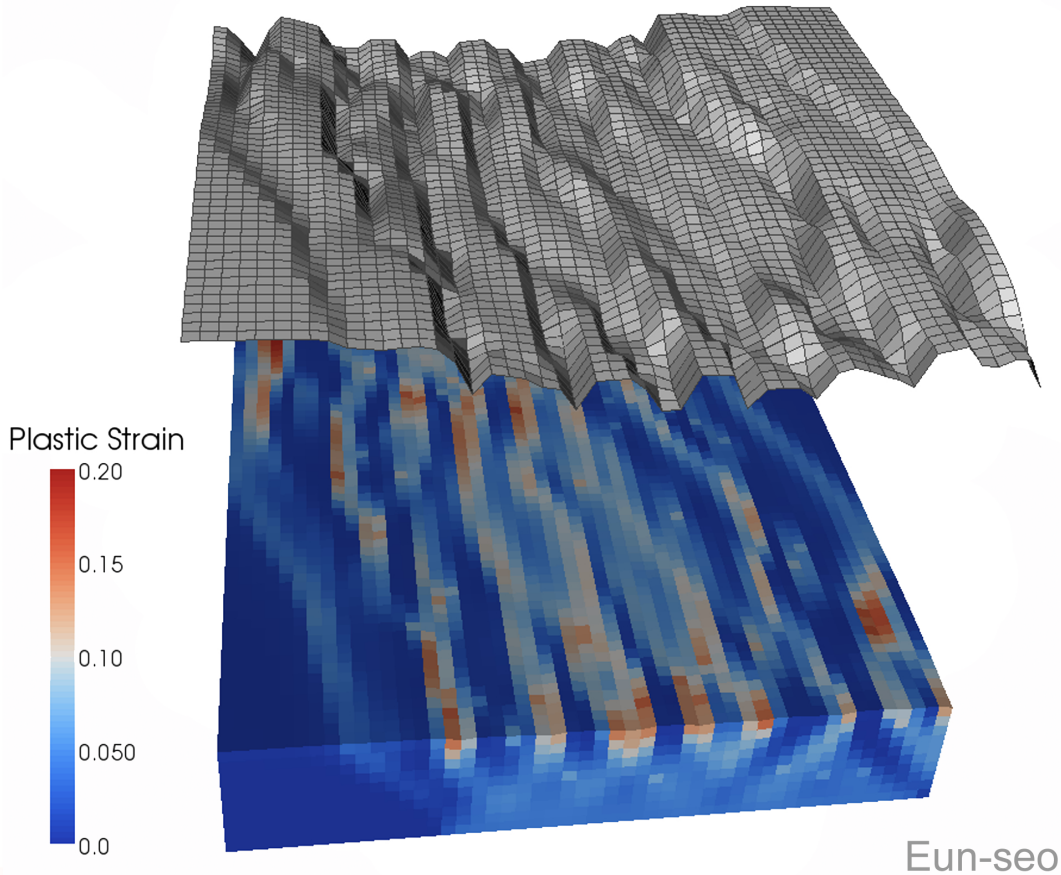
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

SNAC

User Manual

Version 1.2



Eun-seo Choi
Michael Gurnis
Susan Kientz
Colin Stark

www.geodynamics.org

SNAC User Manual

Version 1.2

April 12, 2010

About the cover: Pictured is the plastic strain distribution over a deformed block of elasto-plastic material. While similar to the example problem in Section 5.1, the domain is larger ($100 \times 20 \times 100$ km) and a distributed loading is applied on the bottom surface to simulate an oblique rifting. The warped surface on top of the domain represents 10 times exaggerated surface topography, and the relief between the highest and the lowest point is about 1 km.

Contents

List of Figures

Part I

Preface

Preface

About This Document

This document is organized into three parts. Part I consists of traditional book front matter, including this preface. Part II begins with an introduction to Pyre and the Pyre-compatible version of SNAC and their capabilities and proceeds to the details of implementation. Part III provides appendices and references.

The style of this publication is based on the Apple Publications Style Guide (`developer.apple.com/documentation/UserExperience/Conceptual/APStyleGuide/AppleStyleGuide2003.pdf`), as recommended by Python.org (`www.python.org`). The documentation was produced using LyX (`www.lyx.org`) to facilitate the transformation of files from one format to another. LyX is a document processor that encourages an approach to writing based on the structure of your documents, not their appearance. It is released under a Free Software/Open Source license.

Errors and bug fixes in this manual should be directed to the CIG Long-Term Tectonics Mailing List (`cig-long@geodynamics.org`).

Who Will Use This Document

This documentation is aimed at scientists who prefer to use prepackaged and specialized analysis tools. Users are likely to be experienced computational earth scientists and have familiarity with basic scripting, software installation, and programming; but are not likely to be professional programmers. Of those, there are likely to be two classes of users: those who just run models and those who modify the source code.

Citation

Computational Infrastructure for Geodynamics (CIG) is making this source code available to you in the hope that the software will enhance your research in geophysics. A number of individuals have contributed a significant portion of their careers toward the development of SNAC. It is essential that you recognize these individuals in the normal scientific practice by citing the appropriate peer reviewed papers and making appropriate acknowledgements.

The SNAC development team asks that you cite

- Choi, E., L. Lavier, and M. Gurnis (2008), Thermomechanics of mid-ocean ridge segmentation, *Phys. Earth Planet. Inter.,* 171, 374-386, *(dx. doi. org/ 10. 1016/ j. pepi. 2008. 08. 010).*

The developers also request that in your oral presentations and in your paper acknowledgements that you indicate your use of this code, the authors of the code, and CIG (`geodynamics.org`).

Support

SNAC development is funded by the U.S. Dept. of Energy's Advanced Simulation and Computing program (`www.sandia.gov/NNSA/ASC`) and the National Science Foundation's (`www.nsf.gov`) Information Technology Research (ITR) program (grant #0205653). Continued support of SNAC is based upon work supported by the National Science Foundation under Grant No. EAR-0406751. Any opinions, findings, and conclusions

or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Part II

Chapters

Chapter 1

Introduction

SNAC (StGermaiN Analysis of Continua) is an updated Lagrangian explicit finite difference code for modeling a finitely deforming elasto-visco-plastic solid in 3D, released under the GNU General Public License (see Chapter 8 on page 65). In this code, nodal velocities satisfying a weak-form of the momentum balance are obtained as the nodal solution. SNAC shares a mathematical foundation, and thus major advantages, with a standard finite element method (FEM). However, it departs from the FEM by not making explicit use of shape functions. A Cartesian mesh consisting of 4-node linear or constant-strain tetrahedral elements is used to represent a discretized domain, although a spherical domain can also be used. On top of the tetrahedral discretization, a coarser discretization is constructed by zones, which are defined by eight nodes like a hexhedral element in the standard FEM and subdivided into two layouts of five tetrahedral elements for symmetric response. To avoid the over-stiff response of tetrahedrons in the incompressible limit, mixed discretization is applied [1]. The mixed discretization relieves over-stiffness by replacing the first invariant of tetrahedral strain-rate tensor with the one averaged over a zone.

1.1 SNAC Implementation

1.1.1 Governing Equations

Momentum balance for continuum in current or deformed configuration is as given below:

$$\begin{aligned} \tfrac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i = \rho \tfrac{Dv_i}{Dt},\ in\ \Omega, \\ v_i = g_i\ on\ \Gamma_g, \\ t_i = h_i\ on\ \Gamma_h, \end{aligned} \tag{1.1}$$

where $\partial\Omega$, the domain boundary, is the union of disjoint subsets, Γ_h and Γ_g, σ_{ij}, g_i, and v_i are components of the Cauchy stress tensor, the gravitational acceleration, and the velocity components, respectively. $\frac{D}{Dt}$ is the material or total derivative, and equal to the partial derivative with respect to time in SNAC since the updated Lagrangian viewpoint is taken.

1.1.2 Spatial Descritization

A 3D domain is discretized into hexahedral elements, each of which is filled with two sets of 5 tetrahedra (Fig. 1.1a). In this mesh hierarchy, called the mixed discretization [1], hexahedral elements are used mostly as an averaging unit for volumetric strain. The averaging is enforced at all times although needed only for incompressible viscoelastic or plastic constitutive laws. This is a conservative choice and does influence the overall performance of SNAC. The use of two equivalent sets of tetrahedra is required to ensure a symmetric response. For a given loading, responses of one set of tetrahedra can be different from those of the other set because of the differently orientated faces of tetrahedra in each set, e.g. [10].

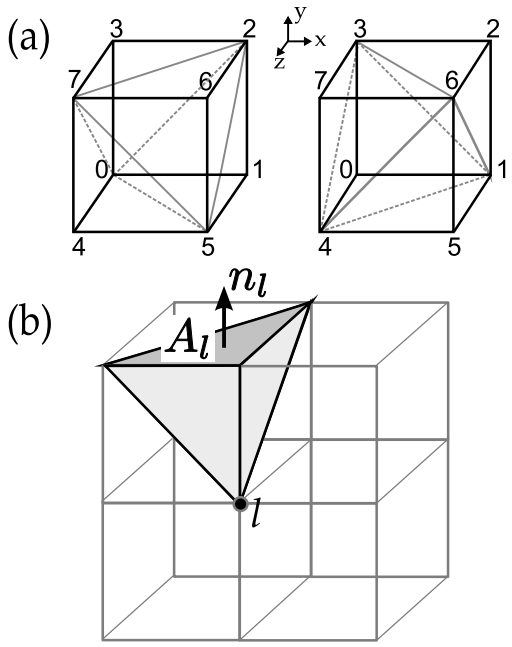


Figure 1.1: Configurations of tetrahedra and conventions for the notation. (a) Two configurations of five tetrahedra in a hexahedral element used in the mixed discretization. Numbers next to apexes indicate the local node numbering. (b) Conventions for the notation. A_l and n_l denote the face and the unit normal vector, respectively, associated with a local node l.

1.1.3 Approximation of Partial Derivatives

The approximation of partial derivatives with respect to spatial variables follows the integral definitions, e.g., [3]:

$$\int_\Omega f_{,i} dV = \int_{\partial\Omega} f n_i d\Gamma, \tag{1.2}$$

where Ω represent a tetrahedron as an integration domain, $\partial\Omega$ is the boundary surfaces of the tetrahedron, $f_{,i}$ is the partial derivative of a variable f with respect to i-th spatial coordinate, n_i is the i-th component of the unit normal vector of the surface. If the partial derivative is constant within a tetrahedron, it is evaluated as

$$f_{,i} = \frac{1}{V} \int_{\partial\Omega} f n_i d\Gamma, \tag{1.3}$$

where V is the volume of the tetrahedron. By further substituting an algebraic expression for the surface integral, reordering terms, and using $\int_{\partial\Omega} n_i d\Gamma = 0$ (when $f = 1$ in Eq. 1.3),

$$\begin{aligned} f_{,i} &= \frac{1}{V} \sum_{l=1}^{4} \bar{f}^l n_i^l A^l = \frac{1}{V} \sum_{l=1}^{4} \frac{1}{3} \sum_{m=1,\neq l}^{4} f^m n_i^l A^l \\ &= \frac{1}{3V} \sum_{m=1}^{4} f^m \sum_{l=1,\neq m}^{4} n_i^l A^l \\ &= -\frac{1}{3V} \sum_{m=1}^{4} f^m n_i^m A^m, \end{aligned} \tag{1.4}$$

where l is the local node index varying from 1 to 4, A^l and n^l are the area and the unit normal vector of the triangular surface not having the node l as one of its apexes (Fig. 1.1b). Hereafter, we call such a face a *corresponding* face to node l. $\bar{f}^l$ is the averaged f on the surface l.

1.1.4 Nodal Assemblage

We can convert the differential equation for momentum balance (Eq. 1.1) (the following description is applied to the heat equation in the same fashion) to a principle of minimum work rate as in the standard finite element formulation:

$$\int_\Omega \delta v_i \rho \frac{Dv_i}{Dt} dV = \int_\Omega \delta v_i \rho g_i dV + \int_\Omega \delta \xi_{ij} \sigma_{ij} dV, \tag{1.5}$$

where ξ_{ij} are components of the strain rate tensor, δv_i and $\delta \xi_{ij}$ represent variations of velocity and strain rate, and Ω here corresponds to the whole domain. The local contribution to nodes corresponding to each term can be computed by following the standard finite element procedure for linear tetrahedral elements. However, our method does not need to construct coefficient matrices such as mass and stiffness matrices since it adopts an explicit time discretization. The resultant momentum equation is

$$M^n \frac{Dv_i^n}{Dt} = \frac{1}{3} T_i^{[n]} + \frac{1}{4} \rho^{[n]} g_i V^{[n]}, \tag{1.6}$$

where the superscript n represents values evaluated at the global node n, the superscript $[n]$ means the sum of contributions from all the tetrahedra having the global node n as an apex, T_i is the traction that is defined as $\sigma_{ij} n_j$ and evaluated on a face of one of the contributing tetrahedra. Any applied traction boundary conditions should be explicitly computed and added on the right hand side of (Eq. 1.6). The nodal mass M^n is not the actual inertial mass but an adjusted one to satisfy a local stability criterion discussed in Section 1.1.6. The correspondence between an apex and a face for the traction calculation is determined as in the derivation of the expression, (Eq. 1.4). Note that the factor of $\frac{1}{3}$ in the traction term is inherited from Eq. 1.4, and the factor of $\frac{1}{4}$ in the body force term implies that the nodal contribution takes one quarter of a tetrahedron's volume-dependent quantity.

While looping over the entire set of nodes, mass and nodal forces are assembled by adding up the contributions from boundary conditions and all the tetrahedra sharing that node as one of their apexes. The structured mesh of SNAC renders the assemblage step conveniently static. The acquired net force (or heat flux) at each node is used to update velocities and node coordinates (or temperature).

1.1.5 Solution Scheme

We seek static or quasi-static solutions through a dynamic relaxation method. Instead of adding a usual velocity-dependent friction term, we adopt a local non-viscous damping scheme [16]:

$$F_i^{damped} = F_i - \alpha \operatorname{sgn}(v_i) \mid F_i \mid, \tag{1.7}$$

where F_i is the i-th component of the residual force vector (the right hand side of Eq. 1.6), α is a positive coefficient less than 1, $\operatorname{sgn}(v_i)$ returns the sign of the i-th component of velocity, v_i. Once net forces are assembled and damped, velocity at that node is updated using a forward Euler method:

$$v(t + \frac{\Delta t}{2}) = v(t - \frac{\Delta t}{2}) + \Delta t \frac{F_i^{damped}}{M} \tag{1.8}$$

$$x(t + \Delta t) = x(t) + \Delta t v(t + \frac{\Delta t}{2}). \tag{1.9}$$

Damping is irrelevant to the update of temperature field, but the same forward Euler method is used.
There are a couple of SNAC-specific numerical artifacts. First, as in the case of an under-damped oscillator, SNAC's solutions will exhibit artificial oscillations, but these are only transient and do not affect the static equilibrium. Another notable artifact is the randomness in the magnitude of residual forces. Suppose the magnitude of assembled residual force is very small compared to those of the contributing internal and external forces. It means that many significant figures in floating-point numbers are lost during the assemblage. The residual force ends up with having random numbers in the floating number corresponding to it. This is related to a fundamental issue of representing real numbers with floating point numbers.

1.1.6 Mass Scaling for Numerical Stability

The conventional Courant-Friedrichs-Lewy (CFL) condition imposes a stringent upper limit for the time step size such that dynamic relaxation takes a long time to get the quasi-static solution over a geological time scale. To overcome this limit, a mass scaling technique is applied. This technique adjusts each nodal mass such that the stability condition for a user-specified time step can be locally satisfied. The stability condition to be satisfied, however, is not the same as in the CFL condition, i.e., $\Delta t \leq (l_{min}/v_p)$, where Δt is the time step, l_{min} is the minimum element size, and v_p is the P wave velocity. Instead, through an analogy of continuum to an infinite mass-spring system, we use a criterion that does not explicitly include length scale and P wave velocity; see Chapter 9 in Bathe [11]:

$$\Delta t \leq \frac{T}{\pi}, \tag{1.10}$$

where T is the period of system, $2\pi(m/k)^{1/2}$, m is a point mass, and k is the stiffness of the spring attached to the point mass. Now, reducing the infinite series of mass and springs in one dimension to a single mass-spring system, the stiffness of that single system becomes $4k$, leading to an expression for the mass scaling:

$$m \geq k(\Delta t)^2. \tag{1.11}$$

For a given size of Δt, the nodal mass is adjusted according to Eq. 1.11 to automatically satisfy the stability criterion, Eq. 1.10. The value of k is computed by equating internal force contribution at a node with $-ku_i$:

$$\begin{aligned} &\frac{1}{3}T_i = -ku_i \\ \Rightarrow &\frac{1}{3}\left(\lambda + 2\mu\right)\left(\dot{\epsilon_{ii}}dt\right) n_i A = -k\left(v_i dt\right) \ \ (\text{no sum}), \end{aligned} \tag{1.12}$$

where only the volumetric contribution from internal forces is taken into account. By substituting the approximation (Eq. 1.4) for the partial derivative, $\dot{\epsilon_{ii}}$, into the above equation and dividing both sides by $v_i dt$, we obtain

$$k_i^l = \frac{1}{9V}(\lambda + 2\mu)(n_i^l A^l)^2, \tag{1.13}$$

where l is the local index for apexes of a tetrahedron, and the surface-related quantities are computed on the corresponding face of the tetrahedron. Finally, a tetrahedron's contribution to the scaled mass is given as

$$m^l = \frac{\lambda + 2\mu}{9V} \max[(n_i^l A^l)^2, i = 1, \ldots, 3]. \tag{1.14}$$

As in the standard FEM, appropriate mappings between local and global indices are required.

1.1.7 Constitutive Update

SNAC uses a general elasto-visco-plastic rheological model to update the Cauchy stress tensor (e.g., [12]). First, the initial guess of stress is acquired by the Maxwell viscoelastic constitutive law [13]. If this initial guess exceeds a given yield stress, it is projected onto the yield surface using a return mapping method [14]; otherwise, the viscoelastic stress update is retained. This elasto-visco-plastic model can deal with various constitutive laws that are typically used for the Earth's crustal and mantle material as its limiting cases. For example, elastic, viscoelastic and elastoplastic material are realized in the following cases:

1. Elastic material corresponds to the limit of infinite viscosity and yield stress.
2. Viscoelastic material corresponds to the limit of infinite yield strength.
3. Elasoplastic material corresponds to the inifinte viscosity.

Using the viscoplastic rheology is physically more realistic than using one of the limiting cases listed above since all materials have dissipative mechanisms and hence viscosity. This viscosity also provides a length scale for the problem of localization, which in turn enables a physically meaningful mesh independent solution when the mesh size is smaller than this length scale.

Since the nodal variables are velocities and whose spatial gradients are deformation rates, we formulate the constitutive update in terms of strain rate. The objective stress rate of choice is the Jaumann or the corotational stress rate ($\Delta\sigma^{\Delta J}$) [15].

$$\Delta\sigma^{\Delta J} = \frac{\partial(\Delta\sigma)}{\partial t} - W\cdot\Delta\sigma - \Delta\sigma\cdot W^T, \tag{1.15}$$

where $W_{ij} = (1/2)(\partial v_i/\partial x_j$-$\partial v_j/\partial x_i)$ are the components of spin tensor and $\Delta\sigma$ is the increment of stress tensor. Correction to the stresses due to rotation can be given as

$$\sigma^{t+\Delta t} = \sigma^t + \Delta\sigma^{\Delta J}.\Delta t \tag{1.16}$$

During the viscoplastic constitutive update, the initial guess of stress increment is first predicted by Maxwell viscoelastic constitutive law. If the guessed stress increment exceeds a given yield criterion, it is projected onto the yield surface using a return mapping method; otherwise, the viscoelastic stress update is kept. The detailed algorithm of stress update at each quadrature point and each time step is as follows:

1. Viscoelastic predictor: Obtain viscoelastic stress increment predictor with Maxwell constitutive update corresponding to the strain increment. Since viscous relaxation affects only the deviatoric part of stress, the deviatoric stress update is given by

$$S_{ij}^{n+1} = \left(C_1 S_{ij}^n + 2G\Delta\varepsilon_{ij}^d\right) C_2 \tag{1.17}$$

   where $S_{IJ}\left(=\sigma_{ij} - \frac{1}{3}\delta_{ij}\sigma_{kk}\right)$ is the deviatoric stress, ε_{ij}^d is the deviatoric part of the strain increment $\varepsilon_{ij}\left(\varepsilon_{ij} = \frac{D\varepsilon_{ij}}{Dt}\Delta t\right)$, $C_1 = 1 - \frac{G\Delta t}{2\eta}$, $C_2 = \frac{1}{1+\frac{G\Delta t}{2\eta}}$ and G is the shear modulus. Volumetric elastic stress increment is given by $\Delta\sigma_{kk} = 3K\Delta\varepsilon_{kk}$ which gives viscoelastic predictor update at t_{n+1}

$$\sigma_{ij}^{n+1} = S_{ij}^{n+1} + \frac{1}{3}\left(\sigma_{kk}^n + \Delta\sigma_{kk}\right) \tag{1.18}$$

2. Plastic correction: If σ^{n+1} is inside or on the yield surface, i.e., $f\left(\sigma^{n+1}\right) \geq 0$, where f is the yield function, then it does not need plastic correction. If σ^{n+1} is outside the yield surface, then yielding occurs and we project σ^{n+1} onto the yield surface using a return-mapping algorithm [14].

In general, for frictional materials, the yield function can be written as

$$f\left(\sigma^{n+1}\right) = q_\phi\sigma_p + C - \tau \tag{1.19}$$

in which τ represents shear stress and σ_p represents normal stress, q_ϕ is a function of the friction angle ϕ, and C is cohesive strength of the material.

In the case of Mohr-Coulomb material, q_ϕ is given as $\tan\phi$,

$$\tau = \frac{1}{2}\left(\sigma_3 - \sigma_1\right)\cos\phi \text{ and } \sigma_p = \frac{1}{2}\left(\sigma_3 + \sigma_1\right) - \frac{1}{2}\left(\sigma_3 - \sigma_1\right)\sin\phi, \tag{1.20}$$

where $\sigma_1 \leq \sigma_2 \leq \sigma_3$ are the principal stresses of stress tensor (Figure 1.2). The actual form of the yield function for *shear failure* used in SNAC is

$$f_s(\sigma_1, \sigma_3) = \sigma_1 - N_\phi\sigma_3 + 2C\sqrt{N_\phi}, \tag{1.21}$$

where $N_\phi = \frac{1+\sin\phi}{1-\sin\phi}$ and $\sqrt{N_\phi} = \frac{\cos\phi}{1-\sin\phi}$ (Figure 1.3). The tensile yield function is defined as

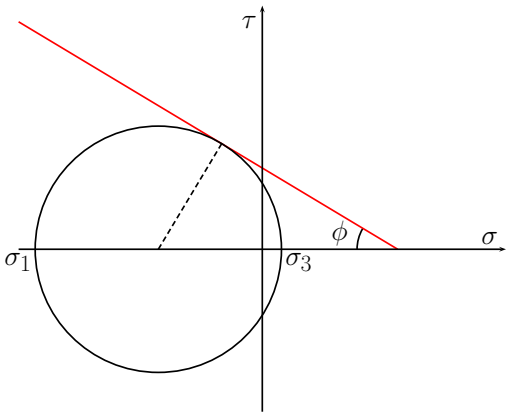


Figure 1.2: A diagram showing a Mohr-Coulomb yield envelope with $\tan\phi = 0.6$ and a non-zero cohesion as well as a Mohr circle corresponding to the principal stresses, σ_1 and σ_3.

$$f_t(\sigma_3) = \sigma_3 - \sigma_t, \tag{1.22}$$

where σ_t is the tension cut-off. If the tension cut-off is given as a parameter, a smaller value between the theoretical limit, $2C/\tan\phi$, and the given value is assigned to σ_t. To make sure a unique decision on the mode of yielding, shear vs. tensile, we define additional function, $h(\sigma_1, \sigma_3)$, which bisects the obtuse angle made by two yield functions (Figure 1.3) as

$$f_h(\sigma_1, \sigma_3) = \sigma_3 - \sigma_t + (\sqrt{N_\phi^2 + 1} + N_\phi)(\sigma_1 - N_\phi\sigma_t + 2c\sqrt{N_\phi}), \tag{1.23}$$

Once yielding is declared (i.e., $f_s < 0$ or $f_t > 0$), then shear or tensile failure is finally decided based on the value of h: *shear if* $h < 0$ *and tensile otherwise.*

In general, flow rule for frictional materials is non-associative, i.e., flow direction differs from the normal of the yield surface normal. As in the definitions of yield functions, the plastic flow potential for the *shear* failure in the Mohr-Coulomb model can be given as

$$g_s(\sigma_1, \sigma_3) = \sigma_1 - N_\psi\sigma_3 \tag{1.24}$$

where $N_\psi = \frac{1+\sin\psi}{1-\sin\psi}$ andψ is the dilation angle. Likewise, the tensile flow potential is given as

$$g_t(\sigma_3) = \sigma_3 - \sigma_t \tag{1.25}$$

In the presence of plasticity, the total strain, $\Delta\varepsilon$, is given by

$$\Delta\varepsilon = \Delta\varepsilon^e + \Delta\varepsilon^p, \tag{1.26}$$

where $\Delta\varepsilon^e$ is the elastic, and $\Delta\varepsilon^p$ is the plastic strain increment. We assume that if plastic yielding implies negligible viscous flow.

The plastic strain increment is

$$\Delta\varepsilon^p = \beta\frac{\partial g}{\partial\sigma}, \tag{1.27}$$

where β is a proportionality constant to be determined. β is determined such that updated stress state is on the yield surface, i.e., $f(\sigma^n + \Delta\sigma^n) = 0$, where $\Delta\sigma^n = \mathbf{C} : (\Delta\varepsilon^n - \Delta\varepsilon^p)$ and $\mathbf{C}$ is the elastic moduli tensor. This condition is called the *consistency condition* and the parameter β is thus called the *consistency parameter* [14]. In the principal component representation, $\sigma_A = a^e_{AB}e_B$ where σ_A and ϵ_A are principal stress and strain, respectively, and $\mathbf{a}^e$ is a corresponding elastic moduli matrix of which components are given in terms of Lame's constants: $a^e_{AB} = \lambda + 2\mu\delta_{AB}$. By expanding each yield function and using$\sigma^{TR} = \sigma^n + \mathbf{C} : \Delta\varepsilon^n$, we get the following formulae for β:

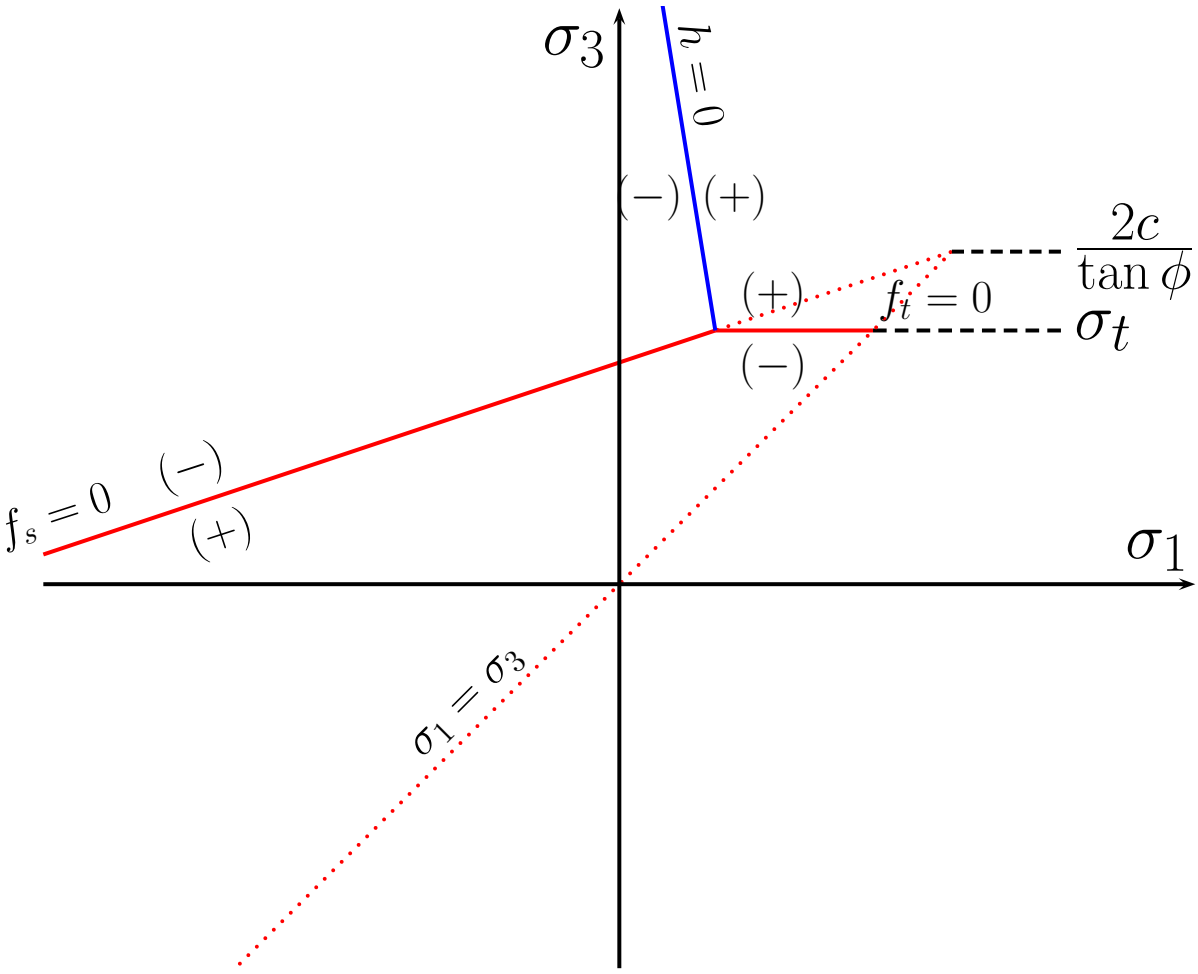


Figure 1.3: Yield functions that are used to declare yielding in shear and tensile mode.

- In case of shear failure

$$\beta = \frac{\sigma_1^{TR} - N_\phi \sigma_3^{TR} + 2c\sqrt{N_\phi}}{a_{1B}^e \frac{\partial g_s}{\partial \sigma_B} - N_\phi a_{3B}^e \frac{\partial g_s}{\partial \sigma_B}} \tag{1.28}$$

- In case of tensile failure

$$\beta = \frac{\sigma_3^{TR} - \sigma_t}{\frac{\partial g^t}{\partial \sigma_B}} \tag{1.29}$$

Hardening/softening If strain hardening (including softening as a negative hardening) is considered, a more general return mapping is required. For completeness, we review the *cutting-plane* algorithm and then discuss the simplified version for (piecewise-)linear hardening implemented in SNAC. For more detailed discussion, readers are referred to [14].

Here are some preliminary definitions. From (1.27), the plastic strain rate is given as

$$\dot{\epsilon}_i^p = \dot{\beta} \frac{\partial G}{\partial \sigma_i} \tag{1.30}$$

We define an internal variable such that plastic parameters are defined as a function of that variable. Our particular choice is

$$\epsilon^{p*} = \sqrt{\frac{1}{2}\left\{(\epsilon_1^p - \bar{\epsilon^p})^2 + (\epsilon_2^p - \bar{\epsilon^p})^2 + (\epsilon_3^p - \bar{\epsilon^p})^2 + \bar{\epsilon^p}^2\right\}} \tag{1.31}$$

where $\bar{\epsilon^p} = \frac{1}{3}(\epsilon_1^p + \epsilon_2^p + \epsilon_3^p)$. The time rate of change of the internal variable is then

$$\begin{aligned} \dot{\epsilon}^{p*} &= \frac{\partial \epsilon^{p*}}{\partial \epsilon_i^p} \dot{\epsilon}_i^p \\ &= \dot{\beta} \frac{\partial \epsilon^{p*}}{\partial \epsilon_i^p} \frac{\partial G(\sigma, \epsilon^{p*})}{\partial \sigma_i} \\ &= -\dot{\beta} r(\sigma, \epsilon^{p*}) \end{aligned} \tag{1.32}$$

For the simpler notation, a new function $r(\sigma, \epsilon^{p*})$ has been defined above.

We require that the updated stress stay on the yield surface (discrete consistency condition): i.e.,

$$\bar{F}(\Delta\beta) \equiv F(\sigma(\Delta\beta), \epsilon^{p*}(\Delta\beta)) = 0, \tag{1.33}$$

where $\Delta\beta$ is the increment of the consistency parameter during a time interval between $t_n$and t_{n+1}. For later uses, we also compute the derivative of $\bar{F}$ with respect to β:

$$\frac{d\bar{F}}{d\Delta\beta} = \frac{\partial F}{\partial \sigma_i} \cdot \frac{\partial \sigma_i}{\partial \Delta\beta} + \frac{\partial F}{\partial \epsilon^{P*}} \frac{\partial \epsilon^{p*}}{\partial \Delta\beta} \tag{1.34}$$

Since $\sigma_{n+1} = \mathbf{a}^e \cdot (\epsilon_{n+1} - \epsilon_{n+1}^p) = \mathbf{a}^e \cdot [\epsilon_{n+1} - (\epsilon_n^p + \Delta\beta \partial_\sigma G(\sigma_{n+1}, \epsilon_{n+1}^{p*}))]$ and ϵ_{n+1} and ϵ_n^p is constant during a time interval,

$$\frac{\partial \sigma_{n+1}}{\partial \Delta\beta} = -\mathbf{a}^e \cdot \frac{\partial G}{\partial \sigma_{n+1}} \tag{1.35}$$

From Eq. 1.32,

$$\frac{\partial \Delta\epsilon^{p*}}{\partial \Delta\beta} = -r \tag{1.36}$$

By substituting Eqs. 1.35 and 1.36 into Eq. 1.34, we get

$$\frac{d\bar{F}}{d\Delta\beta} = \frac{\partial F}{\partial \sigma} \cdot \left(-\mathbf{a}^e \cdot \frac{\partial G}{\partial \sigma}\right) + \frac{\partial F}{\partial \epsilon^{P*}}(-r) \tag{1.37}$$

Now we present the cutting-plane algorithm for updating the stress, internal variable and consistency parameters iteratively in case of a general non-linear hardening.

1. Initialization:

$$\begin{aligned}\epsilon_{n+1}^{p(0)} &= \epsilon_n^p \\ \epsilon_{n+1}^{p*(0)} &= \epsilon_n^{p*} \\ \Delta\beta_{n+1}^{(0)} &= 0\end{aligned}$$

2. Update stresses and evaluate F:

$$\begin{aligned}\sigma_{n+1}^{(k)} &= \mathbf{a}^e \cdot (\epsilon_{n+1} - \epsilon_{n+1}^{p(k)}) \\ F_{n+1}^{(k)} &= F(\sigma_{n+1}^{(k)}, \epsilon_{n+1}^{p*(k)})\end{aligned}$$

   If $F_{n+1}^{(k)} \leq 0$? exit; otherwise go to step 3.

3. Incremental upate:

$$\begin{aligned}\Delta^2\beta &= \frac{F_{n+1}^{(k)}}{-\frac{\partial F}{\partial \Delta\beta}} = \frac{F_{n+1}^{(k)}}{\frac{\partial F}{\partial \sigma}\cdot\mathbf{a}^e\cdot\frac{\partial G}{\partial \sigma} + \frac{\partial F}{\partial \epsilon^{p*}}r} \\ \epsilon_{n+1}^{p(k+1)} &= \epsilon_{n+1}^{p(k)} + \Delta^2\beta\frac{\partial G}{\partial \sigma}(\sigma_{n+1}^{(k)}, \epsilon_{n+1}^{p*(k)}) \\ \epsilon_{n+1}^{p*(k+1)} &= \epsilon_{n+1}^{p*(k)} - \Delta^2\beta r(\sigma_{n+1}^{(k)}, \epsilon_{n+1}^{p*(k)}) \\ &= \epsilon_{n+1}^{p*(k)} + \Delta^2\beta\left[\left(\frac{\partial \epsilon^{p*}}{\partial \epsilon^p}\right)_{n+1}^{(k)}\cdot\frac{\partial G}{\partial \sigma}(\sigma_{n+1}^{(k)}, \epsilon_{n+1}^{p*(k)})\right] \\ \Delta\beta_{n+1}^{(k+1)} &= \Delta\beta_{n+1}^{(k)} + \Delta^2\beta\end{aligned}$$

   Go back to step 2.

This algorithm is just a standard Newton method applied to the discrete consistency equation 1.33. Thus, it should be obvious that the plastic correction is accomplished by the single step update if the yield function is a linear function of the consistency variable. In that case, Eqs. 1.28 and 1.29 are immediately retrieved.

For reference, we further derive the explicit form of the function r in terms of principal plastic strains and stresses by working out the partial derivative, $\partial\epsilon^{p*}/\partial\epsilon^p$. From Eq. 1.31,

$$\begin{aligned}\frac{\partial \epsilon^{p*}}{\partial \epsilon_i^p} &= \frac{1}{2\epsilon^{p*}}\frac{1}{2}\left\{\sum_{j=1}^{3} 2(\epsilon_j^p - \bar{\epsilon^p})(\delta_{ij} - \frac{1}{3}) + \frac{2}{3}\bar{\epsilon^p}\right\} \\ \Longrightarrow \frac{\partial \epsilon^{p*}}{\partial \epsilon_1^p} &= \frac{1}{2\epsilon^{p*}}\left\{\frac{2}{3}(\epsilon_1^p - \bar{\epsilon^p}) - \frac{1}{3}(\epsilon_2^p - \bar{\epsilon^p}) - \frac{1}{3}(\epsilon_3^p - \bar{\epsilon^p}) + \frac{1}{3}\bar{\epsilon^p}\right\} \\ \frac{\partial \epsilon^{p*}}{\partial \epsilon_2^p} &= \frac{1}{2\epsilon^{p*}}\left\{-\frac{1}{3}(\epsilon_1^p - \bar{\epsilon^p}) + \frac{2}{3}(\epsilon_2^p - \bar{\epsilon^p}) - \frac{1}{3}(\epsilon_3^p - \bar{\epsilon^p}) + \frac{1}{3}\bar{\epsilon^p}\right\} \\ \frac{\partial \epsilon^{p*}}{\partial \epsilon_3^p} &= \frac{1}{2\epsilon^{p*}}\left\{-\frac{1}{3}(\epsilon_1^p - \bar{\epsilon^p}) - \frac{1}{3}(\epsilon_2^p - \bar{\epsilon^p}) + \frac{2}{3}(\epsilon_3^p - \bar{\epsilon^p}) + \frac{1}{3}\bar{\epsilon^p}\right\}\end{aligned}$$

With $\partial G/\partial\sigma$ being $[1, 0, -N_\psi]$, the function $r(\sigma, \epsilon^{p*})$ is given by

$$\begin{aligned}r(\sigma, \epsilon^{p*}) &= -\left(\frac{\partial \epsilon^{p*}}{\partial \epsilon^p}\right)\cdot\left(\frac{\partial G}{\partial \sigma}\right) \\ &= -\frac{1}{2\epsilon^{p*}}\left[\frac{1}{3}(2 + N_\psi)(\epsilon_1^p - \bar{\epsilon^p}) - \frac{1}{3}(1 - N_\psi)(\epsilon_2^p - \bar{\epsilon^p}) - \frac{1}{3}(1 + 2N_\psi)(\epsilon_3^p - \bar{\epsilon^p}) + \frac{1}{3}(1 - N_\psi)\bar{\epsilon^p}\right]\end{aligned}$$

Note that when $\psi \simeq 0°$, $N_\psi \simeq 1$ and $\bar{\epsilon^p} \simeq 0$. Then, the value of r becomes ~ -1. Since the dilation angle is often set to be zero or a small value, and the cohesion is always defined as a piecewise linear function, F is approximately linear with regard to the consistency parameter. The current consistency parameter computation implemented in SNAC is thus justified. However, if any non-linearity is introduced such as friction angle varying with the internal variable and non-linearly changing cohesion, the interation presented above should be performed.

1.1.8 Remeshing

Remeshing becomes necessary in SNAC when continued deformation distorts a mesh so severely that the accuracy of solutions is deteriorated. In SNAC, remeshing is done by moving the nodes back to their original positions, under the constraint that physically meaningful features such as top and bottom topography are conserved. Then both nodal (velocity and temperature) and elemental variables (stress, strain rate, heat flux, and plastic strain) are transferred onto the new mesh from the deformed mesh. The nodal transfer is accomplished by locating the tetrahedron in the old mesh which contains a node of the new mesh. Then shape (interpolation) functions corresponding to the location of the new node with respect to the old element is calculated and used to interpolate nodal quantities of the old deformed mesh. The element-associated variables (integrated by 1-point Gauss quadrature) are first "recovered" as nodal fields by Superconvergent Patch Recovery method[17]. Like other nodal fields, these recovered fields are interpolated onto a new mesh. Then, element-associated variables are evaluated at the new tetrahedron's barycenter, coinciding with 1-point Gauss quadrature, based on the interpolated recovered fields. Fig. 1.4 demonstrates how remeshing actually works.

The current remeshing algorithm, however, has some limitations.

- The phase field is not accurately remapped. This field is not a physical variable but a set of integers registered at each tetrahedron to represent the element's material type. Although starting from integer values, a phase field for more than one material types ends up including some real numbers after remeshing, due to numerical diffusion during the recovery process. Fractional material ID is not defined both physically and numerically in the current version of SNAC. Although rounding-off is forced for the continuation of calculation, the mass of each phase is not guaranteed to be conserved.

- Domain decomposition, the parallelism implemented in SNAC, often makes it difficult to map variables between meshes when a parallel boundary (virtual boundary between subdomains assigned to more than one processor) of the old mesh has moved too far away from the corresponding one of the new mesh.

- This remeshing works best in the case of lateral extension because the nodes are moved back to their original positions. If a mesh has shrunk laterally and then remeshing moves all the nodes back to their original horizontal positions, it is impossible to remap variables for the regions of a new mesh that do not have overlapping parts in the deformed mesh. In such a case, one would have to *assume* states and properties of material for the unmappable regions.

- Note that Fig. 1.4b shows that remeshing cannot only generate a more regular mesh but also change the bottom boundary of the domain. Although useful in some cases, this "boundary restoring" technique should be used with caution because it also creates unmappable regions and thus requires us to assume the state and properties of the material. SNAC retains the deformed boundary by default.

1.2 SNAC Design

SNAC follows StGermain's architecture (see Section 1.3). Although primarily implemented in C, it is object-oriented, and makes use of extensibility and inheritance. Having StGermain as a framework, SNAC can directly use or extend many abilities of StGermain available in generic forms: XML input, extensible data structures, classified output streams, etc. The three key design ideas relevant to SNAC are the plugins, SNAC Context and Entry Points.

1.2.1 Plugins

SNAC is readily customizable through writing a "plugin," a shared object that can be loaded dynamically at run-time. Together with the ability to manipulate Entry Points, use of plugins makes it simple and local to extend SNAC. For instance, if a mesh for a quadrant of a cylinder is desired, one can simply include

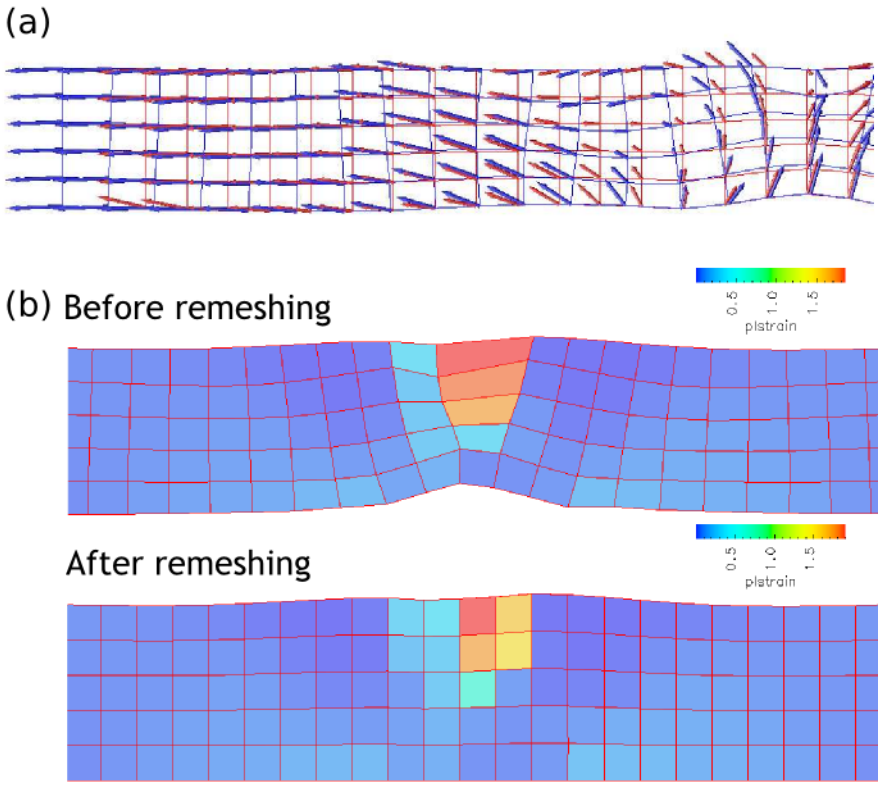


Figure 1.4: (a) Meshes and velocity fields before (blue) and after (red) remeshing. (b) Plastic strain, an element-associated variable, is shown on a deformed mesh (upper panel) and mapped onto a new mesh after remeshing (lower panel).

`SnacCylinderQuad` in the list of plugins in an input file. If a thermal problem needs to be solved in addition to a quasistatic momentum balance problem, loading the plugin, `SnacTemperature`, is all that needs to be done. Trying out different constitutive models can be done in the same fashion: one can load one of the plugins such as `SnacElastic`, `SnacViscoElastic`, or `SnacViscoPlastic`.

1.2.2 The SNAC "Context"

The SNAC Context is the top-level "master" object in a SNAC simulation. Its main purpose is twofold:

1. Providing an interface to control and run a SNAC simulation. This is done mainly through the Entry Points, described in the next section.

2. Packaging together all the high-level data structures that a SNAC run requires, such as the Mesh, boundary conditions, simulation parameters, etc. Most of these objects are extensible and/or configurable using the XML input file.

The SNAC Context inherits from the *StGermain Mesh* and *Abstract Context*, which provide much of the default functionality.

1.2.3 Call-Tree of Entry Points

An entry point (EP) is essentially a list of function pointers, which can be dynamically modified. When an EP is called, it runs each of the function pointers in its list sequentially, with the same arguments. Since the SNAC Context calls the EPs in a specific hierarchy, an application's set of EPs defines the call-tree for an application.

The reason entry points are dynamic lists of function pointers (or "hooks") is that a user, e.g., a research scientist, often wants to extend the default functionality of an application at only a few specific points in the code. In SNAC, this is simply a matter of adding pointers to their custom functions to the relevant entry points. For instance, to add a contribution from the Winkler foundation to the residual force at each node on the bottom surface, one can simply include a plugin called `SnacWinklerForce`. What this plugin does is to add (or "hook up") a pointer to its own force computing routine to the entry point that already has a pointer to the function computing internal residual forces. While being a part of the loop over all the nodes (local to a processor), this plugin's function takes care of deciding if a node is on the bottom and adding the computed restoring force. In this way, the core structure of the code does not need to be modified every time new operations on nodes or elements are added. StGermain provides a flexible way of managing hooked-up function pointers such that one can create a new entry point, set a default function for it, insert one function pointer before or after another within an Entry Point, etc.

Fig. 1.5 outlines the entry points set up in SNAC. It is represented as a tree-like hierarchy: if a "hook" calls other entry points, either in its own function or other normal functions it calls, the called entry points are offset to the right of that hook.

1.3 StGermain

StGermain is a software framework tuned for the development of codes common to the physical sciences, such as geodynamics. It permits developers to build applications that are extensible, adaptable, cleanly designed and interoperable with one other. Moreover, StGermain answers the question of, "Now that I have written my code, how do I share it with my collaborators with the optional inclusion of their additions?" This capability is inherent to StGermain and provides an important feature lacking in most scientific codes. As of the end of 2008, the development of StGermain in conjunction with SNAC has been finished and its interfaces have been frozen.

StGermain is funded primarily through VPAC and Caltech. The project was born with the development of SNAC, and the need to generalize the same abilities that had been incorporated into the development of Snark. It was time to restart from scratch, and was done late at night under a California moon, aided by

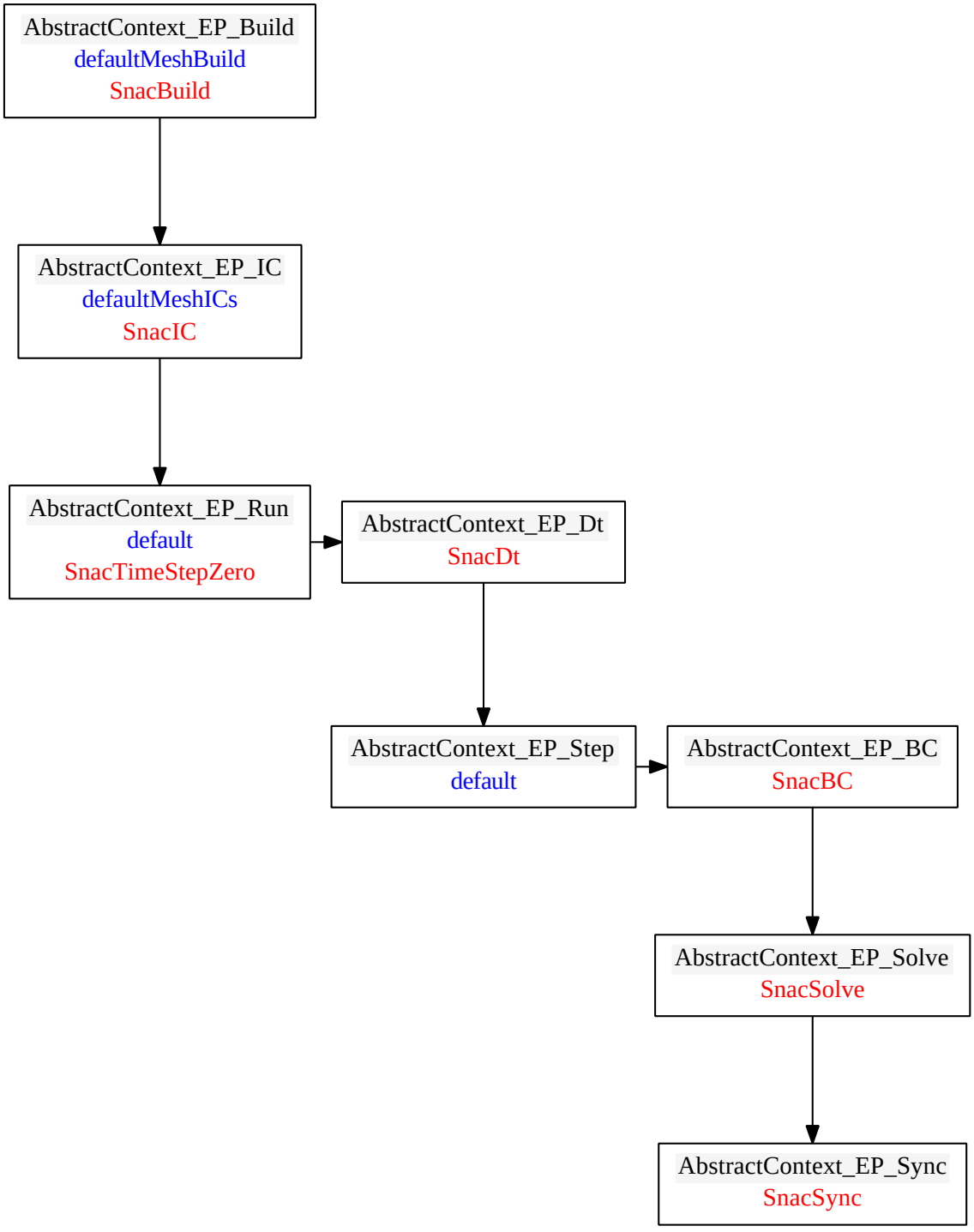


Figure 1.5: A tree-like diagram showing the hierarchy of entry points set up in SNAC together with hooked-up functions in each entry point. The functions defined by StGermain are marked in blue while those defined in SNAC are in red.

some fine Belgian beer and the finest Parisienne electro-jazz. However, it is unclear whether StGermain is named for the French techno-jazz group led by Ludovic Navarre or the famed Paris quartier.

StGermain provides a suite of libraries needed by general physical modeling software such as flow control, domain discretization, and initial/boundary condition management [7]. One of the advantages of using the StGermain framework is that a code can be easily extended through entry points and plugins. By adding or removing entry points, a problem-specific algorithm can be implemented while keeping the modification of source codes limited. Plugins are small pieces of codes that work as a part of the program, but can be compiled separately and dynamically loaded at run time. Thus, their developments can be completely independent of the main program. For example, a new constitutive relation or time-varying boundary conditions can be made available as plugins without modifying or compiling the whole program that already exists.

Chapter 2

Installation and Getting Help

2.1 Introduction

To install SNAC, follow the procedure that is commonly used with other open-source software packages. First, download the source package (in the form of a compressed `tar` file) available at the SNAC web page (`geodynamics.org/cig/software/packages/long/snac`). After unpacking the source, you run a prepackaged shell script to configure SNAC for your system.

2.2 Getting Help

For help, send e-mail to the CIG Long-term Tectonics Mailing List (`cig-long@geodynamics.org`). You can subscribe to the Mailing List and view archived discussion at the Geodynamics Mail Lists web page (`geodynamics.org/cig/lists`).

2.3 System Requirements

Installation of SNAC requires the following:

- A C compiler
- An MPI library
- Libxml2
- GNU Scientific Library

MPI installations are typically configured for a particular compiler, and provide a special wrapper command to invoke the right compiler. Therefore, the choice of MPI implementation often determines which C compiler to use.

2.3.1 C Compiler

On Unix or Linux systems, there is a high likelihood that a usable C compiler is already installed. To check, type `cc` at the shell prompt:

```
$ cc
cc: no input files
$
```

On Linux, if the `cc` command is not found, install GCC using the package manager for your distribution.

The Mac OS X version of GCC is included in a software development suite called Xcode. Xcode is available as a free download at the Apple Developer Connection (`developer.apple.com`).

Warning: If you are using an Intel compiler on an Itanium CPU, do not use the `-O3` optimization flag as reports indicate that this optimization level will generate incorrect codes. For any compiler, you should always be careful about the correctness of the compiled codes when using an `-O3` or higher optimization level.

2.3.2 MPI Library

StGermain requires an implementation of the MPI-1 standard for parallel message passing, even if you only intend to run the code in serial. A popular choice is MPICH. Installing MPICH from source involves walking through the standard GNU build procedure (`configure && make && make install`).

Linux users may have a prebuilt MPI package available for their distribution. On Mac OS X, the Fink package manager offers a prepackaged version of LAM/MPI; so if you have Fink installed, simply enter the following command from a Terminal window to install LAM/MPI:

```
$ fink install lammpi lammpi-dev
```

We highly recommend that you set up the environment variables that allow StGermain to use MPI. This is done by setting the following environment variables so the VMake build system knows where to find MPICH.

In `bash`,

```
export MPI_DIR=${HOME}/opt/mpich-1.2.7p1
export MPI_BINDIR=${MPI_DIR}/bin
export MPI_LIBDIR=${MPI_DIR}/lib
export MPI_INCDIR=${MPI_DIR}/include
export MPI_RUN=${MPI_BINDIR}/mpirun
export PATH=${MPI_BINDIR}:${PATH}
export LD_LIBRATY_PATH=${MPI_LIBDIR}:${LD_LIBRATY_PATH}
```

In `tcsh`,

```
setenv MPI_DIR ${HOME}/opt/mpich-1.2.7p1
setenv MPI_BINDIR ${MPI_DIR}/bin
setenv MPI_LIBDIR ${MPI_DIR}/lib
setenv MPI_INCDIR ${MPI_DIR}/include
setenv MPI_RUN ${MPI_BINDIR}/mpirun
setenv PATH ${MPI_BINDIR}:${PATH}
setenv LD_LIBRATY_PATH ${MPI_LIBDIR}:${LD_LIBRATY_PATH}
```

MPI_DIR should be replaced with the actual path where MPICH is installed.

If you need to specify a machine file every time you run mpich, then in order for VMake's "make check" command to work, you must specify another environment variable, MPI_MACHINES. Here are some examples:

```
In bash: export MPI_MACHINES="/home/raq/machinefile"
In tcsh: setenv MPI_MACHINES -machinefile /home/raq/machinefile
```

2.3.3 Libxml2

The Libxml2 library and toolkit are required for SNAC to handle the input files. Although already included in most of the recent Linux systems, it can be obtained from The XML C parser and toolkit of Gnome and installed manually. Precompiled packages for various Linux distributions and OS X are also available through their own package management software.

2.3.4 GNU Scientific Library

GNU Scientific Library (GSL) is "a numerical library for C and C++ programmers" (http://www.gnu.org/software/gsl). SNAC needs this library for remeshing (see 1.1.8) since the recovery method involves matrix inversion operations. GSL's linear algebra routines are utilized in SNAC's remesher plugin. Although the remesher is a plugin, it is expected to be used in most geodynamic applications. Also, the use of GSL will be expanded in future development. For this reason, GSL has become a requirement.

SNAC's build scripts will be able to find and link GSL if it is installed in a standard location. But if it is installed locally, one should define an environment variable, `GSL_DIR`, as follows:

```
In bash: export GSL_DIR="/home/raq/opt/gsl"
In tcsh: setenv GSL_DIR /home/raq/opt/gsl
```

2.3.5 Other Environment Variables

It is handy, although not necessary, to define the following environment variables for both building and running SNAC:

In `bash`,

```
export SNAC_DIR=${HOME}/opt/SNAC
export SNAC_BINDIR=${SNAC_DIR}/build/bin
export SNAC_INCDIR=${SNAC_DIR}/build/include
export SNAC_LIBDIR=${SNAC_DIR}/build/lib
export PATH=${SNAC_BINDIR}:${PATH}
export LD_LIBRARY_PATH=${SNAC_LIBDIR}:${LD_LIBRARY_PATH}
```

In `tsch`,

```
setenv SNAC_DIR ${HOME}/opt/SNAC
setenv SNAC_BINDIR ${SNAC_DIR}/build/bin
setenv SNAC_INCDIR ${SNAC_DIR}/build/include
setenv SNAC_LIBDIR= ${SNAC_DIR}/build/lib
setenv PATH ${SNAC_BINDIR}:${PATH}
setenv LD_LIBRARY_PATH ${SNAC_LIBDIR}:${LD_LIBRARY_PATH}
```

2.4 Downloading and Unpacking Source

Download SNAC from the SNAC web page. Save the SNAC tarball locally, and unpack it using the `tar` command:

```
$ tar xzf SNAC-1.2.0.tar.gz
```

If you don't have GNU Tar, try the following command instead:

```
$ gunzip -c SNAC-1.2.0.tar.gz | tar xf -
```

2.5 Installation Procedure

After unpacking the source, use the following procedure to install SNAC:

1. Navigate (i.e., `cd`) to the directory containing the SNAC source.

   ```
   $ cd SNAC-1.2.0
   ```

2. Type `./configure.sh` to configure the package for your system.

   ```
   $ ./configure.sh
   ```

3. Type `make` to build the package.

```
$ make
```

All the newly created files during the building procedure are placed in the `build` subdirectory. Upon successful completion, the `make` command executable `Snac` in the `build/bin` subdirectory. Currently, installing SNAC into another location is not supported.

The following options can be included in the options argument during configuration in comma-separated form.

optimised compiles optimized code, with no debugging symbols.

cautious performs additional run-time validation checks.

macro_as_func compiles some macros as functions instead for safety checks. Will incur a performance penalty.

memory_stats enable statistics recording of the memory module.

tau compiles the code ready for use with the tau profiler.

pdt automatically instruments the code for use with tau using.

For example, to include cautious checks and memory statistics recording:

```
$ ./configure.sh --options=cautious,memory_stats
```

Note that the "build" directory under `${SNAC_DIR}` is named according to the given extra options. For example, `${SNAC_DIR}/build-optimised` is created when `optimised` option is given.

2.6 Installing from the Software Repository

The SNAC source code is available via a Subversion server at the Geodynamics website (`geodynamics.org`). This allows users to view the revision history of the code and check out the most recent development version of the software.

NOTE: If you are content with the prepared source package, you may skip this section.

2.6.1 Tools You Will Need

In addition to the usual system requirements, you will need a handful of additional development tools installed in order to work with the source from the CIG software repository.

First, you must have a Subversion client installed. To check, type `svn`; it should return a usage message.

```
$ svn
Type 'svn help' for usage.
```

For more information on Subversion, visit the Subversion website (`subversion.tigris.org`).

2.6.2 Download Source from Subversion

To check out the latest version of the software, use the `svn checkout` command:

```
$ svn checkout http://geodynamics.org/svn/cig/long/3D/SNAC/trunk SNAC
```

This will create the local directory `SNAC` (if it doesn't already exist) and fill it with the latest SNAC source from the CIG software repository.

The `SNAC` directory thus created is called a *working copy*. To merge the latest changes into an existing working copy, use the `svn update` command:

```
$ cd SNAC
$ svn update
```

This will preserve any local changes you have made to your working copy.

Chapter 3

Running SNAC

3.1 Using SNAC

The basic usage is

```
Snac input.xml
```

Running SNAC in parallel depends on the system configuration, but a typical example is

```
mpirun -np 8 `which Snac` ./input.xml > log.stdout
```

3.2 Changing Parameters

Input parameters of SNAC are defined using the XML syntax

```
<param name=parameter> value </param>
```

Many of the parameters defined in this fashion can be conceptually grouped based on their functionality. They can also be a member of a structure, or itemized under a list, which in turn can become a member of a structure.

3.2.1 Simulation Control Parameters

Parameters in this group control the time marching in SNAC.

`startTime`: the starting time of simulation. 0 sec by default.

`stopTime`: the ending time of simulation. 1.0e+20 sec by default.

`outputPath`: the relative or absolute path to a directory where outputs are dumped.

`dumpEvery`: the interval of time steps to dump outputs.

`maxTimeSteps`: the maximum number of time steps. The time marching ends either at "stoptime" or at "maxTimeSteps," whichever comes first.

`checkpointEvery`: the interval of time steps to checkpoint (see 3.4).

`restartTimestep`: the time step to restart (see 3.4).

3.2.2 SNAC-Specific Parameters

Parameters in this group are specific to SNAC.

`density:` the average density of the material SNAC is simulating. Unit is kg/m3.

`gravity:` the gravitational acceleration. 9.8 m/sec2 by default.

`demf:` the dimensionless force damping factor. 0.8 by default.

`dtType:` the type of time marching method. Either "dynamic" or "constant."

`timeStep:` the size of time step in seconds. Used only when dtType is "constant."

`forceCalcType:` the type of assembling forces on the parallel boundary nodes.

`quick:` the least amount of computation. Used only in serial runs (Obsolete).

`normal:` works in both serial and 1D parallel runs (Obsolete).

`complete:` Robust nodal force assembly.

`decomposedAxis:` axis to decompose for parallel runs. 0, 1, or 2 for x, y, and z axis, respectively. Needed only when decompDims=1.

`storeForces:` "yes" to store residual forces for each node. Otherwise, "no."

`forceCheckSum:` If "yes," the sum of forces is checked if it is consistent.

`topo_kappa:` Parameter for topography smoothing.

`alpha:` Volumetric thermal expansion coefficient in the unit of K^{-1}.

3.2.3 Plugins List

All the plugins to load should be listed under the "extensions" or "plugins" list as in the following example:

```
<list name="extensions">
        <param> SnacSpherical </param>
</list>
```

Note the changed syntax used for the item of the list. One can populate the list with other plugins. The following is the list of plugins available for SNAC:

`SnacSpherical:` needed to generate a spherical shell mesh.

`SnacRemesher:` when using remeshing. The related parameters should also be set.

`SnacTemperature:` when solving for heat diffusion equation.

`SnacElastic/SnacPlastic/SnacMaxwell/SnacViscoPlastic:` for elastic, elasto-plastic, Maxwell viscoelastic, and elasto-visco-plastic rheology, respectively.

`SnacHydroStaticIC:` generates an initial stress field assuming hydrostatic equilibrium. Meaningful only when gravity is non-zero.

`SnacWinklerForce:` When gravity is non-zero, and there are no boundary conditions for the vertical velocity component on the bottom surface, the Winkler foundation is applied.

`SnacPlSeeds/SnacVPSeeds:` can control the positions of elements that have the initial non-zero plastic strain. These elements play the role of seeds where strain localization initiates. Used in tandem with SnacPlastic and SnacViscoPlastic, respectively.

`SnacRestart:` required to restart a run. See 3.4.2 for more details.

Other plugins found in the plugins directory are experimental.

3.2.4 Mesh Structure

Parameters needed for generating a mesh are grouped as members of "`mesh`" structure. `meshSize[I,J,K]` specifies the node numbers in `x`, `y`, and `z` axis, respectively, while `[min,max][X,Y,Z]` defines the physical dimensions of the domain in *meter*. In the case of using spherical geometry, `[theta,phi,r][Min,Max]` should have appropriate values so that the `SnacSpherical` plugin can use them to initialize spherical node coordinates. `theta` and `phi` are longitudes and latitudes in *degrees*, and `r[Min,Max]` are the radii in *meters*. SNAC will not be confused even if both sets of parameters are present in one input file. `meshType` is either `cartesian` or `spherical`. If not specified, it is `cartesian` by default. However, it is important to assign a correct value for remeshing. Finally, the two parameters for a parallel run are `shadowDepth` and `decompDims`. `shadowDepth` determines how many elements in one local domain are made to transfer variables to neighbor domains. It takes a non-negative integer value corresponding to the number of layers of elements. To run SNAC in parallel, this parameter should be 1 or larger. `decompDims` determines whether parallel domain decomposition is performed in 1, 2 or 3 dimensions.

3.2.5 Parameters for Material Property

Four groups of parameters define elastic, viscous, plastic, and thermal properties of material. Assigned values should have MKS units. XML syntax is

```
<param name="parameter"> value </param>
```

- `Elastic material parameters`: lambda, mu Lame's constants.
- `Viscous material parameters`: `refvisc, reftemp, activationE, srexponent, vis_min,` and `vis_max`

  The following function form of viscosity is assumed by default:

$$\eta\left(T,\dot{\varepsilon}\right)=\eta_0\dot{\varepsilon}^{(1/n-1)}\exp\left(\frac{E^*}{R}\left(\frac{1}{T}-\frac{1}{T_0}\right)\right), \tag{3.1}$$

  where η_0 is "`refvisc`" (Pa), n is "`srexponent`" (dimensionless integer), E^* is "`activationE`" (J), T_0 is "`reftemp`" ($^\circ C$), , $\dot{\varepsilon}$ is strain rate, and R is the Gas constant. "`vis_min`" and "`vis_max`" are used to set the range of viscosity variation. Other viscosity models can be easily implemented by modifying `Constitutive.c` in the relevant plugins (`SnacMaxwell` and `SnacViscoPlastic`).

- `Plastic material parameters`

  Plastic parameters can dynamically vary and are approximated by piecewise linear function of accumulated plastic strain. So, the number of linear segments is defined first by `nsegments`. If `nsegments` is n, n+1 values of parameters are required to define the n linear segments.

  The working plasticity models in SNAC is Mohr-Coulomb. The Drucker-Prager model has been implemented but not verified. The corresponding parameter name is `yieldcriterion`, and the legitimate values it can assume are `mohrcoulomb` and `druckerprager`.

  Three parameters are necessary to define each model's yield function: friction angle, dilation angle, and cohesion. The parameters should be named as "parameter + [0,...,n]." So the input file should have `frictionAngle0, frictionAngle1,` ...; `dilationAngle0, dilationAngle1,` ...; and `cohession0, cohesion1,`....

  To avoid a singularity of the yield surface at the tensional limit, SNAC defines `tension_off`, a practical limit to tensional stress over which yielding occurs even if the stress state is not on the yield surface. The following is an example of defining plastic material parameters:

```
<param name="nsegments">
    2</param>
<param name="plstrain0">
    0.0</param>
```

```
<param name="plstrain1">
    0.1</param>
<param name="plstrain2">
    1000.0</param>
<param name="frictionAngle0">
    30.0</param>
<param name="frictionAngle1">
    20.0</param>
<param name="frictionAngle2">
    20.0</param>
<param name="dilationAngle0">
    5.0</param>
<param name="dilationAngle1">
    5.0</param>
<param name="dilationAngle2">
    5.0</param>
<param name="cohesion0">
    4.0e+07</param>
<param name="cohesion1">
    1.0e+06</param>
<param name="cohesion2">
    1.0e+06</param>
<param name="ten_off">
    1.0e+13</param>
```

- `Thermal material parameters`

  Values of thermal conductivity and heat capacity can be defined. The parameter names are `thermal_conduct` and `heatCapacity`, and appropriate values in MKS unit should be assigned. The default values are 2.0 W/(m·K) and 1000.0 J/(kg·K), respectively. To be used for initial and boundary conditions, temperature values in $^{\circ}C$ can be assigned to `topTemp` and `bottomTemp`.

- `Remesher parameters`

  Remeshing is triggered when the mesh is distorted too much. The criteria to determine when to remesh are the number of time steps and the minimum length scale. The former requires *a priori* knowledge on the rate of deformation because SNAC's remeshing algorithm would fail if the displacement is larger than one-element size. Using this criterion makes it convenient to force remeshing on a regular basis. The latter criterion allows SNAC to have a dynamically determined frequency of remeshing. The length scale is the global minimum of the ratio of a tetrahedron's volume to one of its surface area. If this minimum length scale decreases below some fraction of the initial value, remeshing is triggered. The parameter name for the type of criterion is `remeshCondition`, of which possible values are `onTimeStep`, `onMinLengthScale`, and `onBothTimeStepLength`. The critical value for time step and minimum length scale should be provided. `remeshLoopCriterion` and `remeshLengthCriterion` are the parameter names, and they take a positive integer and a positive number less than 1 as values, respectively.

```
<param name="remeshCondition">
    onBothLoopLength</param>
<param name="remeshLoopCriterion">
    10</param>
<param name="remeshLengthCriterion">
    0.5</param>
```

3.2.6 Initial Conditions Structure

Two structures need to be constructed for defining the initial conditions. One is for variables associated with nodes, the other for the element-associated ones. A name is assigned to the structure first. This structure will have only a single list as a member. This list in turn can have a series of structures of which members are the type of initial conditions and the list of variables. Finally, the list of variables has structures for each variable to be initialized. The lowest level structure will have name, type and value for the variable. The following is an example of constructing such a structure.

```
<!-- node ICs: initical conditions for nodal variables, i.e., velocity and temperature -->
    <struct name="nodeICs">
        <list name="vcList">
            <struct>
                <param name="type">
                    AllNodesVC</param>
                <list name="variables">
                    <struct>
                        <param name="name">
                            vx</param>
                        <param name="type">
                            double</param>
                        <param name="value">
                            0</param>
                    </struct>
                    <struct>
                        <param name="name">
                            vy</param>
                        <param name="type">
                            double</param>
                        <param name="value">
                            0</param>
                    </struct>
                    <struct>
                        <param name="name">
                            vz</param>
                        <param name="type">
                            double</param>
                        <param name="value">
                            0</param>
                     </struct>
                </list>
            </struct>
            <struct>
                <param name="type">
                     AllNodesVC</param>
                <list name="variables">
                    <struct>
                        <param name="name">
                            temperature</param>
                        <param name="type">
                            double</param>
                        <param name="value">
                            500.0</param> <!-- in degrees Celsius -->
                    </struct>
                </list>
            </struct>
        </list>
    </struct>
```

```
<!-- element ICs: initial conditions for elemental variables. Currently, material type is not used.
 The block for hydrostatic pressure should be accompanied by ‘‘SnacHydroStaticIC’’ module in the above
 module list. -->
    <struct name="elementICs">
        <list name="vcList">
            <struct>
                <param name="type">
                    AllElementsVC</param>
                <list name="variables">
                    <struct>
                        <param name="name">
                            elementMaterial</param>
                        <param name="type">
                            int</param>
                        <param name="value">
                            0</param>
                    </struct>
                </list>
            </struct>
        </list>
    </struct>
```

Note that the type of the data structure used for initial conditions in StGermain are `AllNodesVC` and `AllElementsVC`. These types are used to assign values to all the nodes or elements (see Section 3.2.7 for other types of structure).

One way of assigning a value to all nodes or elements is to put a single value for the "value" member in the lowest-level structure as in the next example:

```
<struct>
    <param name="name">
        temperature</param>
    <param name="type">
        double</param>
    <param name="value">
        500.0</param>
</struct>
```

The other way is to define a *condition function* and assign "func" to type and the function's name to value. More details are given in the section 3.3.

3.2.7 Boundary Conditions Structure

The syntax to define a structure for boundary conditions is almost the same with the initial conditions. As in the case of initial conditions, a user can reuse most parts of the listing given below. For example, the name of the highest-level structure, `velocityBCs`, need not be changed for every different problem. Thus, only a few notable differences are addressed here.

The value for the type of the second-level structure is `wallVC`. This structure type requires specifying which "wall" to apply boundary conditions to. For SNAC, one of left, right, top, bottom, back, or front can be assigned to a parameter named `wall`. Those six values represent surfaces of which nodes have minimum x index, maximum x index, minimum y index, maximum y index, minimum z index, and maximum z index, respectively. By including the following listing in an input XML file, the x component of velocity of all the nodes belonging to the model's left wall will have the value "-5.0e-08 (m/s)" as a double precision number:

```
<struct name="velocityBCs">
    <list name="vcList">
        <struct>
            <param name="type">
                WallVC</param>
```

```
            <param name="wall">
                left</param>
            <list name="variables">
                <struct>
                    <param name="name">
                        vx</param>
                    <param name="type">
                        double</param>
                    <param name="value">
                        -5.0e-08</param>
                </struct>
             </list>
          </struct>
    </list>
</struct>
```

It is straightforward to apply boundary conditions for other components of velocity on the same wall. One can simply put something like

```
<struct>
    <param name="name">
        vy</param>
    <param name="type">
        double</param>
    <param name="value">
        -1.0e-08</param>
</struct>
```

between `<list name="variables">` and `</list>`. However, to work on other walls than "left," one needs to replicate a higher-level structure. The other way of applying boundary conditions is to put "`func`" in the "`type`" member and the function's name in the "`value`" member as follows:

```
<struct>
    <param name="name">
        vx</param>
    <param name="type">
        func</param>
    <param name="value">
        variableVx</param>
</struct>
```

This kind of function is called a *condition function* and should be defined in one of the loaded plugins. One can find an example in `TestCondition.c` in the `SnacSpherical` plugin's source code directory. Using this function makes it possible to assign values varying according to the spatial positions, or the indexes of each element. For more details, see Sec. 3.3.

In most situations, a user can keep the overall structure as it is in the above example and would have only to modify the lowest level structure.

The temperature boundary conditions take the same structure. The following example shows how to apply a uniform temperature of $500.0°C$ to all the top surface nodes:

```
<!-- Temperature BCs -->
    <struct name="temperatureBCs">
        <list name="vcList">
            <struct>
                <param name="type">
                    WallVC</param>
                <param name="wall">
                    top</param>
                <list name="variables">
```

```
                    <struct>
                        <param name="name">
                            temperature</param>
                        <param name="type">
                            double</param>
                        <param name="value">
                            500.0</param>
                    </struct>
                </list>
            </struct>
        </list>
    </struct>
```

Again, one can replace the double type of the lowest-level structure with "func" to use a condition function. In that case, the function's name should be supplied as the structure's value.

3.3 Using Condition Functions

Condition functions (CFs) can be used to define both spatially and temporally non-uniform initial and boundary conditions. Two requirements should be met for a condition function to make effects: 1) a CF must be found at run-time and 2) an input file should specify where to apply it. The first condition is satisfied by properly adding a function in the SnacCondFunc plugin and the second by setting the `type` of variables to be `func` (see 3.2.6 and 3.2.7).

3.3.1 Adding a new condition function

Everything is done in `Snac/plugins/conditionFunctions`. The example below shows the step-by-step procedure for adding a new element CF. A new node CF can be added in the same way but by modifying counterparts for node.

1. Go to `Snac/plugins/conditionFunctions.`
2. Open `SnacCondFunc_ElementCondFunc.h`.
   (a) Copy and paste one of the existing function declarations and rename it (e.g., `void _SnacCondFunc_DeadSea` →`void _SnacCondFunc_MyTest`).
3. Open `SnacCondFunc_ElementCondFunc.c`
   (a) Copy and paste one of the existing function definitions and rename it (e.g., `void _SnacCondFunc_DeadSea(...)` →`void _SnacCondFunc_MyTest(...)`).
   (b) Change the contents of the new function (e.g., remove all the `if`-statement blocks except the first one).
4. Open `Register.c`.
   (a) Copy and paste one of the existing function calls to `ConditionFunction_Register_Add`.
   (b) Note that the third argument of `ConditionFunction_Register_Add` is also a function call to `ConditionFunction_New`, which takes two arguments.
   (c) Change both arguments of `ConditionFunction_New` : `ConditionFunction_new(_SnacCondFunc_DeadSea, ''SnacCF_DeadSea'')` →`ConditionFunction_new(_SnacCondFunc_MyTest, ''SnacCF_MyTest'')`. The string, `''SnacCF_MyTest'',` will be used in an input file.
5. Rebuild the plugin: i.e., run `make`.

3.3.2 Using a newly defined condition function

Continuing from the above example, we have only one place to use an element condition function, which is `elementICs`.

```
<struct name="elementICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllElementsVC</param>
            <list name="variables">
                <struct>
                    <param name="name"> elementMaterial</param>
                    <param name="type"> int </param>
                    <param name="value"> 0</param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
```

In the above code, the `type` parameter for the "`variables`" list should be changed from `int` to `func` and the `value` parameter should be the string used when the condition function was registered. It is ''`SnacCF_MyTest`'' in the current example. The modified code would look like

```
<struct name="elementICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllElementsVC</param>
            <list name="variables">
                <struct>
                    <param name="name"> elementMaterial</param>
                    <param name="type"> func </param>
                    <param name="value"> SnacCF_MyTest </param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
```

Finally, load the "SnacCondFunc" plugin as follows:

```
<list name="extensions">
    ⋮
    <param> SnacCondFunc </param>
</list>
```

Run SNAC and check if the condition function works as intended by visualizing the "`Phase`" variable.

3.4 Checkpointing and Restarting

3.4.1 Checkpointing

To restart an already finished run from a certain point in time, it is necessary to record all the relevant information. Generating such data set is called *checkpointing*. It is possible to checkpoint as a response to an emergency like receiving a certain type of error or disk quota and wall clock limit soon to be exceeded. However, SNAC currently only allows a user to specify the frequency of checkpointing by assigning a non-zero value to "`checkpointEvery`" in an input file. The meaning of the assigned value is the same with "`dumpEvery`": *the number of time steps between events of checkpointing*. The default value of this parameter

is 0, and SNAC checkpoints only at the last time step. Since the size of checkpoint data files is much larger than that of the regular outputs, it is advised to set `checkpointEvery` to be a sufficiently large value.

Just as dumping is logged in `timeStep.0`, checkpointing is logged in `checkpointTimeStep.0`. This file is used when restart files are generated.

Checkpoint data files have `CP` attached to their names (e.g., `velocityCP.0`, `stressTensorCP.2`, etc.). The major difference between the checkpointed (tetrahedral) outputs and the regular (hexahedral) outputs is the storing format of element-associated variables such as stress and plastic strain. To restart with exactly the same state as before, those values need to be recorded at the tetrahedra level. That is because the regular outputs contain one single-precision floating point number per hexahedral element.

3.4.2 Restarting

1. Run `snac2restart` in the original `outputPath` (see the next section for more details).
2. Create a new directory for the restarting run. This is mandatory to prevent accidental overwriting of the existing data files.
3. Move all the `*.restart` files to the new directory.
4. Prepare an input file that is identical with the one used for the original run except that
   (a) the new one has "`SnacRestart`" in the extensions (=plugins) list *AND*
   (b) "`restartTimestep`" is set to be the time step to restart from *AND*
   (c) "`outputPath`" now points to where `*.restart` files are. SNAC will abort if it detects existing outputs there.
5. Start running SNAC as usual but with a new input file prepared as above.

3.4.3 `snac2restart`

The general usage of `snac2restart` is

```
Usage: snac2restart [timeStep] [your "outputPath"] [path to write restart files]
```

`timeStep` is one of the checkpointed time steps and should be one of those recorded in `checkpointTimeStep.0`. The last two specify a path to where those output files are which need processing, and where to write the processed restart files, respectively. All the arguments are optional and, by default, `snac2restart` tries to read and write in the current directory and process for the last time step recorded. Files with the suffix of *.restart are created.

Chapter 4

Postprocessing and Graphics

4.1 Introduction

SNAC writes binary output files which must be converted to a file format that external visualization tools can handle. Two such conversion programs are provided. One generates Visualization Tool Kit (VTK) files. The other is for the open-source Open Visualization Data Explorer, better known as OpenDX. There are several programs that can visualize VTK files: ParaView (`www.paraview.org`), MayaVi2 (`https://svn.enthought.com/enthought/wiki/MayaVi`), or VisIt (`https://wci.llnl.gov/codes/visit/`). OpenDX with documentation is available from the OpenDX website (`www.opendx.org`).

4.1.1 Outputs from SNAC

Once you have run SNAC, you should have a series of files in the `output` directory as set in the input file.

Information on the parallel decompositon is not explicitly given in an input file by a user. Instead, StGermain computes an optimal configuration based on the total number of processors, the global mesh size, and the value of `decompDims`. The resultant global and local element numbers in x, y, and z as well as decomposed processor numbers are written in the file `sim.0`, which is common to all the processors.

4.2 Converting to VTK files

A program called `snac2vtk` is provided to convert the binary outputs from SNAC to ascii files in the XML VTK Structured Grid format (.vts). `snac2vtk` is compiled during the building procedure and installed in `${SNAC_BINDIR}`. The usage is

```
snac2vtk path-to-outputs [time1 time2].
```

The only required argument is a path to output files including `sim.0`, which contains critical information to process data. The last two optional arguments are used to set the range of time steps for data conversion.

`snac2vtk` automatically generates Parallel VTK Structured Grid files (.pvts) for each time step so that there is no need for an extra step to combine data, even for the parallel cases. It is the group of .pvts files that needs to be loaded in ParaView or another visualization tool for VTK. However, .vts files should not be removed because .pvts files only refer to .vts files rather than contain actual data.

`> snac2vtk ./` (the last time step only)

or

`> snac2vtk ./ 10` (from 10th to the last time step)

or

`> snac2vtk ./ 1 1001` (from 1 to 1001th time step)

Example list of SNAC output files in the VTK format: 8 processors and 51 outputs per processor.

```
snac.0.000001.vts snac.0.000003.vts snac.0.000005.vts .... snac.0.000101.vts
snac.1.000001.vts snac.1.000003.vts snac.1.000005.vts .... snac.1.000101.vts
snac.2.000001.vts snac.2.000003.vts snac.2.000005.vts .... snac.2.000101.vts
snac.7.000001.vts snac.7.000003.vts snac.7.000005.vts .... snac.7.000101.vts
snac.000001.pvts snac.000003.pvts snac.000005.pvts .... snac.000101.pvts
```

4.3 Converting to OpenDX files

We recommend that you use VTK files for visualizing model results. The conversion tools for the DX file format will no longer be updated.

To convert binary files into ascii OpenDX files, you need to run the program `snac2dx`. The source code is `Snac/snac2dx/snac2dx.c`, compiled during SNAC's building procedure and placed in `${SNAC_BINDIR}` together with other executables.

Running `snac2dx` without any arguments converts data for all the time steps recorded in `timeStep.0`. Optionally, a range of time steps can be set if two positive integers are given as arguments. The converted output files are always written in the directory where `snac2dx` is run. The naming convention for the ascii dx files is `snac.{processor ID}.{time step in 6 digits}.dx`.

To visualize outputs from a parallel run, one more step is needed to combine the .dx files into a single dx file for each time step. A Python script, `snac_combine.py`, is provided for this purpose. This script is under `Snac/snac2dx`, but not automatically installed in `${SNAC_BINDIR}`. The following eight arguments are required:

- **modelname**: Suffix for the combined data files. "`snac`" unless snac2dx.c is modified.
- **timestep**: Time step of interest.
- **gnodex gnodey gnodez**: Global node numbers in x, y, and z (lon, radius, lat) as set in the input file.
- **nprocx nprocy nprocz**: Number of processors in x, y, and z (lon, radius, lat) calculated based on `sim.0`.

Example list of SNAC output files in the OpenDX format: 8 processors and 51 outputs per processor.

```
snac.0.000001.dx snac.0.000003.dx snac.0.000005.dx .... snac.0.000101.dx
snac.1.000001.dx snac.1.000003.dx snac.1.000005.dx .... snac.1.000101.dx
snac.2.000001.dx snac.2.000003.dx snac.2.000005.dx .... snac.2.000101.dx
snac.7.000001.dx snac.7.000003.dx snac.7.000005.dx .... snac.7.000101.dx
```

Let's assume that the global mesh has $51 \times 31 \times 41$ nodes and was decomposed in 3D (`decompDims`=3) by 8 processors. Also, we assume that the contents of `sim.0` is "25 15 20". We can then infer that 2 processors were assigned in each dimension.

To get a combined dx file for time step 35:

```
> snac_combine.py snac 35 51 31 41 2 2 2
```

The name of a combined file has the format of "`snac.{time step in 6 digits}.dx`". So "`snac.000035.dx`" should be the final product in the above example.

4.3.1 Using OpenDX

An OpenDX visual program, `snac_visualize.net`, is provided for convenience and is placed in `Snac/snac2dx`. It can be opened in OpenDX to read data, map node- and element-based variables on the domain, visualize vector quantities as 3D glyphs, create isosurfaces, and make cross-sections.

Chapter 5

Cookbook Examples

5.1 Cookbook 1: Rifting of elasto-visco-plastic lithosphere

A 3D Cartesian block of elaso-visco-plastic material is extended by the velocities applied to two side walls. The domain has a size of $40 \times 10 \times 80$ km and is discretized into $20 \times 5 \times 40$ elements (Fig. 5.1a).

The two side walls perpendicular to the x-axis are pulled at a constant velocity of 1 cm/yr. The other two side walls have free-slip conditions. The bottom surface is supported by the Winkler restoring forces.

An initial temperature field is generated by a condition function that makes a horizontally uniform temperature field with a constant vertical gradient. The boundary conditions fix the top and bottom temperature at their initial values (0 and $700°C$, respectively) to keep the steady state.

The Mohr-Coulomb model is used with a two-step strain weakening. Cohesion decreases from 40 MPa to 0.4 MPa as the second invariant of plastic strain grows up to 2% but remains constant at 0.4 MPa thereafter. Note that extremely large values of vis_min/max are chosen such that viscosity is uniformly high and thus no viscous flow is allowed.

A finite value of plastic strain (2% in this problem) is assigned to a set of elements. These element have a lower cohesion according to the given strain weakening rule and thus strain localization is initiated from them. The selection of elements is randomly made within the region at a certain distance from the left and right boundaries (Fig. 5.1a).

5.1.1 Results

Continued extension builds up strain, and the seed elements start to yield first. Initiated at the seed elements, localization of plastic strain propagates in the direction normal to the applied velocities. Eventually, a long basin bounded by conjugate normal faults is created (Fig. 5.1b).

5.1.2 Complete Listing of the Input XML File

The problem described above can be reproduced using the input file `Snac/examples/Cookbook1/cookbook1.xml`. The full contents of this input file is listed below for reference.

```
<?xml version="1.0"?>
<!DOCTYPE StGermainData SYSTEM "stgermain.dtd">
<!-- StGermain-Snac input file -->
<StGermainData xmlns="http://www.vpac.org/StGermain/XML_IO_Handler/Jun2003">
<!-- StGermain simulation parameters -->
<param name="start"> 0 </param>
<param name="outputPath">./data</param>
<param name="dumpEvery"> 1000 </param>
<param name="maxTimeSteps"> 100000 </param>
<!-- Snac variables -->
<param name="density"> 2700 </param>
```

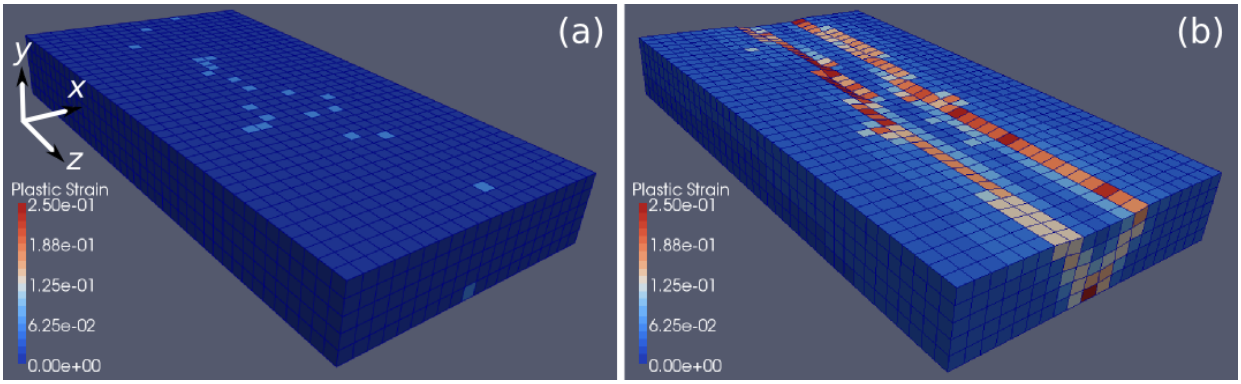


Figure 5.1: (a) The mesh for the example problem with the locations of the "seed" elements. (b) Plastic strain mapped on the deformed mesh after 2 km extension (0.1 My elapsed).

```
<param name="gravity"> 9.8 </param>
<param name="demf"> 0.8 </param>
<param name="dtType"> constant </param>
<param name="timeStep"> 6.3072e+07 </param>
<param name="forceCalcType"> complete </param>
<param name="decomposedAxis"> 0 </param>
<param name="storeForces"> no </param>
<param name="forceCheckSum"> no </param>
<param name="topo_kappa"> 0.0 </param>
<param name="alpha"> 0 </param>
<!-- Extension modules -->
<list name="extensions">
    <param> SnacTemperature </param>
    <param> SnacViscoPlastic </param>
    <param> SnacHydroStaticIC </param>
    <param> SnacVPSeeds </param>
    <param> SnacWinklerForce </param>
</list>
<struct name="mesh">
    <param name="shadowDepth"> 1 </param>
    <param name="decompDims"> 1 </param>
    <!-- Mesh size -->
    <param name="meshSizeI"> 21 </param>
    <param name="meshSizeJ"> 6 </param>
    <param name="meshSizeK"> 41 </param>
    <!-- Initial geometry -->
    <param name="minX"> 0 </param>
    <param name="minY"> 0 </param>
    <param name="minZ"> 0 </param>
    <param name="maxX">  40000 </param>
    <param name="maxY">  10000 </param>
    <param name="maxZ">  80000 </param>
    <!-- Remeshing -->
    <param name="meshType"> cartesian </param>
    <param name="buildNodeNeighbourTbl"> True </param>
</struct>
<!-- Elastic material parameters -->
<param name="lambda"> 1.0e+10 </param>
<param name="mu"> 1.0e+10 </param>
<!-- Viscous material parameters -->
<param name="refvisc"> 1.0e+20 </param>
```

```
<param name="reftemp"> 1400.0 </param>
<param name="activationE"> 45.0e+03 </param>
<param name="vis_min"> 1.0e+30 </param>
<param name="vis_max"> 1.0e+30 </param>
<param name="srexponent"> 1 </param>
<!-- Plastic material parameters -->
<param name="yieldcriterion"> mohrcoulomb </param>
<param name="nsegments"> 2 </param>
<param name="plstrain0"> 0.0 </param>
<param name="plstrain1"> 0.02 </param>
<param name="plstrain2"> 1000.0 </param>
<param name="frictionAngle0"> 30.0 </param>
<param name="frictionAngle1"> 30.0 </param>
<param name="frictionAngle2"> 30.0 </param>
<param name="dilationAngle0"> 5.0 </param>
<param name="dilationAngle1"> 5.0 </param>
<param name="dilationAngle2"> 5.0 </param>
<param name="cohesion0"> 4.0e+07 </param>
<param name="cohesion1"> 4.0e+05 </param>
<param name="cohesion2"> 0.0e+00 </param>
<param name="ten_off"> 1.0e+13 </param>
<!-- Temperature variables -->
<param name="topTemp"> 0.0 </param>
<param name="bottomTemp"> 700.0 </param>
<param name="thermal_conduct"> 1.6 </param>
<param name="heatCapacity"> 1000.0 </param>
<!-- Remesher info -->
<!-- Used only when the SnacRemesher plugin is loaded. -->
<!-- <param name="remeshCondition"> onBothTimeStepLength </param>
     <param name="remeshCondition"> onTimeStep </param>
     <param name="remeshCondition"> onMinLengthScale </param> -->
<param name="remeshCondition"> onMinLengthScale </param>
<param name="remeshTimeStepCriterion"> 15000 </param>
<param name="remeshLengthCriterion"> 0.7 </param>
<param name="bottomResotre"> on </param>
<!-- node ICs -->
<struct name="nodeICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllNodesVC </param>
            <list name="variables">
                <struct>
                    <param name="name"> vx </param>
                    <param name="type"> double </param>
                    <param name="value"> 0 </param>
                </struct>
                <struct>
                    <param name="name"> vy </param>
                    <param name="type"> double </param>
                    <param name="value"> 0 </param>
                </struct>
                <struct>
                    <param name="name"> vz </param>
                    <param name="type"> double </param>
                    <param name="value"> 0 </param>
                </struct>
            </list>
        </struct>
        <struct>
```

```
            <param name="type"> AllNodesVC </param>
            <list name="variables">
                <struct>
                    <param name="name"> temperature </param>
                    <param name="type"> func </param>
                    <param name="value">SnacTemperature_Top2BottomSweep</param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
<!-- element ICs -->
<struct name="elementICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllElementsVC </param>
            <list name="variables">
                <struct>
                    <param name="name">elementMaterial</param>
                    <param name="type"> int </param>
                    <param name="value"> 0 </param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
<!-- Velocity BCs -->
<struct name="velocityBCs">
    <list name="vcList">
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> left </param>
            <list name="variables">
                <struct>
                    <param name="name"> vx </param>
                    <param name="type"> double </param>
                    <param name="value"> -3.17e-10 </param>
                </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> right </param>
            <list name="variables">
                <struct>
                    <param name="name"> vx </param>
                    <param name="type"> double </param>
                    <param name="value"> 3.17e-10 </param>
                </struct>
            </list>
        </struct>
        <struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> back </param>
            <list name="variables">
                <struct>
                    <param name="name"> vz </param>
                    <param name="type"> double </param>
```

```
                    <param name="value"> 0.0 </param>
                </struct>
            </list>
        </struct>
        <struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> front </param>
            <list name="variables">
                <struct>
                    <param name="name"> vz </param>
                    <param name="type"> double </param>
                    <param name="value"> 0.0 </param>
                </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> bottom </param>
            <list name="variables">
                <struct>
                    <param name="name"> vy </param>
                    <param name="type"> double </param>
                    <param name="value"> 0.0 </param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
<!-- Temperature BCs -->
<struct name="temperatureBCs">
    <list name="vcList">
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> top </param>
            <list name="variables">
                <struct>
                    <param name="name"> temperature </param>
                    <param name="type"> double </param>
                    <param name="value"> 0.0 </param>
                </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> bottom </param>
            <list name="variables">
                <struct>
                    <param name="name"> temperature </param>
                    <param name="type"> double </param>
                    <param name="value"> 700.0 </param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
</StGermainData>
```

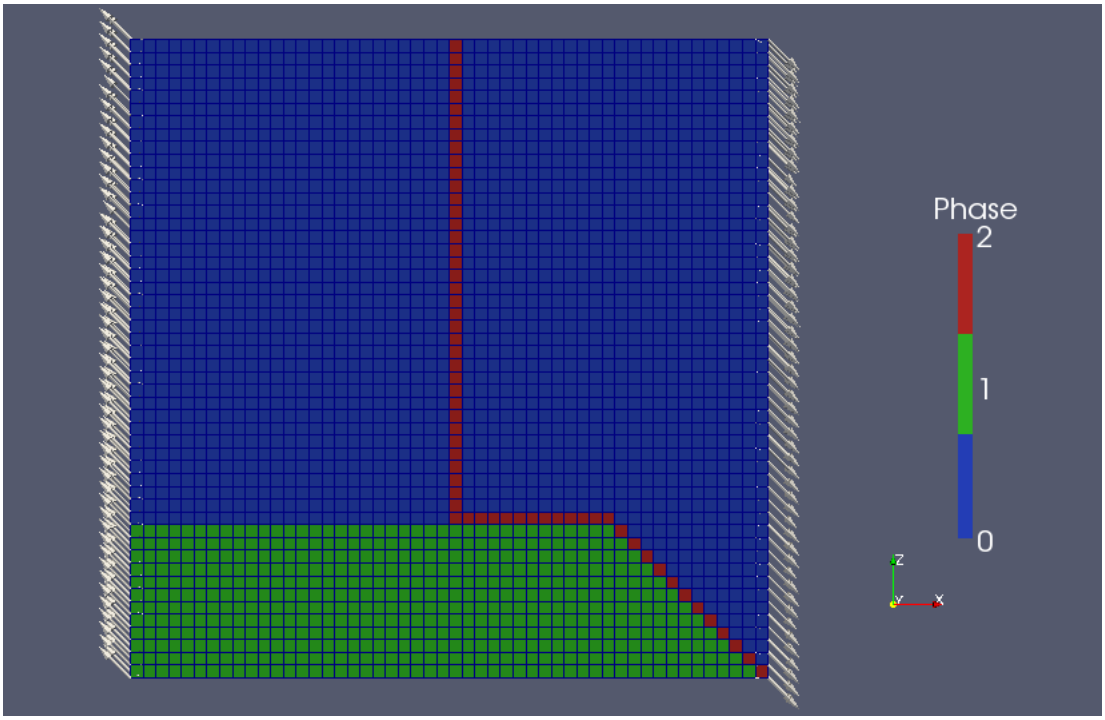


Figure 5.2: Phase index map of Cookbook 2 generated by the condition function `SnacCF_DeadSea`. Arrows indicate the velocity boundary conditions at the initial time step.

5.2 Cookbook 2: Condition functions to assign multiple material types

This example shows how to introduce multiple material types and assign them spatially in a non-trivial fashion. The domain is 300×6×300 km and discretized into 50×1×50 elements. Three types of materials are all elasto-plastic and have the same material properties except cohesions. Each type has the initial value of 10, 100, and 1 MPa, respectively. We use a condition function called `SnacCF_DeadSea` to assign these types to three different subdomains. The boundary conditions set up the plane strain condition in the y direction while an oblique (relative to the z-axis) spreading is applied on the left and right sides. The map of phase index is shown in Fig. 5.2.

5.2.1 Complete listing of the input XML file

```
<?xml version="1.0"?>
<!DOCTYPE StGermainData SYSTEM "stgermain.dtd">
<!-- StGermain-Snac input file -->
<StGermainData xmlns="http://www.vpac.org/StGermain/XML_IO_Handler/Jun2003">
<!-- StGermain simulation parameters -->
<param name="start"> 0 </param>
<param name="outputPath">./data</param>
<param name="dumpEvery"> 10 </param>
<param name="checkpointEvery"> 0 </param>
<param name="maxTimeSteps"> 2 </param>
<param name="restartTimestep"> 0 </param>
<!-- Snac variables -->
<param name="gravity"> 0.0 </param>
<param name="demf"> 0.8 </param>
```

```
<param name="dtType"> constant </param>
<param name="timeStep"> 3.1536e+08 </param>
<param name="forceCalcType"> complete </param>
<param name="storeForces"> no </param>
<param name="forceCheckSum"> no </param>
<param name="topo_kappa"> 0.0 </param>
<!-- Extension modules -->
<!--
    <param> SnacWinklerG3Force </param>
    <param> SnacRemesher </param>
    <param> SnacHydroStaticIC </param>
    <param> SnacCustomCartesian </param>
    <param> SnacViscoPlastic </param>
    <param> SnacVPSeeds </param>
    <param> SnacPlSeeds </param>
-->
<list name="extensions">
    <param> SnacPlastic </param>
    <param> SnacCondFunc </param>
</list>
<struct name="mesh">
    <param name="shadowDepth"> 1 </param>
    <param name="decompDims"> 2 </param>
    <!-- Mesh size -->
    <param name="meshSizeI"> 51 </param>
    <param name="meshSizeJ"> 2 </param>
    <param name="meshSizeK"> 51 </param>
    <!-- Initial geometry -->
    <param name="minX"> 0 </param>
    <param name="minY"> -6000.0 </param>
    <param name="minZ"> 0 </param>
    <param name="maxX"> 300000.0 </param>
    <param name="maxY"> 0.0 </param>
    <param name="maxZ"> 300000.0 </param>
    <!-- Remeshing -->
    <param name="meshType"> cartesian </param>
    <param name="buildNodeNeighbourTbl"> True </param>
</struct>
<list name="materials">  <!-- Three types of material with different cohesions-->
<struct name="mat_normal">
    <param name="density"> 2800 </param>
    <param name="alpha"> 0.0 </param>
    <param name="beta"> 0.0 </param>
    <!-- Elastic material parameters -->
    <param name="lambda"> 3.0e+10 </param>
    <param name="mu"> 3.0e+10 </param>
    <!-- Plastic material parameters -->
    <param name="yieldcriterion"> mohrcoulomb </param>
    <param name="nsegments"> 2 </param>
    <param name="plstrain0"> 0.0 </param>
    <param name="plstrain1"> 0.01 </param>
    <param name="plstrain2"> 1000.0 </param>
    <param name="frictionAngle0"> 0.0 </param>
    <param name="frictionAngle1"> 0.0 </param>
```

```
        <param name="frictionAngle2"> 0.0 </param>
        <param name="dilationAngle0"> 0.0 </param>
        <param name="dilationAngle1"> 0.0 </param>
        <param name="dilationAngle2"> 0.0 </param>
        <param name="cohesion0"> 2.0e+07 </param>
        <param name="cohesion1"> 1.0e+07 </param>
        <param name="cohesion2"> 1.0e+07 </param>
        <param name="ten_off"> 1.0e+12 </param>
    </struct>
    <struct name="mat_strong">
        <param name="density"> 2800 </param>
        <param name="alpha"> 0.0 </param>
        <param name="beta"> 0.0 </param>
        <!-- Elastic material parameters -->
        <param name="lambda"> 3.0e+10 </param>
        <param name="mu"> 3.0e+10 </param>
        <!-- Plastic material parameters -->
        <param name="yieldcriterion"> mohrcoulomb </param>
        <param name="nsegments"> 2 </param>
        <param name="plstrain0"> 0.0 </param>
        <param name="plstrain1"> 0.01 </param>
        <param name="plstrain2"> 1000.0 </param>
        <param name="frictionAngle0"> 0.0 </param>
        <param name="frictionAngle1"> 0.0 </param>
        <param name="frictionAngle2"> 0.0 </param>
        <param name="dilationAngle0"> 0.0 </param>
        <param name="dilationAngle1"> 0.0 </param>
        <param name="dilationAngle2"> 0.0 </param>
        <param name="cohesion0"> 2.0e+08 </param>
        <param name="cohesion1"> 1.0e+08 </param>
        <param name="cohesion2"> 1.0e+08 </param>
        <param name="ten_off"> 1.0e+12 </param>
    </struct>
    <struct name="mat_weak">
        <param name="density"> 2800 </param>
        <param name="alpha"> 0.0 </param>
        <param name="beta"> 0.0 </param>
        <!-- Elastic material parameters -->
        <param name="lambda"> 3.0e+10 </param>
        <param name="mu"> 3.0e+10 </param>
        <!-- Plastic material parameters -->
        <param name="yieldcriterion"> mohrcoulomb </param>
        <param name="nsegments"> 2 </param>
        <param name="plstrain0"> 0.0 </param>
        <param name="plstrain1"> 0.01 </param>
        <param name="plstrain2"> 1000.0 </param>
        <param name="frictionAngle0"> 0.0 </param>
        <param name="frictionAngle1"> 0.0 </param>
        <param name="frictionAngle2"> 0.0 </param>
        <param name="dilationAngle0"> 0.0 </param>
        <param name="dilationAngle1"> 0.0 </param>
        <param name="dilationAngle2"> 0.0 </param>
        <param name="cohesion0"> 2.0e+06 </param>
        <param name="cohesion1"> 1.0e+06 </param>
```

```
    <param name="cohesion2"> 1.0e+06 </param>
    <param name="ten_off"> 1.0e+12 </param>
</struct>
</list>
<!-- Remesher info -->
<!--
    <param name="remeshCondition"> onBothTimeStepLength </param>
    <param name="remeshCondition"> onTimeStep </param>
    <param name="remeshCondition"> onMinLengthScale </param>
-->
<param name="remeshCondition"> onMinLengthScale </param>
<param name="remeshTimeStepCriterion"> 15000 </param>
<param name="remeshLengthCriterion"> 45.0 </param>
<param name="bottomResotre"> on </param>
<!-- node ICs -->
<struct name="nodeICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllNodesVC </param>
            <list name="variables">
            <struct>
                <param name="name"> vx </param>
                <param name="type"> double </param>
                <param name="value"> 0 </param>
            </struct>
            <struct>
                <param name="name"> vy </param>
                <param name="type"> double </param>
                <param name="value"> 0 </param>
            </struct>
            <struct>
                <param name="name"> vz </param>
                <param name="type"> double </param>
                <param name="value"> 0 </param>
            </struct>
            </list>
        </struct>
    </list>
</struct>
<!-- element ICs -->
<struct name="elementICs">
    <list name="vcList">
        <struct>
            <param name="type"> AllElementsVC </param>
            <list name="variables">
                <struct>
                    <param name="name"> elementMaterial </param>
                    <param name="type"> func </param>
                    <param name="value"> SnacCF_DeadSea </param>
                </struct>
            </list>
        </struct>
    </list>
</struct>
```

```
<!-- Velocity BCs -->
<struct name="velocityBCs">
    <list name="vcList">
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> left </param>
            <list name="variables">
            <struct>
                <param name="name"> vx </param>
                <param name="type"> double </param>
                <param name="value"> -1.5e-10 </param>
            </struct>
            <struct>
                <param name="name"> vz </param>
                <param name="type"> double </param>
                <param name="value"> 1.5e-10 </param>
            </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> right </param>
            <list name="variables">
            <struct>
                <param name="name"> vx </param>
                <param name="type"> double </param>
                <param name="value"> 1.5e-10 </param>
            </struct>
            <struct>
                <param name="name"> vz </param>
                <param name="type"> double </param>
                <param name="value"> -1.5e-10 </param>
            </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> bottom </param>
            <list name="variables">
            <struct>
                <param name="name"> vy </param>
                <param name="type"> double </param>
                <param name="value"> 0 </param>
            </struct>
            </list>
        </struct>
        <struct>
            <param name="type"> WallVC </param>
            <param name="wall"> top </param>
            <list name="variables">
            <struct>
                <param name="name"> vy </param>
                <param name="type"> double </param>
                <param name="value"> 0 </param>
```

```
            </struct>
            </list>
        </struct>
    </list>
</struct>
</StGermainData>
```

Chapter 6

Parallel Performance

6.1 Method

We solve the problem of a 3D elastic bar with a given initial geometry deforming by its own weight to acquire SNAC's hard scaling data (i.e., the size of problem is fixed while the number of processors is varied). All the boundaries except the top surface is fixed such that the top surface is warped downward due to gravity. The file I/O has been reduced to a minimal level. The 32^3-element case problem was run until the 1000th step. To save time, however, the 64^3-element and 128^3-element cases were run until 200th and 50fth step, respectively.

These tests were performed on `ranger` at Texas Advanced Computing Center of University of Texas, Austin, by Dr. Colin Stark at Lamont Doherty Earth Observatory of Columbia University.

6.2 Results

The total wall clock times taken for each case is shown in the left panel of Fig. 6.1 as a function of the number of cores. The plots of speedup and efficiency (the middle and right panels of Fig. 6.1) show the acceptable performance of SNAC up to 4096 cores. The performance of the 32^3-element case becomes notably poor for 2048 cores or more. This is because the size of subdomain becomes too small to benefit from parallel processing. These results suggest that the parallel overhead will dominate the computation time if the subdomains become too small and the empirical lower limit for the subdomain size is 3^3 elements. Also, a good scaling is expected to hold for even larger numbers of cores if the problem size is sufficiently large.

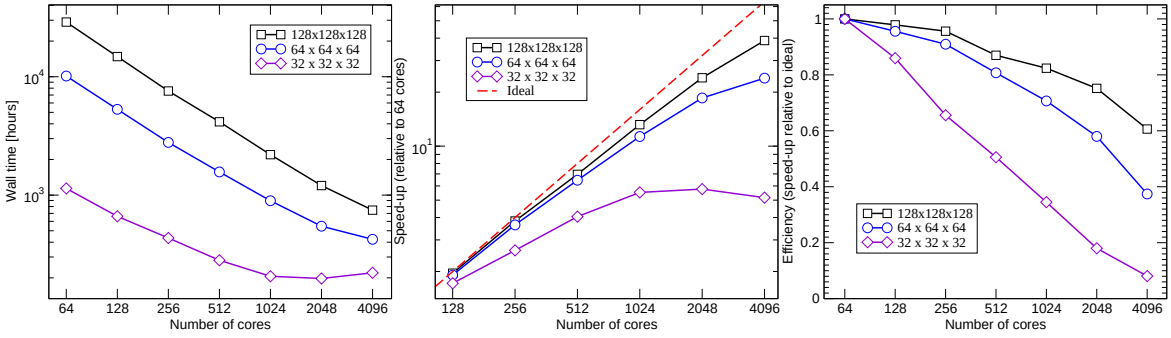


Figure 6.1: (left) Wall clock time as a function of the number of cores. Speedup (middle) and efficiency (right) up to 4096 cores.

Chapter 7

Benchmark Problems

7.1 Odometer Test

This problem concerns the determination of stresses in a Mohr-Coulomb material subjected to an odometer test. In this experiment, two of the principal stress components are equal and, during plastic flow, the stress point evolves along a vertex of the Mohr-Coulomb criterion representation in the principal stress space.

7.1.1 Model Setup

- Bulk modulus = 200 MPa
- Shear modulus = 200 MPa
- Cohesion = 1 MPa
- Friction angle = 10°
- Dilation angle = 10°
- Tension cut-off = 5.67 MPa
- Boundary Conditions: $V_y = -10^{-5}$ m/sec on the top surface. All the other surfaces are confined to their normal directions.
- dt = 1 sec
- Mesh size: 1 m × 1 m × 1 m, and 5 × 5 × 5 nodes.
- σ_{yy} component of stress and ϵ_{yy} component of strain are monitored. These values were averaged over the top elements layer.

7.1.2 Results

- Stress-strain diagram is made using σ_{yy} and ϵ_{yy} from both numerical and analytic solutions.
- Relative error is about 0.8%.

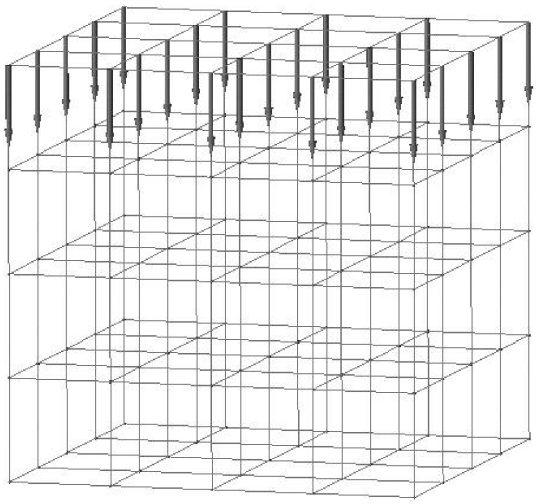


Figure 7.1: Mesh plotted with velocity boundary conditions

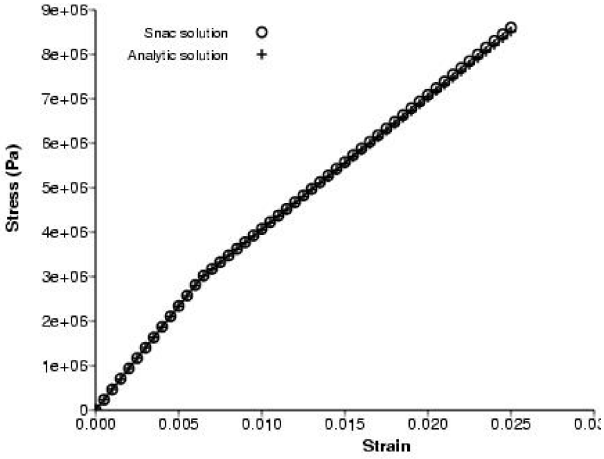


Figure 7.2: Stress-strain plot

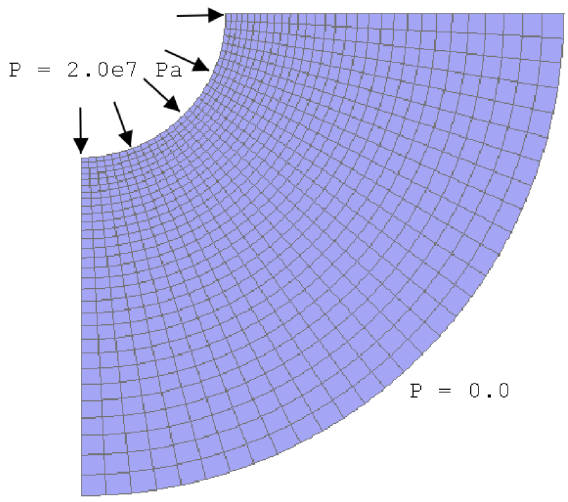


Figure 7.3: Mesh plotted with pressure boundary conditions

7.2 Thick Cylinder with Pressure on the Inner Wall I

A constant and uniform internal pressure is applied to the inner surface of a thick cylinder without pre-stress. Taking advantage of symmetry, only a quadrant of the cylinder is modeled. The outer surface has zero confining pressure. Tresca yield criterion is used (i.e., internal friction is ignored and only cohesion matters).

7.2.1 Model Setup

- Bulk modulus = 200 MPa
- Shear modulus = 200 MPa
- Cohesion = 1 MPa
- Friction angle = 0°
- Dilation angle = 0°
- Tension cut-off = 567 MPa
- Geometry of cylinder: a (inner radius) = 3.0 m, b (outer radius) = 10.0 m.
- Boundary Conditions: P = 20 MPa at r = a, and P = 0.0 at r = b. Top and bottom: V_y=0.0, free-slip.
- dt = 1 sec and results after 5000 steps were used for comparison to the analytic solutions.
- Mesh size: 31 × 3 × 31 nodes.
- The second invariant of stress in xz plane are monitored.

7.2.2 Results

- Relative error is about 0.8%.

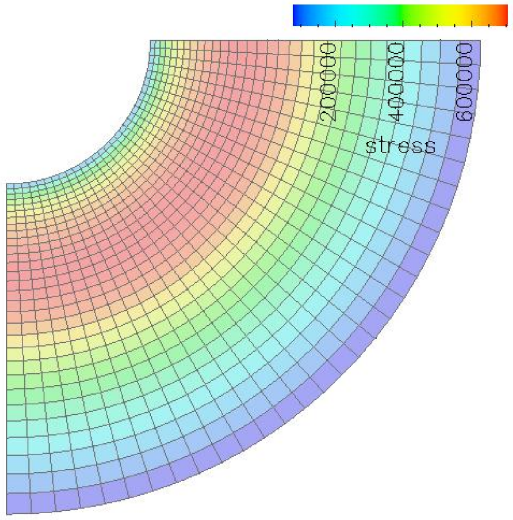


Figure 7.4: Second invariant of stress field

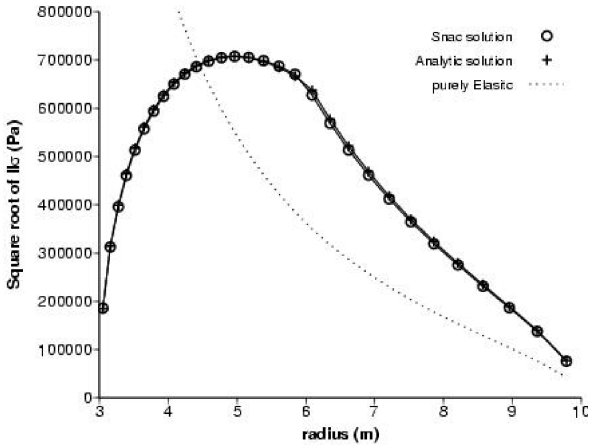


Figure 7.5: Profile of the second invariant of stress along radial direction

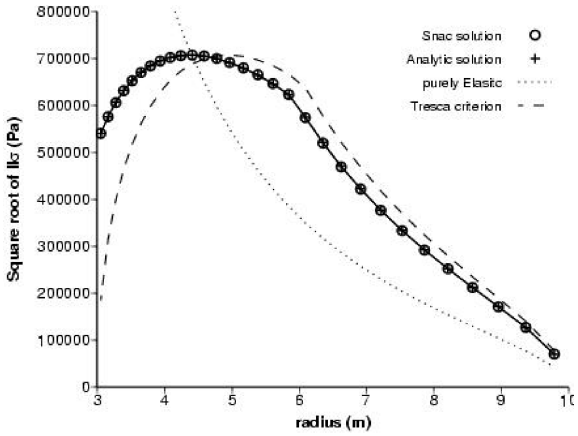


Figure 7.6: Profile of the second invariant of stress along radial direction.

7.3 Thick Cylinder with Pressure on the Inner Wall II

The same problem with the previous section, but the Mohr-Coulomb yield criterion is adopted here.

7.3.1 Model Setup

- Bulk modulus = 200 MPa
- Shear modulus = 200 MPa
- Cohesion = 1 MPa
- Friction angle = 10°
- Dilation angle = 10°
- Tension cut-off = 567 MPa
- Geometry of cylinder: a (inner radius) = 3.0 m, b (outer radius) = 10.0 m.
- Boundary Conditions: P = 2 MPa at r = a, and P = 0.0 at r = b. Top and bottom: V_y= 0.0, free-slip.
- dt = 1 sec and results after 5000 steps were used for comparison to the analytic solutions.
- Mesh size: 31 × 3 × 31 nodes.
- The second invariant of stress in xz plane are monitored.

7.3.2 Results

- Relative error is about 0.1%.

7.4 Parallel-Plate Viscometer Problem

A parallel-plate viscometer problem is simulated, in which viscoelastic material is squeezed between two parallel plates. The plates are moving at a constant velocity, v_0. Each plate has the length of 2L and is at a distance 2h from the other. No slip is assumed between the material and the plates. The approximate analytical solution is given by Jaeger [9].

Resolution	Mohr-Coulomb	Elastic
31 × 3 × 31	0.14%	0.31%
25 × 3 × 25	0.74%	0.39%
21 × 3 × 21	1.05%	0.52%
17 × 3 × 17	1.08%	0.56%
13 × 3 × 13	0.88%	0.59%
9 × 3 × 9	1.12%	0.54%

Table 7.1: Relative errors for Thick Cylinder Benchmark Problem II.

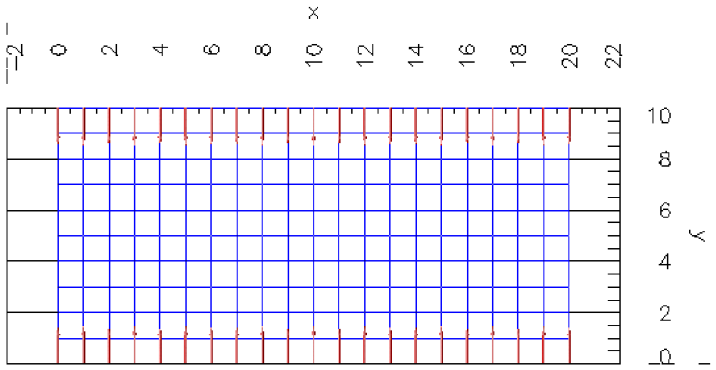


Figure 7.7: The initial mesh (blue) with the velocity boundary condition (red arrows).

7.4.1 Model Setup

- Model Setup
- L = 10 m
- h = 5 m
- Viscosity = 10^9 Pa.sec
- Bulk modulus = 1.5 GPa
- Shear modulus = 500 MPa
- V0 = 10^{-4} m/sec
- dt = 1 sec (results compared after 500 time steps.)
- Mesh size: 20 m × 10 m × 3 m, each element is a 1-m cube.
- Due to the assumption of the original problem setup, artificial forces should be added to left and right surfaces.

7.4.2 Results

Color field represents stress. Colored arrows are for SNAC's velocity, black arrows for the analytic solution. Over the most part of the domain, the two sets of arrows show good agreement.

The relative error of velocity solution is ~5%. This might be acceptable because the analytic solution is only approximate and for viscous fluid. Therefore, refining the mesh does not help reduce the error.

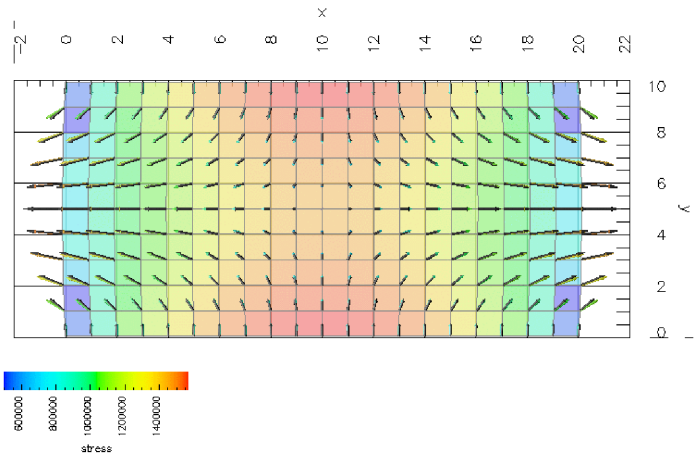


Figure 7.8: The second invariant of stress and velocities plotted on the deformed mesh. Colored arrows are for SNAC's solution, black ones for the analytic solution.

Chapter 8

License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991. Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. Copyright the software, and

2. Offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program" below refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found. For example:

> One line to give the program's name and a brief idea of what it does. Copyright © (year) (name of author)
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items – whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> (signature of Ty Coon)
> 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

[1] Marti, J. and P. Cundall (1982), Mixed discretization procecure for accurate modelling of plastic collapse. *Int. J. Numer. Anal. Methods Geomech., 6,* 129-139.

[2] Cundall, P. (1989), Numerical experiments on localization in frictional materials. *Ingenieur Archiv., 58,* 148-159.

[3] Wilkins, M.L. (1964), Calculation of elastic-plastic flow, *Meth. Comput. Phys., 3,* 211-263.

[4] Needleman, A. (1988), Material rate dependence and mesh sensitivity in localization problems. *Comp. Meth. App. Mech. Eng., 67,* 69-85.

[5] Ortiz, M. and J.C. Simo (1986), An analysis of a new class of integration algorithms for elastoplastic constitutive relations. *Int. J. Num. Meth. Eng., 23,* 353-366.

[6] Ortiz, M. and J.J. Quigley (1991), Adaptive mesh refinement in strain localization problems. *Comput. Methods Appl. Mech. Engrg., 90*, 781-804.

[7] Quenette, S., B. Appelbe, M. Gurnis, L. Hodkinson, L. Moresi, and P. Sunter (2005), An investigation into design for performace and code maintainability in high performance computing. *ANZIAM J. 46* (E), C101–C116.

[8] Tan, E., E. Choi, P. Thoutireddy, M. Gurnis, and M. Aivazis (2004), GeoFramework: Coupling multiple models of mantle convection within a computational framework, *Geochem. Geophys. Geosyst., 7*, Q06001, doi:10.1029/2005GC001155.

[9] Jaeger, J.C. (1969), *Elasticity, Fracture and Flow*, 3rd Ed. New York: John Wiley & Sons, Inc.

[10] Zienkiewicz, O.C., M. Huang, and M. Pastor (1995), Localization problems in plasticity using finite elements with adaptive remeshing, *Int. J. Numer. Anal. Methods Geomech., 19,* 127-148.

[11] Bathe, K.-J. (1996), *Finite Element Procedure.* Upper Saddle River, NJ: Prentice-Hall.

[12] Albert, R., R. Phillips, A. Dombard, and C. Brown (2000), A test of the validity of yield strength envelope with an elastoviscoplastic finite element model. *Geophys. J. Int., 140*, 399-409.

[13] Poliakov, A.N.B., P.A. Cundall, Y.Y. Podladchikov, and V.A. Lyakhovsky (1993), An explicit inertial method for the simulation of viscoelastic flow: An evaluation of elastic effects on diapiric flow in two- and three-layers models. *Flow and Creep in the Solar Systems: Observations, Modeling and Theory*, Kluwer Academic Publishers, 175-195.

[14] Simo, J. and T. Hughes (2004), *Computational Inelasticity.* New York: Springer.

[15] Rudnicki, J. and J. Rice (1975), Conditions for the localization of deformation in pressure-sensitive dilatant materials, *J. Mech. Phys. Solids., 23*, 371-394.

[16] Cundall, P.A. (1987), Distinct Element Models of Rock and Soil Structure, in *Analytical and Computational Methods in Engineering Rock Mechanics*, Chapter 4, pp. 129-163. E.T. Brown, Ed. London: George Allen. & Unwin.

[17] Zienkiewicz, O.C. and Zhu, J.Z. (1992), The superconvergent patch recovery and *a posteriori* error estimates. Part 1: The recovery technique. *Int. J. Num. Meth. Engng.*, *33*, 1331-1364.