FaultMod Finite Element Code

Michael Barall
Invisible Software, Inc.

March 6, 2008

This is a brief description of the FaultMod finite element code, and its application to the spontaneous rupture code validation benchmarks.

Overview

FaultMod is a finite element code developed by Michael Barall, of Invisible Software, Inc., working under contract to the U.S. Geological Survey Earthquake Hazards Team. FaultMod is designed specifically for constructing physics-based models of fault systems which involve complex 3D geometry and 3D variation of material properties.

FaultMod is in the public domain.

Finite Element Core

The finite element core is the heart of the program. Its purpose is to:

- Maintain the sets of elements, faces, and nodes that subdivide the computational domain.
- Construct a set of simultaneous equations that provide force balance at each node, subject to a set of constraints.
- Solve the equations.
- Advance the state of the system to the next time step.

FaultMod currently supports eight different element shapes: hexahedra (cubes), tetrahedra, prisms, pyramids, and four transitional shapes used for special purposes. Each shape is available in both linear and quadratic forms. The code has a well-defined interface for adding more element shapes.

Integration over elements and faces is done using a set of Gauss points and weights. FaultMod always uses full integration, that is, there are enough Gauss points so that element stiffness matrices are non-singular. So, there should be no zero-energy oscillation modes.

FaultMod has two modes of operation: static and dynamic. Static mode is suitable for problems such as coseismic deformation, stress distribution, fault creep, and viscoelastic relaxation. Dynamic mode is suitable for problems such as wave propagation and dynamic rupture.

In dynamic mode, time stepping is performed using the Newmark algorithm. For the TPVx problems, the Newmark parameters are adjusted to provide algorithmic damping.

Mathematical Basis

Mathematically, FaultMod is based on a tensorized version of the finite element method. *Tensorized* means that all physical variables are treated as tensor fields of an appropriate type. For example, displacements are treated as a contravariant vector field, while strains are treated as a covariant rank-2 tensor field.

The fundamental equations of the finite element method were rewritten in a general tensor form, including the use of a metric tensor to convert between covariant and contravariant tensors. Then, the tensor equations were implemented directly into the computer code. The entire finite element core is implemented as tensor code. This has three advantages: it provides a standard mathematical language for describing and implementing the code; it ensures that the code is always dealing with physically meaningful quantities; and it ensures that physical laws are encoded in a proper form.

Within the finite element core, all internal calculations are done in tensor form. Spatial derivatives of physical quantities are calculated as covariant derivatives, including the use of Christoffel symbols. Volume and surface intergrals are calculated using the Levi-Civita tensor. The metric tensor is used whenever needed to perform contractions.

Implementation of Faults

Two coincident nodes are placed at each node location that lies on a fault. They are called a *common node* and a *differential node*.

The common node represents motion common to both sides of the fault. It is the node that would be at the given location, if there were no fault.

The differential node represents motion of one side of the fault, relative to the other side of the fault. Thus, on one side of the fault, the effective displacement is equal to the displacement of the common node. On the other side of the fault, the effective displacement is equal to the displacement of the common node plus the displacement of the differential node.

The advantage of differential nodes is that fault behavior variables become primary system variables. Fault slip, slip rate, and slip acceleration are equal to the displacement, velocity, and acceleration of the differential node. If the nodal force acting on the common node is zero, then the fault traction force is equal to the nodal force acting on the differential node.

In dynamic mode, it is possible to compute the fault traction force even when the common node has a nonzero nodal force. This is done by transforming to an accelerated frame of reference. By choosing the acceleration properly, it is possible to find an accelerated frame of reference in

which the force acting on the common node is zero. Then, the force acting on the differential node, in this special frame of reference, equals the traction force on the fault.

Constraint Mechanism and Constrained Solvers

It is generally necessary to constrain the motion of differential nodes, *e.g.*, to allow motion only tangent to the fault. Often it is also necessary to constrain the motion of boundary nodes.

FaultMod uses projection operators as a mathematical formulation of constraints. Each node is assigned a 3-dimensional projection operator, and the motion of the node is constrained to lie within the range of the projection operator. By appropriate choice of projection operator, the node can be permitted to move freely, or constrained to move within a specified plane or line, or not move at all. The projection operators for all the individual nodes are assembled into a single global projection operator, which represents all the constraints in the entire system.

The finite-element force-balance equations, and the Newmark time-stepping equations, have both been modified to account for the existence of constraints.

FaultMod includes a linear solver that is able to solve a system of equations together with a set of constraints defined by a projection operator. This linear solver is a modified preconditioned conjugate gradient method, created especially for FaultMod. The solver understands constraints intrinsically, and does not require any auxiliary mechanism (such as Lagrange multipliers or penalty functions).

FaultMod also includes a nonlinear solver that is able to solve a system of nonlinear equations together with a set of constraints. It uses Newton's method, and invokes the linear solver on each iteration.

Material Properties

Material properties can vary arbitrarily in space. In the code, each Gauss point has an associated set of material parameters and material state variables.

FaultMod currently supports the following types of materials: linear and non-linear elastic, linear viscous, linear and non-linear viscoelastic, and von Mises plastic. These basic types of materials can be combined to form more complicated materials, such as a viscoelastic material with multiple relaxation times. The code has a well-defined interface for adding new material types, but only isotropic materials are permitted.

Simple distributions of material properties, such as those used in the TPVx problems, can be achieved using FaultMod’s built-in commands. For more complicated distributions, such as real geologic models, FaultMod can import material properties from EarthVision.

Calculations can be performed with or without gravity.

The code offers two options for handling element mass: the mass can be distributed continuously throughout the element (so-called *consistent mass*), or it can be concentrated at the corners of the element (so-called *lumped mass*). This option affect not only gravity, but also the inertial forces that arise during dynamic operation.

In the TPVx benchmark problems, we always use linear elastic materials, as described in the benchmark problem descriptions. We also introduce a small amount of linear viscosity near the fault surface, to help damp out spurious oscillations [method of Day and Dalguer]. Masses are always lumped [recommendation of Andrews].

Fault Behavior Models

Fault behavior models can vary arbitrarily on the fault surface. Each differential node has an associated set of model parameters and state variables.

FaultMod currently supports the following fault behavior models: linear slip weakening (dynamic mode only); rate-state friction (dynamic mode only); coulomb friction (static mode only); freely slipping; locked; and specified kinematic slip. The code has a well-defined interface for adding new fault behavior models.

Boundary Conditions

Boundary conditions can vary arbitrarily on the border of the mesh. Each boundary node has an associated set of boundary condition parameters and state variables.

FaultMod currently supports the following boundary conditions: free motion, no motion, constrained motion, motion with specified traction forces, kinematically prescribed motion, and energy-absorbing boundary conditions. Certain combinations are possible, *e.g.*, kinematically prescribed horizontal motion combined with free vertical motion. The code has a well-defined interface for adding new boundary conditions.

For the TPVx benchmark problems, we use free motion on the top border, and energy-absorbing boundary conditions on the side and bottom borders of the mesh. (For full-space problems, we use energy-absorbing boundary conditions on all borders of the mesh.)

Mesh Generation

FaultMod has a built-in mesh generator. The mesh generator can handle straight faults, curved faults, dipping faults, and faults with 3D geometry. To some extent, the mesh generator can also handle branched faults and multiple faults. Surface topography is also supported.

However, the mesh generator is not a fully general 3D mesh generator. It is best described as a 2D mesh generator with a set of techniques for extending the mesh into the third dimension.

An interesting fact about the mesh generator is that some of its algorithms are adapted from computer graphics. For example, 3D fault geometry and surface topography are produced by a "morphing" process.

Meshes generated by FaultMod consist of hexahedra (cubes), with, in some cases, prisms added surrounding the fault. The mesh generator can output an image of the mesh to a GIF file so you can see what it did.

Output Files

When you set up FaultMod, you specify which types of output files should be produced.

FaultMod currently supports the following types of output files: time-series files (for synthetic seismograms); rupture arrival time files (for rupture contour plots); visualization files (which can be exported to VTK/ParaView for full 3D visualization); and tabular files (for testing purposes). It is possible to generate multiple output files simultaneously. The code has a well-defined interface for adding new output file formats.

Script Processor

FaultMod can be run in two ways: *interactive mode* and *script-driven mode*. When run interactively, FaultMod gives you a prompt and you can type in commands, which are immediately executed. In script-driven mode, FaultMod reads a text file and executes all the commands therein.

Because FaultMod is designed to handle complicated problems, it can take many commands to set up and run a problem. Therefore, it is generally best to use script files.

FaultMod has its own script language. The script language includes:

- Script variables and arrays.
- Integer, floating-point, boolean, and string variables.
- Mathematical function library.
- Branches, loops, and subroutines.
- File I/O.

Because the script language is built-in, you can use the exact same script files on any system where FaultMod runs, regardless of operating system or command shell.

The script language is intended to be powerful enough so that you can set up problems entirely within the script language, without having to write C++ code for each problem.

Self-Test Capabilities

FaultMod has a built-in patch test function. The patch test is a standard set of tests used to validate FEM codes. In FaultMod, the patch test is a series of hundreds of individual tests. They validate the implementation of each type of element, as well as the procedures used by the mesh generator to assemble elements into a complete mesh. They also serve to test the linear and nonlinear solvers, volume and surface integrators, and finite-element engine.

A built-in Okada-type solution calculator can be used to compare certain simple FEM calculations to analytic solutions.

FaultMod can display technical details about the local system and C++ implementation. This is useful when porting FaultMod to a new compiler or system.

Diagnostics scattered throughout the program continually watch for anomalous conditions while FaultMod operates. If an anomaly is detected, FaultMod interrupts the job in progress and notifies the user. In practice, this rarely happens.

Dynamic Rupture Calculation Cycle

FaultMod performs dynamic rupture simulations by executing the following steps:

1. Take a trial step, assuming acceleration is unchanged from the previous time step.
2. Calculate nodal forces at the end of the trial step, and convert them into traction forces on the fault.
3. Solve the fault constitutive equation, obtaining fault slip, slip rate, and acceleration.
4. Apply the calculated fault slip, slip rate, and acceleration to the finite-element model as a kinematic boundary condition.
5. Solve the finite-element force balance equations, obtaining displacement, velocity, acceleration, and nodal forces throughout the mesh.
6. Update internal state variables of the fault behavior model.
7. Go to step 1.

Notice that the fault constitutive equations are not solved simultaneously with the finite-element force balance equations. This permits the use of fault constitutive equations with poor numerical properties, such as rate-state equations.

Also note that the finite element model is actually running a kinematic simulation. The role of the fault behavior model is to calculate the kinematic boundary conditions on the fly.