \boldsymbol{x}\| <$ tolerance **then**
      break
    end if
    update γ, p_{n+1}, $\boldsymbol{s}_{n+1}$ and $\boldsymbol{\alpha}_{n+1}$ using the increment $\Delta \boldsymbol{x}$
  end while ▷ Once this while exists, p_{n+1}, $\boldsymbol{s}_{n+1}$ and $\boldsymbol{\alpha}_{n+1}$ are all new states.
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{s}_{n+1} + p_{n+1}\mathbf{1}$
  compute $\boldsymbol{D}_{n+1}$ ▷ Eq. (11.38)
else
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}$
  $\boldsymbol{\alpha}^p_{n+1} = \boldsymbol{\alpha}^p_n$
  $\boldsymbol{D}_{n+1} = \boldsymbol{D}$
end if

It can be seen that with such a framework, the state determination can be kept quite concise.

11.4. Dafalias–Manzari Sand Model

11.4.1. Theory

Hyperelasticity

The hyperelastic response is defined as

$$G = G_0 \frac{(2.97 - e)^2}{1 + e} \sqrt{p p_{at}}, \qquad K = \frac{2}{3} \frac{1 + \nu}{1 - 2\nu} G. \tag{11.42}$$

The corresponding derivatives are

$$\frac{\partial G}{\partial e} = G_0 \sqrt{p p_{at}} \frac{e^2 + 2e - 14.7609}{(1 + e)^2}, \qquad \frac{\partial G}{\partial p} = G_0 \frac{(2.97 - e)^2}{1 + e} \frac{1}{2} \sqrt{\frac{p_{at}}{p}}. \tag{11.43}$$

The void ratio can be associated to strain so that

$$e = e_0 + (1 + e_0) \operatorname{trace} (\boldsymbol{\varepsilon}_{n+1}) . \tag{11.44}$$

The strain increment can be decomposed into elastic and plastic parts.

$$\boldsymbol{\varepsilon}_{n+1} = \boldsymbol{\varepsilon}_n + \dot{\boldsymbol{\varepsilon}} = \boldsymbol{\varepsilon}_n + \dot{\boldsymbol{\varepsilon}^e} + \dot{\boldsymbol{\varepsilon}^p}. \tag{11.45}$$

As such, the stress increment can be expressed accordingly,

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n + \dot{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_n + 2G \left(\dot{\boldsymbol{\varepsilon}^d} - \dot{\boldsymbol{\varepsilon}^{d,p}} \right) + K \left(\dot{\varepsilon^v} - \dot{\varepsilon^{v,p}} \right) \mathbf{1}. \tag{11.46}$$

We do not use trial state any more since in this model the elastic part is not linear.

In derivatoric and spherical components,

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{s}_{n+1} + p_{n+1} \mathbf{1}, \tag{11.47}$$

$$p_{n+1} = p_n + K \left(\dot{\varepsilon^v} - \dot{\varepsilon^{v,p}} \right) , \tag{11.48}$$

$$\boldsymbol{s}_{n+1} = \boldsymbol{s}_n + 2G \left(\dot{\boldsymbol{\varepsilon}^d} - \dot{\boldsymbol{\varepsilon}^{d,p}} \right) , \tag{11.49}$$

with

$$\dot{\boldsymbol{\varepsilon}} = \dot{\boldsymbol{\varepsilon}^d} + \frac{1}{3} \dot{\varepsilon^v} \mathbf{1}, \tag{11.50}$$

where $\boldsymbol{s} = \operatorname{dev} (\boldsymbol{\sigma})$ is the deviatoric stress, $p = \frac{1}{3} \operatorname{trace} (\boldsymbol{\sigma})$ is the hydrostatic stress.

Critical State

The critical state parameter is chosen as

$$\psi = e - e_0 + \lambda_c \left(\frac{p}{p_{at}} \right)^{\xi}. \tag{11.51}$$

The derivatives are

$$\frac{\partial \psi}{\partial e} = 1, \qquad \frac{\partial \psi}{\partial p} = \lambda_c \xi \left(\frac{p}{p_{at}} \right)^{\xi - 1} \frac{1}{p_{at}}. \tag{11.52}$$

The dilatancy surface is defined as

$$\alpha^d = \alpha^c \exp\left(n^d \psi \right). \tag{11.53}$$

The bounding surface is defined as

$$\alpha^b = \alpha^c \exp\left(-n^b \psi \right). \tag{11.54}$$

The corresponding derivatives are

$$\frac{\partial \alpha^d}{\partial \psi} = n^d \alpha^d, \qquad \frac{\partial \alpha^b}{\partial \psi} = -n^b \alpha^b. \tag{11.55}$$

Yield Function

A wedge-like function is chosen to be the yield surface.

$$F = \| \boldsymbol{s} + p\boldsymbol{\alpha} \| + mp = \| \boldsymbol{\eta} \| + mp, \tag{11.56}$$

where $\boldsymbol{\alpha}$ is the so called back stress and m characterises the size of the wedge. For simplicity, m is assumed to be a constant in this model.

By denoting $\boldsymbol{\eta} = \boldsymbol{s} + p\boldsymbol{\alpha}$, the directional unit tensor is defined as

$$\boldsymbol{n} = \frac{\boldsymbol{\eta}}{\| \boldsymbol{\eta} \|}. \tag{11.57}$$

Flow Rule

A non-associated plastic flow is used, the corresponding flow rule is defined as follows.

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \left(\boldsymbol{n} + \frac{1}{3} D \mathbf{1} \right), \tag{11.58}$$

where D is the dilatancy parameter.

$$D = A_d \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) = A_0 \left(1 + \langle \boldsymbol{z} : \boldsymbol{n} \rangle \right) \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right). \tag{11.59}$$

For $\boldsymbol{z} : \boldsymbol{n} \geqslant 0$,

$$\frac{\partial D}{\partial p} = A_0 \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) (\boldsymbol{z} : n_p) + A_0 \left(1 + \boldsymbol{z} : \boldsymbol{n} \right) \left(\frac{\partial \alpha^d}{\partial p} - \boldsymbol{\alpha} : n_p \right), \tag{11.60}$$

$$\frac{\partial D}{\partial \boldsymbol{s}} = A_0 \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) (\boldsymbol{z} : n_s) - A_0 \left(1 + \boldsymbol{z} : \boldsymbol{n} \right) (\boldsymbol{\alpha} : n_s), \tag{11.61}$$

$$\frac{\partial D}{\partial \boldsymbol{\alpha}} = A_0 \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) (\boldsymbol{z} : n_\alpha) - A_0 \left(1 + \boldsymbol{z} : \boldsymbol{n} \right) (\boldsymbol{\alpha} : n_\alpha + \boldsymbol{n} : \mathbb{I}), \tag{11.62}$$

$$\frac{\partial D}{\partial \boldsymbol{z}} = A_0 \left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n} \right) (\boldsymbol{n} : \mathbb{I}). \tag{11.63}$$

For $\boldsymbol{z} : \boldsymbol{n} < 0$,

$$\frac{\partial D}{\partial p} = A_0 \left(\frac{\partial \alpha^d}{\partial p} - \boldsymbol{\alpha} : n_p \right), \tag{11.64}$$

$$\frac{\partial D}{\partial \boldsymbol{s}} = -A_0 (\boldsymbol{\alpha} : n_s), \tag{11.65}$$

$$\frac{\partial D}{\partial \boldsymbol{\alpha}} = -A_0 (\boldsymbol{\alpha} : n_\alpha + \boldsymbol{n} : \mathbb{I}), \tag{11.66}$$

$$\frac{\partial D}{\partial \boldsymbol{z}} = \mathbf{0}. \tag{11.67}$$

In the above expressions, n_p, n_s and n_α are partial derivatives of $\boldsymbol{n}$ against p, $\boldsymbol{s}$ and $\boldsymbol{\alpha}$. Note due to the change of sign convention, a negative D leads to contractive response. Thus, A_0 often needs to be negative.

Hardening Law

The evolution rate of the back stress $\boldsymbol{\alpha}$ is defined in terms of a proper distance measure from the bounding surface,

$$\dot{\boldsymbol{\alpha}} = \gamma h \left(\left(\alpha^b - m \right) \boldsymbol{n} - \boldsymbol{\alpha} \right), \tag{11.68}$$

where h controls the hardening rate,

$$h = b_0 \exp\left(h_1 \left(\boldsymbol{\alpha}_{in} : \boldsymbol{n} - \boldsymbol{\alpha} : \boldsymbol{n}\right)\right). \tag{11.69}$$

The constant $\boldsymbol{\alpha}_{in}$ is updated whenever load reversal occurs.

The parameter b_0 is defined as a function of current state,

$$b_0 = G_0 h_0 \left(1 - c_h e\right) \sqrt{\frac{p_{at}}{p}}. \tag{11.70}$$

The derivatives are

$$\frac{\partial b_0}{\partial e} = -G_0 h_0 c_h \sqrt{\frac{p_{at}}{p}}, \qquad \frac{\partial b_0}{\partial p} = -G_0 h_0 \left(1 - c_h e\right) \frac{\sqrt{p p_{at}}}{2p^2} = -\frac{b_0}{2p}. \tag{11.71}$$

Hence,

$$\frac{\partial h}{\partial p} = \frac{\partial b_0}{\partial p} \exp\left(h_1 \left(\boldsymbol{\alpha}_{in} : \boldsymbol{n} - \boldsymbol{\alpha} : \boldsymbol{n}\right)\right), \tag{11.72}$$

$$\frac{\partial h}{\partial \boldsymbol{s}} = h h_1 \left(\boldsymbol{\alpha}_{in} - \boldsymbol{\alpha}\right) : n_s, \tag{11.73}$$

$$\frac{\partial h}{\partial \boldsymbol{\alpha}} = h h_1 \left(\left(\boldsymbol{\alpha}_{in} - \boldsymbol{\alpha}\right) : n_a - \boldsymbol{n} : \mathbb{I}\right). \tag{11.74}$$

Fabric Effect

The fabric tensor changes when $\Delta\varepsilon_v^p$ is positive,

$$\dot{\boldsymbol{z}} = c_z \left\langle \dot{\varepsilon^{v,p}} \right\rangle \left(z_m \boldsymbol{n} - \boldsymbol{z}\right) = c_z \gamma \left\langle D \right\rangle \left(z_m \boldsymbol{n} - \boldsymbol{z}\right). \tag{11.75}$$

The sign is flipped compared to the original definition for consistency with other parts.

11.4.2. Formulation

The governing equations of this model are

Elastic Residual

The new state shall be computed assuming there is no plasticity. In which case,

$$p_{n+1} = p_n + K\dot{\varepsilon^v}, \tag{11.76}$$

$$\boldsymbol{s}_{n+1} = \boldsymbol{s}_n + 2G\dot{\boldsymbol{\varepsilon}^d}. \tag{11.77}$$

Constitutive Law	$\boldsymbol{\sigma} = \boldsymbol{\sigma}_n + 2G\left(\dot{\boldsymbol{\varepsilon}}^d - \dot{\boldsymbol{\varepsilon}}^{d,p}\right) + K\left(\dot{\varepsilon}^v - \dot{\varepsilon}^{v,p}\right)\mathbf{1}$
Yield Function	$f = \|\boldsymbol{s} + p\boldsymbol{\alpha}\| + mp = \|\boldsymbol{\eta}\| + mp$
Flow Rule	$\dot{\boldsymbol{\varepsilon}}^p = \gamma\left(\boldsymbol{n} + \frac{1}{3}\left(A_0\left(1 + \langle \boldsymbol{z} : \boldsymbol{n}\rangle\right)\left(\alpha^d - m - \boldsymbol{\alpha} : \boldsymbol{n}\right)\right)\mathbf{1}\right)$
Hardening Law	$\dot{\boldsymbol{\alpha}} = \gamma b_0 \exp\left(h_1\left(\boldsymbol{\alpha}_{in} : \boldsymbol{n} - \boldsymbol{\alpha} : \boldsymbol{n}\right)\right)\left(\left(\alpha^b - m\right)\boldsymbol{n} - \boldsymbol{\alpha}\right)$
Fabric Effect	$\dot{\boldsymbol{z}} = c_z\left\langle \dot{\varepsilon}^{v,p}\right\rangle\left(z_m\boldsymbol{n} - \boldsymbol{z}\right)$
Critical State	$\psi = e - e_0 + \lambda_c\left(\frac{p}{p_{at}}\right)^{\xi}$
	$\alpha^d = \alpha^c \exp\left(n^d\psi\right)$
	$\alpha^b = \alpha^c \exp\left(-n^b\psi\right)$

Elastic Local Iteration

The independent variables are chosen to be $\boldsymbol{x} = \begin{bmatrix} p & \boldsymbol{s}\end{bmatrix}^{\mathrm{T}}$, then the local residual is

$$\boldsymbol{R} = \begin{cases} p_{n+1} - p_n - K\dot{\varepsilon}^v, \\ \boldsymbol{s}_{n+1} - \boldsymbol{s}_n - 2G\dot{\boldsymbol{\varepsilon}}^d. \end{cases} \tag{11.78}$$

The Jacobian can be expressed as

$$\boldsymbol{J} = \begin{bmatrix} 1 - \dot{\varepsilon}^v\dfrac{2+2\nu}{3-6\nu}\dfrac{\partial G}{\partial p} & \mathbf{0} \\ -2\dot{\boldsymbol{\varepsilon}}^d\dfrac{\partial G}{\partial p} & \boldsymbol{I} \end{bmatrix}. \tag{11.79}$$

and

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} -K\mathbf{1}^{\mathrm{T}} - \dot{\varepsilon}^v\dfrac{\partial K}{\partial \boldsymbol{\varepsilon}_{n+1}} \\ -2G\mathbb{I}^{\mathrm{dev}} - 2\dot{\boldsymbol{\varepsilon}}^d\dfrac{\partial G}{\partial \boldsymbol{\varepsilon}_{n+1}} \end{bmatrix}, \tag{11.80}$$

which will be used in computation of consistent tangent stiffness.

When elastic state is computed, the trial yield function can be calculated.

$$f^{\mathrm{trial}} = \|\boldsymbol{s}_{n+1} + p_{n+1}\boldsymbol{\alpha}_n\| + mp_{n+1}. \tag{11.81}$$

Plastic Residual

The are five local residual equations.

$$\boldsymbol{R} = \begin{cases} \|\boldsymbol{\eta}\| + mp, \\ p - p_n + K\left(\gamma D - \dot{\varepsilon^v}\right), \\ \boldsymbol{s} - \boldsymbol{s}_n + 2G\left(\gamma \boldsymbol{n} - \dot{\boldsymbol{\varepsilon}^d}\right), \\ \boldsymbol{\alpha} - \boldsymbol{\alpha}_n + \gamma h\left(\boldsymbol{\alpha} - \left(\alpha^b - m\right)\boldsymbol{n}\right), \\ \boldsymbol{z} - \boldsymbol{z}_n + c_z \gamma \langle D \rangle \left(\boldsymbol{z} - z_m \boldsymbol{n}\right). \end{cases} \tag{11.82}$$

Plastic Local Iteration

By choosing $\boldsymbol{x} = \begin{bmatrix} \gamma & p & \boldsymbol{s} & \boldsymbol{\alpha} & \boldsymbol{z} \end{bmatrix}^{\mathrm{T}}$, the Jacobian consists of the entries that can be listed as follows.

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial \boldsymbol{R}}{\partial \gamma} & \frac{\partial \boldsymbol{R}}{\partial p} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{s}} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\alpha}} & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{z}} \end{bmatrix} \tag{11.83}$$

with

$$\frac{\partial \boldsymbol{R}}{\partial \gamma} = \begin{bmatrix} \cdot \\ DK \\ 2G\boldsymbol{n} \\ h\left(\boldsymbol{\alpha} - \alpha^{bm}\boldsymbol{n}\right) \\ c_z \langle D \rangle \boldsymbol{z}_z \end{bmatrix}, \tag{11.84}$$

$$\frac{\partial \boldsymbol{R}}{\partial p} = \begin{bmatrix} \eta_p + m \\ 1 + \frac{\partial K}{\partial p}\left(\gamma D - \dot{\varepsilon^v}\right) + K\gamma \frac{\partial D}{\partial p} \\ 2\frac{\partial G}{\partial p}\left(\gamma \boldsymbol{n} - \dot{\boldsymbol{\varepsilon}^d}\right) + 2G\gamma n_p \\ \gamma \frac{\partial h}{\partial p}\left(\boldsymbol{\alpha} - \alpha^{bm}\boldsymbol{n}\right) - \gamma h\left(\frac{\partial \alpha^b}{\partial p}\boldsymbol{n} + \alpha^{bm} n_p\right) \\ c_z \gamma \frac{\partial \langle D \rangle}{\partial p}\boldsymbol{z}_z - c_z \gamma \langle D \rangle z_m n_p \end{bmatrix}, \tag{11.85}$$

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{s}} = \begin{bmatrix} \eta_s \\ \gamma K \left(\dfrac{\partial D}{\partial \boldsymbol{s}} \right)^{\mathrm{T}} \\ \boldsymbol{I} + 2G\gamma n_s \\ \gamma \left(\boldsymbol{\alpha} - \alpha^{bm} \boldsymbol{n} \right) \otimes \dfrac{\partial h}{\partial \boldsymbol{s}} - \gamma h \alpha^{bm} n_s \\ c_z \gamma \boldsymbol{z}_z \otimes \dfrac{\partial \langle D \rangle}{\partial \boldsymbol{s}} - c_z \gamma \langle D \rangle z_m n_s \end{bmatrix}, \tag{11.86}$$

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\alpha}} = \begin{bmatrix} \eta_\alpha \\ \gamma K \left(\dfrac{\partial D}{\partial \boldsymbol{\alpha}} \right)^{\mathrm{T}} \\ 2G\gamma n_\alpha \\ (1 + \gamma h) \boldsymbol{I} + \gamma \left(\boldsymbol{\alpha} - \alpha^{bm} \boldsymbol{n} \right) \otimes \dfrac{\partial h}{\partial \boldsymbol{\alpha}} - \gamma h \alpha^{bm} n_a \\ c_z \gamma \boldsymbol{z}_z \otimes \dfrac{\partial \langle D \rangle}{\partial \boldsymbol{\alpha}} - c_z \gamma \langle D \rangle z_m n_\alpha \end{bmatrix}, \tag{11.87}$$

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{z}} = \begin{bmatrix} \cdot \\ \gamma K \left(\dfrac{\partial D}{\partial \boldsymbol{z}} \right)^{\mathrm{T}} \\ \cdot \\ \cdot \\ (1 + c_z \gamma \langle D \rangle) \boldsymbol{I} + c_z \gamma \boldsymbol{z}_z \otimes \dfrac{\partial \langle D \rangle}{\partial \boldsymbol{z}} \end{bmatrix}. \tag{11.88}$$

In which,

$$\alpha^{bm} = \alpha^b - m, \tag{11.89}$$

$$\boldsymbol{z}_z = \boldsymbol{z} - z_m \boldsymbol{n}. \tag{11.90}$$

And η_p, η_s and η_α are partial derivatives of $\|\boldsymbol{\eta}\|$ against p, $\boldsymbol{s}$ and $\boldsymbol{\alpha}$.

Consistent Tangent Operator

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \begin{bmatrix} \cdot \\ \left((\gamma D - \dot{\varepsilon^v}) \dfrac{\partial K}{\partial \boldsymbol{\varepsilon}_{n+1}} + K \left(\gamma \dfrac{\partial D}{\partial \boldsymbol{\varepsilon}_{n+1}} - \mathbf{1} \right) \right)^{\mathrm{T}}, \\ 2 \left(\gamma \boldsymbol{n} - \dot{\boldsymbol{\varepsilon}^d} \right) \otimes \dfrac{\partial G}{\partial \boldsymbol{\varepsilon}_{n+1}} - 2G \mathbb{I}^{\mathrm{dev}}, \\ \gamma \left(\boldsymbol{\alpha} - \alpha^{bm} \boldsymbol{n} \right) \otimes \dfrac{\partial h}{\partial \boldsymbol{\varepsilon}_{n+1}} - \gamma h \boldsymbol{n} \otimes \dfrac{\partial \alpha^b}{\partial \boldsymbol{\varepsilon}_{n+1}}, \\ c_z \gamma \boldsymbol{z}_z \otimes \dfrac{\partial \langle D \rangle}{\partial \boldsymbol{\varepsilon}_{n+1}} \end{bmatrix}. \tag{11.91}$$

Then,

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\varepsilon}_{n+1}} = -\boldsymbol{J}^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{11.92}$$

Similar to the previous procedure,

$$\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \frac{\partial \boldsymbol{s}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} + \mathbf{1} \otimes \frac{\partial p_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{11.93}$$

The corresponding quantities can be extracted and be used to formulate consistent tangent stiffness.

11.4.3. Implementation

Algorithm 11 state determination of Dafalias–Manzari sand model

Parameter: G_0, v, p_{at}, e_0, λ_c, ξ, n^d, n^b, α_c, m, A_0, h_0, h_1, c_z, z_m
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\sigma}_n$, $\boldsymbol{\alpha}_n$, $\boldsymbol{z}_n$
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\sigma}_{n+1}$, $\boldsymbol{\alpha}_{n+1}$, $\boldsymbol{z}_{n+1}$
$\boldsymbol{s}_{n+1} = \boldsymbol{s}_n = \mathrm{dev}\,(\boldsymbol{\sigma}_n)$
$p_{n+1} = p_n = \frac{1}{3}\mathrm{trace}\,(\boldsymbol{\sigma}_n)$
while true **do**
  compute $\boldsymbol{R}$ and $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$ ▷ Eq. (11.78) and Eq. (11.79)
  $\Delta \boldsymbol{x} = \left(\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \right)^{-1} \boldsymbol{R}$
  if $\|\Delta \boldsymbol{x}\| <$ tolerance **then**
    break
  end if
  update p_{n+1} and $\boldsymbol{s}_{n+1}$ using the increment $\Delta \boldsymbol{x}$
end while ▷ Once this while exists, p_{n+1} and $\boldsymbol{s}_{n+1}$ are new elastic states.

compute f^{trial} ▷ Eq. (11.81)
$\boldsymbol{\alpha}_{n+1} = \boldsymbol{\alpha}_n$
$\boldsymbol{z}_{n+1} = \boldsymbol{z}_n$
if $f^{\text{trial}} \geqslant 0$ **then** ▷ plastic
  $\gamma = 0$
  while true **do**
    $\boldsymbol{n} = \dfrac{\boldsymbol{\eta}_{n+1}}{\|\boldsymbol{\eta}_{n+1}\|}$
    compute $\boldsymbol{R}$ and $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$ ▷ Eq. (11.82) and Eq. (11.83)
    $\Delta \boldsymbol{x} = \left(\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \boldsymbol{R}$
    if $\|\Delta x\| <$ tolerance **then**
      break
    end if
    update γ, p_{n+1}, $\boldsymbol{s}_{n+1}$, $\boldsymbol{\alpha}_{n+1}$ and $\boldsymbol{z}_{n+1}$ using the increment $\Delta \boldsymbol{x}$
  end while ▷ Once this while exists, p_{n+1}, $\boldsymbol{s}_{n+1}$, $\boldsymbol{\alpha}_{n+1}$ and $\boldsymbol{z}_{n+1}$ are all new states.
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{s}_{n+1} + p_{n+1}\mathbf{1}$
  compute plastic $\boldsymbol{D}_{n+1}$ ▷ Eq. (11.93)
else ▷ elastic
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{s}_{n+1} + p_{n+1}\mathbf{1}$
  compute elastic $\boldsymbol{D}_{n+1}$ ▷ Eq. (11.80)
end if

The Dafalias–Manzari sand model is probably the most complex model so far as it involves heavy computation of tensor quantities. Here is a example implementation of state determination.

```
int DafaliasManzari::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    const auto current_p = tensor::mean3(current_stress);
    const auto current_s = tensor::dev(current_stress);
    const auto incre_ev = tensor::trace3(incre_strain);
    const vec incre_ed = unit_dev_tensor * incre_strain;

    // assume no plasticity
    // compute trial stress

    auto p = current_p + pr * gi * incre_ev;
    vec s = current_s + 2. * gi * incre_ed;

    const auto void_ratio = e0 + (1. + e0) * tensor::trace3(trial_strain);
    const auto v_term_a = pow(2.97 - void_ratio, 2.) / (1. + void_ratio);
    const auto v_term_b = (void_ratio * (void_ratio + 2.) - 14.7609) * pow(1. + void_ratio,
    ↪  -2.) * (1. + e0);

    double g, pgpe, pgpp;

```

```
    vec residual(7, fill::none), incre;
    mat jacobian(7, 7, fill::eye);

    auto counter = 0u;
    auto ref_error = 1.;

    while(true) {
        if(max_iteration == ++counter) return SUANPAN_FAIL;

        const auto sqrt_term = shear_modulus * sqrt(std::max(datum::eps, pc * p));

        g = sqrt_term * v_term_a;

        if(g > gi) {
            pgpe = sqrt_term * v_term_b;
            pgpp = .5 * g / p;
        }
        else {
            g = gi;
            pgpe = pgpp = 0.;
        }

        residual(sa) = p - current_p - pr * g * incre_ev;
        residual(sb) = s - current_s - 2. * g * incre_ed;

        jacobian(sa, sa) = 1. - pr * incre_ev * pgpp;
        jacobian(sb, sa) = -2. * pgpp * incre_ed;

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        auto error = norm(residual);
        if(1 == counter) ref_error = std::max(1., error);
        suanpan_debug("DafaliasManzari local elastic iteration error: %.5E.\n", error /=
        ↪  ref_error);
        if(error <= tolerance) break;

        p -= incre(sa);
        s -= incre(sb);
    }

    // check if yield

    const vec current_alpha(&current_history(0), 6, false, true);

    vec eta = s + p * current_alpha;
    auto norm_eta = tensor::stress::norm(eta);

    if(norm_eta + m * p < 0.) {
        trial_stress = s + p * tensor::unit_tensor2;

        mat left(7, 6, fill::none), right;

        left.row(sa) = pr * (incre_ev * pgpe + g) * tensor::unit_tensor2.t();
        left.rows(sb) = 2. * pgpe * incre_ed * tensor::unit_tensor2.t() + 2. * g *
        ↪  unit_dev_tensor;

        if(!solve(right, jacobian, left)) return SUANPAN_FAIL;
```

```
78
79          trial_stiffness = right.rows(sb);
80          trial_stiffness.row(0) += right.row(sa);
81          trial_stiffness.row(1) += right.row(sa);
82          trial_stiffness.row(2) += right.row(sa);
83
84          return SUANPAN_SUCCESS;
85      }
86
87      // yield function violated
88
89      const vec current_z(&current_history(6), 6, false, true);
90
91      trial_history = current_history;
92      vec alpha(&trial_history(0), 6, false, true);
93      vec z(&trial_history(6), 6, false, true);
94      vec ini_alpha(&trial_history(12), 6, false, true);
95
96      residual.set_size(20);
97      jacobian.set_size(20, 20);
98      jacobian(si, si) = 0.;
99      jacobian(si, sm).zeros();
100     jacobian(sk, sm).zeros();
101     jacobian(sl, sm).zeros();
102
103     counter = 0u;
104
105     vec n, zz, aabmn;
106     auto gamma = 0.;
107     double pabpe, d, pdpe, h, phpe;
108     auto update_ini_alpha = false;
109
110     while(true) {
111         if(max_iteration == ++counter) return SUANPAN_FAIL;
112
113         // shear modulus
114
115         auto tmp_term = shear_modulus * sqrt(std::max(datum::eps, pc * p));
116         g = tmp_term * v_term_a;
117
118         if(g > gi) {
119             pgpe = tmp_term * v_term_b;
120             pgpp = .5 * g / p;
121         }
122         else {
123             g = gi;
124             pgpe = pgpp = 0.;
125         }
126
127         // state parameter
128
129         tmp_term = lc * pow(std::max(datum::eps, p / pc), xi);
130         const auto psi = void_ratio - e0 + tmp_term;
131         const auto ppsipp = xi * tmp_term / p;
132
133         // surface
134
```

```
        const auto ad = ac * exp(nd * psi);
        const auto ab = ac * exp(-nb * psi);
        const auto adm = ad - m;
        const auto abm = ab - m;

        auto padpe = nd * ad;
        pabpe = -nb * ab;
        const auto padpp = padpe * ppsipp;
        const auto pabpp = pabpe * ppsipp;
        padpe *= 1. + e0;
        pabpe *= 1. + e0;

        // yield function

        eta = s + p * alpha;
        norm_eta = tensor::stress::norm(eta);

        n = eta / norm_eta;
        const vec unit_n = n % tensor::stress::norm_weight;
        const vec unit_alpha = alpha % tensor::stress::norm_weight;
        const auto alpha_n = dot(n, unit_alpha);
        aabmn = alpha - abm * n;

        const vec np = (alpha - alpha_n * n) / norm_eta;
        const mat ns = (eye(6, 6) - n * unit_n.t()) / norm_eta;

        // dilatancy

        const vec unit_z = z % tensor::stress::norm_weight;
        const auto zn = dot(n, unit_z);

        d = a * (adm - alpha_n);

        double pdpp;
        rowvec pdps, pdpa, pdpz;
        if(zn > 0.) {
            const auto term_a = a * (1. + zn);

            pdpe = term_a * padpe;
            pdpp = term_a * padpp + dot(d * unit_z - term_a * unit_alpha, np);
            pdps = (d * unit_z - term_a * unit_alpha).t() * ns;
            pdpa = p * pdps - term_a * unit_n.t();
            pdpz = d * unit_n.t();

            d *= 1. + zn;
        }
        else {
            pdpe = a * padpe;
            pdpp = a * (padpp - dot(unit_alpha, np));
            pdps = -a * unit_alpha.t() * ns;
            pdpa = p * pdps - a * unit_n.t();
            pdpz.zeros(6);
        }

        // hardening

        tmp_term = shear_modulus * h0 * sqrt(std::max(datum::eps, pc / p));
```

```
        const auto b0 = tmp_term * (1. - ch * void_ratio);
        const auto pb0pe = -ch * tmp_term * (1. + e0);
        const auto pb0pp = -.5 * b0 / p;

        update_ini_alpha = false;
        vec diff_alpha = (ini_alpha - alpha) % tensor::stress::norm_weight;
        tmp_term = exp(h1 * dot(diff_alpha, n));

        if(tmp_term > 1.) {
            update_ini_alpha = true;
            diff_alpha = (current_alpha - alpha) % tensor::stress::norm_weight;
            tmp_term = exp(h1 * dot(diff_alpha, n));
        }

        h = tmp_term * b0;

        phpe = tmp_term * pb0pe;
        const auto phpp = tmp_term * pb0pp;
        const rowvec phps = h * h1 * diff_alpha.t() * ns;
        const rowvec phpa = p * phps - h * h1 * unit_n.t();

        // local iteration

        residual(si) = norm_eta + m * p;
        residual(sj) = p - current_p + pr * g * (gamma * d - incre_ev);
        residual(sk) = s - current_s + 2. * g * (gamma * n - incre_ed);
        residual(sl) = alpha - current_alpha + gamma * h * aabmn;
        residual(sm) = z - current_z;

        jacobian(si, sj) = alpha_n + m;
        jacobian(si, sk) = unit_n.t();
        jacobian(si, sl) = p * jacobian(si, sk);

        const auto gk = gamma * pr * g;
        jacobian(sj, si) = d * pr * g;
        jacobian(sj, sj) = 1. + pr * pgpp * (gamma * d - incre_ev) + gk * pdpp;
        jacobian(sj, sk) = gk * pdps;
        jacobian(sj, sl) = gk * pdpa;
        jacobian(sj, sm) = gk * pdpz;

        jacobian(sk, si) = 2. * g * n;
        jacobian(sk, sj) = 2. * pgpp * (gamma * n - incre_ed) + 2. * g * gamma * np;
        jacobian(sk, sk) = 2. * g * gamma * ns;
        jacobian(sk, sl) = p * jacobian(sk, sk);
        jacobian(sk, sk) += eye(6, 6);

        jacobian(sl, si) = h * aabmn;
        jacobian(sl, sj) = gamma * phpp * aabmn - gamma * h * (pabpp * n + abm * np);
        jacobian(sl, sk) = gamma * aabmn * phps - gamma * h * abm * ns;
        jacobian(sl, sl) = (1. + gamma * h) * eye(6, 6) + gamma * aabmn * phpa - gamma * h *
        ↪  abm * p * ns;

        jacobian(sm, sm) = eye(6, 6);

        if(d > 0.) {
            const auto factor_a = cz * gamma;
            const auto factor_b = factor_a * d;
```

```
            const auto factor_c = factor_b * zm;

            zz = z - zm * n;

            residual(sm) += factor_b * zz;

            jacobian(sm, si) = cz * d * zz;
            jacobian(sm, sj) = factor_a * pdpp * zz - factor_c * np;
            jacobian(sm, sk) = factor_a * zz * pdps - factor_c * ns;
            jacobian(sm, sl) = factor_a * zz * pdpa - factor_c * p * ns;
            jacobian(sm, sm) += factor_b * eye(6, 6) + factor_a * zz * pdpz;
        }
        else {
            jacobian(sm, si).zeros();
            jacobian(sm, sj).zeros();
            jacobian(sm, sk).zeros();
            jacobian(sm, sl).zeros();
        }

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        auto error = norm(residual);
        if(1 == counter) ref_error = std::max(1., error);
        suanpan_debug("DafaliasManzari local plastic iteration error: %.5E.\n", error /=
        ↪  ref_error);
        if(error <= tolerance) break;

        gamma -= incre(si);
        p -= incre(sj);
        s -= incre(sk);
        alpha -= incre(sl);
        z -= incre(sm);
    }

    trial_stress = s + p * tensor::unit_tensor2;

    mat left(20, 6, fill::none), right;

    left.row(si).zeros();
    left.row(sj) = pr * (pgpe * (incre_ev - gamma * d) + g - g * gamma * pdpe) *
    ↪  tensor::unit_tensor2.t();
    left.rows(sk) = 2. * g * unit_dev_tensor + 2. * pgpe * (incre_ed - gamma * n) *
    ↪  tensor::unit_tensor2.t();
    left.rows(sl) = (gamma * h * n * pabpe - gamma * aabmn * phpe) *
    ↪  tensor::unit_tensor2.t();

    if(d > 0.) left.rows(sm) = -cz * gamma * pdpe * zz * tensor::unit_tensor2.t();
    else left.rows(sm).zeros();

    if(!solve(right, jacobian, left)) return SUANPAN_FAIL;

    trial_stiffness = right.rows(sk);
    trial_stiffness.row(0) += right.row(sj);
    trial_stiffness.row(1) += right.row(sj);
    trial_stiffness.row(2) += right.row(sj);

    if(update_ini_alpha) ini_alpha = current_alpha;
```

```
301
302     return SUANPAN_SUCCESS;
303 }
```

11.5. Duncan Soil Model

In this section, we present a plane strain soil model. Unlike other models that are typically strain–driven, it is a non-linear elastic phenomenological stress–driven model.

11.5.1. Theory

The incremental form of the constitutive relationship is defined as

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{D}\dot{\boldsymbol{\varepsilon}} = \frac{3B}{9B-E}\begin{bmatrix} 3B+E & 3B-E & 0 \\ 3B-E & 3B+E & 0 \\ 0 & 0 & E \end{bmatrix}\dot{\boldsymbol{\varepsilon}} \tag{11.94}$$

where $\boldsymbol{\varepsilon} = \begin{bmatrix}\varepsilon_x & \varepsilon_y & \gamma_{xy}\end{bmatrix}^{\mathrm{T}}$ and $\boldsymbol{\sigma} = \begin{bmatrix}\sigma_x & \sigma_y & \tau_{xy}\end{bmatrix}^{\mathrm{T}}$ are the strain and stress vector for conditions of plane strain. As the original model is non-linear elastic, it can be fully defined by the stress state. Thus, the elastic modulus E and the bulk modulus K can be determined solely by the stress vector $\boldsymbol{\sigma}$.

Before proceeding, one shall note that the stress vector $\boldsymbol{\sigma}$ can be converted into principal stresses via either Mohr's circle or eigenanalysis.

$$\begin{matrix}\sigma_1 \\ \sigma_3\end{matrix} = \frac{\sigma_x+\sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x-\sigma_y}{2}\right)^2+\tau_{xy}^2}. \tag{11.95}$$

In the context of geotechnical engineering, typically $|\sigma_1| > |\sigma_3|$ and both take compression as positive. Depending on how $\boldsymbol{\sigma}$ is defined, the signs presented in the above expression may be flipped.

The corresponding derivatives $\dfrac{\mathrm{d}\sigma_1}{\mathrm{d}\boldsymbol{\sigma}}$ and $\dfrac{\mathrm{d}\sigma_3}{\mathrm{d}\boldsymbol{\sigma}}$ can be determined accordingly.

The difference between σ_1 and σ_3 is taken as the deviatoric stress.

$$\sigma_d = \operatorname{dev}(\boldsymbol{\sigma}) = |\sigma_1-\sigma_3| = 2\sqrt{\left(\frac{\sigma_x-\sigma_y}{2}\right)^2+\tau_{xy}^2}. \tag{11.96}$$

Elastic Modulus

The elastic modulus E can be expressed as the product of two components, namely, the initial part and the reduction part.

$$E = E_i \left(1 - \frac{\sigma_d}{\sigma_d^{\max}}\right)^2. \tag{11.97}$$

The initial part E_i increases when σ_3 grows larger.

$$E_i = E_r \left|\frac{\sigma_3}{p_a}\right|^n, \tag{11.98}$$

where E_r is the reference elastic modulus, p_a is the atmospheric pressure, and n is a model parameter that is typically between zero and unity.

The reduction part physically implies that the elastic modulus decreases when the current deviatoric stress approaches the maximum deviatoric stress. This is effectively the Mohr–Coulomb failure criterion. As a result, the deviatoric stress would asymptotically converge to the maximum deviatoric stress that is defined as follows.

$$\sigma_d^{\max} = \frac{\sigma_d^{\text{ult}}}{r_f}, \tag{11.99}$$

with the ultimate deviatoric stress σ_d^{ult} being defined as

$$\sigma_d^{\text{ult}} = 2\frac{c\cos\phi + \sigma_3 \sin\phi}{1 - \sin\phi}. \tag{11.100}$$

It can be solved graphically using the following trigonometry.

$$\left(\sigma_d^{\text{ult}}/2 + \sigma_3 - c\tan\phi\right)\sin\phi = \sigma_d^{\text{ult}}/2 - \frac{c}{\cos\phi}. \tag{11.101}$$

The ratio $r_f < 1$ is introduced to fine–tune the behaviour.

Bulk Modulus

The bulk modulus B takes a similar approach. However, it only monotonically increases with an increasing σ_3.

$$B = B_i = B_r \left|\frac{\sigma_3}{p_a}\right|^m, \tag{11.102}$$

where B_r is the reference bulk modulus and m is another modal parameter similar to n.

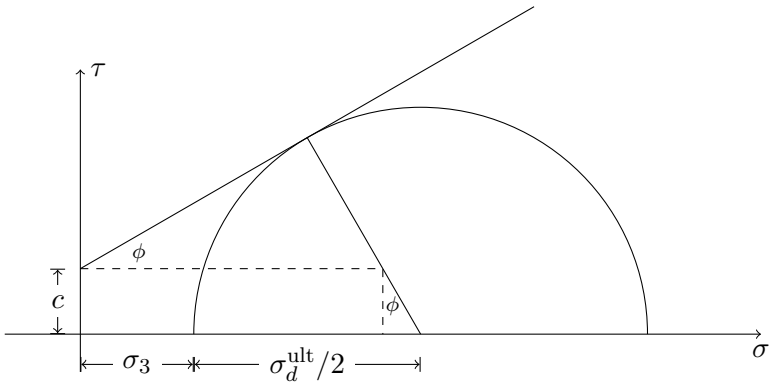


Figure 11.1.: σ_d^{ult} determined by Mohr's circle

11.5.2. Formulation

Local Iteration

The incremental form Eq. (11.94) resembles the elastic Hooke's law. However, since E and B rely on $\boldsymbol{\sigma}$, the consistent tangent stiffness requires the computation of the full derivative.

$$\frac{\mathrm{d}E}{\mathrm{d}\boldsymbol{\sigma}} = \frac{\partial E}{\partial \sigma_1}\frac{\mathrm{d}\sigma_1}{\mathrm{d}\boldsymbol{\sigma}} + \frac{\partial E}{\partial \sigma_3}\frac{\mathrm{d}\sigma_3}{\mathrm{d}\boldsymbol{\sigma}}, \qquad \frac{\mathrm{d}B}{\mathrm{d}\boldsymbol{\sigma}} = \frac{\mathrm{d}E}{\mathrm{d}\sigma_3}\frac{\mathrm{d}\sigma_3}{\mathrm{d}\boldsymbol{\sigma}}. \tag{11.103}$$

Adopting the implicit Euler method, the local residual can be expressed as

$$\boldsymbol{R} = \boldsymbol{\sigma} - \boldsymbol{\sigma}_n - \boldsymbol{D}\Delta\boldsymbol{\varepsilon}. \tag{11.104}$$

The Jacobian can be derived as

$$\boldsymbol{J} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\sigma}} = \boldsymbol{I} - \frac{\mathrm{d}\boldsymbol{D}}{\mathrm{d}\boldsymbol{\sigma}}\Delta\boldsymbol{\varepsilon}. \tag{11.105}$$

The consistent tangent stiffness is thus

$$\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\boldsymbol{\varepsilon}} = \boldsymbol{C} = \boldsymbol{J}^{-1}\boldsymbol{D}. \tag{11.106}$$

One shall note the following two points.

1. The derivative $\dfrac{\mathrm{d}\boldsymbol{D}}{\mathrm{d}\boldsymbol{\sigma}}$ is a cube, since $\boldsymbol{D}$ is a matrix.
2. Although Eq. (11.94) is symmetric and resembles the elastic Hooke's law, the consistent tangent is not symmetric.

Here, we present an example computation of the Jacobian. The first row of the residual vector $\boldsymbol{R}$ can be explicitly written as

$$\boldsymbol{R}^{\langle 1\rangle} = \sigma_x - \sigma_{x,n} - \frac{3B}{9B-E}\left((3B+E)\,\Delta\varepsilon_x + (3B-E)\,\Delta\varepsilon_y\right). \tag{11.107}$$

To avoid computing the derivative of fractions, the following equivalent form is adopted.

$$\boldsymbol{R}^{\langle 1\rangle} = (9B-E)\,(\sigma_x - \sigma_{x,n}) - \left(\left(9B^2+3BE\right)\Delta\varepsilon_x + \left(9B^2-3BE\right)\Delta\varepsilon_y\right). \tag{11.108}$$

The derivative can then be derived as

$$\begin{aligned}\frac{\partial \boldsymbol{R}^{\langle 1\rangle}}{\partial \boldsymbol{\sigma}} = (9B-E)\begin{bmatrix}1 & 0 & 0\end{bmatrix} + (\sigma_x - \sigma_{x,n})\frac{\mathrm{d}\,(9B-E)}{\mathrm{d}\boldsymbol{\sigma}} \\ - \left(\Delta\varepsilon_x \frac{\mathrm{d}\left(9B^2+3BE\right)}{\mathrm{d}\boldsymbol{\sigma}} + \Delta\varepsilon_y \frac{\mathrm{d}\left(9B^2-3BE\right)}{\mathrm{d}\boldsymbol{\sigma}}\right).\end{aligned} \tag{11.109}$$

Similar procedures can be applied to the other two components of $\boldsymbol{R}$.

Plasticity

The original model is a non-linear elastic model, implying no plasticity whatsoever. Similar to the concept discussed in [16], to enable plasticity–like behaviour, one could introduce the concept of yield surface. Since the Mohr–Coulomb failure criterion is used in Eq. (11.97), the yield function $f(\boldsymbol{\sigma})$ can be simply chosen as

$$f(\boldsymbol{\sigma}) = \sigma_d - \sigma_d^{\max}. \tag{11.110}$$

Similar to conventional plasticity theory based models, a return mapping style algorithm can be applied.

The maximum $\sigma_d^{\max}$ shall be recorded, and it is only updated when local iteration converges and the converged stress yields a larger σ_d. Then, whenever $\sigma_d > \sigma_d^{\max}$, $f > 0$ indicating the development of plasticity, the elastic modulus shall be computed according to Eq. (11.97). For $\sigma_d \leqslant \sigma_d^{\max}$, $f < 0$ indicating elastic loading/unloading, then the reduction in E is not considered, viz., $E = E_i$.

Based on the above discussion, one can rewrite Eq. (11.97) as

$$E = E_i\left(1 - \beta\frac{\sigma_d}{\sigma_d^{\max}}\right)^2, \tag{11.111}$$

with

$$\beta = \begin{cases} 0 & \sigma_d \leqslant \sigma_d^{\max}, \\ 1 & \sigma_d > \sigma_d^{\max}. \end{cases} \tag{11.112}$$

The factor β is defined as a function of $\Delta\boldsymbol{\sigma}$ in [16]. In that case, the computation of derivatives is more cumbersome.

11.5.3. Implementation

The state determination algorithm is shown in Algorithm 12. One must note the fact that, since the model is stress driven, it is not as straightforward to obtain plastic strain alike quantities as in conventional strain driven models. If one wants, the plastic strain can be defined as the residual strain when fully unloaded (strain is zero). However, due to its non-linearity nature, it is not guaranteed that the development of plasticity is irreversible, which mainly depends on model parameters.

Algorithm 12 state determination of Duncan soil model

Parameter: p_a, E_r, n, B_r, m, ϕ_i, $\Delta\phi$, r_f, c
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\sigma}_n$, $\sigma_d^{\max}$
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\sigma}_{n+1}$
$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n$
while true **do**
  compute σ^d based on $\boldsymbol{\sigma}_{n+1}$
  if $\sigma^d \geqslant \sigma_d^{\max}$ **then**
    compute E and B ▷ with reduction
  else
    compute E and B ▷ without reduction
  end if
  compute $\boldsymbol{D}_{n+1}$ based on E and B
  compute $\boldsymbol{R}$ and $\boldsymbol{J}$
  $\Delta\boldsymbol{\sigma} = \boldsymbol{J}^{-1}\boldsymbol{R}$
  if convergence **then**
    update $\sigma_d^{\max}$ if σ_d exceeds $\sigma_d^{\max}$
    $\boldsymbol{C} = \boldsymbol{J}^{-1}\boldsymbol{D}_{n+1}$
    return
  end if
  $\boldsymbol{\sigma}_{n+1} \leftarrow \boldsymbol{\sigma}_{n+1} - \Delta\boldsymbol{\sigma}$
end while

Numerical Robustness

The above algorithm combines both elastic and plastic branches. For a given time step, if the stress state stays within the elastic/plastic domain, the above algorithm would not typically

fail. However, if it crosses the boundary, for example, from elastic to plastic, since the tangent stiffness matrix is discontinuous, the local Newton iteration may fail.

To address this issue, the state determination can be divided into two stages. In the first stage, assume proportional loading and find the elastic portion that projects the stress state onto the yield surface, in specific, find the scalar parameter γ such that

$$\operatorname{dev}(\boldsymbol{\sigma}_r) - \sigma_d^{\max} = 0, \tag{11.113}$$

with

$$\boldsymbol{\sigma}_r = \boldsymbol{\sigma}_n + \boldsymbol{D}\dot{\boldsymbol{\varepsilon}}\gamma. \tag{11.114}$$

After obtaining γ, for the second stage, assume plastic response, consume the remaining $(1-\gamma)\,\dot{\boldsymbol{\varepsilon}}$. The residual becomes

$$\boldsymbol{R} = \boldsymbol{\sigma} - \boldsymbol{\sigma}_r - \boldsymbol{D}\dot{\boldsymbol{\varepsilon}}\,(1-\gamma)\,. \tag{11.115}$$

Algorithm 13 dual–stage state determination of Duncan soil model

Parameter: p_a, E_r, n, B_r, m, ϕ_i, $\Delta\phi$, r_f, c
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\sigma}_n$, $\sigma_d^{\max}$
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\sigma}_{n+1}$
return if Algorithm 12 succeeds
assume elastic response, find γ using Eq. (11.113) and Eq. (11.114) as residuals
assume plastic response, iterate out Eq. (11.115)

Reference Implementation

The main body of the implementation can be shown as follows.

```
int DuncanSelig::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= datum::eps) return SUANPAN_SUCCESS;

    trial_history = current_history;

    trial_stress = current_stress + current_stiffness * incre_strain;

    const auto update_dev_stress = [&] {
        auto& max_dev_stress = trial_history(0);
        if(const auto trail_dev_stress = dev(trial_stress); trail_dev_stress >
        ↪  max_dev_stress) max_dev_stress = trail_dev_stress;
        return SUANPAN_SUCCESS;
    };

    // first try a whole step size local iteration
    if(SUANPAN_SUCCESS == local_update(current_stress, incre_strain, false)) return
    ↪  update_dev_stress();
```

```
18
19      // if that fails, mostly likely due to discontinuity of the gradient
20      // assume proportional loading, project the stress onto the yield surface using elastic
        ↪ moduli
21      auto multiplier = 1.;
22      if(SUANPAN_SUCCESS != project_onto_surface(multiplier)) return SUANPAN_FAIL;
23
24      // then using plastic moduli to compute the new plastic state
25      // !!! the tangent operator is not algorithmically consistent in this case
26      if(SUANPAN_SUCCESS != local_update(vec(trial_stress), (1. - multiplier) * incre_strain,
        ↪ true)) return SUANPAN_FAIL;
27
28      return update_dev_stress();
29  }
```

The local iteration is a standard Newton iteration implementation.

```
1   int DuncanSelig::local_update(const vec& ref_stress, const vec& ref_strain, const bool
    ↪ two_stage) {
2       const auto update_moduli = [&] {
3           const auto max_dev_stress = trial_history(0);
4           return two_stage || dev(trial_stress) > max_dev_stress ? compute_plastic_moduli() :
            ↪ compute_elastic_moduli();
5       };
6
7       auto ref_error = 0.;
8       vec3 incre;
9
10      auto counter = 0u;
11      while(true) {
12          if(max_iteration == ++counter) {
13              suanpan_error("Local iteration cannot converge within {} iterations.\n",
                ↪ max_iteration);
14              return SUANPAN_FAIL;
15          }
16
17          const auto [elastic, bulk, deds, dkds] = update_moduli();
18
19          const auto factor_a = 3. * bulk * (3. * bulk + elastic);
20          const auto factor_b = 3. * bulk * (3. * bulk - elastic);
21
22          mat33 right(fill::zeros);
23          right(0, 0) = right(1, 1) = factor_a;
24          right(0, 1) = right(1, 0) = factor_b;
25          right(2, 2) = 3. * bulk * elastic;
26
27          const vec3 t_stress = trial_stress - ref_stress;
28          const vec3 residual = (9. * bulk - elastic) * t_stress - right * ref_strain;
29
30          const rowvec3 factor_c = 3. * (elastic * dkds + bulk * deds);
31          const rowvec3 dfads = 18. * bulk * dkds + factor_c;
32          const rowvec3 dfbds = 18. * bulk * dkds - factor_c;
33
34          mat33 jacobian = (9. * bulk - elastic) * eye(3, 3) + t_stress * (9. * dkds - deds);
```

```
        jacobian.row(0) -= ref_strain(0) * dfads + ref_strain(1) * dfbds;
        jacobian.row(1) -= ref_strain(0) * dfbds + ref_strain(1) * dfads;
        jacobian.row(2) -= ref_strain(2) * factor_c;

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        const auto error = inf_norm(incre);
        if(1u == counter) ref_error = error;
        suanpan_debug("Local iteration error: {:.5E}.\n", error / ref_error);
        if(error < tolerance * ref_error || ((error < tolerance || inf_norm(residual) <
        ↪  tolerance) && counter > 5u)) {
            if(!solve(trial_stiffness, jacobian, right)) return SUANPAN_FAIL;

            return SUANPAN_SUCCESS;
        }

        trial_stress -= incre;
    }
}
```

The computation of two moduli follows the formulation with extra cumbersome computation of derivatives. To improve numerical robustness, additional limits are applied. Stress components are bounded to the atmospheric pressure.

```
std::tuple<double, double> DuncanSelig::compute_elastic(const double s3) const {
    double elastic, deds3;
    if(s3 < -min_ratio * p_atm) {
        elastic = ref_elastic * std::pow(.01, n);
        deds3 = 0.;
    }
    else if(s3 < min_ratio * p_atm) {
        elastic = ref_elastic * std::pow(min_ratio, n);
        deds3 = 0.;
    }
    else {
        elastic = ref_elastic * std::pow(s3 / p_atm, n);
        deds3 = n * elastic / s3;
    }

    return {elastic, deds3};
}

std::tuple<double, double> DuncanSelig::compute_bulk(const double s3) const {
    double bulk, dkds3;
    if(s3 < -min_ratio * p_atm) {
        bulk = ref_bulk * std::pow(.01, m);
        dkds3 = 0.;
    }
    else if(s3 < min_ratio * p_atm) {
        bulk = ref_bulk * std::pow(min_ratio, m);
        dkds3 = 0.;
    }
    else {
```

```
        bulk = ref_bulk * std::pow(s3 / p_atm, m);
        dkds3 = m * bulk / s3;
    }

    return {bulk, dkds3};
}

std::tuple<double, double, rowvec3, rowvec3> DuncanSelig::compute_elastic_moduli() {
    // principal stresses

    const auto radius = .5 * dev(trial_stress);
    rowvec3 drds(fill::zeros);
    if(radius > datum::eps) drds = der_dev(trial_stress) / radius * .25;

    const auto center = -.5 * (trial_stress(0) + trial_stress(1));
    const rowvec3 dcds{-.5, -.5, 0.};

    const auto s3 = center - radius;

    const rowvec3 ds3ds = dcds - drds;

    // for elastic modulus

    const auto [elastic, deds3] = compute_elastic(s3);

    const rowvec3 deds = deds3 * ds3ds;

    // for bulk modulus

    auto [bulk, dkds3] = compute_bulk(s3);

    rowvec3 dkds = dkds3 * ds3ds;

    if(3. * bulk < elastic) {
        bulk = elastic / 3.;
        dkds = deds / 3.;
    }

    return {elastic, bulk, deds, dkds};
}

std::tuple<double, double, rowvec3, rowvec3> DuncanSelig::compute_plastic_moduli() {
    // principal stresses

    const auto center = -.5 * (trial_stress(0) + trial_stress(1));
    const rowvec3 dcds{-.5, -.5, 0.};

    auto radius = .5 * dev(trial_stress);
    rowvec3 drds(fill::zeros);
    if(radius > datum::eps) drds = der_dev(trial_stress) / radius * .25;

    const auto s1 = center + radius, s3 = center - radius;

    const rowvec3 ds1ds = dcds + drds, ds3ds = dcds - drds;

    // for elastic modulus

```

```
    double phi, dphids3;
    if(s3 < p_atm) {
        phi = ini_phi;
        dphids3 = 0.;
    }
    else {
        phi = ini_phi - ten_fold_phi_diff * log10(s3 / p_atm);
        dphids3 = -ten_fold_phi_diff / (s3 * log(10));
        if(phi < 0.) phi = dphids3 = 0.;
    }

    const auto denom = 1. - std::sin(phi);
    auto max_dev_stress = 2. / r_f * (cohesion * std::cos(phi) + s3 * std::sin(phi)) / denom;
    auto dmdsds3 = 0.;
    if(max_dev_stress > min_ratio * p_atm) {
        const auto pmdspphi = 2. / r_f * (s3 * std::cos(phi) / denom / denom + cohesion /
        ↪  denom);
        const auto pmdsps3 = 2. / r_f * std::sin(phi) / denom;
        dmdsds3 = pmdspphi * dphids3 + pmdsps3;
    }
    else max_dev_stress = min_ratio * p_atm;

    const auto dev_stress = s1 - s3;
    const auto pdsps1 = 1.;
    const auto pdsps3 = -1.;

    const auto [ini_elastic, deids3] = compute_elastic(s3);

    const auto pepei = std::pow(1. - dev_stress / max_dev_stress, 2.);
    const auto elastic = ini_elastic * pepei;
    const auto pepds = -2. * ini_elastic * (1. - dev_stress / max_dev_stress) /
    ↪  max_dev_stress;
    const auto pepmds = 2. * ini_elastic * (1. - dev_stress / max_dev_stress) * dev_stress /
    ↪  max_dev_stress / max_dev_stress;

    const auto peps1 = pepds * pdsps1;
    const auto peps3 = pepei * deids3 + pepds * pdsps3 + pepmds * dmdsds3;

    const rowvec3 deds = peps1 * ds1ds + peps3 * ds3ds;

    // for bulk modulus

    auto [bulk, dkds3] = compute_bulk(s3);

    rowvec3 dkds = dkds3 * ds3ds;

    if(3. * bulk < elastic) {
        bulk = elastic / 3.;
        dkds = deds / 3.;
    }

    return {elastic, bulk, deds, dkds};
}
```

12. Viscoplasticity

12.1. VAFCRP Model

Here we present a versatile viscoplastic model that can be used to model metals under dynamic loadings. It is essentially a multiaxial extension of the aforementioned Armstrong–Fredrick uniaxial model with additional viscosity component. Due to the presence of viscosity, the typical formulation and implementation shall be modified a bit, as now transient effects are accounted for and yield function does not need to be non-positive. It is only used to determine whether a trial state is elastic or plastic.

12.1.1. Theory

Yield Function

A von Mises type yielding function is used.

$$f = \sqrt{\frac{3}{2}}\|\boldsymbol{\eta}\| - k = q - k, \tag{12.1}$$

in which $\boldsymbol{\eta} = \boldsymbol{s} - \boldsymbol{\beta}$ is the shifted stress, $\boldsymbol{s}$ is the deviatoric stress, $\boldsymbol{\beta}$ is the back stress, k is the isotropic hardening stress and $q = \sqrt{\frac{3}{2}}\|\boldsymbol{\eta}\|$ is the equivalent stress.

Flow Rule

The associated plasticity flow is adopted. The plastic strain rate is then

$$\dot{\boldsymbol{\varepsilon}}^p = \gamma \frac{\partial f}{\partial \boldsymbol{\sigma}} = \sqrt{\frac{3}{2}}\gamma \boldsymbol{n}, \tag{12.2}$$

where $\boldsymbol{n} = \dfrac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|}$. The corresponding accumulated plastic strain rate is

$$\dot{p} = \sqrt{\frac{2}{3}\dot{\boldsymbol{\varepsilon}}^p : \dot{\boldsymbol{\varepsilon}}^p} = \gamma. \tag{12.3}$$

One shall note that strictly speaking, it shall be the viscous plastic strain rather than the plastic strain. For brevity, the superscript $(\cdot)^p$ is used. If one wishes, $(\cdot)^{vp}$ can also be used.

Plastic Multiplier

Unlike plasticity, viscoplasticity adopts a different way to determine plastic multiplier. The rate of plastic multiplier is defined as

$$\frac{\gamma}{\Delta t} = \frac{1}{\mu}\left(\left(\frac{q}{k}\right)^{\frac{1}{\epsilon}} - 1\right), \tag{12.4}$$

in which Δt is increment of pseudo time which shall be available from global time integration method, μ and ϵ are two material constants. Here a [17] type rule is used for easier numerical implementation. Equivalently, after some rearrangement, it is

$$q\left(\frac{\Delta t}{\Delta t + \mu\gamma}\right)^{\epsilon} - k = 0. \tag{12.5}$$

Hardening Law

Similar to the previous model, a Voce type exponential function with a linear component is used for isotropic hardening stress.

$$k = \sigma_y + k_l p + k_s - k_s e^{-mp}. \tag{12.6}$$

In which, m is a parameter controls the speed of hardening.

The incremental form of multiplicative back stress [6] $\boldsymbol{\beta} = \sum \boldsymbol{\beta}^i$ is defined as

$$\dot{\boldsymbol{\beta}}^i = \sqrt{\frac{2}{3}} a^i \dot{\boldsymbol{\varepsilon}}^p - b^i \boldsymbol{\beta}^i \dot{p}.$$

Again, a^i and b^i are two sets of model parameters. In terms of γ, it is $\dot{\boldsymbol{\beta}}^i = a^i \gamma \boldsymbol{n} - b^i \gamma \boldsymbol{\beta}^i$.

12.1.2. Formulation

In the following derivation, the subscript $(\cdot)_{n+1}$ is omitted for brevity.

Elastic Loading/Unloading

The trial state can be computed as usual.

$$\boldsymbol{\sigma}^{\text{trial}} = \boldsymbol{\sigma}_n + \boldsymbol{D} : (\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n)\,. \tag{12.7}$$

The trial yield function can be then expressed as

$$f^{\text{trial}} = \sqrt{\frac{3}{2}}\left\|\text{dev}\left(\boldsymbol{\sigma}^{\text{trial}}\right) - \sum \boldsymbol{\beta}_n^i\right\| - k_n. \tag{12.8}$$

Plastic Evolution

Given that

$$\boldsymbol{\beta}^i = \boldsymbol{\beta}_n^i + a^i\gamma\boldsymbol{n} - b^i\gamma\boldsymbol{\beta}^i, \quad \rightarrow \quad \boldsymbol{\beta}^i = \frac{\boldsymbol{\beta}_n^i + a^i\gamma\boldsymbol{n}}{1 + b^i\gamma}, \tag{12.9}$$

the shifted stress can be computed as

$$\begin{aligned}\boldsymbol{\eta} = \boldsymbol{s} - \boldsymbol{\beta} &= 2G\mathbb{I}^{\text{dev}} : \left(\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p - \sqrt{\frac{3}{2}}\gamma\boldsymbol{n}\right) - \boldsymbol{\beta} \\ &= \boldsymbol{s}^{\text{trial}} - \sqrt{6}G\gamma\boldsymbol{n} - \sum \frac{\boldsymbol{\beta}_n^i + a^i\gamma\boldsymbol{n}}{1 + b^i\gamma}\end{aligned} \tag{12.10}$$

with $\boldsymbol{s}^{\text{trial}} = 2G\mathbb{I}^{\text{dev}}\left(\boldsymbol{\varepsilon}_{n+1} - \boldsymbol{\varepsilon}_n^p\right)$. Hence,

$$\|\boldsymbol{\eta}\|\boldsymbol{n} + \sqrt{6}G\gamma\boldsymbol{n} + \sum \frac{a^i\gamma}{1 + b^i\gamma}\boldsymbol{n} = \boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1 + b^i\gamma},$$

it is easy to see that

$$\left(\|\boldsymbol{\eta}\| + \sqrt{6}G\gamma + \sum \frac{a^i\gamma}{1 + b^i\gamma}\right)\boldsymbol{n} = \boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1 + b^i\gamma}, \tag{12.11}$$

meaning $\boldsymbol{n}$ and $\boldsymbol{s}^{\text{trial}} - \sum \dfrac{\boldsymbol{\beta}_n^i}{1 + b^i\gamma}$ are coaxial, thus

$$\boldsymbol{n} = \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} = \frac{\boldsymbol{s}^{\text{trial}} - \sum \dfrac{\boldsymbol{\beta}_n^i}{1 + b^i\gamma}}{\left\|\boldsymbol{s}^{\text{trial}} - \sum \dfrac{\boldsymbol{\beta}_n^i}{1 + b^i\gamma}\right\|}. \tag{12.12}$$

Now both $\boldsymbol{\eta}$ and $\boldsymbol{n}$ shall be functions of γ, which allows simplification of local system. Based on this fact,

$$\boldsymbol{\eta} = \|\boldsymbol{\eta}\|\boldsymbol{n} = \left(\left\| \boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1 + b^i \gamma} \right\| - \sqrt{6} G \gamma - \sum \frac{a^i \gamma}{1 + b^i \gamma} \right) \boldsymbol{n}. \tag{12.13}$$

Furthermore, $q = \sqrt{\frac{3}{2}} \left(\left\| \boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1 + b^i \gamma} \right\| - \sqrt{6} G \gamma - \sum \frac{a^i \gamma}{1 + b^i \gamma} \right)$. Its derivative with regard to γ is

$$\frac{\partial q}{\partial \gamma} = \sqrt{\frac{3}{2}} \sum \frac{b^i \boldsymbol{n} : \boldsymbol{\beta}_n^i - a^i}{(1 + b^i \gamma)^2} - 3G. \tag{12.14}$$

Local Scalar Residual

For viscoplasticity, the yield function f is not necessarily zero. The rate form of plastic multiplier is used here as the residual.

$$R = q \left(\frac{\Delta t}{\Delta t + \mu \gamma} \right)^{\epsilon} - k.$$

The corresponding derivatives are then

$$\frac{\partial R}{\partial \gamma} = \left(\frac{\Delta t}{\Delta t + \mu \gamma} \right)^{\epsilon} \left(\frac{\partial q}{\partial \gamma} - \frac{q \epsilon \mu}{\Delta t + \mu \gamma} \right) - \frac{\mathrm{d}k}{\mathrm{d}\gamma} \tag{12.15}$$

and

$$\frac{\partial R}{\partial \boldsymbol{\varepsilon}_{n+1}} = \left(\frac{\Delta t}{\Delta t + \mu \gamma} \right)^{\epsilon} \sqrt{6} G \boldsymbol{n} : \mathbb{I}^{\text{dev}} = \left(\frac{\Delta t}{\Delta t + \mu \gamma} \right)^{\epsilon} \sqrt{6} G \boldsymbol{n}, \tag{12.16}$$

with

$$\frac{\mathrm{d}k}{\mathrm{d}\gamma} = k_l + k_s m e^{-m(p_n + \gamma)}. \tag{12.17}$$

Consistent Tangent Stiffness

For stiffness, $\boldsymbol{\varepsilon}_{n+1}$ is now varying, then

$$\frac{\partial R}{\partial \boldsymbol{\varepsilon}_{n+1}} + \frac{\partial R}{\partial \gamma} \frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} = 0, \qquad \frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}} = - \left(\frac{\partial R}{\partial \gamma} \right)^{-1} \frac{\partial R}{\partial \boldsymbol{\varepsilon}_{n+1}}. \tag{12.18}$$

The stress is

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}} - \sqrt{6} G \gamma \boldsymbol{n}. \tag{12.19}$$

The derivative is

$$\begin{aligned}\frac{\partial \boldsymbol{\sigma}_{n+1}}{\partial \boldsymbol{\varepsilon}_{n+1}} &= \boldsymbol{D} - \sqrt{6}G\left(\gamma\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\varepsilon}_{n+1}} + \left(\boldsymbol{n} + \gamma\frac{\partial \boldsymbol{n}}{\partial \gamma}\right)\frac{\partial \gamma}{\partial \boldsymbol{\varepsilon}_{n+1}}\right) \\ &= \boldsymbol{D} + \sqrt{6}G\left(\left(\boldsymbol{n} + \gamma\frac{\partial \boldsymbol{n}}{\partial \gamma}\right)\left(\frac{\partial R}{\partial \gamma}\right)^{-1}\frac{\partial R}{\partial \boldsymbol{\varepsilon}_{n+1}} - \gamma\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\varepsilon}_{n+1}}\right).\end{aligned} \tag{12.20}$$

In which,

$$\frac{\partial \boldsymbol{n}}{\partial \gamma} = \frac{\sum \frac{b^i}{(1+b^i\gamma)^2}\left(\boldsymbol{\beta}_n^i - (\boldsymbol{n}:\boldsymbol{\beta}_n^i)\,\boldsymbol{n}\right)}{\left\|\boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1+b^i\gamma}\right\|}, \qquad \frac{\partial \boldsymbol{n}}{\partial \boldsymbol{\varepsilon}_{n+1}} = \frac{2G\left(\mathbb{I}^{\text{dev}} - \boldsymbol{n}\otimes\boldsymbol{n}\right)}{\left\|\boldsymbol{s}^{\text{trial}} - \sum \frac{\boldsymbol{\beta}_n^i}{1+b^i\gamma}\right\|}. \tag{12.21}$$

12.1.3. Implementation

The state determination algorithm of this VAFCRP model is given in Algorithm 14.

Algorithm 14 state determination of VAFCRP model

Parameter: λ, G, u, ϵ, m, k_l, k_s, σ_y, a^i, b^i
Input: $\boldsymbol{\varepsilon}_{n+1}$, $\boldsymbol{\varepsilon}_n$, $\boldsymbol{\sigma}_n$, $\boldsymbol{\beta}_n^i$, p_n, Δt
Output: $\boldsymbol{D}_{n+1}$, $\boldsymbol{\sigma}_{n+1}$, $\boldsymbol{\beta}_{n+1}^i$, p_{n+1}
compute $\boldsymbol{\sigma}^{\text{trial}}$, $\boldsymbol{s}^{\text{trial}}$, $\boldsymbol{n}$ and f^{trial} ▷ Eq. (12.7) and Eq. (12.8)
if $f^{\text{trial}} \geqslant 0$ **then**
  $\gamma = 0$
  compute $\boldsymbol{s}^{\text{trial}}$
  while true **do**
    compute $\boldsymbol{n}$
    compute R and $\frac{\partial R}{\partial \gamma}$ ▷ Eq. (12.1.2) and Eq. (12.15)
    $\Delta\gamma = \left(\frac{\partial R}{\partial \gamma}\right)^{-1} R$
    if $|\Delta\gamma| <$ tolerance **then**
      break
    end if
    $\gamma \leftarrow \gamma - \Delta\gamma$
    $p_{n+1} = p_n + \gamma$
  end while
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}} - \sqrt{6}G\gamma\boldsymbol{n}$
  compute all $\boldsymbol{\beta}_{n+1}^i$ ▷ Eq. (12.9)
  compute $\boldsymbol{D}_{n+1}$ ▷ Eq. (12.20)
else
  $\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^{\text{trial}}$
  $\boldsymbol{\beta}_{n+1}^p = \boldsymbol{\beta}_n^p$
  $p_{n+1} = p_n$
  $\boldsymbol{D}_{n+1} = \boldsymbol{D}$

end if

```cpp
int VAFCRP::update_trial_status(const vec& t_strain) {
    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * (incre_strain = (trial_strain = t_strain) - current_strain);

    trial_history = current_history;
    auto& p = trial_history(0);

    const auto trial_s = tensor::dev(trial_stress);

    auto eta = trial_s;
    for(unsigned I = 0; I < size; ++I) eta -= vec{&trial_history(1 + 6llu * I), 6, false, true};

    // const auto residual = root_three_two * tensor::stress::norm(eta) - std::max(0., yield + hardening * p + saturated * (1. - exp(-m * p)));

    if(root_three_two * tensor::stress::norm(eta) < std::max(0., yield + hardening * p + saturated * (1. - exp(-m * p)))) return SUANPAN_SUCCESS;

    vec xi;
    auto gamma = 0., exp_gamma = 1.;
    double norm_xi, jacobian;

    unsigned counter = 0;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("VAFCRP cannot converge within %u iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        const auto exp_term = saturated * exp(-m * p);

        auto k = yield + saturated + hardening * p - exp_term;
        auto dk = hardening + m * exp_term;
        if(k < 0.) k = dk = 0.;

        vec sum_a(6, fill::zeros);
        auto sum_b = 0.;
        for(unsigned I = 0; I < size; ++I) {
            const auto denom = 1. + b(I) * gamma;
            sum_a += vec{&trial_history(1 + 6llu * I), 6, false, true} / denom;
            sum_b += a(I) * gamma / denom;
        }

        norm_xi = tensor::stress::norm(xi = trial_s - sum_a);

        const auto q = root_three_two * (norm_xi - root_six_shear * gamma - sum_b);
        exp_gamma = pow(*incre_time / (*incre_time + mu * gamma), epsilon);

        sum_b = 0.;
        for(unsigned I = 0; I < size; ++I) sum_b += (b(I) / norm_xi * tensor::stress::double_contraction(xi, vec{&trial_history(1 + 6llu * I), 6, false, true}) - a(I)) * pow(1. + b(I) * gamma, -2.);

        jacobian = exp_gamma * (root_three_two * sum_b - three_shear - q * epsilon * mu / (*incre_time + mu * gamma)) - dk;
```

```
50
51          const auto incre = (q * exp_gamma - k) / jacobian;
52          suanpan_extra_debug("VAFCRP local iterative loop error: %.5E.\n", fabs(incre));
53          if(fabs(incre) <= tolerance) break;
54
55          gamma -= incre;
56          p -= incre;
57      }
58
59      const vec u = xi / norm_xi;
60
61      vec sum_c(6, fill::zeros);
62      for(unsigned I = 0; I < size; ++I) {
63          vec beta(&trial_history(1 + 6llu * I), 6, false, true);
64          sum_c += b(I) * pow(1. + b(I) * gamma, -2.) * (beta -
            ↪  tensor::stress::double_contraction(u, beta) * u);
65          beta = (beta + a(I) * gamma * u) / (1. + b(I) * gamma);
66      }
67
68      trial_stress -= root_six_shear * gamma * u;
69
70      trial_stiffness += (root_six_shear * (double_shear * gamma / norm_xi + root_six_shear *
        ↪  exp_gamma / jacobian) * u + root_six_shear * root_six_shear * exp_gamma * gamma /
        ↪  jacobian / norm_xi * sum_c) * u.t() - double_shear * root_six_shear * gamma / norm_xi
        ↪  * unit_dev_tensor;
71
72      return SUANPAN_SUCCESS;
73  }
```

12.2. Maxwell Model

The above viscoplasticity models often requires the explicit reference to time t (and its increment Δt). By knowing strain increment $\Delta\boldsymbol{\varepsilon}$ and Δt, strain rate can be computed by using, for example, constant rate assumption. By such, the whole model can be converted to a form that only depends on strain input, assuming (pseudo-)time is always accessible to local material points. Even with a static analysis setup, creep can be modelled.

Here we present a general framework [18] to solve the viscoplasticity model that truly responsive to both strain and strain rate inputs.

12.2.1. Background

For viscous dampers, the mechanical response is often defined in displacement/force. Without loss of generality, a typical viscosity model often defines a linear relationship between stress

(force/resistance) σ and strain rate (velocity) $\dot{\varepsilon}$,

$$\sigma = \eta \cdot \dot{\varepsilon}, \tag{12.22}$$

where η is a non-zero constant that is known as the viscosity. Such a linear relationship is known as Newtonian viscosity. For non-Newtonian behaviour, often the power-law fluid is assumed for simplicity [19], that is,

$$\sigma = \eta \cdot \text{sign}\left(\dot{\varepsilon}\right) \cdot \left|\dot{\varepsilon}\right|^{\alpha}, \tag{12.23}$$

where α is a positive constant often known as the flow behaviour index which shall be determined by experiments. If $\alpha = 1.0$, the Newtonian viscosity is recovered. A value greater than 1.0 represents shear thickening behaviour. For structural dampers, often shear thinning fluid is used, typical values of α range from 0.3 to 1.0 [20]. For recent applications, this exponent can be as small as 0.1 [21].

12.2.2. Modified Power-law Viscosity

Viscous dampers can show different behaviour in four quadrants of strain versus strain rate space. Such a change of behaviour may stem from, for example, applying one-way valves, varying chamber geometries [22], changing material types [23] and/or effective fluid velocity. A constant damping coefficient (viscosity) is not capable of describing such a behaviour (of the device). Instead, it should be defined as a positive function of current state to account for various mechanisms, which is

$$\eta = f\left(\varepsilon, \dot{\varepsilon}\right) > 0. \tag{12.24}$$

Thus the stress can be written as

$$\sigma = \eta\left(\varepsilon, \dot{\varepsilon}\right) \cdot \text{sign}\left(\dot{\varepsilon}\right) \cdot \left|\dot{\varepsilon}\right|^{\alpha}. \tag{12.25}$$

It shall be noted that Eq. (12.25) can also be expressed as

$$\sigma = \tilde{\eta} \cdot \dot{\varepsilon} \tag{12.26}$$

with $\tilde{\eta} = \eta\left(\varepsilon, \dot{\varepsilon}\right) \cdot \left|\dot{\varepsilon}\right|^{\alpha-1}$ known as *apparent viscosity*, so that such a modification can still be categorised as the generalised Newtonian fluid model. The definition of $\eta\left(\varepsilon, \dot{\varepsilon}\right)$ can be quite flexible in order to describe the desired response. Such a modification mimics semi-active control schemes in which stress feedback can be adjusted based on different strain and strain rates as inputs. For more complex (semi-)active schemes, it can be further defined as a function of other quantities, such as system energy and its history. Two examples are shown as follows to illustrate this feature.

Proposed Quadrant Modification

The simplest case would be using four different constants for four quadrants of the ε-$\dot{\varepsilon}$ space.

$$\eta\left(\varepsilon,\dot{\varepsilon}\right)=\begin{cases}\eta_1, & \varepsilon>0,\ \dot{\varepsilon}>0,\\ \eta_2, & \varepsilon<0,\ \dot{\varepsilon}>0,\\ \eta_3, & \varepsilon<0,\ \dot{\varepsilon}<0,\\ \eta_4, & \varepsilon>0,\ \dot{\varepsilon}<0.\end{cases} \tag{12.27}$$

Although Eq. (12.27) possesses a simple form that could be easily understood, sudden changes of damping coefficient, as observed on two axes, are practically unrealistic. The rate of transition from one quadrant to another also plays a vital role and affects the overall response. Furthermore, discontinuities in damping coefficient may cause numerical difficulties. Ideally, a smooth transition is required to improve both numerical stability and robustness of the model. To this end, a sigmoid function can be applied. The following arctangent functions provide controllable smooth transition between two sides of the $\dot{\varepsilon}$-axis.

$$\eta_{12}\left(\varepsilon\right)=\frac{\eta_1+\eta_2}{2}+\frac{\eta_1-\eta_2}{\pi}\arctan\left(g_1\varepsilon\right), \tag{12.28}$$

$$\eta_{43}\left(\varepsilon\right)=\frac{\eta_4+\eta_3}{2}+\frac{\eta_4-\eta_3}{\pi}\arctan\left(g_1\varepsilon\right), \tag{12.29}$$

where g_1 is a constant that controls the steepness of the transition region. In a similar fashion, for the ε-axis, the following function can be defined,

$$\eta\left(\varepsilon,\dot{\varepsilon}\right)=\frac{\eta_{12}\left(\varepsilon\right)+\eta_{43}\left(\varepsilon\right)}{2}+\frac{\eta_{12}\left(\varepsilon\right)-\eta_{43}\left(\varepsilon\right)}{\pi}\arctan\left(g_2\dot{\varepsilon}\right), \tag{12.30}$$

where g_2 is another constant that serves a similar purpose to that of g_1. The damping coefficient η can now be expressed as a function of four material constants, viz.,

$$\begin{aligned}\eta\left(\varepsilon,\dot{\varepsilon}\right)=&\frac{\eta_1+\eta_2+\eta_3+\eta_4}{4}+\frac{\eta_1-\eta_2+\eta_3-\eta_4}{\pi^2}\arctan\left(g_1\varepsilon\right)\arctan\left(g_2\dot{\varepsilon}\right)\\&+\frac{\eta_1-\eta_2-\eta_3+\eta_4}{2\pi}\arctan\left(g_1\varepsilon\right)+\frac{\eta_1+\eta_2-\eta_3-\eta_4}{2\pi}\arctan\left(g_2\dot{\varepsilon}\right).\end{aligned} \tag{12.31}$$

Noting that a sudden change of damping response is not achievable in real life, constants g_1 and g_2 (strictly speaking, only the one corresponds to displacement tolerance, viz., g_1) can hence represent physical manufacturing tolerance of pistons and chambers.

Apart from the arctangent function, other types of sigmoid curves can also be used. The following is an alternative using the logistic function.

$$\eta\left(\varepsilon,\dot{\varepsilon}\right)=\eta_3+\frac{\eta_4-\eta_3}{1+e^{-g_1\varepsilon}}+\frac{\eta_2-\eta_3}{1+e^{-g_2\dot{\varepsilon}}}+\frac{\eta_1+\eta_3-\eta_2-\eta_4}{\left(1+e^{-g_1\varepsilon}\right)\left(1+e^{-g_2\dot{\varepsilon}}\right)}. \tag{12.32}$$

It shall be noted that the derivatives of Eq. (12.31) have a simpler form than that of the above definition. Other simple functions such as a linear function can also be applied, in which the

transition range can be explicitly defined.

The quadrant modification can be customised to mimic the effect of negative-stiffness damping [24, 25, 26, 27] by choosing large η_2 and η_4 and small η_1 and η_3. Further elaborations are shown in numerical examples. Readers who are interested in practical applications of such a quadrant damper can refer to, for example, the work by [22].

12.2.3. Extension to Maxwell Model

In some certain applications, dampers could be idealised as Maxwell models given the fact that the extender braces are not fully rigid [28]. For classic viscoelasticity and viscoplasticity, theories have been developed [3]. Simple cases can be solved analytically by using convolution integrals. Consider a typical Maxwell model, the rheology model is often represented by Fig. 12.1. It is normally represented by two components in series: a viscous dashpot and a

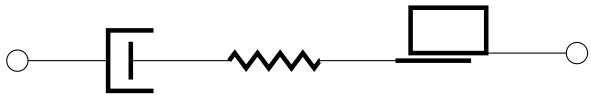


Figure 12.1.: rheology model of the Maxwell model with inelastic spring

rate-independent spring which can be either elastic (without frictional device) or elasto-plastic (with frictional device), then the total strain ε and stress σ of the model can be expressed as

$$\varepsilon = \varepsilon_d + \varepsilon_s, \tag{12.33}$$

$$\sigma = \sigma_d = \sigma_s. \tag{12.34}$$

For a generalised case, it is also possible to further write stress feedback as functions of the corresponding strain and strain rate, which is

$$\sigma_d = f(\varepsilon_d, \dot{\varepsilon}_d), \qquad \sigma_s = g(\varepsilon_s). \tag{12.35}$$

The subscripts $\cdot_d$ and $\cdot_s$ represent dashpot and spring part, respectively. By differentiating total strain with respect to time, one could obtain

$$\dot{\varepsilon} = \dot{\varepsilon}_d + \dot{\varepsilon}_s. \tag{12.36}$$

The governing equation can be established via Eq. (12.34) so that

$$\sigma_d - \sigma_s = 0. \tag{12.37}$$

12.2.4. Formulation

In terms of numerical simulation, using a dashpot alone does not require special treatments since the corresponding damping force can be treated as external load and directly applied to the system/structure/model. If needed, the damping modulus can be derived accordingly. This also applies to the case if the damper is idealised as a Kelvin–Voigt model. However, for a Maxwell model, due to the presence of coupling between dashpot and spring, a proper algorithm is required for state determinations of both components. Some researchers use the popular Bouc-Wen [29] model to simulate viscous dampers [30, 31]. However, the identification and calibration of model parameters often impose unnecessary complexities to the model. Alternatively, the Maxwell system can be solved by using ODE solvers. In this section, the drawbacks of the ODE solver based approach are first discussed, followed by the proposition of a new iterative algorithm with better accuracy and efficiency.

ODE Solver Based Approach

If a linear elastic spring and a constant η are adopted, the whole system can be converted into an ordinary differential equation via Eq. (12.36), that is,

$$\dot{\varepsilon} = \dot{\varepsilon}_d + \dot{\varepsilon}_s = \operatorname{sign}(\sigma) \sqrt[\alpha]{\frac{|\sigma|}{\eta}} + \frac{\dot{\sigma}}{E}, \tag{12.38}$$

where E denotes the elastic modulus of the spring element, so that

$$\dot{\sigma} = E\left(\dot{\varepsilon} - \operatorname{sign}(\sigma) \sqrt[\alpha]{\frac{|\sigma|}{\eta}}\right). \tag{12.39}$$

By assuming a proper distribution of total strain rate $\dot{\varepsilon}$ over time t, viz., $\dot{\varepsilon} = \dot{\varepsilon}(t)$, Eq. (12.39) can be written as a function of σ and t, viz., $\dot{\sigma} = f(\sigma, t)$. Sophisticated solvers for ordinary differential equations, including explicit, implicit and semi-implicit solvers, can be applied to solve this system. Such an approach has been used in prior research [32, 33]. For some simple cases, analytical solutions can also be derived [34].

Although the above method is simple, straightforward and easy to implement, it suffers from three main drawbacks.

1. Applicability. Such an approach cannot be applied to inelastic spring. For which, unless the corresponding spring constitutive model is **stress driven**, which is not the case for most constitutive models, spring strain (rate) cannot be recovered from total stress (rate) since it depends on loading history. As a consequence, dashpot strain (rate) cannot be recovered. Because of this reason, the damping coefficient η can only be a function of total strain and total strain rate (instead of that of dashpot), however, dashpot response solely depends on its own strain and strain rate according to the definition. This is **theoretically incorrect**. As can be seen later, the discrepancy due to different strain and strain rate measures adopted can sometimes be significant, depending on the specific

parameter set used.

Even with elastic spring, either linear or nonlinear, since spring strain is not explicitly included in Eq. (12.39), there is no consistent way to isolate spring/dashpot strain (rate) from total strain (rate) without interpolating both total strain and total strain rate. This then leads to the second issue.

2. Kinematics Compatibility. This problem stems from the assumed distribution of total strain rate $\dot{\varepsilon}$, which should be carefully defined by taking the global level time integration method into account. On one hand, the global time integration scheme is often not known to local material points. Analysts may switch from one global time integration method to another, resulting in different integrations of total strain at local points. On the other hand, an arbitrarily defined distribution of $\dot{\varepsilon}$, such as a simple linear relationship [33], would lead to a corresponding strain increment which is independent from the one computed by the global time integration. A consistency/compatibility issue arises. This is less concerning if the time step is sufficiently small.

3. Efficiency. Existing ODE solvers are less efficient in terms of solving such a Maxwell model. Consider the classic Fehlberg method [35] as an example, by construction, it requires six evaluations of Eq. (12.39) for every new σ. If the error of the current step is unacceptable, all previously evaluated function values would be discarded and a smaller step size needs to be chosen to repeat the whole computation procedure. Sub-iterations may be further required to meet a small tolerance.

   Besides, if a shear thinning power-law fluid is used, when α deviates from unity ($\alpha < 1$), Eq. (12.39) tends to be stiff when velocity (strain rate) is close to zero and results in potential numerical instability. In that case, explicit methods fail and lower order implicit solvers have to be used, and the computational cost skyrockets due to their low order of convergence (second order at most due to the second Dahlquist barrier [36]). Similar stability issues may also occur if spring stiffness is disproportionally too large.

   Furthermore, since the only independent variable is time t, it is in general difficult to derive the tangent moduli for global equation solving, resulting in a superlinear global convergence rate at most.

More accurate results can be achieved with fewer function evaluations and thus less computation time if a better solving method is available. The ideal algorithm shall strictly comply with the theory and possess a higher convergence rate. Moreover, there is still room for improvements of both applicability and efficiency/robustness.

Proposed Iterative Approach

Here a typical strain driven framework is assumed. For numerical simulation, at a single material point, it is generally difficult to obtain the exact relationship between total strain ε and total strain rate $\dot{\varepsilon}$ as they are determined by the global integration algorithm that could

be changed on demand. To ensure kinematic compatibility, similar to the strategy adopted in many time integration methods, the following assumption can be made between two adjacent steps t^n and $t^{n+1} = t^n + \Delta t$ with $\chi = \varepsilon$ denoting the total strain of the Maxwell model and $\dot{\chi} = \dot{\varepsilon}$ denoting its strain rate,

$$\chi^{n+1} = \chi^n + \left((1-\beta)\,\dot{\chi}^n + \beta\dot{\chi}^{n+1}\right)\Delta t, \tag{12.40}$$

or equivalently with $\Delta\chi = \chi^{n+1} - \chi^n$ and $\Delta\dot{\chi} = \dot{\chi}^{n+1} - \dot{\chi}^n$ denoting increments of χ and its rate,

$$\Delta\chi = (\dot{\chi}^n + \beta\Delta\dot{\chi})\,\Delta t, \tag{12.41}$$

where β is an integration parameter. For $\beta = 0.5$, a constant acceleration rule is implied. Since there is no other constraint imposed, such an integration relationship can be alternatively applied to either spring component ($\chi = \varepsilon_s$) or dashpot component ($\chi = \varepsilon_d$). By such a manner, kinematic compatibility can be rigorously satisfied within the Maxwell model and is independent from the global time integration method. The parameter β can be defined as a user input (if $\chi = \varepsilon_s$ or $\chi = \varepsilon_d$) or be solved internally (if $\chi = \varepsilon$), which is

$$\beta = \frac{\Delta\varepsilon - \dot{\varepsilon}^n\Delta t}{\Delta\dot{\varepsilon}\Delta t}, \tag{12.42}$$

by using total strains ε^{n+1} and ε^n and total strain rates $\dot{\varepsilon}^n$ and $\dot{\varepsilon}^{n+1}$, since they are given at the beginning of each time step at a specific material point.

As aforementioned, the damping coefficient can be defined as a function of dashpot strain ε_d and dashpot strain rate $\dot{\varepsilon}_d$, it is necessary to compute them explicitly as history variables. Here an iterative method is presented to solve for all strain and strain rate components.

With the above basic formulae at hand, the problem can now be rephrased as: knowing ε^n, $\dot{\varepsilon}^n$, ε_d^n, $\dot{\varepsilon}_d^n$, ε_s^n and $\dot{\varepsilon}_s^n$, given the total increments $\Delta\varepsilon = \varepsilon^{n+1} - \varepsilon^n$ and $\Delta\dot{\varepsilon} = \dot{\varepsilon}^{n+1} - \dot{\varepsilon}^n$, find increments $\Delta\varepsilon_d$, $\Delta\varepsilon_s$, $\Delta\dot{\varepsilon}_d$ and $\Delta\dot{\varepsilon}_s$ that satisfy

$$\Delta\varepsilon_d + \Delta\varepsilon_s = \Delta\varepsilon, \tag{12.43}$$
$$\Delta\dot{\varepsilon}_d + \Delta\dot{\varepsilon}_s = \Delta\dot{\varepsilon}, \tag{12.44}$$
$$\Delta\varepsilon_s - \beta\Delta\dot{\varepsilon}_s\Delta t = \dot{\varepsilon}_s^n\Delta t, \tag{12.45}$$
$$\sigma_d(\varepsilon_d^{n+1}, \dot{\varepsilon}_d^{n+1}) - \sigma_s(\varepsilon_s^{n+1}) = 0. \tag{12.46}$$

It shall be pointed out that Eq. (12.45) is the implementation of Eq. (12.41) on spring strain ε_s. In this case, the parameter β can be either computed from Eq. (12.42) or given as a model constant. Eq. (12.41) can also be applied on dashpot strain ε_d. However, additional conversions between different quantities may be required.

Since $\dot{\varepsilon}_s$ does not enter the constitutive relationship of spring, it can be condensed out so that

Eq. (12.44) and Eq. (12.45) can be combined to

$$\Delta\varepsilon_s + \beta\Delta\dot{\varepsilon}_d\Delta t = \dot{\varepsilon}_s^n\Delta t + \beta\Delta\dot{\varepsilon}\Delta t. \tag{12.47}$$

Eq. (12.43), Eq. (12.46) and Eq. (12.47) can be iteratively solved with the classic Newton-Raphson method. Alternatively, other optimisers can be applied. Linearisation of these equations leads to the Jacobian matrix $\boldsymbol{J}$ and the corresponding residual $\boldsymbol{R}$ for increment $\mathbf{x} = \begin{bmatrix}\delta\varepsilon_s & \delta\varepsilon_d & \delta\dot{\varepsilon}_d\end{bmatrix}^{\mathrm{T}}$, which are

$$\boldsymbol{J} = -\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & \beta\Delta t \\ -\dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} & \dfrac{\partial\sigma_d}{\partial\varepsilon_d} & \dfrac{\partial\sigma_d}{\partial\dot{\varepsilon}_d} \end{bmatrix}, \qquad \boldsymbol{R} = \begin{bmatrix} \Delta\varepsilon - \Delta\varepsilon_s^k - \Delta\varepsilon_d^k \\ \dot{\varepsilon}_s^n\Delta t + \beta\Delta\dot{\varepsilon}\Delta t - \Delta\varepsilon_s^k - \beta\Delta\dot{\varepsilon}_d^k\Delta t \\ \sigma_s^k - \sigma_d^k \end{bmatrix}, \tag{12.48}$$

where the superscript $(\cdot)^k$ denotes the k-th local iteration.

The determinant of $\boldsymbol{J}$ reads

$$\det\boldsymbol{J} = \beta\Delta t\left(\frac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} + \frac{\partial\sigma_d}{\partial\varepsilon_d}\right) + \frac{\partial\sigma_d}{\partial\dot{\varepsilon}_d}. \tag{12.49}$$

Clearly in the current setup, there is no guarantee for $\boldsymbol{J}$ to be strictly invertible. If $\boldsymbol{J}$ appears to be ill-conditioned, low rank updates such as the BFGS method, which do not rely on Jacobian, can be used to solve the system. The inverse $\boldsymbol{J}^{-1}$ can be analytically expressed as

$$\boldsymbol{J}^{-1} = \frac{-1}{\det\boldsymbol{J}}\begin{bmatrix} \dfrac{\partial\sigma_d}{\partial\varepsilon_d}\beta\Delta t & \dfrac{\partial\sigma_d}{\partial\dot{\varepsilon}_d} & -\beta\Delta t \\ \dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s}\beta\Delta t + \dfrac{\partial\sigma_d}{\partial\dot{\varepsilon}_d} & -\dfrac{\partial\sigma_d}{\partial\dot{\varepsilon}_d} & \beta\Delta t \\ -\dfrac{\partial\sigma_d}{\partial\varepsilon_d} & \dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} + \dfrac{\partial\sigma_d}{\partial\varepsilon_d} & 1 \end{bmatrix}. \tag{12.50}$$

Eq. (12.50) can be directly used in iterations. The computation of the numerical inverse of the Jacobian can be avoided so that the rounding error in float point arithmetic can be minimised as long as $\det\boldsymbol{J}$ is not zero.

A Simple Case

In the case of nonlinear spring and constant damping coefficient η,

$$\frac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} = E(\varepsilon_s), \qquad \frac{\partial\sigma_d}{\partial\varepsilon_d} = 0, \qquad \frac{\partial\sigma_d}{\partial\dot{\varepsilon}_d} = \eta\alpha\left|\dot{\varepsilon}_d\right|^{\alpha-1}. \tag{12.51}$$

Given that $\beta > 0$, $\eta > 0$, $\alpha > 0$ and $\Delta t > 0$, for the determinant to be strictly posi-

tive,

$$E(\varepsilon_s) > -\frac{\eta\alpha}{\beta\Delta t}\left|\dot{\varepsilon}_d\right|^{\alpha-1}. \tag{12.52}$$

Thus a non-softening spring would lead to a nonsingular Jacobian. For softening response, viz., $E(\varepsilon_s) < 0$,

$$\dot{\varepsilon}_d \neq \pm\sqrt[\alpha-1]{\frac{\beta\Delta t\left|E(\varepsilon_s)\right|}{\eta\alpha}} \tag{12.53}$$

guarantees an invertible Jacobian.

Furthermore, if the spring is linear elastic and $\alpha = 1$, the Jacobian is constant so all quantities can be solved in one step. To solve such a system with the aforementioned ODE solver based approach, multiple function evaluations are inevitable, although the analytical solution is available.

Tangent Stiffness and Damping Moduli

For an easier derivation of tangent stiffness and damping moduli, the stress response can be rewritten as follows,

$$\sigma = \frac{1}{2}\left(\sigma_s + \sigma_d\right). \tag{12.54}$$

Since $\sigma_s = \sigma_d$, in fact, any weighted average with non-zero weights can be used. At equilibrium, the residual equals zero vector, viz., $\boldsymbol{R} = \boldsymbol{0}$, differentiation results in

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{a}}\mathrm{d}\boldsymbol{a} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\mathrm{d}\boldsymbol{x} = \boldsymbol{0}, \tag{12.55}$$

in which $\boldsymbol{a} = \begin{bmatrix}\Delta\varepsilon & \Delta\dot{\varepsilon}\end{bmatrix}^{\mathrm{T}}$ is the increments of total strain and total strain rate and $\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \boldsymbol{J}$. Thus,

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}} = \begin{bmatrix}\dfrac{\partial \boldsymbol{x}}{\partial \Delta\varepsilon} & \dfrac{\partial \boldsymbol{x}}{\partial \Delta\dot{\varepsilon}}\end{bmatrix} = -\boldsymbol{J}^{-1}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{a}} = \frac{1}{\det \boldsymbol{J}}\begin{bmatrix} \dfrac{\partial \sigma_d}{\partial \varepsilon_d}\beta\Delta t & \dfrac{\partial \sigma_d}{\partial \dot{\varepsilon}_d}\beta\Delta t \\ \dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s}\beta\Delta t + \dfrac{\partial \sigma_d}{\partial \dot{\varepsilon}_d} & -\dfrac{\partial \sigma_d}{\partial \dot{\varepsilon}_d}\beta\Delta t \\ -\dfrac{\partial \sigma_d}{\partial \varepsilon_d} & \dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s}\beta\Delta t + \dfrac{\partial \sigma_d}{\partial \varepsilon_d}\beta\Delta t \end{bmatrix}. \tag{12.56}$$

It shall be pointed out that the parameter β is assumed to be a constant user input in the above derivation for brevity. If it is expressed as a function of $\boldsymbol{a}$ via Eq. (12.42), the corresponding $\partial\boldsymbol{x}/\partial\boldsymbol{a}$ can be derived accordingly. The procedure is presented in the appendix. The stiffness and damping moduli can be readily computed by using the chain rule. Knowing

that

$$\frac{\partial \sigma}{\partial \boldsymbol{x}} = \frac{1}{2}\begin{bmatrix} \dfrac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} & \dfrac{\partial \sigma_d}{\partial \varepsilon_d} & \dfrac{\partial \sigma_d}{\partial \dot{\varepsilon}_d} \end{bmatrix}, \tag{12.57}$$

then after some rearrangements,

$$K = \frac{\partial \sigma}{\partial \varepsilon} = \frac{\beta \Delta t}{\det \boldsymbol{J}} \frac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} \frac{\partial \sigma_d}{\partial \varepsilon_d}, \qquad C = \frac{\partial \sigma}{\partial \dot{\varepsilon}} = \frac{\beta \Delta t}{\det \boldsymbol{J}} \frac{\mathrm{d}\sigma_s}{\mathrm{d}\varepsilon_s} \frac{\partial \sigma_d}{\partial \dot{\varepsilon}_d}. \tag{12.58}$$

Eq. (12.58) is independent from the global time integration scheme and quadratic convergence is recovered. It can be easily verified that if the dashpot behaves like a spring so that $\mathrm{d}\sigma_s/\mathrm{d}\varepsilon_s = K_1$, $\mathrm{d}\sigma_d/\mathrm{d}\varepsilon_d = K_2$ and $\mathrm{d}\sigma_d/\mathrm{d}\dot{\varepsilon}_d = 0$, then the stiffness modulus K falls back to the widely recognised form of two springs in series,

$$K = \frac{K_1 K_2}{K_1 + K_2}. \tag{12.59}$$

Circumventions of Potential Numerical Difficulties

Numerical difficulties may arise if $\partial \eta / \partial \varepsilon_d$ becomes too large due to an improperly large steepness parameter g_1. The same situation occurs with $\partial \eta / \partial \dot{\varepsilon}_d$ and g_2. This problem can be eased by scaling steepness parameters according to the maximum strain and strain rate. Another major problem is that $\partial \sigma_d / \partial \dot{\varepsilon}_d$ approaches infinity at origin if the exponent α in power-law model is smaller than unity. An enough-close-to-origin strain rate can be often met, especially in forced vibrations. This may not be a problem if the corresponding damping modulus term $\partial \sigma_d / \partial \dot{\varepsilon}_d$ is not involved in the global equation of motion, in which case the quadratic convergence rate cannot be recovered. However, when combined with a spring to form a Maxwell model, an disproportionally large $\partial \sigma_d / \partial \dot{\varepsilon}_d$ may fail the local iteration. A possible workaround would be limiting the apparent viscosity to a finite value via extrapolation [37].

Here in lieu of the original power-law relationship, a cubic segment within a small width around origin is used to limit the corresponding derivative within a finite value so that the stress feedback can be rewritten as a piecewise-defined function such as

$$\sigma_d = \begin{cases} \eta\left(\varepsilon_d, \dot{\varepsilon}_d\right) \cdot \left(A\dot{\varepsilon}_d^3 + B\dot{\varepsilon}_d\right), & |\dot{\varepsilon}_d| \leqslant v, \\ \eta\left(\varepsilon_d, \dot{\varepsilon}_d\right) \cdot \operatorname{sign}\left(\dot{\varepsilon}_d\right) \cdot |\dot{\varepsilon}_d|^{\alpha}, & \text{else}, \end{cases} \tag{12.60}$$

where v is a user-defined constant that controls the size of cubic replacement. The cubic function, as an odd function, is chosen given the fact that the power-law Eq. (12.25) is also an odd function. With a cubic replacement, C^1 continuity can be ensured. By enforcing continuity of both σ_d and $\partial \sigma_d / \partial \dot{\varepsilon}_d$ at $|\dot{\varepsilon}_d| = v$, constants A and B can be solved as

$$A = \frac{\alpha - 1}{2} v^{\alpha - 3}, \qquad B = \frac{3 - \alpha}{2} v^{\alpha - 1}. \tag{12.61}$$

Hence,

$$\left.\frac{\partial\sigma_d}{\partial\dot{\varepsilon}_d}\right|_{\dot{\varepsilon}_d=0}=\left.\eta\left(\varepsilon_d,\dot{\varepsilon}_d\right)\right|_{\dot{\varepsilon}_d=0}B=\left.\eta\left(\varepsilon_d,\dot{\varepsilon}_d\right)\right|_{\dot{\varepsilon}_d=0}\frac{3-\alpha}{2}v^{\alpha-1}. \tag{12.62}$$

A properly defined v can greatly improve the performance of the proposed model for a small α as the apparent viscosity is bounded with such a modification. In the meantime, the damping stress is not largely affected since the strain rate is close to zero.

12.2.5. Implementation

```
int Maxwell::update_trial_status(const vec& t_strain, const vec& t_strain_rate) {
    incre_strain = (trial_strain = t_strain) - current_strain;
    incre_strain_rate = (trial_strain_rate = t_strain_rate) - current_strain_rate;

    if(fabs(incre_strain(0)) + fabs(incre_strain_rate(0)) <= datum::eps) return SUANPAN_SUCCESS;

    const auto& K1 = spring->get_trial_stiffness().at(0);
    const auto& K2 = damper->get_trial_stiffness().at(0);
    const auto& K3 = damper->get_trial_damping().at(0);
    const auto& F1 = spring->get_trial_stress().at(0);
    const auto& F2 = damper->get_trial_stress().at(0);

    // \beta\Delta{}t
    const auto factor_a = beta * *incre_time;

    const auto target = *incre_time * (current_strain_rate(0) - damper->get_current_strain_rate().at(0)) + factor_a * incre_strain_rate(0);

    vec solution(3, fill::zeros);

    counter = 0;

    if(double error, ref_error = 1.; use_matrix) {
        mat inv_jacobian(3, 3);

        inv_jacobian(0, 2) = -factor_a;
        inv_jacobian(1, 2) = factor_a;
        inv_jacobian(2, 2) = 1.;

        while(++counter < max_iteration) {
            const vec residual{incre_strain(0) - solution(0) - solution(1), target - solution(0) - factor_a * solution(2), F1 - F2};

            inv_jacobian(0, 0) = factor_a * K2;
            inv_jacobian(1, 0) = factor_a * K1 + K3;
            inv_jacobian(2, 0) = -K2;

            inv_jacobian(0, 1) = K3;
            inv_jacobian(1, 1) = -K3;
            inv_jacobian(2, 1) = K1 + K2;
```

```
39
40              const vec incre = inv_jacobian * residual / (factor_a * (K1 + K2) + K3);
41
42              if(1 == counter) ref_error = std::max(1., norm(residual));
43              suanpan_extra_debug("Maxwell local iteration error: %.4E.\n", error =
                ↪  norm(residual) / ref_error);
44              if(norm(incre) <= tolerance && error <= tolerance) break;
45              solution += incre;
46              spring->update_incre_status(solution(0));
47              damper->update_incre_status(solution(1), solution(2));
48          }
49      }
50      else
51          while(++counter < max_iteration) {
52              const auto residual_a = incre_strain(0) - solution(0) - solution(1);
53              const auto residual_b = target - solution(0) - factor_a * solution(2);
54              const auto residual_c = F1 - F2;
55              const auto residual = residual_a * K2 - residual_c + residual_b / factor_a * K3;
56              const auto jacobian = K1 + K2 + K3 / factor_a;
57              const auto incre = residual / jacobian;
58              if(1 == counter) ref_error = std::max(1., fabs(residual));
59              suanpan_extra_debug("Maxwell local iteration error: %.4E.\n", error =
                ↪  fabs(residual) / ref_error);
60              if(fabs(incre) <= tolerance && error <= tolerance) break;
61              solution(0) += incre;
62              solution(1) += residual_a - incre;
63              solution(2) += (residual_b - incre) / factor_a;
64              spring->update_incre_status(solution(0));
65              damper->update_incre_status(solution(1), solution(2));
66          }
67
68      if(max_iteration != counter) {
69          delay_counter = 0;
70
71          trial_stress = .5 * (F1 + F2);
72
73          trial_damping = trial_stiffness = factor_a / (factor_a * (K1 + K2) + K3) * K1;
74          trial_stiffness *= K2;
75          trial_damping *= K3;
76
77          return SUANPAN_SUCCESS;
78      }
79
80      if(1 >= proceed || ++delay_counter == proceed) {
81          suanpan_error("Maxwell: local iteration cannot converge within %u iterations.\n",
            ↪  max_iteration);
82          return SUANPAN_FAIL;
83      }
84
85      return reset_status();
86  }
```

13. Other

13.1. Gurson Model

13.1.1. Theory

13.1.2. Formulation

13.1.3. Implementation

```
int NonlinearGurson::update_trial_status(const vec& t_strain) {
    incre_strain = (trial_strain = t_strain) - current_strain;

    if(norm(incre_strain) <= tolerance) return SUANPAN_SUCCESS;

    trial_stress = current_stress + (trial_stiffness = initial_stiffness) * incre_strain;

    trial_history = current_history;
    auto& pe = trial_history(0); // equivalent plastic strain
    auto& f = trial_history(1);  // volume fraction
    const auto& current_pe = current_history(0);
    const auto& current_f = current_history(1);

    auto trial_s = tensor::dev(trial_stress);                              // trial deviatoric stress
    const auto trial_q = sqrt_three_two * tensor::stress::norm(trial_s); // trial von Mises stress
    const auto trial_p = tensor::mean3(trial_stress);                      // trial hydrostatic stress
    auto p = trial_p;                                                      // hydrostatic stress

    mat jacobian(4, 4);
    vec incre, residual(4);
    auto gamma = 0.;
    double denom;

    unsigned counter = 0;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("NonlinearGurson cannot converge within %u iterations.\n", max_iteration);
```

```
            return SUANPAN_FAIL;
        }

        const auto hardening = compute_hardening(pe);
        const auto &k = hardening(0), &dk = hardening(1);
        const auto hyper_term = 1.5 * q2 * p / k;
        const auto cosh_term = cosh(hyper_term);
        const auto sinh_term = sinh(hyper_term);
        const auto q = trial_q / (denom = 1. + six_shear * gamma);
        const auto an = para_b * exp(-.5 * pow((pe - en) / sn, 2.));
        const auto para_d = para_a * sinh_term;

        const auto diff_pe = pe - current_pe, diff_p = p - trial_p;

        residual(0) = q * q + k * k * (f * q1 * (2. * cosh_term - q1 * f) - 1.);

        if(1 == counter && residual(0) < 0.) return SUANPAN_SUCCESS;

        residual(1) = (1. - f) * k * diff_pe - 2. * gamma * q * q + p * diff_p / bulk;
        residual(2) = f - current_f + (1. - f) * diff_p / bulk - an * diff_pe;
        residual(3) = diff_p + para_a * gamma * f * k * sinh_term;

        jacobian(0, 0) = -2. * six_shear / denom * q * q;
        jacobian(0, 1) = (f * (4. * q1 * k * cosh_term - para_d / bulk * p) - 2. * k * (q1 *
        ↪  q1 * f * f + 1.)) * dk;
        jacobian(0, 2) = 2. * k * k * q1 * (cosh_term - q1 * f);
        jacobian(0, 3) = para_d / bulk * f * k;
        jacobian(1, 0) = 2. * q * q * (six_shear * gamma - 1.) / denom;
        jacobian(1, 1) = (1. - f) * (dk * diff_pe + k);
        jacobian(1, 2) = -k * diff_pe;
        jacobian(1, 3) = (p + diff_p) / bulk;
        jacobian(2, 0) = 0.;
        jacobian(2, 1) = an / sn / sn * (pe - en) * diff_pe - an;
        jacobian(2, 2) = 1. - diff_p / bulk;
        jacobian(2, 3) = (1. - f) / bulk;
        jacobian(3, 0) = para_d * f * k;
        jacobian(3, 1) = para_a * gamma * f * (sinh_term - hyper_term * cosh_term) * dk;
        jacobian(3, 2) = para_d * gamma * k;
        jacobian(3, 3) = 1. + 1.5 * para_a * q2 * gamma * f * cosh_term;

        if(!solve(incre, jacobian, residual)) return SUANPAN_FAIL;

        const auto error = norm(residual);
        suanpan_debug("NonlinearGurson local iteration error: %.5E.\n", error);
        if(error <= tolerance || norm(incre) <= tolerance) break;

        gamma -= incre(0);
        pe -= incre(1);
        f -= incre(2);
        p -= incre(3);

        f = std::min(std::max(f, 0.), 1.); // avoid overshoot
    }

    trial_s /= denom;

    mat left, right(4, 6);
```

```
84
85      right.row(0) = -six_shear / denom * trial_s.t();
86      right.row(1) = -2. * gamma * right.row(0) + p * tensor::unit_tensor2.t();
87      right.row(2) = (1. - f) * tensor::unit_tensor2.t();
88      right.row(3) = bulk * tensor::unit_tensor2.t();
89
90      if(!solve(left, jacobian, right)) return SUANPAN_FAIL;
91
92      trial_stress = trial_s + p * tensor::unit_tensor2;
93
94      trial_stiffness = six_shear / denom / 3. * unit_dev_tensor - six_shear / denom * trial_s
        ↪  * left.row(0) + tensor::unit_tensor2 * left.row(3);
95
96      return SUANPAN_SUCCESS;
97  }
```

13.2. The N-M Frame Element

In this section, we present a frame element that supports customisation of the N-M interaction with the generalised plasticity theory. The core model is taken mainly from the literature, further discussions can be seen elsewhere [38].

This element fuses the concepts of element, section and material altogether, thus, it can be deemed as a material model as well.

13.2.1. Preliminaries

Definitions and Kinematics

Consider a two–node beam[1] element connecting nodes i and j with its rigid body motions removed, the resulting degrees of freedom are axial deformation u, rotational deformation $\theta_{z,i}$ of end i and rotational deformation $\theta_{z,j}$ of end j, and additional two end rotations, $\theta_{y,i}$ and $\theta_{y,j}$ about weak axis in case of a 3D beam. It is assumed that the beam is rigid against torsion (no torsion deformation).

[1] The word 'beam' is used interchangeably with 'frame' and/or 'beam–column'.

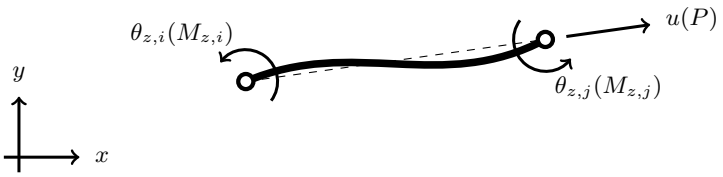


Figure 13.1.: deformation and resistance of a 2D beam

The elemental deformation vector $\boldsymbol{v}$ can then be defined as

$$\boldsymbol{v} = \begin{bmatrix} u & \theta_{z,i} & \theta_{z,j} & \theta_{y,i} & \theta_{y,j} \end{bmatrix}^{\mathrm{T}} \tag{13.1}$$

for 3D beam elements, where u is the axial deformation and θ is the nodal deformation about its chord, subscripts $(\cdot)_i$ and $(\cdot)_j$ denote two nodes/ends, $(\cdot)_z$ and $(\cdot)_y$ denote strong and weak axis, respectively. Accordingly, the elemental resistance $\boldsymbol{q}$ can be defined as

$$\boldsymbol{q} = \begin{bmatrix} P & M_{z,i} & M_{z,j} & M_{y,i} & M_{y,j} \end{bmatrix}^{\mathrm{T}}, \tag{13.2}$$

where P denotes axial force while M denotes end moment. The corresponding yield forces are denoted as P^y, M_z^y and M_y^y and it is assumed both ends have the same yield forces. Such definitions of elemental deformation and resistance are independent from the transformation between global and local reference frames, and thus, can be combined with either linear or corotational transformation. For this reason, the subsequent discussion is confined to the local reference frame only.

The elastic constitutive relationship, denoted by the superscript $(\cdot)^e$, is conventionally known as

$$\boldsymbol{q} = \boldsymbol{K}\boldsymbol{v}^e, \qquad \text{with} \qquad \boldsymbol{K} = \begin{bmatrix} \frac{EA}{L} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \frac{4EI_z}{L} & \frac{2EI_z}{L} & \cdot & \cdot \\ \cdot & \frac{2EI_z}{L} & \frac{4EI_z}{L} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \frac{4EI_y}{L} & \frac{2EI_y}{L} \\ \cdot & \cdot & \cdot & \frac{2EI_y}{L} & \frac{4EI_y}{L} \end{bmatrix}, \tag{13.3}$$

where L is the initial length of beam element, EA is the axial rigidity and EI denotes the flexural rigidity.

Basic Quantities

The above definition is widely adopted as the basic quantities of beam elements. However, it complicates plasticity formulation due to the coupling of (rotational) DoFs. For example, consider the moment $M_{z,i}$ at end i, which can be explicitly written as

$$M_{z,i} = \frac{EI_z}{L}\left(4\theta_{z,i} + 2\theta_{z,j}\right), \tag{13.4}$$

the above expression implies that both $\theta_{z,i}$ and $\theta_{z,j}$ contribute to $M_{z,i}$. Once node j yields, the plasticity developed on far end DoF $\theta_{z,j}$ would also affect near end moment $M_{z,i}$, thus, it is difficult to find a yield rotation that corresponds to yield moment $M^y_{z,i}$ by solely using near end rotation $\theta_{z,i}$.

Instead of $\boldsymbol{v}$, the proposed formulation is developed based on the following quantity

$$\boldsymbol{e} = \begin{bmatrix} \varepsilon \\ \chi_{z,i} \\ \chi_{z,j} \\ \chi_{y,i} \\ \chi_{y,j} \end{bmatrix} = \boldsymbol{S}\boldsymbol{v}, \qquad \boldsymbol{S} = \frac{1}{L}\begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 4 & 2 & \cdot & \cdot \\ \cdot & 2 & 4 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 4 & 2 \\ \cdot & \cdot & \cdot & 2 & 4 \end{bmatrix}. \tag{13.5}$$

As a result, the coupling of rotational degrees of freedom is removed. The stiffness matrix becomes a diagonal matrix. One may observe that the magnitudes of χ_i and χ_j correspond to that of sectional curvatures at two ends (using displacement interpolation as in the conventional displacement based Euler–Bernoulli beam element). By using which, it is possible to find a yield curvature χ^y that corresponds to M^y for each DoF. It is worth noting that the element length L is optionally moved from elasticity matrix $\boldsymbol{E}$ to $\boldsymbol{e}$, hence $\boldsymbol{e}$ is a strain–like quantity, rather than deformation in the conventional sense. However, we do not distinguish between those two terminologies and use elemental 'deformation' to refer to $\boldsymbol{e}$ as defined in Eq. (13.5).

For each end, the nodal deformation can be extracted as

$$\boldsymbol{e}_i = \begin{bmatrix} \varepsilon \\ \chi_{z,i} \\ \chi_{y,i} \end{bmatrix} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix}\boldsymbol{e} = \boldsymbol{T}_i\boldsymbol{e}, \qquad \boldsymbol{e}_j = \begin{bmatrix} \varepsilon \\ \chi_{z,j} \\ \chi_{y,j} \end{bmatrix} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix}\boldsymbol{e} = \boldsymbol{T}_j\boldsymbol{e}, \tag{13.6}$$

or concisely,

$$\boldsymbol{e}_{\aleph} = \boldsymbol{T}_{\aleph}\boldsymbol{e}, \tag{13.7}$$

where subscript $(\cdot)_{\aleph}$ denotes either $(\cdot)_i$ or $(\cdot)_j$. In this work, the presence of subscript $(\cdot)_{\aleph}$ implies that it is a nodal quantity, the same symbol without subscript $(\cdot)_{\aleph}$ denotes its elemental counterpart. We define $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$ to be transformation/selection matrices. The main purpose of adopting $\boldsymbol{e}$ in the formulation is two–fold: 1) to decouple nodal response so that plasticity (on rotational DoFs) can be developed **independently** at each end, and 2) to obtain a better

implication of the corresponding plastic deformation.

The nodal elasto-plastic constitutive relationship can now be conveniently expressed as

$$\boldsymbol{q}_{\aleph} = \boldsymbol{E}_{\aleph}\boldsymbol{e}^{e}_{\aleph} = \boldsymbol{E}_{\aleph}\left(\boldsymbol{e}_{\aleph} - \boldsymbol{e}^{p}_{\aleph}\right), \qquad \boldsymbol{E}_{\aleph} = \operatorname{diag}\begin{pmatrix} EA & EI_z & EI_y \end{pmatrix}. \tag{13.8}$$

The corresponding elemental version is

$$\boldsymbol{q} = \boldsymbol{E}\boldsymbol{e}^{e} = \boldsymbol{E}\left(\boldsymbol{e} - \boldsymbol{e}^{p}\right), \qquad \boldsymbol{E} = \operatorname{diag}\begin{pmatrix} EA & EI_z & EI_z & EI_y & EI_y \end{pmatrix}. \tag{13.9}$$

We further rewrite the above expression in the normalised space (normalised by yield force and yield deformation) as

$$\overline{\boldsymbol{q}}_{\aleph} = \overline{\boldsymbol{e}}^{e}_{\aleph} = \overline{\boldsymbol{e}}_{\aleph} - \overline{\boldsymbol{e}}^{p}_{\aleph}, \qquad \text{or} \qquad \overline{\boldsymbol{q}} = \overline{\boldsymbol{e}}^{e} = \overline{\boldsymbol{e}} - \overline{\boldsymbol{e}}^{p}, \tag{13.10}$$

where the overbar $\overline{(\cdot)}$ denotes the normalised counterpart, for example,

$$\boldsymbol{q}_{\aleph} = \begin{bmatrix} P^y & \cdot & \cdot \\ \cdot & M^y_z & \cdot \\ \cdot & \cdot & M^y_y \end{bmatrix} \overline{\boldsymbol{q}}_{\aleph}. \tag{13.11}$$

The elemental version with subscript $(\cdot)_{\aleph}$ omitted in Eq. (13.10) holds due to the fact that DoFs are now decoupled.

13.2.2. Generalised Plasticity Framework

The generalised plasticity concept [39] is followed loosely in this work. However, the two–surface (yield surface and bounding surface) concept is not adopted as the presence of which makes the quantification of hardening behaviour difficult.

The main challenge comes from the fact that N-M interaction should be considered at each node, the activation of plasticity (of moment) at either end should be relatively independent from the other end. However, as there is only one axial force which is shared between two ends, its plasticity history affects both ends. Simply adopting a conventional multisurface formulation [3] leads to potential local bifurcation issues, as under certain conditions, one surface would become **redundant**. In the following, a special formulation that allows independent activation of plasticity at each end while enables proper hardening behaviour is presented.

Yield Function

Nodal Yield Function For each end, the nodal resistance $\boldsymbol{q}_{\aleph}$ should be bounded by the corresponding nodal yield surface $f_{\aleph}$, then, the admissible region is defined by $f_{\aleph} \leqslant 0$ while

the inadmissible region is equivalent to $f_{\aleph} > 0$. For each node, $f_{\aleph}$ can be simply chosen as

$$f_{\aleph} = \Phi_{\aleph}, \tag{13.12}$$

where $\Phi_{\aleph} = \Phi_{\aleph}(\boldsymbol{s}_{\aleph}, \alpha_{\aleph})$ is the N-M interaction surface based on the shifted resistance $\boldsymbol{s}_{\aleph} = \boldsymbol{q}_{\aleph} - \boldsymbol{\beta}_{\aleph}$ and the equivalent plastic strain $\alpha_{\aleph}$, $\boldsymbol{\beta}_{\aleph}$ is similar to the concept of back stress in conventional plasticity models, here back resistance that defines the centre of interaction surface. The N-M interaction surface can not only change its location (governed by $\boldsymbol{\beta}_{\aleph}$) but also grow its size (governed by $\alpha_{\aleph}$) accordingly.

Elemental Yield Function The elemental yield surface f should be able to capture yielding of any ends, a possible option is

$$f = \sum \langle f_{\aleph} \rangle = \sum \langle \Phi_{\aleph} \rangle = \langle \Phi_i \rangle + \langle \Phi_j \rangle, \tag{13.13}$$

The $\langle \cdot \rangle$ symbol denotes the Macaulay bracket. By Eq. (13.13), as long as one end yields (or both yield), $f > 0$. Eq. (13.13) is closely related to the multisurface plasticity theory [3] but not identical. The exact multisurface plasticity based formulation [40] would cause local bifurcation as one of $\Phi_{\aleph}$ becomes redundant under pure axial loading. By such, a single elemental yield function f can be used to properly capture yielding of any ends.

Alternatively, a bounding surface concept, which is frequently adopted in models of geomaterials [41], concrete and metals, can be used to simulate more complex evolution of plasticity. For simplicity, the above single surface formulation is adopted in this work. It is worth noting that the major discrepancy compared to conventional plasticity models is that plasticity can develop at either end in a **relatively** independent manner, in the meantime, two ends are linked with each other via the shared axial force.

Often, f is a non-dimensional function of $\boldsymbol{s}$ and α. Dimensional analysis shows a plasticity framework based on normalised quantities can simplify both formulation and implementation, thus, in this work, the yield surface $f_{\aleph}$ is defined as follows instead.

$$f_{\aleph} = \Phi_{\aleph}(\overline{\boldsymbol{s}}_{\aleph}, \overline{\alpha}_{\aleph}), \tag{13.14}$$

where $\overline{\boldsymbol{s}}_{\aleph} = \overline{\boldsymbol{q}}_{\aleph} - \overline{\boldsymbol{\beta}}_{\aleph}$ with $\overline{\boldsymbol{q}}_{\aleph}$ be the normalised nodal resistance Eq. (13.11) and $\overline{\boldsymbol{\beta}}_{\aleph}$ be the normalised nodal back resistance, viz.,

$$\boldsymbol{\beta}_{\aleph} = \begin{bmatrix} P^y & \cdot & \cdot \\ \cdot & M_z^y & \cdot \\ \cdot & \cdot & M_y^y \end{bmatrix} \overline{\boldsymbol{\beta}}_{\aleph}. \tag{13.15}$$

N-M Interaction Surface

The **initial** N-M interaction surface is often defined as a non-homogeneous polynomial. For example, for a 2D element, a common option is

$$\Phi_{\aleph} = 1.15\left(\overline{P}-\overline{\beta}_P\right)^2+\left(\overline{M}_z-\overline{\beta}_{M_z}\right)^2+3.67\left(\overline{P}-\overline{\beta}_P\right)^2\left(\overline{M}_z-\overline{\beta}_{M_z}\right)^2-c,$$

where $\overline{\beta}_P$ and $\overline{\beta}_{M_z}$ denote the corresponding components of $\overline{\boldsymbol{\beta}}_{\aleph}$, c is a constant that determines the initial size of the surface.

Introducing isotropic hardening to the above equation via $c = c\left(\overline{\alpha}\right)$ indeed introduces hardening into the model but cannot recover the desired hardening behaviour due to the non-homogeneous attribute of $\Phi_{\aleph}$. Instead, accounting for the arbitrariness of $\Phi_{\aleph}$, one can, for example, define the interaction surface to be

$$\Phi_{\aleph} = 1.15\left(\frac{\overline{P}-\overline{\beta}_P}{h_P\left(\overline{\alpha}_{\aleph}\right)}\right)^2+\left(\frac{\overline{M}_z-\overline{\beta}_{M_z}}{h_{M_z}\left(\overline{\alpha}_{\aleph}\right)}\right)^2+3.67\left(\frac{\overline{P}-\overline{\beta}_P}{h_P\left(\overline{\alpha}_{\aleph}\right)}\right)^2\left(\frac{\overline{M}_z-\overline{\beta}_{M_z}}{h_{M_z}\left(\overline{\alpha}_{\aleph}\right)}\right)^2-c, \tag{13.16}$$

where $h\left(\overline{\alpha}_{\aleph}\right)$ is the isotropic hardening function that satisfies the conditions $h\left(\overline{\alpha}_{\aleph}\right) \geqslant 0$ and $h\left(0\right) = 1$.

We further express the interaction surface in its general form as

$$\Phi_{\aleph} = \sum_{i=1}^{n}\left(a_i\prod_r\left(\frac{\overline{r}-\overline{\beta}_r}{h_r\left(\overline{\alpha}_{\aleph}\right)}\right)^{b_{i,r}}\right)-c, \tag{13.17}$$

where n is the number of terms, a_i is the constant coefficient of each product, $b_{i,r}$ is the order of each bracket and r represents the specific force component $r \in (P,\ M_z,\ M_y)$. Eq. (13.17) serves as the formal definition of the nodal N-M interaction surface with hardening.

It is worthing noting that $h_P\left(\overline{\alpha}_{\aleph}\right)$ does not need to be the same as $h_{M_z}\left(\overline{\alpha}_{\aleph}\right)$. Different functions can be assigned to different components so that the interaction surface can change its shape during evolution (to mimic anisotropic hardening). An example is shown in Fig. 13.2.

Flow Rule

The evolution of plastic deformation $\overline{\boldsymbol{e}}^p$ shall be linked to the gradient $\boldsymbol{g}$ of plastic potential, which is simply taken as f, leading to $\boldsymbol{g} = \dfrac{\partial f}{\partial \overline{\boldsymbol{q}}}$. By denoting $\boldsymbol{\zeta} = \Gamma\left(\boldsymbol{g}\right)$, one can obtain

$$\dot{\overline{\boldsymbol{e}}}^p = \gamma\boldsymbol{\zeta}, \tag{13.18}$$

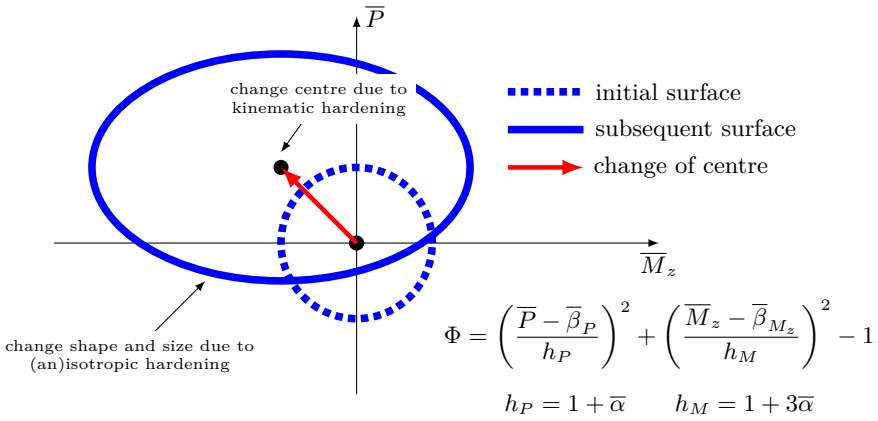


Figure 13.2.: mixed evolution of an example N-M interaction surface

in which γ denotes the plastic multiplier. It must be noted that since f is now based on normalised quantities, $\dot{\overline{\boldsymbol{e}}}^p$ denotes the normalised plastic deformation increment. Different options of function $\Gamma(\cdot)$ are available. In this work, the simplest form $\boldsymbol{\zeta} = \boldsymbol{g} = \dfrac{\partial f}{\partial \overline{\boldsymbol{q}}}$, implying the associative flow rule, is chosen, which can be further explicitly expressed as

$$\boldsymbol{g} = \sum \boldsymbol{T}_{\aleph}^{\mathrm{T}} \frac{\partial \langle \Phi_{\aleph} \rangle}{\partial \overline{\boldsymbol{q}}_{\aleph}} = \boldsymbol{T}_i^{\mathrm{T}} \frac{\partial \langle \Phi_i \rangle}{\partial \overline{\boldsymbol{q}}_i} + \boldsymbol{T}_j^{\mathrm{T}} \frac{\partial \langle \Phi_j \rangle}{\partial \overline{\boldsymbol{q}}_j}. \tag{13.19}$$

It should also be emphasised that only one plastic multiplier γ is adopted in the present formulation. Other options of $\boldsymbol{\zeta}$ include

$$\boldsymbol{\zeta} = \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}, \tag{13.20}$$

which has a fixed size (unity) that is beneficial in terms of alleviating potential numerical instability issues when computing the derivative of $\boldsymbol{\zeta}$.

Isotropic Hardening

For isotropic hardening, $\dot{\overline{\alpha}}_{\aleph}$ shall be related to some scalar measure of plastic deformation $\dot{\overline{\boldsymbol{e}}}^p$. The simplest one would be

$$\dot{\overline{\alpha}}_{\aleph} = \left\| \dot{\overline{\boldsymbol{e}}}_{\aleph}^p \right\| = \| \boldsymbol{T}_{\aleph} \dot{\overline{\boldsymbol{e}}}^p \| = \gamma \| \boldsymbol{T}_{\aleph} \boldsymbol{\zeta} \|. \tag{13.21}$$

The above definition implies that nodal plastic deformation $\dot{\overline{\boldsymbol{e}}}_{\aleph}^p$ can be extracted from elemental plastic deformation $\dot{\overline{\boldsymbol{e}}}^p$ via

$$\dot{\overline{\boldsymbol{e}}}_{\aleph}^p = \boldsymbol{T}_{\aleph} \dot{\overline{\boldsymbol{e}}}^p. \tag{13.22}$$

The elemental equivalent plastic deformation $\dot{\overline{\alpha}}$ is not used in the formulation, and one must be aware of the fact that $\dot{\overline{\alpha}} \neq \dot{\overline{\alpha}}_i + \dot{\overline{\alpha}}_j$ in general cases. Essentially, $\dot{\overline{\alpha}}_{\aleph}$ is the length of projection

of $\dot{\overline{\boldsymbol{e}}}^p$ (a 5D vector in deformation space) onto the 3D sub-space. An illustration of such a projection for 2D elements is depicted in Fig. 13.3.

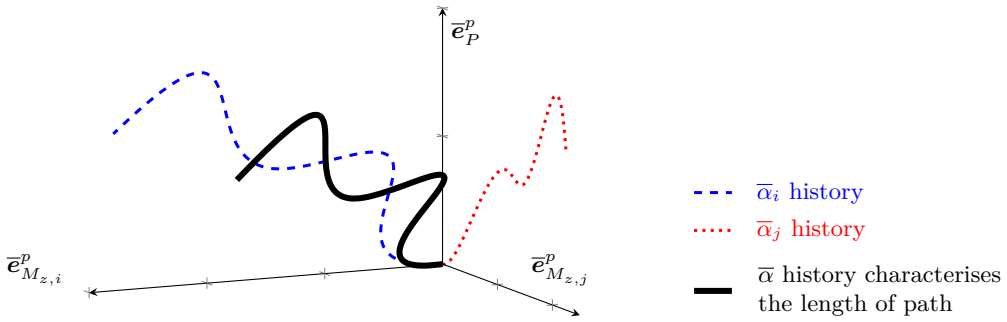


Figure 13.3.: definition of nodal equivalent plastic deformation of 2D beam

A popular isotropic hardening rule that adopts a linear hardening base and a saturation [5] can be defined as

$$h\left(\overline{\alpha}_{\aleph}\right) = 1 + H\overline{\alpha}_{\aleph} + s - se^{-m\overline{\alpha}_{\aleph}}, \tag{13.23}$$

where H is the linear isotropic hardening ratio, s is the saturation level and m controls hardening speed. In absence of H, it is easy to see that

$$\lim_{\overline{\alpha}_{\aleph}\to\infty} h\left(\overline{\alpha}_{\aleph}\right) = 1 + s. \tag{13.24}$$

Setting either $s = 0$ or $m = 0$ leads to pure linear isotropic hardening.

In this work, no anisotropic hardening is considered so that

$$h_P\left(\overline{\alpha}_{\aleph}\right) = h_{M_z}\left(\overline{\alpha}_{\aleph}\right) = h_{M_y}\left(\overline{\alpha}_{\aleph}\right) = h\left(\overline{\alpha}_{\aleph}\right). \tag{13.25}$$

Apparent Hardening Ratio Assume the interaction surface incorporates a linear isotropic hardening and is defined as

$$\Phi_{\aleph} = a\left(\frac{\overline{P} - \overline{\beta}_P}{1 + H\overline{\alpha}_{\aleph}}\right)^2 - c, \tag{13.26}$$

which is equivalent to

$$\Phi_{\aleph} = \left(\frac{\overline{P} - \overline{\beta}_P}{\sqrt{\frac{c}{a}} + \sqrt{\frac{c}{a}}H\overline{\alpha}_{\aleph}}\right)^2 - 1, \tag{13.27}$$

Noting that H is the hardening ratio defined in plastic deformation–force space, the counterpart, the apparent hardening ratio $\overline{H}$, in deformation–force space is then [3]

$$\overline{H} = \frac{\sqrt{\frac{c}{a}}H}{1+\sqrt{\frac{c}{a}}H} = \frac{H}{\sqrt{\frac{a}{c}}+H}. \tag{13.28}$$

This is useful in quantitative verification of the proposed model.

Kinematic Hardening

For kinematic hardening, the rate form of back resistance $\overline{\boldsymbol{\beta}}$ can be generally expressed as a function of itself and the increment of plastic deformation $\dot{\bar{\boldsymbol{e}}}^p$,

$$\dot{\overline{\boldsymbol{\beta}}} = \Xi\left(\overline{\boldsymbol{\beta}}, \dot{\bar{\boldsymbol{e}}}^p\right). \tag{13.29}$$

The bounding type hardening can be achieved by implementing a proper kinematic hardening model explained as follows. Let the size of back resistance $\overline{\boldsymbol{\beta}}$ be bounded by a fixed size denoted by B_s, that is

$$\left\|\overline{\boldsymbol{\beta}}\right\| \leqslant B_s. \tag{13.30}$$

The evolution direction of $\overline{\boldsymbol{\beta}}$ is determined by both the current position of $\overline{\boldsymbol{\beta}}$ and that of the corresponding projected image (onto the bounding surface), which is revolving around the origin and can be determined by the direction of plastic flow $\dot{\bar{\boldsymbol{e}}}^p$, meaning that $\dot{\overline{\boldsymbol{\beta}}}$ always points to $B_s\frac{\dot{\bar{\boldsymbol{e}}}^p}{\|\dot{\bar{\boldsymbol{e}}}^p\|}$. The evolution speed is governed by how close the current $\overline{\boldsymbol{\beta}}$ is to the bounding limit, such a distance is denoted by D and can be characterised by, for example,

$$D = \frac{1}{2} - \frac{\overline{\boldsymbol{\beta}}}{2B_s}\cdot\frac{\dot{\bar{\boldsymbol{e}}}^p}{\|\dot{\bar{\boldsymbol{e}}}^p\|}, \tag{13.31}$$

which ranges from 0 to 1. The $\cdot$ operator denotes inner product.

Combing the direction and speed together, accounting for that $\frac{\dot{\bar{\boldsymbol{e}}}^p}{\|\dot{\bar{\boldsymbol{e}}}^p\|} = \frac{\boldsymbol{\zeta}}{\|\boldsymbol{\zeta}\|}$ holds for associative plastic flow, the rate form of $\overline{\boldsymbol{\beta}}$ can be expressed as

$$\dot{\overline{\boldsymbol{\beta}}} = \gamma D\left(B_s\frac{\boldsymbol{\zeta}}{\|\boldsymbol{\zeta}\|} - \overline{\boldsymbol{\beta}}\right). \tag{13.32}$$

The concept is depicted in Fig. 13.4. More complex formulations of kinematic hardening of this type are often seen in constitutive models for geomaterials and metals.

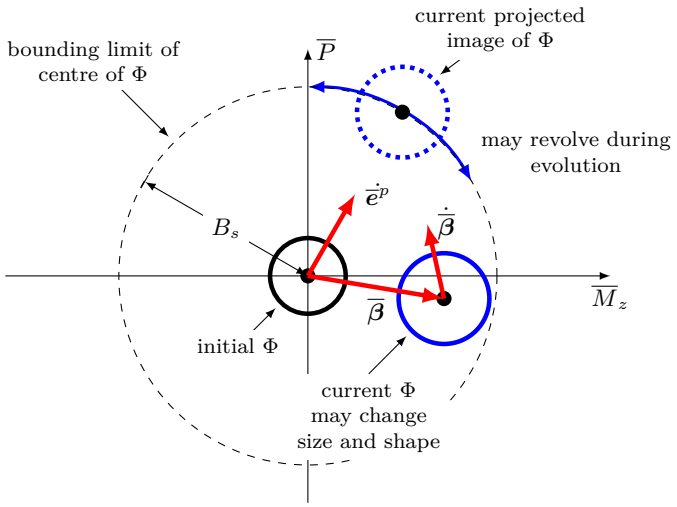


Figure 13.4.: bounded evolution of nodal back resistance of a 2D beam

Armstrong–Fredrick Type By choosing $D = K_a\|\boldsymbol{\zeta}\|$, the above bounding type hardening rule falls back to the Armstrong–Fredrick type [4] kinematic hardening that can be expressed as

$$\dot{\overline{\boldsymbol{\beta}}} = K_b\dot{\overline{\boldsymbol{e}}}^p - K_a\|\dot{\overline{\boldsymbol{e}}}^p\|\overline{\boldsymbol{\beta}}. \tag{13.33}$$

In the above definition, K_a and $K_b = K_aB_s$ are two kinematic hardening ratios. Setting $K_a = 0$ with $K_b \neq 0$ leads to pure linear kinematic hardening behaviour. Alternatively, a Chaboche type multiplicative model [6] can also be used. Furthermore, similar to isotropic hardening, different kinematic hardening rules can be assigned to axial/moment components, see the example presented in the appendix. In this work, Eq. (13.33) is adopted accounting for both generality and simplicity.

Apparent Hardening Ratio Assume a linear kinematic hardening is defined as

$$\dot{\overline{\boldsymbol{\beta}}} = K\dot{\overline{\boldsymbol{e}}}^p. \tag{13.34}$$

The apparent hardening ratio $\overline{K}$ observed in the deformation–force space is then [3]

$$\overline{K} = \frac{K}{1+K}. \tag{13.35}$$

This is useful in quantitative verification of the proposed model.

Loading/Unloading Conditions

Since the elemental yield function $f = \sum\langle f_{\aleph}\rangle$ defined in this work cannot take negative values, $f \equiv 0$. The conventional Kuhn–Tucker complementarity conditions should be derived based

on nodal surfaces accordingly [3]. In summary,

1. for all cases, $\gamma \geqslant 0$ and $\Phi_i \leqslant 0$ and $\Phi_j \leqslant 0$,
2. for elastic loading/unloading, $\gamma = 0$ and $\Phi_i < 0$ and $\Phi_j < 0$,
3. for plastic loading, $\gamma > 0$ and one of following cases:
   a) node i yields: $\Phi_i = 0$ and $\dot{\Phi}_i = 0$ and $\Phi_j < 0$,
   b) node j yields: $\Phi_i < 0$ and $\Phi_j = 0$ and $\dot{\Phi}_j = 0$,
   c) both nodes yield: $\Phi_i = 0$ and $\dot{\Phi}_i = 0$ and $\Phi_j = 0$ and $\dot{\Phi}_j = 0$,
4. for neutral loading, $\gamma = 0$ and one of the same three cases above.

Following the conventional style, the Kuhn–Tucker complementarity condition and consistency condition can be expressed as

$$\gamma \Phi_i \Phi_j = 0 \qquad \text{and} \qquad \gamma \dot{\Phi}_i \dot{\Phi}_j = 0. \tag{13.36}$$

Remarks

1. There is only **one** elemental yield function f, which contains **two** nodal interaction surfaces $\Phi_{\aleph}$.
2. The flow rule is derived from the unique elemental yield function f, thus the plastic flow defined in Eq. (13.18) is strictly associative. This differs from the multisurface plasticity framework. The resulting plastic deformation $\overline{\boldsymbol{e}}^p$ is an elemental quantity.
3. The equivalent plastic deformation $\overline{\alpha}_{\aleph}$ is computed based on part of elemental plastic deformation $\overline{\boldsymbol{e}}^p$ and may have different histories for two ends. Furthermore, in general, $\overline{\alpha} \neq \overline{\alpha}_i + \overline{\alpha}_j$.
4. The back resistance $\overline{\boldsymbol{\beta}}$ is based on elemental plastic deformation $\overline{\boldsymbol{e}}^p$. No matter which end yields, the same evolution rule applies to both elemental and nodal quantities.
5. The conventional fibre based beam elements have symmetric tangent stiffness matrix. Depending on the specific plastic flow and hardening rules used, such a symmetry is, in general, not guaranteed for generalised plasticity based beam elements.

13.2.3. Discrete System

The discrete local system in vectorial form is derived in this section for 3D beam elements accounting for five elemental deformation/force components, that is equivalent to three nodal components. In the case of 2D elements, it reduces to three elemental components and two nodal components. The formulation is almost identical for both 2D and 3D elements, thus, only the 3D version is presented.

Elastic Loading/Unloading

By freezing plasticity, one can compute the elemental trial state as

$$\boldsymbol{q}^{\text{trial}} = \boldsymbol{q}_n + \boldsymbol{E}\left(\boldsymbol{e}_{n+1} - \boldsymbol{e}_n\right), \tag{13.37}$$

the nodal normalised version $\overline{\boldsymbol{q}}_{\aleph}^{\text{trial}}$ can be extracted accordingly via the relationship

$$\boldsymbol{T}_{\aleph}\boldsymbol{q}^{\text{trial}} = \begin{bmatrix} P^y & \cdot & \cdot \\ \cdot & M_z^y & \cdot \\ \cdot & \cdot & M_y^y \end{bmatrix} \overline{\boldsymbol{q}}_{\aleph}^{\text{trial}}. \tag{13.38}$$

Then trial yield function can be evaluated for each node.

$$f_{\aleph}^{\text{trial}} = \Phi_{\aleph}^{\text{trial}}, \tag{13.39}$$

with $\Phi_{\aleph}^{\text{trial}} = \Phi\left(\overline{\boldsymbol{s}}_{\aleph}^{\text{trial}}, \overline{\alpha}_{\aleph,n}\right)$ in which $\overline{\boldsymbol{s}}_{\aleph}^{\text{trial}} = \overline{\boldsymbol{q}}_{\aleph}^{\text{trial}} - \overline{\boldsymbol{\beta}}_{\aleph,n}$.

If $f^{\text{trial}} = \sum \langle f_{\aleph}^{\text{trial}} \rangle = 0$, implying both $\Phi_{\aleph}^{\text{trial}}$ are non-positive, indicating both ends are undergoing elastic loading/unloading. Otherwise $f^{\text{trial}} > 0$ and local return mapping is required to determine the plastic state to meet conditions $\gamma > 0$ and $f = 0$.

Plasticity Evolution

For resistance and back resistance, one can obtain the following by using the implicit (backward Euler) integration.

$$\overline{\boldsymbol{q}}_{n+1} = \overline{\boldsymbol{q}}^{\text{trial}} - \dot{\overline{\boldsymbol{e}}}^p = \overline{\boldsymbol{q}}^{\text{trial}} - \gamma\boldsymbol{\zeta}_{n+1}, \tag{13.40}$$

$$\overline{\boldsymbol{\beta}}_{n+1} = \overline{\boldsymbol{\beta}}_n + K_b\gamma\boldsymbol{\zeta}_{n+1} - K_a\|\gamma\boldsymbol{\zeta}_{n+1}\|\overline{\boldsymbol{\beta}}_{n+1}. \tag{13.41}$$

Along with Eq. (13.13), Eq. (13.18), Eq. (13.19), Eq. (13.21) and Eq. (13.23), the system is complete.

Local Residual

For 3D beam elements with kinematic hardening, we take

$$\underbrace{\boldsymbol{x}}_{13} = \left[\underbrace{\overline{\boldsymbol{q}}}_{5} \quad \underbrace{\overline{\boldsymbol{\beta}}}_{5} \quad \underbrace{\overline{\alpha}_i}_{1} \quad \underbrace{\overline{\alpha}_j}_{1} \quad \underbrace{\gamma}_{1}\right] \tag{13.42}$$

as the local variable with subscript $(\cdot)_{n+1}$ omitted for brevity, the local residual is

$$\boldsymbol{R} = \begin{cases} \overline{\boldsymbol{q}} - \overline{\boldsymbol{q}}^{\text{trial}} + \gamma\boldsymbol{\zeta}, \\ (1 + K_a\gamma\|\boldsymbol{\zeta}\|)\,\overline{\boldsymbol{\beta}} - \overline{\boldsymbol{\beta}}_n - K_b\gamma\boldsymbol{\zeta}, \\ \overline{\alpha}_i - \overline{\alpha}_{i,n} - \gamma\|\boldsymbol{T}_i\boldsymbol{\zeta}\| \\ \overline{\alpha}_j - \overline{\alpha}_{j,n} - \gamma\|\boldsymbol{T}_j\boldsymbol{\zeta}\| \\ \langle\Phi_i\rangle + \langle\Phi_j\rangle, \end{cases} \tag{13.43}$$

Physically, plasticity evolution can be activated when either end yields, or both yield. Thus, each end shall be considered separately (but not strictly independently due to the shared axial force). The (plastic) deformation defined in this work can be transformed into the conventional sense via the $\boldsymbol{S}$ matrix, see Eq. (13.5).

Jacobian

By treating $\boldsymbol{\zeta}$ as an intermediate variable, the Jacobian $\boldsymbol{J}$ can be analytically expressed as

$$\boldsymbol{J} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\zeta}} \frac{\mathrm{d}\boldsymbol{\zeta}}{\mathrm{d}\boldsymbol{g}} \frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{x}}, \tag{13.44}$$

where

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} = \begin{bmatrix} \boldsymbol{I} & \cdot & \cdot & \cdot & \boldsymbol{\zeta} \\ \cdot & (1 + K_a\gamma\|\boldsymbol{\zeta}\|)\,\boldsymbol{I} & \cdot & \cdot & K_a\|\boldsymbol{\zeta}\|\overline{\boldsymbol{\beta}} - K_b\boldsymbol{\zeta} \\ \cdot & \cdot & 1 & \cdot & -\|\boldsymbol{T}_i\boldsymbol{\zeta}\| \\ \cdot & \cdot & \cdot & 1 & -\|\boldsymbol{T}_j\boldsymbol{\zeta}\| \\ \boldsymbol{\zeta}^{\mathrm{T}} & -\boldsymbol{\zeta}^{\mathrm{T}} & \dfrac{\partial\langle\Phi_i\rangle}{\partial\overline{\alpha}_i} & \dfrac{\partial\langle\Phi_j\rangle}{\partial\overline{\alpha}_j} & \cdot \end{bmatrix}, \tag{13.45}$$

and

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\zeta}} = \begin{bmatrix} \gamma \boldsymbol{I} \\ \frac{K_a \gamma}{\|\boldsymbol{\zeta}\|} \overline{\boldsymbol{\beta}} \boldsymbol{\zeta}^{\mathrm{T}} - K_b \gamma \boldsymbol{I} \\ -\gamma \frac{\boldsymbol{\zeta}^{\mathrm{T}} \boldsymbol{T}_i^{\mathrm{T}} \boldsymbol{T}_i}{\|\boldsymbol{T}_i \boldsymbol{\zeta}\|} \\ -\gamma \frac{\boldsymbol{\zeta}^{\mathrm{T}} \boldsymbol{T}_j^{\mathrm{T}} \boldsymbol{T}_j}{\|\boldsymbol{T}_j \boldsymbol{\zeta}\|} \\ \cdot \end{bmatrix}, \qquad \frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial \boldsymbol{g}}{\partial \overline{\boldsymbol{q}}} & -\frac{\partial \boldsymbol{g}}{\partial \overline{\boldsymbol{q}}} & \frac{\partial \boldsymbol{g}}{\partial \overline{\alpha}_i} & \frac{\partial \boldsymbol{g}}{\partial \overline{\alpha}_j} & \cdot \end{bmatrix}, \tag{13.46}$$

Note the above expression also takes advantage of the fact that $\dfrac{\partial}{\partial \overline{\boldsymbol{q}}} = -\dfrac{\partial}{\partial \overline{\boldsymbol{\beta}}}$.

The choice of local variable is not unique. Other options are available. However, with the above formulation, the Jacobian is greatly simplified at the cost of increasing the size of local system to 13. In absence of back resistance $\overline{\boldsymbol{\beta}}$, the size reduces to 8. For 2D beams, those two numbers are 9 and 6. The chosen scheme is believed to be a good balance between analytical expressiveness (simplicity) and numerical performance.

Consistent Tangent Operator

Full differentiation the local residual $\boldsymbol{R}$ at equilibrium $\boldsymbol{R} = \boldsymbol{0}$ gives

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{e}_{n+1}} \mathrm{d}\boldsymbol{e}_{n+1} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \mathrm{d}\boldsymbol{x} = \boldsymbol{0}, \tag{13.47}$$

which leads to the following expression after rearrangement,

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{e}_{n+1}} = -\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{e}_{n+1}}, \qquad \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{e}_{n+1}} = \begin{bmatrix} -\dfrac{\mathrm{d}\overline{\boldsymbol{q}}^{\mathrm{trial}}}{\mathrm{d}\boldsymbol{e}_{n+1}} \\ \boldsymbol{0} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{13.48}$$

with

$$\frac{\mathrm{d}\overline{\boldsymbol{q}}^{\mathrm{trial}}}{\mathrm{d}\boldsymbol{e}_{n+1}} = \mathrm{diag}\begin{pmatrix} P^y & M_z^y & M_z^y & M_y^y & M_y^y \end{pmatrix}^{-1} \boldsymbol{E}. \tag{13.49}$$

The consistent tangent operator can then be derived via the chain rule as

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\boldsymbol{e}_{n+1}} &= \frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\overline{\boldsymbol{q}}_{n+1}}\frac{\mathrm{d}\overline{\boldsymbol{q}}_{n+1}}{\mathrm{d}\boldsymbol{e}_{n+1}} = \frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\overline{\boldsymbol{q}}_{n+1}}\left(\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{e}_{n+1}}\right)^{\langle 1-5\rangle} \\ &= -\mathrm{diag}\begin{pmatrix}P^y & M_z^y & M_z^y & M_y^y & M_y^y\end{pmatrix}\left(\boldsymbol{J}^{-1}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{e}_{n+1}}\right)^{\langle 1-5\rangle},\end{aligned} \tag{13.50}$$

in which $(\cdot)^{\langle 1-5\rangle}$ denotes the first five rows of target quantity $(\cdot)$. By further appending a $\boldsymbol{S}$ matrix, the consistent tangent operator against $\boldsymbol{v}_{n+1}$ can be expressed as

$$\frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\boldsymbol{v}_{n+1}} = \frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\boldsymbol{e}_{n+1}}\frac{\mathrm{d}\boldsymbol{e}_{n+1}}{\mathrm{d}\boldsymbol{v}_{n+1}} = \frac{\mathrm{d}\boldsymbol{q}_{n+1}}{\mathrm{d}\boldsymbol{e}_{n+1}}\boldsymbol{S}. \tag{13.51}$$

13.2.4. Summary of The Proposed Model

The key expressions of the proposed model are listed in Table 13.1.

Table 13.1.: summary of key expressions and parameters

	proposed model
nodal extraction	$(\cdot)_{\aleph} = \boldsymbol{T}_{\aleph}(\cdot)$ for all nodal quantities
kinematics	$\overline{\boldsymbol{q}} = \overline{\boldsymbol{e}} - \overline{\boldsymbol{e}}^p$
yield surface	$f = \langle \Phi_i \rangle + \langle \Phi_j \rangle$
interaction surface	$\Phi_{\aleph} = \Phi_{\aleph}\left(\overline{\boldsymbol{q}}_{\aleph}, \overline{\boldsymbol{\beta}}_{\aleph}, \overline{\alpha}_{\aleph}\right)$
	$\Phi_{\aleph} = \sum_{i=1}^{n}\left(a_i \prod_r \left(\frac{\overline{r} - \overline{\beta}_r}{h_r\left(\overline{\alpha}_{\aleph}\right)}\right)^{b_{i,r}}\right) - c$
flow rule	$\dot{\overline{\boldsymbol{e}}}^p = \gamma\boldsymbol{\zeta} = \gamma\frac{\partial f}{\partial \overline{\boldsymbol{q}}}$
isotropic hardening	$\dot{\overline{\alpha}}_{\aleph} = \left\|\dot{\overline{\boldsymbol{e}}}^p_{\aleph}\right\| = \left\|\boldsymbol{T}_{\aleph}\dot{\overline{\boldsymbol{e}}}^p\right\| = \gamma\|\boldsymbol{T}_{\aleph}\boldsymbol{\zeta}\|$
	$h\left(\overline{\alpha}_{\aleph}\right) = 1 + H\overline{\alpha} + s - se^{-m\overline{\alpha}_{\aleph}}$
kinematic hardening	$\dot{\overline{\boldsymbol{\beta}}} = K_b\dot{\overline{\boldsymbol{e}}}^p - K_a\left\|\dot{\overline{\boldsymbol{e}}}^p\right\|\overline{\boldsymbol{\beta}}$

An example implementation of the proposed frame element can be seen in Algorithm 15. Additional stabilisation considerations, such as line search [42], can be accounted for to improve numerical performance.

Algorithm 15 state determination of the proposed frame element

Input: $\boldsymbol{e}_{n+1}$, $\boldsymbol{e}_n$, $\boldsymbol{q}_n$, $\overline{\boldsymbol{\beta}}_n$, $\overline{\alpha}_{\aleph,n}$
Output: $\boldsymbol{E}_{n+1}$, $\boldsymbol{q}_{n+1}$, $\overline{\boldsymbol{\beta}}_{n+1}$, $\overline{\alpha}_{\aleph,n+1}$

$\boldsymbol{q}^{\text{trial}} = \boldsymbol{q}_n + \boldsymbol{E}\left(\boldsymbol{e}_{n+1} - \boldsymbol{e}_n\right)$ and transform $\boldsymbol{q}^{\text{trial}}$ to $\overline{\boldsymbol{q}}^{\text{trial}}$
$\overline{\boldsymbol{q}}_{n+1} = \overline{\boldsymbol{q}}^{\text{trial}}$, $\overline{\boldsymbol{\beta}}_{n+1} = \overline{\boldsymbol{\beta}}_n$, $\overline{\alpha}_{\aleph,n+1} = \overline{\alpha}_{\aleph,n}$, $\boldsymbol{E}_{n+1} = \boldsymbol{E}$
compute f using $\overline{\boldsymbol{q}}_{n+1}$, $\overline{\boldsymbol{\beta}}_{n+1}$, $\overline{\alpha}_{\aleph,n+1}$ ▷ Eq. (13.39)
if $f > 0$ **then** ▷ plasticity evolution
  $\gamma = 0$
  while true **do**
    check if each $\Phi_\aleph$ yields or has yielded, assemble $\Phi_\aleph$, $\boldsymbol{\zeta}$ and their derivatives
    compute $\boldsymbol{R}$ and $\boldsymbol{J}$ ▷ Eq. (13.43) and Eq. (13.44)
    $\Delta = (\boldsymbol{J})^{-1}\boldsymbol{R}$ ▷ $\Delta = \left[\delta\overline{\boldsymbol{q}} \quad \delta\overline{\boldsymbol{\beta}} \quad \delta\overline{\alpha}_i \quad \delta\overline{\alpha}_j \quad \delta\gamma\right]$
    if $\|\Delta\| <$ tolerance **then**
      break
    end if
    $\overline{\boldsymbol{q}}_{n+1} \leftarrow \overline{\boldsymbol{q}}_{n+1} - \delta\overline{\boldsymbol{q}}$
    $\overline{\boldsymbol{\beta}}_{n+1} \leftarrow \overline{\boldsymbol{\beta}}_{n+1} - \delta\overline{\boldsymbol{\beta}}$
    $\overline{\alpha}_{\aleph,n+1} \leftarrow \overline{\alpha}_{\aleph,n+1} - \delta\overline{\alpha}_\aleph$
    $\gamma \leftarrow \gamma - \delta\gamma$
  end while
  transform $\overline{\boldsymbol{q}}_{n+1}$ to $\boldsymbol{q}_{n+1}$
  compute $\boldsymbol{E}_{n+1}$ ▷ Eq. (13.50)
end if

13.2.5. Implementation

The N-M element introduced in this section can be implemented as an element, or a section, or a material model. A more refined implementation can split the element into two parts: 1) an element model that handles converting global nodal quantities to local quantities Eq. (13.5) and 2) a section model that handled the integration at two end nodes.

Element Skeleton

The following skeleton handles elemental quantities and sends them to section model to compute the corresponding response.

```
1 int NMB21::update_status() {
2     b_trans->update_status();
3
4     if(b_section->update_trial_status(b_trans->to_local_vec(get_trial_displacement()) /
      ↪  length) != SUANPAN_SUCCESS) return SUANPAN_FAIL;
5
6     trial_stiffness = b_trans->to_global_stiffness_mat(b_section->get_trial_stiffness() /
      ↪  length);
```

```
    trial_resistance = b_trans->to_global_vec(b_section->get_trial_resistance());

    if(nlgeom) trial_geometry =
    ↪  b_trans->to_global_geometry_mat(b_section->get_trial_resistance());

    return SUANPAN_SUCCESS;
}
```

Section Integration

The section integration can be implemented as following. Various hardening rules can be applied, the forms of which are not explicitly required.

```
int NonlinearNM::update_trial_status(const vec& t_deformation) {
    const vec incre_deformation = (trial_deformation = t_deformation) - current_deformation;

    if(norm(incre_deformation) <= datum::eps) return SUANPAN_SUCCESS;

    trial_resistance = current_resistance + (trial_stiffness = initial_stiffness) *
    ↪  incre_deformation;

    const vec trial_q = trial_resistance.head(d_size) / yield_diag;

    const vec current_beta(&current_history(0), d_size);
    const vec bni = current_beta(ni), bnj = current_beta(nj);
    const auto &ani = current_history(d_size), &anj = current_history(d_size + 1llu);

    if(compute_f(trial_q(ni) - bni, compute_h(ani)) <= 0. && compute_f(trial_q(nj) - bnj,
    ↪  compute_h(anj)) <= 0.) return SUANPAN_SUCCESS;

    vec q;
    mat jacobian;

    if(SUANPAN_SUCCESS != compute_local_integration(q, jacobian)) return SUANPAN_FAIL;

    if(SectionType::NM2D == section_type) {
        trial_resistance = yield_diag % q;

        mat left(g_size, 3, fill::zeros);
        left.rows(ga) = diagmat(1. / yield_diag) * initial_stiffness;

        trial_stiffness = diagmat(yield_diag) * solve(jacobian, left).eval().head_rows(3);
    }
    else {
        trial_resistance.head(5) = yield_diag % q;

        mat left(g_size, 5, fill::zeros);
        left.rows(ga) = diagmat(1. / yield_diag) * initial_stiffness(0, 0, size(5, 5));

        trial_stiffness(0, 0, size(5, 5)) = diagmat(yield_diag) * solve(jacobian,
        ↪  left).eval().head_rows(5);
```

```
    }

    return SUANPAN_SUCCESS;
}
```

A typical implementation of computing Jacobian may look like this.

```
int VAFNM::compute_local_integration(vec& q, mat& jacobian) {
    trial_history = current_history;
    const vec current_beta(&current_history(0), d_size);
    const auto &ani = current_history(d_size), &anj = current_history(d_size + 1llu);

    vec beta(&trial_history(0), d_size, false, true);
    auto &ai = trial_history(d_size), &aj = trial_history(d_size + 1llu);
    auto &flagi = trial_history(d_size + 2llu), &flagj = trial_history(d_size + 3llu); // yield flag

    const vec trial_q = q = trial_resistance.head(d_size) / yield_diag;

    flagi = 0.;
    flagj = 0.;

    auto gamma = 0.;

    auto counter = 0u;
    auto ref_error = 1.;
    while(true) {
        if(max_iteration == ++counter) {
            suanpan_error("VAFNM cannot converge within %u iterations.\n", max_iteration);
            return SUANPAN_FAIL;
        }

        vec z(d_size, fill::zeros);
        mat pzpq(d_size, d_size, fill::zeros);
        vec pzpai(d_size, fill::zeros);
        vec pzpaj(d_size, fill::zeros);
        vec residual(g_size, fill::none);

        jacobian.eye(g_size, g_size);
        residual(ge).fill(0.);

        {
            const vec si = q(ni) - beta(ni), hi = compute_h(ai);
            if(const auto fi = compute_f(si, hi); fi > 0. || static_cast<bool>(flagi)) {
                flagi = 1.;
                residual(ge) += fi;

                const vec g = compute_df(si, hi);
                const vec dh = -si % compute_dh(ai) / hi;
                const mat dg = ti * compute_ddf(si, hi);
                z(ni) += g;
                pzpq += dg * ti.t();
                pzpai = dg * dh;
                jacobian(ge, gc).fill(dot(g, dh));
```

```
            }
        }

        {
            const vec sj = q(nj) - beta(nj), hj = compute_h(aj);
            if(const auto fj = compute_f(sj, hj); fj > 0. || static_cast<bool>(flagj)) {
                flagj = 1.;
                residual(ge) += fj;

                const vec g = compute_df(sj, hj);
                const vec dh = -sj % compute_dh(aj) / hj;
                const mat dg = tj * compute_ddf(sj, hj);
                z(nj) += g;
                pzpq += dg * tj.t();
                pzpaj = dg * dh;
                jacobian(ge, gd).fill(dot(g, dh));
            }
        }

        const vec m = normalise(z);

        const auto norm_mi = norm(m(ni));
        const auto norm_mj = norm(m(nj));

        if(1 == counter) {
            gamma = .5 * residual(ge(0)) / dot(m, z);
            q -= gamma * m;
            if(has_kinematic) {
                beta += gamma * kin_modulus % m;
                beta /= 1. + kin_base * gamma;
            }
            ai += gamma * norm_mi;
            aj += gamma * norm_mj;
            continue;
        }

        residual(ga) = q - trial_q + gamma * m;
        residual(gc).fill(ai - ani - gamma * norm_mi);
        residual(gd).fill(aj - anj - gamma * norm_mj);

        jacobian(ga, ge) = m;
        jacobian(gc, ge).fill(-norm_mi);
        jacobian(gd, ge).fill(-norm_mj);
        jacobian(ge, ga) = z.t();
        jacobian(ge, ge).fill(0.);

        mat prpz(g_size, d_size, fill::zeros), dzdx(d_size, g_size, fill::zeros);

        prpz.rows(ga) = gamma * eye(d_size, d_size);
        prpz.rows(gc) = -gamma * (ti * normalise(m(ni))).t();
        prpz.rows(gd) = -gamma * (tj * normalise(m(nj))).t();

        dzdx.cols(ga) = pzpq;
        dzdx.cols(gc) = pzpai;
        dzdx.cols(gd) = pzpaj;

        if(has_kinematic) {
```

```
            residual(gb) = (1. + kin_base * gamma) % beta - current_beta - gamma *
            ↪  kin_modulus % m;

            jacobian(gb, gb) += diagmat(kin_base * gamma);
            jacobian(gb, ge) = kin_base % beta - kin_modulus % m;
            jacobian(ge, gb) = -z.t();

            prpz.rows(gb) = diagmat(-gamma * kin_modulus);

            dzdx.cols(gb) = -pzpq;
        }

        const vec incre = solve(jacobian += prpz * (eye(d_size, d_size) - m * m.t()) /
        ↪  norm(z) * dzdx, residual);

        auto error = norm(residual);
        if(2 == counter) ref_error = std::max(1., error);
        suanpan_debug("VAFNM local iteration error: %.5E.\n", error /= ref_error);
        if(norm(incre) <= tolerance && error <= tolerance) return SUANPAN_SUCCESS;

        q -= incre(ga);
        if(has_kinematic) beta -= incre(gb);
        ai -= incre(gc(0));
        aj -= incre(gd(0));
        gamma -= incre(ge(0));
    }
}
```

Bibliography

[1] T. L. Chang, suanpan — an open source, parallel and heterogeneous finite element analysis framework (2022). doi:10.5281/ZENODO.1285221.

[2] P. Helnwein, Some remarks on the compressed matrix representation of symmetric second-order and fourth-order tensors, Computer Methods in Applied Mechanics and Engineering 190 (22-23) (2001) 2753–2770. doi:10.1016/s0045-7825(00)00263-2.

[3] J. C. Simo, T. J. R. Hughes, Computational Inelasticity, Springer-Verlag, 1998. doi:10.1007/b98904.

[4] C. O. Frederick, P. J. Armstrong, A mathematical representation of the multiaxial Bauschinger effect, Materials at High Temperatures 24 (1) (2007) 1–26. doi:10.3184/096034007x207589.

[5] E. Voce, Analysis of stress strain curves, The Aeronautical Journal 59 (534) (1955) 442–442. doi:10.1017/s0368393100118759.

[6] J. L. Chaboche, Constitutive equations for cyclic plasticity and cyclic viscoplasticity, International Journal of Plasticity 5 (3) (1989) 247–302. doi:10.1016/0749-6419(89)90015-6.

[7] A. Zona, A. Dall'Asta, Elastoplastic model for steel buckling-restrained braces, Journal of Constructional Steel Research 68 (1) (2012) 118–125. doi:10.1016/j.jcsr.2011.07.017.

[8] M. Kenawy, S. Kunnath, S. Kolwankar, A. Kanvinde, Concrete uniaxial nonlocal damage-plasticity model for simulating post-peak response of reinforced concrete beam-columns under cyclic loading, Journal of Structural Engineering 146 (5) (may 2020). doi:10.1061/(asce)st.1943-541x.0002592.

[9] J. Lemaitre, Coupled elasto-plasticity and damage constitutive equations, Computer Methods in Applied Mechanics and Engineering 51 (1-3) (1985) 31–49. doi:10.1016/0045-7825(85)90026-x.

[10] P. Grassl, D. Xenos, U. Nyström, R. Rempling, K. Gylltoft, CDPM2: A damage-plasticity approach to modelling the failure of concrete, International Journal of Solids and Structures 50 (24) (2013) 3805–3816. doi:10.1016/j.ijsolstr.2013.07.008.

[11] S. Oller, E. Car, J. Lubliner, Definition of a general implicit orthotropic yield criterion, Computer Methods in Applied Mechanics and Engineering 192 (7-8) (2003) 895–912. doi:10.1016/s0045-7825(02)00605-9.

[12] E. E. Krasnovskiy, Finite element procedure for plastic flow of orthotropic composites with hoffman yield criterion, mathesis, University of Wales, Swansea (Mar. 2004).

[13] E. A. de Souza Neto, D. Perić, D. R. J. Owen, Computational Methods for Plasticity, Wiley, 2008. doi:10.1002/9780470694626.

[14] L. F. Sirumbal-Zapata, C. Málaga-Chuquitaype, A. Y. Elghazouli, A three-dimensional plasticity-damage constitutive model for timber under cyclic loads, Computers & Structures 195 (2018) 47–63. doi:10.1016/j.compstruc.2017.09.010.

[15] J. Lee, G. L. Fenves, Plastic-damage model for cyclic loading of concrete structures, Journal of Engineering Mechanics 124 (8) (1998) 892–900. doi:10.1061/(asce)0733-9399(1998)124:8(892).

[16] M. G. Katona, Modifying Duncan–Selig soil model for plasticlike behavior, Transportation Research Record: Journal of the Transportation Research Board 2511 (1) (2015) 53–62. doi:10.3141/2511-07.

[17] D. Perić, On a class of constitutive equations in viscoplasticity: Formulation and computational issues, International Journal for Numerical Methods in Engineering 36 (8) (1993) 1365–1393. doi:10.1002/nme.1620360807.

[18] T. L. Chang, C.-L. Lee, Numerical simulation of generalised maxwell-type viscous dampers with an efficient iterative algorithm, Mechanical Systems and Signal Processing 170 (2022) 108795. doi:10.1016/j.ymssp.2021.108795.

[19] Y.-S. Wu, Immiscible displacement of non-newtonian fluids, in: Multiphase Fluid Flow in Porous and Fractured Reservoirs, Elsevier, 2016, pp. 127–166. doi:10.1016/b978-0-12-803848-2.00007-6.

[20] D. Lee, D. P. Taylor, Viscous damper development and future trends, The Structural Design of Tall Buildings 10 (5) (2001) 311–320. doi:10.1002/tal.188.

[21] A. Lago, D. Trabucco, A. Wood, Damping Technologies for Tall Buildings, Elsevier, 2018.

[22] N. K. Hazaveh, G. W. Rodgers, J. G. Chase, S. Pampanin, Experimental test and validation of a direction- and displacement-dependent viscous damper, Journal of Engineering Mechanics 143 (11) (2017) 04017132. doi:10.1061/(asce)em.1943-7889.0001354.

[23] Z. Lu, J. Li, C. Jia, Studies on energy dissipation mechanism of an innovative viscous damper filled with oil and silt, Sustainability 10 (6) (2018) 1777. doi:10.3390/su10061777.

[24] H. Iemura, M. H. Pradono, Advances in the development of pseudo-negative-stiffness dampers for seismic response control, Structural Control and Health Monitoring (2009) n/a–n/adoi:10.1002/stc.345.

[25] J. Høgsberg, The role of negative stiffness in semi-active control of magneto-rheological dampers, Structural Control and Health Monitoring 18 (3) (2011) 289–304. doi:10.1002/stc.371.

[26] P. Zhou, H. Li, Modeling and control performance of a negative stiffness damper for suppressing stay cable vibrations, Structural Control and Health Monitoring 23 (4) (2015) 764–782. doi:10.1002/stc.1809.

[27] M. Javanbakht, S. Cheng, F. Ghrib, Refined damper design formula for a cable equipped with a positive or negative stiffness damper, Structural Control and Health Monitoring 25 (10) (2018) e2236. doi:10.1002/stc.2236.

[28] N. Makris, M. C. Constantinou, Fractional-derivative maxwell model for viscous dampers, Journal of Structural Engineering 117 (9) (1991) 2708–2724. doi:10.1061/(asce)0733-9445(1991)117:9(2708).

[29] Y.-K. Wen, Method for random vibration of hysteretic systems, Journal of the Engineering Mechanics Division 102 (2) (1976) 249–263. doi:10.1061/jmcea3.0002106.

[30] S. Gong, Y. Zhou, Experimental study and numerical simulation on a new type of viscoelastic damper with strong nonlinear characteristics, Structural Control and Health Monitoring 24 (4) (2016) e1897. doi:10.1002/stc.1897.

[31] C.-M. Chang, S. Strano, M. Terzo, Modelling of hysteresis in vibration control systems by means of the bouc-wen model, Shock and Vibration 2016 (2016) 1–14. doi:10.1155/2016/3424191.

[32] K. Kasai, K. Oohara, Y. Sekiguchi, JSSI manual for building passive control technology part-II time-history analysis model for viscous dampers, in: 13th World Conference on Earthquake Engineering, 2004.

[33] S. Akcelyan, D. G. Lignos, T. Hikino, Adaptive numerical method algorithms for nonlinear viscous and bilinear oil damper models subjected to dynamic loading, Soil Dynamics and Earthquake Engineering 113 (2018) 488–502. doi:10.1016/j.soildyn.2018.06.021.

[34] T. Hatada, T. Kobori, M. Ishida, N. Niwa, Dynamic analysis of structures with maxwell model, Earthquake Engineering & Structural Dynamics 29 (2) (2000) 159–176. doi:10.1002/(sici)1096-9845(200002)29:2<159::aid-eqe895>3.0.co;2-1.

[35] E. Fehlberg, Low–order classical Runge–Kutta formulas with stepsize control and their application to some heat transfer problems, techreport NASA TR R-315, NASA (1969).

[36] G. G. Dahlquist, A special stability problem for linear multistep methods, BIT 3 (1) (1963) 27–43. doi:10.1007/bf01963532.

[37] Y.-S. Wu, K. Pruess, A numerical method for simulating non-newtonian fluid flow and displacement in porous media, Advances in Water Resources 21 (5) (1998) 351–362. doi:10.1016/s0309-1708(97)00004-3.

[38] T. L. Chang, C.-L. Lee, Reformulation of concentrated plasticity frame element with N-M interaction and generalised plasticity, Journal of Structural Engineering 149 (9) (sep 2023). doi:10.1061/JSENDH/STENG-12176.

[39] F. Auricchio, R. L. Taylor, A generalized elastoplastic plate theory and its algorithmic implementation, International Journal for Numerical Methods in Engineering 37 (15) (1994) 2583–2608. doi:10.1002/nme.1620371506.

[40] S. M. Kostic, F. C. Filippou, C.-L. Lee, An efficient beam-column element for inelastic 3d frame analysis, in: M. Papadrakakis, M. Fragiadakis, V. Plevris (Eds.), Computational Methods in Applied Sciences, Springer Netherlands, 2013, pp. 49–67. doi:10.1007/978-94-007-6573-3_3.

[41] Y. F. Dafalias, M. T. Manzari, Simple plasticity sand model accounting for fabric change effects, Journal of Engineering Mechanics 130 (6) (2004) 622–634. doi:10.1061/(asce)0733-9399(2004)130:6(622).

[42] M. Dutko, D. Perić, D. Owen, Universal anisotropic yield criterion based on superquadric functional representation: Part 1. algorithmic issues and accuracy analysis, Computer Methods in Applied Mechanics and Engineering 109 (1-2) (1993) 73–93. doi:10.1016/0045-7825(93)90225-m.