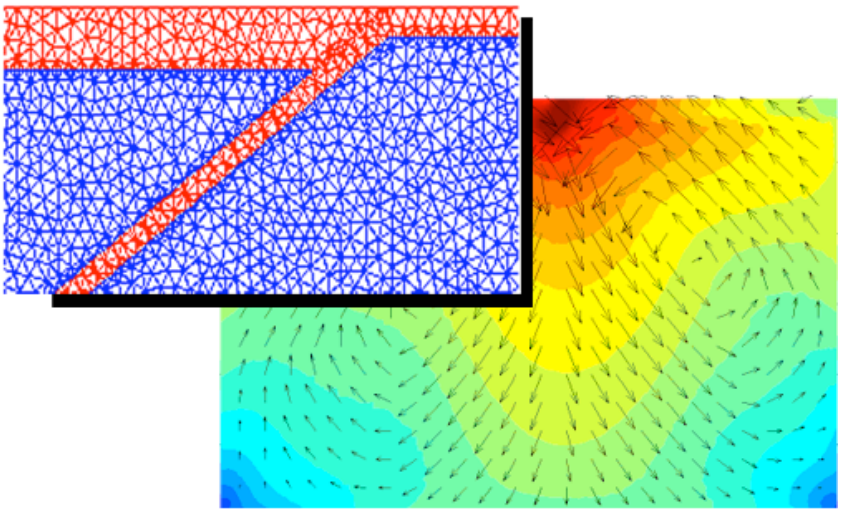
GEOFEST v. 4.5



GEOPHYSICAL FINITE ELEMENT SIMULATION TOOL

User's Guide

rev 5: 04/01/04

Andrea Donnellan (Andrea.Donnellan@jpl.nasa.gov)
Greg Lyzenga (Gregory.A.Lyzenga@jpl.nasa.gov)
Jay Parker (Jay.W.Parker@jpl.nasa.gov)
Charles Norton (Charles.Norton@jpl.nasa.gov)
Maggi Glasscoe (Maggi.Glasscoe@jpl.nasa.gov)
Teresa Baker (Teresa.S.Baker@jpl.nasa.gov)

GeoFEST (Geophysical Finite Element Simulation Tool) is a two- and three-dimensional finite element software package for the modeling of solid stress and strain in geophysical and other continuum domain applications. The program source is written in C, and consists of approximately 11000 lines of source code. The program is targeted to be compiled and run on UNIX systems, and is running on diverse UNIX derivatives including LINUX, HPUX, and SunOS. The present code is descended from earlier generation FORTAN code running under VAX VMS (VISELAS) and an earlier UNIX C code (VISCO). The program uses input and output in the form of formatted plain text files; the data formats can be adapted to accommodate visualization and graphically oriented i/o. This document incorporates a description of parallel GeoFEST, a version of the code designed to run in parallel on distributed memory/cluster parallel architecture computers. The computational engine of the program employs Crout factorization for the direct inversion of the finite element matrices as well as conjugate gradient for an iterative solution option. (At present, only the iterative solver option is supported by the parallel code.) The physics models supported by the code include isotropic linear elasticity and both Newtonian and power-law viscoelasticity, via implicit/explicit quasi-static time stepping. In addition to triangular, quadrilateral, tetrahedral and hexahedral continuum elements, the program supports split-node faulting, body forces and surface tractions. Capabilities under development include frictional faulting and buoyancy driving. Scientific applications of the code include the modeling of static and transient co- and post-seismic earth deformation, Earth response to glacial, atmospheric and hydrological loading, and other scenarios involving the bulk deformation of geologic media.

TABLE OF CONTENTS

Appendices

INTRODUCTION

In order to simulate viscoelastic stress and flow in a realistic model of the Earth's crust and upper mantle, the modeling technique must be able to accommodate a vertically layered structure with imbedded faults which may cut at arbitrary angles. Stress and displacement features will vary most rapidly near the faults and particularly near fault-terminations. These features argue for fully three-dimensional finite element modeling in the time domain. Two-dimensional modeling, semi-analytical techniques, finite difference and semi-spectral methods either cannot model significant features or geometry of interest, or require gross over-sampling in regions of little interest, leading to impossible computational requirements.

Finite element modeling in three dimensions allows faithful modeling of complex faulting geometry, inhomogeneous materials, realistic viscous flow, and a wide variety of fault slip models and boundary conditions. While there are particular problems that are more efficiently expressed using alternative approaches, finite elements represent the most generally useful method for inhomogeneous elastostatic and viscoelastic problems. Because finite elements conform to (nearly) any surface geometry and support wide variations in mesh density, solutions may be made arbitrarily accurate with high computational efficiency. This flexibility comes with a price tag for the user or the tool-builder: that of generating and adapting the mesh of elements upon which the solution is computed. When such generation tools are primitive, researchers may spend substantially more time creating a mesh than solving their problem and interpreting the results. Therefore we describe automated tools for creating and adapting the mesh, including using an initial coarse mesh to generate a solution, whose computable error characteristics inform a further mesh generation cycle and produce an efficient and accurate solution.

GeoFEST uses stress-displacement finite elements to model stress and strain due to: 1) elastostatic response to an earthquake event in the region of the slipping fault, 2) the time dependent viscoelastic relaxation, and 3) the net effects from a series of earthquakes. The physical domain may be two- or fully three-dimensional and may contain heterogeneous rheology and an arbitrary network of faults. The software is intended to simulate viscoelastic stress and flow in a realistic model of the earth’s crust and upper mantle in a complex region such as the Los Angeles Basin.

We describe the supported features of the GeoFEST code as of this writing in the next section. The precise description of the codes' operation and mathematical specification follows in the Theory of Operation section. The entries in input and output files are listed next, enabling a user to set up an input file with a text editor, and run GeoFEST on that basis. We next describe how to compile and run both the sequential and parallel versions of the code. We include a sample annotated input file and finally describe the program structure and functional routines.

FEATURES

The primary quantity computed by GeoFEST is the displacement at each point in a domain. The stress tensor is also computed as a necessary byproduct. The computational domain represents a region of the earth's crust and possibly underlying mantle. It is typically a square or rectangular domain in map view, with a flat upper free surface and constant depth, but the domain may deviate from this. The only requirement is that it be a bounded 3D domain with appropriate surface boundary conditions to render the problem well defined. These boundary conditions may be specified as surface tractions and/or displacements, which are usually specified on all surfaces and possibly on interior surfaces such as faults. Free surfaces have zero surface traction by definition. Faults are interior surfaces, and may have associated dislocation increments at set times, when a stress threshold is exceeded, or according to some other fault friction/triggering model. The solid domain may contain layers or other distributions of material with associated rheological properties. Currently supported materials are isotropic, Newtonian elastic, Newtonian viscoelastic, and non-Newtonian power-law viscosity.

Elastostatic solutions are supported, such as computing the displacements and stresses immediately caused by a specified slip distribution on a fault or finding the interior displacement and stress distribution due to a surface traction or displacement. These solutions are not time-dependent.

Viscoelastic solutions are also supported, in which the material flows and relaxes in response to imposed stress, such as an earthquake event. One may compute the viscoelastic response to a single event, or to multiple events in a sequence. The sequence may be user- specified, or may entail fault slips that are dynamically determined to occur according to a fault triggering model. Location-specific body forces are supported.

Boundary conditions and solutions apply to a finite-element discretized approximation to this domain. The domain is defined internally as a mesh of space-filling tetrahedral or hexahedral elements, with three components of displacement at each mesh node constituting the solution. Stress is computed for each element, and is element-wise constant for the current linear tetrahedral element type. Surface nodes carry special boundary conditions such as tractions or specified displacements. Nodes on faults are special split-nodes that define screw or tensile dislocation on the fault without perturbing the mesh geometry. Temporal evolution is by discrete time steps using an implicit solution technique, allowing large time steps without numerical instability.

The code may be used with all nodes, elements, faults, boundary conditions and time history control created or modified by word processor, constituting the only needed input file. For large meshes hand-construction becomes impractical, and we have sought to support tools for automated mesh generation. Initial attempts have focused on having the user specify fault rectangles, materials, and mesh density, and then semi-automated tools produce the ASCII input file.

Other supported features include:

- Specification of temporal epochs, each with differing steady boundary conditions
- Boundary velocity condition (steady change in the displacement, imposed)
- Controls for generating output on a subset of nodes/elements
- Control of implicit integration parameter
- Ability to shortcut temporal advance by fixing the sparse system for several time steps
- Control of checkpointing, saving state and allowing restart
- Checks a control file at each iteration, allowing clean interruption by the user.

THEORY OF OPERATION

It is worth noting that a central theme to finite elements is the set of shape functions for individual elements. For each linear tetrahedron these are quite simple: for each node, define the function that is unity at that node and linearly declines to zero at the entire opposite face. We may also speak of global shape functions: for each element that shares a particular node, define the function that is unity at that node and declines linearly to zero at the opposite faces of each of the sharing tetrahedra. Shape functions so defined are called interpolatory: if we determine a list of displacements at each node, we may interpolate those values to any point interior to any element. Simply multiply each node displacement by that node's shape function, and sum over those product terms that are nonzero (these will be terms from the nodes defining the enclosing tetrahedron of the interpolation point). Such interpolated displacements will be continuous across all element boundaries (tetrahedral facets).

Within the volume of an isolated finite element, appropriate derivatives of the element shape functions weight the spatial derivatives of the stress tensor (defined from the strain and generalized Hooke's law, relying on locally defined element rheology) to enforce local equilibrium. The shape functions constitute a finite degrees-of-freedom approximation to the continuous system, and a volume integral enforces the equilibrium in a weighted average sense. Elements sharing a node contribute such weighted average terms to an equation for a single displacement. Away from boundary conditions and absent body forces, this set of displacement equation terms is set to zero. Body forces and surface tractions add forcing terms to the right-hand side.

The ensemble system of equations so defined is sparse, and if boundary conditions are sufficient the system is closed and solvable by standard sparse matrix techniques (currently by direct Crout factorization, or optionally by iterative techniques). Closed boundary conditions are usually easy to obtain; the user does have to ensure that enough of the boundary is constrained that there are no unconstrained body translations or rotations permitted to the domain.

The solution is the elastostatic solution for the posed problem in terms of displacements at each node. Local linear combinations of these displacements with shape function derivatives yield the stress tensor in each element.

The elastostatic solution is required for any viscous relaxation computation. Once the static step is complete, the time evolution of quasi-static viscous relaxation may begin by computing the viscoplastic strain rate, which is directly determined by the stress and the viscosity parameters. Conceptually this rate adds a force term to the right-hand-side of a sparse system similar to the elastostatic equilibrium. In practice, to obtain the advantages of implicit temporal development, terms are rearranged to modify the sparse equation coefficients as well. Each time step involves a solution to a sparse equation system, of similar cost as the elastostatic solution.

Faults are specified as either fault elements or as split nodes. With fault elements the user specifies fault properties such as failure criterion in an input record similar to that for a finite element. With split nodes the user specifies for each node on the fault its direction and amount of slip. The fault will slip at the initial time step, and also add the same increment of slip at each fault failure event, specified by the user. The split node formalism may be considered to represent the screw dislocation at a node as a separate entity from the displacement at that node, but is implemented here as an equivalent increment in the stress affecting the nodes immediately adjacent.

§ Mathematical Equations for the Viscoelastic Mechanics Problem

We describe the quasi-static mathematical equations for viscoelastic materials, which is the assumed material type of the solid earth being modeled. In the following, σ and ε denote second-order stress tensors for stress and strain fields, respectively, and *u* is the displacement field. The summation convention is used for repeated indices; a comma is used to denote a partial derivative with respect to a spatial dimension in a Cartesian coordinate system. In R^3, for example, we have:

$$\frac{\partial \sigma_{ij}}{\partial x_j} = \sigma_{ij,j} = \sigma_{i1,1} + \sigma_{i2,2} + \sigma_{i3,3}$$

The considered equations include:

$$\sigma_{ij,j} + f_i = 0, \tag{1.1}$$

the equilibrium equation, where f_i is the given body force,

$$\frac{\partial \sigma_{ij}}{\partial t} = c_{ijkl}\left(\frac{\varepsilon_{kl}}{\partial t} - \frac{\varepsilon_{kl}^{vp}}{\partial t}\right), \tag{1.2}$$

the constitutive equation, where c_{ijkl} are material-specific constants, and

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right), \tag{1.3}$$

$$\frac{\partial \varepsilon_{ij}^{vp}}{\partial t} = \beta_{ij}\left(\sigma_{ij}\right), \tag{1.4}$$

where ε^{vp} is the viscoplastic strain, and β_{ij} are viscoplastic strain rates which are given functions of the stress field. The problem to be solved is formulated as an initial boundary value problem in a domain $\Omega \subset R^n$, where n = 2 or 3. We want to find a displacement field $u(x,t)$ and a stress tensor field $\sigma_{ij}(x,t)$ which satisfy equations (1.1) to (1.4) for all $x \in \Omega$ and $t \in [0,T]$, $T > 0$, such that:

$$\begin{aligned} &u_i(x,t) = u_0(x),\ x \in \Omega \\ &\sigma_{ij}(x,0) = \sigma_{0ij}(x),\ x \in \Omega \\ &u_i(x,t) = g_i(x,t),\ x \in \Omega_1,\ t \in [0,T] \\ &\sigma_{ij} n_j = h_i(x,t),\ x \in \Omega_2,\ t \in [0,T] \end{aligned} \tag{1.5}$$

where u_0 and σ_0 are the initial displacement and stress fields, respectively, $\partial\Omega = \partial\Omega_1 + \partial\Omega_2$ is the domain boundary, n is an outward normal vector to $\partial\Omega_2$, and $g_i(x,t)$ and $h_i(x,t)$ are prescribed boundary displacement and tractions, respectively.

For isotropic (Newtonian) material, the material constants in (1.2) can be expressed as:

$$c_{ijkl} = \mu(x)\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right) + \lambda(x)\delta_{ij}\delta_{kl}, \tag{1.6}$$

where λ and μ are known as *Lamé parameters,* which are related to Young's modulus, E and Poisson's ratio, ν by

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)},\ \mu = \frac{E}{2(1+\nu)}. \tag{1.7}$$

§ Finite Element Formulation

In a finite element approximate solution to problem (1.1) - (1.5), we seek an approximate displacement field $u_i(x,t) \in S$, where S is a finite-dimensional trial solution space with each u_i in S satisfying $u_i = g_i$ (the essential boundary condition) on $\partial\Omega_1$. We also define a finite-dimensional variation space V_i with each $w_i \in V$ satisfying w_i = 0 on $\partial\Omega_2$. u_i must satisfy the "weak form" of the problem (1.1) - (1.5), given below:

$$\boxed{\begin{array}{l}\text{Find } u_i \in S_i \text{ such that for all } w_i \in V_i, \\ \int_\Omega w_{(i,j)}\sigma_{ij}d\Omega = \int_\Omega w_i f_i d\Omega + \sum_{i=1}^{n}\left(\int_{\partial\Omega_2} w_i h_i d\Omega\right)\end{array}} \tag{2.1}$$

where $w_{ij} = w_{i,j} + w_{j,i}, \sigma_{ij}$ is related to u_i through (1.2) and (1.3), and n is the spatial dimension. w_i is sometimes referred to as *virtual displacements* in solid mechanics. Under some smoothness assumptions on the involved variables, it can be shown that a solution to (2.1) is a solution to (1.1) - (1.5) and vice versa.

To find a numerical solution to the finite element problem (2.1), all the variables and the integral equation in (2.1) are discretized on a *finite element mesh*. In the GeoFEST program implementation, the discrete displacement field u^h is defined at nodal points of the mesh, and stress field σ^h and strain field ε^h are defined at the center of a mesh cell (an element).

Using the definition of (1.6), and using a certain mapping of the indices of *i, j, k, l,* to indices *I, J* [2], it can be shown that:

$$\sigma = D\varepsilon(u) \tag{2.2}$$

where, in R^2,

$$D = \begin{bmatrix} \lambda+2\mu & \lambda & 0 \\ \lambda & \lambda+2\mu & 0 \\ 0 & 0 & \lambda+2\mu \end{bmatrix},\ \varepsilon(u) = \begin{Bmatrix} u_{1,1} \\ u_{2,2} \\ u_{1,2}+u_{2,1} \end{Bmatrix}$$

Now define

$$a(w,u) = \int_\Omega \varepsilon(w)^T D\varepsilon(u)d\Omega.$$

Let

$$u^h = v^h + g^h,$$

where $u^h = \left\{u_1^h,\ldots,u_n^h\right\}^T$, $v^h = \left\{v_1^h,\ldots,v_n^h\right\}^T$, and $g^h = \left\{g_1^h,\ldots,g_n^h\right\}^T$ are vectors in R^n, $v_i^h \in V_i$ vanishes on $\partial\Omega_2$, g_i^h satisfies the boundary conditions on $\partial\Omega_1$, so $u_i = g_i$ on $\partial\Omega_1$. In particular, let

A = total nodes in the mesh,
η_{eb} = set of nodes on which $u_i = g_i$,

and

$$v_i^h = \sum_{a \in (A - \eta_{eb})} N_a d_a, \quad g_i = \sum_{a \in (A - \eta_{eb})} N_a g_a$$

where N_a is the "shape function" associated with node *a*; N_a takes unit value at node *a* and vanishes on neighboring nodes of *a* ; d_a is the displacement value at node *a* which is an unknown to be computed. Let e_i be a basis vector in R^n with its i-th component equal to one and other components equal to zero. We have:

$$v^h = v_i^h e_i, \quad g^h = g_i^h e_i.$$

Also let

$$w^h = w_i^h e_i, \quad w_i^h = \sum_{a \in \eta_{eb}} N_a c_a,$$

where c_a are arbitrary constants.

Substituting the previous definitions into (2.1), we get a matrix equation for the displacement vector *d:*

$$Kd = F = (F_1 + F_2), \tag{2.3}$$

where $K = [k_{pq}] \in R^{m \times m}$ is the so-called stiffness matrix. *K* is symmetric and positive definite, and

$$m = \sum_{a \in A} n_a^{dof}$$

where n_a^{dof} is the degree of freedom at node *a* . An entry of matrix *K*, k_{pq}, has the form

$$k_{pq} = a(N_a e_i, N_b e_j) = \int_\Omega B_a D B_b d\Omega e_j$$

where global equation numbers, *p q* and global node numbers, *a b* are related through a certain defined mapping. In R^2:

$$B_a = \begin{bmatrix} N_{a,1} & 0 \\ 0 & N_{a,2} \\ N_{a,2} & N_{a,1} \end{bmatrix}.$$

F_1 and F_2 on the right-hand side of (2.3) are known vectors in R^m. *F_1* includes the contributions from the body force and boundary condition terms. And $F_2 = \int_\Omega B_a^T D\varepsilon^{vp} d\Omega$ is the contribution from the viscoplastic strain.

§ An Implicit Time-Stepping Scheme (Hughes & Taylor)

A time-stepping scheme is needed to compute a viscoelastic finite element solution of displacement and stress fields at discrete time points over a given time period. Both explicit and implicit time-stepping schemes can be formulated. The GeoFEST program adopted an implicit scheme because of its unconditional numerical stability with respect to time step sizes. The entire solution process consists of an initial solution of a pure elastic problem for which the viscoplastic strain rate is set to zero. The pure elastic solution provides an initial stress field, which is then relaxed over a time period in a viscoelastic solution for which an implicit stepping scheme is used. This algorithm used by GeoFEST is described in the following steps:

1. Initialize, set n=0
   a. Form K_0 and f_0
   b. Solve $Ku_0 = f_0$
   c. $\sigma_0 = DBu_0$

2. Form step stiffness matrix and right-hand side

$$K_{n+1} = \int_\Omega B^T \left(S + \alpha\Delta t\beta_n^{'}\right)^{-1} B d\Omega$$

$$F_{n+1} = \int_\Omega B^T \left(S + \alpha\Delta t\beta_n^{'}\right)^{-1} \left(\Delta t\beta_n\right) d\Omega + f_{n+1}$$

   where $S = D^{-1}$, $0 < \alpha < 1$.

3. Solve $K_{n+1}\delta u_{n+1} = F_{n+1}$

4. Stress increment: $\delta\sigma_{n+1} = \left(S + \alpha\Delta t\beta_n^{'}\right)^{-1}\left(B\delta u_{n+1} - \Delta t\beta_n\right)$

5. Update displacement and stress fields:

$$u_{n+1} = u_n + \delta u_{n+1}$$

$$\sigma_{n+1} = \sigma_n + \delta\sigma_{n+1}$$

6. If (last_time_step)
   stop
   Else
   set n = n+1 ,
   go back to 2.

In the above scheme, the viscoplastic strain rate, $\beta(\sigma)$, and its Jacobian matrix, $\beta'(\sigma)$, need to be specified. In R^2 , they are:

$$\beta(\sigma) = \frac{\kappa}{4\eta}\begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 4 \end{bmatrix}, \quad \beta'(\sigma) = \frac{\kappa}{4\eta}\begin{bmatrix} a & -a & d\sigma_{xy} \\ -a & a & -d\sigma_{xy} \\ d\sigma_{xy} & -d\sigma_{xy} & 4b \end{bmatrix}$$

where

$$\kappa = \sqrt{\left(\frac{\sigma_{xx} - \sigma_{xy}}{2}\right)^2 + \sigma_{xy}^2}$$

$$a = 1 + \left(\frac{\sigma_{xx} - \sigma_{xy}}{2\kappa}\right)^2, \quad b = 1 + \left(\frac{\sigma_{xy}}{\kappa}\right)^2, \quad d = \left(\frac{\sigma_{xx} - \sigma_{xy}}{\kappa^2}\right)$$

§ Fault Specification and Split Node Implementation

Fault conditions can be specified either as fault elements or as split nodes. With fault elements, one can specify fault properties such as failure criterion. With split nodes, one can represent the rate of displacement of a fault surface by assigning the direction and amount of slip for each node on the fault surface. Typically, a split node has different slip rates assigned to it on each side of the fault surface, which introduces a discontinuity in the displacement field to simulate real fault slip. This idea can be illustrated by a simple one-dimensional example with two elements, as shown in Figure 1.

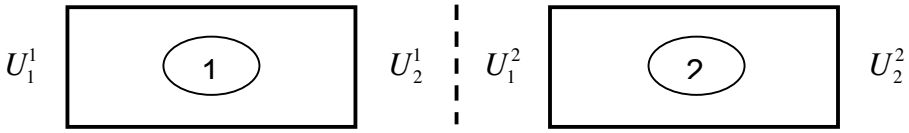


Figure 1

It is assumed that elements 1 and 2 are located adjacent to the opposite sides of the fault surface represented by a dash line between the two elements, and *U* is the displacement field. Away from the fault, displacement field has a single value defined at each node of the 1-D finite element mesh, such as U_1^1 on the left node of element 1 and

U_2^2 on the right node of element 2. The node between the two elements is considered a split node since it lands on the fault. The displacement field has different values at the split node, which are U_2^1 on the side of elements *A* and U_1^2 on the side of element *B*. Specifically we can write:

$$U_2^1 = \overline{U_2^1} + \Delta U_2^1, \;\; U_1^2 = \overline{U_2^2} + \Delta U_2^1$$

where $\overline{U_2^1} = \overline{U_1^2}$ is the mean value of displacement at the split node, and $\Delta U_2^1 = -\Delta U_1^2$ is the "splitting" part of displacement that has opposite signs on two sides of the fault. In a finite element implementation, the contribution from the splitting displacements can be formulated as an additional forcing term. This fact can also be shown using the two element example. The local stiffness matrix for element 1 can be written as:

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix} \begin{bmatrix} U_1^1 \\ \overline{U_2^1} + \Delta U_2^1 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 \end{bmatrix}$$

which relates local displacements to local force terms. By moving the known quantities of the above equation to the right-hand side, we have:

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix} \begin{bmatrix} U_1^1 \\ \overline{U_2^1} \end{bmatrix} = \begin{bmatrix} F_1^1 - K_{12}^1 \Delta U_2^1 \\ F_2^1 - K_{22}^1 \Delta U_2^1 \end{bmatrix}. \tag{3.1}$$

Similarly for element 2, we have:

$$\begin{bmatrix} K_{11}^2 & K_{12}^2 \\ K_{21}^2 & K_{22}^2 \end{bmatrix} \begin{bmatrix} U_1^2 \\ \overline{U_2^2} \end{bmatrix} = \begin{bmatrix} F_1^2 - K_{12}^2 \Delta U_1^2 \\ F_2^2 - K_{22}^2 \Delta U_1^2 \end{bmatrix}. \tag{3.2}$$

"Assembling" the local stiffness matrix equations into a global stiffness matrix equation, we get:

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 \\ K_{21}^1 & K_{22}^1 + K_{12}^2 & K_{12}^2 \\ 0 & K_{21}^2 & K_{22}^2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} = \begin{bmatrix} F_1 - K_{12}^1 \Delta U_2^1 \\ F_2 - K_{22}^1 \Delta U_2^1 - K_{11}^2 \Delta U_1^2 \\ F_3 - K_{21}^2 \Delta U_1^2 \end{bmatrix} \tag{3.3}$$

where U_i 's are global displacements, which are related to the node local displacements by

$$U_1 = U_1^1, \; U_2 = U_2^1 = U_1^2, \; U_3 = U_2^2$$

The global force terms F_i are related to the local ones by

$$F_1 = F_1^1,\ F_2 = F_2^1 + F_1^2,\ F_3 = F_2^2$$

Equations (3.1)-(3.3) show that the effect of the slips on the split nodes is equivalent to adding those additional terms on the right-hand side of the finite element matrix equations.

Stress and displacement at each time are the accumulations of incremental stresses and displacements for past time steps. When a slip event occurs, the incremental displacements are found by applying the split nodes adjustments to the right hand side of the stiffness equation. After the incremental displacement is obtained, the incremental stress is found by including the split node contribution to the stress for that time step. In this way the displacement and stress effects of a slip event are correctly carried forward into future time steps, without any need for additional storage for the slip history of the fault.

§ Basis of Parallel Computation

The parallel version of GeoFEST is designed to be as functionally similar to the original sequential code as possible. From the user perspective, the code is essentially identical, with a few additional steps in order to convert the sequential input file into one that the parallel code can utilize. The basis for the parallel computation performed by GeoFEST is the concept of domain decomposition. The machine model assumed for this style of parallel computing consists of some number of independent processors, each with its own addressable core memory space. This corresponds to a multiple-instruction, multiple-data, or MIMD environment. The processors are each executing identical code, but not synchronously, as each processor acts and branches in distinct ways on its unique data. The processors interact and exchange data with one another by *message passing*, and this communication is mediated in the GeoFEST code through use of the widely known MPI protocol.

At the algorithmic level, domain decomposition requires each of the processors to work on a given spatially contiguous piece of the finite element grid. Such communication as is necessary to update and maintain consistency between the sub-domains where they join one another is the principal challenge of the parallel programming problem. In the GeoFEST parallel decomposition scheme, each processor has exclusive ownership of a block of *finite elements*; from this it follows that there will exist *shared nodes*. These are the nodes that are simultaneously members of elements that belong to two or more different processors. From this scheme it follows that certain tasks (those which are inherently element-based) can be carried out completely in parallel, without need for interprocessor communication. On the other hand, tasks that are inherently node-based will generally require addition of updating steps that communicate shared nodal information between processors.

The calculation and storage of element stiffness matrix contributions is a task of the first kind; once GeoFEST has been given processor assignments for each element (see below for how this is accomplished), the formation of each element contribution can proceed independently in each processor. However, operations involving the assembled vector of nodal displacements that comprise the fundamental unknowns of the problem (see equation 2.3) are of the second kind. This fact leads us to a decision point in choosing the solver for the global finite element matrix equation.

Although there exist means of performing the direct solution of the assembled sparse matrix problem in parallel, our choice of domain decomposition makes the implementation of an iterative solver considerably simpler. Added to this is the expectation that for large three-dimensional problems, the computation time for the iterative method may scale significantly more favorably than direct methods. For this reason, initial development of the parallel GeoFEST implementation has focused on the iterative preconditioned conjugate gradient (PCG) solver exclusively; research on a parallel direct solver will be carried out later.

The PCG algorithm does not require the stiffness matrix to be assembled in global form; it is sufficient to retain the individual element contributions (and accompanying indexing information) in element-specific storage distributed among processors. As for node-based vectors such as the vectors of displacements and forces, each processor stores that subset of the vectors that correspond to the nodes exclusively within its region, along with redundant storage of all the nodal degrees of freedom that are *shared*, that is, that are located on a boundary between processor regions.
The following flow chart outlines the principal steps of the CG solver algorithm:

k=0: $x^{(0)}$ *initial guess*

$r^{(0)} = b - Ax^{(0)}$ *residual*

$d^{(0)} = M^{-1}r^{(0)}$

---

iterate:

$$\alpha^{(k)} = \frac{\left(r^{(k)}, M^{-1}r^{(k)}\right)}{\left(d^{(k)}, Ad^{(k)}\right)}$$

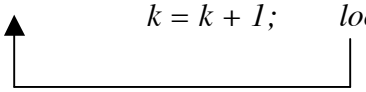
$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$$

$$r^{(k+1)} = r^{(k)} - \alpha^{(k)} Ad^{(k)}$$

$$\beta^{(k+1)} = \frac{\left(r^{(k+1)}, M^{-1}r^{(k+1)}\right)}{\left(r^{(k)}, M^{-1}r^{(k)}\right)}$$

$$d^{(k+1)} = M^{-1}r^{(k+1)} + \beta^{(k+1)} d^{(k)}$$

$k = k + 1;$ *loop*



In the above algorithm, $\boldsymbol{A}$ represents the stiffness matrix, which although it is never actually globally assembled into a matrix as such, its distributed elements are used in calculating intermediate quantities in the iteration. The vector $\boldsymbol{x}$ represents the array of fundamental unknown nodal degrees of freedom; solving for $\boldsymbol{x}$ is the ultimate aim of the algorithm. The superscripts denote the iteration number, so that successive refinements of the approximation to x have successive superscript indices. The vector $\boldsymbol{r}$ is the vector of nodal residuals, which ideally goes to zero as the iteration converges. The iterations are terminated when the magnitude (norm) of this vector is reduced by some specified factor (usually 10^{-7}). $\boldsymbol{M}$ is the preconditioning matrix, which in our case is simply the matrix of diagonal elements in the global stiffness matrix. The vector $\boldsymbol{d}$ and the scalar quantities α and β are intermediate quantities used exclusively internally to the PCG algorithm.

Note that the two important tasks in the algorithm that require interprocessor communication are the vector dot product (denoted above by parentheses) and the stiffness matrix-vector product (denoted by $\boldsymbol{Ad}$). In the case of the former, each processor calculates its contribution to the global scalar product, using the vector entries in its local storage. This is immediately followed by MPI communication calls that combine the pieces into a global result and distribute the product to all processors. At the conclusion of this "blocking" operation, each processor is then free to carry on with its independent process.

The matrix-vector product is carried out similarly, although the communication pattern is somewhat more complex. In this task, each processor carries out the multiplication of locally stored matrix elements with locally stored vector entries. The result is usually a vector entry that "lives" in the local processor, but some of the results will fall on a boundary node which is shared with another processor. In this case, rather than a global (all processors) MPI communication, a pair-wise communication between the involved processors is used to update and reconcile the vector results at all shared nodes, so that at the conclusion of the communication step. All processors will contain vector values that agree with one another, and with the values that would be obtained in the equivalent single-processor sequential calculation.

INPUT/OUTPUT

§ The input file

The user (perhaps via higher-level tools) is required to specify the model domain, constraints, temporal effects, and other solution controls in a single input file. Note that the model domain is a discretized version of the continuous domain described above. Therefore the volume is defined by its tetrahedral/hexahedral elements (or quad/triangular in 2-D), faults and other surfaces are sets of tetrahedral facets

(triangles), and many constraints are specified on nodes. Higher-level constructs such as lines or surfaces may not have explicit descriptive records (although there is a construct for element groups sharing material properties; read on). This input file may be roughly characterized by the following brief list of items, which is in file order:

An output filename
Arbitrary comments
Number of processors and node decomposition (unsupported)
Number of free displacement dimensions
Number of time periods (0 for pure elastic; else number of steady epochs)
Flags: "Save shape functions", "Solver option"
Node free/fixed axis flags
Node coordinates
Node imposed conditions: displacement, velocity
Properties for clusters of elements: type, integration rule, material
Element records (nodes, property ID) (may be a fault element)
Split nodes
Surface tractions
List of nodes and elements for results printing
Number of time groups, plus reform interval, save time, quake time
Time group start times, implicitness parameter, time step
Output times (list, or flag for every earthquake)
Restart file name (optional) (i.e., start where another run left off, using this file)
Save-state file name (optional) (i.e., save this run so another run can start where we left off).

The input file affords considerable power to the user, but is very terse and cryptic. Without a full description of its options and features mistakes and misunderstandings inevitably arise. As a guide to the proper construction and formatting of a GeoFEST input file, we provide an annotated sample input file in the next section. The example is for a two dimensional problem, in order to facilitate easy visualization of the domain; however the generalization to 3-D is straightforward. Note the convention used in the input file that many vertical lists use the special flag "0 0" to indicate termination.

§ The output file

Outputs come in three flavors:
Standard Out: comments, summary info, progress messages.
Named output file (the name is in the first line of the input): the requested displacement and stress information at the requested times.
Restart file: contains simulation state. This is in ASCII, but is intended for machine reading.

The output file contains information on the cumulative displacements, the delta displacements (change in displacement between time steps) and the element stress information for each reported time step.

The displacement and stress information is presented in the following format:

2D

Global coordinates & displacements & delt displacements
Simulation time = 0.000000 ; step size = 0

Coordinates			Displacements		Delta displacements	
Node #	x-location	y-location	u_x	u_y	du_x	du_y

Element stresses -- element group #1
Simulation time = 0.000000 ; step size = 0

Coordinates			Stresses				Element
Node #	x-location	y-location	σ_{xx}	σ_{yy}	σ_{zz}	σ_{xy}	Element #

3D

Global coordinates & displacements & delt displacements
Simulation time = 0.000000 ; step size = 0

Coordinates				Displacements			Delta displacements		
Node #	x-loc	y-loc	z-loc	u_x	u_y	u_z	du_x	du_y	du_z

Element stresses -- element group #1
Simulation time = 0.000000 ; step size = 0

Coordinates				Stresses						Element
Node #	x-loc	y-loc	z-loc	σ_{xx}	σ_{yy}	σ_{zz}	σ_{xy}	σ_{xz}	σ_{yz}	Element #

RUNNING GEOFEST

§ Compiling the GeoFEST sequential version

Compiling the code is a straightforward process. The user downloads the .tar file with the latest GeoFEST version available. Once the .tar file is expanded, the directory /geofest is created. Within this directory is a Makefile. The user simply types

```
% make
```

And the code will compile, creating an executable named GeoFEST.

§ Running the GeoFEST sequential version

To run the sequential version of the GeoFEST code, the user simply calls the executable in the directory in which the input file is located. The input file specifies the output file name and will write the output in the same directory in which the executable is called.

For example, to run a model with the input file input.dat, the user types

```
% GeoFEST input.dat
```

The output file specified on the first line of the input file will be written in ./ by default.

If the user does not specify the input file in the command line argument, the code will ask for the name of the input file and then proceed.

§ Compiling the GeoFEST parallel version

Again, compiling the parallel version of the code is fairly straightforward. Once downloading the latest distribution and expanding the .tar file, there are a series of steps to follow in order to compile the parallel code and the associated tools packaged in the distribution.

The GeoFEST source code in the GeoFEST/geofest/ directory is maintained separately from Pyramid and ParMETIS distributions. The GeoFEST Makefiles in the GeoFEST/geofest/ directory specify the location of external library archives and header files relative to that directory so that the GeoFEST directory is relocatable.

The Pyramid/ directory or a link to the Pyramid/ directory must appear in the GeoFEST/ directory. The ParMETIS directory or a link to the ParMETIS directory must appear, in turn, in the GeoFEST/Pyramid/Pyramid/ directory.

A platform is the combination of machine architecture, operating system and Fortran 90 compiler. By default,

```
PLATFORM=$(MACHTYPE)-$(OSTYPE)-$(F90TYPE)
```

where MACHTYPE and OSTYPE are UNIX environment variables and the F90TYPE make variable is overridden in the make command line. The appropriate $(PLATFORM).mk file is included in the Makefile to define platform specific Makefile variables.

By default, the Makefiles build GeoFEST using Absoft library archives. This could be overridden for the Numerical Analysis Group (NAG) Fortran 90 compiler by typing

```
make LD='mpicc -v' F90TYPE='Intel' OPTIONS='-Wall -std=c99 -pedantic -O2'
```

or indirectly by simply typing

make -f Makefile.Intel

§ Running the GeoFEST parallel version

Running the parallel version of GeoFEST is very similar to the sequential version, with two main differences. The first is that a .jpl file is required in addition to the regular GeoFEST input file in order for the parallel version to successfully execute. The second is that the output directory path must be specified if the user does not want the output to be written in the ./ directory (which is the default).

One additional step must be taken with the input for the parallel version. The user will use the meshgen program to convert the regular GeoFEST input into an input file that the parallel code can use (with the .jpl extension).

To run the parallel version using MPI (on 8 processors), the user types

% mpirun –np 8 GeoFEST [inputFileName] [outputDirectoryPath]

The outputDirectoryPath is ./ by default. The inputFileName is input.dat by default.

GeoFEST requires both inputFileName and inputFileName.jpl to be in the same directory -- input.dat and input.dat.jpl in the current working directory for example.

ANNOTATED SAMPLE 2D INPUT FILE

1	example1.out	1	output file name – UNIX-style file specification of output file
2	An example 2-dimensional problem* These two lines are for user comments *	2	Two lines terminated by the "*" character are provided for user comments. The comments are echoed in the output files for run identification, but are otherwise ignored by the program.
3	25	3	numnp = number of node points

```
4    2 2 1 1
```

4 global parameters
(a) (b) (c) (d) (e)

- (a) nsd -- number of space dimensions; either 2 or 3
- (b) ndof -- number of degrees of freedom; either 1, 2, or 3
- (c) nrates – number of velocity boundary condition time periods to be specified; 0 if the problem is elastic or has no velocity b.c.'s; 1 if a single set of rates will be used for the whole simulation; more than 1 if the rates change during the run
- (d) shape flag – 1 if element shape functions are to be save in memory; 0 if they are to be recalculated each time elements are formed; should usually be 1 unless memory is tight
- (e) solver flag – 1 if direct matrix inversion (Crout factorization) is to be used to solve the FE equations; 2 if iterative preconditioned conjugate gradient solver is to be used

```
5    1 0 0 0
     2 0 0 0
     3 0 0 0
     4 0 0 0
     5 0 0 0
     6 0 1 0
     7 0 1 1
     8 0 1 1
     9 0 1 1
     10 0 1 0
     11 0 1 0
     12 0 1 1
     13 0 1 1
     14 0 1 1
     15 0 1 0
     16 0 1 0
     17 0 1 1
     18 0 1 1
     19 0 1 1
     20 0 1 0
     21 0 0 0
     22 0 0 0
     23 0 0 0
     24 0 0 0
     25 0 0 0
     0 0
```

5 nodal activity assignments
(a) (b) (c)

- (a) node # -- global index number of node
- (b) gen order – used to specify a range of nodes; generally just leave it 0 unless doing something special
- (c) bc codes – there are ndof entries (two in this example); tells whether each dof for this node is fixed or free; 0 = fixed, 1 = free; end nodal activity box with two zeros

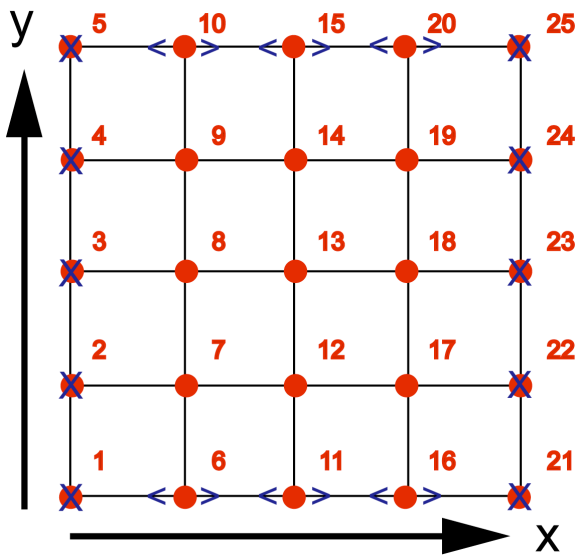


Figure 1. Degrees of freedom in this example. Unlabeled nodes are free (1 1), nodes with a blue cross are fixed (0 0), and nodes with arrows are free in the x-direction only (1 0).

6 1 0 0.0 0.0
2 0 0.0 0.25
3 0 0.0 0.50
4 0 0.0 0.75
5 0 0.0 1.0
6 0 0.25 0.0
7 0 0.25 0.25
8 0 0.25 0.50
9 0 0.25 0.75
10 0 0.25 1.0
11 0 0.50 0.0
12 0 0.50 0.25
13 0 0.50 0.50
14 0 0.50 0.75
15 0 0.50 1.0
16 0 0.75 0.0
17 0 0.75 0.25
18 0 0.75 0.50
19 0 0.75 0.75
20 0 0.75 1.0
21 0 1.0 0.0
22 0 1.0 0.25
23 0 1.0 0.50
24 0 1.0 0.75
25 0 1.0 1.0
0 0

6 nodal coordinates
(a) (b) (c)
18 0 0.75 0.50
(a) node # -- global index number of node
(b) gen order – used to specify a range of nodes; generally just leave it 0 unless doing something special
(c) x and y (and z) coordinates – there are nsd entries; these specify the physical location of this node in the grid

End nodal coordinate block with two zeros.

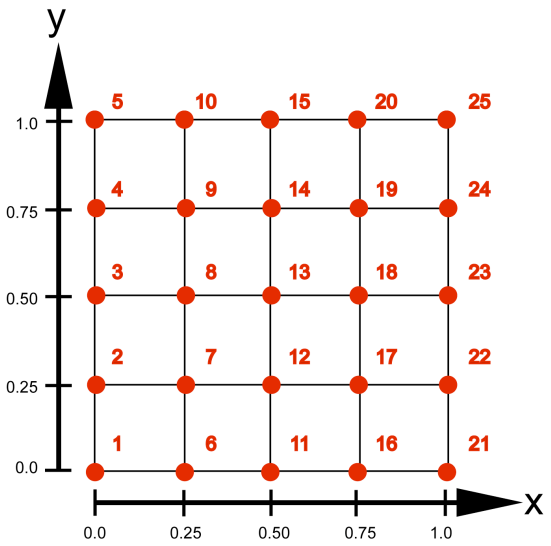


Figure 2. Correspondence of between node numbers and physical coordinates for this example.

7 21 0 –0.1 0.0
22 0 –0.1 0.0
23 0 –0.1 0.0
24 0 –0.1 0.0
25 0 –0.1 0.0
0 0

7 nodal displacement/force boundary conditions – these need not be entered for every node; if unspecified, it is assumed zero; if no b.c.'s are to be given, just enter the two ending zeros

(a) (b) (c)
23 0 -0.1 0.0
(a) node # -- global index number of node
(b) gen order – used to specify a range of nodes; generally just leave it 0 unless doing something speclal
(c) displacement values – there are ndof numbers; these give the 1-, 2-, or 3-dimensional displacement to apply to this node; there are interpreted as specified displacements as long as the corresponding degree of freedom is fixed (0); if the d.o.f. is free (1) then these entries are interpreted as nodal forces

End nodal boundary conditions with two zeros

8 0.000

8 Turn-on time for the boundary velocities in the block immediately following; enter value in units of physical (simulation) time

9	21 0 –0.1 0.0 22 0 –0.1 0.0 23 0 –0.1 0.0 24 0 –0.1 0.0 25 0 –0.1 0.0 0 0	9 nodal velocity boundary conditions – these need not be entered for every node; if unspecified it is assumed zero; these velocities are entered in the same format as displacement boundary conditions (just above); the velocities in this list are applied starting at the time given immediately before this block; the velocity specified is multiplied by the current time step and applied at each step for the designated period; this block (and the preceding turn-on time) are repeated “nrates” times and are absent entirely if nrates = 0 (no velocity b.c.’s) (a) (b) (c) 23 0 -0.1 0.0 (a) node # -- global index number of node (b) gen order – used to specify a range of nodes; generally just leave it 0 unless doing something special (c) velocity values – there are ndof numbers; these give the 1-, 2-, or 3-dimensional velocity to apply to this node End nodal velocity boundary condition block with two zeros
10	1	10 numgrp – number of element groups; the grid may consist of more than one type or group of elements; the following material and ien array blocks would be reprated n times if numgrp = n instead of 1 as in this example
11	16	11 numel – number of elements belonging to this group
12	1 2 0 4 3	12 element parameters (a) (b) (c) (d) (e) (a) element type code 1 = bilinear 4-node quadrilateral (can be degenerated to triangles) 2 = biquadratic 8-node serendipity (can be degenerated to triangles) 4 = linear 4-node tetrahedron 6 = trilinear “brick” hexahedron (can be degenerated into prisms or tetrahedra) coming in a future version – linear truss (bar) elements (b) numat – number of different material property types in this group (c) numsuf – number of element sides to have surface forces applied (*note that this option is currently under development for support in the future*) (d) parameter not used – formerly, it was used to choose the integration rule used in forming the stiffness matrix; that is no longer the case as of version 4.3; now every split node counts only once, regardless of how many elements it is a member of

```
13    1000.0 1000.0 20000.0 1.0 0.0 0.0
      2000.0 2000.0 0.00000 0.0 0.0 0.0
```

13 material properties; this line is repeated numat times

(a) (b) (c) (d) (e)
1000.0 1000.0 20000.0 1.0 0.0 0.0

- (a) mu – elastic rigidity modulus
- (b) lambda – second Lamé elastic modulus
- (c) viscosity coefficient – set to zero for pure elastic behavior
- (d) viscosity exponent – set to 1 for linear Newtonian case
- (e) body forces (gravity) – there are ndof numbers; these give the 1-, 2-, or 3-dimensional body force to apply to elements in this group

```
14    1 0 1 1 6 7 2
      2 0 1 2 7 8 3
      3 0 1 3 8 9 4
      4 0 1 4 9 10 5
      5 0 1 6 11 12 7
      6 0 1 7 12 13 8
      7 0 1 8 13 14 9
      8 0 1 9 14 15 10
      9 0 1 11 16 17 12
      10 0 1 12 17 18 13
      11 0 1 13 18 19 14
      12 0 1 14 19 20 1
      13 0 1 16 21 22 17
      14 0 1 17 22 23 18
      15 0 1 18 23 24 19
      16 0 1 19 24 25 20
      0 0
```

14 element node data; this line repeats numel times, one for each element

(a) (b) (c) (d)
12 0 1 14 19 20 15

- (a) element number – global index of this element
- (b) gen order – used to specify a range of elements; generally just leave it 0 unless doing something special
- (c) material type – which of the above material types to use for this element
- (d) node numbers – global node numbers corresponding to the four corners of this element; ordered in counterclockwise rotation for 2-d cases; repeat third number for collapsed triangles

End element node block with two zeros

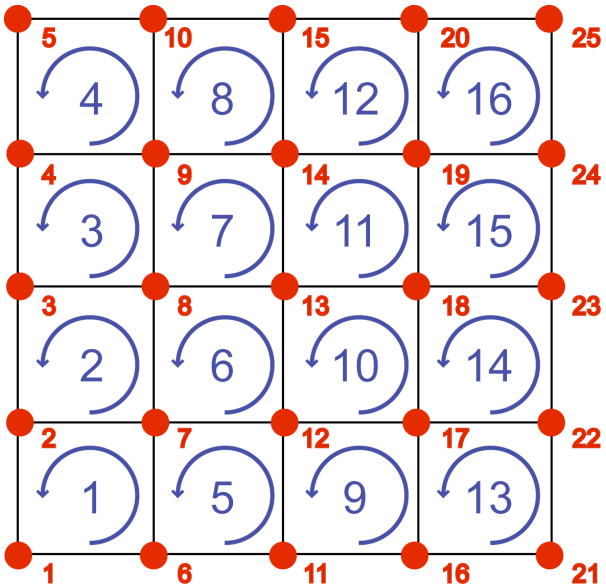


Figure 3. Relationship between global node numbers and element numbers for this example.

```
15    13 0.0 0.0 –1.0 0.0 1.0 0.0 0.4
      14 0.0 0.0 –1.0 0.0 1.0 0.0 0.4
      15 0.0 0.0 –1.0 0.0 1.0 0.0 0.4
```

```
15    split node parameters
      (a)  (b)          (c)          (d)
      14   0.0 0.0 –1.0  0.0 1.0 0.0  0.4
```

(a) node number – global node number of this split node

(b) B-vector – in this case, pointing in the –Z direction

(c) S-vector – in this case, pointing in the +Y direction

(d) slip amplitude – in this case, 0.4 (displacement units)

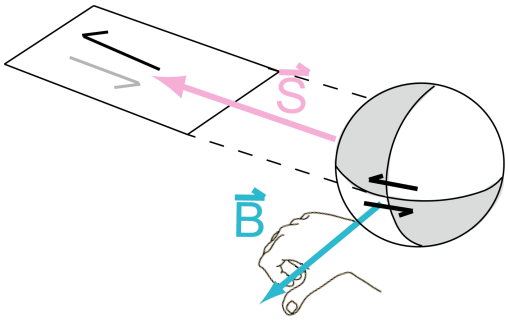


Figure 4. Right-hand-rule convention used for defining the direction of the B-vector for a given fault orientation and sense of slip.

Note that even in 1- and 2-degree of freedom problems, three components are used to specify the B and S vectors; in such cases, the Z-direction is assumed to be the out-of-plane direction, with positive being defined by “X cross Y;” for example, a strike-slip problem with ndof = 1 would have S-vector in the Z-direction; a thrust fault with ndof = 2 would have B-vector in the Z-direction

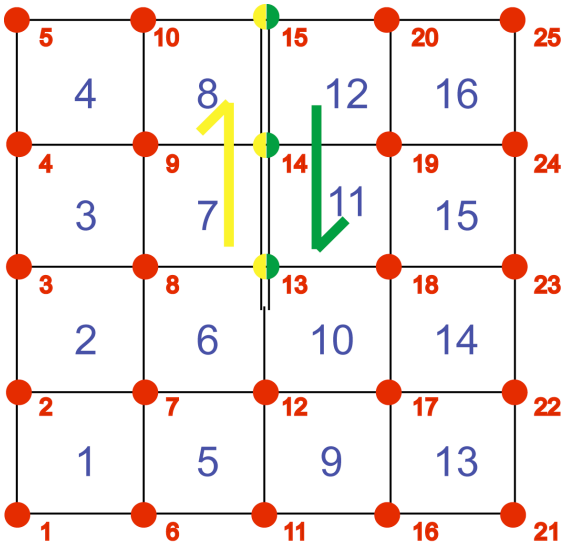


Figure 5. Numbering used for the three split nodes in this example.

16	5 4	16 reporting parameters (a) (b) (a) total number of nodes to report on in output file; set to zero if no nodal output is requested and set to –1 to output every node in the grid (b) total number of elements to report on in output file; set to zero if no element output is requested and set to –1 to output every element in the grid
17	5 10 15 20 25	17 list of which nodes to print in the output file; in this example, there are a total of 5 and note that they can be separated by any white space, including returns; omitted if either 0 or –1 is given as number of elements to output above
18	1 6 1 7 1 10 1 11	18 list of which group numbers and elements to print in output file; note in this example, all elements are from the same group (1): there are a total of four; omitted if either 0 or –1 is given as number of elements to output above
19	3 1 5000 1500.0	19 earthquake and backup parameters (a) (b) (c) (d) (a) number of time groups – number of different time intervals of time step size; this number of lines will follow in the next input block (b) reform steps – number of time steps to take before reforming the stiffness matrix; the standard Hughes implicit algorithm calls for this to be 1; when using the iterative conjugate gradient solver, there is no advantage to setting any value other than 1; when using the direct solver significant times savings may be realized by setting it to a modest number such as 5, but be warned that this is a heuristic shortcut that is without rigorous justification; the stability of the time-stepping ordinarily guaranteed by the implicit scheme may not be assured if this is set > 1 (c) backup steps – number of time steps to take before saving the simulationstate to disk as backup or restart (d) time interval between repeat faulting events; enter in units of physical (simulation) time

20	20.0 1.0 1.0 50.0 1.0 0.5 100.0 1.0 0.2	20	timestep parameters – repeat this line for each time group (a) (b) (c) 50.0 1.0 0.5 (a) use present time step size until this time is reached (b) implicit/explicit parameter; 0 < alpha < 1; usually set to 1.0 (c) time step size (a) and (c) are in the natural time units of the problem and (b) is dimensionless
21	7	21	
22	0.0 1.0 5.0 10.0 30.0 60.0 100.0	22	list of scheduled times to print output
23	NO_RESTART	23	UNIX path name of the file to restart this run from a saved state; if there is no restart file, enter NO_RESTART
24	backup.dat	24	UNIX path name of the file to use to save state of run for backups; if no backup is desired, enter NO_SAVE; this file (if not turned off) will be written at the conclusion of the run, as well as at the periodic intervals specified above as the backup interval
25	This is the end of the file.	25	EOF – end of file; optionally there can be any number of comments written here; they are for the benefit and convenience of the user and are not read by the program

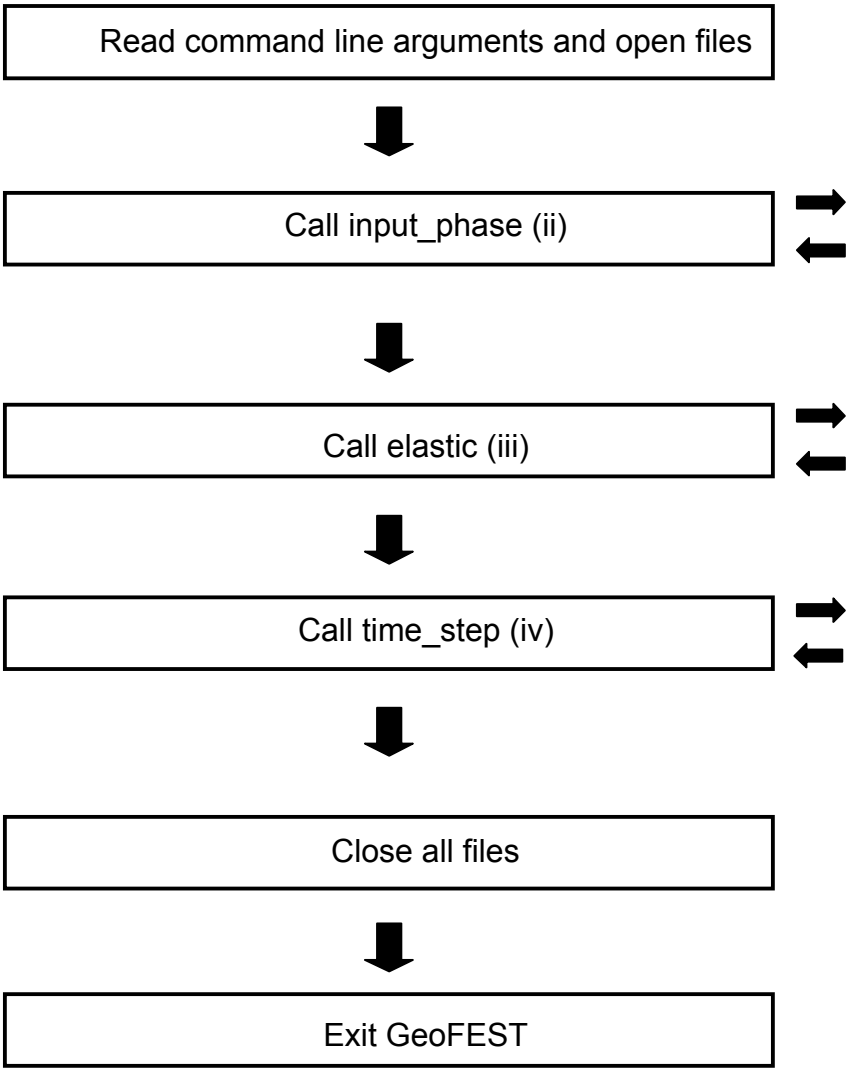
During the run of the program, the user can communicate with and control the execution of the run. A text file named “monitor.fem” should be located in the same directory as the GeoFEST executable program. GeoFEST will check the contents of this file each time step and take action based on the contents of this file:

OK – continue running normally
REPORT – write information about the current executing time step and conitnue running normally
KILL – immediately stop the run without saving anything
SAVE_STOP – stop execution and save the current simulation state to the designated backup file (or to “save.def” if none is specified
SAVE_GO – continue execution after saving the current simulation state to the designated backup file (or to “save.def” if none was specified)

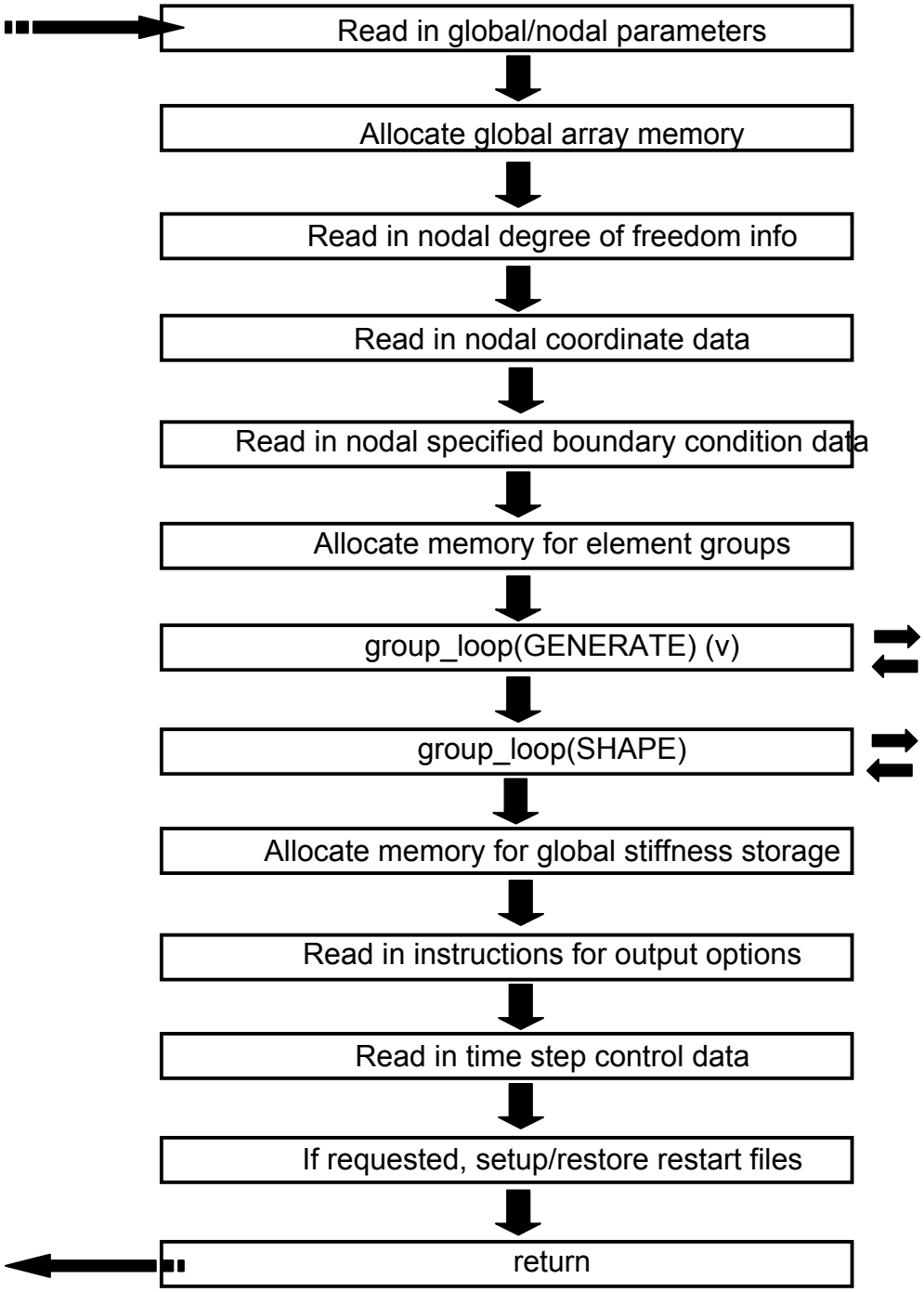
APPENDIX A1.

The following linked flow charts describe the basic execution flow and organization of GeoFEST, identifying the principal functional tasks and processes.

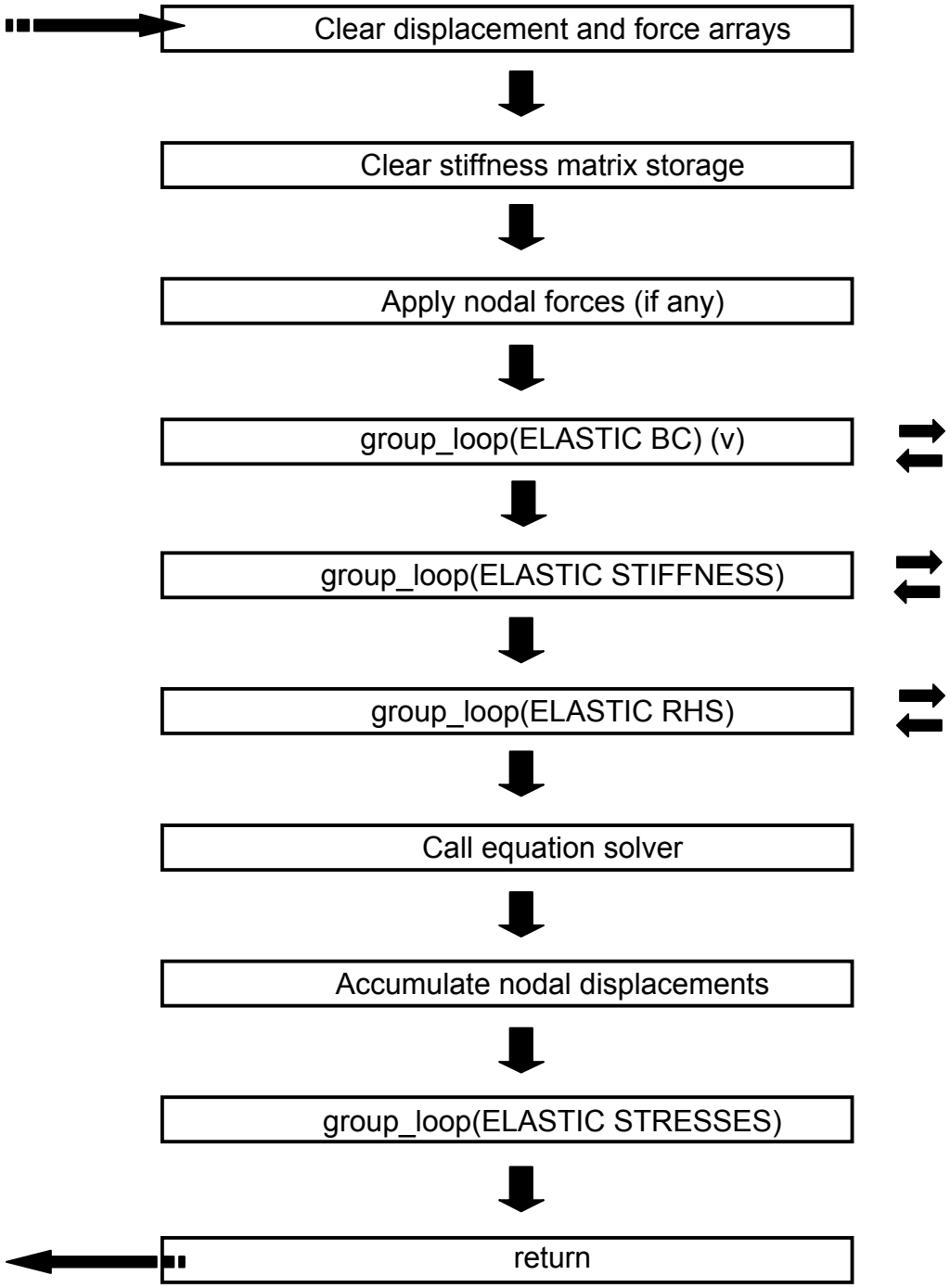
(i) GeoFEST main() entry point



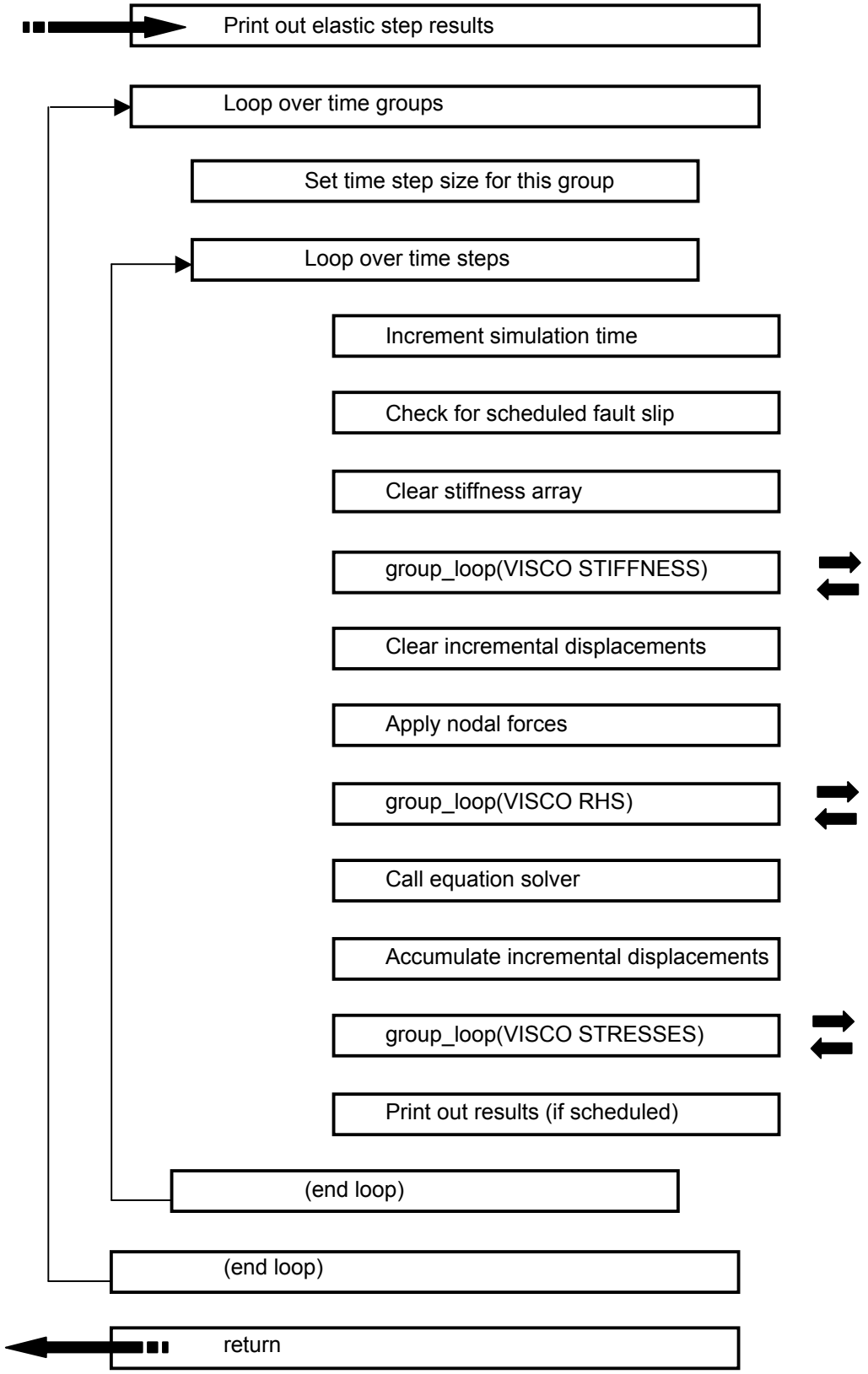
(ii) input phase routine



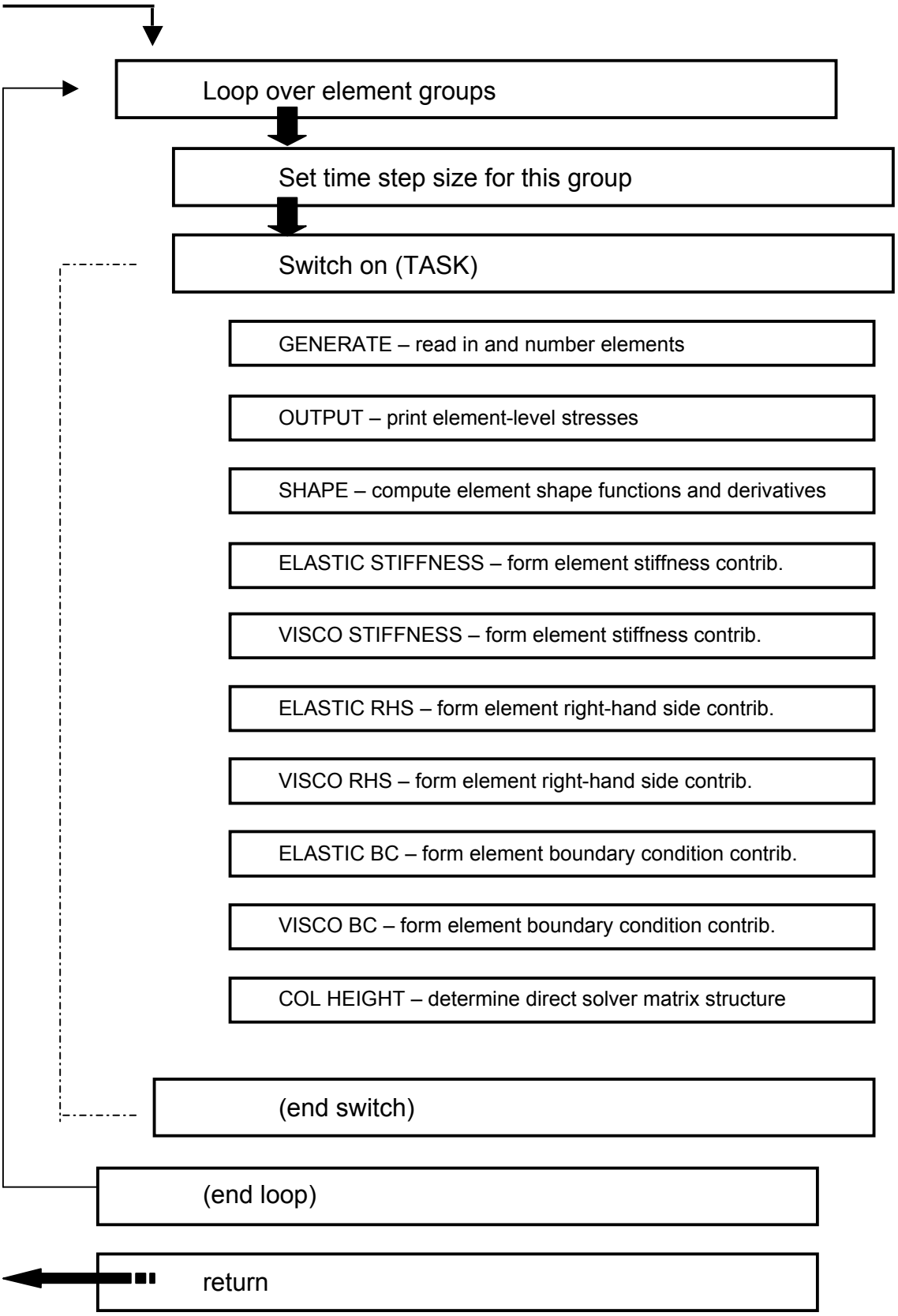
(iii) elastic routine



(iv) time_step routine



(v) group_loop routine



APPENDIX A2.

The following is a listing of all GeoFEST functional routines and the header comments describing their usage and purpose.

FILE MAIN.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains the program main entry point, the main task driver, and modules for driving high-level functions and interactions with the operator:

§ main
§ elastic
§ time_step
§ clear_stiff
§ elgrp_loop
§ node_load
§ accumulate
§ completion
§ wrt_save
§ vrestart
§ el_save
§ check_monitor

ROUTINE: main

- main prints a banner, interprets up to two args as input and output files, scans two comment lines in the input, and divides (and times) the processing into input, elastic solution, time-stepping, and output.

ROUTINE: elastic

- elastic performs an elastic solution of the finite element problem.

ROUTINE: time_step

- time_step computes the visco-elastic time stepping solutions to the finite element problem.

ROUTINE: clear_stiff

- clear_stiff nulls the stiffness matrix memory.

ROUTINE: elgrp_loop

- elgrp_loop invokes a particular task that relies on element structures and affects all the elements in every element group.

- Supported tasks are:
  - GENERATE: read or generate element data, fill ien, lm arrays
  - OUTPUT: compute and write out element centroid stresses
  - SHAPE: compute parent-space shape functions and gradients
  - FORMS_ELAS: compute elastic stiffness matrix
  - FORMS_STEP: compute VE single-step stiffness matrix
  - RHS_ELAS: compute elastic right-hand side vector for FE problem
  - RHS_STEP: compute VE right-hand side vector for FE problem
  - BC_ELAS: compute boundary-condition terms for elastic FE problem
  - BC_STEP: compute boundary-condition terms for VE FE single step
  - COL_HT: compute the column heights for profile stiffness storage
  - E_STRESS: compute the stresses from elastic solution
  - V_STRESS: compute the stresses from a VE step solution
  - DUMP: dump information needed for restarts
  - RESTORE: read and restore solution vector and info for a restart
  - FAIL_CHECK: check for stress-driven fault failure
  - SMOOTH_BEGIN: perform pressure smoothing set-up
  - SMOOTH_END: apply pressure-smoothing function from coefficients
  - REORDER: analyze adjacency, find profile-optimizing node permutation

ROUTINE: node_load

- node_load applies node-based forces that may accumulate with time.

ROUTINE: accumulate

- accumulate updates the total displacement based on the single-step increment.

ROUTINE: completion

completion syncronizes multiple processors. Parallel vestige; keep for now. Keeps track of incurred errors in each processor.

ROUTINE: wrt_save

- wrt_save dumps state to disk for restarts.

ROUTINE: vrestart

- vrestart restores state from previous dump and continues simulation.

ROUTINE: el_save

- el_save dumps or restores element data.

ROUTINE: check_monitor

- check_monitor allows user run intervention via strings stored in the file monitor.fem.

- Supported options:
  - OK: close the dump file (?)
  - KILL: close files and abort
  - SAVE_STOP: write state, then abort
  - SAVE_GO: write dumps as requested in input, and continue

FILE INPHASE.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains subroutines that handle input, equation numbering, and memory allocation.

§ input_phase
§ matrix_alloc
§ gen_number
§ output_phase
§ el_output
§ locate_pt

ROUTINE: input_phase

- input_phase is the top-level input file reading function. Most input-file records are read here or in routines in generat.c that are called by this function. A few items are read by main.c as well.

ROUTINE: matrix_alloc

- matrix allocates space for the sparse stiffness matrix _or_ the PCG solver arrays

ROUTINE: gen_number

- gen_number assigns equation numbers based on nodes and ndof

ROUTINE: output_phase

- output_phase writes out requested data at a quake or scheduled time.

ROUTINE: el_output

- el_output prints out element stress information.

ROUTINE: locate_pt

- locate_pt maps the integration points into physical space. Global coordinates are placed in xpt[j]. Called by el_output element stress output function (above).

FILE GENERAT.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains modules for generating and reading node and element data and related isoparametric tasks.

§ gen_element
§ next_element
§ read_surf
§ read_slip
§ load_element
§ gen_map
§ gen_real
§ gen_isopar_grid
§ gen_shape_grid
§ dot_sh

ROUTINE: gen_element

- gen_element generates element storage and ien array. It reads in element group information, individual elements, and surface traction records.

ROUTINE: next_element

- next_element reads the first part of a record in a double null-terminated list, so the calling function may catch the termination.

- The use of this function is not limited to finite elements, but is also employed for other lists, such as nodes and tractions.

ROUTINE: read_surf

- read_surf reads in surface traction records.

ROUTINE: read_slip

- read_slip reads split-node records for symmetric fault slip accumulation.

ROUTINE: load_element

- load_element loads element arrays with ien and equation data. The information for the element has been previously read into a temporary array or generated.

- This function checks for degeneracies of 2D elements, right-hand rule order of tet nodes (swapping when necessary), and fills in the ien and lm arrays for the element.

ROUTINE: gen_map

- gen_map fills the nodal activity map array.

ROUTINE: gen_real

- gen_real reads or generates floating point global array. Used to read or generate nodes, displacement bcs, or velocity bcs.

ROUTINE: gen_isopar_grid

- isopar_grid generates real data via isoparametric interpolation

ROUTINE: gen_shape_grid

- gen_shape_grid finds shape function for isoparametric generation. It appears to JWP to use bilinear and trilinear interpolation.

ROUTINE: dot_sh

- dot_sh performs the dot product of the vector of shape functions (may be gradients of shape functions) with the vector "variable"

FILE STIFF.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains modules which construct the finite element stiffness matrix and the right-hand-side "force" vector.

§ form_stiff
§ form_rhs
§ form_bc
§ form_slip
§ adjust_bc
§ lame_form
§ force_form
§ surf_form
§ shape
§ dotsh
§ adfldp
§ p_shape
§ form_smooth - disabled -
§ addsmooth - disabled -
§ apply_smooth - disabled -

ROUTINE: form_stiff

- form_stiff computes the element-wise stiffness array for the elt group. "code" indicates if this is for an elastic problem or a VE step.

ROUTINE: form_rhs

- form_rhs computes the right-hand-side vector for the FE problem. "code" indicates if this is for the elastic problem or a VE step.

ROUTINE: form_bc

- form_bc computes the boundary condition terms for stiffness and rhs due to imposed conditions.

ROUTINE: form_slip

- form_slip computes the split-nodes fault offsets and their influence on the finite element stiffness and RHS.

ROUTINE: adjust_bc

- adjust_bc computes the necessary terms that modify the right-hand-side due to imposed displacements.

ROUTINE: lame_form

- lame_form computes the volumetric stiffness term for one element, that is based on the constitutive volume constants. Results are stored in the element stiffness array.

ROUTINE: force_form

- force_form computes the "dumb" gravity contribution to the element right-hand side term for the elastic case, and the strain-rate term for the VE case.

ROUTINE: surf_form

- surf_form computes the surface traction forcing term for the right-hand-side in the finite element problem.

ROUTINE: shape

- shape computes the shape functions and gradients for a single element.

ROUTINE: dotsh

- dotsh computes the dot product of a set of shape-functions (or gradients) with a vector of values, hence computing an interpolated coordinate or function. If st_flag, x is assumed to represent displacement, and a correction for split-nodes is applied prior to the dot product.

ROUTINE: adfldp

- adfldp returns to dotsh() the added nodal displacements needed to account for split node fault slip in calculating strain

ROUTINE: p_shape

- p_shape computes the parent-space shape function for common element types.

FILE SOLVER.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains modules for the sparse matrix storage,
substructure reduction or other solvers of the matrix system:

§ solver
§ addstiff
§ addfor
§ estiffprod
§ reorder
§ genrcm
§ rcm
§ degree
§ fnroot
§ rootls
§ colht
§ profile_diag
§ factor
§ full_back
§ pcg_loop
§ converged
§ put_soln

ROUTINE: solver

- solver performs single right-hand-side matrix solution, currently using factorization and back substitution on single processor OR using PCG iteration.

ROUTINE: addstiff

- addstiff adds element stiffness to global profile array, or to element storage if using PCG solver

ROUTINE: addfor

- addfor assembles the global r.h.s. vector.

ROUTINE: estiffprod

- calculates the product of a given vector with stiffness in element storage t = A * d

ROUTINE: reorder

- reorder uses ien information to build adjacency information and call permutation optimizing routines for minimizing matrix profile.

ROUTINE: genrcm

- genrcm computes the reverse cuthill mckee ordering for a general adjacency graph such as a sparse finite element matrix.

- Original FORTRAN comments: (transcribed into C by JWP from p_reorder.f of PHOEBUS3_T3D):

GENRCM (GENERAL REVERSE CUTHILL MCKEE)

PURPOSE:
GENRCM FINDS THE REVERSE CUTHILL-MCKEE ORDERING FOR A GENERAL GRAPH. FOR EACH CONNECTED COMPONENT IN THE GRAPH, GENRCM OBTAINS THE ORDERING BY CALLING THE SUBROUTINE RCM.

INPUT PARAMETERS:
NEQNS - NUMBER OF EQUATIONS
(XADJ, ADJNCY) - ARRAY PAIR CONTAINING THE ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX.

OUTPUT PARAMETER:
PERM - VECTOR THAT CONTAINS THE RCM ORDERING.

WORKING PARAMETERS:
MASK - IS USED TO MARK VARIABLES THAT HAVE BEEN NUMBERED DURING THE ORDERING PROCESS. IT IS INITIALIZED TO 1, AND SET TO ZERO AS EACH NODE IS NUMBERED.
XLS - THE INDEX VECTOR FOR A LEVEL STRUCTURE. THE LEVEL STRUCTURE IS STORED IN THE CURRENTLY UNUSED SPACES IN THE PERMUTATION VECTOR PERM.

PROGRAM SUBROUTINES:
FNROOT, RCM.

ROUTINE: rcm

- rcm does connectivity analysis (jwp interpretation; see below).

- Original comments from Fortran code:

RCM (REVERSE CUTHILL-MCKEE ORDERING)

PURPOSE:
RCM NUMBERS A CONNECTED COMPONENT SPECIFIED BY MASK AND ROOT, USING THE RCM ALGORITHM. THE NUMBERING IS TO BE STARTED AT THE NODE ROOT.

INPUT PARAMETERS:
ROOT - IS THE NODE THAT DEFINES THE CONNECTED COMPONENT AND IT IS USED AS THE STARTING NODE FOR THE RCM ORDERING.
(XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR THE GRAPH.

UPDATED PARAMETERS:
MASK - ONLY THOSE NODES WITH NONZERO INPUT MASK VALUES ARE CONSIDERED BY THE ROUTINE. THE NODES NUMBERED BY RCM WILL HAVE THEIR MASK VALUES SET TO ZERO.

OUTPUT PARAMETERS:
PERM - WILL CONTAIN THE RCM ORDERING.
CCSIZE - IS THE SIZE OF THE CONNECTED COMPONENT THAT HAS BEEN NUMBERED BY RCM.

WORKING PARAMETER:
DEG - IS A TEMPORARY VECTOR USED TO HOLD THE DEGREE OF THE NODES IN THE SECTION GRAPH SPECIFIED BY MASK AND ROOT.

PROGRAM SUBROUTINES:
DEGREE

ROUTINE: degree

- degree computes the graph steps to each node in the subgraph.

- Original Comments from Fortran code:

DEGREE (DEGREE IN MASKED COMPONENT)

PURPOSE:
THIS ROUTINE COMPUTES THE DEGREES OF THE NODES IN THE CONNECTED COMPONENT SPECIFIED BY MASK AND ROOT. NODES FOR WHICH MASK IS ZERO ARE IGNORED.

INPUT PARAMETERS:
ROOT - IS THE INPUT NODE THAT DEFINES THE COMPONENT.
(XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR.
MASK - SPECIFIES A SECTION SUBGRAPH.

OUTPUT PARAMETERS:
DEG - ARRAY CONTAINING THE DEGREES OF THE NODES IN THE COMPONENT.
CCSIZE-SIZE OF THE COMPONENT SPECIFED BY MASK AND ROOT.

WORKING PARAMETER:
LS - A TEMPORARY VECTOR USED TO STORE THE NODES OF THE COMPONENT LEVEL BY LEVEL.

ROUTINE: fnroot

- fnroot finds the pseudo-peripheral node for a given subgraph.

- Original comments from Fortran:

FNROOT (FIND PSEUDO-PERIPHERAL NODE)

PURPOSE:
FNROOT IMPLEMENTS A MODIFIED VERSION OF THE SCHEME BY GIBBS, POOLE, AND STOCKMEYER TO FIND PSEUDO-PERIPHERAL NODES. IT DETERMINES SUCH A NODE FOR THE SECTION SUBGRAPH SPECIFIED BY MASK AND ROOT.

INPUT PARAMETERS:
(XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR THE GRAPH.
MASK - SPECIFIES A SECTION SUBGRAPH. NODES FOR WHICH MASK IS ZERO ARE IGNORED BY FNROOT.

UPDATED PARAMETER:
ROOT - ON INPUT, IT (ALONG WITH MASK) DEFINES THE COMPONENT FOR WHICH A PSEUDO-PERIPHERAL NODE IS TO BE FOUND. ON OUTPUT, IT IS THE NODE OBTAINED.

OUTPUT PARAMETERS:
NLVL - IS THE NUMBER OF LEVELS IN THE LEVEL STRUCTURE ROOTED AT THE NODE ROOT.
(XLS,LS) - THE LEVEL STRUCTURE ARRAY PAIR CONTAINING THE LEVEL STRUCTURE FOUND.

PROGRAM SUBROUTINES:
ROOTLS

ROUTINE: rootls

- rootls generates the level structure corresponding to "root".

- Original Fortran comments:

ROOTLS (ROOTED LEVEL STRUCTURE)

PURPOSE:
ROOTLS GENERATES THE LEVEL STRUCTURE ROOTED AT THE INPUT NODE CALLED ROOT. ONLY THOSE NODES FOR WHICH MASK IS NONZERO WILL BE CONSIDERED.

INPUT PARAMETERS:
ROOT - THE NODE AT WHICH THE LEVEL STRUCTURE IS TO BE ROOTED.
(XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR THE GIVEN GRAPH.
MASK - IS USED TO SPECIFY A SECTION SUBGRAPH. NODES WITH MASK(I)=0 ARE IGNORED.

OUTPUT PARAMETERS -
NLVL - IS THE NUMBER OF LEVELS IN THE LEVEL STRUCTURE.
(XLS, LS) - ARRAY PAIR FOR THE ROOTED LEVEL STRUCTURE.

ROUTINE: colht

- colht computes column heights of global array.

ROUTINE: profile_diag

- profile diag computes the diagonal addresses.

ROUTINE: factor

- factor performs profile-based factorization: ta=(u) * d * u (crout)

ROUTINE: full_back

- full_back performs all three steps of backsubstitution.

ROUTINE: pcg_loop

- pcg_loop performs the preconditioned conjugate gradient iteration

ROUTINE: converged

- function converged() monitors convergence of CG solver from magnitude and history of the residual norm

ROUTINE: put_soln

- put_soln transfers solution to nodal storage.

FILE STRAIN.C

This file contains modules which perform calculations
related to element stresses and strains:

§ form_stress
§ form_beta
§ form_dbar

ROUTINE: form_stress

- form_stress computes the stress or time-step stress increment based on the FE solution.

ROUTINE: form_beta

- form_beta computes beta, the viscoplastic strain rate for this element and time step.

ROUTINE: form_dbar

- form_dbar computes dbar, the VE single-step constitutive matrix:

  dbar === $(S + \alpha\ \Delta t\ \beta')^{-1}$,

  but we compute the factored form, not the explicit inverse.

FILE UTILITY.C

GeoFEST version 4.2

U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged

This file contains miscellaneous utility routines used throughout the finite element program:

§ move_real
§ clear_real
§ dot_real
§ vadd
§ vouter

ROUTINE: move_real

- move_real copies data to a new location.

ROUTINE: clear_real

- clear_real nulls a data area.

ROUTINE: dot_real

- dot_real forms dot product of two vectors.

ROUTINE: vadd

- calculates the linear combination of vectors, dest = v1 + mult*v2

ROUTINE: vouter

- calculates the outer product of two vectors; actually a misnomer -- it's really the element-by-element product

APPENDIX B.

References used for this guide:

[1] Thomas J. R. Hughes and Robert Taylor, “Unconditionally Stable Algorithms For Quasi-Static Elasto-Plastic Finite Element Analysis.” Computers & Structures, Vol. 8, pp. 169-173, 1978.

[2] Thomas J. R. Hughes, “The Finite Element Method: Linear Static and Dynamic Finite Element Analysis.” Dover, Publication, INC., Mineola, New York, 2000.

[3] H. J. Melosh and Raefsky, “A Simple and Efficient Method for Introducing Faults into Finite Element Computations.” Bulletin of the Seismological Society of America, Vol. 71, No. 5, October, 1981.