Virtual California User Manual

Version 1.0.0

January 30, 2014

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 Virtual California, the QuakeLib library, and their capabilities then proceeds to the details of implementation. Part III provides appendices and references for input and output files and parameters.

The style of this publication is based on the Apple Publications Style Guide (`developer.apple.com/library/mac/documentation/UserExperience/Conceptual/APStyleGuide/APSG_2009.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 Mantle Convection Mailing List (`cig-mc@geodynamics.org`).

Who Will Use This Document

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

The manual was written for the usage of Virtual California on a variety of different platforms. Virtual California has run on shared memory computers (Sun, Hewlett-Packard, SGI, and IBM), commercial distributed memory machines (Intel and Cray/SGI), clusters (including machines on NSF XSEDE), Linux PCs and Mac OS X desktops.

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, and probabilistic seismic hazard analysis. The underlying C++ code for the Greens function calculations, stress evolution, simulation framework, and the QuakeLib library and associated Python bindings and testing framework were donated to CIG in December of 2012. A number of individuals have contributed a significant portion of their careers toward the development of Virtual California. 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 Virtual California development team asks that you cite both of the following:

- Heien, E.M., Sachs, M.K., "Understanding Long-Term Earthquake Behavior through Simulation," Computing in Science and Engineering, pp. 10-20, Sept.-Oct., 2012

- Sachs, M.K., Heien, E.M., Turcotte, D.L., Yikilmaz, M.B., Rundle, J.B., Kellogg, L.H. "Virtual California Earthquake Simulator" Seismological Research Letters, November/December 2012, v. 83, p. 973-978, doi:10.1785/0220120052

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

Support for this work and researchers was provided by multiple sources. This work was supported by the National Aeronautics and Space Administration (NASA) grant number NNX08AF69G, JPL subcontract number 1291967, and NASA Earth and Space Science Fellowship number NNX11AL92H. Support was also given by the Southern California Earthquake Center (SCEC). SCEC is funded by the National Science Foundation Cooperative Agreement EAR-0529922 and U.S. Geological Survey (USGS) Cooperative Agreement 07HQAG0008. This work also used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. 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.

Conventions

Throughout this documentation, any mention of "username" is meant to indicate the user, meaning you should substitute your account name in its place.

Part II

Chapters

Chapter 1

Introduction

Virtual California is a boundary element code designed to investigate long term fault system behavior and interactions between faults through stress transfer. It is released under the MIT Public License (see Appendix D). The core code is written in C++ and can be run on a variety of parallel processing computers, including shared and distributed memory platforms. To allow increased functionality including development of other simulators, analysis scripts, and visualization tools, some key components of Virtual California have been placed in the QuakeLib library which can be called from C/C++ programs or Python scripts.

1.1 About Virtual California

Virtual California is a boundary element code that performs simulations of fault systems based on stress interactions between fault elements to understand long term statistical behavior. It performs these simulations using a model of faults embedded in a homogeneous elastic half space with arbitrary dips and rakes.

The code performs calculation assuming linear stress increase in the long term based on element-element interaction calculations governed by Okada's implementation of Green's functions. During the rupture (earthquake) phase elements may fail and release stress based on a combination of static and dynamic stress thresholds. The behavior of the system is determined by interactions between elements from the Green's function and the stress release from elements during events. More detail about the equations and physics of the simulation is described in Section 2.

1.2 History

Virtual California (abbreviated VC) started as a limited simulation model for distributed seismicity on the San Andreas and adjacent faults in southern California, developed by Rundle [5] in Fortran. This model included stress accumulation, release and interactions between faults to investigate earthquake dynamics in southern California. The model was updated in the early to mid-2000s [6, 4, 7] to include major strike-slip faults in California and was named Virtual California. This model and simulation was used to examine recurrence time statistics on California faults by Yakovlev [10], where it was concluded that the return times on a fault is well approximated by a Weibull distribution.

In 2010 Virtual California was rewritten by Eric Heien in C++ to have a more modular simulation framework and add support for multiprocessor simulation using OpenMP and/or MPI. The fault model in Virtual California was also more cleanly separated from the fundamental stress calculation to allow simulation of other fault systems, including the Nankai trough in Japan [11]. Additional features were also added, including a branching aftershock sequence (BASS) model simulation of aftershocks [12], improved stress Green's function calculations, more sophisticated rupture propagation, and support of parallel HDF5 output. Virtual California was also used in an effort by the Southern California Earthquake Center to unify and compare the results from several earthquake simulators [8].

In 2011 and 2012, core components of Virtual California were separated into the QuakeLib library and used to create analysis and visualization tools. Improvements to simulation performance were also developed,

including speculative execution for rupture propagation and a Barnes-Hut style approximation scheme for the Green's functions.

1.3 About QuakeLib

QuakeLib is a C++ library containing key mathematics, geophysics and I/O functionality related to earthquake simulation and result analysis. QuakeLib is currently distributed with and is used by Virtual California, though it can be compiled and installed by itself on a machine. More specifically, QuakeLib contains 1) functions to read, write and validate fault models and earthquake catalogs in the EqSim format, 2) C++ classes to represent and access these models and catalogs, 3) C++ classes to represent faults and associated fault parameters as well as functionality related to the faults, 4) C++ classes and functions to perform vector mathematics and unit/geographic conversions related to modeling, 5) functions to evaluate stress and displacement fields based on Okada's equations [3] given a rectangular fault or point source.

QuakeLib is also written to support extension to other scripting languages through the use of the Simplified Wrapper and Interface Generator (SWIG) [1]. Currently this supports wrapping the QuakeLib library in a Python interface though additional scripting languages can be easily added. The scripting extension allows researchers to write analysis and visualization scripts based on the same equations and data formats as the simulation. The Python interface is also used as the basis for a testing framework to ensure that any changes made to the code do not affect the scientific results and that computations on different platforms yield the same results within a specified tolerance.

Chapter 2

VC Components and Governing Equations

There are three major components that make up Virtual California: a fault model, a set of quasi-static interactions (Green's functions), and an event model. In spite of the name, the only component of Virtual California that is specific to California is the fault model. This model can be changed to any physically realistic model and still correctly work with the simulation physics and event model.

2.1 Fault Model

The model that is currently in use is based on the EQSIM format developed by the Working Group on California Earthquake Probabilities [2]. The model includes 181 fault sections roughly corresponding to known faults in California, with some faults modeled by multiple sections. Each fault section is meshed into square elements that are roughly 3 km $\times$ 3 km, for a total of 14,474 elements. In the present version of our model, the creeping section of the San Andreas fault is removed. This section produces many events, slowing the simu- lation down considerably. Each element in the model is given a constant back-slip velocity along a fixed rake vector and a failure stress. The rake vector always lies in the plane of the element; and, the angle of the vector relative to the surface of the Earth, along with the rate of slip, are quantities that are taken from the UCERF2 model. The failure stresses, which are also required for the model, are derived from paleoseismic event recurrence times.

2.1.1 Model Parameters and Initial Conditions

Virtual California uses established scaling laws to determine certain model parameters and initial conditions. Past simulations required that the user specify the stress parameters on each element then run the simulation based on this. However, the parameters to produce realistic results depend strongly on the model fault geometry, size of fault elements and friction properties. Furthermore these parameters can be easily modified to produce arbitrary fault behavior. Rather than require the user to specify these parameters, they are internally calculated by VC to match established physical laws and empirical observations.

Wells and Coppersmith (1994) showed that ?????

For each fault VC calculates the expected moment magnitude from an earthquake that ruptures all elements on the fault. This is based on the equation:

$$M = a + b * log(RA) \tag{2.1}$$

where RA is the total area of all elements on the fault, and a and b are based on empirical observation. By default $a = 4.07$ and $b = 0.98$. These values do not vary significantly for strike-slip vs. normal vs. reverse faults, and they are measured for a wide range of both fault lengths (from a 2-5000 kilometers) and earthquake magnitudes (4.8 to 7.9).

Given this moment magnitude, VC determines the maximum slip on the elements that would generate such an event:

$$\Delta s = \frac{10^{(3/2)(M+10.7)}}{lame * RA} \tag{2.2}$$

TODO: is this slip-moment relationship correct?

VC also calculates the lithostatic normal force acting on each element based on the depth using the equation:

$$\sigma_n = \rho g d \tag{2.3}$$

where $\rho = 5515 \frac{kg}{m^3}$, $g = 9.81 \frac{m}{s^2}$ and d is the average depth of the element.

Since an element moves at a velocity s_l and

2.1.2 Setting Fault Parameters and Building a Fault Model

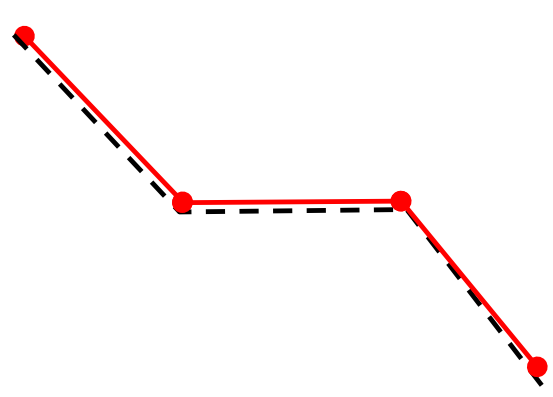
VC treats a system of faults as multiple planar elements embedded in a flat homogeneous halfspace. To run Virtual California the first step is to define a fault system using a set of traces. Each trace describes the points along a given fault closest to the surface as well as fault characteristics at each trace point, such as long term velocity, dip angle, rake angle, depth, etc. Details about the trace file format are shown in Appendix B.2.

In this example we look at the earthquake cycle and rupture mechanics on a single 12 km square fault. For simplicity, the trace of this fault runs eastward for 12km starting from latitude/longitude (0,0) and ending at latitude/longitude . The definition of this fault is in the file examples/fault_traces/single_fault_trace.txt and is also shown below.

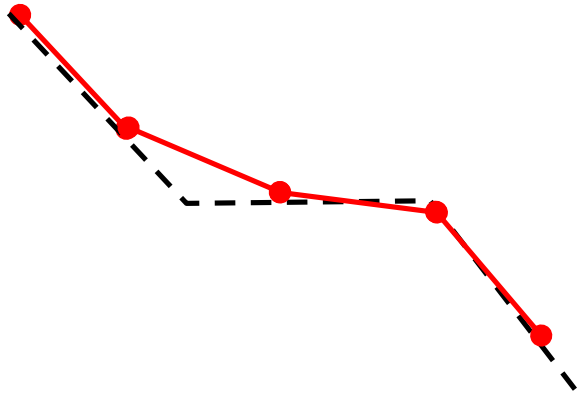
```
# fault_id: ID number of the parent fault of this section
# num_points: Number of trace points comprising this section
# section_name: Name of the section
0 2 One_Element_Example
# latitude: Latitude of trace point
# longitude: Longitude of trace point
# altitude: Altitude of trace point (meters)
# depth_along_dip: Depth along dip (meters)
# slip_rate: Slip rate at trace point (centimeters/year)
# aseismic: Fraction of slip that is aseismic at point
# rake: Fault rake at trace point (degrees)
# dip: Fault dip at trace point (degrees)
# lame_mu: Lame's mu parameter at trace point (Pascals)
# lame_lambda: Lame's lambda parameter at trace point (Pascals)
0 0 0 12000 1 0 180 90 3e+10 3.2e+10
0 0.1078 0 12000 1 0 180 90 3e+10 3.2e+10
```

The first non-comment line in this file gives the fault ID, number of trace points, and fault name. In this example, the fault is named "One_Fault_Example". The remaining non-comment lines give the fault characteristics at each trace point. This file defines two trace points, the first at latitude/longitude (0,0) and the second at (0, 0.1078), with both at altitude 0. At each point the fault extends 12,000 meters down and has a slip rate of 1 cm/year with 0 aseismicity. The fault is right lateral strike slip (rake of 180°, dip of 90°). The Lamé parameters of $\mu = 3e10$ and $\lambda = 3.2e10$ indicate the material properties of the fault.

Given this fault definition we can create a mesh which fits within the fault dimensions. Each fault is meshed by specifying the fault trace file in the format described above, and a fault element size. Currently all fault elements in Virtual California are square though future versions will allow triangulare elements. Furthermore, all elements of a single fault are meshed at the same resolution. This means that if the meshing resolution is not a perfect multiple of the trace length or depth at a given point, the meshed elements will not completely cover the trace.

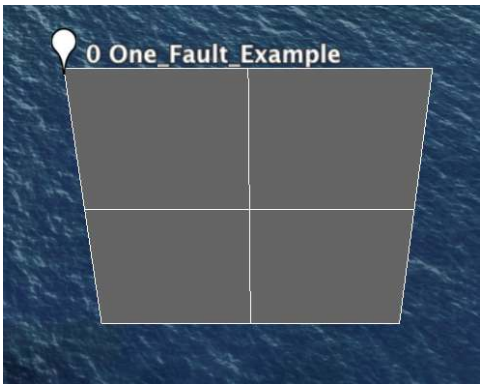


(a) Using an element size that exactly aligns with the trace, the generated elements will exactly follow the trace.

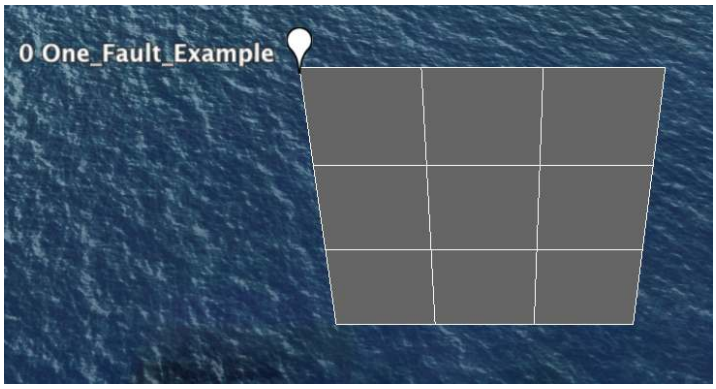


(b) A slightly smaller element size results in a mesh that does not exactly follow the trace.

Figure 2.1: An example of how element size will affect the tracking of the mesh along a fault trace.



(a) Example 12km x 12km fault meshed at 6km resolution giving 2 x 2 = 4 total elements.



(b) Example 12km x 12km fault meshed at 4km element resolution giving 3 x 3 = 9 total elements.

Figure 2.2: A demonstration of meshing with different resolutions.

Figure 2.1 shows this for an example fault trace consisting of 4 trace points, where the fault trace is represented by a dashed line and the meshed elements represented by red segments. Figure 2.1a shows the meshed model on this fault with elements that match the lengths between the trace points. Because the distance between trace points is exactly a multiple of the element size, the meshed elements exactly cover the fault trace. Figure 2.1b shows the same fault trace with smaller meshed elements. The first element follows the trace but the second element deviates in order to fit the meshed element along the trace. In practice, this discrepancy is usually not a big issue because fault traces are not significantly non-linear. Also, as elements become smaller relative to the distance between trace points this discrepancy becomes smaller.

When creating a meshed element along a fault trace it is necessary to assign characteristics to the element such as slip rate, aseismic slip, rake, dip, and Lamé parameters. These are determined by linear interpolation of the fault trace values at the midpoint of the meshed element.

The use of linear interpolation for values between fault trace points also means that if the meshed elements are larger than the distance between fault trace points and there is significant variation between trace point characteristics, then this variation may be lost during the meshing process. In general it is recommended to use an element size smaller than the smallest distance between fault trace points unless there are memory or computing constraints. In the event that the meshing process skips a trace point because of overly large element size, a warning will be output. Appropriate element size is further discussed in Section 4.3.

(Describe how to run meshing program)

The meshing program and related parameters are described in Section XXX.

Figure 2.2 shows the result of meshing the One_Fault_Example trace with different element resolutions. Figures 2.2a and 2.2b show the KML output of the program in Google Earth (citation). Since the fault is defined to start at latitude/longitude (0,0) it will appear in the middle of the Atlantic ocean. In output KML files the depth is reversed so faults are visible above the surface. When performing simulations with realistic fault systems it is better to use actual fault latitude/longitude in the traces to help visualize the simulation results.

2.2 Element Stress Interactions

Unlike actual fault systems where the fault geometry is dynamic over long time periods, Virtual California simplifies calculations assuming a geometrically static fault system. In this way Virtual California is intended to explore seismicity in fault systems as they appear today rather than attempting to model their long term evolution. Back slip is used to model the effects of stress buildup and release along elements approximating the fault plane. In a back slip model, the equilibrium and initial positions of an element are the same, thus when an element fails it moves towards the original position and the fault system geometry remains static.

2.2.1 Green's Functions

Interactions between fault elements depend on the relative position and orientation of each element, and are calculated using stress Green's functions at the start of the simulation. The change in stress at a location x due to movement of all elements is given by [7]:

$$\sigma_{ij}(x,t) = \int dx_k^{'} T_{ij}^{kl}(x-x')s_l(x',t) \tag{2.4}$$

where $s_l(x',t)$ is the three-dimensional slip density of element l, $T_{ij}^{kl}(x-x')$ is the Green's function tensor, and l goes over all elements. In Virtual California this field is evaluated only at the center of elements and slip is assumed to be uniform across the surface of an element and along the element rake angle defined in the model. Under these conditions equation 2.4 simplifies to:

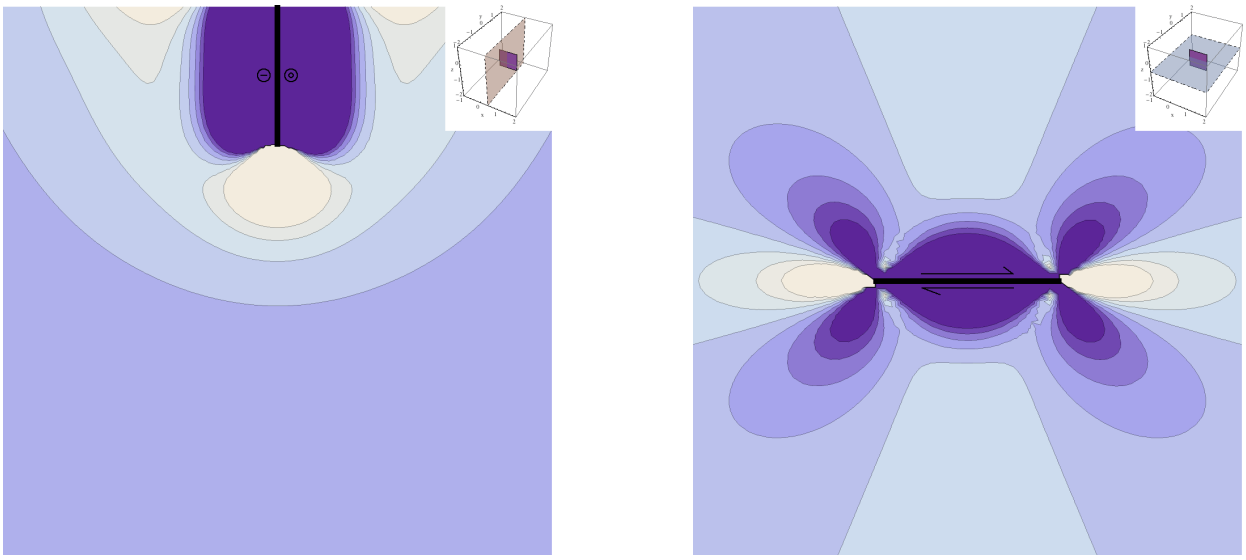
$$\sigma_{ij}^{A}(t) = \sum T_{ij}^{AB} s_B(t) \tag{2.5}$$

where B runs over all elements. Finally, since Virtual California only uses the shear stress along the element rake vector and normal stress perpendicular to the element, the tensor T_{ij}^{AB} reduces to T_s for shear stresses and T_n for normal stresses. This means the shear and normal stresses on an element in Virtual California are calculated as:

$$\sigma_{s}^{A}(t) = \sum T_{s}^{AB} s_B(t) \tag{2.6}$$

$$\sigma_{n}^{A}(t) = \sum T_{n}^{AB} s_B(t) \tag{2.7}$$

Thus, for a fault model with N elements Virtual California requires two $N \times N$ element matrices to represent all interactions. These are also referred to as the Green's function matrices. The actual values for the matrix entries are calculated using Okada's half-space deformation equations [3]. Examples of the output generated by the Virtual California implementation of the Okada Green's functions are shown in Figure 2.3.



(a) Front view

(b) Top view

Figure 2.3: The shear stress field σ_{xy} created by horizontal backslip as viewed from the front and top of an element. The direction of backslip is indicated by the arrows. Tan color indicates $\sigma_{xy} > 0$ and blue indicates $\sigma_{xy} < 0$.

2.2.2 Event Transition Time

Virtual California uses a combined static-dynamic friction law to calculate element failures. This law is based on the Coulomb failure function (CFF):

$$CFF^A(t) = \sigma_s^A(t) - \mu_s^A \sigma_n^A(t) \tag{2.8}$$

where μ_s^A is the static coefficient of friction on element A based on model element strengths. During long term stress accumulation, an element is defined to fail at time t_f when $CFF^A(t_f) = 0$, which is referred to as static failure. At this point the simulation changes to the rupture event model described below.

Given the change in stress over time it is relatively straightforward to calculate the time to failure for an element. Since effective long term slip rates during stress accumulation are assumed to be constant the change in CFF over time is governed by the equation:

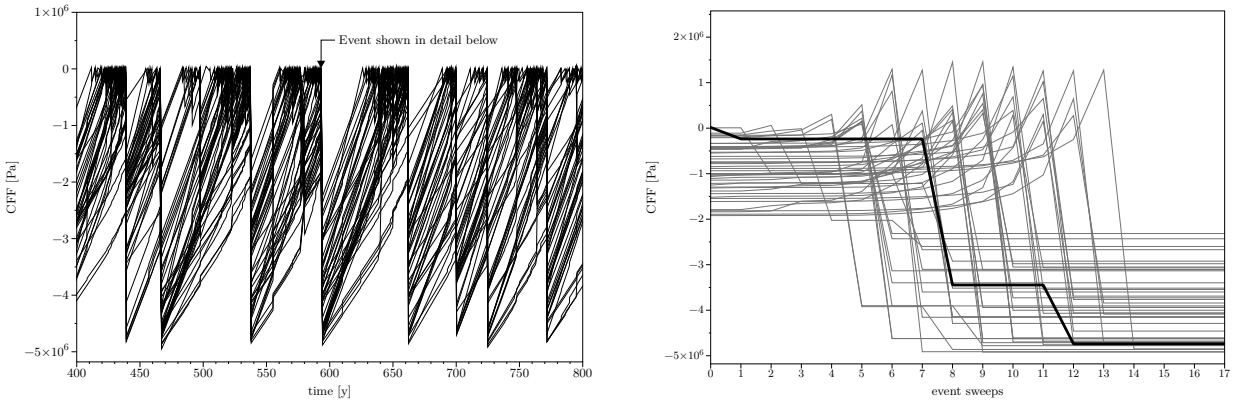
$$\frac{dCFF^A}{dt} = (\sigma_s^A - \mu_s^A \sigma_n^A) + \alpha T_\alpha^A \tag{2.9}$$

where α represents the fraction of fault slip that is aseismic and $T_\alpha^A = \frac{CFF^A T^{AA}}{rec}$ represents the instanteous change in CFF due to aseismic slip. Aseismic slip on an element transmits stress to other elements but not on the element itself. If α is 0, the equation simplifies to:

$$t_f = t_0 + \frac{CFF^A}{(\sigma_s^A - \mu_s^A \sigma_n^A)} \tag{2.10}$$

where t_0 is the starting time of long term stress accumulation in the system. If $\alpha > 0$ then Equation 2.9 can be solved analytically to yield:

$$t_f = -log((\sigma_s^A - \mu_s^A \sigma_n^A) +$$



(a) CFF vs. time for a series of stress buildup and rupture events.

(b) CFF vs. sweeps for the event at t=593

Figure 2.4: Top: The CFF for each of the 48 elements comprising the Parkfield section of the San Andreas fault. Drops in CFF correspond to stress release in events, with larger events consisting of many elements releasing stress. Bottom: detail of rupture sweeps from the event at t=593. The trigger element is shown bold. Note that elements may experience multiple failures, such as the trigger element failing during sweeps 0, 7, and 11.

2.3 Rupture Event Model

During rupture propagation the first element to fail slips back towards the equilibrium position. The amount of slip during the initial failure, Δs, is related to the stress drop defined for the element in the model, $\Delta\sigma$, by [7]:

$$\Delta s = \frac{1}{K_L}\frac{N_{ef}}{S_t}(\Delta\sigma - CFF), \text{if } N_{ef} \leq S_t \tag{2.11}$$

$$\Delta s = \frac{1}{K_L}(\Delta\sigma - CFF), \text{otherwise.} \tag{2.12}$$

where K_L is the element's stiffness or self-stress defined as $K_L = T_s^{AA} - \mu_s^A T_n^{AA}$. The factor $\frac{N_{ef}}{S_t}$ captures the current size of the rupture with N_{ef} representing the number of failed elements on the currently rupturing fault and S_t representing the slip-scaling threshold parameter. This factor prevents small ruptures from excessively slipping.

Once the slip is calculated for one or more elements, a new stress state for the entire system is calculated using Equations 2.6 and 2.7. Additional elements will fail if their CFF = 0 (static failure). To better model rupture propagation dynamic failure is also allowed during rupture events. Dynamic failure allows elements on the same fault and physically nearby to failed elements to themselves fail at a lower stress level than the static failure criterion. This dynamic failure is based on the increase in stress during the rupture event. An element experiences dynamic failure if it is on the same fault as an already failed element and satisfies:

$$\frac{CFF_{init} - CFF_{final}}{CFF_{init}} > \eta \tag{2.13}$$

where η is a user defined dynamic triggering parameter either for the whole system or uniquely defined for each element. This parameter approximates the stress intensity factor at the tip of a propagating rupture.

During rupture propagation, elements that have not completely slipped back to equilibrium may fail again (potentially multiple times) and release more stress. However, elements may not slip away from equilibrium meaning they cannot absorb stress released by other failed elements. This reflects the fact that

failed elements may not heal during a rupture but also may not release all accumulated stress immediately. It is also important to note that an element may not release all accumulated stress during a rupture, due to the slip-scaling threshold (Equations 2.11 and 2.12) and dynamic triggering (Equation 2.13), as well as the stress state of all elements at the start of rupture. For example, if all elements are in a high stress state then a small initial rupture can quickly propagate and become a large stress release event, whereas if most elements are in a low stress state then a large initial rupture may quickly stop propagating and release relatively little stress.

Examples of stress accumulation and release over multiple phases of long term stress accumulation and rupture events are shown in Figure 2.4. The first figure shows how a single element failure leads to a rupture spanning multiple elements and how stress builds up and releases in cycles over time. The second figure shows how elements may fail multiple times during different sweeps in an event, but do not accumulate additional stress after failing.

2.4 Simulation Flow

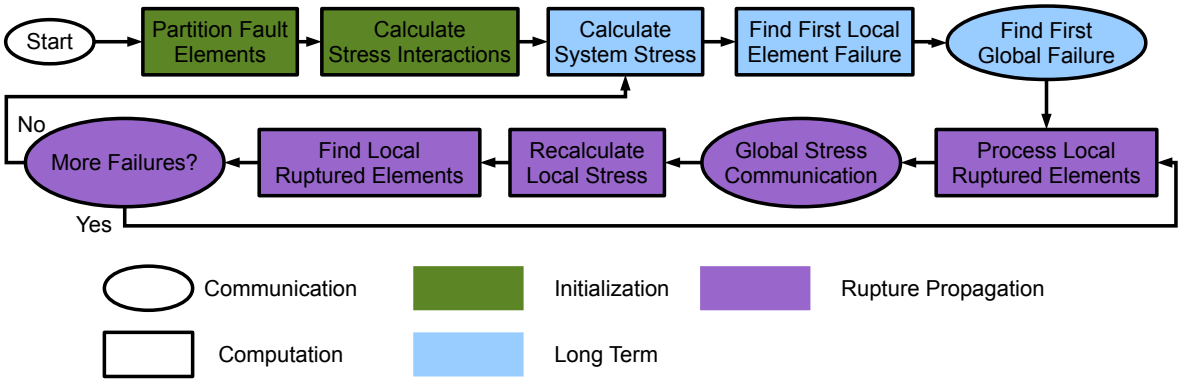


Figure 2.5: Simulation flow of Virtual California.

Figure 2.5 shows the flow of simulation in Virtual California. The simulation begins by reading in a set of faults and converting them to the internal data structures. When running a parallel simulation, these are partitioned over multiple processes to ensure that each process is responsible for roughly an equal number of elements and that elements on the same processor are on the same fault or geographically close to each other. Next, each processor calculates the stress Green's functions on the local elements for all other elements in the model or loads precomputed Green's functions from a file. This comprises the initialization phase of the simulation shown in green in Figure 2.5.

The core of the simulation consists of repeated cycling between two phases until the end of the simulation time. The first phase, shown in blue, calculates the long term stress buildup and time to first element failure in the system based on Equation 2.8. In parallel simulations this time is calculated locally on each process then reduced to a global time to failure.

The second phase is the rupture propagation phase, shown in purple in Figure 2.5. Virtual California uses a cellular automata style approach to modeling rupture propagation. The rupture phase does not involve time domain solutions to differential equations, but rather iterative calculations of stress and element failure to approximate a rupture propagating through the fault system.

Chapter 3

Getting Started and Installation

3.1 Introduction

Virtual California and QuakeLib have been tested on Linux, Mac OS X and several other UNIX based platforms. Virtual California has also been successfully run in parallel on several XSEDE systems and commodity cluster systems. You should have no problems compiling, installing, and running Virtual California on most Unix-like systems. Virtual California is currently only available as a source package that users must compile on their own. The following sections will lead you through the installation process.

3.2 Getting Help

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

3.3 System Requirements

Installation of Virtual California requires the following:

- A C++ compiler
- An MPI library
- OpenMP
- HDF5
- SWIG
- CMAKE

You can install the first two dependencies in one command on most Linux machines. On Debian, Ubuntu or similar distributions, use this command (as `root`):

```
$ apt-get install build-essential python-dev \
          openmpi-dev openmpi-bin
```

On Red Hat, Fedora, CentOS, OpenSuSE or similar distributions, use this command (as `root`):

```
$ yum install make automake gcc gcc-c++ kernel-devel \
      openmpi openmpi-libs openmpi-devel
```

You will need to install a C compiler, MPI library, OpenMP, HDF5, SWIG, and CMAKE (see sections below).

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.

3.3.1 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.

3.3.2 MPI Library

Virtual California requires a library which implements the MPI standard (either version 1 or 2). Several free, open-source implementations of MPI are available.

A popular choice is MPICH (`www-unix.mcs.anl.gov/mpi/mpich`). Other choices include LAM/MPI (`www.lam-mpi.org`) and Open MPI (`www.open-mpi.org`). Installing MPI from source involves walking through the standard GNU build procedure (`configure && make && make install`) while logged in as `root`.

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 (`www.lam-mpi.org`); so if you have Fink (`fink.sourceforge.net`) installed, simply enter the following command from a Terminal window to install LAM/MPI:

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

3.3.2.1 MPI C Compiler Command

Once you have an MPI library installed, make sure its C complier command is on your PATH. Unfortunately, the name of this command varies from one MPI implementation to the next. The Virtual California configuration script searches for the following MPI C command names:

```
mpicc hcc mpcc mpcc_r mpxlc cmpicc
```

3.3.3 Remaining Dependencies

You will need to also install OpenMP, HDF5, SWIG, and CMAKE.

3.4 Downloading and Unpacking Source

To obtain Virtual California, go to the Geodynamics Software Packages web page (`geodynamics.org/cig/software/packages/mc/citcoms`), download the source archive and unpack it using the `tar` command:

```
$ tar xzf vc_xxxxxxxx.tar.gz
```

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

```
$ gunzip -c vc_xxxxxxxx.tar.gz | tar xf -
```

3.5 Installation Procedure

After unpacking the source, use the following procedure to install Virtual California:

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

   ```
   $ cd Virtual_California_xxxxxx
   ```

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

   ```
   $ ./configure
   ```

3. Navigate to the sub-directory called "build"

   ```
   $ cd build
   ```

4. Type `make` to build the package.

   ```
   $ make
   ```

If you are content to run Virtual California from the build directory, then you are done. Upon successful completion, the `make` command creates a script called `vc` in the `xxxxx` subdirectory; this is the script you will use to run Virtual California. You may wish to add the `bin` directory to your `PATH`.

For more details about `configure`, see Section 3.6 below.

3.5.1 Windows

This is untested so use at your own risk!

If using Cygwin:

```
$ ./configure make make install
```

If using Visual Studio:

Follow the directions at the link for running CMake on Windows:(`opentissue.org/wikitissue/index.php/Using_CMake`),

NOTE: Select the "build" folder as the location to build the binaries.

3.5.2 Mac OS X

This is untested so use at your own risk! With the OS X default python:

```
$ ./configure
$ cd build
$ make
$ sudo make install
```

If you have a third party installation of python (ie. from homebrew or macports) Virtual California will build and install, but the python quakelib module may not work. This is because cmake builds against the system python. To fix this do the following:

```
$ ./configure
```

```
    cmake will output the following:
    "-- Found PythonLibs: /usr/lib/libpythonx.x.dylib"
    where "x.x" is a version number (ie. "2.7").
    We need to move this file and create a link to the correct python.

$ sudo mv /usr/lib/libpythonx.x.dylib /usr/lib/libpythonx.x.sys.dylib
$ which python | xargs otool -L

    This will output /the/path/to/python/library.

$ sudo ln -s /the/path/to/python/library /usr/lib/libpythonx.x.dylib
$ rm -r build/
$ ./configure
$ cd build
$ make
$ sudo make install
```

3.5.3 Linux

This is untested so use at your own risk!

```
$ ./configure
$ cd build
$ make
$ sudo make install
```

3.5.4 Install Locations

Quakelib libraries will be installed in standard library directories based on your system configuration. Cmake will generate a file named "install_manifest.txt" in the build directory detailing the locations of installed files. The Virtual California binary is build/src/vc.

3.6 Configuration

The `configure` script checks for various system features. As it runs, it prints messages informing you of which features it is checking for. Upon successful completion, it generates a `Makefile` in each source directory of the package. It also generates a `config.h` header file, which contains system-dependent definitions.

The `configure` script will attempt to guess the correct values of various installation parameters. In the event that the default values used by `configure` are incorrect for your system, or `configure` is unable to guess the value of a certain parameter, you may have to specify the correct value by hand.

> **Important:** If the `configure` script fails, and you don't know what went wrong, examine the log file `config.log`. This file contains a detailed transcript of all the checks `configure` performed. More importantly, it includes the error output (if any) from your compiler. When seeking help for `configure` failures on the CIG XXXXXXXXX Mailing List (`cig-mc@geodynamics.org`), please send `config.log` as an attachment.

Upon successful completion, `configure` will print a brief configuration summary.

3.6.1 Environment Variables

The following is a summary of the variables and options that are important when installing Virtual California. Environment variables may be specified as arguments to `configure`, e.g.,

```
$ ./configure CC=icc # use the Intel compiler
```

Variable	Description

CC	C compiler command. This is usually set to the name of an MPI wrapper command, e.g.,
	`./configure \`
	`CC=/opt/mpich-1.2.6/bin/mpicc`

CXX	C++ compiler command. This is usually set to the name of an MPI wrapper command, e.g.,
	`./configure \ CC=g++`
	or
	`./configure \ CC=mpicxx`

3.6.2 MPI Configuration

By default, `configure` will search for a C compiler using your `PATH` environment variable. It prefers MPI wrapper commands (such as `mpicc`) to ordinary compiler commands (such as `cc` or `gcc`). You may specify the compiler command name manually using the `CC` variable:

```
$ ./configure CC=/opt/mpich-1.2.6/bin/mpicc
```

The `configure` script will test for the presence of the MPI header (`mpi.h`) and an MPI library using the C compiler command. If `CC` is set to an MPI wrapper command such as `mpicc`, and/or the MPI header and library files are installed in a standard location (i.e., `/usr/include` and `/usr/lib`), these `configure` tests should succeed without difficulty.

3.6.3 Additional Tools

While the following software is not necessary for the normal operation of Virtual California, you may find it useful for accessing Virtual California data in HDF5 files.

3.6.3.1 NumPy

NumPy is an extension to Python which adds support for multi-dimensional arrays for use in scientific computing. You may download NumPy from the NumPy home page (`numpy.scipy.org`). To compile and install this extension, download it and issue the following commands after extracting it:

```
$ cd numpy-1.0
$ python setup.py install --prefix=$HOME/cig
```

Alternatively, under Debian Linux you can install the `python-numpy` package. On Gentoo Linux, NumPy is available in the `dev-python/numpy` ebuild.

3.6.3.2 PyTables

PyTables is a package for managing hierarchical datasets and designed to efficiently and easily cope with extremely large amounts of data. PyTables is built on top of the HDF5 library, using the Python language and the NumPy package. After checking its dependencies, you may download PyTables from the PyTables home page (`pytables.org`). To install, follow instructions on the installation page (`http://pytables.github.io/usersguide/installation.html`).

3.6.3.3 HDFView

HDFView is a visual tool written in Java for browsing and editing HDF5 files. You may download it from the HDFView home page (`hdf.ncsa.uiuc.edu/hdf-java-html/hdfview`).

3.7 Testing

After following the instructions in Section 3.5, you should run the install tests. From within the 'test' directory just run the 'run_test' script:

```
$ ./run_test

    Virtual California will run a small simulation creating the files:
    test_greens.h5 - The interaction matrices.
    test_dyn-0-5_st-5.h5 - The simulation file.
    block_info.dat - A element mapping file.

TODO: Update with Eric's unit tests.
```

Chapter 4

Running VC

4.1 Introduction

Now that installation and testing is finished, let's get into the specifics of building a fault model, compiling Virtual California, and finally running a custom simulation. The following chapter serves to illustrate the main features of a Virtual California simulation, and will prepare the user for the explicit examples in Chapter 4.

The only input needed to run a Virtual California simulation is the fault geometry and friction parameters

4.2 Setting Simulation Parameters

4.2.1 Units

All units used in calculations, unless otherwise specified, adhere to the metric system (M-K-S).

4.3 Simulation Performance and Scaling

The size of the meshed elements will affect both simulation accuracy and computational resource requirements. The appropriate size of fault elements for a given simulation depends on the desired minimum earthquake magnitude and the available computational resources.

4.3.1 Element Size and Minimum Magnitude

The relationship between element size and minimum magnitude is given by:

$$M_{min} = \frac{2}{3} log_{10}(1e7 \mu s L^2) - 10.7 \tag{4.1}$$

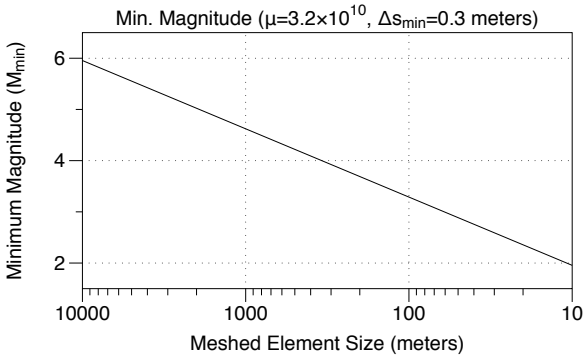
where μ is the Lamé parameter, s is the minimum slip distance, and L is the element size in meters. The minimum slip distance, s, will depend on the orientation of the element, the total fault size, and the interaction with other elements in the system, but will generally range from approximately 0.1 to 1.0 meters. For example, for the 12km x 12km fault described in Section 2.1.2 the minimum slip distance is 0.3 meters. Figure 4.1b shows the relationship in VC between element size and minimum earthquake magnitude.

4.3.2 Element Size and Required Memory

The number of elements in a model will also affect the required memory. Since VC uses a Barnes-Hut style approximation algorithm for fault interaction the actual memory required will depend on the accuracy of the approximation. For non-approximated fault interaction in a model with N elements, the memory requirements are:

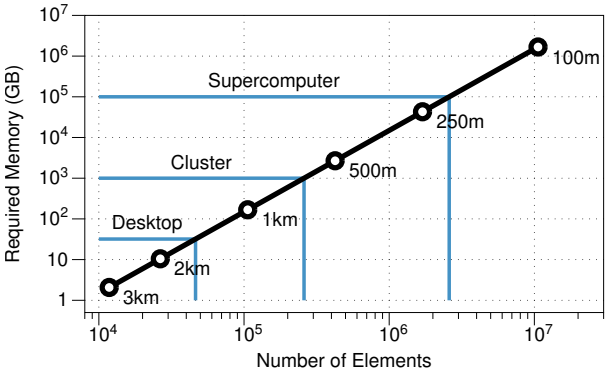
Operation	Minimum	Maximum
Addition	3,622	10,804
Multiplication	8,708	25,566
Square Root	260	780
Branch	1,148	3,924
Other	571	1,923
Total	14,309	42,997

(a) Operations required to calculate a single element-element stress interaction.

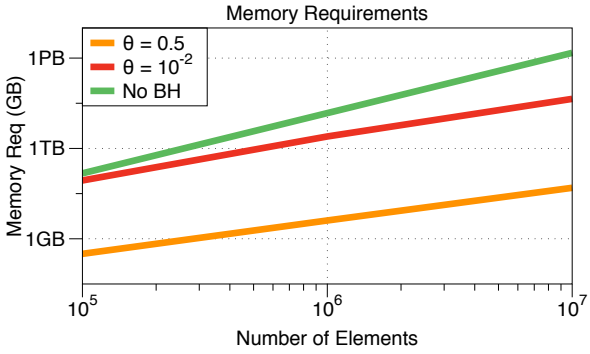


(b) Element size versus minimum event magnitude.

Figure 4.1: Number of elements determines computational cost. Element size governs simulated earthquake magnitude range.



(a) Element size determines computational regime.



(b) Number of elements vs. memory requirements with approximations.

Figure 4.2: Tradeoffs in element size and resource requirements.

$$Memory = 16N^2 bytes \tag{4.2}$$

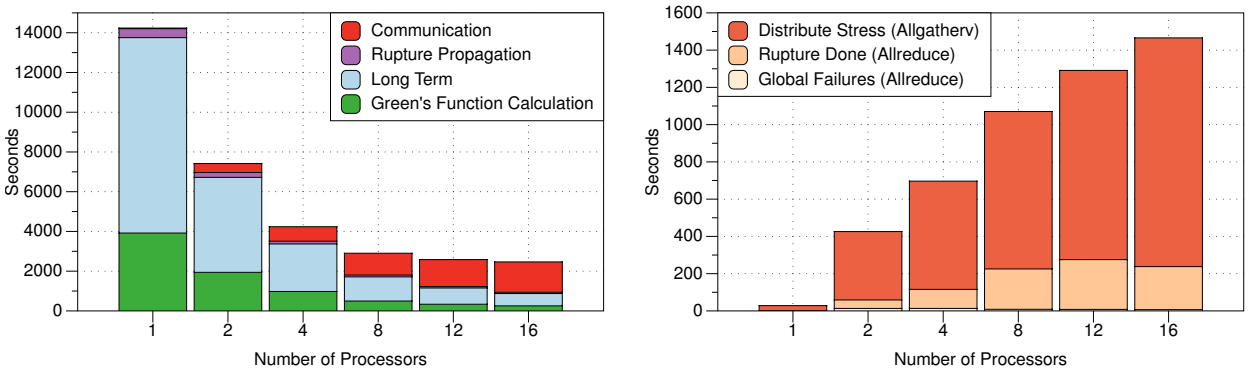
Because the degree of approximation possible with Barnes-Hut will vary depending on the system geometry it is not possible to give a function for memory requirements. For the UCERF model of all major faults in California (cite) we find the following equation gives a reasonable estimate of memory requirements for a given Barnes-Hut θ parameter and number of elements N:

$$Memory = XXXbytes \tag{4.3}$$

4.3.3 Parallel Performance

The program flow in Figure 2.5 shows three main communication points. The global nature of these communications is a serious bottleneck to simulation performance on a parallel system.

Figure 4.3a shows simulation run times for a 13,398 element 100,000 year simulation on a simple cluster system using different numbers of processors. The cluster has 16 nodes, with two 2.4 GHz quad-core Xeon processors per node and a Gigabit ethernet network. For a single processor, most of the time is spent in the long term stress calculation and the initial Green's function calculations. As the number of processors



(a) Parallel simulation time breakdown.

(b) Parallel communication time breakdown.

Figure 4.3: Breakdown of total simulation time.

increases these calculations scale well but the time spent in communi- cation significantly rises. By 16 processors the majority of simulation time is spent communicating and there is very poor strong scalability.

Figure 4.3b shows the breakdown in time for different types of communication. For all numbers of processors, the majority of time is spent distributing stress values over the processors. This is required because the changes in stress during a rupture potentially lead to other elements rupturing. However, at a given rupture propagation step it is highly unlikely the rupture will spread a significant distance past where it already has traveled.

Chapter 5

Tutorials

5.1 Overview

These tutorials are meant to serve as a guide to some of the types of problems VC can solve. These cookbook examples are distributed with the package under the `examples` directory. However, you might need to edit these example scripts slightly to launch the job on your cluster (see Section **??** on page ?? for more information). Each tutorial is a self-contained lesson in how to use Virtual California. The tutorials increase in degree of complexity from one to the next.

5.1.1 Prerequisites

Before you begin any of the tutorials, you will need to install Virtual California following the instructions in Chapter 3 on page 23. If you do not wish to create your own mesh at this time, the meshes are also provided as part of the tutorial.

5.1.1.1 Input Files

The fault model and a file named params.d must be in the same directory as the compiled application. An example params.d file can be found in example/sample_params.d. In order to run, the model files must be named in the QuakeLib file parameters section of the file. Only a friction and geometry file are necessary. See section A on page 37 and section B.2 on page 41 for more information on the input fault model files and parameters.

5.2 Tutorial Using Single Element

5.2.1 Overview

This tutorial is the simplest possible implementation

5.2.2 Mesh Description

Figure 2.2a shows the result of meshing the One_Fault_Example trace in Section 2.1.2 with

Part III

Appendices

Appendix A

Input Parameters for Virtual California

A.1 Input Parameters Grouped by Functionality

This section explains the meaning of the input parameters for Virtual California. These parameters are grouped by their functionality. Parameters are given with their default values.

A.1.1 Simulation time parameters

`sim.time.start_year = 0`	The starting year of the simulation
`sim.time.end_year = 1000`	The ending year of the simulation.

A.1.2 System Parameters

`sim.system.rawfile.lat0 = 31.5` `sim.system.rawfile.lon0 = -126`	Base latitude and longitude for coordinates of meshed model. These values are used in the conversion from This should be roughly within the bounds of your fault model to avoid warping from conversion to a Cartesian coordinate system. If undefined, defaults to (-126.0, 31.5) (about 500 miles W-SW from Los Angeles).
`sim.system.kill_cff`	Below what CFF a fault will be deactivated. Certain fault configurations can have stress sinks, and this parameter will effectively remove them. If undefined, defaults to being turned off (negative infinity).
`sim.system.sanity_check = false`	Whether to perform sanity checks on simulation values and abort if any values are outside acceptable ranges.
`sim.system.transpose_matrix = true`	Whether to store the Green's matrix in a transposed form to significantly improve performance. This should only be set to false for performance profiling.
`sim.system.progress_period = 0`	How frequently (in real seconds) to display simulation progress. If undefined or $<= 0$, simulation progress will not be displayed.
`sim.system.depth_dependent_slip = false`	Whether the fault velocity (slip rate) should be altered by depth. This is based on the equation $v = v_s(1 - (\frac{z}{d})^2)$ where v_s is the original long term slip velocity, d is the total fault depth, and z is the depth of the block center.
`sim.system.checkpoint_period`	How frequently (in simulation events) a checkpoint is saved. If undefined or $<= 0$, checkpoints will not be saved.
`sim.system.checkpoint_prefix=sim_state_`	The prefix of the state save files, which will also include the event count.

A.1.3 Friction parameters

`sim.friction.dynamic`	The dynamic rupture value to use in the simulation from 0 to 1. Higher values indicate ruptures are more likely to propagate along a fault and result in larger earthquakes, while lower values indicate ruptures are less likely to propagate and result in smaller earthquakes. If undefined or outside the allowed range, use the dynamic values specified for each fault element.
`sim.friction.law=original`	Either use the `original` or `stepped` friction law.
`sim.friction.slip_scaling_threshold=1.0`	The slip scaling threshold parameter used to scale slip with rupture size as per Equation 2.11.

A.1.4 Green's function parameters

`sim.greens.method = standard`	The method to calculate the Green's functions for fault element stress interactions. There are three possible choices for this parameter - `standard`, `bh` and `file`. The `standard` option will calculate the Green's functions using the normal Okada equations with all element-element interactions. The `bh` option will use a Barnes Hut style approximation. The `file` option will read precalculated values from an input file (specified using `sim.greens.input`).
`sim.greens.bh_theta = 0.0`	Parameter for Barnes Hut calculation of Green's function (between 0 and 1). Lower values mean less of an approximation. If undefined, defaults to 0 (meaning it effectively doesn't use Barnes Hut approximation).
`sim.greens.input`	If `sim.greens.method` is defined as `file`, this is the name of the HDF5 file to read the Green's function values from.
`sim.greens.output`	The name of the HDF5 file to write the Green's function values to.
`sim.greens.use_normal = true`	Whether to use the Green's normal stress function in calculations or just the Green's shear function.
`sim.greens.kill_distance = 0.0`	Kills interaction between any two blocks greater than this distance (in km) apart. If undefined or $<= 0$, all interactions will remain the same.

A.1.5 File name parameters

`sim.file.system_output`	The system file of the simulation. If undefined, no system information file will be written.
`sim.file.events`	The file which events will be written to. If undefined, events will not be recorded in an ASCII format.
`sim.file.hdf5_output`	The file to write HDF5 formatted simulation and event data into. If undefined, HDF5 data will not be recorded.

A.1.6 EqSim File Parameters

`sim.eqsim.file.output`	The file which EqSim style event logging will be written to. If undefined, the EqSim events file will not be created.
`sim.eqsim.file.condition`	The file to read EqSim style initial condition information from. If undefined, initial shear and normal stresses will be 0 and rhogd will be 1.557e8.
`sim.eqsim.file.friction`	The file to read EqSim style friction information from. If undefined, EqSim input files will not be used.
`sim.eqsim.file.geometry`	The file to read EqSim style geometry information from. If undefined, EqSim input files will not be used.
`sim.eqsim.file.slipmap_mag = 7.5`	The magnitude cutoff to print slip maps in the EqSim event file.

A.1.7 Noise parameters

`sim.noise.event=0.0`	Noise in single stress drop events.
`sim.noise.slip_deficit=0.0`	Noise in initial slip_deficit.
`sim.noise.stress.stress=0.0`	Noise applied to stress drop at start of simulation (affects all of simulation).
`sim.noise.resolution=10.0`	Stress noise resolution in km (over what range to apply a single noise value)

A.1.8 BASS (Branching Aftershock Sequence) model parameters

`sim.bass.max_generations = 0`	Maximum number of aftershock generations to generate in BASS model. If this is 0 then the BASS aftershock model will not be used.

`sim.bass.mm = 4.0`
`sim.bass.dm = 1.25`
`sim.bass.b = 1.0`
`sim.bass.c = 0.1`
`sim.bass.p = 1.25`
`sim.bass.d = 300`
`sim.bass.q = 1.35`
	Different parameters for BASS model. See paper [9] for details.
	Mm: minimum magnitude
	dM: strength of aftershock sequence (intensity of aftershocks)
	b: scaling of frequency magnitude
	c: start of aftershocks in days
	p: time decay rate of aftershocks
	d: distance parameter for aftershocks in meters
	q: distance decay rate

A.1.9 Parallel simulation parameters

`sim.parallel.spec_exec = none`	Choose the type of speculative execution to be used to improve simulation speed. Currently experimental. Can be either `none`, `fixed`, or `adaptive`. If `none` is chosen, speculative execution will not be used. If `fixed` or `adaptive` are chosen, the speculative execution scheme is used with either a fixed boundary distance or adaptive distance.
`sim.parallel.spec_exec_distance = 0`	The boundary distance to use in the `fixed` speculative execution scheme. This affects how frequently the simulation goes into speculative mode.

Appendix B

Virtual California Input File Format

B.1 Introduction

There are several input files used in VC.

B.2 Trace File Format

The initial fault geometry for VC runs is defined by the trace file. Each trace file describes a single fault by the location of points along the fault trace and associated fault characteristics at each of the points. A single VC model for a simulation can be generated by combining multiple fault traces using the vc_edit program (see Cookbook XXX for an example).

The trace files are ASCII format with comments indicated by a # (hash) mark. Lines that begin with a # will be ignored and any values after a # in a line will be ignored. The initial line in the trace file outlines the fault described in the file using the following attributes:

`fault_id`	ID number of the parent fault of this section. Used to unify multi-segment faults defined in separate files.
`num_points`	The number of trace points comprising this section.
`section_name`	Name of the section, may not contain whitespace.

The remainder of the file defines each of the trace points for the fault. Each trace point is described using the following attributes. The units of the attributes in the file were chosen to be easily human understandable, however in the simulation they are converted to SI units.

`latitude`	Latitude of trace point (must be in [-90, 90]).
`longitude`	Longitude of trace point (must be in [-180, 180]).
`altitude`	Altitude of trace point in meters above ground. All faults should be underground (negative altitude). Faults defined above ground will have undefined results.
`depth_along_dip`	The depth of the fault along the dip in meters (must be greater than 0). For a dip angle of θ the actual depth of the fault will be depth_along_dip*$sin(\theta)$.
`slip_rate`	The long term slip rate of the fault in centimeters per year.
`aseismic`	The fraction of slip that is aseismic (must be in [0,1]).
`rake`	The fault rake angle in degrees (must be in [-180, 180]).
`dip`	The fault dip angle in degrees (must be in [0,90]).
`lame_mu`	Lame's mu parameter describing material properties in Pascals (must be greater than 0).
`lame_lambda`	Lame's lambda parameter describing material properties in Pascals.

Appendix C

Virtual California Output File Format

C.1 Introduction

The format of the output files of Virtual California is described here. All outputs are in non-dimensional units unless specified.

C.2 HDF5 Output

If Virtual California is compiled with the HDF5 library it is possible to output simulation results in this format. This format aims to be a full description of the simulation and results, containing both a description of the fault system and events that occurred on the system. In this way the user can write scripts that can read all information from a single source. The output file is composed of several tables and datasets describing the input and output to a simulation. Depending on user options some of these tables may be empty but they will always exist in the file.

C.2.1 About HDF5

The Hierarchical Data Format (HDF) is a portable file format developed at the National Center for Supercomputing Applications (NCSA) (`hdf.ncsa.uiuc.edu/HDF5`). It is designed for storing, retrieving, analyzing, visualizing, and converting scientific data. The current and most popular version is HDF5, which stores multi-dimensional arrays together with ancillary data in a portable self-describing format.

HDF5 files are organized in a hierarchical structure, similar to a Unix file system. Two types of primary objects, *groups* and *datasets*, are stored in this structure. A group contains instances of zero or more groups or datasets, while a dataset stores a multi-dimensional array of data elements. Both kinds of objects are accompanied by supporting metadata.

A dataset is physically stored in two parts: a header and a data array. The header contains miscellaneous metadata describing the dataset as well as information that is needed to interpret the array portion of the dataset. Essentially, it includes the name, datatype, dataspace, and storage layout of the dataset. The name is a text string identifying the dataset. The datatype describes the type of the data array elements. The dataspace defines the dimensionality of the dataset, i.e., the size and shape of the multi-dimensional array. The dimensions of a dataset can be either fixed or unlimited (extensible). The storage layout specifies how the data arrays are arranged in the file.

C.2.1.1 Accessing Data Using HDFView

NCSA HDFView is a visual tool for accessing HDF files. You can use it for viewing the internal file hierarchy in a tree structure, creating new files, adding or deleting groups and datasets, and modifying existing datasets. HDFView is capable of displaying 2D slices of multi-dimensional datasets, with navigation arrow buttons that enable you to range over the entire extent of a third dimension.

C.2.2 VC HDF5 Datasets

The HDF5 file output by VC contains two datasets describing some simulation parameters. The first dataset is named `base_lat_lon` and describes the latitude and longitude point used as the reference for converting input element locations to a Cartesian grid system for simulation. The second dataset is named `sim_years` and describes the beginning and ending simulation year.

C.2.3 Element Table

The table titled `block_info_table` describes the set of all elements used in the simulation. The information for each element is a row in the table, with each column representing an attribute of the block. The attributes are defined as follows:

Attribute Name	Attribute Description
`block_id`	A numerical ID of the element. These IDs should be in contiguousascending numerical order.
`fault_id`	A numerical ID for the fault the element belongs to. This allows elements to be grouped by fault.
`section_id`	A numerical ID for the subsection of the fault the element belongs to (if any). This allows faults to be subdivided into geologically distinct zones.
`m_x_pt1`	The X coordinate of the 1st point defining the element (in meters).
`m_y_pt1`	The Y coordinate of the 1st point defining the element (in meters).
`m_z_pt1`	The Z coordinate of the 1st point defining the element (in meters).
`m_das_pt1`	The distance of the 1st point along the fault strike (in meters).
`m_trace_flag_pt1`	EqSim specific flag defining where the 1st point is on the fault trace.
`m_x_pt2`	The X coordinate of the 2nd point defining the element (in meters).
`m_y_pt2`	The Y coordinate of the 2nd point defining the element (in meters).
`m_z_pt2`	The Z coordinate of the 2nd point defining the element (in meters).
`m_das_pt2`	The distance of the 2nd point along the fault strike (in meters).
`m_trace_flag_pt2`	EqSim specific flag defining where the 2nd point is on the fault trace.
`m_x_pt3`	The X coordinate of the 3rd point defining the element (in meters).
`m_y_pt3`	The Y coordinate of the 3rd point defining the element (in meters).
`m_z_pt3`	The Z coordinate of the 3rd point defining the element (in meters).
`m_das_pt3`	The distance of the 3rd point along the fault strike (in meters).
`m_trace_flag_pt3`	EqSim specific flag defining where the 3rd point is on the fault trace.
`m_x_pt4`	The X coordinate of the 4th point defining the element (in meters).
`m_y_pt4`	The Y coordinate of the 4th point defining the element (in meters).
`m_z_pt4`	The Z coordinate of the 4th point defining the element (in meters).
`m_das_pt4`	The distance of the 4th point along the fault strike (in meters).
`m_trace_flag_pt4`	EqSim specific flag defining where the 4th point is on the fault trace.
`slip_velocity`	The long term slip velocity of the element (in meters/second).
`aseismicity`	The fraction of the element slip that is aseismic (from 0 to 1).
`rake_rad`	The rake angle of the element (in radians).
`dip_rad`	The dip angle of the element (in radians).
`dynamic_strength`	The dynamic strength of the element (in Pascals).
`static_strength`	The static strength of the element (in Pascals).
`lame_mu`	The Lamé mu parameter used to calculate Green's function values on the element (in Pascals).
`lame_lambda`	The Lamé lambda parameter used to calculate Green's function values on the element (in Pascals).
`fault_name`	The name of the fault associated with the element.

C.2.4 Event Table

The table titled `event_table` describes the events that occurred during a simulation. This table gives a more general overview of the event information, with detailed information (e.g. which blocks slipped by how much in what order releasing how much stress) given in `event_sweep_table`.

Attribute Name	Attribute Description
`event_number`	A unique numerically ascending ID number of the event.
`event_year`	The simulation year that the event occurred in.
`event_trigger`	The ID of the block that triggered the event, i.e. the ID of the first block to fail.
`event_magnitude`	The moment magnitude of the event calculated based on the area, slip and shear modulus.
`event_shear_init`	The summed initial shear stress on all blocks involved in the event (in Pascals).
`event_normal_init`	The summed initial normal stress on all blocks involved in the event (in Pascals).
`event_shear_final`	The summed final shear stress on all blocks involved in the event (in Pascals).
`event_normal_final`	The summed final normal stress on all blocks involved in the event (in Pascals).
`start_sweep`	The ID of the first sweep of the event.
`end_sweep`	The ID of the final sweep of the event.
`start_aftershock`	The ID of the first aftershock caused by the event.
`end_aftershock`	The ID of the last aftershock caused by the event.

C.2.5 Event Sweep Table

The table titled `event_sweep_table` describes the individual block failures in the sweeps during an event. There can be multiple sweeps within an event and there can be multiple block failures within each sweep. Each line in the table represents a block failure in a given sweep

Attribute Name	Attribute Description
`event_number`	The event which this sweep is a part of.
`sweep_num`	The numerical ID of the sweep within the event (starts at 0 for each event).
`block_id`	The numerical ID of the block that failed.
`slip`	The amount that the block slipped in this sweep, not cumulative (in meters).
`area`	The area of the block that slipped (in square meters)
`mu`	The shear modulus of the block (in Pascals).
`shear_init`	The shear stress on the block before the sweep (in Pascals).
`normal_init`	The normal stress on the block before the sweep (in Pascals).
`shear_final`	The shear stress on the block at the end of the sweep (in Pascals).
`normal_final`	The normal stress on the block at the end of the sweep (in Pascals).

C.2.6 Aftershock Table

The table titled `aftershock_table` describes a set of aftershocks generated by the BASS model.

Attribute Name	Attribute Description
`event_number`	The event which generated this aftershock based on the BASS model.
`generation`	The BASS model generation number of the aftershock.
`magnitude`	The magnitude of the aftershock.
`time`	The simulation time when the aftershock occurred.
`x`	The X position of the aftershock.
`y`	The Y position of the aftershock.

Appendix D

License

Bibliography

[1] David M. Beazley. Swig: an easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[2] E. H. Field, T. E. Dawson, K. R. Felzer, A. D. Frankel, V. Gupta, T. H. Jordan, T. Parsons, M. D. Peterson, R. S. Stein, R. J. Weldon II, and C. J. Willis. The uniform california earthquake rupture forecast. *U.S. Geol. Surv. Open-File Rept. 2007-1437*, 2007.

[3] Yoshimitsu Okada. Internal deformation due to shear and tensile faults in a half-space. *Bulletin of the Seismological Society of America*, 82(2):1018–1040, 1992.

[4] J. B. Rundle, P. B. Rundle, W. Klein, J. de sa Martins, K. F. Tiampo, A. Donnellan, and L. H. Kellogg. Gem plate boundary simulations for the plate boundary observatory: A program for understanding the physics of earthquakes on complex fault networks via observations, theory and numerical simulation. *pure and applied geophysics*, 159:2357–2381, 2002.

[5] John B Rundle. A Physical Model for Earthquakes 2. Application to Southern California. *J. Geophys. Res.*, 93(B6):6255–6274, 1988.

[6] P. B. Rundle, J. B. Rundle, K. F. Tiampo, J. S. Sa Martins, S. McGinnis, and W. Klein. Nonlinear Network Dynamics on Earthquake Fault Systems. *Physical Review Letters*, 87(14):148501, October 2001.

[7] P.B. Rundle, J.B. Rundle, K.F. Tiampo, A. Donnellan, and D.L. Turcotte. Virtual california: Fault model, frictional parameters, applications. *pure and applied geophysics*, 163:1819–1846, 2006.

[8] Terry E. Tullis, Keith Richards-Dinger, Michael Barall, James H. Dieterich, Edward H. Field, Eric M. Heien, Louise H. Kellogg, Fred F. Pollitz, John B. Rundle, Michael K. Sachs, Donald L. Turcotte, Steven N. Ward, and M. Burak Yikilmaz. A comparison among observations and earthquake simulator results for the allcal2 california fault model. *Seismological Research Letters*, 83(6):994–1006, November/December 2012.

[9] D Turcotte, J Holliday, and J Rundle. BASS, an alternative to ETAS. *Geophys. Res. Lett*, 2007.

[10] G. Yakovlev, D. L. Turcotte, J. B. Rundle, and P. B. Rundle. Simulation Based Distributions of Earthquake Recurrence Times on the San Andreas Fault System. *AGU Fall Meeting Abstracts*, page A3, December 2005.

[11] M. B. Yikilmaz. *Studies of Fault Interactions and Regional Seismicity Using Numerical Simulations*. PhD thesis, University of California, Davis, 2010.

[12] M. B. Yikilmaz, E. M. Heien, D. L. Turcotte, J. B. Rundle, and L. H. Kellogg. A fault and seismicity based composite simulation in northern california. *Nonlinear Processes in Geophysics*, 18(6):955–966, 2011.