SANDIA REPORT

SAND2018-12601
Unlimited Release
Printed November, 2018

Enhancements to the Munson-Dawson Model for Rock Salt

Benjamin Reedlunn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND2018-12601
Unlimited Release
Printed November, 2018

Enhancements to the Munson-Dawson Model for Rock Salt

Benjamin Reedlunn
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0840

Abstract

The Munson-Dawson (MD) constitutive model was originally developed in the 1980's to predict the thermomechanical behavior of rock salt. Since then, it has been used to simulate the evolution of the underground in nuclear waste repositories, mines, and storage caverns for gases and liquids. This report covers three enhancements to the MD model. (1) New transient and steady-state rate terms were added to capture salt's creep behavior at low equivalent stresses (below about 8 MPa). These new terms were calibrated against a series of triaxial compression creep experiments on salt from the Waste Isolation Pilot Plant. (2) The equivalent stress measure was changed from the Tresca stress to the Hosford stress. By varying a single exponent, the Hosford stress can reduce to the Tresca stress, the von Mises stress, or a range of behaviors in-between. This exponent was calibrated against true triaxial compression experiments on salt hollow cylinders. (3) The MD model's numerical implementation was overhauled, adding a line search algorithm to the implicit solution scheme. The new implementation was verified against analytical solutions, and benchmarked against a pre-existing implementation on a room closure simulation. The new implementation predicted virtually identical room closure, yet sped up the simulation by 16×. (The source code of the new implementation is included in an appendix of this report.)

Acknowledgment

Although I, Benjamin Reedlunn, am the only author listed, many other people contributed to this work in small and not-so-small ways. The members of Joint Project WEIMOS, Andreas Hampel, Klaus Salzer, Ralf-Michael Günther, Christoph Lüdeling, Karl-Heinz Lux, Kai Herchen, Joachim Stahlmann, Andreas Gährken, Savas Yildirim, and Kurt Staudtmeister, informally reviewed the work, gave thoughtful suggestions (especially Christoph Lüdeling), and were wonderful hosts on my trips to Germany. Christi Leigh saw the importance of this work and found funding to support it. My co-workers, William Scherzinger and Brian Lester introduced me to the Hosford equivalent stress and helped me create the numerical implementation discussed herein. My other colleagues, Steven Sobolik and James Bean reviewed drafts of this report and made useful suggestions for improvements. Last, but not least, I appreciate the stimulating discussions with former Sandians, Frank Hansen, J. Guadalupe Argüello, Harold Morgan, and Charles M. Stone.

Contents

Appendix

List of Figures

List of Tables

Chapter 1

Introduction

The Munson-Dawson model (also known as the MD or Multi-mechanism Deformation model) is a constitutive model for the thermomechanical behavior of rock salt (Munson and Dawson 1979; Munson and Dawson 1982; Munson et al. 1989). Rock salt constitutive models are used to simulate the evolution of the underground in salt mines, storage caverns for gases and liquids, and nuclear waste repositories. For example, Reedlunn (2016) recently used the MD model to reinvestigate predictions of Room D's closure at the Waste Isolation Pilot Plant (WIPP).

Munson and Dawson (1979) originally introduced their model with an ordinary differential equation for transient creep and three steady-state creep mechanisms: dislocation climb, an undefined mechanism, and dislocation glide. Munson and Dawson (1982) extended the model to better capture transient creep behavior during recovery (after a decrease in stress), and added stress dependence to the rate of strain hardening. In their last changes to the model, Munson et al. (1989) changed the equivalent stress measure from von Mises to Tresca, added a heavi-side step function in front of the dislocation glide mechanism, made the transient strain limit a function of temperature, and added an exponent of 2 to the transient strain differential equation. After the 1980's, damage and healing were added to the MD model, but it was renamed as the Munson-Dawson-Chan-Fossum (MDCF) model (Chan et al. 2001) and the original viscoplastic portion of the model was not modified. Thus, the MD model itself has not changed since 1989.

While the MD model has its strengths, such as ease of calibration and being relatively well known in the salt mechanics community, the Room D exercise in Reedlunn (2016) revealed some shortcomings:

1. The model cannot capture the creep behavior at low equivalent stresses (< 8 MPa).
2. The equivalent stress measure was not flexible enough to capture true triaxial creep measurements of Mellegard et al. (1992).
3. The numerical implementation in Sierra/Solid Mechanics (2017) frequently failed to converge, causing long simulation times.

This report discusses model enhancements that help resolve these shortcomings. Chapter 2 details the improvements to the model formulation. Chapter 3 covers the new numerical implementation. Chapter 4 discusses three verification tests and a benchmark test. Chapter 5 documents two new calibrations that fit new features in Chapter 2 against experimental data.

Chapter 2

Model Formulation

Section 2.1 presents the model in a infinitesimal strain setting and discusses the enhancements to the formulation. Section 2.4 then uses hypoelasticity to extend the model into the finite deformation realm. As is common in the geomechanics literature, compressive strains and stresses are treated as positive.

2.1 Infinitesimal Strain Formulation

The MD model is an isotropic, unified viscoplastic, material model. The total strain rate $\dot{\boldsymbol{\varepsilon}}$ is decomposed into an elastic strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{el}}$, a thermal strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{th}}$, and a viscoplastic strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$:

$$\dot{\boldsymbol{\varepsilon}} = \dot{\boldsymbol{\varepsilon}}^{\mathrm{el}} + \dot{\boldsymbol{\varepsilon}}^{\mathrm{th}} + \dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}. \tag{2.1}$$

The elastic portion of the MD model utilizes the following simple linear relationship between the elastic strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{el}}$ and the stress rate $\dot{\boldsymbol{\sigma}}$,

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\mathcal{C}} : \dot{\boldsymbol{\varepsilon}}^{\mathrm{el}} = \boldsymbol{\mathcal{C}} : \left(\dot{\boldsymbol{\varepsilon}} - \dot{\boldsymbol{\varepsilon}}^{\mathrm{th}} - \dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}\right) \tag{2.2}$$

$$\boldsymbol{\mathcal{C}} = (B - 2/3\,\mu)\,\boldsymbol{I} \otimes \boldsymbol{I} + 2\,\mu\,\boldsymbol{\mathcal{I}}, \tag{2.3}$$

where $\boldsymbol{\mathcal{C}}$ is the fourth-order elastic stiffness tensor composed of the bulk modulus B, the shear modulus μ, the second-order identity tensor $\boldsymbol{I}$, and the fourth-order symmetric identity tensor $\boldsymbol{\mathcal{I}}$. The thermal strain portion of the model is simply

$$\dot{\boldsymbol{\varepsilon}}^{\mathrm{th}} = -\alpha\,\dot{T}\,\boldsymbol{I} \tag{2.4}$$

where α is the coefficient of thermal expansion, and T is the temperature.

Plastic deformation of intact salt is isochoric and only occurs in the presence of shear stress. Originally, the MD model utilized the von Mises stress as its equivalent shear stress measure $\bar{\sigma}$, but Munson et al. (1989) switched $\bar{\sigma}$ to the Tresca stress. Here, it is changed again to an equivalent stress measure proposed by Hosford (1972):

$$\bar{\sigma} = \left\{\frac{1}{2}\left[|\sigma_1 - \sigma_2|^a + |\sigma_2 - \sigma_3|^a + |\sigma_1 - \sigma_3|^a\right]\right\}^{1/a}, \tag{2.5}$$

where σ_i are the principal stresses and a is a material parameter. As shown in Appendix A, Eq. (2.5) encompasses the Tresca stress ($a = 1$), the von Mises stress ($a = 2$), and a range of behaviors in-between ($1 < a < 2$). One can also reproduce the Tresca stress with $a = \infty$, the von Mises stress with $a = 4$, and behaviors in-between with $4 < a < \infty$. This second range avoids a potential singularity in the second derivative of Eq. (2.5), (see Appendix B), so the exponent is restricted to $a \geq 4$.

The viscoplastic strain evolves according to an associated flow rule

$$\dot{\boldsymbol{\varepsilon}}^{\text{vp}} = \dot{\bar{\varepsilon}}^{\text{vp}} \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}, \tag{2.6}$$

where $\dot{\bar{\varepsilon}}^{\text{vp}}$ is the equivalent viscoplastic strain rate. This rate can be decomposed into two components

$$\dot{\bar{\varepsilon}}^{\text{vp}} = \dot{\bar{\varepsilon}}^{\text{tr}} + \dot{\bar{\varepsilon}}^{\text{ss}}, \tag{2.7}$$

where $\dot{\bar{\varepsilon}}^{\text{tr}}$ is the transient equivalent viscoplastic strain rate and $\dot{\bar{\varepsilon}}^{\text{ss}}$ is the steady state equivalent viscoplastic strain rate.

The original MD model decomposed the steady state behavior into three mechanisms. Several investigators (e.g. Bérest et al. (2005), Bérest et al. (2015), Salzer et al. (2015), and Düsterloh et al. (2015)), however, have reported greater steady-state creep rates at $\bar{\sigma} < 8$ MPa than would be expected from extrapolating rates from higher stresses. Thus, a fourth mechanism is added to capture the steady-state creep at low equivalent stresses. The new expression for steady state creep is

$$\dot{\bar{\varepsilon}}^{\text{ss}} = \sum_{i=0}^{3} \dot{\bar{\varepsilon}}_i^{\text{ss}} \tag{2.8}$$

$$\dot{\bar{\varepsilon}}_i^{\text{ss}} = A_i \exp\left(-\frac{Q_i}{RT}\right)\left(\frac{\bar{\sigma}}{\mu}\right)^{n_i} \quad \text{for } i = 0,\ 1,\ \text{and } 2 \tag{2.9}$$

$$\dot{\bar{\varepsilon}}_3^{\text{ss}} = H(\bar{\sigma} - \bar{\sigma}_{\text{g}}) \sum_{i=0}^{2} B_i \exp\left(-\frac{Q_i}{RT}\right) \sinh\left(q \frac{(\bar{\sigma} - \bar{\sigma}_{\text{g}})}{\mu}\right), \tag{2.10}$$

where the variables A_i, B_i, Q_i, n_i, $\bar{\sigma}_{\text{g}}$, and q are all model parameters. All four mechanisms have an Arrhenius temperature dependence, where Q_i is an activation energy and $R = 8.314$ J/(K mol) is the universal gas constant. An actual micro-mechanical mechanism for the creep behavior at low equivalent stresses has not been identified yet, so the new mechanism, Mechanism 0, is simply given the same mathematical form as Mechanisms 1 and 2. Mechanism 1 is meant to capture dislocation climb, which dominates at high temperatures and low equivalent stresses. Mechanism 2 dominates at low temperatures and medium equivalent stresses. The micro-mechanical cause for mechanism 2 is also unknown, but cross-slip has been recently suggested (Hansen 2014). Regardless, the macroscopic behavior corresponding to the second mechanism has been well characterized. Mechanism 3 models dislocation glide, which is only activated when $\bar{\sigma}$ exceeds $\bar{\sigma}_{\text{g}}$, as reflected in the heaviside function $H(\bar{\sigma} - \bar{\sigma}_{\text{g}})$. Typically, the parameters B_j are chosen to produce a smooth transition to mechanism 3 at $\bar{\sigma}_{\text{g}}$.

The simple functional forms of Eqs. (2.9) and (2.10) suffice for the steady-state behavior, but the transient behavior is somewhat more complex. During work hardening under constant stress, $\bar{\varepsilon}^{\text{tr}}$ approaches the transient equivalent strain limit $\bar{\varepsilon}^{\text{tr}^*}$ from below, and the total viscoplastic strain rate slows down over time. During recovery under constant stress, $\bar{\varepsilon}^{\text{tr}}$ approaches $\bar{\varepsilon}^{\text{tr}^*}$ from above, and the total viscoplastic strain rate speeds up over time. The rate that $\bar{\varepsilon}^{\text{tr}}$ approaches $\bar{\varepsilon}^{\text{tr}^*}$ is governed by

$$\dot{\bar{\varepsilon}}^{\text{tr}} = (F-1)\,\dot{\bar{\varepsilon}}^{\text{ss}}, \tag{2.11}$$

where

$$F = \exp\left[\text{sign}\left(\bar{\varepsilon}^{\text{tr}^*} - \bar{\varepsilon}^{\text{tr}}\right)\kappa\left(1 - \frac{\bar{\varepsilon}^{\text{tr}}}{\bar{\varepsilon}^{\text{tr}^*}}\right)^2\right]. \tag{2.12}$$

The quantitiy κ depends on whether the material is work hardening or recovering:

$$\kappa = \begin{cases} \alpha_{\text{h}} + \beta_{\text{h}}\,\log_{10}\left(\dfrac{\bar{\sigma}}{\mu}\right) & \text{for} \quad \bar{\varepsilon}^{\text{tr}} \leq \bar{\varepsilon}^{\text{tr}^*} \\ \alpha_{\text{r}} + \beta_{\text{r}}\,\log_{10}\left(\dfrac{\bar{\sigma}}{\mu}\right) & \text{for} \quad \bar{\varepsilon}^{\text{tr}} > \bar{\varepsilon}^{\text{tr}^*}, \end{cases} \tag{2.13}$$

where α_j and β_j are model parameters. Note that the parameter κ must be non-negative, otherwise Eq. (2.11) produces a negative/positive $\dot{\bar{\varepsilon}}^{\text{tr}}$ when $\bar{\varepsilon}^{\text{tr}}$ is below/above $\bar{\varepsilon}^{\text{tr}^*}$. (Such behavior occurs during reverse creep, but the MD model is only designed to model forward creep (Munson and Dawson 1982).) To enforce this, Eq. (2.13) is calculated first, and then

$$\kappa \leftarrow \max(\kappa, 0) \tag{2.14}$$

is applied.

The legacy MD model used a single mechanism to endow $\bar{\varepsilon}^{\text{tr}^*}$ with stress and temperature dependence. Reedlunn (2016), however, analyzed the data from Salzer et al. (2015) and Düsterloh et al. (2015) and discovered larger $\bar{\varepsilon}^{\text{tr}^*}$ values for $\bar{\sigma} < 8$ MPa than would be expected from extrapolating $\bar{\varepsilon}^{\text{tr}^*}$ values from higher stresses. Consequently, a second mechanism is added to capture the transient creep at low equivalent stress

$$\bar{\varepsilon}^{\text{tr}^*} = \sum_{i=0}^{1} \bar{\varepsilon}_i^{\text{tr}^*} \tag{2.15}$$

$$\bar{\varepsilon}_i^{\text{tr}^*} = K_i\,\exp(c_i\,T)\left(\frac{\bar{\sigma}}{\mu}\right)^{m_i} \tag{2.16}$$

where K_i, c_i, and m_i are parameters to be calibrated against experimental results. As before, the new transient strain limit mechanism, Mechanism 0, is given the same mathematical form as the original mechanism. This may be revisited if new information comes to light.

To summarize, the changes to the legacy MD model formulation are:

1. The equivalent stress (and flow potential) was generalized from Tresca to Hosford in Eq. (2.5)

2. Steady-state mechanism 0 was added in Eq. (2.8)
3. Transient strain limit mechanism 0 was added in Eq. (2.15).

2.2 A Simple Analysis of a Triaxial Compression Creep Test

This section analyzes a triaxial compression creep test to make $\dot{\bar{\varepsilon}}^{\text{ss}}$ and $\bar{\varepsilon}^{\text{tr}}$ more concrete. The purpose of a triaxial creep test is to apply a known stress difference (a shear stress upon coordinate transformation) and monitor the amount of creep strain.

Figure 2.1 depicts a triaxial creep specimen with a length to diameter ratio of $L/D = 2$ and results from a creep experiment. Specimens are placed in a triaxial cell in a specially outfitted load frame that allows the test operator to independently control the test temperature T, the axial Cauchy stress σ_{zz}, and the radial Cauchy stress σ_{rr}, while monitoring the axial strain ε_{zz}. The hoop stress $\sigma_{\theta\theta}$ is equal to σ_{rr}. Axisymmetric compression would perhaps be more appropriate name for this stress state, but triaxial compression is the common name. Usually, the axial log strain (positive in compression) is calculated from the axial compressive displacement of the platens u as $\varepsilon_{\text{zz}} = \ln(1 + u/L)$. Although triaxial compression creep specimens can barrel outwards due to friction at the top and bottom platens, the deformation and stresses are assumed to be spatially uniform herein.

The following sequence occurs in Fig. 2.1b. First, the temperature is raised to the test temperature and the hydrostatic pressure is raised to $\sigma_{\text{rr}} = \sigma_{\theta\theta} = \sigma_{\text{zz}} = 20$ MPa, causing a strain $\varepsilon_{\text{zz}}(t_{-1})$. At $t = t_0 = 0$, the axial stress is quickly raised to $\sigma_{\text{zz}} = 32$ MPa, changing $\bar{\sigma}$ from 0 to 12 MPa. This causes a rapid increase in the axial strain ε_{zz}. As the stress difference $\bar{\sigma}$ is held fixed for the next 53 days, the axial strain rate $\dot{\varepsilon}_{\text{zz}}$ slows down and eventually approaches a steady state rate.

The quantities $\dot{\bar{\varepsilon}}^{\text{ss}}$ and $\bar{\varepsilon}^{\text{tr}}$ can now be identified. For triaxial compression, one can combine Eqs. (2.5) and (2.6), reduce the result to

$$\begin{bmatrix} \dot{\varepsilon}^{\text{vp}}_{\text{rr}} & \dot{\varepsilon}^{\text{vp}}_{\text{r}\theta} & \dot{\varepsilon}^{\text{vp}}_{\text{rz}} \\ \dot{\varepsilon}^{\text{vp}}_{\theta\text{r}} & \dot{\varepsilon}^{\text{vp}}_{\theta\theta} & \dot{\varepsilon}^{\text{vp}}_{\theta\text{z}} \\ \dot{\varepsilon}^{\text{vp}}_{\text{zr}} & \dot{\varepsilon}^{\text{vp}}_{\text{z}\theta} & \dot{\varepsilon}^{\text{vp}}_{\text{zz}} \end{bmatrix} = \dot{\bar{\varepsilon}}^{\text{vp}} \begin{bmatrix} -1/2 & 0 & 0 \\ 0 & -1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.17}$$

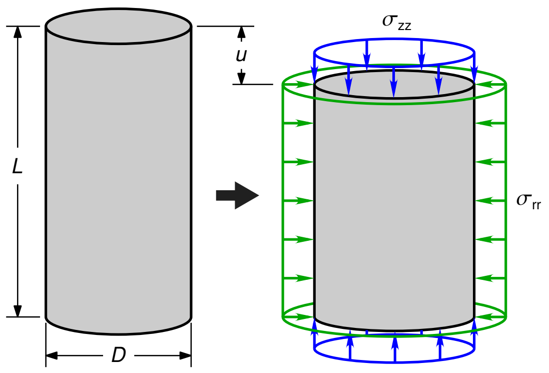
combine with Eqs. (2.1) and (2.7), and isolate the axial direction to obtain

$$\dot{\varepsilon}_{\text{zz}} = \dot{\varepsilon}^{\text{el}}_{\text{zz}} + \dot{\varepsilon}^{\text{th}}_{\text{zz}} + \dot{\bar{\varepsilon}}^{\text{tr}} + \dot{\bar{\varepsilon}}^{\text{ss}}. \tag{2.18}$$

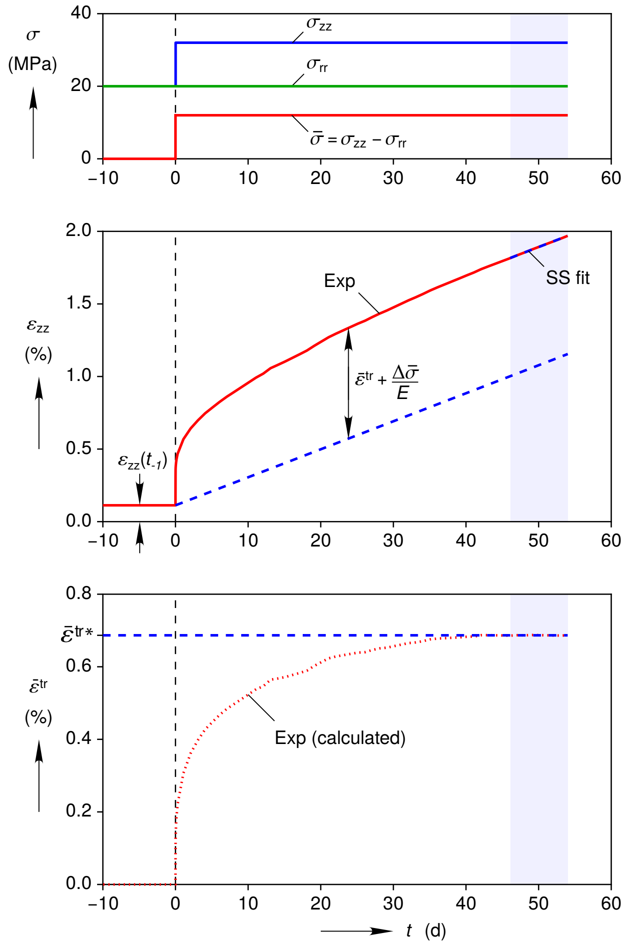
For $t > t_0$ the thermal and elastic strain rates are zero ($\dot{\varepsilon}^{\text{th}}_{\text{zz}} = \dot{\varepsilon}^{\text{el}}_{\text{zz}} = 0$), which simplifies Eq. (2.18) further to

$$\dot{\varepsilon}_{\text{zz}} = \dot{\bar{\varepsilon}}^{\text{tr}} + \dot{\bar{\varepsilon}}^{\text{ss}}. \tag{2.19}$$

By the end of the experiment, Eq. (2.19) becomes $\dot{\varepsilon}_{\text{zz}} = \dot{\bar{\varepsilon}}^{\text{ss}}$, because the transient strain rate is zero. In other words, $\dot{\bar{\varepsilon}}^{\text{ss}}$ is the slope of the "SS fit" line in Fig. 2.1b. Let t_i^- be the instant



(a) Triaxial compression schematic



(b) Analysis of triaxial compression results

Figure 2.1: A simple triaxial creep experiment

in time immediately before t_i, and t_i^+ be the instant in time immediately after t_i. To find $\bar{\varepsilon}^{\mathrm{tr}}(t)$, integrate Eq. (2.19) from the initial time t_0^+ to the current time t, and rearrange to obtain

$$\bar{\varepsilon}^{\mathrm{tr}}(t) - \bar{\varepsilon}^{\mathrm{tr}}(t_0^+) = \left[\varepsilon_{\mathrm{zz}}(t) - \varepsilon_{\mathrm{zz}}(t_0^+)\right] - \left[\bar{\varepsilon}^{\mathrm{ss}}(t) - \bar{\varepsilon}^{\mathrm{ss}}(t_0^+)\right]. \tag{2.20}$$

The initial total strain at t_0^+ can be related back to the total strain at t_0^- as

$$\varepsilon_{\mathrm{zz}}(t_0^+) = \varepsilon_{\mathrm{zz}}(t_0^-) + \frac{\Delta\bar{\sigma}(t_0)}{E}, \tag{2.21}$$

where $\Delta\bar{\sigma}(t_0)$ is the change in equivalent stress at t_0 and $E = 9\,B\,\mu/(3\,B + \mu)$ is Young's modulus. The total strain $\varepsilon_{\mathrm{zz}}(t_0^-)$ is

$$\varepsilon_{\mathrm{zz}}(t_0^-) = \varepsilon_{\mathrm{zz}}(t_{-1}) = -\alpha\,(T - T_0) + \frac{\sigma_{\mathrm{zz}}(t_{-1})}{3\,B} + \varepsilon_{\mathrm{zz}}(t_{-1}^-), \tag{2.22}$$

where T_0 is a reference temperature with zero thermal strain and $\varepsilon_{\mathrm{zz}}(t_{-1}^-)$ is the axial strain prior to the test. The viscoplastic strain cannot immediately respond to a jump in stress, so $\bar{\varepsilon}^{\rm tr}(t_0^+) = \bar{\varepsilon}^{\rm tr}(t_0^-)$ and $\bar{\varepsilon}^{\rm ss}(t_0^+) = \bar{\varepsilon}^{\rm ss}(t_0^-)$. Once $\bar{\sigma} > 0$, the change in steady-state equivalent viscoplastic strain is

$$\bar{\varepsilon}^{\rm ss}(t) - \bar{\varepsilon}^{\rm ss}(t_0^-) = \dot{\bar{\varepsilon}}^{\rm ss}\,\left(t - t_0^-\right), \tag{2.23}$$

because $\bar{\sigma}$ and T remain constant for $t > t_0$. Plugging these into Eq. (2.20) gives,

$$\bar{\varepsilon}^{\rm tr}(t) - \bar{\varepsilon}^{\rm tr}(t_0^-) = \left[\varepsilon_{\rm zz}(t) - \varepsilon_{\rm zz}(t_0^-)\right] - \frac{\Delta\bar{\sigma}(t_0)}{E} - \dot{\bar{\varepsilon}}^{\rm ss}\,\left(t - t_0^-\right). \tag{2.24}$$

Typically, the specimen is *assumed* to be initially virgin, such that $\bar{\varepsilon}^{\rm tr}(t_0^-) = \varepsilon_{\rm zz}(t_{-1}^-) = 0$. Equation Eq. (2.24) with $\bar{\varepsilon}^{\rm tr}(t_0^-) = 0$ is shown schematically in the $\varepsilon_{\rm zz}$ vs. t plot in Fig. 2.1b. The resulting $\bar{\varepsilon}^{\rm tr}$ vs. t curve is shown in the plot below. As one might expect, the transient strain reaches a limiting value, labeled $\bar{\varepsilon}^{\rm tr^*}$, once the creep curve reaches steady state.

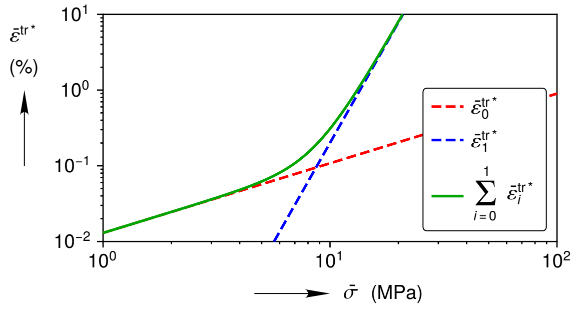
The example in Fig. 2.1 only includes one step in $\bar{\sigma}$, but experimentalists often shift $\bar{\sigma}$ during a triaxial experiment. (See Fig. 3.4 in Reedlunn (2016) for examples.) Measuring $\dot{\bar{\varepsilon}}^{\rm ss}$ after changing $\bar{\sigma}$ at an arbitrary time t_i is still just the slope of the $\varepsilon_{\rm zz}$ curve after the transient response has completed. Calculating $\bar{\varepsilon}^{\rm tr}(t)$ simply requires replacing t_0 with t_i in Eq. (2.24):

$$\bar{\varepsilon}^{\rm tr}(t) - \bar{\varepsilon}^{\rm tr}(t_i^-) = \left[\varepsilon_{\rm zz}(t) - \varepsilon_{\rm zz}(t_i^-)\right] - \frac{\Delta\bar{\sigma}(t_i)}{E} - \dot{\bar{\varepsilon}}^{\rm ss}\,\left(t - t_i^-\right). \tag{2.25}$$

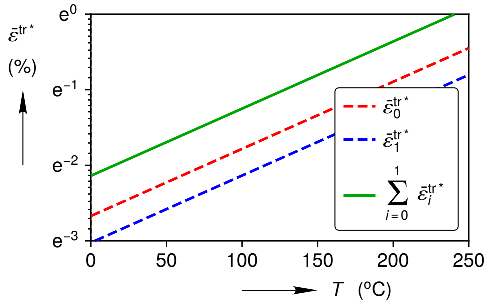
Of course, $\bar{\varepsilon}^{\rm tr}(t_i^-)$ is typically non-zero when $t_i \neq 0$.

2.3 Temperature and Stress Dependence

In an effort to make the equations that govern the transient strain limit and steady-state strain rate more familiar, they have been plotted in Figs. 2.2 and 2.3 for Calibration 2B. (The process used to generate Calibration 2B is discussed in Chapter 5.) These plots can be understood by taking logarithms of the individual transient strain limit and steady-state strain rate mechanisms.



(a) Stress Dependence at $T = 27$ °C



(b) Temperature Dependence at $\bar{\sigma} = 8$ MPa

Figure 2.2: Stress and temperature dependence of the transient strain limit for Calibration 2B.

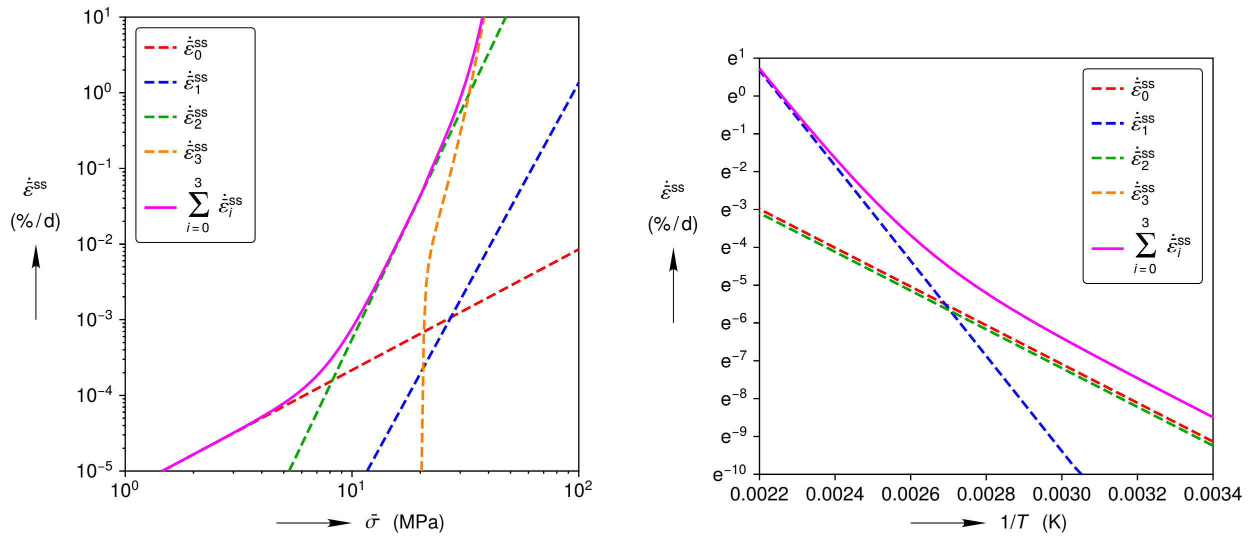
Starting with the transient strain limit, one can apply a base 10 logarithm to Eq. (2.16) to obtain

$$\log_{10} \bar{\varepsilon}_i^{\mathrm{tr}*} = \log_{10}\left[\frac{K_i}{\mu_i^m} \exp\left[c_i\, T\right]\right] + m_i \log_{10} \bar{\sigma}. \tag{2.26}$$

Each $\bar{\varepsilon}_i^{\mathrm{tr}*}$ has a linear dependence on $\bar{\sigma}$ in log-log space, with a slope of m_i. At low $\bar{\sigma}$, mechanism 0 dominates ($\bar{\varepsilon}^{\mathrm{tr}*} \approx \bar{\varepsilon}_0^{\mathrm{tr}*}$), as shown in Fig. 2.2a. At high $\bar{\sigma}$, mechanism 1 dominates ($\bar{\varepsilon}^{\mathrm{tr}*} \approx \bar{\varepsilon}_1^{\mathrm{tr}*}$). Both mechanisms contribute at intermediate $\bar{\sigma}$, leading to a transition zone in-between $\bar{\varepsilon}_0^{\mathrm{tr}*}$ and $\bar{\varepsilon}_1^{\mathrm{tr}*}$. Taking a natural logarithm of Eq. (2.16) results in

$$\ln \bar{\varepsilon}_i^{\mathrm{tr}*} = \ln\left[K_i \left(\frac{\bar{\sigma}}{\mu}\right)^{m_i}\right] + c_i\, T. \tag{2.27}$$

Each $\bar{\varepsilon}_i^{\mathrm{tr}*}$ has a linear dependence on T in natural log-linear space, with a slope of c_i. In Calibration 2B, $c_0 = c_i$, so both mechanisms have the same slope in Fig. 2.2b. To summarize, m_i controls the stress dependence, c_i controls the temperature dependence, and K_i shifts the height of the line for mechanism i.



(a) Stress Dependence at $T = 27$ °C

(b) Temperature Dependence at $\bar{\sigma} = 8$ MPa

Figure 2.3: Stress and temperature dependence of the steady-state strain rate for Calibration 2B.

A similar analysis applies to the steady-state strain rate. One can take the base 10 logarithm of Eq. (2.9) to obtain

$$\log_{10} \dot{\bar{\varepsilon}}_i^{\mathrm{ss}} = \log_{10}\left[\frac{A_i}{\mu^{n_i}} \exp\left(-\frac{Q_i}{R\,T}\right)\right] + n_i \log_{10} \bar{\sigma}. \tag{2.28}$$

Mechanisms 0, 1, and 2 each have a linear dependence on $\bar{\sigma}$ in log-log space, each with a slope of n_i. In Fig. 2.3a, mechanism 0 dominates at low stresses, mechanism 1 does not contribute due to the low temperature, and mechanism 2 dominates at medium stresses. Mechanism 3 is more complex due to the hyperbolic sine and heaviside function in Eq. (2.10), but it dominates at high stresses. Taking the natural logarithm of both sides of Eq. (2.9) results in

$$\ln \dot{\bar{\varepsilon}}_i^{\mathrm{ss}} = \ln \left[A_i \left(\frac{\bar{\sigma}}{\mu} \right)^{n_i} \right] - \frac{Q_i}{RT}. \tag{2.29}$$

Mechanisms 0, 1, and 2 each have a linear dependence on $1/T$ in natural log-linear space, each with a slope of Q_i/R. In Fig. 2.3b, mechanism 1 dominates at high temperatures, while mechanism 0 and 2 both contribute at low temperatures. Mechanism 0 and 2 have the same slope because $Q_0 = Q_2$. Mechanism 3 does not contribute due to the low stress. In summary, n_i controls the stress dependence, Q_i controls the temperature dependence, and A_i shifts the height of the line for mechanism $i = 0$, 1, or 2.

2.4 Finite Strain Formulation

Hypoelasticity allows one to extend infinitesimal strain constitutive models into the finite deformation realm. One typically converts an infinitesimal strain model into a hypoelastic model by replacing the strain rates with the corresponding (spatial) rates of deformation and replacing the stress rate with an objective rate of the Cauchy or Kirchhoff stress. These substitutions allow one to evaluate a model in the spatial configuration and still ensure that the rates of spatial strains and stresses transform properly under rigid body rotations.

Although most hypoelastic frameworks evaluate constitutive models in the spatial configuration, Sierra/Solid Mechanics, evaluates hypoelastic models in the unrotated configuration. The rate of deformation $\boldsymbol{D}$ and the Cauchy stress $\boldsymbol{\Sigma}$ are respectively pulled back to the unrotated configuration as

$$\boldsymbol{d} = \boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{D} \cdot \boldsymbol{R}, \tag{2.30}$$

$$\boldsymbol{\sigma} = \boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{R}, \tag{2.31}$$

where $\boldsymbol{R}$ is the rigid-body rotation from the polar decomposition of the deformation gradient. The infinitesimal strain MD model in Section 2.1 is reformulated in the unrotated configuration by swapping out the strain rates with the corresponding unrotated rates of deformation and replacing the stress rate with the unrotated rate of the Cauchy stress, such that Eqs. (2.1), (2.2), (2.4) and (2.6) respectively become

$$\boldsymbol{d} = \boldsymbol{d}^{\mathrm{el}} + \boldsymbol{d}^{\mathrm{th}} + \boldsymbol{d}^{\mathrm{vp}}, \tag{2.32}$$

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{\mathcal{C}} : \boldsymbol{d}^{\mathrm{el}}, \tag{2.33}$$

$$\boldsymbol{d}^{\mathrm{th}} = -\alpha \dot{T} \boldsymbol{I}, \tag{2.34}$$

$$\boldsymbol{d}^{\mathrm{vp}} = \dot{\bar{\varepsilon}}^{\mathrm{vp}} \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}. \tag{2.35}$$

The other constitutive equations in Section 2.1 remain unchanged. Sierra/Solid Mechanics's objective stress rate is the Green-McInnis rate of the Cauchy stress, which is defined as

$$\overset{\circ}{\boldsymbol{\Sigma}} = \boldsymbol{R} \cdot \dot{\boldsymbol{\sigma}} \cdot \boldsymbol{R}^{\mathrm{T}}. \tag{2.36}$$

The following procedure is used to calculate $\overset{\circ}{\boldsymbol{\Sigma}}$:

1. Calculate $\boldsymbol{d}$ using Eq. (2.30).
2. Evaluate the constitutive model in the unrotated configuration to obtain $\dot{\boldsymbol{\sigma}}$.
3. Calculate $\overset{\circ}{\boldsymbol{\Sigma}}$ using Eq. (2.36).

With the procedure to compute $\overset{\circ}{\boldsymbol{\Sigma}}$ defined, this section closes with some important remarks:

1. Anisotropic material models are the primary reason to evaluate constitutive models in the unrotated configuration instead of the spatial configuration. On the other hand, if one is evaluating a isotropic model, such as the MD model, then the forgoing procedure is actually equivalent to evaluating in the spatial configuration. To demonstrate, substitute Eq. (2.36) and the elastic version of Eq. (2.30) into Eq. (2.33):

   $$\boldsymbol{R}^{\mathrm{T}} \cdot \overset{\circ}{\boldsymbol{\Sigma}} \cdot \boldsymbol{R} = \boldsymbol{\mathcal{C}} : \left(\boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{D}^{\mathrm{el}} \cdot \boldsymbol{R}\right). \tag{2.37}$$

   Now, reorganize the rotation tensors and recall that an isotropic stiffness tensor is invariant to rotations, i.e.

   $$\mathcal{C}_{ijkl} = R_{im}\, R_{jn}\, \mathcal{C}_{mnop}\, R_{ko}\, R_{lp}, \tag{2.38}$$

   to obtain

   $$\overset{\circ}{\boldsymbol{\Sigma}} = \boldsymbol{\mathcal{C}} : \boldsymbol{D}^{\mathrm{el}}. \tag{2.39}$$

   Thus, it is not necessary to appeal to the unrotated configuration when dealing with isotropic materials. That said, Sierra/Solid Mechanics evaluates all hypoelastic material models in the unrotated configuration, so the numerical implementation in this report will also evaluate the MD model in the unrotated configuration.

2. One uses the aforementioned procedure to calculate $\overset{\circ}{\boldsymbol{\Sigma}}$ for general deformations. For principal deformations, however, the procedure simplifies considerably because $\overset{\circ}{\boldsymbol{\Sigma}} = \dot{\boldsymbol{\Sigma}} = \dot{\boldsymbol{\sigma}}$ and $\boldsymbol{D} = \boldsymbol{d} = \dot{\boldsymbol{\varepsilon}}$, where $\dot{\boldsymbol{\varepsilon}}$ is the logarithmic strain rate.

3. The hypoelastic model discussed herein is not derivable from a conservative potential (i.e. a hyperelastic model), which means elastic deformation is path dependent and dissipates energy. Hypoelastic elastic dissipated energy, however, is typically very small compared to (visco)plastic dissipated energy when modeling plastic deformation of crystalline materials. Brepols et al. (2014) verified this by comparing three hypoelastic-plastic models, each with a different stress rate, against a hyperelastic-plastic model in metal forming simulations. All four models produced nearly identical results when the elastic strains were negligible compared to the plastic strains.

4. Sierra/Solid Mechanics pairs up $\boldsymbol{\Sigma}$ and $\boldsymbol{D}$ in its hypoelastic constitutive models, yet the Kirchoff stress tensor $\boldsymbol{\tau}$ and $\boldsymbol{D}$ are the proper conjugate pair (i.e. $\boldsymbol{\tau} : \boldsymbol{D}$ equals the mechanical work power per unit reference volume). The Kirchoff stress tensor is defined as $\boldsymbol{\tau} = J\,\boldsymbol{\Sigma}$, where J is the ratio of the current volume over the reference volume of a material element. Fortunately, the error in utilizing $\boldsymbol{\Sigma}$ instead of $\boldsymbol{\tau}$ is usually negligible because, again, elastic strains are typically very small compared to plastic strains and plasticity is isochoric in the MD model, such that $J \approx 1$.

Chapter 3

Numerical Implementation

This chapter describes a new MD model implementation that includes the new features discussed in Section 2.1. The MD model has been previously implemented thrice before in Sierra/Solid Mechanics, but the present author chose to add the new features to a new model implementation rather than modify an old implementation. The three previous numerical implementations of the MD model in Sierra/SM are relatively slow, and they are written in the FORTRAN computer programming language, which is being depreciated from Sierra/Solid Mechanics. The new implementation, called `md_viscoplastic`, has an improved algorithm to integrate the ordinary differential equations and is written in the C++ computer programming language.

The three previous MD model implementations are called `munson_dawson`, `md_creep`, and `implicit_wipp_crushed_salt`. The `munson_dawson` implementation (Weatherby et al. 1996) integrates the ordinary differential equations explicitly using the forward Euler method. The implementation uses the unrotated rate of deformation $\boldsymbol{d}_n$ at time step n and the time step size $\Delta t_n = t_n - t_{n-1}$ from Sierra/Solid Mechanics to calculate a critical time step size Δt_n^{crit}. If $\Delta t_n \leq \Delta t_n^{\text{crit}}$, then explicit integration can proceed without numerical instabilities. If $\Delta t_n > \Delta t_n^{\text{crit}}$, then the implementation sub-divides the time step into smaller time steps equal to or less than Δt_n^{crit} so it can explicitly integrate through the sub-time steps. This approach is acceptable if the deformation increment is sufficiently small, but Sierra/Solid Mechanics can infrequently supply a large rate of deformation to a material point. If the latter happens, the `munson_dawson` implementation will compute an extremely small Δt_n^{crit} relative to Δt_n, resulting in excessively long computation times. Simulations are known to "halt" for hours while the problematic material point integrates the constitutive equations. The `md_creep` implementation was designed to avoid the extremely small critical time step issue by using an implicit, backward Euler, scheme to integrate its constitutive equations. Backward Euler algorithms can handle arbitrarily large Δt_n without numerical oscillations or instabilities, making them well suited to simulations that extend for many years. The `implicit_wipp_crushed_salt` implementation is an implementation of the Callahan model for crushed salt (Callahan 1999), but the Callahan model reduces down to the MD model when the crushed salt density matches that of intact salt. Accordingly, the `implicit_wipp_crushed_salt` implementation also reduces down to the `md_creep` implementation when the crushed salt density matches that of intact salt. The `md_creep` and `implicit_wipp_crushed_salt` implementations were never formally documented due to funding constraints. In practice, the `md_creep` and `implicit_wipp_crushed_salt` implementations often finished simulations faster than the `munson_dawson` implementation, but the implicit algorithm frequently struggled to converge. These convergence issues caused numerous time step cut backs, ultimately slowing down simulations.

The task of creating a faster MD model implementation was part of the motivation for changing from the Tresca to the Hosford equivalent stress. Scherzinger (2017) showed that a line search algorithm, in conjunction with Newton's method, significantly improved the convergence of a plasticity model with a Hosford yield surface. The line search stopped the Newton iterations from bouncing back and forth between facets of Hosford's rounded hexagon yield surface. One could potentially pair up a line search algorithm with the Tresca equivalent stress in the MD model, but it would likely be more complicated than combining a line search with the Hosford equivalent stress and one would not have the added benefit of fitting the Hosford exponent a to better match experimental measurements.

This chapter discusses how Scherzinger's approach was applied to the MD model, which is a rate-dependent viscoplastic model, instead of the rate-independent model utilized by Scherzinger (2017). The development herein also explains some topics further and corrects some errors in Scherzinger (2017). The chapter begins by showing how to numerically calculate the Hosford equivalent stress (Section 3.1), a slight generalization of the transient strain differential equation (Section 3.2), and the addition of a viscoplastic scale factor (Section 3.3). This is followed by discretizing the differential equations in time (Section 3.4), the formulation of Newton's method (Section 3.6), how one modifies Newton's method to include a line search phase (Section 3.7), the calculation of the material tangent (Section 3.8), and a summary of the `md_viscoplastic` implementation.

3.1 Equivalent Stress

Calculation of the Hosford equivalent stress requires the principal values of the stress tensor. The `md_viscoplastic` implementation utilizes the algorithm presented in Scherzinger and Dohrmann (2008) to quickly and accurately compute the principal stresses σ_i and principal stress directions $\tilde{\boldsymbol{e}}_i$.

After calculating σ_i accurately, additional care must be taken when calculating the Hosford equivalent stress if a is large. If a principal stress difference in Eq. (2.5) is not close to unity, a large value of a will lead to numerical overflow or underflow errors. Following Scherzinger's lead, this issue is mitigated by normalizing the principal stresses by a factor σ_n to make the principal stresses close to one. Let $\check{\sigma}_i = \sigma_i/\sigma_\text{n}$ and substitute into Eq. (2.5) to obtain

$$\bar{\sigma} = \sigma_\text{n} \left\{ \frac{1}{2} \left[|\check{\sigma}_1 - \check{\sigma}_2|^a + |\check{\sigma}_2 - \check{\sigma}_3|^a + |\check{\sigma}_1 - \check{\sigma}_3|^a \right] \right\}^{1/a}, \tag{3.1}$$

The von Mises stress $\bar{\sigma}_\text{vm}$ would be a suitable normalization factor, except it can be zero.

The von Mises stress is defined as

$$\bar{\sigma}_{\text{vm}} = \sqrt{\frac{3}{2}\boldsymbol{\sigma}^{\text{dev}} : \boldsymbol{\sigma}^{\text{dev}}} \tag{3.2}$$

where $\boldsymbol{\sigma}^{\text{dev}} = \boldsymbol{\sigma} - 1/3\,\text{tr}(\boldsymbol{\sigma})\,\boldsymbol{I}$ is the deviatoric stress. To avoid dividing by zero when calculating $\breve{\sigma}_i$, the normalization factor is defined as

$$\sigma_{\text{n}} = \bar{\sigma}_{\text{vm}} + \bar{\sigma}_{\text{min}}, \tag{3.3}$$

where $\bar{\sigma}_{\text{min}}$ is a positive value, small enough to ensure $\breve{\sigma}_i$ remain close to one. The default is $\bar{\sigma}_{\text{min}} = \mu \times 10^{-10}$. If $\mu = 12.4$ GPa (a typical value for rock salt), then $\bar{\sigma}_{\text{min}} = 1.24$ Pa.

Another division by zero issue arises in calculating the first and second derivatives of the flow potential $\bar{\sigma}$. As discussed in Appendix B, the first and second derivatives are computed using normalized principal stresses $\hat{\sigma}_i = \sigma_i/\bar{\sigma}$. This normalization runs into trouble if $\bar{\sigma} = 0$, so $\bar{\sigma}$ is calculated using Eq. (3.1) first, and then

$$\bar{\sigma} \leftarrow \max\left(\bar{\sigma},\ \bar{\sigma}_{\text{min}}\right) \tag{3.4}$$

is applied. Of course, $\bar{\sigma}_{\text{min}}$ must be small enough induce negligible creep.

Note that $\bar{\sigma}_{\text{min}} = 0$ in Scherzinger (2017) because he implemented a rate-independent plasticity model. If $\bar{\sigma}$ is less than the yield stress in a rate-independent model, plastic flow is categorically prohibited and the Jacobian and Hessian of the flow potential do not need to be calculated.

3.2 Transient Strain Ordinary Differential Equation

One can obtain an analytical solution to the MD model's ordinary differential equations if the exponent in Eq. (2.12) is changed from 2 to 1. In fact, the original definition of F in Munson and Dawson (1982) was

$$F = \exp\left[\kappa\left(1 - \frac{\bar{\varepsilon}^{\text{tr}}}{\bar{\varepsilon}^{\text{tr}*}}\right)\right] \tag{3.5}$$

before it was changed to Eq. (2.12) in Munson et al. (1989). To accommodate both possibilties, Eq. (2.12) is numerically implemented in `md_viscoplastic` as

$$F = \exp\left[\text{sign}\left(\bar{\varepsilon}^{\text{tr}*} - \bar{\varepsilon}^{\text{tr}}\right)^{\chi-1}\kappa\left(1 - \frac{\bar{\varepsilon}^{\text{tr}}}{\bar{\varepsilon}^{\text{tr}*}}\right)^{\chi}\right] \tag{3.6}$$

where χ is a user specified integer that is equal to 2 by default, but one can set $\chi = 1$ for verification testing. See Section 4.1 for further discussion of the analytical solution when $\chi = 1$.

3.3 Viscoplastic Rate Scale Factor

Each steady state creep mechanism is implemented in `md_viscoplastic` with a viscoplastic rate scale factor s, such that Eqs. (2.8) to (2.10) become

$$\begin{aligned}
\dot{\bar{\varepsilon}}^{\text{ss}} =& \sum_{i=0}^{3} \dot{\bar{\varepsilon}}_i^{\text{ss}} \\
\dot{\bar{\varepsilon}}_i^{\text{ss}} =& s\, A_i \, \exp\left(-\frac{Q_i}{R\,T}\right) \left(\frac{\bar{\sigma}}{\mu}\right)^{n_i} \qquad \text{for } i = 0,\ 1, \text{ and } 2 \\
\dot{\bar{\varepsilon}}_3^{\text{ss}} =& s\, H(\bar{\sigma} - \bar{\sigma}_{\text{g}}) \sum_{i=0}^{2} B_i \, \exp\left(-\frac{Q_i}{R\,T}\right) \sinh\left(q \frac{(\bar{\sigma} - \bar{\sigma}_{\text{g}})}{\mu}\right),
\end{aligned} \tag{3.7}$$

This scale factor can be used to speed up or slow down the equivalent steady-state strain rate and the equivalent transient strain rate because $\dot{\bar{\varepsilon}}^{\text{tr}} = (F - 1)\, \dot{\bar{\varepsilon}}^{\text{ss}}$. The default value is $s = 1$, but it can be useful to set s to some small value to "freeze" the material's viscoplasticity for a period of time, or increase s to larger values to squeeze hundreds of years into a few seconds. Speeding up the viscoplasticity can allow one to make quasi-static simulations using explicit dynamics, provided inertial effects are kept to a minimum.

The variable s is implemented as an internal state variable, rather than a material parameter, so a user can modify it in the middle of a simulation. In Sierra/Solid Mechanics, internal state variables can be altered by creating a "user variable" with the same name as the internal state variable (`viscoplastic_rate_scale_factor` in this case) in an input deck and modifying the user variable with a user function or user subroutine.

3.4 Time Discretization

The `md_viscoplastic` implementation approximates the MD model's time derivatives using the backwards Euler method. The following notation is introduced to handle the time discretization. If x is a generic variable, x_n is the value of x at the current time step n, x_{n-1} is the value of x at the previous time step, and $\Delta x_n = x_n - x_{n-1}$. Using this notation, time derivatives are approximated as $\dot{x} \approx \Delta x_n / \Delta t_n$. In addition, the following notation

$$\Delta \boldsymbol{\epsilon}_n = \boldsymbol{d}_n \, \Delta t_n \tag{3.8}$$

$$\Delta \boldsymbol{\epsilon}_n^{\text{th}} = \boldsymbol{d}_n^{\text{th}} \, \Delta t_n \tag{3.9}$$

$$\Delta \boldsymbol{\epsilon}_n^{\text{vp}} = \boldsymbol{d}_n^{\text{vp}} \, \Delta t_n \tag{3.10}$$

is introduced for convenience. Note that $\Delta\boldsymbol{\epsilon}$ is not a logarithmic strain increment $\Delta\boldsymbol{\varepsilon}$, in general. The two coincide only the absence of rotations.

The stress-strain-temperature relation in Eq. (2.33) is discretized as

$$\Delta \boldsymbol{\sigma}_n = \boldsymbol{\mathcal{C}} : \left(\Delta \boldsymbol{\epsilon}_n - \Delta \boldsymbol{\epsilon}_n^{\text{th}} - \Delta \boldsymbol{\epsilon}_n^{\text{vp}}\right). \tag{3.11}$$

This relation can be reorganized as

$$\boldsymbol{\sigma}_n = \boldsymbol{\sigma}_n^{\text{trial}} - \boldsymbol{\mathcal{C}} : \Delta\boldsymbol{\epsilon}_n^{\text{vp}}, \tag{3.12}$$

where $\boldsymbol{\sigma}_n^{\text{trial}} = \boldsymbol{\sigma}_{n-1} + \boldsymbol{\mathcal{C}} : \left(\Delta\boldsymbol{\epsilon}_n - \Delta\boldsymbol{\epsilon}_n^{\text{th}}\right)$ is the trial (elastic) stress and $\boldsymbol{\mathcal{C}} : \Delta\boldsymbol{\epsilon}_n^{\text{vp}}$ is the viscoplastic corrector. The trial stress is computed immediately upon entering the model, while the viscoplastic corrector reduces the trial stress if viscoplastic deformation evolves during the time step. The viscoplastic deformation increment $\Delta\boldsymbol{\epsilon}_n^{\text{vp}}$ depends on the stress at the end of the time step $\boldsymbol{\sigma}_n$, which depends on $\Delta\boldsymbol{\epsilon}_n^{\text{vp}}$. In other words, one must integrate the two coupled ordinary differential Eqs. (2.6) and (2.11) to solve for $\Delta\boldsymbol{\epsilon}_n^{\text{vp}}$ and $\boldsymbol{\sigma}_n$ simultaneously.

To solve Equations (2.6) and (2.11), they are first reformatted as the following residuals

$$\boldsymbol{\mathcal{R}} = -\boldsymbol{d}^{\text{vp}} + \left(\dot{\bar{\varepsilon}}^{\text{tr}} + \dot{\bar{\varepsilon}}^{\text{ss}}\right) \frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}} = \mathbf{0} \tag{3.13}$$

$$r = -\dot{\bar{\varepsilon}}^{\text{tr}} + (F-1)\,\dot{\bar{\varepsilon}}^{\text{ss}} = 0. \tag{3.14}$$

After discretizing in time, Eqs. (3.13) and (3.14) become

$$\boldsymbol{\mathcal{R}}_n = -\Delta\boldsymbol{\epsilon}_n^{\text{vp}} + \left(\Delta\bar{\varepsilon}_n^{\text{tr}} + \Delta t_n\,\dot{\bar{\varepsilon}}_n^{\text{ss}}\right) \frac{\partial\bar{\sigma}_n}{\partial\boldsymbol{\sigma}_n} = \mathbf{0} \tag{3.15}$$

$$r_n = -\Delta\bar{\varepsilon}_n^{\text{tr}} + (F_n - 1)\,\Delta t_n\,\dot{\bar{\varepsilon}}_n^{\text{ss}} = 0. \tag{3.16}$$

(Note, $\dot{\bar{\varepsilon}}^{\text{ss}}$ was not discretized because Eq. (3.7) is a closed form expression for $\dot{\bar{\varepsilon}}^{\text{ss}}$.) The non-linear differential equations have now been converted into a series of non-linear algebraic equations that must be solved iteratively.

3.5 Frame for Tensor Components

Although this Chapter expresses tensors in fully general direct notation, note that the vast majority of the MD model implementation operates on the principal components of tensors. The total deformation increment arrives in the model with its components in an arbitrary input frame (potentially six non-zero components) and this frame is also used to compute the trial stress, but the trial stress state is converted to the principal frame in order to evaluate the Hosford equivalent stress. The rest of the calculations are done in principal frame and then converted back to the input frame. Newton or Line Search iterations simply update the principal stresses and principal viscoplastic strains, but the principal directions do not change during the iterations. To prove this, first consider Eq. (3.15). The viscoplastic deformation increment $\Delta\boldsymbol{\epsilon}_n^{\text{vp}}$ has the same principal directions as $\boldsymbol{\sigma}_n$ because the normal to the flow potential $\partial\bar{\sigma}_n/\partial\boldsymbol{\sigma}_n$ has the same principal directions as $\boldsymbol{\sigma}_n$. Second, reorganize Eq. (3.12) as

$$\boldsymbol{\sigma}_n^{\text{trial}} = \boldsymbol{\sigma}_n + \boldsymbol{\mathcal{C}} : \Delta\boldsymbol{\epsilon}_n^{\text{vp}}. \tag{3.17}$$

Both $\boldsymbol{\sigma}_n$ and the viscoplastic corrector $\boldsymbol{\mathcal{C}} : \Delta\boldsymbol{\epsilon}_n^{\text{vp}}$ have the same principal directions because the elastic stiffness tensor is isotropic, so they have same principal directions as $\boldsymbol{\sigma}_n^{\text{trial}}$. Thus, the principal directions can only rotate during the trial stress calculation.

3.6 Newton's Method

The non-linear algebraic relations in Eqs. (3.15) and (3.16) are solved using Newton's method. To apply Newton's method, Taylor series expand $\boldsymbol{\mathcal{R}}_n$ and r_n about $\boldsymbol{\sigma}_n^{(k-1)}$ and $\bar{\varepsilon}_n^{\mathrm{tr}\,(k-1)}$,

$$\mathbf{0} = \boldsymbol{\mathcal{R}}_n^{(k-1)} + \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} : \delta\boldsymbol{\sigma}_n^{(k)} + \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \bar{\varepsilon}^{\mathrm{tr}}}\right)_n^{(k-1)} \delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}, \tag{3.18}$$

$$0 = r_n^{(k-1)} + \left(\frac{\partial r}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} : \delta\boldsymbol{\sigma}_n^{(k)} + \left(\frac{\partial r}{\partial \bar{\varepsilon}^{\mathrm{tr}}}\right)_n^{(k-1)} \delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}, \tag{3.19}$$

where k is the iteration number and $\delta x^{(k)} = x^{(k)} - x^{(k-1)}$, for a generic variable x. One could rewrite Eqs. (3.18) and (3.19) in matrix-vector form as

$$-\left\{\check{\mathcal{R}}\right\}_n^{(k-1)} = \left[\check{\mathcal{J}}\right]_n^{(k-1)} \left\{\check{y}\right\}_n^{(k)}, \tag{3.20}$$

where

$$\left\{\check{\mathcal{R}}_i\right\}_n^{(k-1)} = \left\{\begin{array}{c} \boldsymbol{\mathcal{R}} \\ r \end{array}\right\}_n^{(k-1)} \tag{3.21}$$

$$\left[\check{\mathcal{J}}_{ij}\right]_n^{(k-1)} = \begin{bmatrix} \dfrac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\sigma}} & \dfrac{\partial \boldsymbol{\mathcal{R}}}{\partial \bar{\varepsilon}^{\mathrm{tr}}} \\ \dfrac{\partial r}{\partial \boldsymbol{\sigma}} & \dfrac{\partial r}{\partial \bar{\varepsilon}^{\mathrm{tr}}} \end{bmatrix}_n^{(k-1)} \tag{3.22}$$

$$\left\{\check{y}_j\right\}_n^{(k)} = \left\{\begin{array}{c} \delta\boldsymbol{\sigma} \\ \delta\bar{\varepsilon}^{\mathrm{tr}} \end{array}\right\}_n^{(k)}, \tag{3.23}$$

and simply solve for the increments $\delta\boldsymbol{\sigma}_n^{(k)}$ and $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$, but that is more computationally intensive than necessary.

Instead of solving the 4×4 system of equations directly, Eqs. (3.18) and (3.19) are manipulated further to calculate $\delta\boldsymbol{\sigma}_n^{(k)}$ first and calculate $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ second. To do so, select derivatives in Eqs. (3.18) and (3.19) are first rewritten as

$$\mathbf{0} = \boldsymbol{\mathcal{R}}_n^{(k-1)} + \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} : \delta\boldsymbol{\sigma}_n^{(k)} + \left(\frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} \delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}, \tag{3.24}$$

$$0 = r_n^{(k-1)} + \left(\frac{\partial r}{\partial \bar{\sigma}}\,\frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} : \delta\boldsymbol{\sigma}_n^{(k)} + \left(\frac{\partial r}{\partial \bar{\varepsilon}^{\mathrm{tr}}}\right)_n^{(k-1)} \delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}, \tag{3.25}$$

Next, solve Eq. (3.25) for $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ to obtain

$$\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)} = -\left(r\,\frac{\partial \bar{\varepsilon}^{\mathrm{tr}}}{\partial r}\right)_n^{(k-1)} - \left(\frac{\partial \bar{\varepsilon}^{\mathrm{tr}}}{\partial r}\,\frac{\partial r}{\partial \bar{\sigma}}\,\frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\right)_n^{(k-1)} : \delta\boldsymbol{\sigma}_n^{(k)}. \tag{3.26}$$

This result is inserted into Eq. (3.24) so that one can solve for $\delta\boldsymbol{\sigma}_n^{(k)}$ as

$$\delta\boldsymbol{\sigma}_n^{(k)} = -\left(\boldsymbol{\mathcal{C}}^{\mathrm{t}} \cdot \hat{\boldsymbol{\mathcal{R}}}\right)_n^{(k-1)}, \tag{3.27}$$

where

$$\hat{\boldsymbol{\mathcal{R}}}_n^{(k-1)} = \left(\boldsymbol{\mathcal{R}} - r\,\frac{\partial\bar{\varepsilon}^{\mathrm{tr}}}{\partial r}\,\frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}}\right)_n^{(k-1)} \tag{3.28}$$

$$\begin{aligned}\boldsymbol{\mathcal{C}}_n^{\mathrm{t}(k-1)} &= \left(\frac{\partial\boldsymbol{\mathcal{R}}}{\partial\boldsymbol{\sigma}} - \frac{\partial\bar{\varepsilon}^{\mathrm{tr}}}{\partial r}\,\frac{\partial r}{\partial\bar{\sigma}}\,\frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}} \otimes \frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}}\right)_n^{(k-1)} \\ &= \left\{\left[\boldsymbol{\mathcal{C}}^{-1} + \left(\Delta\bar{\varepsilon}^{\mathrm{tr}} + \Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\right)\frac{\partial^2\bar{\sigma}}{\partial\boldsymbol{\sigma}^2} + \left(\Delta t\,\frac{\partial\dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial\bar{\sigma}} - \frac{\partial\bar{\varepsilon}^{\mathrm{tr}}}{\partial r}\,\frac{\partial r}{\partial\bar{\sigma}}\right)\frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}} \otimes \frac{\partial\bar{\sigma}}{\partial\boldsymbol{\sigma}}\right]^{-1}\right\}_n^{(k-1)},\end{aligned} \tag{3.29}$$

With $\delta\boldsymbol{\sigma}_n^{(k)}$ in hand, one can subsequently compute $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ using Eq. (3.26). The remaining derivatives embedded in Eqs. (3.26) and (3.27) can found in Appendix B. Thus, the approach to compute $\delta\boldsymbol{\sigma}_n^{(k)}$ followed by $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ requires one to invert a 3×3 tensor $\left(\boldsymbol{\mathcal{C}}^{\mathrm{t}}\right)_n^{-1(k-1)}$ instead of inverting the 4×4 Jacobian $\left[\check{\mathcal{J}}\right]_n^{(k-1)}$ initially considered. Furthermore, the converged version of the inverted 3×3 tensor $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$ has important physical meaning and utility, as will be described in Section 3.8. As an aside, one could alternatively solve for $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ first and $\delta\boldsymbol{\sigma}_n^{(k)}$ second, but that approach does not produce an explicit expression for $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$.

After computing the iteration increments $\delta\boldsymbol{\sigma}_n^{(k)}$ and $\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$, one updates the unknowns as

$$\boldsymbol{\sigma}_n^{(k)} = \boldsymbol{\sigma}_n^{(k-1)} + \delta\boldsymbol{\sigma}_n^{(k)}, \tag{3.30}$$

$$\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k)} = \Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k-1)} - \mathcal{C}^{-1} : \delta\boldsymbol{\sigma}_n^{(k)}, \tag{3.31}$$

$$\bar{\varepsilon}_n^{\mathrm{tr}\,(k)} = \bar{\varepsilon}_n^{\mathrm{tr}\,(k-1)} + \delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}. \tag{3.32}$$

The updated values of $\boldsymbol{\sigma}_n^{(k)}$, $\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k)}$, and $\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$ are then used to compute $\boldsymbol{\mathcal{R}}_n^{(k)}$ and $r_n^{(k)}$, which are used to assess convergence.

Convergence is checked by comparing the following merit function

$$\omega^{(k)} = \frac{1}{2}\left(\boldsymbol{\mathcal{R}}_n^{(k)} : \boldsymbol{\mathcal{R}}_n^{(k)} + {r_n^{(k)}}^2\right) \tag{3.33}$$

against a maximum allowable value $\omega_{\max}$. The default is $\omega_{\max} = 10^{-22}$. Eq. (3.33) has units of strain squared, so $\sqrt{\omega_{\max}} = 10^{-11}$ is nine orders of magnitude beneath 1 % strain. This value is quite low, but it only requires a few more iterations than, say $\sqrt{\omega_{\max}} = 10^{-6}$, due to the Newton algorithm's quadratic convergence rate.

One could alternatively check for convergence by comparing the L_2 norm of $\boldsymbol{\mathcal{R}}_n^{(k)}$ and L_2 norm of $r_n^{(k)}$ against two different maximum allowable values. A single merit function $\omega^{(k)}$, however, is preferable herein because $\omega^{(k)}$ feeds directly into the line search algorithm.

3.7 Line Search Algorithm

The search direction and step size are determined all at once in the standard Newton's method, but a line search augmented Newton's method attempts to select the step size by minimizing the merit function. To construct the line search sub-loop within a Newton iteration, replace Eqs. (3.30) to (3.32) with

$$\boldsymbol{\sigma}_n^{(k)} = \boldsymbol{\sigma}_n^{(k-1)} + \zeta^{(j)}\,\delta\boldsymbol{\sigma}_n \tag{3.34}$$

$$\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k)} = \Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k-1)} - \zeta^{(j)}\,\boldsymbol{\mathcal{C}}^{-1} : \delta\boldsymbol{\sigma}_n^{(k)} \tag{3.35}$$

$$\bar{\varepsilon}_n^{\mathrm{tr}\,(k)} = \bar{\varepsilon}_n^{\mathrm{tr}\,(k-1)} + \zeta^{(j)}\,\delta\bar{\varepsilon}_n^{\mathrm{tr}} \tag{3.36}$$

and replace Eq. (3.33) with

$$\omega^{(k)}(\zeta^{(j)}) = \frac{1}{2}\left(\boldsymbol{\mathcal{R}}_n^{(k)}(\zeta^{(j)}) : \boldsymbol{\mathcal{R}}_n^{(k)}(\zeta^{(j)}) + r_n^{(k)}(\zeta^{(j)})^2\right) \tag{3.37}$$

where $\zeta^{(j)} \in (0,\,1]$ controls the step size for line search iteration j. One could numerically vary $\zeta^{(j)}$ find the minimum of $\omega^{(k)}$, but that is often more computationally expensive than it is worth since the search direction is often less than optimal. Instead, we quickly solve for an approximate minimum and update the search direction.

Many techniques exist for finding an approximate minimum, but here we will follow the method outlined in Scherzinger (2017). Replace $\omega^{(k)}(\zeta^{(j)})$ with a simple quadratic approximation, defined as

$$\omega^{(k)}(\zeta^{(j)}) \approx \hat{\omega}^{(k)}(\zeta^{(j)}) = P_0 + P_1\,\zeta^{(j)} + P_2\,\zeta^{(j)^2} \tag{3.38}$$

where P_i are constants. These constants are found for the first line search iteration ($j = 1$) by enforcing

$$\omega^{(k)}(0) = \hat{\omega}^{(k)}(0) = P_0 \tag{3.39}$$

$$\omega^{(k)}(1) = \hat{\omega}^{(k)}(1) = P_0 + P_1 + P_2 \tag{3.40}$$

$$\left.\frac{\partial\omega^{(k)}}{\partial\zeta^{(j)}}\right|_{\zeta^{(j)}=\zeta^{(1)}=0} = \left.\frac{\partial\hat{\omega}^{(k)}}{\partial\zeta^{(j)}}\right|_{\zeta^{(j)}=\zeta^{(1)}=0} = P_1 \tag{3.41}$$

The quantities $\omega^{(k)}(0)$ and $\omega^{(k)}(1)$ are simply the merit function at the beginning and end of the most recent Newton step, so they are known upon entering the line search sub-loop, but $\partial\omega^{(k)}/\partial\zeta^{(j)}\big|_{\zeta^{(j)}=\zeta^{(1)}=0}$ is less obvious. First, expand $\partial\omega^{(k)}/\partial\zeta^{(j)}\big|_{\zeta^{(j)}=\zeta^{(1)}=0}$ as

$$\begin{aligned}\left.\frac{\partial\omega^{(k)}}{\partial\zeta^{(j)}}\right|_{\zeta^{(j)}=\zeta^{(1)}=0} &= \left.\left(\boldsymbol{\mathcal{R}}_n^{(k)} : \frac{\partial\boldsymbol{\mathcal{R}}_n^{(k)}}{\partial\zeta^{(j)}} + r_n^{(k)}\frac{\partial r_n^{(k)}}{\partial\zeta_n^{(j)}}\right)\right|_{\zeta^{(j)}=\zeta^{(1)}=0}\\ &=\boldsymbol{\mathcal{R}}_n^{(k)} : \left.\left(\frac{\partial\boldsymbol{\mathcal{R}}_n^{(k)}}{\partial\boldsymbol{\sigma}_n^{(k)}} : \delta\boldsymbol{\sigma}_n^{(k)} + \frac{\partial\boldsymbol{\mathcal{R}}_n^{(k)}}{\partial\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}}\,\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}\right)\right|_{\zeta^{(j)}=\zeta^{(1)}=0}\\ &\quad + r_n^{(k)}\left.\left(\frac{\partial r_n^{(k)}}{\partial\boldsymbol{\sigma}_n^{(k)}} : \delta\boldsymbol{\sigma}_n^{(k)} + \frac{\partial r_n^{(k)}}{\partial\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}}\,\delta\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}\right)\right|_{\zeta^{(j)}=\zeta^{(1)}=0}.\end{aligned} \tag{3.42}$$

The terms in the parenthesizes of Eq. (3.42) are respectively $-\boldsymbol{\mathcal{R}}_n^{(k)}$ and $-r_n^{(k)}$ (see Eqs. (3.18) and (3.19)), which were already calculated at the end of the last Newton step. Thus,

$$\left.\frac{\partial \omega^{(k)}}{\partial \zeta^{(j)}}\right|_{\zeta^{(j)}=\zeta^{(1)}=0} = -\boldsymbol{\mathcal{R}}_n^{(k)} : \boldsymbol{\mathcal{R}}_n^{(k)} - {r_n^{(k)}}^2 = -2\,\omega^{(k)}(0) = P_1 \tag{3.43}$$

Now, one can solve for P_2 using Eqs. (3.39), (3.40) and (3.43) and plug P_i into Eq. (3.38) to construct $\hat{\omega}(\zeta^{(1)})$ as

$$\hat{\omega}^{(k)}(\zeta^{(1)}) = \omega^{(k)}(0) - 2\,\omega^{(k)}(0)\,\zeta^{(1)} + \left(\omega^{(k)}(0) + \omega^{(k)}(1)\right)\,{\zeta^{(1)}}^2. \tag{3.44}$$

The local minimum of this quadratic approximation is located at

$$\zeta^{(1)} = \frac{\omega^{(k)}(0)}{\omega^{(k)}(0) + \omega^{(k)}(1)}. \tag{3.45}$$

This is appropriate for the first iteration, but there is no guarantee that one iteration will decrease the merit function, so the algorithm does more iterations to find a value of $\zeta^{(j)}$ that produces sufficient improvement. Following Pérez-Foguet and Armero (2002) and Scherzinger (2017), a decrease in $\omega^{(k)}(\zeta^{(j)})$ is considered sufficient if it meets Goldstein's condition, which is[1],

$$\omega^{(k)}(\zeta^{(j)}) < (1 - 2\,\xi\,\zeta^{(j-1)})\,\omega^{(k)}(0) \tag{3.46}$$

where ξ is a constant. Pérez-Foguet and Armero (2002) and Scherzinger (2017) set $\xi = 10^{-4}$, which means almost any decrease in $\omega^{(k)}(\zeta^{(j)})$ is sufficient, and `md_viscoplastic` adopts this as its default value. For the second or greater iteration j, replace Eq. (3.40) with

$$\omega^{(k)}(\zeta^{(j-1)}) = \hat{\omega}^{(k)}(\zeta^{(j-1)}) = P_0 + P_1\,\zeta^{(j-1)} + P_2\,{\zeta^{(j-1)}}^2 \tag{3.47}$$

Solve for P_2 again using Eqs. (3.39), (3.43) and (3.47) and plug P_i into Eq. (3.38) to construct $\hat{\omega}(\zeta^{(j)})$ as

$$\hat{\omega}^{(k)} = \omega^{(k)}(0) - 2\,\omega^{(k)}(0)\,\zeta^{(j)} + \frac{\omega^{(k)}(\zeta^{(j-1)}) - \omega^{(k)}(0) + 2\,\omega^{(k)}(0)\,\zeta^{(j-1)}}{{\zeta^{(j-1)}}^2}\,{\zeta^{(j)}}^2. \tag{3.48}$$

One now obtains the following more general expression for the local minimum[2],

$$\zeta^{(j)} = \frac{\omega^{(k)}(0)\,{\zeta^{(j-1)}}^2}{\omega^{(k)}(\zeta^{(j-1)}) - \omega^{(k)}(0) + 2\,\omega^{(k)}(0)\,\zeta^{(j-1)}}. \tag{3.49}$$

As expected, Eq. (3.49) collapses down to Eq. (3.45) if $\zeta^{(j-1)} = 1$, so Eq. (3.49) is used to calculate $\zeta^{(j)}$ for every line search iteration in the `md_viscoplastic` implementation.

[1]Note, Eq. (3.46) matches Eq. (A11) in Pérez-Foguet and Armero (2002), but it does not match Eq. (51) in Scherzinger (2017). Eq. (3.46) compares $\omega^{(k)}(\zeta^{(j)})$ against $\omega^{(k)}(0)$, yet Scherzinger (2017) compares $\omega^{(k)}(\zeta^{(j)})$ against $\omega^{(k)}(\zeta^{(j-1)})$, which is incorrect.

[2]Note that Eq. (3.49) matches the corresponding equation inside Box A1 of Pérez-Foguet and Armero (2002), but it does not match Eq. (50) in Scherzinger (2017). Eq. (50) in Scherzinger (2017) is equivalent to Eq. (3.45) with $\omega^{(k)}(1)$ replaced with $\omega^{(k)}(\zeta^{(j)})$, which is incorrect.

One other requirement is that $\zeta^{(j)}$ isn't too small, so $\zeta^{(j)}$ is first calculated using Eq. (3.49), and then

$$\zeta^{(j)} \leftarrow \max\left(\gamma\,\zeta^{(j-1)}, \zeta^{(j)}\right) \tag{3.50}$$

is applied, where γ is another constant. Pérez-Foguet and Armero (2002) and Scherzinger (2017) set $\gamma = 0.1$, so that is the default in `md_viscoplastic`.

See Fig. 7 of Scherzinger (2017) and the associated discussion for a graphical demonstration of how the line search augmented Newton's method improves convergence when utilizing the Hosford equivalent stress.

One line search iteration is usually sufficient to meet Eq. (3.46) in the present author's limited testing, but there may be cases where the line search algorithm discussed in this section may not perform so well. If so, tuning γ or ξ could be worth investigating. One could also devise higher order approximations to $\omega^{(k)}$ than the quadratic approximation employed herein. As previously mentioned, however, it is often a better use of resources to attain some improvement with one line search iteration and change search directions with another Newton iteration than to find the optimum value of $\zeta^{(j)}$ along a fixed search direction.

3.8 Material Tangent

Material models in Sierra/Solid Mechanics (2018) do not return the material tangent because Sierra/Solid Mechanics numerically probes to find the stiffness of each finite element. On the other hand, many other finite element codes require the material tangent in order to construct element stiffness matrices. In addition, codes that evaluate a material model at a point, called material point drivers, typically also require the material tangent to enforce stress controlled loading. Therefore expressions for the material tangent will be developed here to enable porting of the `md_viscoplastic` implementation to other codes.

Simo and Hughes (1998) emphasize the distinction between the continuum material tangent $\partial\boldsymbol{\sigma}/\partial\boldsymbol{d}$ and the algorithmic material tangent $\partial\boldsymbol{\sigma}_n/\partial\boldsymbol{d}_n$. As shown in Section 1.5.2.3 of Simo and Hughes (1998), the continuum and algorithmic material tangents are equivalent in one-dimension so it is not trivial to visualize the difference between the two. Nevertheless, they show that the algorithmic material tangent is needed to ensure second order convergence during the global finite element solve, so it is clearly preferred.

The algorithmically consistent material tangent is constructed from the following three differential forms

$$\mathrm{d}\boldsymbol{\sigma}_n = \boldsymbol{\mathcal{C}} : (\mathrm{d}\boldsymbol{\epsilon}_n - \mathrm{d}\boldsymbol{\epsilon}_n^{\mathrm{vp}}) \tag{3.51}$$

$$\mathrm{d}\boldsymbol{\epsilon}_n^{\mathrm{vp}} = \left\{ \left[\left(\Delta\bar{\varepsilon}^{\mathrm{tr}} + \Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\right) \frac{\partial^2 \bar{\sigma}}{\partial \boldsymbol{\sigma}^2} + \Delta t\, \frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \otimes \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \right] : \mathrm{d}\boldsymbol{\sigma} + \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\, \mathrm{d}\bar{\varepsilon}^{\mathrm{tr}} \right\}_n \tag{3.52}$$

$$\mathrm{d}\bar{\varepsilon}_n^{\mathrm{tr}} = \left\{ \Delta t\, \dot{\bar{\varepsilon}}^{\mathrm{ss}} \frac{\partial F}{\partial \bar{\varepsilon}^{\mathrm{tr}}}\, \mathrm{d}\bar{\varepsilon}^{\mathrm{tr}} + \Delta t \left[\dot{\bar{\varepsilon}}^{\mathrm{ss}} \frac{\partial F}{\partial \bar{\sigma}} + (F-1) \frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} \right] \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} : \mathrm{d}\boldsymbol{\sigma} \right\}_n . \tag{3.53}$$

Equation (3.51) is Eq. (3.11) with Δx replaced with $\mathrm{d}x$ to represent differential increments about the converged state. In addition, $\mathrm{d}\boldsymbol{\epsilon}_n^{\mathrm{th}} = 0$ because momentum balance codes only require the stiffness to deformation increments. Equations (3.52) and (3.53) are the discretized flow rule and transient strain ordinary differential equation (Eqs. (3.15) and (3.16), respectively) differentiated about the converged state. In fact, one can rewrite Eqs. (3.52) and (3.53) respectively as

$$\mathbf{0} = \left(\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\sigma}}\right)_n : \mathrm{d}\boldsymbol{\sigma}_n + \left(\frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\right)_n \mathrm{d}\bar{\varepsilon}_n^{\mathrm{tr}}, \tag{3.54}$$

$$0 = \left(\frac{\partial r}{\partial \bar{\sigma}} \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\right)_n : \mathrm{d}\boldsymbol{\sigma}_n + \left(\frac{\partial r}{\partial \bar{\varepsilon}^{\mathrm{tr}}}\right)_n \mathrm{d}\bar{\varepsilon}_n^{\mathrm{tr}}, \tag{3.55}$$

which are respectively similar to Eqs. (3.24) and (3.25), except they pertain to the converged state, so the iteration k superscripts are absent and $\boldsymbol{\mathcal{R}}_n = \mathbf{0}$ and $r_n = 0$.

The process to derive the algorithmically consistent material tangent is somewhat similar to the process to derive the Newton iterations in Section 3.6. Equation (3.54) is inserted into Eq. (3.51) and the result is rearranged to obtain

$$\left\{ \left[\boldsymbol{\mathcal{C}}^{-1} + \left(\Delta\bar{\varepsilon}^{\mathrm{tr}} + \Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\right) \frac{\partial^2 \bar{\sigma}}{\partial \boldsymbol{\sigma}^2} + \Delta t\, \frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \otimes \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \right] : \mathrm{d}\boldsymbol{\sigma} + \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}\, \mathrm{d}\bar{\varepsilon}^{\mathrm{tr}} \right\}_n = \mathrm{d}\boldsymbol{\epsilon}_n \tag{3.56}$$

Equation (3.55) is then combined with Eq. (3.56) to eliminate $\mathrm{d}\bar{\varepsilon}_n^{\mathrm{tr}}$:

$$\left\{ \left[\boldsymbol{\mathcal{C}}^{-1} + \left(\Delta\bar{\varepsilon}^{\mathrm{tr}} + \Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\right) \frac{\partial^2 \bar{\sigma}}{\partial \boldsymbol{\sigma}^2} + \left(\Delta t\, \frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} - \frac{\partial \bar{\varepsilon}^{\mathrm{tr}}}{\partial r} \frac{\partial r}{\partial \bar{\sigma}} \right) \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \otimes \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \right] : \mathrm{d}\boldsymbol{\sigma} \right\}_n = \mathrm{d}\boldsymbol{\epsilon}_n . \tag{3.57}$$

This result can be re-expressed as

$$\mathrm{d}\boldsymbol{\sigma}_n = \boldsymbol{\mathcal{C}}_n^{\mathrm{t}} : \mathrm{d}\boldsymbol{\epsilon}_n \tag{3.58}$$

where the algorithmically consistent material tangent is

$$\boldsymbol{\mathcal{C}}_n^{\mathrm{t}} = \left[\boldsymbol{\mathcal{C}}^{-1} + \left(\Delta\bar{\varepsilon}^{\mathrm{tr}} + \Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\right) \frac{\partial^2 \bar{\sigma}}{\partial \boldsymbol{\sigma}^2} + \left(\Delta t\, \frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} - \frac{\partial \bar{\varepsilon}^{\mathrm{tr}}}{\partial r} \frac{\partial r}{\partial \bar{\sigma}} \right) \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \otimes \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \right]_n^{-1} . \tag{3.59}$$

Comparing Eq. (3.29) to Eq. (3.59), one can see that $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}(k-1)}$ is simply the unconverged algorithmic material tangent. Thus, one can compute $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$ for the converged state using the same algorithm used to compute $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}(k-1)}$ inside the Newton iteration loop. This reuse of code is demonstrated in Appendix C.2.2, except the final calculation of $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$ is commented out since Sierra/Solid Mechanics does not make use of the material tangent.

3.9 Implementation Outline

The last several sections contain a number of derivations and explanations that can cause one to lose track of the procedure being developed. To assist the reader, the following outline describes how the `md_viscoplastic` implementation calculates the unrotated stress at the end of the time step.

1. Receive $\boldsymbol{\sigma}_{n-1}$, $\boldsymbol{d}_n$, Δt_n, T_n, and ΔT_n from the host code.
2. Evaluate $\omega^{(0)}$.
    (a) Assume $\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(0)} = 0$, evaluate $\boldsymbol{\sigma}_n^{(0)} = \boldsymbol{\sigma}_n^{\mathrm{trial}}$.
    (b) Calculate the principal stresses (and directions) of $\boldsymbol{\sigma}_n^{(0)}$ in order to compute $\bar{\sigma}^{(0)}$.
    (c) Complete evaluation of $\omega^{(0)}$ with all tensors expressed in the principal frame of $\boldsymbol{\sigma}_n^{(0)}$.
3. If $\omega^{(0)} \leq \omega_{\max}$, then accept $\boldsymbol{\sigma}_n = \boldsymbol{\sigma}_n^{\mathrm{trial}}$ and skip the Newton iteration loop.
4. If $\omega^{(0)} > \omega_{\max}$, Newton iterations ensue with all tensors expressed in the principal frame of $\boldsymbol{\sigma}_n^{(0)}$. The quantities $\boldsymbol{\sigma}_n^{(k)}$, $\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k)}$, $\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$, and $\omega^{(k)}$ are calculated with each iteration k.
    (a) If $\omega^{(k)}$ fails to meet Eq. (3.46), then a sub-loop of Line Search iterations commences. Each iteration j changes $\zeta^{(j)}$ according to Eq. (3.49) and updates $\boldsymbol{\sigma}_n^{(k)}$, $\Delta\boldsymbol{\epsilon}_n^{\mathrm{vp}\,(k)}$, $\bar{\varepsilon}_n^{\mathrm{tr}\,(k)}$, and $\omega^{(k)}$.
        i. If $\omega^{(k)}$ fails to meet Eq. (3.46) after a predetermined number of iterations, return an error to the host code.
        ii. If $\omega^{(k)}$ meets Eq. (3.46), the Line Search returns control to the Newton iteration loop.
    (b) If $\omega^{(k)} > \omega_{\max}$ after a predetermined number of iterations, return an error to the host code.
    (c) If $\omega^{(k)} \leq \omega_{\max}$, exit the Newton iteration loop.
5. Save the internal state variable $\bar{\varepsilon}_n^{\mathrm{tr}}$ (and any other desired internal variables).
6. Optionally compute the material tangent $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$ using the converged state.
7. Transform $\boldsymbol{\sigma}_n$ (and optionally $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$) from the principal frame to the original input frame.
8. Return $\boldsymbol{\sigma}_n$ (and optionally $\boldsymbol{\mathcal{C}}_n^{\mathrm{t}}$) to the host code.

3.10 Further Information

See Section 5.2.21 of the Sierra/Solid Mechanics (2018) User Guide for the proper input syntax to use the md_viscoplastic implementation in Sierra. Also, see Appendix C for a `md_viscoplastic` implementation in the C++ programming language that could be ported to other momentum balance codes.

Chapter 4

Verification

Four tests were used to verify the new `md_viscoplastic` implementation. Each of the three tests in Section 4.1 involve applying a specific prescribed stress and temperature history to a material point, and comparing the numerically calculated strain history against an analytical solution. The test in Section 4.2 benchmarks the `md_viscoplastic` implementation against the `implicit_wipp_crushed_salt` implementation in a room closure simulation.

4.1 Material Point Verification Tests

This section discusses the material point tests used to verify the MD model's numerical implementation. The MD model contains two ordinary differential equations (Eqs. (2.6) and (2.11)) that make it non-trivial to verify. A straightforward analytical solution, however, can be constructed to these equations if $\chi = 1$ and if the stresses and temperatures remain piecewise constant in time.

Temporally constant stresses and temperatures allow Eqs. (2.1) to (2.8) to be integrated to

$$\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}(t_j) = \boldsymbol{\mathcal{C}}^{-1} : (\boldsymbol{\sigma} - \boldsymbol{\sigma}(t_j)) - \alpha \, (T - T(t_j)) \, \boldsymbol{I} + \boldsymbol{\varepsilon}^{\text{vp}} - \boldsymbol{\varepsilon}^{\text{vp}}(t_j) \tag{4.1}$$

$$\boldsymbol{\varepsilon}^{\text{vp}} - \boldsymbol{\varepsilon}^{\text{vp}}(t_j) = \left[\bar{\varepsilon}^{\text{tr}} - \bar{\varepsilon}^{\text{tr}}(t_j) + \dot{\bar{\varepsilon}}^{\text{ss}} \, (t - t_j)\right] \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} \tag{4.2}$$

where t_j is the time at the end of the previous time period j. The quantities from the previous time period ($\boldsymbol{\varepsilon}(t_j)$, $\boldsymbol{\sigma}(t_j)$, $T(t_j)$, $\boldsymbol{\varepsilon}^{\text{vp}}(t_j)$, and $\bar{\varepsilon}^{\text{tr}}(t_j)$) are assumed to be known. Setting $\chi = 1$ causes Eq. (3.6) to reduce to Eq. (3.5), which enables an analytical solution to Eq. (2.11). Under these conditions, the general solution to Eq. (2.11) is

$$\bar{\varepsilon}^{\text{tr}} = \frac{\bar{\varepsilon}^{\text{tr}*}}{\kappa} \ln \left\{ \exp(\kappa) + \exp\left[\frac{\kappa}{\bar{\varepsilon}^{\text{tr}*}} \left(C_1 - \dot{\bar{\varepsilon}}^{\text{ss}} t\right)\right] \right\}. \tag{4.3}$$

One can solve for the integration constant C_1 using the initial condition $\bar{\varepsilon}^{\text{tr}} = \bar{\varepsilon}^{\text{tr}}(t_j)$ at $t = t_j$. After substituting the result back into Eq. (4.3), one obtains

$$\bar{\varepsilon}^{\text{tr}} = \frac{\bar{\varepsilon}^{\text{tr}*}}{\kappa} \ln \left\{ \exp(\kappa) + \left[\exp\left(\frac{\bar{\varepsilon}^{\text{tr}}(t_j) \, \kappa}{\bar{\varepsilon}^{\text{tr}*}} \right) - \exp(\kappa) \right] \exp\left[-\frac{\kappa}{\bar{\varepsilon}^{\text{tr}*}} \, \dot{\bar{\varepsilon}}^{\text{ss}} \, (t - t_j) \right] \right\}. \tag{4.4}$$

Combining Eqs. (4.1), (4.2) and (4.4) produces the following closed form expression for the total strain change over a time period

$$
\begin{aligned}
\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}(t_j) = \boldsymbol{\mathcal{C}}^{-1} : (\boldsymbol{\sigma} - \boldsymbol{\sigma}(t_j)) - \alpha\ (T - T(t_j))\ \boldsymbol{I} + \\
&\left[\frac{\bar{\varepsilon}^{\text{tr}*}}{\kappa} \ln \left\{ \exp(\kappa) + \left[\exp\left(\frac{\bar{\varepsilon}^{\text{tr}}(t_j)\, \kappa}{\bar{\varepsilon}^{\text{tr}*}} \right) - \exp(\kappa) \right] \exp\left[-\frac{\kappa}{\bar{\varepsilon}^{\text{tr}*}}\, \dot{\bar{\varepsilon}}^{\text{ss}}\, (t - t_j) \right] \right\} \right. \\
&\left. - \bar{\varepsilon}^{\text{tr}}(t_j) + \dot{\bar{\varepsilon}}^{\text{ss}}\, (t - t_j) \right] \frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}}.
\end{aligned}
\tag{4.5}
$$

The next three subsections compare numerical solutions against analytical solutions for axisymmetric compression, pure shear, and unequal biaxial compression. In each case, the numerical solution for the total strain is denoted as $\boldsymbol{\varepsilon}$, while the analytical solution for the total strain in Eq. (4.5) is denoted as $\hat{\boldsymbol{\varepsilon}}$. All the verification tests only involve principal deformations, so the hypoelastic framework discussed in Section 2.4 reinterprets the stress and infinitesimal strain in Section 2.1 as the Cauchy stress and logarithmic strain, respectively. Note again that compressive stresses and strains are treated as positive, as is common in the geomechanics literature.

All three material point verification tests utilize Calibration 3B of the MD model. The full parameter set can be found in Table 5.2, while Figs. 2.2, 2.3 and 5.3c depict the calibration graphically. Figures 2.2 and 2.3 show the individual mechanisms $\bar{\varepsilon}_i^{\text{tr}*}$ and $\dot{\bar{\varepsilon}}_i^{\text{ss}}$, as well as the sums $\bar{\varepsilon}^{\text{tr}*} = \sum_{i=0}^{1} \bar{\varepsilon}_i^{\text{tr}*}$ and $\dot{\bar{\varepsilon}}^{\text{ss}} = \sum_{i=0}^{3} \dot{\bar{\varepsilon}}_i^{\text{ss}}$, so that one can visualize where each mechanism dominates the total behavior. Figure 5.3c shows the shape of the Hosford equivalent stress surface for $a = 16$. The Hosford surface and the angle ϕ of its normal $\boldsymbol{n} = \partial\bar{\sigma}/\partial\boldsymbol{\sigma}$ depend on the Lode angle ψ of the deviatoric stress $\boldsymbol{\sigma}^{\text{dev}}$.

4.1.1 Triaxial Compression

Triaxial compression tests are frequently used to characterize the creep and strength behavior of geomaterials, such as rock salt. Cylindrical specimens are subjected to a radial confining pressure σ_{rr} and an axial stress σ_{zz}. The hoop stress $\sigma_{\theta\theta}$ is equal to σ_{rr}. See Section 2.2 for an in depth analysis of a typical triaxial compression test in the context of the MD model.

The applied stress and temperature histories for the verification test are shown in the top two plots in Fig. 4.1. The test begins with an isothermal, 20 MPa hydrostatic, hold period for 10 days, where the strain is purely elastic. At $t = 0$ d, σ_{zz} is increased to 35 MPa, while the other stresses are held fixed, causing a 15 MPa equivalent stress. This stress state is held for the next 50 days. The strain evolves quickly at first, but slows down to the steady-state rate as the material work hardens. At $t = 50$ d, σ_{zz} is decreased to 33 MPa, while the other stresses are held fixed. The 2 MPa drop in $\bar{\sigma}$ causes the strain rate to slow down markedly, but it gradually builds to a new steady-state rate as the material recovers over the next 50 days.

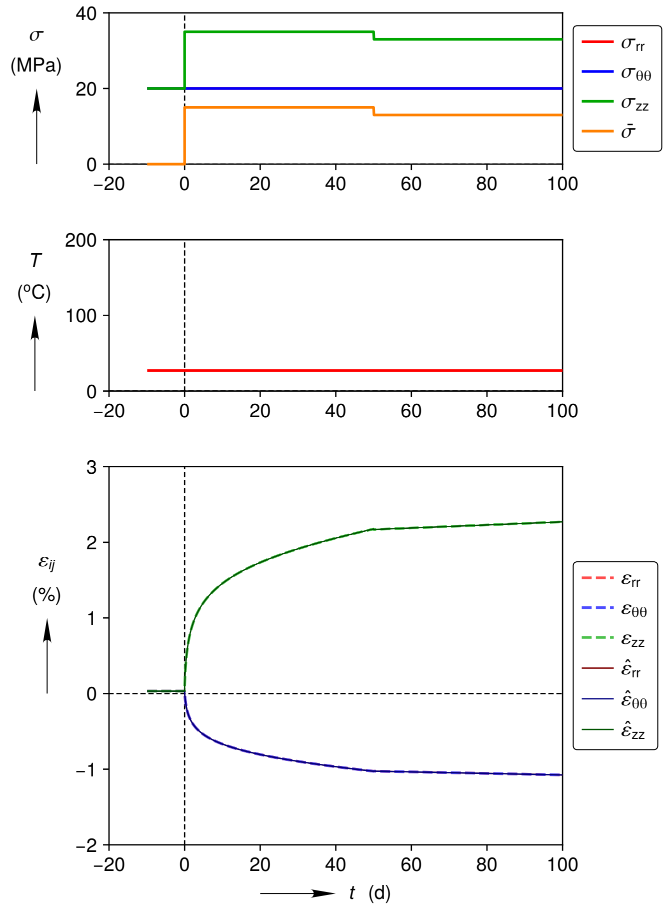


Figure 4.1: Triaxial Compression Verification Test

The numerical and analytical solutions for the total strain match very well throughout the test, which verifies:

- Linear elasticity under triaxial compression
- Zero viscoplastic strain evolution under hydrostatic loading
- Viscoplastic strain evolution for $\psi = -30°$
- Work hardening ($\bar{\varepsilon}^{\rm tr} < \bar{\varepsilon}^{\rm tr*}$) governed by transient strain limit mechanism 1
- Recovery ($\bar{\varepsilon}^{\rm tr} > \bar{\varepsilon}^{\rm tr*}$) governed by transient strain limit mechanism 1
- Steady-state strain accumulation governed by mechanism 2.

4.1.2 Pure shear

The Hosford equivalent stress depends on a for $-30° < \psi < 30°$, but it is independent of a for $\psi = -30°$(triaxial compression) and $\psi = 30°$ (triaxial extension). Pure shear is a simple stress state that exercises the Hosford stress at a Lode angle other than $\psi = \pm 30°$. Pure shear can be expressed in the principal frame as

$$\boldsymbol{\sigma} = \sigma_1 \, \tilde{\boldsymbol{e}}_1 \otimes \tilde{\boldsymbol{e}}_1 + \sigma_3 \, \tilde{\boldsymbol{e}}_3 \otimes \tilde{\boldsymbol{e}}_3, \tag{4.6}$$

where $\sigma_3 = -\sigma_1$, and $\tilde{\boldsymbol{e}}_i$ are the principal stress directions. In addition to exercising the model under pure shear, this test also varies the temperature to verify thermal expansion and creep at elevated temperatures.

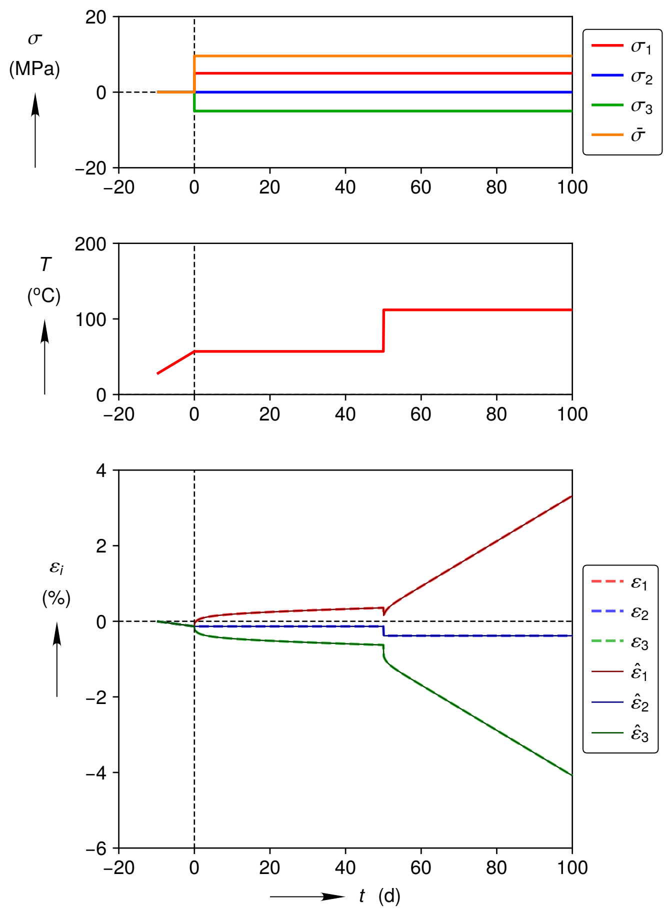


Figure 4.2: Pure Shear Verification Test

The applied stress and temperature histories for the test are shown in the top two plots in Fig. 4.2. The test begins with a 0 MPa hydrostatic hold period for 10 days while the temperature is linearly ramped from 27 °C to 57 °C. Some thermal strains develop during this time. At $t = 0$ d, the temperature ramp stops, σ_1 is increased to 5 MPa, σ_2 is held at zero, and σ_3 is reduced to -5 MPa. This state is held for the next 50 days, while the material creeps. At $t = 50$ d, T is increased to 112 °C, but the stresses remain fixed. The sharp increase in T causes a step change in thermal strain, and then accelerated creep is observed over the over the next 50 days.

The numerical and analytical solutions for the total strain match very well throughout the test, which verifies:

- Linear elasticity under pure shear
- Thermal expansion
- Viscoplastic strain evolution for $\psi = 0°$
- Temperature dependence of transient strain limit mechanism 1
- Steady-state strain accumulation governed by mechanism 1
- Steady-state strain accumulation governed by mechanism 2.

4.1.3 Unequal Biaxial Compression

Unequal biaxial compression is another stress state that exercises the Hosford stress at a Lode angle other than $\psi = \pm 30°$. Unequal compressive stresses σ_{xx} and σ_{yy} are applied to two faces of a cube, while $\sigma_{\mathrm{zz}} = 0$. This stress state is slightly more complex than triaxial compression or pure shear because all three stress magnitudes are unequal. This test also alters the stress component ratios after 50 days of creep to verify the model's ability to change Lode angle.

The applied stress and temperature histories for the test are shown in the top two plots in Fig. 4.3. The test begins with a stress free hold period for 10 days. At $t = 0$ d, σ_{xx} is increased to 3.5 MPa, σ_{yy} is increased to 5 MPa, and σ_{zz} is held at zero. In this stress state, $\psi = 13.0°$ and the intermediate principal stress is σ_{xx}. The intermediate principal strain rate $\dot{\varepsilon}_{\mathrm{xx}} \approx 0$ and $\dot{\varepsilon}_{\mathrm{yy}} \approx -\dot{\varepsilon}_{\mathrm{zz}}$ because the flow rule (Eq. (2.6)) causes $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$ to be coaxial with the flow potential normal $\boldsymbol{n} = \partial\bar{\sigma}/\partial\boldsymbol{\sigma}$, and $\boldsymbol{n}$ is nearly horizontal at $\psi = 13.0°$ in Calibration 3B (see Fig. 5.3c). At $t = 50$ d, σ_{xx} is increased to 6.0 MPa, while the other stresses remain fixed. The sharp increase in σ_{xx} causes a step change in elastic strain that is visible because the viscoplastic strains are small at these low values of $\bar{\sigma}$. In this stress state, $\psi = 21.1°$ and the intermediate principal stress is σ_{yy}. Accordingly, $\dot{\varepsilon}_{\mathrm{yy}} \approx 0$ and $\dot{\varepsilon}_{\mathrm{xx}} \approx -\dot{\varepsilon}_{\mathrm{zz}}$. If one looks more closely, however, $\dot{\varepsilon}_{\mathrm{yy}}$ is slightly positive and $\dot{\varepsilon}_{\mathrm{xx}} > -\dot{\varepsilon}_{\mathrm{zz}}$ because $\psi = 21.1°$ is beginning to approach the corner of the Tresca hexagon (see again Fig. 5.3c).

The numerical and analytical solutions for the total strain match very well throughout the test, which verifies:

- Linear elasticity under unequal biaxial compression

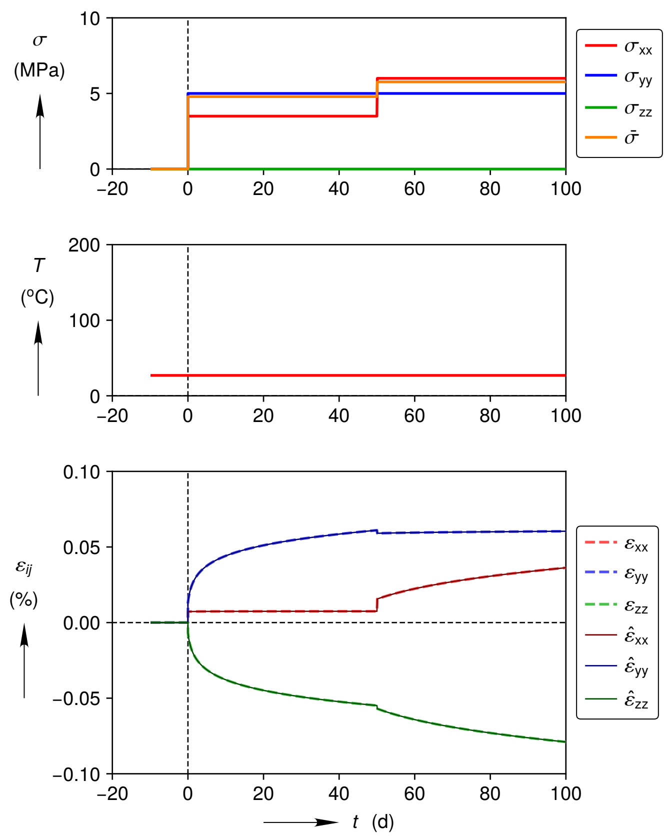


Figure 4.3: Unequal Biaxial Compression Verification Test

- Viscoplastic strain evolution for $\psi = 13.0°$ and a subsequent change to $\psi = 21.1°$
- Transient strain accumulation governed by transient strain limit mechanism 0
- Steady-state strain accumulation governed by mechanism 0.

4.2 A Code-to-Code Benchmark Test

A typical structural simulation exercises the `md_viscoplastic` implementation over a much greater space than the material point verification tests in Section 4.1, but such simulations often have no known analytical solution. One can, however, compare simulations using different MD model implementations as a less rigorous, but still useful, benchmarking exercise. This section compares simulations of Room D closure using the `md_viscoplastic`, `munson_-dawson`, and `implicit_wipp_crushed_salt` implementations. Recall that Room D was a room temperature underground experiment at the Waste Isolation Pilot Plant (Reedlunn 2016), and recall that the `implicit_wipp_crushed_salt` implementation reduces to the `md_creep` implementation when the crushed salt density matches that of the intact salt.

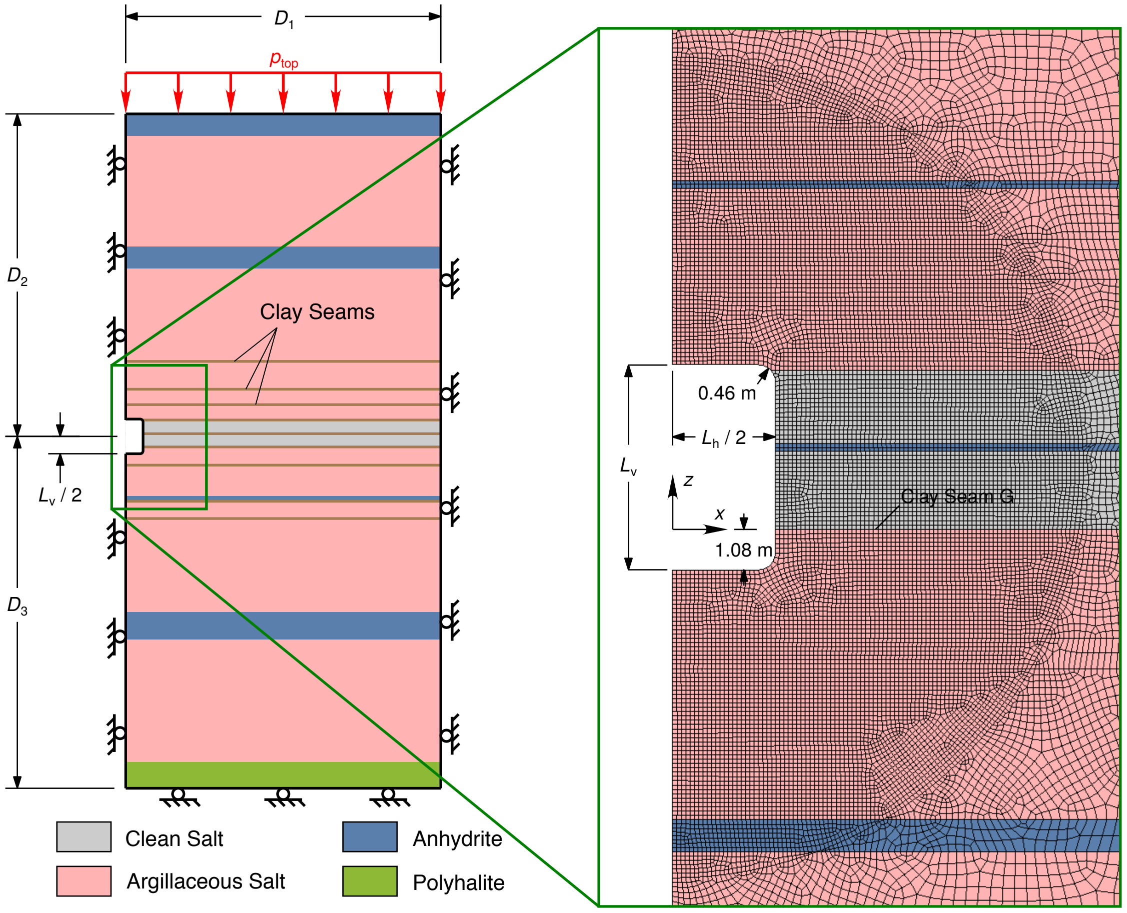
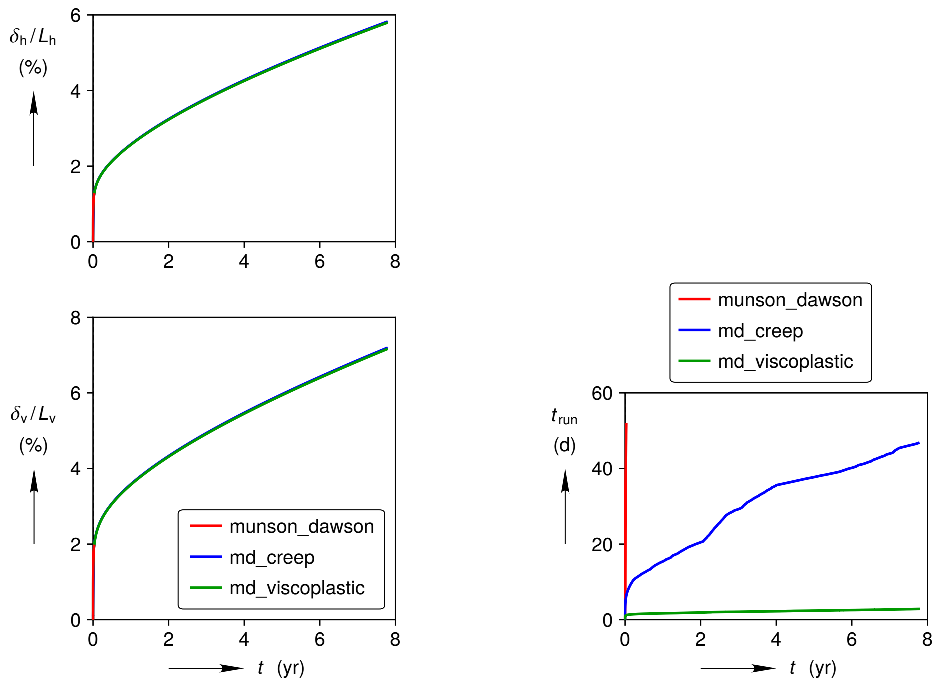


Figure 4.4: Stratigraphy, boundary conditions, and dimensions. The close up of the room also shows the finite element mesh.

Figure 4.4 depicts the simulation domain, boundary conditions, select dimensions, and a close-up of the mesh for the Room D simulations. The room's 93.3 m length in the y-direction (into the page) is assumed to be long enough to be treated as plane strain. The left side is a mirror boundary condition. The distances from the room center are $D_1 = 50$ m, $D_2 = 51.2$, and $D_3 = 55.86$. The initial width and height of Room D are $L_\mathrm{h} = 5.5$ m and $L_\mathrm{v} = 5.5$ m, respectively. The strata are drawn to scale in Fig. 4.4, but see Figure 3.3 of Munson et al. (1989) for finer details. The horizontal closure δ_h was measured at room mid-height, while the vertical closure δ_v was measured at room mid-width. The clean salt, argillaceous salt, anhydrite, and polyhalite had a density of $\rho = 2,300$ kg/m^3, and gravity was $g = 9.79$ m/s^2. All material points were initialized with a hydrostatic stress state that varied linearly from $p_\mathrm{top} = 13.57$ MPa at the domain top to $p_\mathrm{bot} = 15.97$ MPa at the domain bottom. The room appeared virtually instantaneously as a void at time $t = 0$, rather than modeling the excavation process. For simplicity, the rock mass temperature was spatially uniform at 300 K. The clean salt and argillaceous salt were modeled using their respective legacy MD model calibrations listed in Table 5.1, except the `md_visco-plastic` implementation used $a = 1000$ to approximate the Tresca equivalent stress, while the other implementations required no such approximation. The lateral sliding across clay seams was captured using Coulomb friction, with a friction coefficient of 0.2. The anhydrite and polyhalite were modeled with a perfectly plastic Drucker-Prager model (see Appendix A.1 and Chapter 2 of Reedlunn (2016) for further information).

A few numerical details bear mentioning. All simulations utilized the implicit quasi-statics capability in Sierra/Solid Mechanics (2017). After instantaneous excavation, the time step increment began at 0.1 s. During room closure, the time step increment grew by 1 % after each successful step, and shrank by 10 % after each failed step. The equilibrium equations, including contact and friction at the clay seams, were solved to a maximum relative residual norm of $R_{\max} = 10^{-6}$, where $R_{\max}$ is the L_2 norm of the total residual divided by the L_2 norm of the externally applied load p_{top}. The selective deviatoric element was used and the finite element mesh near Room D is shown in the close-up view in Fig. 4.4. The room had about 24 elements across its half width. The maximum relative residual norm, element type, and element size were chosen based on the convergence studies in Section 2.2 of Reedlunn (2016).

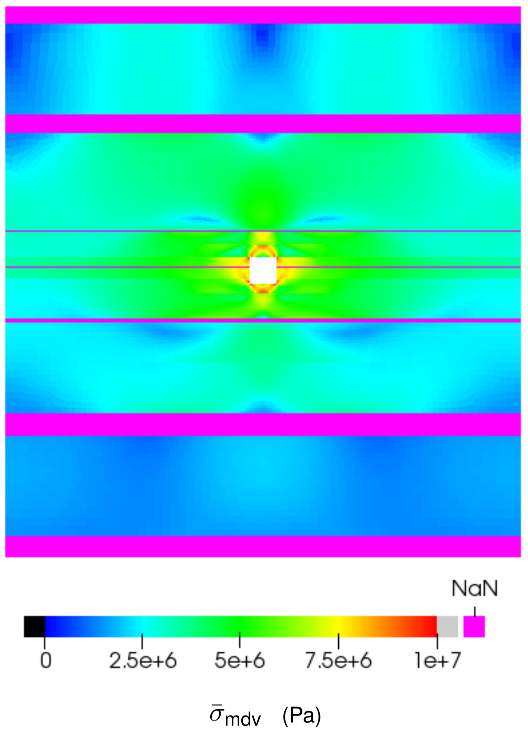


(a) Percent Horizontal and Vertical Closure

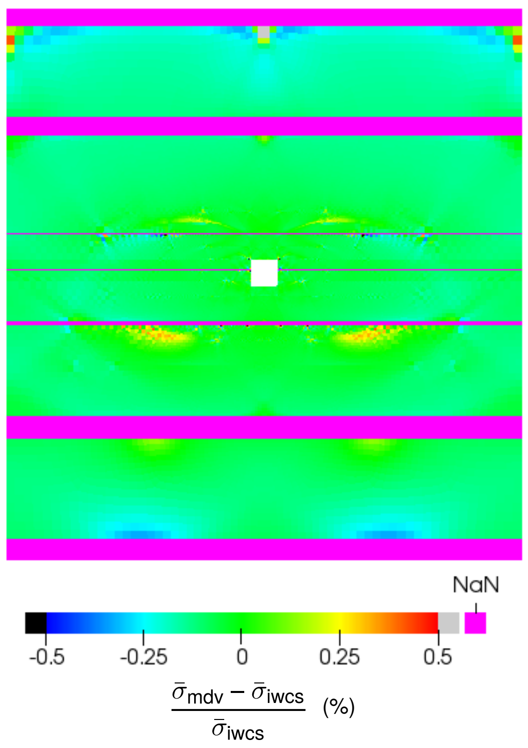
(b) Run time

Figure 4.5: Room D simulations with two old MD model implementations (`munson_dawson` and `implicit_wipp_crushed_salt`) and the new MD model implementation (`md_viscoplastic`).

The horizontal and vertical closure, normalized by the width and height of the room, respectively, are plotted against time t in Fig. 4.5a, and the simulation run time t_{run} on 24 processors is plotted against time t in Fig. 4.5b. The `munson_dawson` implementation only simulated 11.4 days after 52 days of run time. This slow performance was due to the critical time step issue discussed at the start of Chapter 3, so further comparisons will only concern the `implicit_wipp_crushed_salt` and `md_viscoplastic` implementations. The closure results from the two implementations are nearly identical, but close inspection reveals that the closure predictions differ by 0.4 % at $t = 7.8$ yr. This near negligible difference in closure between the `implicit_wipp_crushed_salt` and `md_viscoplastic` implementations is due in part to using $a = 1000$ to approximate the Tresca equivalent stress in the `md_visco-plastic` implementation. The Hosford stress differs by 0.07 % from the Tresca stress in pure shear for $a = 1000$ (as discussed in Appendix A.4), which may seem small, but this difference is amplified by the exponent $n_2 = 5$ in Eq. (2.9) to become a 0.35 % difference in $\dot{\varepsilon}_2^{\text{ss}}$. Other causes for the difference in simulated room closure may be related to how tightly the equations were solved inside the two implementations, time step size differences in the two simulations, or difficulties `implicit_wipp_crushed_salt` may have experienced while selecting the appropriate face of the Tresca hexagon for a given stress state. Regardless, a 0.4 % difference in both horizontal and vertical closure was considered small enough to not pursue further here.



(a) Equivalent stress field from `md_visco-plastic`



(b) Equivalent stress field percent difference

Figure 4.6: Equivalent stress fields around Room D calculated with the `md_viscoplastic` implementation ($\bar{\sigma}_{\text{mdv}}$) and the `implicit_wipp_crushed_salt` implementation ($\bar{\sigma}_{\text{iwcs}}$) after 7.8 years of closure.

The equivalent stress fields around Room D after 7.8 years of closure are compared in Fig. 4.6. In each subfigure, the results have been mirrored to account for the left mirror boundary condition. Also, the open room is white and the anhydrite/polyhalite are magenta since $\bar{\sigma}$ was not calculated inside those non-salt rock masses. Figure 4.6a depicts the equivalent stress field from the `md_viscoplastic` implementation $\bar{\sigma}_{\text{mdv}}$, which is quite complex due to the clay seams, anhydrite layers, and proximity of the simulation boundaries to the room. The percent differences between $\bar{\sigma}_{\text{mdv}}$ and the equivalent stress from the `implicit_wipp_crushed_salt` implementation $\bar{\sigma}_{\text{iwcs}}$ are shown in Fig. 4.6b. The max difference is 1 %, which is next to a clay seam, but the vast majority of the differences are between -0.25 % and 0.25 %. These differences are likely due to the same minor issues causing the closure differences in Fig. 4.5a, and, again, the differences are considered small enough to not pursue further.

Returning to Fig. 4.5b, it is worth noting that the `md_viscoplastic` implementation finished the simulation 16× faster than the `implicit_wipp_crushed_salt` implementation. The `implicit_wipp_crushed_salt` implementation failed to converge 2983 times, while the `md_viscoplastic` implementation failed to converge only 17 times. Each material model failure resulted in a 10 % time step size cut-back, greatly slowing down the `implicit_wipp_crushed_salt` simulation. The `implicit_wipp_crushed_salt` and `md_viscoplastic` implementations are different enough that it is not possible to conclude exactly why the `md_viscoplastic` implementation is more robust, but it is likely due in part to the line search algorithm in the `md_viscoplastic` implementation. It should be possible to make a direct comparison in the future by running the `md_viscoplastic` implementation with the line search algorithm turned off.

Chapter 5

Calibration

This chapter documents the calibration of the three new features added to the MD model in Section 2.1. Section 5.1 details the re-calibration of the steady-state equivalent strain rate and transient equivalent strain limit, while Section 5.2 details the calibration of the Hosford exponent.

5.1 Steady-State Equivalent Strain Rate and Transient Equivalent Strain Limit

The MD model was recently calibrated against a series of triaxial compression creep experiments on WIPP salt (see Chapter 3 of Reedlunn (2016)). At the time, the MD model was not flexible enough to capture the observed creep behavior at low (< 8 MPa) equivalent stresses, so measurements of $\dot{\bar{\varepsilon}}^{\text{ss}}$ and $\bar{\varepsilon}^{\text{tr}^*}$ at those low stresses were ignored. Now that new equivalent steady-state strain rate and transient strain limit mechanisms have been added, it is possible to recalibrate the MD model to include this low stress regime.

The triaxial compression creep experiments were performed by the Institut für Gebirgsmechanik (IfG) in Germany. Salzer et al. (2015), Düsterloh et al. (2015), and Reedlunn (2016) provide details of how the experiments were conducted and analyzed, but a few important points are worth repeating here.

- Accurate measurement of creep behavior at low equivalent stresses is a challenging task. The strain measurement technique must be quite precise and one must wait months to reach steady-state. During this time, small changes in temperature and humidity that would be negligible for $\bar{\sigma} \geq 8$ MPa can be significant for $\bar{\sigma} < 8$ MPa. To mitigate these issues, the IfG increased the creep strain by conducting their tests at 60 °C, and they used a technique to approach $\dot{\bar{\varepsilon}}^{\text{ss}}$ from "above" and from "below", as detailed in Günther et al. (2014).
- The technique to approach $\dot{\bar{\varepsilon}}^{\text{ss}}$ from "above" and from "below" utilized two different non-zero values of $\bar{\sigma}$ for each specimen, such that a $\dot{\bar{\varepsilon}}^{\text{ss}\,(j)}$ and $\bar{\varepsilon}^{\text{tr}^*\,(j)}$ could be measured from each test segment j. Each test segment was assigned a weighting factor $w^{(j)}$ based on qualitatively judging the quality of the creep response curve.

- The IfG began each creep experiment with 10 days of hydrostatic compression to heal microcracks due to specimen extraction and preparation. Judging by the $\varepsilon_{\mathrm{zz}}$ accumulation, some specimens would have continued to consolidate further had the hydrostatic compression been applied for longer. Although incomplete consolidation probably had negligible impact on the medium to high ($\geq$ 8 MPa) equivalent stress experiments, it may have affected the low equivalent stress experiments where the creep strains were much smaller.

- Düsterloh et al. (2015) and Reedlunn (2016) independently analyzed the IfG's creep measurements on "argillaceous" and "clean" salt from WIPP, and both concluded the differences between the two salt types were negligible. Reedlunn (2016) consequently chose to create one WIPP salt calibration for both argillaceous and clean salt. The same approach will be utilized here, but the differences between the two salt types could use further investigation, as discussed in Section 3.1 and 4.1.4 of Reedlunn (2016).

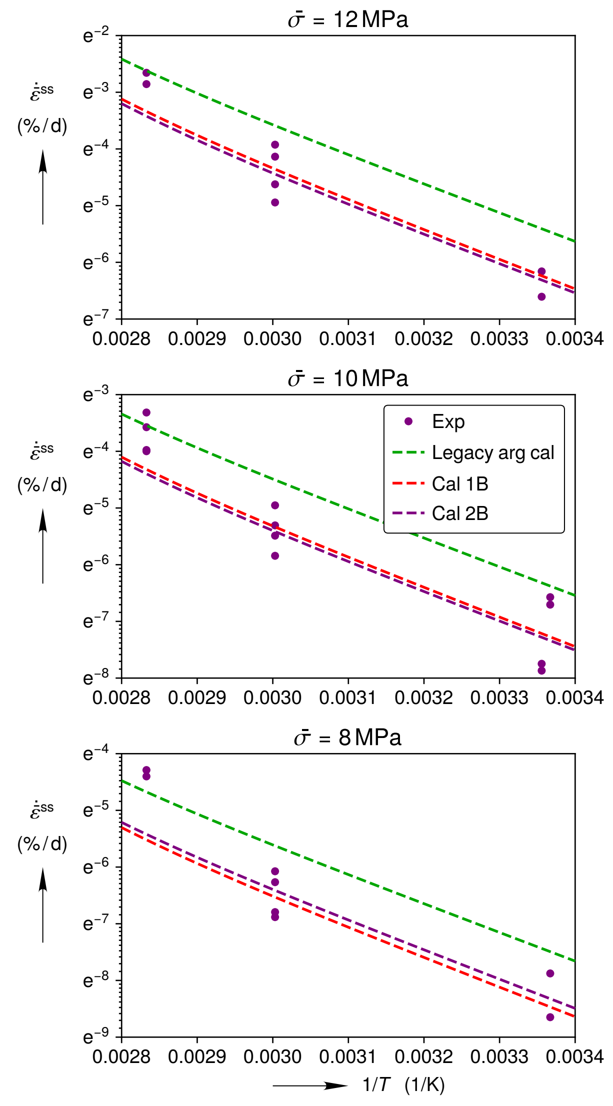
- Section 3.3 in Reedlunn (2016) analyzed the IfG creep experiments using two different methods. In short, method A measured $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$ and $\bar{\varepsilon}^{\mathrm{tr}*}$ directly from each creep response curve, while method B fit the MD model to each creep response curve to extract $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$, $\bar{\varepsilon}^{\mathrm{tr}*}$, and κ_{h}. Method A has the advantage that it is "model agnostic", while method B does not require the creep curve to fully reach steady-state. As both methods produced similar results (see Fig. 3.6 and 4.1 in Reedlunn (2016)), this report uses the values of $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$ and $\bar{\varepsilon}^{\mathrm{tr}*}$ from method B.

The $\dot{\bar{\varepsilon}}^{\mathrm{ss}\,(j)}$, $\bar{\varepsilon}^{\mathrm{tr}*\,(j)}$, and $w^{(j)}$ values for all the IfG experiments are listed in Table A.2 of Reedlunn (2016).

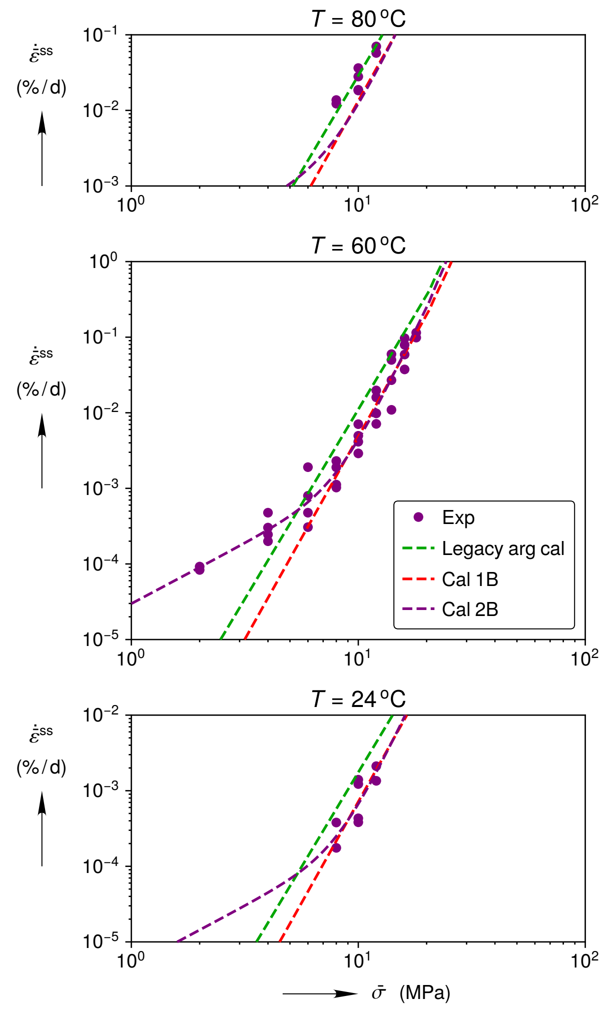
The experimental results are shown at three different temperatures in Figs. 5.1 and 5.2. The $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$ vs. $\bar{\sigma}$ plot in Fig. 5.1b at $T = 60$ °C exhibits what appears to be bi-linear behavior, with a change in slope at $\bar{\sigma} \approx 8$ MPa. This change in $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$'s slope has been reported before (e.g. Bérest et al. (2005), Bérest et al. (2015), Salzer et al. (2015), and Düsterloh et al. (2015)), but Reedlunn (2016) observed that $\bar{\varepsilon}^{\mathrm{tr}*}$ also exhibits a change in slope at the same value of $\bar{\sigma} \approx 8$ MPa in Fig. 5.2b. The new MD model calibration (Calibration 2B) was designed to capture both bi-linear behaviors.

The matrix of $\bar{\sigma}$ and T values in the IfG tests was not extensive enough to fully parameterize the MD model's equivalent viscoplastic functions. Consequently Calibration 2B was limited in several ways:

1. Calibration 2B used the legacy clean salt values for B_1, B_2, q, and $\bar{\sigma}_{\mathrm{g}}$ because all IfG creep tests utilized $\bar{\sigma}$ values that did not appear to activate steady-state mechanism 3. According to Munson et al. (1989), steady-state mechanism 3 begins to be activated at $\bar{\sigma}_{\mathrm{g}} = 20.57$ MPa, yet $\bar{\sigma} \leq 18$ MPa for all IfG creep tests. Furthermore, $\log_{10} \dot{\bar{\varepsilon}}^{\mathrm{ss}}$ vs. $\log_{10} \bar{\sigma}$ looks reasonably linear for 8 MPa$\leq \bar{\sigma} \leq$18 MPa in Fig. 5.1b, which confirms that $\bar{\sigma}_{\mathrm{g}} \geq 18$ MPa.



(a) Temperature dependence



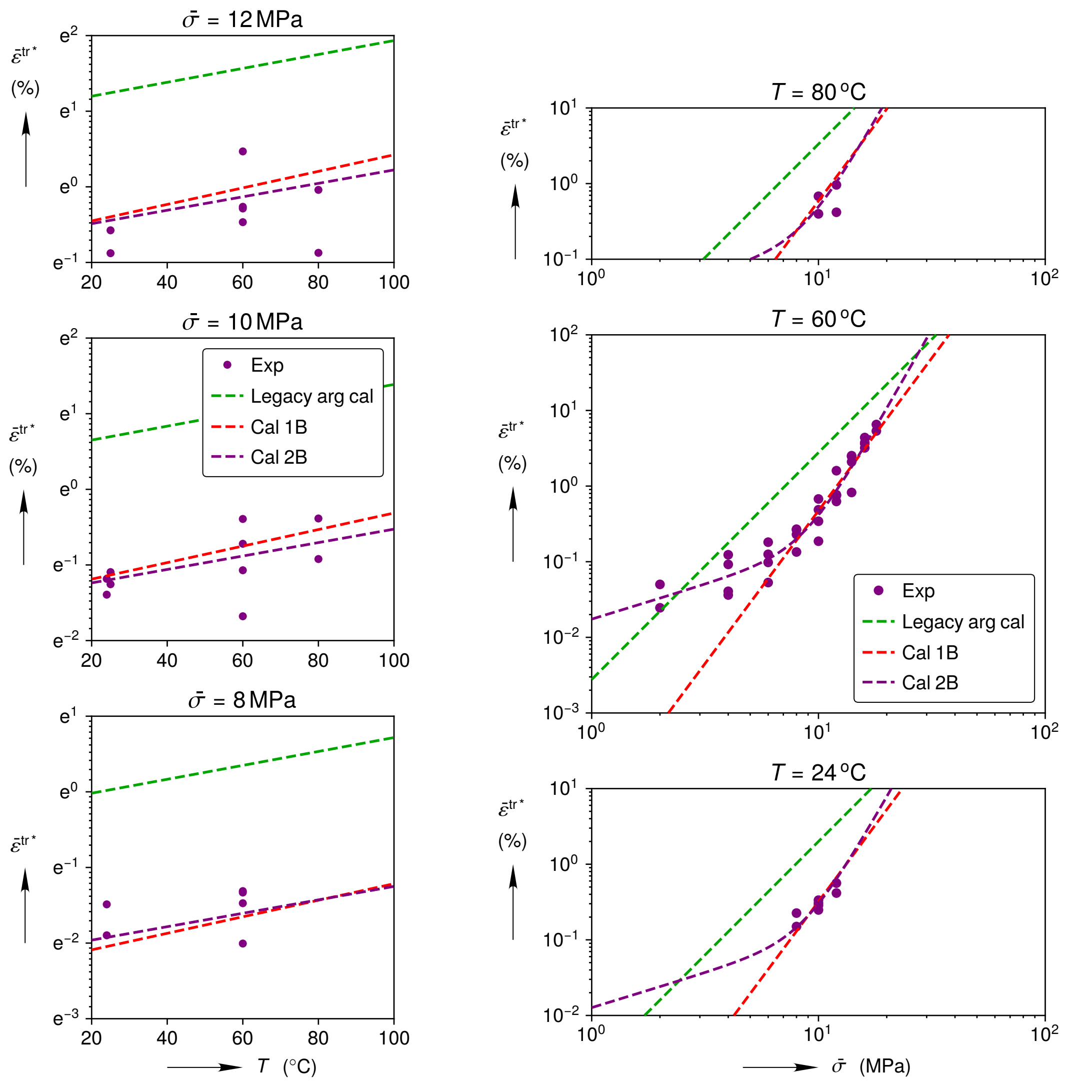
(b) Stress dependence

Figure 5.1: MD model calibrations of the steady-state equivalent strain rate compared against creep experiments.

2. B_0 was set to 0 because steady-state mechanism 0's contribution was negligible for $\bar{\sigma} \geq \bar{\sigma}_g = 20.57$ MPa (see Fig. 2.3a).

3. The temperature range tested by the IfG was wide enough to detect a non-linear dependence in the $\ln \dot{\bar{\varepsilon}}^{ss}$ versus $1/T$ plots in Fig. 5.1a, but not wide enough to properly calibrate mechanism 1, which dominates at higher temperatures. In the clean salt legacy calibration, mechanism 1 begins to contribute above roughly 80 °C, so Calibration 2B adopted the legacy clean salt mechanism 1 values of A_1, Q_1/R, and n_1.

4. The IfG tests only exercised the low stress transient and steady-state mechanisms at $T = 60$ °C, so the low stress mechanisms were assumed to have the same temperature



(a) Temperature dependence (b) Stress dependence

Figure 5.2: MD model calibrations of transient equivalent strain limit compared against creep experiments.

dependence as their medium stress counterparts. In other words, $Q_0 = Q_2$ and $c_0 = c_1$ during the calibration.

Steady-state mechanisms 0 and 2 were calibrated against the $T \leq 60$ °C experimental data in Fig. 5.1. The Nelder-Mead algorithm (Nelder and Mead 1965) was used to iteratively optimize new values of A_0, n_0, A_2, Q_2, and n_2 that minimized the following objective function

$$r = \sum_{j=1}^{J} \left[w^{(j)} \left(\ln \dot{\bar{\varepsilon}}^{ss(j)}_{exp} - \ln \dot{\bar{\varepsilon}}^{ss(j)}_{sim} \right) + w^{(j)} \left(\log_{10} \dot{\bar{\varepsilon}}^{ss(j)}_{exp} - \log_{10} \dot{\bar{\varepsilon}}^{ss(j)}_{sim} \right) \right]^2, \tag{5.1}$$

where J is the total number of test segments, $\dot{\bar{\varepsilon}}^{ss(j)}_{exp}$ is the experimental measurement of $\dot{\bar{\varepsilon}}^{ss}$ in test segment j, and $\dot{\bar{\varepsilon}}^{\mathrm{ss}(j)}_{\mathrm{sim}}$ is the simulated value for the same T and $\bar{\sigma}$ as test segment j. The objective function in Eq. (5.1) utilized logarithms with two different bases in order to place equal importance on fit quality in $\ln \dot{\bar{\varepsilon}}^{\mathrm{ss}}$ versus $1/T$ space and $\log_{10} \dot{\bar{\varepsilon}}^{\mathrm{ss}}$ versus $\log_{10} \bar{\sigma}$ space. Although not shown here, $\dot{\bar{\varepsilon}}^{\mathrm{ss}}_0 + \dot{\bar{\varepsilon}}^{\mathrm{ss}}_2$ was compared against $\dot{\bar{\varepsilon}}^{\mathrm{ss}}_1$ after the fitting operation to verify that mechanism 1 had a very small contribution to $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$ for $T \leq 60$ °C.

The resulting Calibration 2B is shown in Fig. 5.1, and compared against the legacy argillaceous salt calibration (Munson et al. 1989) and Calibration 1B (Reedlunn 2016). As expected, Calibration 2B does not represent the data at $T > 60$ °C, but it does represent the $T \leq 60$ °C data well. The slope $n_2 = 6.279$ is steeper than the legacy $n_2 = 5.0$ and the height of the mechanism 2 dominated region is lower. For example, Calibration 2B predicts a 2.6× smaller steady-state creep rate than the legacy argillaceous calibration at $\bar{\sigma} = 10$ MPa and $T = 24$ °C. On the other hand, Calibration 2B predicts far larger steady-state rates at low stresses. At $\bar{\sigma} = 3$ MPa and $T = 24$ °C, Calibration 2B's $\dot{\bar{\varepsilon}}^{\mathrm{ss}}$ is 6.5× and 24× larger those predicted by the legacy calibration and Calibration 1A, respectively.

The transient strain limit parameters K_0, m_0, K_1, c_1, and m_1 were calibrated in the same manner as the steady-state mechanism 0 and 2 parameters. The experimental values above 60 °C were ignored and an analogous objective function to Eq. (5.1) was utilized. (The $w^{(j)}$ and $\bar{\varepsilon}^{\mathrm{tr}(j)}_{\mathrm{exp}}$ values are also listed in Reedlunn (2016), Table A.2.)

The resulting Calibration 2B captures the transient strain limit data at $T \leq 60$ °C data well, including the data at $\bar{\sigma} < 8$ MPa (see Fig. 5.2). At $\bar{\sigma} = 3$ MPa and $T = 24$ °C, the Calibration 2B transient limit is 14× larger than that predicted by Calibration 1B, which ignores the low stress behavior. Equally striking, though, is the poor match between the legacy argillaceous salt calibration and the experimental data. To compare, Calibration 2B, which does match the data, predicts a 6.6× smaller transient strain limit than the legacy argillaceous calibration at $\bar{\sigma} = 10$ MPa and $T = 24$ °C. (See Section 1.4.2, 4.1.4, and 4.1.5 of Reedlunn (2016) for further discussion of the legacy salt tests and calibrations.)

5.2 Hosford Exponent

The MD model originally used a von Mises equivalent stress, without any facets or sharp corners, but the equivalent stress measure was changed to Tresca in Munson et al. (1989). As shown in the π-plane plot in Fig. 5.3c, the maximum difference between these two equivalent stress measures is only 15.5 %, so one might not expect a large impact on structural simulations. This difference, however, gets amplified by the exponents in Eqs. (2.9) and (2.16). For example, typically $n_2 \approx 5$, so a 15.5 % increase in $\bar{\sigma}$ causes a 2.05× increase in steady state mechanism 2. Munson et al. (1989) justified the switch to the Tresca equivalent stress by inspecting measurements on hollow cylinders of salt subjected to axial compression, internal pressurization, and external pressurization (Mellegard et al. 1992). Here, the same hollow cylinder experiments are re-analyzed and used to fit the exponent a in Eq. (2.5).

The Mellegard et al. (1992) hollow cylinders were fabricated out of cores extracted from the 150 m level at the International Salt Company Mine in Avery Island, Lousiana. Ideally the cores would have been extracted from the WIPP site, but something as fundamental as the equivalent stress measure should not change from location to location, so the Avery Island cores were deemed sufficient. The 330 mm diameter cores were machined into 610 mm long hollow cylinders with a 254 mm inside diameter and a 305 mm outside diameter. This corresponds to a 25.4 mm wall thickness and a 5.5 mean radius-to-wall thickness ratio. They felt this was thin enough to neglect stress and strain gradients through the thickness, and therefore analyze their results as true triaxial compression at a material point. The mean wall thickness-to-grain size ratio was approximately 3.5. They concluded that 3.5 grains through the thickness was adequate based on the uniformity of diametral deformation around the circumference during the creep experiments.

Each experiment started with a hydrostatic pressure of 18.33 MPa for three days. At $t = t_0 = 0$, the Cauchy radial stress σ_{rr}, hoop stress $\sigma_{\theta\theta}$, and axial stress σ_{zz} were raised to specific values and held constant in time (see top plot in Fig. 5.3a). The values were chosen to achieve a certain von Mises stress $\bar{\sigma}_{\mathrm{vm}}$ and Lode angle ψ, which is defined as

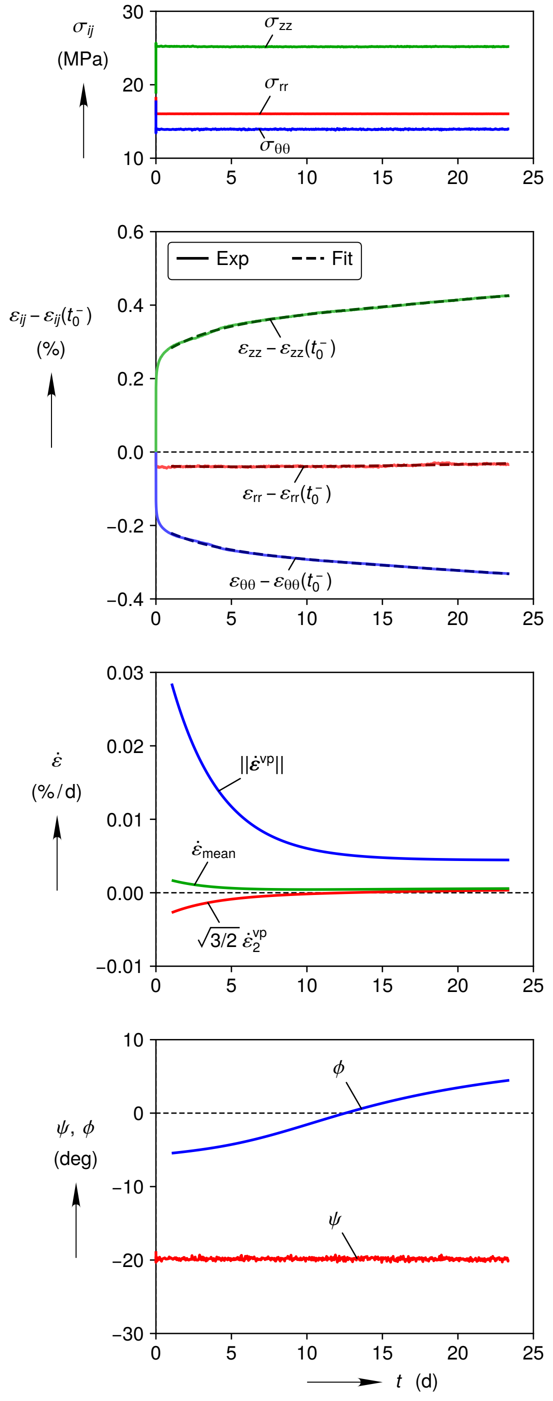
$$\sin\psi = \frac{\sqrt{3/2}\,\sigma_2^{\mathrm{dev}}}{||\boldsymbol{\sigma}^{\mathrm{dev}}||_2} \tag{5.2}$$

where $||\cdot||_2$ is the L_2 (Euclidean) norm of a quantity and σ_2^{dev} is the intermediate principal deviatoric stress. Let t_0^- and t_0^+ be the instant in time immediately before and after t_0, respectively. The change in radial log strain $\varepsilon_{\mathrm{rr}} - \varepsilon_{\mathrm{rr}}(t_0^-)$, hoop log strain $\varepsilon_{\theta\theta} - \varepsilon_{\theta\theta}(t_0^-)$, and axial log strain $\varepsilon_{\mathrm{zz}} - \varepsilon_{\mathrm{zz}}(t_0^-)$ resulting from the change in stress at t_0 were measured and reported (see second from top plot in Fig. 5.3a).

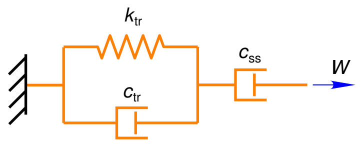
Mellegard et al. (1992) originally planned to use the variation in the strain rate with respect to ψ to distinguish between the von Mises and Tresca flow potentials. The cores, however, were extracted from two different sites at Avery Island, and one site exhibited significantly higher strain rates than the other site, even for the same ψ. Instead, Mellegard et al. (1992) elected to use the angle ϕ of the viscoplastic strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$ to distinguish between the two flow potentials. The angle ϕ is defined as

$$\sin\phi = \frac{\sqrt{3/2}\,\dot{\varepsilon}_2^{\mathrm{vp}}}{||\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}||_2} \tag{5.3}$$

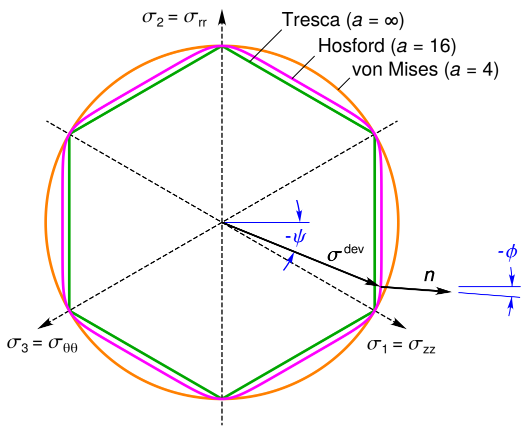
where $\dot{\varepsilon}_2^{\mathrm{vp}}$ is the intermediate principal viscoplastic strain rate. As previously mentioned, the flow rule (Eq. (2.6)) causes $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$ to be coaxial with the flow potential normal $\boldsymbol{n} = \partial\bar{\sigma}/\partial\boldsymbol{\sigma}$, therefore the angle ϕ is also the angle of $\boldsymbol{n}$. Consider Fig. 5.3c and the dashed lines in Fig. 5.3d (ignore the experimental data for now). For a von Mises flow potential, $\boldsymbol{n}$ is coaxial with the deviatoric stress $\boldsymbol{\sigma}^{\mathrm{dev}}$, such that $\psi = \phi$. For a Tresca flow potential, $\phi = 0°$ for -30°< ψ <30°, and $\phi = \pm 30°$ for $\psi = \pm 30°$. For something in-between von Mises and Tresca, ϕ varies nonlinearly for -30°< ψ <30°. Mellegard et al. (1992) applied seven different values of ψ, measured $\dot{\boldsymbol{\varepsilon}}^{\mathrm{vp}}$ at the end of each experiment, and calculated seven (ψ, ϕ) value pairs. They concluded the Tresca equivalent stress fit the data better than the von Mises equivalent stress.



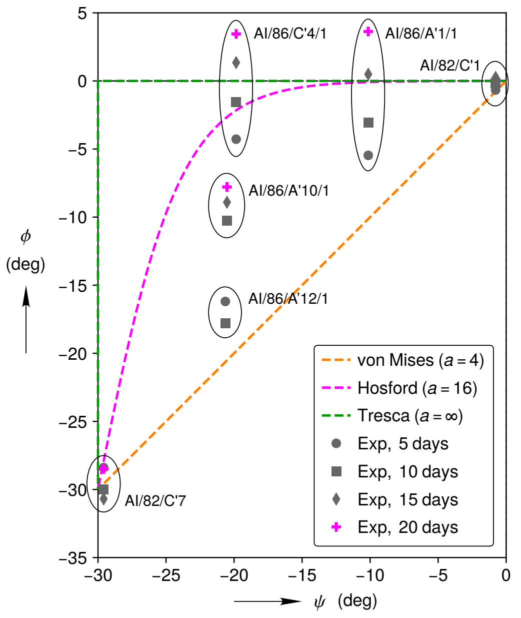
(a) Analysis of hollow cylinder experiment AI/86/C'4/1



(b) Kelvin-Voigt Element in series with a dashpot



(c) Hosford surfaces in the π-plane



(d) Hosford exponent a fit

Figure 5.3: Determination of the Hosford exponent a

Here, the (ψ, ϕ) value pairs were calculated again from six of the seven experiments in order to fit a. The analysis of AI/86/C'4/1 is shown in Fig. 5.3a to help visualize the following discussion, but similar plots can be found in Figs. D.1 to D.4 for all seven experiments. (Experiment AI/86/C'3/1 in Fig. D.4 was discarded due to issues with controlling the stresses.) The Lode angle ψ was calculated as a function of t by plugging the raw values of the principal deviatoric stresses into Eq. (5.2). Next, each $\varepsilon_{ij} - \varepsilon_{ij}(t_0^-)$ vs. t curve was fit with a simple function to provide a smooth curve for temporal derivatives. The simple function was constructed from a Kelvin-Voigt element for the transient response (spring constant k_{tr} and dashpot constant c_{tr}) in series with a dashpot (dashpot constant c_{ss}) for the steady-state response (see Fig. 5.3b). These elements respond to a step change in load W with a displacement $u(t)$ of the form

$$u(t) = p_0 \left[1 - \exp\left(-\frac{t - p_1}{p_2}\right)\right] + p_3\,(t - p_1), \tag{5.4}$$

where $p_0 = W/k_{\text{tr}}$, p_1 is the time when W is applied, $p_2 = c_{\text{tr}}/k_{\text{tr}}$, and $p_3 = c_{\text{ss}}$. The function $u(t)$ was fit to each $\varepsilon_{ij} - \varepsilon_{ij}(t_0^-)$ vs. t curve, neglecting the first day of creep, by varying the parameters p_j. As shown in the second from top plot in Fig. 5.3a, $u(t)$ fit the curves quite well. The fits were then differentiated with respect to time to obtain $\dot{\boldsymbol{\varepsilon}}$. Provided sufficient confining pressure has been applied, the mean strain rate $\dot{\varepsilon}_{\text{mean}} = (\dot{\varepsilon}_1 + \dot{\varepsilon}_2 + \dot{\varepsilon}_3)/3$ during creep of rock salt should ideally be zero. The mean strain rate was small compared to $||\dot{\boldsymbol{\varepsilon}}||$, but not small compared to $\dot{\varepsilon}_2$ (or $\dot{\varepsilon}_2^{\text{vp}}$), as shown in Fig. 5.3a. This non-zero mean strain rate was assumed to be due to measurement error, so the viscoplastic strain rate tensor $\dot{\boldsymbol{\varepsilon}}^{\text{vp}}$ was forced to be isochoric by subtracting off the mean strain rate:

$$\dot{\boldsymbol{\varepsilon}}^{\text{vp}} = \dot{\boldsymbol{\varepsilon}} - \dot{\varepsilon}_{\text{mean}}\,\boldsymbol{I}. \tag{5.5}$$

Once $\dot{\boldsymbol{\varepsilon}}^{\text{vp}}$ was computed, ϕ was calculated using Eq. (5.3).

The values of ψ and ϕ are plotted at five day intervals for each experiment in Fig. 5.3d. Multiple values from a given experiment are grouped together and labeled with the experiment ID to aid the reader's eye. The results are reasonably similar to those in Mellegard et al. (1992), except that one can now see how ϕ evolves with t, instead of a single value per experiment. Unfortunately, a significant amount of variation exists at $\psi = 10°$ and $20°$. The numerator $\dot{\varepsilon}_2^{\text{vp}}$ in Eq. (5.3) corresponds to $\dot{\varepsilon}_{\text{rr}}$ in these experiments, and ε_{rr} is difficult to accurately measure because it involves the difference between the inner and outer diameter of the hollow cylinder. Furthermore, in experiment AI/86/A'12/1, the magnitude of $\dot{\varepsilon}_{\text{rr}}$ mysteriously increased (instead of slowly decreasing) at $t = 12.5$ days (see Fig. D.2b). Mellegard et al. (1992) noted this anomaly, but they did not determine the cause. Experiment AI/86/A'12/1 ended at $t = 16.9$ days.

Despite these issues, the author is not aware of a better experimental data set to calibrate a against for Calibration 3B. For a given value of a, the Hosford flow potential normal $\boldsymbol{n}$ was computed using Eqs. (B.4) to (B.6) and its angle was computed using

$$\sin\phi = \frac{\sqrt{3/2}\,n_2}{||\boldsymbol{n}||_2}. \tag{5.6}$$

A least squares fit of a against the $t = 20$ day experimental data set, which does not include AI/86/A'12/1, resulted in $a = 16$. The $a = 16$ surface is plotted in Figs. 5.3c and 5.3d for reference.

5.3 Calibration Summary

To end this section, the legacy clean salt and legacy argillaceous salt calibrations are listed next to three recent calibrations in Table 5.1. The legacy calibrations are the same as those listed in Munson et al. (1989) and Butcher (1997). The colored parameters indicate if a value deviates from both the legacy clean salt calibration and the preceding calibration from which it was derived. (See the end of Section 3.4 in Reedlunn (2016) for a comment about $K_1 = 2.470 \times 10^6$ in the legacy argillaceous salt calibration. Also note that K_1 was assigned the variable name K_0 in Reedlunn (2016).)

The MD model calibration for WIPP salt has evolved over the years and it is not always easy to track down how each parameter value was selected. In an effort to have a traceable parameter set, Table 5.2 lists Calibration 3B (the most recent calibration) with a reference for each material parameter. The steady-state mechanism 1 (high temperature behavior), steady-state mechansim 3 (high stresses), and work hardening recovery have not been re-calibrated since the 1980's. These legacy parameters were calibrated against laboratory experiments on WIPP salt, but the legacy and IfG experiments did not follow exactly the same procedures. For example, the legacy experiments did not include a 10 day hydrostatic consolidation phase. Furthermore, it is not entirely clear to the present author how Munson et al. (1989) selected parameters such as A_1 or α_r from looking at Section 2.3.3.2 of Munson et al. (1989). Caution is, therefore, recommended when using Calibration 3B in a simulation where a quantity of interest depends heavily on these legacy parameters.

Table 5.1: Munson-Dawson Calibrations. (Colors designate where a calibration deviated from both the legacy clean salt and the preceding calibration.)

Parameter	Units	Legacy Clean Salt	Legacy Arg. Salt	Cal 1B	Cal 2B	Cal 3B
μ	GPa	12.4	12.4	12.4	12.4	12.4
B	GPa	20.7	20.7	20.7	20.7	20.7
a	–	∞	∞	∞	∞	16.0
A_0	s^{-1}	0	0	0	5.617×10^{1}	5.617×10^{1}
Q_0/R	K	0	0	0	5123	5123
n_0	–	0	0	0	1.595	1.595
A_1	s^{-1}	8.386×10^{22}	1.407×10^{23}	8.386×10^{22}	8.386×10^{22}	8.386×10^{22}
Q_1/R	K	12580.5	12580.5	12580.5	12580.5	12580.5
n_1	–	5.5	5.5	5.5	5.5	5.5
A_2	s^{-1}	9.672×10^{12}	1.314×10^{13}	1.074×10^{14}	4.415×10^{16}	4.415×10^{16}
Q_2/R	K	5032.2	5032.2	5177	5123	5123
n_2	–	5.0	5.0	5.353	6.279	6.279
$\bar{\sigma}_{\mathrm{g}}$	MPa	20.57	20.57	20.57	20.57	20.57
B_0	s^{-1}	0	0	0	0	0
B_1	s^{-1}	6.086×10^{6}	8.998×10^{6}	6.086×10^{6}	6.086×10^{6}	6.086×10^{6}
B_2	s^{-1}	3.034×10^{-2}	4.289×10^{-2}	3.034×10^{-2}	3.034×10^{-2}	3.034×10^{-2}
q	–	5335	5335	5335	5335	5335
K_0	–	0	0	0	5.277×10^{-2}	5.277×10^{-2}
c_0	K^{-1}	0	0	0	8.882×10^{-3}	8.882×10^{-3}
m_0	–	0	0	0	0.9201	0.9201
K_1	–	6.275×10^{5}	2.470×10^{6}	3.918×10^{8}	3.052×10^{12}	3.052×10^{12}
c_1	K^{-1}	9.198×10^{-3}	9.198×10^{-3}	1.093×10^{-2}	8.882×10^{-3}	8.882×10^{-3}
m_1	–	3.0	3.0	4.041	5.282	5.282
α_{h}	–	-17.37	-14.96	3.367	3.367	3.367
β_{h}	–	-7.738	-7.738	-0.6838	-0.6838	-0.6838
α_{r}	–	0.58	0.58	0.58	0.58	0.58
β_{r}	–	0.0	0.0	0.0	0.0	0.0
α	K^{-1}	45.0×10^{-6}	45.0×10^{-6}	45×10^{-6}	45×10^{-6}	45×10^{-6}
ρ	$\mathrm{kg/m}^3$	2300	2300	2300	2300	2300

Table 5.2: Munson-Dawson Calibration 3B

Parameter	Units	Value	Reference
μ	GPa	12.4	Munson et al. (1989), Section 2.3.3.1
B	GPa	20.7	Munson et al. (1989), Section 2.3.3.1
a	–	16.0	Herein
A_0	s^{-1}	5.617×10^{1}	Herein
Q_0/R	K	5123	Herein
n_0	–	1.595	Herein
A_1	s^{-1}	8.386×10^{22}	Munson et al. (1989), Section 2.3.3.2
Q_1/R	K	12580.5	Munson and Dawson (1979), Section III
n_1	–	5.5	Munson and Dawson (1979), Section III
A_2	s^{-1}	4.415×10^{16}	Herein
Q_2/R	K	5123	Herein
n_2	–	6.279	Herein
$\bar{\sigma}_{\mathrm{g}}$	MPa	20.57	Munson et al. (1989), Section 2.3.3.2
B_0	s^{-1}	0	Herein
B_1	s^{-1}	6.086×10^{6}	Munson et al. (1989), Section 2.3.3.2
B_2	s^{-1}	3.034×10^{-2}	Munson et al. (1989), Section 2.3.3.2
q	–	5335	Munson et al. (1989), Section 2.3.3.2
K_0	–	5.277×10^{-2}	Herein
c_0	K^{-1}	8.882×10^{-3}	Herein
m_0	–	0.9201	Herein
K_1	–	3.052×10^{12}	Herein
c_1	K^{-1}	8.882×10^{-3}	Herein
m_1	–	5.282	Herein
α_{h}	–	3.367	Reedlunn (2016), Section 3.4
β_{h}	–	-0.6838	Reedlunn (2016), Section 3.4
α_{r}	–	0.58	Munson et al. (1989), Section 2.3.3.2
β_{r}	–	0.0	Munson et al. (1989), Section 2.3.3.2
α	K^{-1}	45×10^{-6}	Krieg (1984), Section IV
ρ	$\mathrm{kg/m}^3$	2300	Krieg (1984), Section IV

Chapter 6

Summary

The MD model is a hypoelastic, unified viscoplastic, material model for the thermomechanical behavior of rock salt. It was developed in the 1980's, but the core viscoplastic formulation has not been modified until now. This report documents three new enhancements:

1. The model can now capture the most recent observations of creep behavior at low equivalent stresses ($<$8 MPa).
2. The equivalent stress measure was generalized to the Hosford stress, which can be adjusted to represent the Tresca stress, von Mises stress, or a range of behaviors in-between.
3. The implicit integration scheme was recreated from scratch, adding a line search algorithm, to make the model substantially more robust.

In addition to the enhancements, the new numerical implementation was fully documented and verified against three analytical solutions. The verification tests consisted of the strain response histories due to specific applied stress and temperature histories at a material point. To further test the implementation, it was also benchmarked against an old implementation on a representative initial boundary value problem, a room closure simulation. The new implementation gave virtually the same room closure curve and equivalent stress fields, yet finished the simulation 16$\times$ faster due to the robustness improvements. Finally, the new model features were recalibrated against laboratory experimental data. Preliminary simulations of room closure using the new calibrations show improved agreement with underground measurements (Reedlunn 2018).

Bibliography

Bérest, P., Béraud, J. F., Gharbi, H., Brouard, B., and DeVries, K. (2015). “A very slow creep test on an Avery Island salt sample”. In: *Rock Mechanics and Rock Engineering* 48.6, pp. 2591–2602.

Bérest, P., Blum, P. A., Charpentier, J. P., Gharbi, H., and Valès, F. (2005). “Very slow creep tests on rock samples”. In: *International Journal of Rock Mechanics and Mining Sciences* 42.4, pp. 569–576.

Brepols, T., Vladimirov, I. N., and Reese, S. (2014). “Numerical comparison of isotropic hypo- and hyperelastic-based plasticity models with application to industrial forming processes”. In: *International Journal of Plasticity* 63, pp. 18–48.

Butcher, B. M. (1997). *A summary of the sources of input parameter values for the Waste Isolation Pilot Plant final porosity surface calculations.* Tech. rep. SAND97-0796. Albuquerque, NM, USA: Sandia National Laboratories.

Callahan, G. D. (1999). *Crushed salt constitutive model.* Tech. rep. SAND98-2680. Albuquerque, NM, USA: Sandia National Laboratories.

Chadwick, P. and Ogden, R. (1971). “A theorem of tensor calculus and its application to isotropic elasticity”. In: *Archive for Rational Mechanics and Analysis* 44.1, pp. 54–68.

Chan, K. S., Bodner, S. R., and Munson, D. E. (2001). “Permeability of WIPP salt during damage evolution and healing”. In: *International Journal of Damage Mechanics* 10.4, pp. 347–375.

Düsterloh, U., Herchen, K., Lux, K.-H., Salzer, K., Günther, R.-M., Minkley, W., Hampel, A., Argüello Jr, J. G., and Hansen, F. D. (2015). “Joint Project III on the comparison of constitutive models for the thermomechanical behavior of rock salt. III. Extensive laboratory test program with argillaceous salt from WIPP and comparison of test results”. In: *Proc. 8th Conference on the Mechanical Behavior of Salt.* Ed. by L. Roberts, K. D. Mellegard, and F. D. Hansen, pp. 13–21.

Günther, R.-M., Salzer, K., Popp, T., and Lüdeling, C. (2014). “Steady State-Creep of Rock Salt-Improved Approaches for Lab Determination and Modeling to Describe Transient, Stationary and Accelerated Creep, Dilatancy and Healing”. In: *48th US Rock Mechanics/Geomechanics Symposium.* American Rock Mechanics Association.

Hansen, F. D. (2014). “Micromechanics of Isochoric Salt Deformation”. In: *48th US Rock Mechanics/Geomechanics Symposium.* American Rock Mechanics Association.

Hosford, W. (1972). “A generalized isotropic yield criterion”. In: *Journal of Applied Mechanics* 39.2, pp. 607–609.

Krieg, R. D. (1984). *Reference stratigraphy and rock properties for the Waste Isolation Pilot Plant (WIPP) project.* Tech. rep. SAND83-1908. Albuquerque, NM, USA: Sandia National Laboratories.

Mellegard, K. D., Callahan, G. D., and Senseny, P. E. (1992). *Multiaxial creep of natural rock salt*. Tech. rep. SAND91-7083. Sandia National Laboratories, Albuquerque, NM, USA; RE/SPEC, Inc., Rapid City, SD, USA.

Munson, D. E. and Dawson, P. (1979). *Constitutive model for the low temperature creep of salt (with application to WIPP)*. Tech. rep. SAND79-1853. Albuquerque, NM, USA: Sandia National Laboratories.

Munson, D. E. and Dawson, P. (1982). *Transient creep model for salt during stress loading and unloading*. Tech. rep. SAND82-0962. Albuquerque, NM, USA: Sandia National Laboratories.

Munson, D. E., Fossum, A. F., and Senseny, P. E. (1989). *Advances in resolution of discrepancies between predicted and measured in situ WIPP room closures*. Tech. rep. SAND88-2948. Albuquerque, NM, USA: Sandia National Laboratories.

Nelder, J. A. and Mead, R. (1965). "A simplex method for function minimization". In: *The computer journal* 7.4, pp. 308–313.

Pérez-Foguet, A. and Armero, F. (2002). "On the formulation of closest-point projection algorithms in elastoplasticity—part II: Globally convergent schemes". In: *International Journal for numerical Methods in Engineering* 53.2, pp. 331–374.

Reedlunn, B. (2016). *Reinvestigation into Closure Predictions of Room D at the Waste Isolation Pilot Plant*. Tech. rep. SAND2016-9961. Albuquerque, NM, USA: Sandia National Laboratories. DOI: 10.2172/1333709.

Reedlunn, B. (2018). "Joint Project III on the Comparison of Constitutive Models for the Mechanical Behavior of Rock Salt: Reinvestigation into Isothermal Room Closure Predictions at the Waste Isolation Pilot Plant". In: *The Mechanical Behavior of Salt IX*. Ed. by S. Fahland, J. Hammer, F. D. H. Hansen, S. Heusermann, K.-H. Lux, and W. Minkley. BGR (Federal Institute for Geosciences and Natural Resources). ISBN: 978-3-9814108-6-0.

Salzer, K., Günther, R.-M., Minkley, W., Naumann, D., Popp, T., Hampel, A., Lux, K.-H., Herchen, K., Düsterloh, U., Argüello Jr, J. G., and Hansen, F. D. (2015). "Joint Project III on the comparison of constitutive models for the thermomechanical behavior of rock salt. III. Extensive laboratory test program with clean salt from WIPP". In: *Proc. 8th Conference on the Mechanical Behavior of Salt*. Ed. by L. Roberts, K. D. Mellegard, and F. D. Hansen, pp. 3–12.

Scherzinger, W. M. (2017). "A return mapping algorithm for isotropic and anisotropic plasticity models using a line search method". In: *Computer Methods in Applied Mechanics and Engineering* 317, pp. 526–553.

Scherzinger, W. M. and Dohrmann, C. R. (2008). "A robust algorithm for finding the eigenvalues and eigenvectors of 3× 3 symmetric matrices". In: *Computer Methods in Applied Mechanics and Engineering* 197.45-48, pp. 4007–4015.

Scherzinger, W. M. and Lester, B. T. (2018). *Library of Advanced Materials for Engineering (LAME) 4.48*. Tech. rep. SAND2018-3231. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Sierra/Solid Mechanics (2017). *Sierra/Solid Mechanics User's Guide*. 4.46. SAND2017-9759. Sandia National Laboratories. Albuquerque, NM, USA; Livermore, CA, USA.

Sierra/Solid Mechanics (2018). *Sierra/Solid Mechanics User's Guide*. 4.50. SAND2018-10673. Sandia National Laboratories. Albuquerque, NM, USA; Livermore, CA, USA.

Simo, J. C. and Hughes, T. J. (1998). "Computational inelasticity". In: *New York*.

Weatherby, J., Munson, D. E., and Arg´uello, J. G. (1996). “Three-dimensional finite element simulation of creep deformation in rock salt”. In: *Engineering computations* 13.8, pp. 82–105. DOI: 10.1108/02644409610153023.

Appendix A

Notes on the Hosford Equivalent Stress

The Hosford equivalent stress is useful because one can vary the exponent a to capture the Tresca equivalent stress, the von Mises equivalent stress, and a range of behaviors in-between. The following subsections will demonstrate this statement mathematically and graphically.

A.1 Interpretation as a L_a Norm

The Hosford equivalent stress was originally presented as Eq. (2.5), but it can also be helpful to think of it as a L_a norm of the principal stress differences, multiplied by a prefactor. If one defines the components of the principal stress differences vector $\breve{\boldsymbol{\sigma}}$ as

$$\{\breve{\sigma}_i\} = \left\{ \begin{array}{c} \sigma_1 - \sigma_2 \\ \sigma_2 - \sigma_3 \\ \sigma_1 - \sigma_3 \end{array} \right\} = \left\{ \begin{array}{c} \sigma_1^{\mathrm{dev}} - \sigma_2^{\mathrm{dev}} \\ \sigma_2^{\mathrm{dev}} - \sigma_3^{\mathrm{dev}} \\ \sigma_1^{\mathrm{dev}} - \sigma_3^{\mathrm{dev}} \end{array} \right\}, \tag{A.1}$$

then one can re-express Eq. (2.5) as

$$\bar{\sigma} = \left(\frac{1}{2}\right)^{1/a} \left(\sum_{i=1}^{3} |\breve{\sigma}_i|^a\right)^{1/a} = \left(\frac{1}{2}\right)^{1/a} ||\breve{\boldsymbol{\sigma}}||_a, \tag{A.2}$$

where $||\breve{\boldsymbol{\sigma}}||_a$ is the L_a norm of $\breve{\boldsymbol{\sigma}}$.

A.2 Equivalence with Tresca

The Tresca stress is defined as

$$\bar{\sigma}_{\mathrm{t}} = \max\left(|\breve{\sigma}_i|\right). \tag{A.3}$$

This expression is invariant to the order of the principal stresses by design, but we can chose the order $\sigma_1 \geq \sigma_2 \geq \sigma_3$ for convenience. With this ordering, Eq. (A.3) becomes

$$\bar{\sigma}_{\mathrm{t}} = \sigma_1 - \sigma_3. \tag{A.4}$$

It will now be shown that the Hosford equivalent stress is equivalent to the Tresca stress if $a = 1$ and limits to the Tresca stress as $a \to \infty$.

A.2.1 Hosford, $a = 1$

The L_1 norm of a vector is the sum of the absolute values of the components. Accordingly, Eq. (A.2) reduces to

$$\bar{\sigma} = \frac{1}{2}||\breve{\boldsymbol{\sigma}}||_1 = \frac{1}{2}\sum_{i=1}^{3}|\breve{\sigma}_i| \tag{A.5}$$

when $a = 1$. The absolute value signs can be dropped by again chosing $\sigma_1 \geq \sigma_2 \geq \sigma_3$, so that Eq. (A.5) becomes

$$\bar{\sigma} = \sigma_1 - \sigma_3, \tag{A.6}$$

which is identical to the right hand side of Eq. (A.4).

A.2.2 Hosford, $a \to \infty$

The L_∞ norm of a vector is the maximum absolute value of all the components. To see this, let x_j be the component with the largest absolute value in a generic vector $\boldsymbol{x}$. Then $x_j^\infty \gg x_i^\infty$ for $i \neq j$, which means

$$||\boldsymbol{x}||_\infty = \left(\sum_{i=1}|x_i|^\infty\right)^{1/\infty} = \left(|x_j|^\infty\right)^{1/\infty} = |x_j| = \max\left(|x_i|\right). \tag{A.7}$$

If $\boldsymbol{x}$ has y components with the same absolute value $|x_j|$, then one still obtains the same result because $||\boldsymbol{x}||_\infty = \left(y\,|x_j|^\infty\right)^{1/\infty} = |x_j|$. Thus, the L_∞ norm, along with the ordering $\sigma_1 \geq \sigma_2 \geq \sigma_3$, allows us to conclude

$$\bar{\sigma} = \left(\frac{1}{2}\right)^{1/\infty}||\breve{\boldsymbol{\sigma}}||_\infty = \max\left(|\breve{\sigma}_i|\right) = \sigma_1 - \sigma_3, \tag{A.8}$$

which is again identical to the right hand side of Eq. (A.4).

A.3 Equivalence with von Mises

The von Mises stress was originally expressed in Eq. (3.2), but it can also be thought of as the Euclidean magnitude (the L_2 norm) of $\boldsymbol{\sigma}^{\text{dev}}$ multiplied by a prefactor:

$$\bar{\sigma}_{\text{vm}} = \sqrt{\frac{3}{2}}\,||\boldsymbol{\sigma}^{\text{dev}}||_2 \tag{A.9}$$

It will now be shown that the Hosford equivalent stress coincides with the von Mises stress if $a = 2$ or 4.

A.3.1 Hosford, $a = 2$

To show equivalence with the von Mises stress for $a = 2$, expand Eq. (A.2) to obtain

$$\begin{aligned}\bar{\sigma} &= \left(\frac{1}{2}\right)^{1/2} ||\breve{\boldsymbol{\sigma}}||_2 = \left(\frac{1}{2}\right)^{1/2} \left(\sum_{i=1}^{3} \breve{\sigma}_i^2\right)^{1/2} \\ &= \left[\left(\sigma_1^{\text{dev }2} + \sigma_2^{\text{dev }2} + \sigma_3^{\text{dev }2}\right) - \left(\sigma_1^{\text{dev}}\,\sigma_2^{\text{dev}} + \sigma_2^{\text{dev}}\,\sigma_3^{\text{dev}} + \sigma_3^{\text{dev}}\,\sigma_1^{\text{dev}}\right)\right]^{1/2}. \end{aligned} \tag{A.10}$$

One can now make use of the deviatoric nature of $\boldsymbol{\sigma}^{\text{dev}}$,

$$\left[\text{tr}\left(\boldsymbol{\sigma}^{\text{dev}}\right)\right]^2 = \left(\sigma_1^{\text{dev }2} + \sigma_2^{\text{dev }2} + \sigma_3^{\text{dev }2}\right) + 2\,\left(\sigma_1^{\text{dev}}\,\sigma_2^{\text{dev}} + \sigma_2^{\text{dev}}\,\sigma_3^{\text{dev}} + \sigma_3^{\text{dev}}\,\sigma_1^{\text{dev}}\right) = 0, \tag{A.11}$$

to simplify the expression in Eq. (A.10) to

$$\bar{\sigma} = \left[\frac{3}{2}\left(\sigma_1^{\text{dev }2} + \sigma_2^{\text{dev }2} + \sigma_3^{\text{dev }2}\right)\right]^{1/2} = \sqrt{\frac{3}{2}}\,||\boldsymbol{\sigma}^{\text{dev}}||_2, \tag{A.12}$$

which matches the right hand side of Eq. (A.9).

A.3.2 Hosford, $a = 4$

The proof of equivalence with the von Mises stress for $a = 4$ is similar to the proof for $a = 2$, but more tedious. Expand Eq. (A.2) to obtain

$$\begin{aligned}\bar{\sigma} &= \left(\frac{1}{2}\right)^{1/4} ||\breve{\boldsymbol{\sigma}}||_4 = \left(\frac{1}{2}\right)^{1/4} \left(\sum_{i=1}^{3} \breve{\sigma}_i^4\right)^{1/4} \\ &= \Big[\left(\sigma_1^{\text{dev }4} + \sigma_2^{\text{dev }4} + \sigma_3^{\text{dev }4}\right) \\ &\quad - 2\,\left(\sigma_1^{\text{dev }3}\,\sigma_2^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev }3} + \sigma_2^{\text{dev }3}\,\sigma_3^{\text{dev}} + \sigma_2^{\text{dev}}\,\sigma_3^{\text{dev }3} + \sigma_3^{\text{dev }3}\,\sigma_1^{\text{dev}} + \sigma_3^{\text{dev}}\,\sigma_1^{\text{dev }3}\right) \\ &\quad + 3\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right)\Big]^{1/4}. \end{aligned} \tag{A.13}$$

Make use of the traceless property of the deviatoric stress to state

$$\begin{aligned}\left[\text{tr}\left(\boldsymbol{\sigma}^{\text{dev}}\right)\right]^4 &= \left(\sigma_1^{\text{dev }4} + \sigma_2^{\text{dev }4} + \sigma_3^{\text{dev }4}\right) \\ &\quad + 4\,\left(\sigma_1^{\text{dev }3}\,\sigma_2^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev }3} + \sigma_2^{\text{dev }3}\,\sigma_3^{\text{dev}} + \sigma_2^{\text{dev}}\,\sigma_3^{\text{dev }3} + \sigma_1^{\text{dev }3}\,\sigma_3^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_3^{\text{dev }3}\right) \\ &\quad + 6\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right) \\ &\quad + 12\,\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev}}\,\sigma_3^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev}}\,\sigma_3^{\text{dev }2}\right) = 0. \end{aligned} \tag{A.14}$$

All terms with odd powers of the principal deviatoric stresses in these past two expressions can be replaced or eliminated upon recognizing that

$$\sigma_i^{\text{dev }3}\,\sigma_j^{\text{dev}} + \sigma_i^{\text{dev }3}\,\sigma_k^{\text{dev}} = \sigma_i^{\text{dev }3}\,\left(\sigma_j^{\text{dev}} + \sigma_k^{\text{dev}}\right) = -\sigma_i^{\text{dev }4} \tag{A.15}$$

for $i \neq j \neq k$, and recognizing that

$$\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev}}\,\sigma_3^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev}} + \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev}}\,\sigma_3^{\text{dev }2} = \sigma_1^{\text{dev}}\,\sigma_2^{\text{dev}}\,\sigma_3^{\text{dev}}\,\left(\sigma_1^{\text{dev}} + \sigma_2^{\text{dev}} + \sigma_3^{\text{dev}}\right) = 0. \tag{A.16}$$

Inserting Eq. (A.15) into Eq. (A.13) results in

$$\bar{\sigma} = \left[3\left(\sigma_1^{\text{dev }4} + \sigma_2^{\text{dev }4} + \sigma_3^{\text{dev }4}\right) + 3\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right)\right]^{1/4} \tag{A.17}$$

and substituting Eqs. (A.15) and (A.16) into Eq. (A.14) produces

$$-\left(\sigma_1^{\text{dev }4} + \sigma_2^{\text{dev }4} + \sigma_3^{\text{dev }4}\right) + 2\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right) = 0. \tag{A.18}$$

Now, one can combine Eqs. (A.17) and (A.18), and manipulate further to obtain

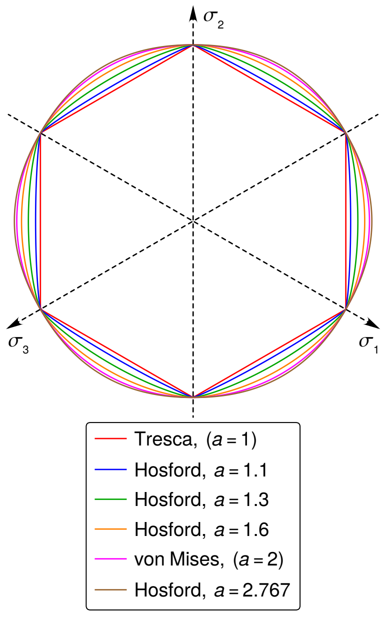
$$\begin{aligned}
\bar{\sigma} &= \left[9\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right)\right]^{1/4} \\
&= \left\{\frac{9}{4}\left[\left(\sigma_1^{\text{dev }4} + \sigma_2^{\text{dev }4} + \sigma_3^{\text{dev }4}\right) + 2\left(\sigma_1^{\text{dev }2}\,\sigma_2^{\text{dev }2} + \sigma_2^{\text{dev }2}\,\sigma_3^{\text{dev }2} + \sigma_1^{\text{dev }2}\,\sigma_3^{\text{dev }2}\right)\right]\right\}^{1/4} \\
&= \left[\left(\frac{3}{2}\right)^2\left(\sigma_1^{\text{dev }2} + \sigma_2^{\text{dev }2} + \sigma_3^{\text{dev }2}\right)^2\right]^{1/4} = \sqrt{\frac{3}{2}}\,||\boldsymbol{\sigma}^{\text{dev}}||_2\,,
\end{aligned} \tag{A.19}$$

which again agrees with the right hand side of Eq. (A.9).

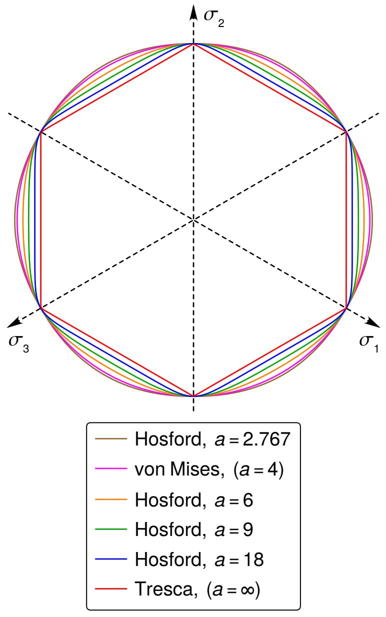
A.4 Behavior That Does Not Match Tresca or von Mises

Fig. A.1 depicts Hosford equivalent stress surfaces in the π-plane for selected values of a that fall in the range $1 \leq a \leq \infty$. (Hosford (1972) restricted the exponent to be $1 \leq a \leq \infty$ probably because $a < 1$ produces non-convex surfaces.) Between $a = 1$ and 2, the surface ranges between the Tresca and von Mises surfaces. Increasing a further causes the Hosford surface to slightly exceed the von Mises surface between $a = 2$ and ≈ 2.767, but then it returns to von Mises between $a \approx 2.767$ and 4. Between $a = 4$ and ∞, the surface again ranges between the Tresca and von Mises surfaces. Thus, a has no impact at the corners of the Tresca hexagon, but it clearly affects other stress states.

Pure shear (with some arbitrary amount of hydrostatic stress) is the most sensitive stress state to a, and this dependence is plotted in Fig. A.2a. In this plot, the pure shear stress τ where $\bar{\sigma} = 1$ is normalized by the pure shear stress where $\bar{\sigma}_{\text{t}} = 1$. Accordingly, $\tau/\tau_{\text{t}} = 1$ at $a = 1$ and limits to 1 as $a \to \infty$. Raising the exponent to sufficiently high values can lead to numerical issues, as mentioned in Section 3.1, so it is useful to know how quickly τ approaches τ_{t} for large finite values of a. At $a = 10^3$, τ is within 0.07 % of τ_{t}. The plot also exhibits the previously identified local maximum at $a \approx 2.767$. One can precisely compute this location by plugging in $\sigma_1 - \sigma_2 = \sigma_2 - \sigma_3$ into Eq. (2.5), taking a derivative, setting it to zero, and solving for a. That said, equivalent stress surfaces that lie outside the von Mises surface do not match experimental data, so the local maximum is of little use and further attention is restricted to the ranges $1 \leq a \leq 2$ and $4 \leq a \leq \infty$.

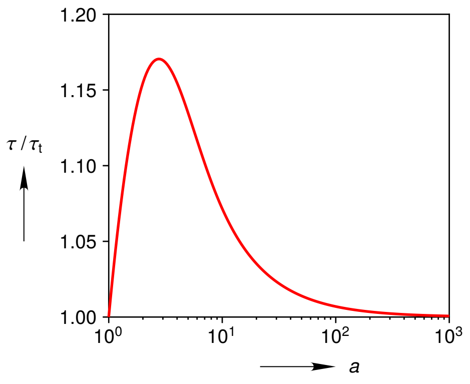


(a) $1 \leq a \leq 2.767$

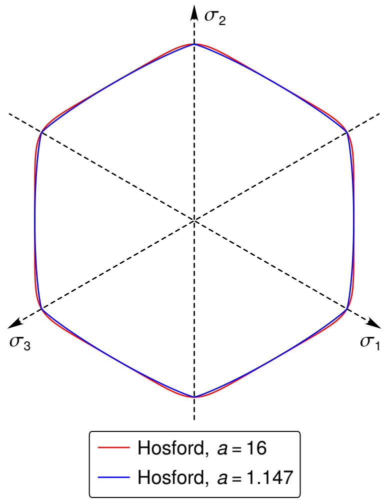


(b) $2.767 \leq a \leq \infty$

Figure A.1: Hosford equivalent stress surfaces in the π-plane for selected values of a.



(a) Pure shear stress magnitude



(b) Two surfaces with same equivalent stress in pure shear

Figure A.2: Investigation of Hosford equivalent stress in pure shear.

Figs. A.1 and A.2a seem to suggest that a mapping exists between the ranges $1 \leq a \leq 2$ and $4 \leq a \leq \infty$. Both ranges include Tresca and von Mises, they both span the space between Tresca and von Mises, and Fig. A.2a shows that two values of a produce the same τ/τ_t. Fig. A.2b, however, demonstrates that an exact mapping does not exist. The two surfaces with $a = 16$ and 1.147 have the same equivalent stress in pure shear and at the corners of the Tresca hexagon, but the surfaces differ slightly elsewhere. The slight differences between any pair of Hosford surfaces with the same equivalent stress in pure shear are considered small enough to focus solely on $4 \leq a \leq \infty$ for practical purposes. Appendix B shows that a potential singularity exists in the MD model numerical implementation if $1 \leq a \leq 2$, so it is advantageous to utilize $4 \leq a \leq \infty$ instead.

Appendix B

MD Model Derivatives

This appendix contains derivatives of select quantities in the MD model that do not appear explicitly in the main body of the report, but remain useful for model calibration and numerical implementation.

B.1 Derivatives of the Flow Potential

Recall that the MD model uses an associated flow rule, so the equivalent stress $\bar{\sigma}$ and the flow potential are one and the same.

B.1.1 Jacobian

The Jacobian of the flow potential is the normal to the flow potential surface. The normal is used to calibrate a in Section 5.2 and also utilized in the `md_viscoplastic` implementation. In the principal frame, the normal to the flow potential is

$$\frac{\partial \bar{\sigma}}{\partial \boldsymbol{\sigma}} = \sum_{i=1}^{3} \frac{\partial \bar{\sigma}}{\partial \sigma_i} \tilde{\boldsymbol{e}}_i \otimes \tilde{\boldsymbol{e}}_i \tag{B.1}$$

where, again, $\tilde{\boldsymbol{e}}_i$ are the principal stress directions[1]. One can take a straightforward derivative of Eq. (2.5) with respect to the first principal stress to obtain

$$\frac{\partial \bar{\sigma}}{\partial \sigma_1} = \frac{\bar{\sigma}}{2\,\bar{\sigma}^a} \left[\frac{|\sigma_1 - \sigma_2|^a}{\sigma_1 - \sigma_2} - \frac{|\sigma_3 - \sigma_1|^a}{\sigma_3 - \sigma_1} \right], \tag{B.2}$$

but this expression is susceptible to numerical overflow and underflow errors if a is large. Section 3.1 avoided numerical issues when calculating $\bar{\sigma}$ by normalizing the principal stresses by the von Mises stress $\bar{\sigma}_{\mathrm{vm}}$ to make σ_i close to unity. When calculating $\partial\bar{\sigma}/\partial\boldsymbol{\sigma}$, however, it is more convenient to normalize the principal stresses by $\bar{\sigma}$. After ensuring $\bar{\sigma} \geq \bar{\sigma}_{\min}$ via

[1]Although repeated indices are usually summed from 1 to 3 in indicial notation, the summation in Eq. (B.1) is explicitly stated because the index is repeated more than twice.

Eq. (3.4), substitute $\hat{\sigma}_i = \sigma_i/\bar{\sigma}$ into Eq. (B.2) to produce

$$\frac{\partial \bar{\sigma}}{\partial \sigma_1} = \frac{1}{2}\left[\frac{|\hat{\sigma}_1 - \hat{\sigma}_2|^a}{\hat{\sigma}_1 - \hat{\sigma}_2} - \frac{|\hat{\sigma}_3 - \hat{\sigma}_1|^a}{\hat{\sigma}_3 - \hat{\sigma}_1}\right]. \tag{B.3}$$

Equation (B.3) has a singularity if two principal stresses are equal, so $\partial\bar{\sigma}/\partial\sigma_1$ is actually computed as

$$\frac{\partial \bar{\sigma}}{\partial \sigma_1} = \frac{1}{2}\left[(\hat{\sigma}_1 - \hat{\sigma}_2)\,|\hat{\sigma}_1 - \hat{\sigma}_2|^{a-2} - (\hat{\sigma}_3 - \hat{\sigma}_1)\,|\hat{\sigma}_3 - \hat{\sigma}_1|^{a-2}\right], \tag{B.4}$$

and the Hosford exponent is restricted to $a \geq 2$. Similarly, the other two components of the flow potential normal are computed as

$$\frac{\partial \bar{\sigma}}{\partial \sigma_2} = \frac{1}{2}\left[-(\hat{\sigma}_1 - \hat{\sigma}_2)\,|\hat{\sigma}_1 - \hat{\sigma}_2|^{a-2} + (\hat{\sigma}_2 - \hat{\sigma}_3)\,|\hat{\sigma}_2 - \hat{\sigma}_3|^{a-2}\right] \tag{B.5}$$

$$\frac{\partial \bar{\sigma}}{\partial \sigma_3} = \frac{1}{2}\left[-(\hat{\sigma}_2 - \hat{\sigma}_3)\,|\hat{\sigma}_2 - \hat{\sigma}_3|^{a-2} + (\hat{\sigma}_3 - \hat{\sigma}_1)\,|\hat{\sigma}_3 - \hat{\sigma}_1|^{a-2}\right]. \tag{B.6}$$

B.1.2 Hessian

The Hessian of the flow potential

$$\frac{\partial^2 \bar{\sigma}}{\partial \boldsymbol{\sigma}^2} = \mathcal{H}_{ijkl}\,\tilde{\boldsymbol{e}}_i \otimes \tilde{\boldsymbol{e}}_j \otimes \tilde{\boldsymbol{e}}_k \otimes \tilde{\boldsymbol{e}}_l \tag{B.7}$$

is utilized in the `md_viscoplastic` implementation. (It is also useful to demonstrate convexity of the flow potential, but $\bar{\sigma}$ has been assumed to be convex without proof herein.) As shown in Chadwick and Ogden (1971), the non-zero components of the Hessian in the principal frame are

$$\mathcal{H}_{1111} = \frac{\partial^2 \bar{\sigma}}{\partial {\sigma_1}^2} \tag{B.8}$$

$$\mathcal{H}_{1122} = \frac{\partial^2 \bar{\sigma}}{\partial \sigma_1 \partial \sigma_2} \tag{B.9}$$

$$\mathcal{H}_{1212} = \frac{1}{2}\,\frac{\frac{\partial \bar{\sigma}}{\partial \sigma_1} - \frac{\partial \bar{\sigma}}{\partial \sigma_2}}{\sigma_1 - \sigma_2} \tag{B.10}$$

with the other non-zero components found by permuting the indices. If two principal stresses are equal, then

$$\mathcal{H}_{1212} = \lim_{\sigma_1 \to \sigma_2} \frac{1}{2}\,\frac{\frac{\partial \bar{\sigma}}{\partial \sigma_1} - \frac{\partial \bar{\sigma}}{\partial \sigma_2}}{\sigma_1 - \sigma_2} = \frac{1}{2}\left(\frac{\partial^2 \bar{\sigma}}{\partial {\sigma_1}^2} - \frac{\partial^2 \bar{\sigma}}{\partial \sigma_1 \partial \sigma_2}\right). \tag{B.11}$$

The expressions in Eqs. (B.10) and (B.11), however, are not needed herein, because the $\mathcal{H}_{ijij}$ terms, where $i \neq j$, only operate on shear stress and shear strains. As explained in Section 3.5, tensors remain in the principal frame throughout the Newton (and line search) iterations, so the `md_viscoplastic` implementation only requires the $\mathcal{H}_{iijj}$ terms, where i

may or may not equal j. To finish explicitly defining the Hessian, the second derivatives of the flow potential are

$$\frac{\partial^2\bar{\sigma}}{\partial{\sigma_1}^2} = \frac{a-1}{\bar{\sigma}}\left\{\frac{1}{2}\left[\frac{|\hat{\sigma}_1-\hat{\sigma}_2|^a}{(\hat{\sigma}_1-\hat{\sigma}_2)^2}+\frac{|\hat{\sigma}_3-\hat{\sigma}_1|^a}{(\hat{\sigma}_3-\hat{\sigma}_1)^2}\right]-\left(\frac{\partial\bar{\sigma}}{\partial\sigma_1}\right)^2\right\} \tag{B.12}$$

$$\frac{\partial^2\bar{\sigma}}{\partial\sigma_1\partial\sigma_2} = \frac{a-1}{\bar{\sigma}}\left[-\frac{1}{2}\frac{|\hat{\sigma}_1-\hat{\sigma}_2|^a}{(\hat{\sigma}_1-\hat{\sigma}_2)^2}-\frac{\partial\bar{\sigma}}{\partial\sigma_1}\frac{\partial\bar{\sigma}}{\partial\sigma_2}\right], \tag{B.13}$$

where the normalization $\hat{\sigma}_i = \sigma_i/\bar{\sigma}$ has been employed again. As before, Eqs. (B.12) and (B.13) have a singularity if two principal stresses are equal, so $\partial^2\bar{\sigma}/\partial{\sigma_1}^2$ and $\partial^2\bar{\sigma}/\left(\partial\sigma_1\partial\sigma_2\right)$ are actually computed as

$$\frac{\partial^2\bar{\sigma}}{\partial{\sigma_1}^2} = \frac{a-1}{\bar{\sigma}}\left\{\frac{1}{2}\left[|\hat{\sigma}_1-\hat{\sigma}_2|^{a-2}+|\hat{\sigma}_3-\hat{\sigma}_1|^{a-2}\right]-\left(\frac{\partial\bar{\sigma}}{\partial\sigma_1}\right)^2\right\} \tag{B.14}$$

$$\frac{\partial^2\bar{\sigma}}{\partial\sigma_1\partial\sigma_2} = \frac{a-1}{\bar{\sigma}}\left[-\frac{1}{2}|\hat{\sigma}_1-\hat{\sigma}_2|^{a-2}-\frac{\partial\bar{\sigma}}{\partial\sigma_1}\frac{\partial\bar{\sigma}}{\partial\sigma_2}\right] \tag{B.15}$$

and the Hosford exponent is restricted to $a \geq 2$. Similarly, the remaining second derivatives are

$$\frac{\partial^2\bar{\sigma}}{\partial{\sigma_2}^2} = \frac{a-1}{\bar{\sigma}}\left\{\frac{1}{2}\left[|\hat{\sigma}_1-\hat{\sigma}_2|^{a-2}+|\hat{\sigma}_3-\hat{\sigma}_2|^{a-2}\right]-\left(\frac{\partial\bar{\sigma}}{\partial\sigma_2}\right)^2\right\} \tag{B.16}$$

$$\frac{\partial^2\bar{\sigma}}{\partial{\sigma_3}^2} = \frac{a-1}{\bar{\sigma}}\left\{\frac{1}{2}\left[|\hat{\sigma}_1-\hat{\sigma}_3|^{a-2}+|\hat{\sigma}_3-\hat{\sigma}_2|^{a-2}\right]-\left(\frac{\partial\bar{\sigma}}{\partial\sigma_3}\right)^2\right\} \tag{B.17}$$

$$\frac{\partial^2\bar{\sigma}}{\partial\sigma_1\partial\sigma_3} = \frac{a-1}{\bar{\sigma}}\left[-\frac{1}{2}|\hat{\sigma}_1-\hat{\sigma}_3|^{a-2}-\frac{\partial\bar{\sigma}}{\partial\sigma_1}\frac{\partial\bar{\sigma}}{\partial\sigma_3}\right] \tag{B.18}$$

$$\frac{\partial^2\bar{\sigma}}{\partial\sigma_2\partial\sigma_3} = \frac{a-1}{\bar{\sigma}}\left[-\frac{1}{2}|\hat{\sigma}_2-\hat{\sigma}_3|^{a-2}-\frac{\partial\bar{\sigma}}{\partial\sigma_2}\frac{\partial\bar{\sigma}}{\partial\sigma_3}\right]. \tag{B.19}$$

B.2 Derivatives of the Discretized Transient Strain Differential Equation

Lastly, a few further derivatives used in Chapter 3 are explicitly defined. Equations (3.26), (3.28), (3.29) and (3.59) all utilize the following derivatives of the discretized transient strain ordinary differential equation:

$$\left(\frac{\partial r}{\partial\bar{\sigma}}\right)_n^{(k-1)} = \left\{\left[\dot{\bar{\varepsilon}}^{\mathrm{ss}}\frac{\partial F}{\partial\bar{\sigma}}+(F-1)\frac{\partial\dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial\bar{\sigma}}\right]\Delta t\right\}_n^{(k-1)} \tag{B.20}$$

$$\left(\frac{\partial r}{\partial\bar{\varepsilon}^{\mathrm{tr}}}\right)_n^{(k-1)} = \left(\Delta t\,\dot{\bar{\varepsilon}}^{\mathrm{ss}}\frac{\partial F}{\partial\bar{\varepsilon}^{\mathrm{tr}}}-1\right)_n^{(k-1)}. \tag{B.21}$$

Inside Eqs. (B.20) and (B.21) are several lower level derivatives. Those derivatives are

$$\begin{aligned}
\frac{\partial \dot{\bar{\varepsilon}}^{\mathrm{ss}}}{\partial \bar{\sigma}} =& \frac{s}{\mu} \left[\sum_{i=0}^{2} A_i \, \exp\left(-\frac{Q_i}{R\,T}\right) \, n_i \, \left(\frac{\bar{\sigma}}{\mu}\right)^{n_i - 1} \right. \\
& \left. + H(\bar{\sigma} - \bar{\sigma}_{\mathrm{g}}) \sum_{i=0}^{2} B_i \, \exp\left(-\frac{Q_i}{R\,T}\right) \cosh\left(q \frac{\bar{\sigma} - \bar{\sigma}_{\mathrm{g}}}{\mu}\right) \, q \right] \qquad \text{(B.22)} \\
\frac{\partial F}{\partial \bar{\varepsilon}^{\mathrm{tr}}} =& - \operatorname{sign}(\bar{\varepsilon}^{\mathrm{tr}*} - \bar{\varepsilon}^{\mathrm{tr}})^{\chi - 1} \, \frac{\kappa \, \chi \, F}{\bar{\varepsilon}^{\mathrm{tr}*}} \, \left(1 - \frac{\bar{\varepsilon}^{\mathrm{tr}}}{\bar{\varepsilon}^{\mathrm{tr}*}}\right)^{\chi - 1} \qquad \text{(B.23)} \\
\frac{\partial F}{\partial \bar{\sigma}} =& \operatorname{sign}(\bar{\varepsilon}^{\mathrm{tr}*} - \bar{\varepsilon}^{\mathrm{tr}})^{\chi - 1} \, F \, \left[\frac{\beta}{\ln(10)\,\bar{\sigma}} \, \left(1 - \frac{\bar{\varepsilon}^{\mathrm{tr}}}{\bar{\varepsilon}^{\mathrm{tr}*}}\right)^{\chi} \right. \\
& \left. + \frac{\chi \, \kappa \, \bar{\varepsilon}^{\mathrm{tr}}}{\mu \, \bar{\varepsilon}^{\mathrm{tr}*2}} \, \left(1 - \frac{\bar{\varepsilon}^{\mathrm{tr}}}{\bar{\varepsilon}^{\mathrm{tr}*}}\right)^{\chi - 1} \sum_{i=0}^{1} K_i \, \exp(c_i \, T) \, m_i \, \left(\frac{\bar{\sigma}}{\mu}\right)^{m_i - 1} \right] \qquad \text{(B.24)}
\end{aligned}$$

where

$$\beta = \begin{cases} \beta_{\mathrm{h}} & \bar{\varepsilon}^{\mathrm{tr}} \leq \bar{\varepsilon}^{\mathrm{tr}*} \\ \beta_{\mathrm{r}} & \bar{\varepsilon}^{\mathrm{tr}} > \bar{\varepsilon}^{\mathrm{tr}*}. \end{cases} \qquad \text{(B.25)}$$

Appendix C

md_viscoplastic C++ Code

This chapter displays the `md_viscoplastic` implementation in the C++ programming language. This source code is part of the Library of Advanced Materials for Engineering (LAME) (Scherzinger and Lester 2018), which is an integral part of Sierra/Solid Mechanics (2018). As a result, several programming constructs are not fully defined here, such as the material base class. On the other hand, the core of the `md_viscoplastic` implementation in Appendix C.2.2 should be reasonably self-contained and easy to follow when combined with Chapter 3.

This source code is licensed for public use under the terms of the MIT license. Hopefully, publishing the source code herein will enable porting of `md_viscoplastic` to other momentum balance codes.

Copyright 2018 National Technology and Engineering Solutions of Sandia, LLC. Under the terms of Contract DE-NA0003525, there is a non-exclusive license for use of this work by or on behalf of the U.S. Government.

C.1 Constructor

C.1.1 Header

```
#ifndef _MD_VISCOPLASTIC_H_
#define _MD_VISCOPLASTIC_H_

#include <models/Material.h>
#include <Lame_Fortran.h>

// md_viscoplastic
//
// Copyright 2018 National Technology & Engineering Solutions of Sandia, LLC (NTESS).
// Under the terms of Contract DE-NA0003525 with NTESS, the U.S. Government retains
// certain rights in this software.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this
// software and associated documentation files (the "Software"), to deal in the Software
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject to the following
// conditions:
//
// The above copyright notice and this permission notice shall be included in all copies
// or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
// CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
// OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

namespace lame {

  class MD_Viscoplastic : public Material {

  public:

    explicit MD_Viscoplastic( const MatProps & props );
    virtual ~MD_Viscoplastic(){}

    static Material * createMaterial( const MatProps & props );

    int initialize( matParams * p );
    int getStress(matParams * p);

  private:

    //
    // private and unimplemented to prevent use
    //

    MD_Viscoplastic( const MD_Viscoplastic & );
    MD_Viscoplastic & operator= ( const MD_Viscoplastic & );

  };

  //*****************************************************************
  //
  // FORTRAN subroutine definitions
  //
  //   The MD_Viscoplastic model uses no FORTRAN subroutines
  //
  //*****************************************************************

} // lame

#endif
```

C.1.2 Source Code

```
#include <models/development/MD_Viscoplastic.h>
#include <models/development/MD_Viscoplastic_Model.h>

// md_viscoplastic
//
// Copyright 2018 National Technology & Engineering Solutions of Sandia, LLC (NTESS).
// Under the terms of Contract DE-NA0003525 with NTESS, the U.S. Government retains
// certain rights in this software.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this
// software and associated documentation files (the "Software"), to deal in the Software
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject to the following
// conditions:
//
// The above copyright notice and this permission notice shall be included in all copies
// or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
// CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
// OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

namespace lame {

  Material * MD_Viscoplastic::createMaterial( const MatProps & props ) {
    return new MD_Viscoplastic( props );
  }

  //*****************************************************************
  //
  // This is the constructor for the MD_Viscoplastic model.
  //
  //*****************************************************************

  MD_Viscoplastic::MD_Viscoplastic(const MatProps & props) :
    Material(props) {

    //
    // Material Property Definitions
    //

    mat_name = "MD_Viscoplastic";

    // Initialize the properties array
    initializeProperties(35);

    // Fill in the properties array with default values
```

```
setMaterialPropertyDefault(   2, 0.0 ); // default: no thermal expansion
setMaterialPropertyDefault(   3, 1000.0 ); // default: Tresca surface
setMaterialPropertyDefault(   4, 0.0 ); // default: zero steady state mechanism 0
setMaterialPropertyDefault(   5, 0.0 ); // default: zero steady state mechanism 1
setMaterialPropertyDefault(   6, 0.0 ); // default: zero steady state mechanism 2
setMaterialPropertyDefault(   7, 0.0 ); // default: no steady state mechanism 0
 ↪   temperature dependence
setMaterialPropertyDefault(   8, 0.0 ); // default: no steady state mechanism 1
 ↪   temperature dependence
setMaterialPropertyDefault(   9, 0.0 ); // default: no steady state mechanism 2
 ↪   temperature dependence
setMaterialPropertyDefault(  10, 0.0 ); // default: no steady state mechanism 0
 ↪   stress dependence
setMaterialPropertyDefault(  11, 0.0 ); // default: no steady state mechanism 1
 ↪   stress dependence
setMaterialPropertyDefault(  12, 0.0 ); // default: no steady state mechanism 2
 ↪   stress dependence
// no default for sigma_g
setMaterialPropertyDefault(  14, 0.0 ); // default: zero steady state mechanism 3
 ↪   coefficient 0
setMaterialPropertyDefault(  15, 0.0 ); // default: zero steady state mechanism 3
 ↪   coefficient 1
setMaterialPropertyDefault(  16, 0.0 ); // default: zero steady state mechanism 3
 ↪   coefficient 2
setMaterialPropertyDefault(  17, 0.0 ); // default: no steady state mechanism 3
 ↪   stress dependence
setMaterialPropertyDefault(  18, 0.0 ); // default: zero transient limit mechanism 0
setMaterialPropertyDefault(  19, 0.0 ); // default: zero transient limit mechanism 1
setMaterialPropertyDefault(  20, 0.0 ); // default: no transient limit mechanism 0
 ↪   temperature dependence
setMaterialPropertyDefault(  21, 0.0 ); // default: no transient limit mechanism 1
 ↪   temperature dependence
setMaterialPropertyDefault(  22, 0.0 ); // default: no transient limit mechanism 0
 ↪   stress dependence
setMaterialPropertyDefault(  23, 0.0 ); // default: no transient limit mechanism 1
 ↪   stress dependence
setMaterialPropertyDefault(  24, 0.0 ); // default: no stress independent hardening
setMaterialPropertyDefault(  25, 0.0 ); // default: no stress independent recovery
setMaterialPropertyDefault(  26, 0.0 ); // default: no hardening stress dependence
setMaterialPropertyDefault(  27, 0.0 ); // default: no recovery stress dependence
setMaterialPropertyDefault(  28, 2.0 ); // default: quadratic dependence on transient
 ↪   strain in transient function F (SAND89-2948 set _chi = 2, but we can integrate
 ↪   analytically if _chi = 1.  Thus, _chi is a developer variable that is set to 1
 ↪   in order to verify the model.)
// Property 29 is set below
setMaterialPropertyDefault(  30, 1e-11 ); // default: merit function must be below
 ↪   this number for model to converge
setMaterialPropertyDefault(  31, 1e-4 ); // default: a small improvement in merit
 ↪   function is sufficient to exit the line search
setMaterialPropertyDefault(  32, 0.1 ); // default: the current line search iteration
 ↪   cannot produce a parameter that is too small relative to the previous parameter
setMaterialPropertyDefault(  33, 100 ); // default: plenty of Newton iterations are
 ↪   allowed
```

```
setMaterialPropertyDefault(  34, 10 ); // default: only a few Line Search iterations
 ↪  are allowed

// Note about the max merit function value omega^0.5 is in units of strain, so 1e-11
// is nine orders of magnitude less than 1% strain.  This value was chosen by trying
// several different values in a triaxial compression creep simulation using a
// single element.  A step increase in equivalent stress of 16 MPa was applied, with
// a time step of 1 second, using the clean salt parameters in Table 2 and Table3 of
// SAND97-0795.  The equilibrium residual was normalized by the external loads, and
// the solver was allowed to take 1000 iterations for each time step.   All
// simulations failed during the first 86400 seconds into the traxial compression
// creep test, because the maximum allowable equilibrium relative residual was set
// exceedingly tight. The lowest equilibrium relative residual reached during the
// time step where the simulation failed is recorded.  Here are the results:
//
// material model residual          lowest equilibrium relative residual
// 1e-7                             ~5e-6
// 1e-8                             ~1e-6
// 1e-9                             ~1e-7
// 1e-10                            ~2e-10
//
// A relative equilibrium residual of 1e-7 is tight enough to give high quality
// results, but not so tight to cause excessive computation times.  Most likely,
// analysts will not require a tighter relative equilibrium residual than 1e-7,
// but a material model residual of 1e-11 was selected to give some wiggle room and
// allow analysts to achieve tighter relative equilibrium residuals.

// Fill in the user specified properties
setMaterialProperty( 0,"SHEAR_MODULUS",   props); // This must be capitalized, or
 ↪  else it will not be recognized
setMaterialProperty( 1,"BULK_MODULUS",    props); // This must be capitalized, or
 ↪  else it will not be recognized
// The remaining property names must match the XML file exactly, respecting upper
 ↪  case vs. lower case.
setMaterialProperty( 2,"alpha",           props);
setMaterialProperty( 3,"a",               props);
setMaterialProperty( 4,"A0",              props);
setMaterialProperty( 5,"A1",              props);
setMaterialProperty( 6,"A2",              props);
setMaterialProperty( 7,"Q0/R",            props);
setMaterialProperty( 8,"Q1/R",            props);
setMaterialProperty( 9,"Q2/R",            props);
setMaterialProperty(10,"n0",              props);
setMaterialProperty(11,"n1",              props);
setMaterialProperty(12,"n2",              props);
setMaterialProperty(13,"sigma_g",         props);
setMaterialProperty(14,"B0",              props);
setMaterialProperty(15,"B1",              props);
setMaterialProperty(16,"B2",              props);
setMaterialProperty(17,"q",               props);
setMaterialProperty(18,"K0",              props);
setMaterialProperty(19,"K1",              props);
setMaterialProperty(20,"c0",              props);
setMaterialProperty(21,"c1",              props);
```

```
  setMaterialProperty(22,"m0",              props);
  setMaterialProperty(23,"m1",              props);
  setMaterialProperty(24,"alpha_h",         props);
  setMaterialProperty(25,"alpha_r",         props);
  setMaterialProperty(26,"beta_h",          props);
  setMaterialProperty(27,"beta_r",          props);
  setMaterialProperty(28,"_chi",            props);
  // Property 29 is set below
  setMaterialProperty(30,"_sqrt_omega_max", props);
  setMaterialProperty(31,"_xi",             props);
  setMaterialProperty(32,"_gamma",          props);
  setMaterialProperty(33,"_k_max",          props);
  setMaterialProperty(34,"_j_max",          props);

  // Set property 29
  setMaterialPropertyDefault(  29, properties[0] * 1e-10 ); // default: the minimum
   ↪   equivalent stress should be around 1 Pa
  setMaterialProperty(29,"_sigma_min",      props);

  // Now that the properties array has been populated, check the properties
  MD_Viscoplastic_PropertyCheck( properties );

  //
  // State Variable Definitions
  //

  num_state_vars = 4;

  set_state_variable_alias("eq_tr_strain",         0);
  set_state_variable_alias("eq_vp_strain",         1);
  set_state_variable_alias("eq_stress",            2);
  set_state_variable_alias("vp_rate_scale_factor", 3);

}

//*******************************************************************
//
// The initialize method for the MD_Viscoplastic model
// zeros out the state variables
//
//*******************************************************************

int MD_Viscoplastic::initialize( matParams * p ) {

    // This verifies that temperature has been specified.  If not,
    // we should get a graceful exit instead of bombing.
    checkTemperature(p);

    MD_Viscoplastic_Initialize( p->nelements,
                                num_state_vars,
                                p->state_old,
                                p->state_new );

  return 0;
  }

  //*****************************************************************
  //
  // The getStress method for the MD_Viscoplastic model
  // finds the new stress for the material
  //
  //*****************************************************************

  int MD_Viscoplastic::getStress( matParams * p ) {

      MD_Viscoplastic_getStress( p->nelements,
               p->dt,
               properties,
                           p->strain_rate,
                           p->stress_old,
                           p->stress_new,
                           p->temp_old,
                           p->temp_new,
                           num_state_vars,
                           p->probingElement,
                           p->state_old,
                           p->state_new );

    return 0;

  }

} // lame
```

C.2 Model

C.2.1 Header

```
#ifndef _MD_VISCOPLASTIC_MODEL_H_
#define _MD_VISCOPLASTIC_MODEL_H_

#include <vector>
#include <models/Material.h>

// md_viscoplastic
//
// Copyright 2018 National Technology & Engineering Solutions of Sandia, LLC (NTESS).
// Under the terms of Contract DE-NA0003525 with NTESS, the U.S. Government retains
// certain rights in this software.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this
// software and associated documentation files (the "Software"), to deal in the Software
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject to the following
// conditions:
//
// The above copyright notice and this permission notice shall be included in all copies
// or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
// CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
// OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

namespace lame {

    void MD_Viscoplastic_Initialize( const int num_gauss_pts,
                                     const int num_sv,
                                     double * sv_old,
                                     double * sv);

    void MD_Viscoplastic_PropertyCheck( std::vector<double> & prop );

    void MD_Viscoplastic_getStress( const int num_gauss_pts,
                 const double dt,
                 std::vector<double> & props,
                 double * edot,
                 double * s_old,
                 double * s,
                 double * T_old,
                 double * T,
                 const int num_sv,
                 const bool probing_elements,
                 double * sv_old,
                 double * sv );

    void MD_Viscoplastic_Calc_Merit_Function(double dt,
                                   double vp_rate_scale_factor,
                                   double * s_p,
                                   double sigma_s,
                                   double _sigma_min,
                                   double a,
                                   double A0,
                                   double A1,
                                   double A2,
                                   double Q0oR,
                                   double Q1oR,
                                   double Q2oR,
                                   double n0,
                                   double n1,
                                   double n2,
                                   double mu,
                                   double sigma_g,
                                   double B0,
                                   double B1,
                                   double B2,
                                   double q,
                                   double K0,
                                   double K1,
                                   double c0,
                                   double c1,
                                   double m0,
                                   double m1,
                                   double alpha_h,
                                   double alpha_r,
                                   double beta_h,
                                   double beta_r,
                                   double _chi,
                                   double e_eq_tr,
                                   double e_eq_tr_old,
                                   double * de_vp_p,
                                   double * T,
                                   double & omega,
                                   double * R,
                                   double & r,
                                   double & s_eq,
                                   double * df_ds,
                                   double & edot_eq_ss,
                                   double & e_eq_tr_lmt,
                                   double & de_eq_vp,
                                   double & F,
                                   double & sign_r,
                                   double & sign,
                                   double & kappa,
                                   double & beta);

    void MD_Viscoplastic_Calc_Tangent(double dt,
                                      double * T,
                                      double mu,
                                      double kappa,
                                      double beta,
                                      double A0,
                                      double n0,
                                      double K0,
                                      double c0,
                                      double m0,
                                      double Q0oR,
                                      double B0,
                                      double A1,
                                      double n1,
                                      double K1,
                                      double c1,
                                      double m1,
                                      double Q1oR,
                                      double B1,
                                      double A2,
                                      double n2,
                                      double Q2oR,
                                      double B2,
                                      double q,
                                      double sigma_g,
                                      double a,
                                      double vp_rate_scale_factor,
                                      double _chi,
                                      double xi_a,
                                      double xi_b,
                                      double * df_ds,
                                      double * s_p,
                                      double sign,
                                      double F,
                                      double de_eq_vp,
                                      double e_eq_tr_lmt,
                                      double e_eq_tr,
                                      double s_eq,
                                      double edot_eq_ss,
                                      double & dr_ds_eq,
                                      double & dr_de_eq_tr,
                                      double & det_C_t_p_inv,
                                      double * C_t_p
                                      );

}

#endif // SIERRA_MD_Viscoplastic_Model_h
```

C.2.2 Source Code

```
#include <models/development/MD_Viscoplastic_Model.h>
#include <kinematics/Kinematics.h>
#include <cstdio>
#include <iostream>
#include <cmath>
#include <vector>
#include <kinematics/LameUtil.h>
#include <threading/Threading.h>

// md_viscoplastic
//
// Copyright 2018 National Technology & Engineering Solutions of Sandia, LLC (NTESS).
// Under the terms of Contract DE-NA0003525 with NTESS, the U.S. Government retains
// certain rights in this software.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this
// software and associated documentation files (the "Software"), to deal in the Software
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject to the following
// conditions:
//
// The above copyright notice and this permission notice shall be included in all copies
// or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
// CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
// OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

namespace lame {

void MD_Viscoplastic_getStress( const int num_gauss_pts,
                                const double dt,
                                std::vector<double> & properties,
                                double * edot,
                                double * s_old,
                                double * s,
                                double * T_old,
                                double * T,
                                const int num_sv,
                                const bool probing_elements,
                                double * sv_old,
                                double * sv ) {
//
//**************************************************************************************
//
//   DESCRIPTION:
//      This subroutine calculates the stresses at the end of the time step for the
//      Munson-Dawson Viscoplastic model with a Hosford flow potential.
//
//**************************************************************************************
//

    // Get material model parameters
    double mu                 = properties[0];  // shear modulus
    double B                  = properties[1];  // bulk modulus
    double alpha              = properties[2];  // coefficient of thermal expansion
    double a                  = properties[3];  // Hosford exponent
    double A0                 = properties[4];  // steady state mechanism 0 coefficient
    double A1                 = properties[5];  // steady state mechanism 1 coefficient
    double A2                 = properties[6];  // steady state mechanism 2 coefficient
    double Q0oR               = properties[7];  // steady state mechanism 0 activation
     ↪   energy
    double Q1oR               = properties[8];  // steady state mechanism 1 activation
     ↪   energy
    double Q2oR               = properties[9];  // steady state mechanism 2 activation
     ↪   energy
    double n0                 = properties[10]; // steady state mechanism 0 stress exponent
    double n1                 = properties[11]; // steady state mechanism 1 stress exponent
    double n2                 = properties[12]; // steady state mechanism 2 stress exponent
    double sigma_g            = properties[13]; // steady state mechanism 3 stress cutoff
    double B0                 = properties[14]; // steady state mechanism 3 coefficient 0
    double B1                 = properties[15]; // steady state mechanism 3 coefficient 1
    double B2                 = properties[16]; // steady state mechanism 3 coefficient 2
    double q                = properties[17]; // steady state mechanism 3 stress
     ↪  coefficient
    double K0               = properties[18]; // transient limit mechanism 0 coefficient
    double K1               = properties[19]; // transient limit mechanism 1 coefficient
    double c0               = properties[20]; // transient limit mechanism 0 temperature
     ↪  coefficient
    double c1               = properties[21]; // transient limit mechanism 1 temperature
     ↪  coefficient
    double m0               = properties[22]; // transient limit mechanism 0 stress
     ↪  exponent
    double m1               = properties[23]; // transient limit mechanism 1 stress
     ↪  exponent
    double alpha_h          = properties[24]; // hardening constant
    double alpha_r          = properties[25]; // recovery constant
    double beta_h           = properties[26]; // hardening stress coefficient
    double beta_r           = properties[27]; // recovery stress coefficient
    double _chi             = properties[28]; // exponent in transient function F
         ↪  (normally _chi = 2, but we can integrate analytically if _chi = 1.  Thus, _chi
         ↪  is a developer variable that is set to 1 in order to verify the model.)
    double _sigma_min       = properties[29]; // minimum equivalent stress
    double _sqrt_omega_max = properties[30]; // merit function tolerance
    double _xi              = properties[31]; // line search minimum improvement factor
    double _gamma           = properties[32]; // line search parameter minimum size factor
    double _k_max           = properties[33]; // max number of newton iterations
    double _j_max           = properties[34]; // max number of line search iterations

    // Grab the viscoplastic rate scale factor
    // (This is a global quantity in concept, even though it is stored at every
    // integration point, so we do not need to grab it from every integration point.
    // Instead, we just grab it once.)
    double vp_rate_scale_factor = sv_old[3];

    // Elasticity constants
    double two_mu = 2.0 * mu;
    double lambda = B - two_mu / 3.0;
    double xi_a = -lambda / (3.0 * B * two_mu);
    double xi_b = 1.0 / two_mu;
    // We normalize the principal stresses to something close to 1 to avoid
    // overflow errors if the Hosford exponent is large.  We use another equivalent,
    // pressure independent, stress measure to normalize the principal stresses
    // which works in most cases, but if we have a state of hydrostatic pressure
    // then the principal stresses will be divided by zero.  Hence we need to add a very
    // small stress to our other equivalent stress measure.  One could create a new small
    // stress value for this purpose, but it is easier to just use the minimum Hosford
    // equivalent stress.
    double sigma_s = _sigma_min;

    // SAND82-0962 defined the recovery branch F as
    // F = exp(kappa * (1.0 - e_eq_tr / e_eq_tr_lmt))
    // while SAND89-2948 redefined the recovery branch F as
    // F = exp(-kappa * (1.0 - e_eq_tr / e_eq_tr_lmt)^2)
    // This makes it so that e_eq_tr decreases as long as kappa > 0 and
    // e_eq_tr > e_eq_tr_lmt.  Here, we want to accommodate both cases so that _chi can
    // be 1 or 2.  (_chi = 2 is the default value, while _chi = 1 allows for analytical
    // solutions to the transient strain ODE.) To allow for both cases we must also
    // change the sign for the recovery branch according to whether
    // _chi is odd or even.
    double sign_r;
    if ( (int)_chi % 2 == 0.0){
        sign_r = -1.0;
    } else {
        sign_r = 1.0;
    }

    // Sierra/SM computes the stress for every Gauss point in an
    // element block with one call to the constitutive model.  Thus, we
    // need to loop over the number of Gauss points in the element block.
    for ( int m = 0 ; m < num_gauss_pts ; m++ ){

        // Get internal variables (which may or may not actually be formal state
        // variables)
        double e_eq_tr_old = sv_old[0];
        double e_eq_vp_old = sv_old[1];
        double e_eq_tr = e_eq_tr_old;

        // Calculate the thermal strain increment
        double de_th = alpha * (T[0] - T_old[0]);

        // Compute the trial strain increment, assuming no plasticity
        double de_tr[6] = {0.0};
        de_tr[0] = edot[0] * dt - de_th;
        de_tr[1] = edot[1] * dt - de_th;
        de_tr[2] = edot[2] * dt - de_th;
        de_tr[3] = edot[3] * dt;
        de_tr[4] = edot[4] * dt;
        de_tr[5] = edot[5] * dt;

        // Compute the trial stress, assuming a fully elastic strain increment
        double de_tr_trace = de_tr[0] + de_tr[1] + de_tr[2];
        s[0] = s_old[0] + lambda * de_tr_trace + two_mu * de_tr[0];
        s[1] = s_old[1] + lambda * de_tr_trace + two_mu * de_tr[1];
        s[2] = s_old[2] + lambda * de_tr_trace + two_mu * de_tr[2];
        s[3] = s_old[3]                         + two_mu * de_tr[3];
        s[4] = s_old[4]                         + two_mu * de_tr[4];
        s[5] = s_old[5]                         + two_mu * de_tr[5];

        // Compute principal stresses
        double s_p[3] = {0.0};
        double p_s_d[9] = {0.0};
        kinematics::eigen_new(s, s_p, p_s_d);

        // Calculate merit function
        // Declare variables necessary for calculating merit function
        double omega;
        double R[3] = {0.0};
        double r;
        double s_eq;
        double df_ds[3] = {0.0};
        double edot_eq_ss;
        double de_vp_p[3] = {0.0};
        double de_eq_vp;
        double e_eq_tr_lmt;
        double F;
        double sign;
        double kappa;
        double beta;
        // Calculate any quantities needed for residuals and calculate merit function
        MD_Viscoplastic_Calc_Merit_Function(dt, vp_rate_scale_factor, s_p, sigma_s,
            _sigma_min, a, A0, A1, A2, Q0oR, Q1oR, Q2oR, n0, n1, n2, mu, sigma_g, B0, B1,
            B2, q, K0, K1, c0, c1, m0, m1, alpha_h, alpha_r, beta_h, beta_r, _chi,
            e_eq_tr, e_eq_tr_old, de_vp_p, T, omega, R, r, s_eq, df_ds, edot_eq_ss,
            e_eq_tr_lmt, de_eq_vp, F, sign_r, sign, kappa, beta);

        // Initialize the iteration numbers
        int k = 0;
        int j = 0;
        // Initialize the determinant of the algorithmic tangent inverse, the
        // derivatives of the transient ODE residual, and the algorithmic tangent.
        // (The determinant needs to be initialized before the Newton iteration loop
        // because error checking looks for it after the Newton iteration loop even if
        // no Newton iterations ensue. Similarly, the derivatives of the transient ODE
        // residual and the algorithmic tangent may be computed after the Newton
        // iteration loop, even if no Newton iterations ensue.)
        double det_C_t_p_inv = 1.0;
        double dr_ds_eq = 0;
        double dr_de_eq_tr = 0;
        double C_t_p[6] = {0.0};
        // Iterate until tolerance on merit function is met
        while (std::sqrt(omega) > _sqrt_omega_max){
            // Increment iteration counter and exit while loop if too many iterations
            k = k + 1;
            if (k > _k_max){
                break;
            }
            // Save the values from the previous iteration in case the line search
            // algorithm is activated.
            double s_p_im1[3] = {0.0};
            s_p_im1[0] = s_p[0];
            s_p_im1[1] = s_p[1];
            s_p_im1[2] = s_p[2];
            double e_eq_tr_im1 = e_eq_tr;
            double de_vp_p_im1[3] = {0.0};
            de_vp_p_im1[0] = de_vp_p[0];
            de_vp_p_im1[1] = de_vp_p[1];
            de_vp_p_im1[2] = de_vp_p[2];
            double omega_im1 = omega;

            // Compute the algorithmic tangent
            MD_Viscoplastic_Calc_Tangent(dt, T, mu, kappa, beta, A0, n0, K0, c0, m0,
                Q0oR, B0, A1, n1, K1, c1, m1,Q1oR, B1, A2, n2, Q2oR, B2, q, sigma_g, a,
                vp_rate_scale_factor, _chi, xi_a, xi_b, df_ds, s_p, sign, F, de_eq_vp,
                e_eq_tr_lmt, e_eq_tr, s_eq, edot_eq_ss, dr_ds_eq, dr_de_eq_tr,
            det_C_t_p_inv, C_t_p);

        // Compute increment in stress
        double dds_p[3] = {0.0};
        double pf = r / dr_de_eq_tr;
        dds_p[0] = - C_t_p[0] * (R[0] - pf * df_ds[0])
                   - C_t_p[3] * (R[1] - pf * df_ds[1])
                   - C_t_p[5] * (R[2] - pf * df_ds[2]);
        dds_p[1] = - C_t_p[3] * (R[0] - pf * df_ds[0])
                   - C_t_p[1] * (R[1] - pf * df_ds[1])
                   - C_t_p[4] * (R[2] - pf * df_ds[2]);
        dds_p[2] = - C_t_p[5] * (R[0] - pf * df_ds[0])
                   - C_t_p[4] * (R[1] - pf * df_ds[1])
                   - C_t_p[2] * (R[2] - pf * df_ds[2]);
        // Compute increment in transient strain
        double dde_eq_tr = - (r + dr_ds_eq *
                (df_ds[0] * dds_p[0] + df_ds[1] * dds_p[1] + df_ds[2] * dds_p[2])) /
                 ↪  dr_de_eq_tr;

        // Update variables
        // Update the principal stresses
        s_p[0] = s_p[0] + dds_p[0];
        s_p[1] = s_p[1] + dds_p[1];
        s_p[2] = s_p[2] + dds_p[2];
        // Update the equivalent transient strain
        e_eq_tr = e_eq_tr + dde_eq_tr;
        // Update the viscoplastic strain tensor increment
        double dde_vp_p[3] = {0.0};
        dde_vp_p[0] = -dds_p[0] / two_mu;
        dde_vp_p[1] = -dds_p[1] / two_mu;
        dde_vp_p[2] = -dds_p[2] / two_mu;
        de_vp_p[0] = de_vp_p[0] + dde_vp_p[0];
        de_vp_p[1] = de_vp_p[1] + dde_vp_p[1];
        de_vp_p[2] = de_vp_p[2] + dde_vp_p[2];

        // Calculate any quantities needed for residuals and calculate merit function
        MD_Viscoplastic_Calc_Merit_Function(dt, vp_rate_scale_factor, s_p, sigma_s,
            _sigma_min, a, A0, A1, A2, Q0oR, Q1oR, Q2oR, n0, n1, n2, mu, sigma_g, B0,
            B1, B2, q, K0, K1, c0, c1, m0, m1, alpha_h, alpha_r, beta_h, beta_r,
            _chi, e_eq_tr, e_eq_tr_old, de_vp_p, T, omega, R, r, s_eq, df_ds,
            edot_eq_ss, e_eq_tr_lmt, de_eq_vp, F, sign_r, sign, kappa, beta);
        if (det_C_t_p_inv <= 0.0){
            break;
        }

        // Perform Line Search
        double zeta = 1.0;
        // Force the line search iteration number to be reset with every Newton
        // iteration.
        j = 0;
        // Check the Goldstein condition
        // (Note Scherzinger.W-2016a has a typo for equation (51).
        // He compares omega against the previous line search iteration.
        // He should compare omega against the previous accepted Newton/Line Search
```

```
        // iteration omega_im1.)
        while (omega > (1.0 - 2.0 * _xi * zeta) * omega_im1){
            // Update the line search iteration count and break out of the loop if
            // too many iterations
            j = j + 1;
            if (j > _j_max){
                break;
            }
            // Save the value of zeta from the last iteration
            double zeta_jm1 = zeta;
            // Solve for the value of zeta that minimizes the quadratic
            // approximation for the merit function.  This assumes we have a
            // local minimum, which is not always true.
            // (Note Scherzinger.W-2016a has a typo for equation (50).  He
            // has zeta = omega_im1 / (omega + omega_im1), which is only true
            // for the first iteration.)
            zeta = omega_im1 * zeta_jm1*zeta_jm1 / (omega - omega_im1 + 2.0 *
             ↪  omega_im1 * zeta_jm1);
            zeta = std::max(_gamma * zeta_jm1, zeta);
            // Update variables
            s_p[0] = s_p_im1[0] + zeta * dds_p[0];
            s_p[1] = s_p_im1[1] + zeta * dds_p[1];
            s_p[2] = s_p_im1[2] + zeta * dds_p[2];
            e_eq_tr = e_eq_tr_im1 + zeta * dde_eq_tr;
            de_vp_p[0] = de_vp_p_im1[0] + zeta * dde_vp_p[0];
            de_vp_p[1] = de_vp_p_im1[1] + zeta * dde_vp_p[1];
            de_vp_p[2] = de_vp_p_im1[2] + zeta * dde_vp_p[2];
            // Calculate any quantities needed for residuals and calculate merit
            // function
            MD_Viscoplastic_Calc_Merit_Function(dt, vp_rate_scale_factor, s_p,
                sigma_s, _sigma_min, a, A0, A1, A2, Q0oR, Q1oR, Q2oR, n0, n1, n2, mu,
                sigma_g, B0, B1, B2, q, K0, K1, c0, c1, m0, m1, alpha_h, alpha_r,
                beta_h, beta_r, _chi, e_eq_tr, e_eq_tr_old, de_vp_p, T, omega, R, r,
                s_eq, df_ds, edot_eq_ss, e_eq_tr_lmt, de_eq_vp, F, sign_r, sign,
                kappa, beta);

        }
    }

    // Error Handling
    if (k > _k_max || j > _j_max || det_C_t_p_inv < 0.0 || !std::isfinite(omega)){
        // As of October 2017, Sierra/SM cannot catch a material model error during
        // probing so we simply report that it happened in the log file and use a
        // fraction of the elastic trial stress for the stress increment.
        int err_code = 3;
        if (probing_elements) {
            err_code = 1;
        }
        std::stringstream err_msg;
        err_msg.precision(16);
        err_msg << "The MD_Viscoplastic model ODE integration scheme failed to
         ↪  converge at Gauss point #" << k << "." << std::endl;
        if (k > _k_max){
```

```
        err_msg << "The model performed the maximum number of Newton iterations
         ↪  (" << _k_max << "),\n"
                << "yet the square root of the merit function is " <<
                 ↪  std::sqrt(omega) << ", which exceeds the maximum allowed ("
                 ↪  << _sqrt_omega_max << ")." << std::endl;
    }
    if (j > _j_max){
        err_msg << "The model performed the maximum number of Line Search
         ↪  iterations (" << _j_max << "), \n"
                << "without sufficient improvement.\n" << std::endl;
    }
    if (det_C_t_p_inv < 0.0){
        err_msg << "The model's algorithmic tangent is not positive definite. \n"
                << "This issue is often associated with a large deformation
                 ↪  increment being input into the MD_Viscoplastic model, which
                 ↪  leads to a large equivalent stress. \n"
                << "The equivalent stress is " << s_eq << "." << std::endl;
    }
    if ( !std::isfinite(omega) ){
        err_msg << "The model returned a nan." << std::endl;
    }
    err_msg << "Here are the inputs to the model that caused the issue:" << "\n"
            << "s_old = [" << s_old[0] << ", " << s_old[1] << ", " << s_old[2] <<
             ↪  ", " << s_old[3] << ", " << s_old[4] << ", " << s_old[5] <<
             ↪  "]\n"
            << "edot = [" << edot[0] << ", " << edot[1] << ", " << edot[2] << ",
             ↪  " << edot[3] << ", " << edot[4] << ", " << edot[5] << "]\n"
            << "e_eq_tr_old = " << e_eq_tr_old << "\n"
            << "dt = " << dt << "\n"
            << "vp_rate_scale_factor = " << vp_rate_scale_factor << "\n"
            << "T_old = " << T_old[0] << "\n"
            << "dT = " << T[0] - T_old[0] << std::endl;
    lame::Material::reportError(err_code, err_msg.str());
    if (probing_elements) {
        std::stringstream err_msg1;
        err_msg1 << "The failure occurred while probing the element stiffnesses,
         ↪  so the model will simply increment the stress by 1/10 the trial
         ↪  elastic stress increment." << std::endl;
        lame::Material::reportError(1, err_msg1.str());
        s[0] = s_old[0] + (lambda * de_tr_trace + two_mu * de_tr[0]) / 10.0;
        s[1] = s_old[1] + (lambda * de_tr_trace + two_mu * de_tr[1]) / 10.0;
        s[2] = s_old[2] + (lambda * de_tr_trace + two_mu * de_tr[2]) / 10.0;
        s[3] = s_old[3]                         + two_mu * de_tr[3] / 10.0;
        s[4] = s_old[4]                         + two_mu * de_tr[4] / 10.0;
        s[5] = s_old[5]                         + two_mu * de_tr[5] / 10.0;
        // I am not sure Sierra/SM is printing the reportError(1, str) to the log
        // file when probing, so we print to screen to troubleshoot.
        std::cout << "The MD_Viscoplastic model ODE integration scheme failed to
         ↪  converge while probing Gauss point #" << k << " at dt = " << dt <<
         ↪  "." << std::endl;
    }
}

// Although Sierra/Solid Mechanics does not make use of the tangent, this is
```

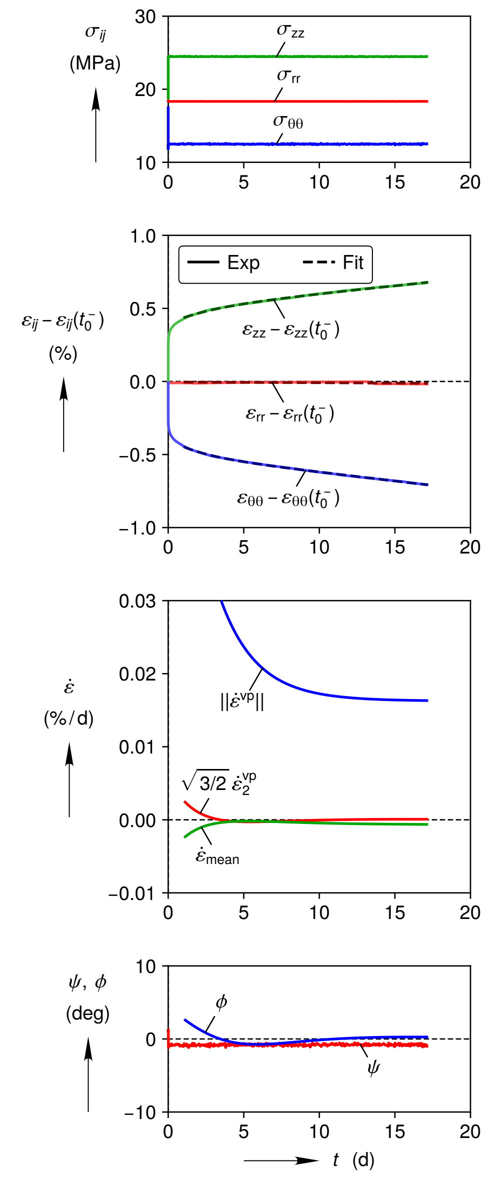
```
        // where one would compute the algorithmic tangent at the end of the time step in
        // order to return it to the host momentum balance code.
//        MD_Viscoplastic_Calc_Tangent(dt, T, mu, kappa, beta, A0, n0, K0, c0, m0, Q0oR,
//            B0, A1, n1, K1, c1, m1,Q1oR, B1, A2, n2, Q2oR, B2, q, sigma_g, a,
//            vp_rate_scale_factor, _chi, xi_a, xi_b, df_ds, s_p, sign, F, de_eq_vp,
//            e_eq_tr_lmt, e_eq_tr, s_eq, edot_eq_ss, dr_ds_eq, dr_de_eq_tr,
//            det_C_t_p_inv, C_t_p);
        // Convert the tangent from the principal basis back to the original basis
//         double C_t[21] = {0.0}
//         Transform_Fourth_Order_Tensor(C_t, C_t_p, p_s_d)

        // Convert the stress tensor from the principal basis back to the original basis
        s[0] = s_p[0]*p_s_d[0]*p_s_d[0] + s_p[1]*p_s_d[3]*p_s_d[3] +
         ↪  s_p[2]*p_s_d[8]*p_s_d[8];
        s[1] = s_p[0]*p_s_d[6]*p_s_d[6] + s_p[1]*p_s_d[1]*p_s_d[1] +
         ↪  s_p[2]*p_s_d[4]*p_s_d[4];
        s[2] = s_p[0]*p_s_d[5]*p_s_d[5] + s_p[1]*p_s_d[7]*p_s_d[7] +
         ↪  s_p[2]*p_s_d[2]*p_s_d[2];
        s[3] = s_p[0]*p_s_d[0]*p_s_d[6] + s_p[1]*p_s_d[3]*p_s_d[1] +
         ↪  s_p[2]*p_s_d[8]*p_s_d[4];
        s[4] = s_p[0]*p_s_d[6]*p_s_d[5] + s_p[1]*p_s_d[1]*p_s_d[7] +
         ↪  s_p[2]*p_s_d[4]*p_s_d[2];
        s[5] = s_p[0]*p_s_d[5]*p_s_d[0] + s_p[1]*p_s_d[7]*p_s_d[3] +
         ↪  s_p[2]*p_s_d[2]*p_s_d[8];

        // Update the internal state variables
        sv[0] = e_eq_tr;
        sv[1] = e_eq_vp_old + de_eq_vp;
        sv[2] = s_eq;
        sv[3] = vp_rate_scale_factor;

        // The stress/strain rate/state variable pointers correspond to
        // arrays that contain all the stresses/strain rates/state variables
        // for an entire element block, so we need to increment the pointers
        // in preparation for the next Gauss point.
        s_old = s_old + 6;
        s = s + 6;
        edot = edot + 6;
        T_old = T_old + 1;
        T = T + 1;
        sv_old = sv_old + num_sv;
        sv = sv + num_sv;

    }

}

//----------------------------------------------------------------------
//
// Merit Function
//
//----------------------------------------------------------------------

void MD_Viscoplastic_Calc_Merit_Function(double dt,
                        double vp_rate_scale_factor,
                        double * s_p,
                        double sigma_s,
                        double _sigma_min,
                        double a,
                        double A0,
                        double A1,
                        double A2,
                        double Q0oR,
                        double Q1oR,
                        double Q2oR,
                        double n0,
                        double n1,
                        double n2,
                        double mu,
                        double sigma_g,
                        double B0,
                        double B1,
                        double B2,
                        double q,
                        double K0,
                        double K1,
                        double c0,
                        double c1,
                        double m0,
                        double m1,
                        double alpha_h,
                        double alpha_r,
                        double beta_h,
                        double beta_r,
                        double _chi,
                        double e_eq_tr,
                        double e_eq_tr_old,
                        double * de_vp_p,
                        double * T,
                        double & omega,
                        double * R,
                        double & r,
                        double & s_eq,
                        double * df_ds,
                        double & edot_eq_ss,
                        double & e_eq_tr_lmt,
                        double & de_eq_vp,
                        double & F,
                        double & sign_r,
                        double & sign,
                        double & kappa,
                        double & beta) {

// Calculate the Hosford equivalent stress
// Calculate the von Mises stress in order to normalize the principal stresses
double s_p01 = s_p[0] - s_p[1];
double s_p12 = s_p[1] - s_p[2];
double s_p20 = s_p[2] - s_p[0];
double von_Mises = std::sqrt(0.5 * ((s_p01 * s_p01) + (s_p12 * s_p12) + (s_p20 *
 ↪  s_p20)));
// Add a very small stress to the von Mises stress to make sure it is not zero
von_Mises = von_Mises + sigma_s;
// Normalize the principal stresses to something close to 1
double s_hat_p[3] = {0.0};
s_hat_p[0] = s_p[0] / von_Mises;
s_hat_p[1] = s_p[1] / von_Mises;
s_hat_p[2] = s_p[2] / von_Mises;

// Calculate the normalized equivalent stress using the normalized
// stresses so that we do not run into overflow errors if exponent is
// large.
double s_hat_eq = std::pow(1.0 / 2.0 * (std::pow(std::fabs(s_hat_p[0] - s_hat_p[1]),
 ↪  a) \
                     + std::pow(std::fabs(s_hat_p[1] - s_hat_p[2]), a) \
                     + std::pow(std::fabs(s_hat_p[0] - s_hat_p[2]), a)), 1.0/a);
// Convert back to the un-normalized equivalent stress
s_eq = von_Mises * s_hat_eq;
// The MD model has trouble if the equivalent stress is zero, so we
// do not let that occur.
if ( s_eq < _sigma_min){
    s_eq = _sigma_min;
}

// Calculate normal to flow potential.  (Normalize with the Hosford equivalent
//stress instead of the von Mises stress.)
s_hat_p[0] = s_p[0] / s_eq;
s_hat_p[1] = s_p[1] / s_eq;
s_hat_p[2] = s_p[2] / s_eq;
double ex1 = a - 2.0;
df_ds[0] = 0.5 * ( (s_hat_p[0] - s_hat_p[1])*std::pow(std::abs(s_hat_p[0] -
 ↪  s_hat_p[1]),ex1) + (s_hat_p[0] - s_hat_p[2])*std::pow(std::abs(s_hat_p[0] -
 ↪  s_hat_p[2]),ex1));
df_ds[1] = 0.5 * (-(s_hat_p[0] - s_hat_p[1])*std::pow(std::abs(s_hat_p[0] -
 ↪  s_hat_p[1]),ex1) + (s_hat_p[1] - s_hat_p[2])*std::pow(std::abs(s_hat_p[1] -
 ↪  s_hat_p[2]),ex1));
df_ds[2] = 0.5 * (-(s_hat_p[1] - s_hat_p[2])*std::pow(std::abs(s_hat_p[1] -
 ↪  s_hat_p[2]),ex1) - (s_hat_p[0] - s_hat_p[2])*std::pow(std::abs(s_hat_p[0] -
 ↪  s_hat_p[2]),ex1));

// Calculate the steady-state equivalent strain rate
double edot_eq_ss_0 = A0 * std::exp(-Q0oR / T[0]) * std::pow(s_eq / mu, n0);
double edot_eq_ss_1 = A1 * std::exp(-Q1oR / T[0]) * std::pow(s_eq / mu, n1);
double edot_eq_ss_2 = A2 * std::exp(-Q2oR / T[0]) * std::pow(s_eq / mu, n2);
double edot_eq_ss_3;
if (s_eq < sigma_g){
    edot_eq_ss_3 = 0.0;
} else {
    edot_eq_ss_3 = (B0 * std::exp(-Q0oR / T[0]) + B1 * std::exp(-Q1oR / T[0]) + B2 *
     ↪  std::exp(-Q2oR / T[0])) * std::sinh(q * (s_eq - sigma_g) / mu);
}
edot_eq_ss = vp_rate_scale_factor * (edot_eq_ss_0 + edot_eq_ss_1 + edot_eq_ss_2 +
 ↪  edot_eq_ss_3);
```

```
    // Calculate the transient equivalent strain limit
    double e_eq_tr_lmt_0 = K0 * std::exp(c0 * T[0]) * std::pow(s_eq / mu, m0);
    double e_eq_tr_lmt_1 = K1 * std::exp(c1 * T[0]) * std::pow(s_eq / mu, m1);
    e_eq_tr_lmt = e_eq_tr_lmt_0 + e_eq_tr_lmt_1;

    // Calculate F
    double alpha;
    if (e_eq_tr <= e_eq_tr_lmt){
        alpha = alpha_h;
        beta = beta_h;
        sign = 1.0;
    } else {
        alpha = alpha_r;
        beta = beta_r;
        sign = sign_r;
    }
    kappa = alpha + beta * std::log10(s_eq / mu);
    // kappa should always be greater than 0.0, otherwise one will get a
    // decreasing / increasing rate of equivalent transient strain even
    // though e_eq_tr is below / above e_eq_tr_lmt
    // (It is possible that kappa < 0.0 violates the second law of thermo,
    //  but I would need to check.)
    if (kappa < 0.0){
        kappa = 0.0;
        // We also make alpha = beta = 0, since that makes sure that the
        // derivative of F wrt the equivalent stress is 0
        alpha = 0.0;
        beta = 0.0;
    }
    F = std::exp(sign * kappa * std::pow(1.0 - e_eq_tr / e_eq_tr_lmt, _chi));

    // Calculate the residuals
    de_eq_vp = (e_eq_tr - e_eq_tr_old) + dt * edot_eq_ss;
    R[0] = -de_vp_p[0] + de_eq_vp * df_ds[0];
    R[1] = -de_vp_p[1] + de_eq_vp * df_ds[1];
    R[2] = -de_vp_p[2] + de_eq_vp * df_ds[2];
    r = -(e_eq_tr - e_eq_tr_old) + (F - 1.0) * dt * edot_eq_ss;

    // Combine the residuals into a single scalar merit function
    omega = 0.5 * ((R[0]*R[0]) + (R[1]*R[1]) + (R[2]*R[2]) + (r*r));

}

//----------------------------------------------------------------------
//
// Algorithmic Tangent Stiffness
//
//----------------------------------------------------------------------

void MD_Viscoplastic_Calc_Tangent(double dt,
                                  double * T,
                                  double mu,
                                  double kappa,
                                     double beta,
                                     double A0,
                                     double n0,
                                     double K0,
                                     double c0,
                                     double m0,
                                     double Q0oR,
                                     double B0,
                                     double A1,
                                     double n1,
                                     double K1,
                                     double c1,
                                     double m1,
                                     double Q1oR,
                                     double B1,
                                     double A2,
                                     double n2,
                                     double Q2oR,
                                     double B2,
                                     double q,
                                     double sigma_g,
                                     double a,
                                     double vp_rate_scale_factor,
                                     double _chi,
                                     double xi_a,
                                     double xi_b,
                                     double * df_ds,
                                     double * s_p,
                                     double sign,
                                     double F,
                                     double de_eq_vp,
                                     double e_eq_tr_lmt,
                                     double e_eq_tr,
                                     double s_eq,
                                     double edot_eq_ss,
                                     double & dr_ds_eq,
                                     double & dr_de_eq_tr,
                                     double & det_C_t_p_inv,
                                     double * C_t_p
                                     ){

// Calculate the derivative of F wrt the transient equivalent strain
double dF_de_eq_tr = -sign * F * _chi * kappa / e_eq_tr_lmt * std::pow(1.0 - e_eq_tr
 ↪  / e_eq_tr_lmt, _chi - 1.0);
// Calculate the derivative of F wrt the equivalent stress
double dF_ds_eq = sign * F * (beta * std::pow(1.0 - e_eq_tr / e_eq_tr_lmt, _chi) /
 ↪  (std::log(10.0) * s_eq)
    + _chi * kappa * std::pow(1.0 - e_eq_tr / e_eq_tr_lmt, _chi - 1.0) * e_eq_tr /
     ↪  (mu * e_eq_tr_lmt*e_eq_tr_lmt )
    * (K0 * std::exp(c0 * T[0]) * m0 * std::pow(s_eq / mu, m0 - 1.0)
    +  K1 * std::exp(c1 * T[0]) * m1 * std::pow(s_eq / mu, m1 - 1.0)));

// Calculate the derivative of the steady state rate wrt the equivalent stress
double dedot_eq_ss_ds_eq =
      (A0 * std::exp(-Q0oR / T[0]) * n0 * std::pow(s_eq / mu, n0 - 1.0)
     + A1 * std::exp(-Q1oR / T[0]) * n1 * std::pow(s_eq / mu, n1 - 1.0)
     + A2 * std::exp(-Q2oR / T[0]) * n2 * std::pow(s_eq / mu, n2 - 1.0)) / mu;
if (s_eq > sigma_g){
    dedot_eq_ss_ds_eq = dedot_eq_ss_ds_eq
                      + (B0 * std::exp(-Q0oR / T[0])
                       + B1 * std::exp(-Q1oR / T[0])
                       + B2 * std::exp(-Q2oR / T[0]))
                       * q * std::cosh(q * (s_eq - sigma_g) / mu) / mu;
}
// Scale the steady-state rate derivative
dedot_eq_ss_ds_eq = vp_rate_scale_factor * dedot_eq_ss_ds_eq;

// Calculate second derivatives of flow potential
double df_dsds[6] = {0.0}; // Symmetric 3x3 tensor, in 6x1 form
double pf = (a - 1.0) / s_eq;
double ex2 = a - 2.0;
double sh_p[3] = { s_p[0] / s_eq, s_p[1] / s_eq, s_p[2] / s_eq };
df_dsds[0] = pf * ( 0.5 * (std::pow(std::abs(sh_p[0] - sh_p[1]), ex2) +
 ↪  std::pow(std::abs(sh_p[0] - sh_p[2]), ex2)) - df_ds[0]*df_ds[0]);
df_dsds[1] = pf * ( 0.5 * (std::pow(std::abs(sh_p[0] - sh_p[1]), ex2) +
 ↪  std::pow(std::abs(sh_p[1] - sh_p[2]), ex2)) - df_ds[1]*df_ds[1]);
df_dsds[2] = pf * ( 0.5 * (std::pow(std::abs(sh_p[1] - sh_p[2]), ex2) +
 ↪  std::pow(std::abs(sh_p[0] - sh_p[2]), ex2)) - df_ds[2]*df_ds[2]);
df_dsds[3] = pf * (-0.5 * std::pow(std::abs(sh_p[0] - sh_p[1]), ex2) - df_ds[0] *
 ↪  df_ds[1]);
df_dsds[4] = pf * (-0.5 * std::pow(std::abs(sh_p[1] - sh_p[2]), ex2) - df_ds[1] *
 ↪  df_ds[2]);
df_dsds[5] = pf * (-0.5 * std::pow(std::abs(sh_p[0] - sh_p[2]), ex2) - df_ds[0] *
 ↪  df_ds[2]);

// Calculate derivatives of transient strain residual equation
dr_ds_eq = (dF_ds_eq * edot_eq_ss + (F - 1.0) * dedot_eq_ss_ds_eq) * dt;
dr_de_eq_tr = -1.0 + dF_de_eq_tr * dt * edot_eq_ss;
// Compute the inverse of the algorithmic tangent stiffness in principal frame
double C_t_p_inv[6] = {0.0};
pf = dt * dedot_eq_ss_ds_eq - dr_ds_eq / dr_de_eq_tr;
C_t_p_inv[0] = (xi_a + xi_b) + de_eq_vp * df_dsds[0] + pf * df_ds[0] * df_ds[0];
C_t_p_inv[1] = (xi_a + xi_b) + de_eq_vp * df_dsds[1] + pf * df_ds[1] * df_ds[1];
C_t_p_inv[2] = (xi_a + xi_b) + de_eq_vp * df_dsds[2] + pf * df_ds[2] * df_ds[2];
C_t_p_inv[3] = xi_a + de_eq_vp * df_dsds[3] + pf * df_ds[0] * df_ds[1];
C_t_p_inv[4] = xi_a + de_eq_vp * df_dsds[4] + pf * df_ds[1] * df_ds[2];
C_t_p_inv[5] = xi_a + de_eq_vp * df_dsds[5] + pf * df_ds[0] * df_ds[2];

// Check to see if the matrix is has a positive determinant
// (This is the standard formula for a determinant of a 3x3,
// except specialized to a symmetric 3x3, since we only have the
// upper triangular portion of the 3x3.)
det_C_t_p_inv = C_t_p_inv[0] * C_t_p_inv[1] * C_t_p_inv[2]
              + C_t_p_inv[3] * C_t_p_inv[4] * C_t_p_inv[5]
              + C_t_p_inv[5] * C_t_p_inv[3] * C_t_p_inv[4]
              - C_t_p_inv[5] * C_t_p_inv[1] * C_t_p_inv[5]
              - C_t_p_inv[0] * C_t_p_inv[4] * C_t_p_inv[4]
              - C_t_p_inv[3] * C_t_p_inv[3] * C_t_p_inv[2];
    // Use the analytical expression for the inverse of a 3x3
    C_t_p[0] = (C_t_p_inv[1] * C_t_p_inv[2] - C_t_p_inv[4] * C_t_p_inv[4]) /
     ↪  det_C_t_p_inv;
    C_t_p[1] = (C_t_p_inv[2] * C_t_p_inv[0] - C_t_p_inv[5] * C_t_p_inv[5]) /
     ↪  det_C_t_p_inv;
    C_t_p[2] = (C_t_p_inv[0] * C_t_p_inv[1] - C_t_p_inv[3] * C_t_p_inv[3]) /
     ↪  det_C_t_p_inv;
    C_t_p[3] = (C_t_p_inv[4] * C_t_p_inv[5] - C_t_p_inv[3] * C_t_p_inv[2]) /
     ↪  det_C_t_p_inv;
    C_t_p[4] = (C_t_p_inv[5] * C_t_p_inv[3] - C_t_p_inv[4] * C_t_p_inv[0]) /
     ↪  det_C_t_p_inv;
    C_t_p[5] = (C_t_p_inv[3] * C_t_p_inv[4] - C_t_p_inv[5] * C_t_p_inv[1]) /
     ↪  det_C_t_p_inv;
}

//----------------------------------------------------------------------
//
// Initialize Model
//
//----------------------------------------------------------------------

void MD_Viscoplastic_Initialize( const int num_gauss_pts,
                                 const int num_sv,
                                 double * sv_old,
                                 double * sv) {

    for ( int ie = 0 ; ie < num_gauss_pts ; ie++ ){
        // Equivalent transient viscoplastic strain
        sv_old[0] = 0.0;
        sv[0] = 0.0;
        // Equivalent viscoplastic strain
        sv_old[1] = 0.0;
        sv[1] = 0.0;
        // Equivalent stress
        sv_old[2] = 0.0;
        sv[2] = 0.0;
        // Viscoplastic rate scale factor
        sv_old[3] = 1.0;
        sv[3] = 1.0;
        // pointer arithmetic
        sv_old  += num_sv;
        sv  += num_sv;
    }
}

//----------------------------------------------------------------------
//
// Property Check
//
//----------------------------------------------------------------------

void MD_Viscoplastic_PropertyCheck( std::vector<double> & prop ) {

    bool fatal_errors = false;
    std::string message_error = "\n";

    if ( prop[0] < 0.0 ){
        message_error += "Error MD VISCOPLASTIC Model:";
        message_error += " 'SHEAR MODULUS' must be positive\n";
        fatal_errors = true;
    }

    if ( prop[1] < 0.0 ){
        message_error += "Error MD VISCOPLASTIC Model:";
        message_error += " 'BULK MODULUS' must be positive\n";
        fatal_errors = true;
    }

    if ( prop[3] < 4.0 ){
        message_error += "Error MD VISCOPLASTIC Model:";
        message_error += " 'a' must be greater than or equal to 4\n";
        fatal_errors = true;
    }

    if (fatal_errors){
        message_error += "\nFatal Errors in MD VISCOPLASTIC model\n";
        Material::reportError(3,message_error);
    }
}

}
```

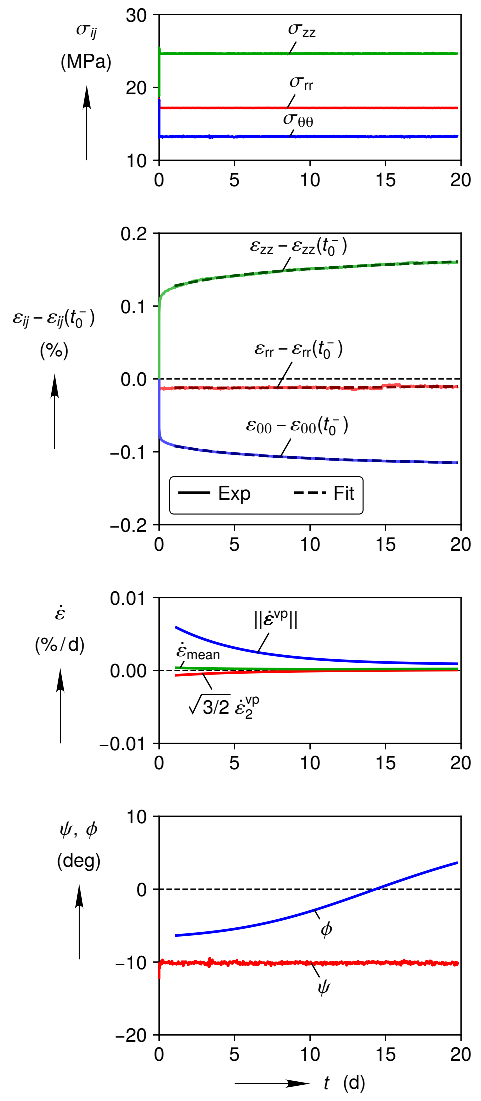
Appendix D

Analysis of Hollow Cylinder Experiments

This chapter contains plots of the seven Mellegard et al. (1992) hollow cylinder experimental results. Each series of plots in Figs. D.1a, D.1b, D.2a, D.2b, D.3a, D.3b and D.4 depicts the applied stress, Lode angle, and measured strain histories for each experiment. Each plot series also shows how the strain histories were fit using Eq. (5.4) in order to calculate the strain rate and subsequently calculate the flow potential normal angle. See Section 5.2 for further description of these plots and for a discussion of how the results were used to fit the Hosford exponent a.

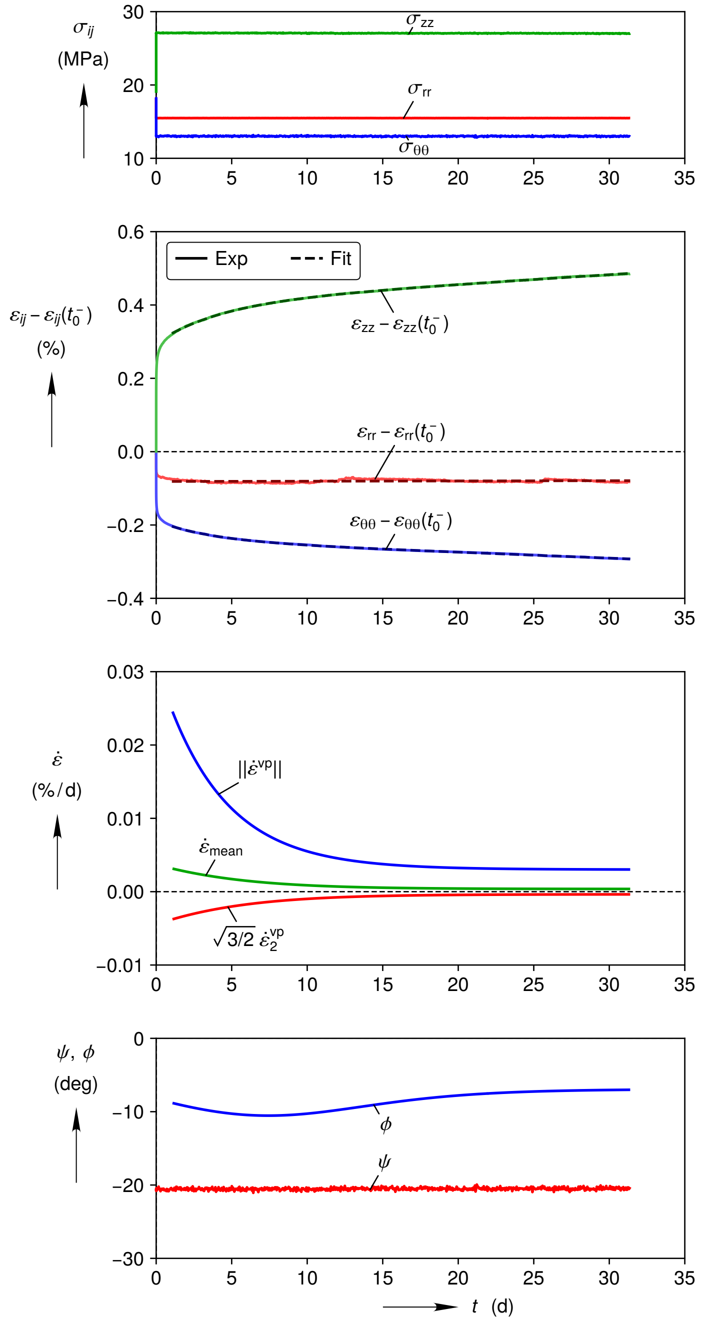


(a) Experiment AI/82/C'1



(b) Experiment AI/86/A'1/1

Figure D.1: Analysis of hollow cylinder experiment AI/82/C'1 and AI/86/A'1/1


(a) Experiment AI/86/A'10/1

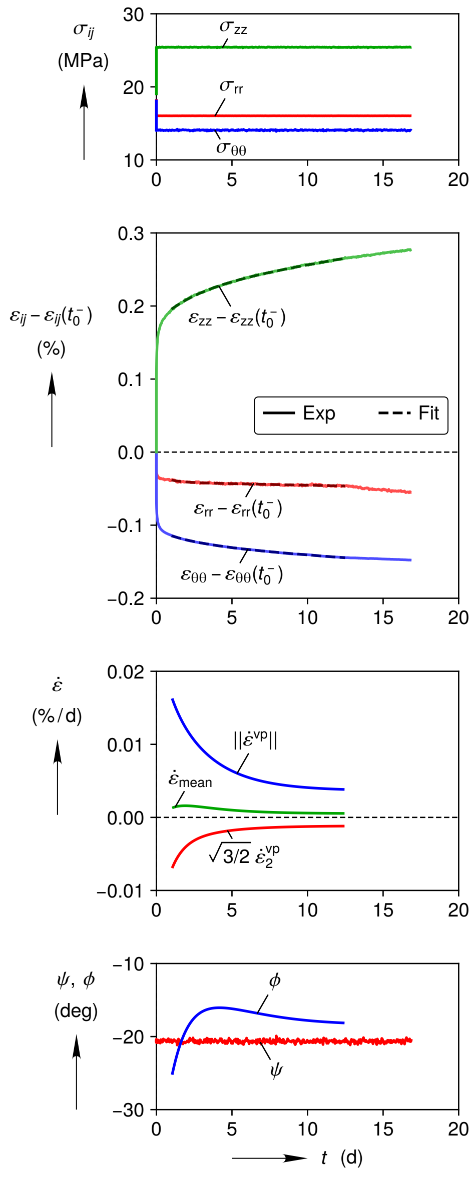
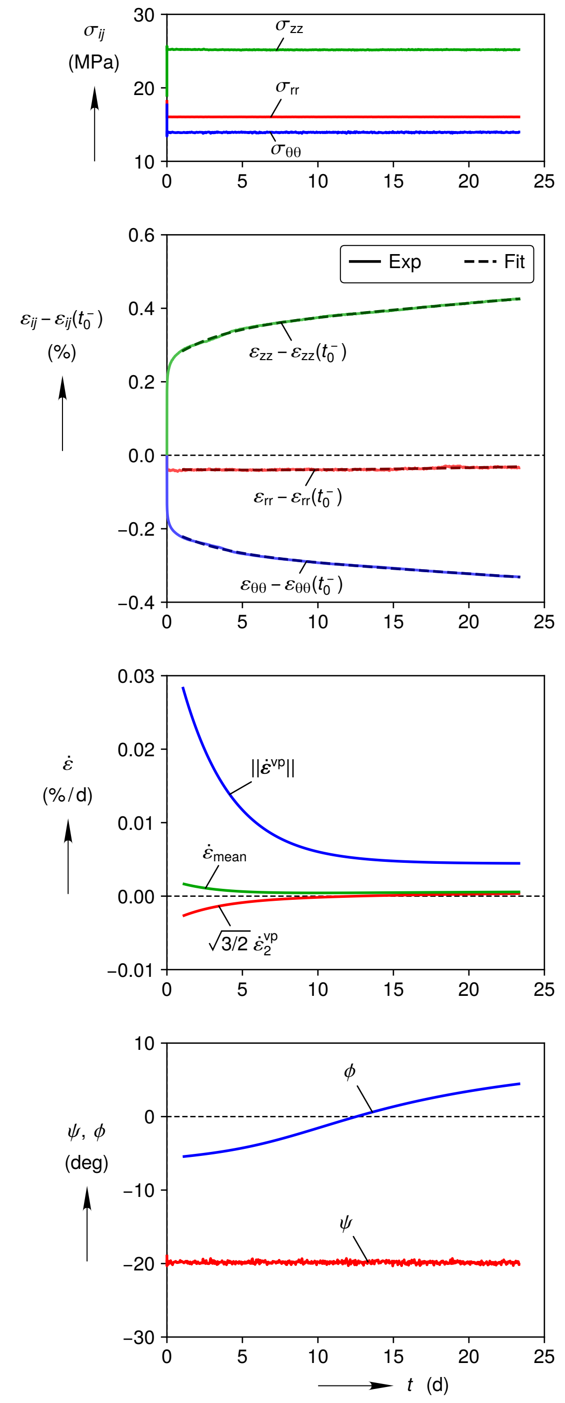
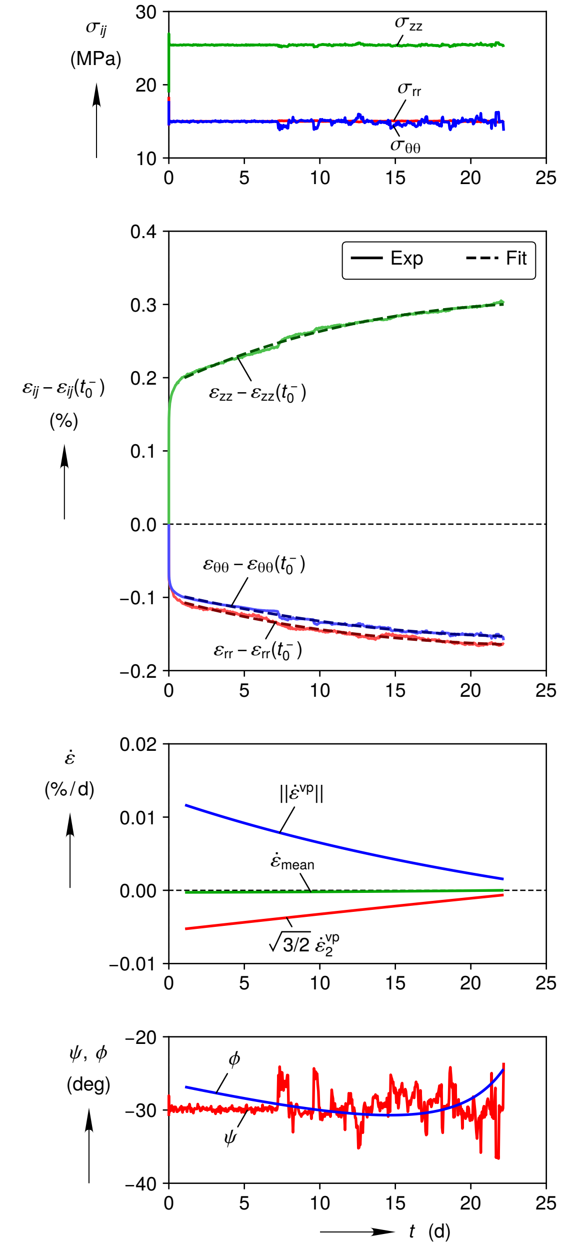

(b) Experiment AI/86/A'12/1

Figure D.2: Analysis of hollow cylinder experiment AI/86/A'10/1 and AI/86/A'12/1



(a) Experiment AI/86/C'4/1



(b) Experiment AI/82/C'7

Figure D.3: Analysis of hollow cylinder experiment AI/86/C'4/1 and AI/82/C'7

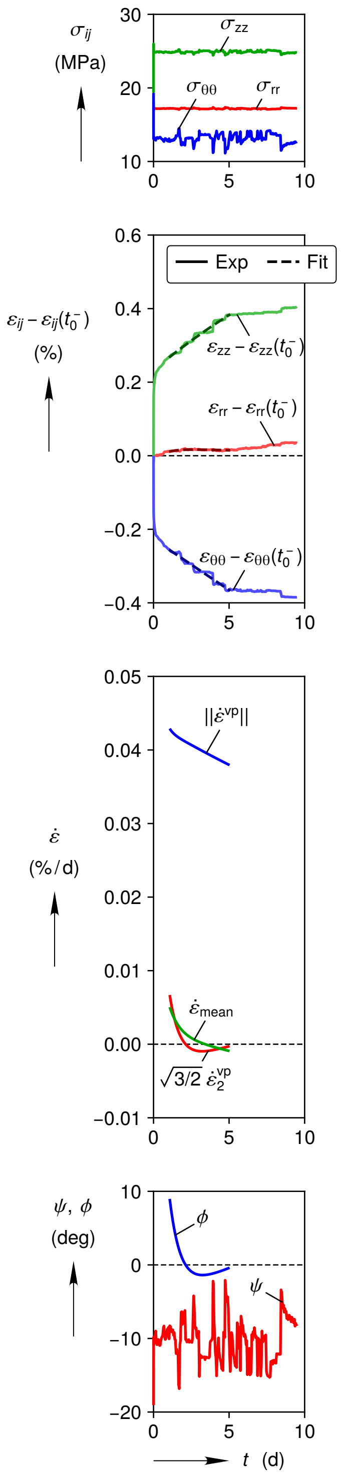


Figure D.4: Analysis of hollow cylinder experiment AI/86/C'3/1 (discarded due to stress control issues)

DISTRIBUTION:

1 J. Guadalupe Argüello
8328 S. Sandoval Ct. N.E.
Albuquerque, NM 87122

1 Frank Hansen
3005 La Villita Place NE
Albuquerque, NM 87111

1 Andreas Hampel
Grünberger Straße 56
55129 Mainz
Germany

2 Leibniz Universität Hannover
Institut für Geotechnik
Abteilung Unterirdisches Bauen
Attn: Kurt Staudtmeister, Savas Yildirim
Welfengarten 1a
30167 Hannover
Germany

2 Technische Universität Braunschweig
Institut für Grundbau und Bodenmechanik
Attn: Joachim Stahlmann, Ida Epkenhans
Beethovenstraße 51b
38106 Braunschweig
Germany

3 Institut für Gebirgsmechanik GmbH
Attn: Klaus Salzer, Ralf-Michael Günther, Christoph Lüdeling
Friederikenstraße 60
4279 Leipzig
Germany

4 Technische Universität Clausthal
Institut für Aufbereitung, Deponietechnik und Geomechanik
Attn: Karl-Heinz Lux, Ralf Wolters, Kai Herschen, Uwe Düsterloh
Erzstraße 20
38678 Clausthal-Zellerfeld
Germany

4 RESPEC
Attn: Kirby Mellegard, Kerry DeVries, Leo Van Sambeek, Stuart Buchholz
3824 Jet Drive
Rapid City, SD 57703

2 Southwest Research Institute
Center for Nuclear Waste Regulatory Analyses
Attn: Goodluck Ofoegbu, Biswajit Dasgupta
6220 Culebra Rd.
P.O. Drawer 28510
San Antonio, TX 78228

1 Dennis Powers
170 Hemley Rd.
Anthony, TX 79821

1 Lawrence Berkeley National Laboratory Nuclear Energy and Waste Program
Attn: Jens Birkholzer
One Cyclotron Road, Room 308, Mail Stop 74R316C
Berkeley, CA 94720

2 US Department of Energy
Attn: Prasad Nair, Tim Gunter
DOE-NE
232 Energy Way
North Las Vegas, NV 89030

1 US Nuclear Regulatory Commission
Office of Nuclear Material Safety and Safeguards
Attn: Jin-Ping Gwo
11545 Rockville Pike
Rockville, MD 20852

1	MS 0735	E. Webb, 8860
1	MS 0735	C. Herrick, 8864
1	MS 0734	C. Leigh, 8888
1	MS 0747	R. MacKinnon, 8844
1	MS 0747	M. Mills, 8844
1	MS 0747	K. Kuhlman, 8844
1	MS 0747	E. Hardin, 8844
1	MS 0751	S. Sobolik, 8862
1	MS 0751	B. Park, 8862
1	MS 0779	K. McMahon, 8842

1	MS	0779	E. Matteo, 8842
1	MS	0836	M. Martinez, 1516
1	MS	0840	J. Bean, 1554
1	MS	0840	W. Scherzinger, 1554
1	MS	0840	B. Lester, 1554
1	MS	0840	E. Fang, 1554
1	MS	0840	J. Pott, 1550
1	MS	0840	J. Rath, 1555
1	MS	1033	S. Bauer, 8866
1	MS	1033	S. Broome, 8864
1	MS	1395	P. Shoemaker, 8880
1	MS	1395	T. Zeitler, 8881
1	MS	1395	B. Day, 8881
1	MS	1395	D. Kicker, 8881
1	MS	1395	R.C. Camphouse, 8881
1	MS	1395	C. Sisk-Scott, 8882
1	MS	1395	R. Kirkes, 8883
1	MS	1395	S. Dunagan, 8883
1	MS	1395	G. Duran, 8883
1	MS	0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories