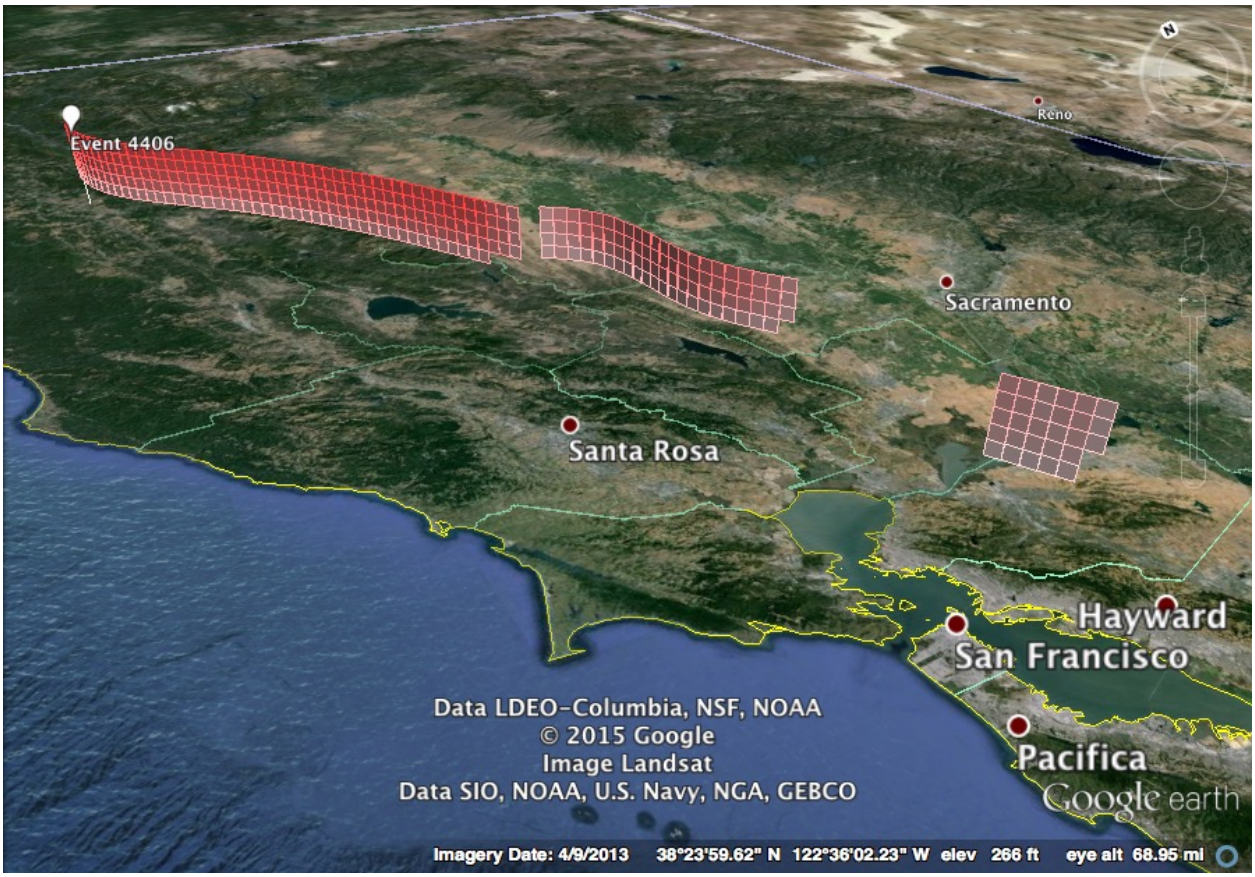
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

Virtual Quake

User Manual

Version 3.1.1



John M. Wilson Kasey W. Schultz Eric M. Heien Michael K. Sachs
Mark R. Yoder John B. Rundle Donald L. Turcotte

www.geodynamics.org

Virtual Quake User Manual

John M. Wilson
Kasey W. Schultz
Eric M. Heien
Michael K. Sachs
Mark R. Yoder
John B. Rundle
Donald L. Turcotte

Version 3.1.1

December 8, 2017

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 Quake, 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.

Errors and bug fixes in this manual should be directed to the CIG Short Term Crustal Dynamics Mailing List (`cig-short@geodynamics.org`). Please specify which code and version you are using when reporting a problem.

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 Quake on a variety of different platforms. Virtual Quake 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 July of 2014. A number of individuals have contributed a significant portion of their careers toward the development of Virtual Quake. It is essential that you recognize these individuals in the normal scientific practice by citing the appropriate peer reviewed papers and making appropriate acknowledgements.

To cite the Virtual Quake project:

- Wilson J.M., Schultz K.W., Heien E.M., Sachs M.K., Yoder M.R., Rundle J.B., Turcotte D.L. (2017), Virtual Quake [software], Computational Infrastructure for Geodynamics, Available from: geodynamics.org, doi: 10.5281/zenodo.797896, url: (`https://geodynamics.org/cig/software/vq/`)

To cite a specific version of the software, please see the citation builder for that version at (`https://geodynamics.org/cig/software/vq/`)

Additionally, the Virtual Quake development team asks that you cite the following:

- Schultz, K. W. and Yoder, M. R. and Wilson, J. M. and Heien, E. M. and Sachs, M. K. and Rundle, J. B. and Turcotte, D. L. (2017) "Parametrizing physics-based earthquake simulations", Pure Appl. Geophys. doi:10.1007/s00024-016-1428-3

- Schultz, K. W. and Sachs, M. K. and Heien, E. M. and Rundle, J. B. and Turcotte, D. L. and Donnellan, A. (2016) "Simulating Gravity Changes in Topologically Realistic Driven Earthquake Fault Systems: First Results", Pure Appl. Geophys. doi:10.1007/s00024-014-0926-4

- Schultz, K. W. and Sachs, M. K. and Heien, E. M. and Yoder, M. R. and Rundle, J. B. and Turcotte, D. L. and Donnellan, A. (2015) "Virtual Quake: Statistics, Co-Seismic Deformations and Gravity Changes for Driven Earthquake Fault Systems", International Association of Geodesy Symposia doi:10.1007/1345_2015_134
- Eric M. Heien, Michael Sachs, "Understanding Long-Term Earthquake Behavior through Simulation," Computing in Science and Engineering, vol. 14, no. 5, pp. 10-20, Sept.-Oct. 2012, doi:10.1109/MCSE.2012.39
- 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

And to cite the manual:

- Wilson J.M., Schultz K.W., Heien E.M., Sachs M.K., Yoder M.R., Rundle J.B., Turcotte D.L. (2017), Virtual Quake User Manual, Version 3.1.1. Davis, California, USA: Computational Infrastructure for Geodynamics. URL: https://geodynamics.org/cig/software/vq/vq_manual_3.1.1.pdf

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

Introductions

Virtual Quake 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 E). 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 Quake have been placed in the QuakeLib library which can be called from C/C++ programs or Python scripts. The QuakeLib library uses the SWIG framework and can therefore be extended to a wide variety of other languages as well.

1.1 About Virtual Quake

Virtual Quake 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 Quake (abbreviated VQ) was originally named Virtual California (VC) until release 1.1 in November 2014. It 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 [11], 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 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 [12]. Additional features were also added, including a branching aftershock sequence (BASS) model simulation of aftershocks [13], improved stress Green's function calculations, more sophisticated rupture propagation, and support for 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.

To reflect the fact that it supports a flexible fault model, Virtual California was renamed Virtual Quake (abbreviated VQ) on release 1.1 in November 2014.

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 Quake, 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) [2]. 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. Currently the QuakeLib library is wrapped into a python module called "quakelib" that is utilized by the plotting and analysis scripts located in the "pyvq" folder, examples are given later in the text.

Chapter 2

VQ Components and Governing Equations

There are three major components that make up Virtual Quake: a fault model, a set of quasi-static interactions (Green's functions), and an event model.

The first three sections cover the main components of a VQ simulation, the fourth section describes the simulation flow, and the last section gives a visual tour of the QuakeLib module's functionality.

2.1 Fault Model

The basic components of the fault model are the fault elements and their parameters. Any fault system, specified by a trace file listing the latitude and longitude of each vertex along the traces, is split up into the functional members of a Virtual Quake simulation, the fault elements. The resolution of the fault system will determine the total number of elements, and each element's parameters determine the local fault geometry and motion.

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. The failure stresses, which are also required for the model, are derived from paleoseismic event recurrence times.

2.1.1 Included California Fault Model

The full model for the California fault system that is included with VQ is based on the ALLCAL2 fault model, shown in Figure 2.1. A detailed description of the ALLCAL2 model is at (`http://scec.usc.edu/research/eqsims/documentation.html`), and description of file formats is in [1]. 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 × 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 is because in the current simulation physics, this section produces many small events and considerably slows down the simulation.

2.1.2 Model Parameters and Initial Conditions

Virtual Quake currently requires that the user specify the fault geometry and the stress parameters on each element then run the simulation based on this. These parameters are briefly described in the following section, and explicit example fault models are constructed in chapter 5.

2.1.3 Setting Fault Parameters and Building a Fault Model

VQ treats a system of faults as multiple planar elements embedded in a flat homogeneous halfspace. To run Virtual Quake 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.1.

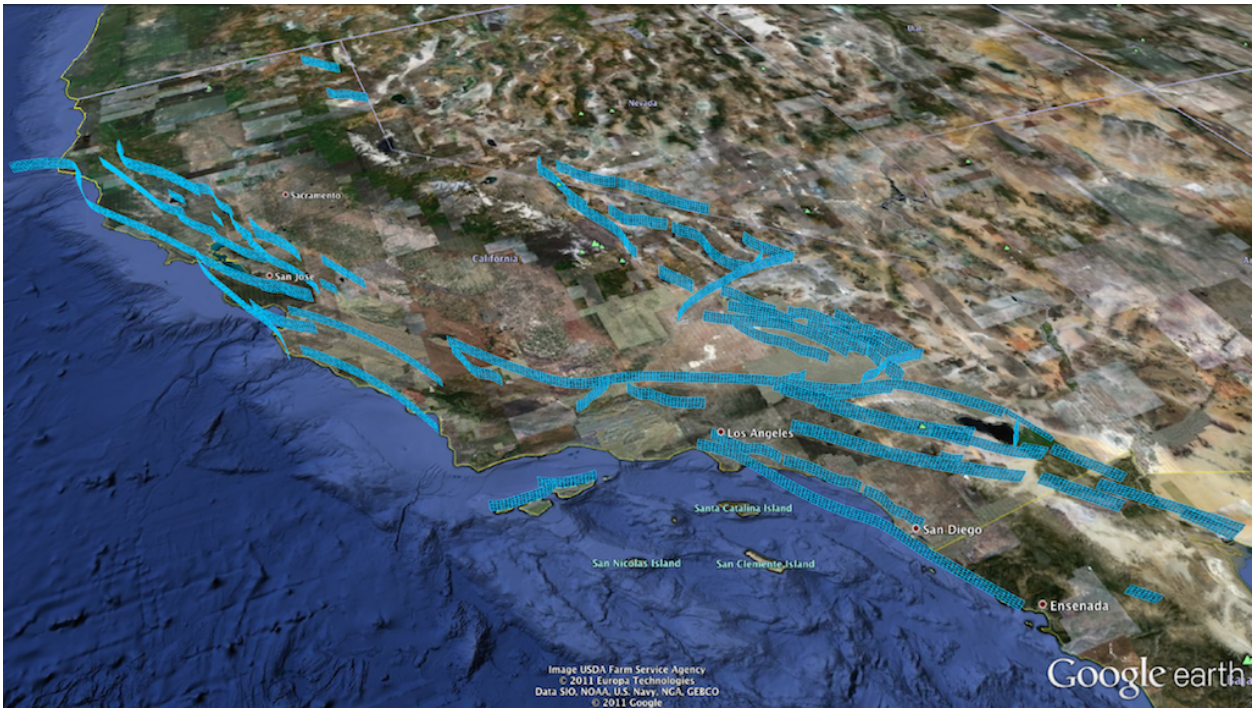


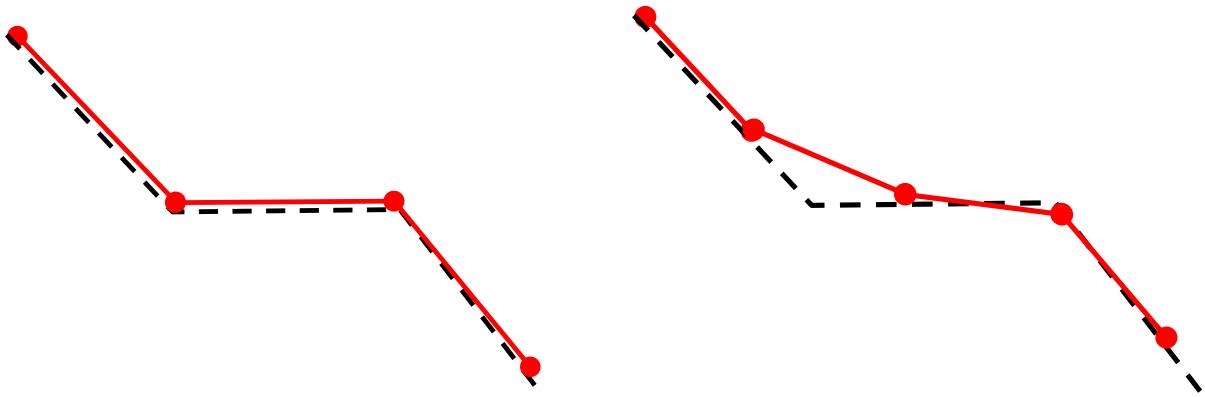
Figure 2.1: California fault system based on UCERF2, meshed into fault elements and shown above ground.

In this example we look at the earthquake cycle and rupture mechanics on a single 12 km by 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 (0,0.1078). 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
# sec_id: ID number of this section
# num_points: Number of trace points comprising this section
# section_name: Name of the section
0 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, section ID, number of trace points, and section name. In this example, the section is named "One_Fault_Example" and has section number 0. The section is associated with fault number 0. This naming convention is used to allow a large fault to be split into multiple sections for identification, but maintain the same physical properties as a single full fault.

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



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

(rake of 180°, dip of 90°). The Lamé parameters of $\mu = 3e10$ and $\lambda = 3.2e10$ indicate the material properties of the fault interface.

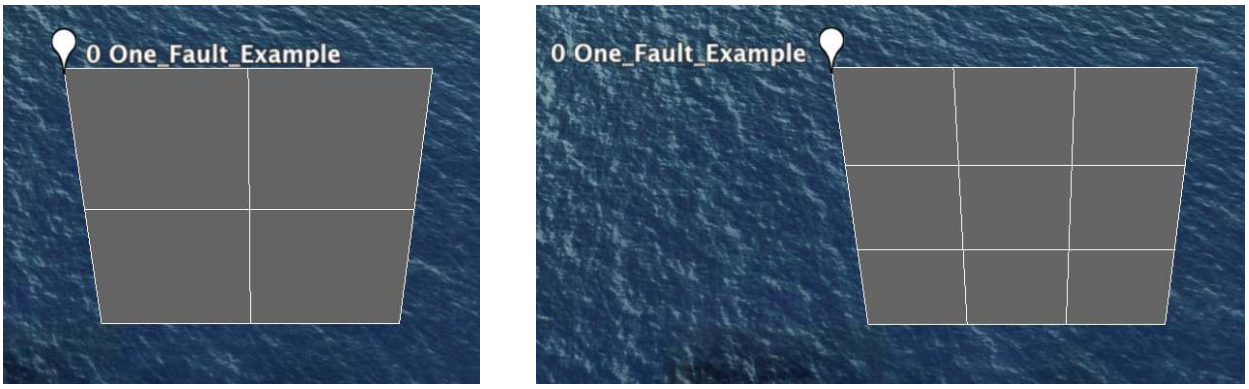
Given this fault definition we can create a mesh which fits within the fault dimensions. Each fault is meshed by specifying the fault trace file in the format described above, and a fault element size. Currently all fault elements in Virtual Quake are square though future versions will allow triangular 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.

Figure 2.2 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 2.2a 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 2.2b 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 during the meshing process. Appropriate element size is further discussed in Section 4.3.2. The meshing program and related parameters are described in Section 5.2.

Figure 2.3 shows the result of meshing the One_Fault_Example trace with different element resolutions. Figures 2.3a and 2.3b show the KML output of the program in Google Earth. 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 rather than centering the faults on (0,0) in the traces to help visualize the simulation results.



(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 2.3: A demonstration of meshing with different resolutions.

2.2 Element Stress Interactions

Unlike actual fault systems where the fault geometry is dynamic over long time periods, Virtual Quake simplifies calculations assuming a geometrically static fault system. In this way Virtual Quake 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]:

$$\sigma_{ij}(x,t) = \int dx_k^{'} T_{ij}^{kl}(x-x')s_l(x',t) \tag{2.1}$$

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 Quake 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 2.1 simplifies to:

$$\sigma_{ij}^{A}(t) = \sum T_{ij}^{AB} s_B(t) \tag{2.2}$$

where B runs over all elements. Finally, since Virtual Quake 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 Quake are calculated as:

$$\sigma_{s}^{A}(t) = \sum T_{s}^{AB} s_B(t) \tag{2.3}$$

$$\sigma_{n}^{A}(t) = \sum T_{n}^{AB} s_B(t) \tag{2.4}$$

Thus, for a fault model with N elements Virtual Quake 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]. Figure 2.7 on page 25 shows the stress field for a single vertical strike-slip fault element.

2.2.2 Event Transition Time

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

$$CFF^A(t) = \sigma_s^A(t) - \mu_s^A \sigma_n^A(t) \tag{2.5}$$

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 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:

$$\frac{dCFF^A}{dt} = (\sigma_s^A - \mu_s^A \sigma_n^A) + \alpha T_\alpha^A \tag{2.6}$$

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

Knowing the relationship between slip and stress (equations 2.3 and 2.4), it is not necessary to evolve the system time step by time step. Rather, the simulation time is advanced directly to the point at which the next element fails. Equations 2.5 and 2.6 allow us to analytically solve for the time when the next element will fail.

2.3 Rupture Model

A rupture event in VQ is comprised of multiple sweeps. During each sweep one or more elements fail and one or more elements slip. The sweeps continue as long as there are elements that fail - once no more elements fail the event is complete.

Rupture propagation consists of two internal phases - rupture/slip caused by static or dynamic failure and slip caused by the stress influence of other elements. The first phase involves elements rupturing and slipping due to static stress failure (CFF = 0) or dynamic failure (Equation 2.7). The second phase involves ruptured elements continuing to slip due to the changing stress influence of other elements. Each of these phases are described below.

During rupture propagation elements are either in a ruptured or non-ruptured state. Elements move from the non-ruptured to ruptured state either by static or dynamic failure. Elements move from ruptured to non-ruptured only when the event is complete. Static failure occurs when $CFF \geq 0$. 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 in turn 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. Dynamic failure is caused when there is a significant change in CFF during the event, satisfying:

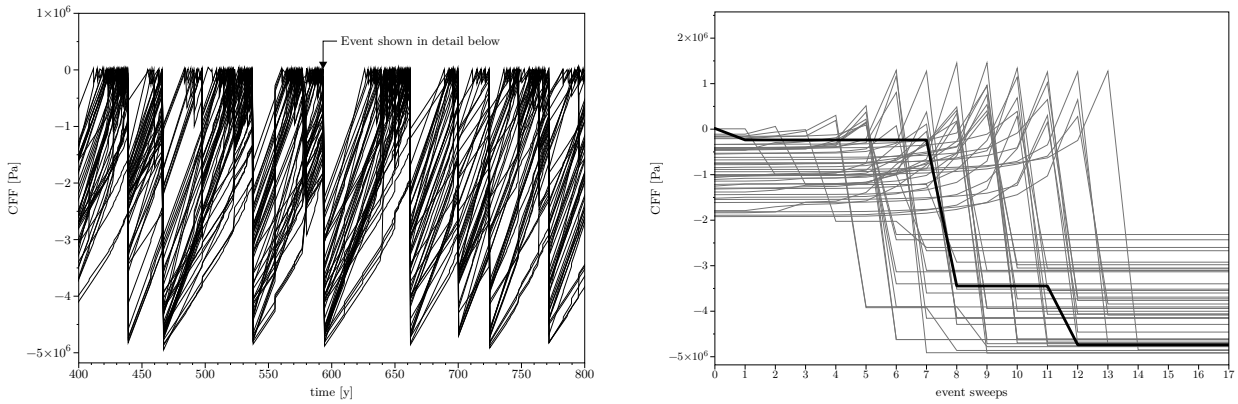
$$\frac{CFF_{init} - CFF_{final}}{CFF_{init}} > \eta \tag{2.7}$$

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.

The initial element rupture during an event is always caused by static failure. 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]:

$$\Delta s = \frac{1}{K_L}(\Delta\sigma - CFF). \tag{2.8}$$

where K_L is the element's stiffness or self-stress defined as $K_L = T_s^{AA} - \mu_s^A T_n^{AA}$. Once an element has ruptured due to static or dynamic failure it will no longer slip in the event due to these failures. However, ruptured elements may slip further due to the movement of other elements.



(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.

During each rupture sweep, ruptured elements may slip further due to movement on other elements. The amount of slip on a failed element is related to the movement of other elements through the Green's function. A simplistic approach to determining this is to calculate the slip on all other elements when a given element moves. However, this is highly inefficient because the slip on a given element will in turn cause slip on all ruptured elements, which will cause slip on all ruptured elements, and so on forever. Instead, the system is described using a set of linear equations relating the slip of each element to the stress on all other elements in the final state of a given sweep.

This relationship between element slips during a sweep is determined for each ruptured element A as:

$$\Delta\sigma_A - CFF_A = \sum (T_s^{AB} - \mu_s^B T_n^{AB}) s_B \tag{2.9}$$

Since non-ruptured blocks do not change their slip, they are excluded from the system. This system is then solved using Gaussian elimination. Once the slip is calculated for all ruptured elements, a new stress state for the entire system is calculated using Equations 2.3 and 2.4. The rupture process is repeated by again checking the static/dynamic failure until no more elements have failed.

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.

2.3.1 Stress Drops

The stress drops for the fault elements are computed via Wells and Coppersmith 1994 scaling relations along with an analytical solution for the shear stress change per unit slip on a vertical rectangular fault. The simulation is pretty sensitive to these values; large stress drops tend to produce larger earthquakes with longer periods between events, while smaller stress drops produce many more small earthquakes. We provide a tuning parameter that globally adjusts the stress drops, and you must specify this when using the mesher to generate your fault model file. The mesher parameter is `--stress_drop_factor=X.X` where X.X is the value of the stress drop factor. The default value is 0.3 and it's logarithmically scaled, and increasing this value by 0.1 multiplies the stress drops by 1.4, a 40% increase; typical values range from 0.2-0.8.

We begin by using scaling relations to determine a different characteristic magnitude M^{char} and a characteristic slip Δs^{char} for each element in a fault based on the fault's geometry from Leonard 2010 (improved upon Wells and Coppersmith 1994) scaling relations

$$M^{char} = 4.0 + \log_{10}(A) + \text{stress\_drop\_factor} \tag{2.10}$$

where A is the surface area of the fault in km^2. We then use the definition of moment magnitude to determine the mean slip Δs^{char}

$$\Delta s^{char} = \frac{10^{\frac{3}{2}(M^{char}+6.0)}}{\mu A} \tag{2.11}$$

where μ is the rigidity (shear modulus) of the elastic half space, and A here is the area of the fault in m^2. We then use this slip to determine the characteristic stress drop for each fault element

$$\Delta\sigma = -\frac{2\mu\Delta s^{char}}{(1-\nu)\pi R}\left((1-\nu)\frac{L}{W} + \frac{W}{L}\right). \tag{2.12}$$

where ν is the Poisson ratio of the elastic half space, W is the down-dip width of the fault, L is the length of the fault in the direction of slip, and

$$R = \sqrt{L^2 + W^2}. \tag{2.13}$$

2.3.2 ETAS Aftershock Model

We include an Epidemic Type Aftershock Model in Virtual Quake, specifically the BASS variant of the model described in [9]. By setting the simulation parameter `sim.bass.max_generations = 1` you can turn on the aftershock model. The aftershock model draws location from a prescribed Omori-type distribution, is mapped onto the closest simulation element to the randomly drawn location, and the aftershock is processed as a regular simulated earthquake and changes the local stress field accordingly. The aftershocks parameters are described in Section A.1.6.

2.4 Simulation Flow

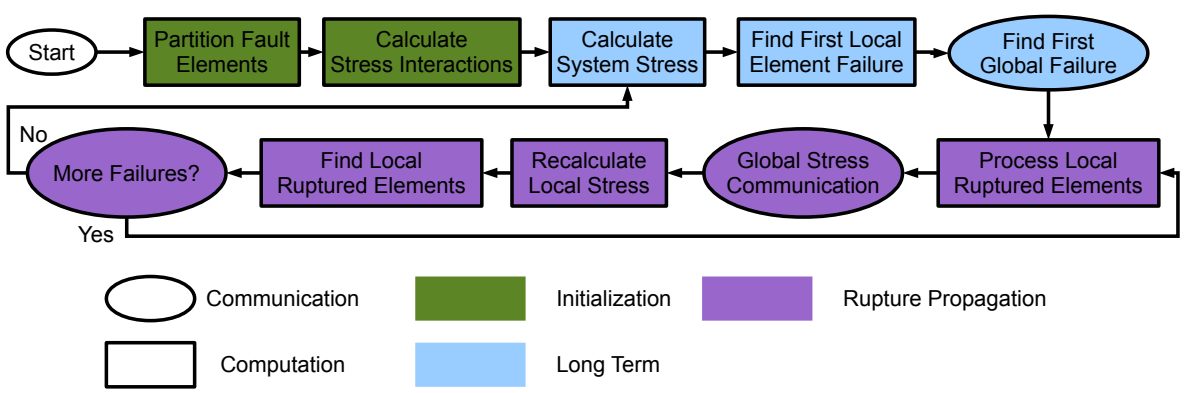
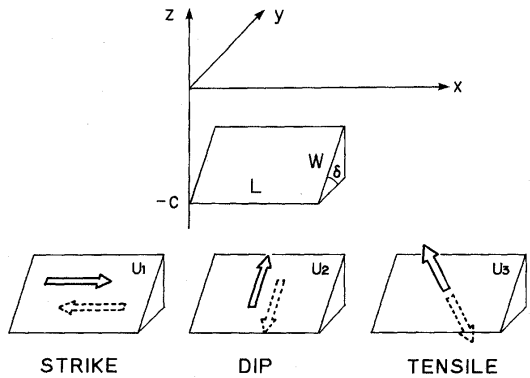


Figure 2.5: Simulation flow of Virtual Quake.

Figure 2.5 shows the flow of simulation in Virtual Quake. 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



Parameter	Description
L, W	Fault length, down-dip width
δ	Dip angle
C	Fault plane depth
U1	Slip unit vector for strike-slip
U2	Slip unit vector for dip-slip (U2>0 thrust; U2<0 normal)
U3	Slip unit vector for tensile
z	Distance below surface
x/y	Distance on surface

Figure 2.6: QuakeLib defines each fault element with parameters following Okada's convention.

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 2.5. 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 Quake 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.

2.5 QuakeLib Tour

As mentioned in Section 1.3, the QuakeLib library provides tools and a Python interface to develop earthquake simulations, read/write EqSim or VQ format files for geometry, friction, initial conditions and events, and calculate Okada's functions for arbitrary fault geometries. QuakeLib can be compiled and used independently from Virtual Quake. To only compile QuakeLib, follow the install instructions in section 3.3.3 but from the quakelib subdirectory.

This chapter provides a brief visual tour of a few analytical utilities in QuakeLib and to illustrate its capacity as the computational backbone for impressive and informative visualizations. The script that generates these plots is vq/pyvq/pyvq.py, and a tutorial for generating plots like these is given in section 5.7.

2.5.1 Conventions

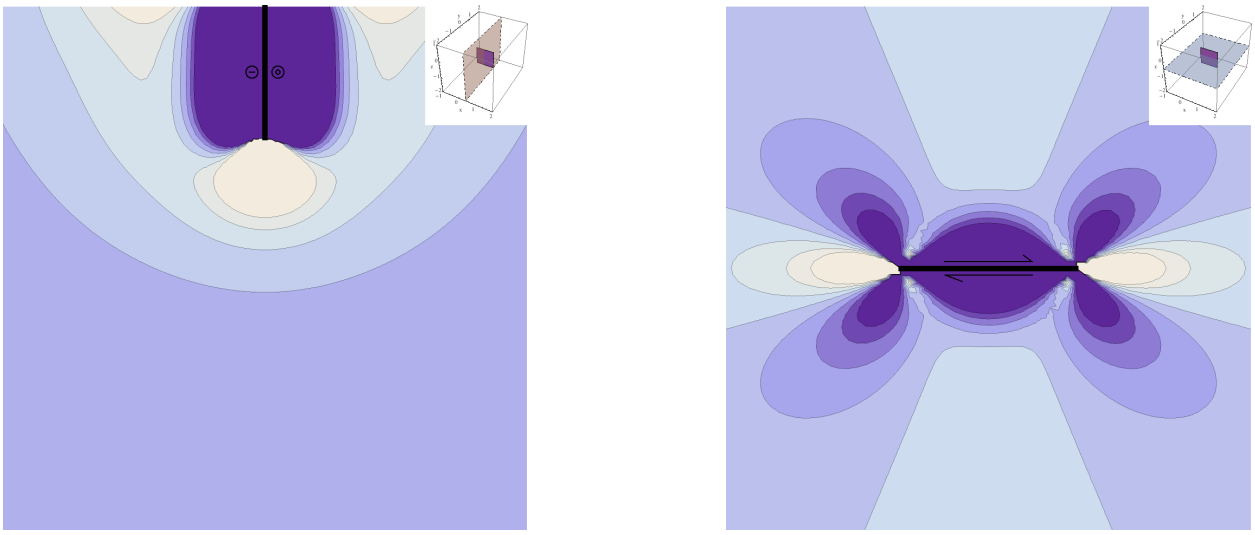
The QuakeLib functions act on single fault elements, and compute various dynamic quantities like the stress tensor, surface deformation field, and gravity anomalies. These functions take fault parameter values following Okada's convention [3]. Figure 2.6 shows Okada's convention for fault plane elements in an elastic halfspace $z \leq 0$. The two Lamé parameters (λ,μ) that describe the fault element's elasticity are also required. These parameters and their units are described in detail in Appendix B.1.

2.5.2 Single Element Examples

The following example plots provide a brief window into QuakeLib's analytical tools applied to single fault elements.

2.5.2.1 Stress Field

The stress field Green's function is defined in quakelib/src/QuakeLibOkada.cpp as **calc_stress_tensor.** This function computes the stress tensor at an arbitrary location in the elastic halfspace around the fault plane. Examples of the shear



(a) Front view

(b) Top view

Figure 2.7: The shear stress field σ_{xy} (tan $\sigma_{xy} > 0$, blue $\sigma_{xy} < 0$) created by horizontal backslip, viewed from the front and top of an element. The direction of backslip is indicated by the arrows.

stress field – equation 2.3 – computed by QuakeLib for a single vertical ($\delta = 90°$) strike slip fault element are shown in Figure 2.7.

2.5.2.2 Displacement Field

The Green's function is defined in quakelib/src/QuakeLibOkada.cpp as **calc_displacement_vector.** This function computes the co-seismic displacement vector for an arbitrary location in the elastic halfspace around the fault plane. Figure 2.8 shows displacement fields for faults of different dips as calculated by this function. In this figure the parameters for each fault element are L = 10km, W = 10km, slip = 5m, and the horizontal/vertical axes measure distance on the surface of the halfspace in km. The fault plane depth values (C) are 10km for strike-slip, 11km for normal, and 6km for thrust. The view is from above the fault plane looking straight down at the surface, and the thick black lines are the projections of the buried fault plane onto the surface of the halfspace.

2.5.2.3 Gravity Field Anomalies

The gravity Green's function is defined in quakelib/src/QuakeLibOkada.cpp as **calc_dg.** This function computes the gravity anomalies at an arbitrary surface location on the elastic halfspace around the fault plane. This is the total gravity field anomaly Green's functions, which includes contributions from subsurface density changes (dilatational) and from surface displacement (free-air). Examples of the gravity anomaly field for single fault elements is given in Figure 2.9. The fault parameters are the same as above.

2.5.3 Topologically Realistic Examples

The true power of QuakeLib lies in its ability to serve as the analytical backbone for visualizing the results of Virtual Quake's simulated seismic histories. The following example plots illustrate this point by showing QuakeLib's tools applied over many fault elements involved in a single simulated earthquake. The earthquakes visualized below come from a simulation involving all major fault sections in California, the UCERF2 model described in section 2.1.

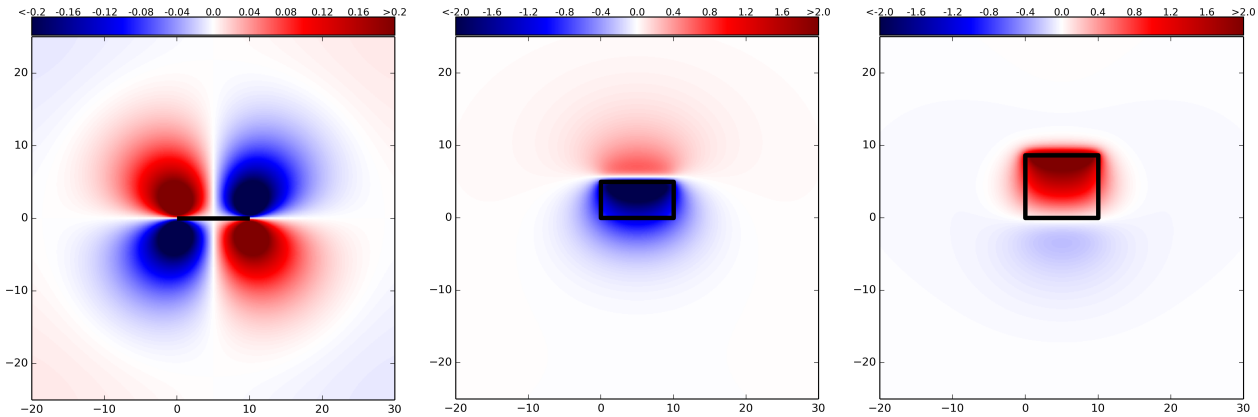


Figure 2.8: Vertical displacement at the surface for buried fault elements, depth to top of each fault plane is 1km, colorbar units are meters. **Left:** strike-slip $\delta = 90°$. **Center:** normal $\delta = 60°$. **Right:** thrust $\delta = 30°$.

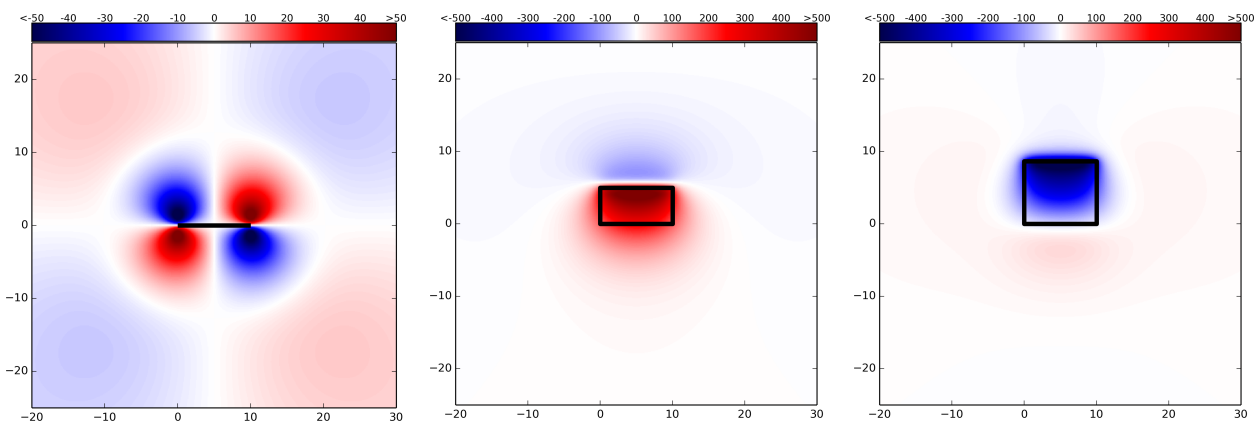


Figure 2.9: Gravitational anomalies for the fault elements in Figure 2.8, colorbar units are μgal.

2.5.3.1 Displacement Field

Figure 2.10 shows the vertical displacement field on the surface as seen by an orbiting satellite for a very large (moment magnitude 8.0) earthquake involving multiple sections of the San Andreas Fault.

2.5.3.2 Gravity Field Anomalies

Figure 2.11 shows co-seismic gravitational anomaly field on the surface, as measured by an orbiting satellite for the same simulated earthquake as Figure 2.10.

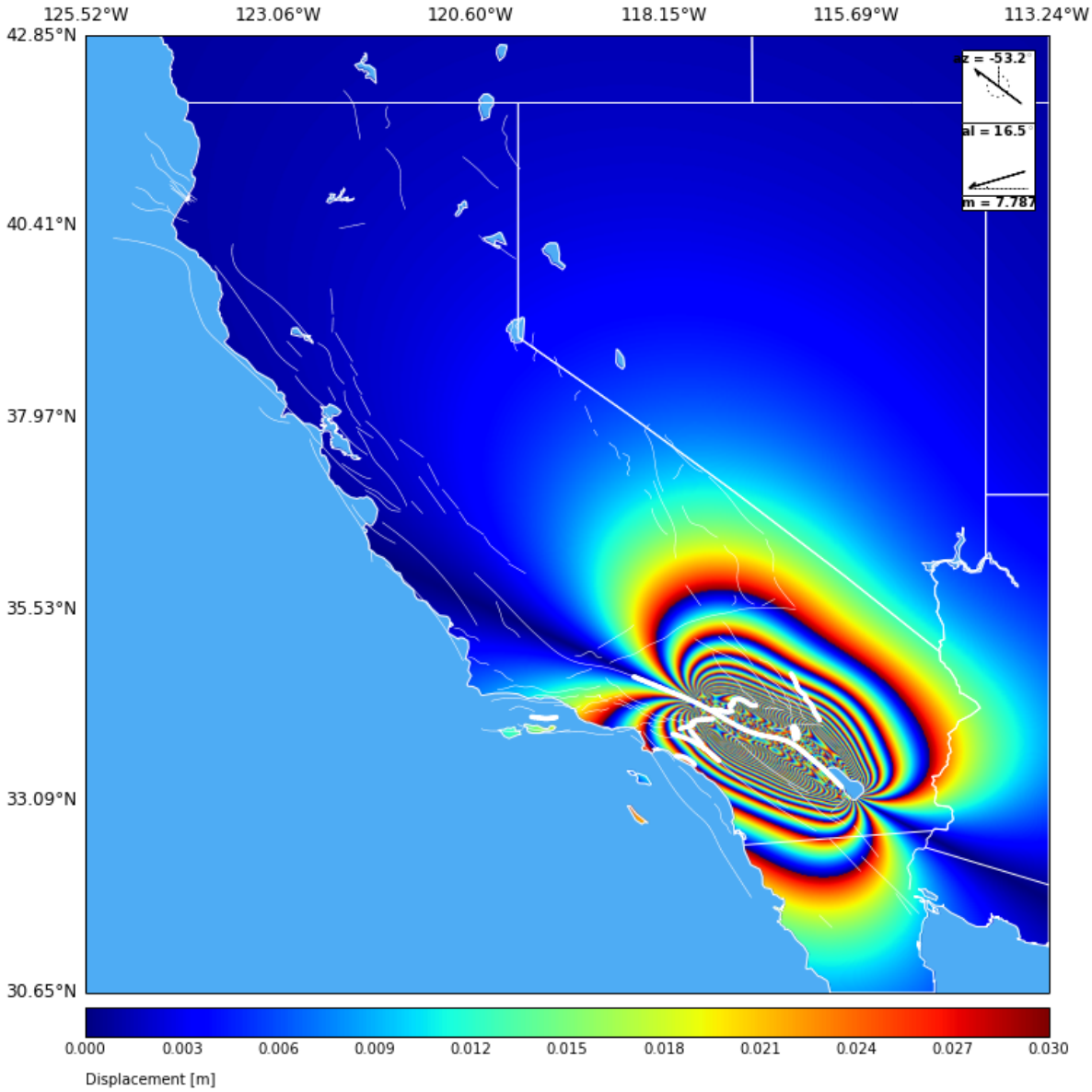


Figure 2.10: Simulated InSAR interferogram showing vertical displacements for a large earthquake (moment magnitude 7.79) involving multiple sections of the southern San Andreas Fault (thick white lines), as seen by an orbiting satellite.

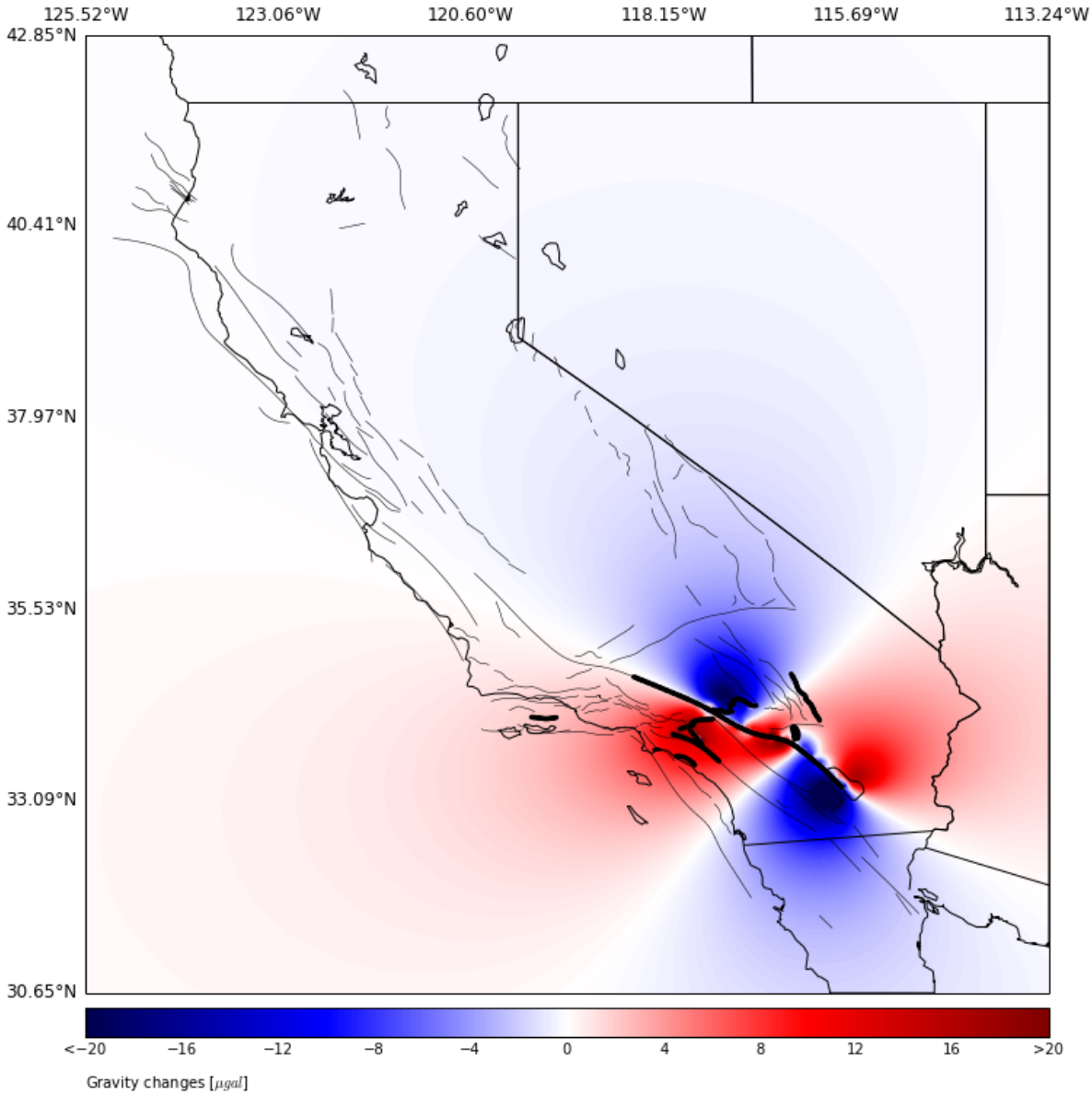


Figure 2.11: Simulated surface gravity anomalies for the same simulated earthquake as in Figure 2.10.

Chapter 3

Getting Started and Installation

3.1 Introduction

Virtual Quake is available in two forms: as a source package that users can compile on their own, and as a Docker image which is pre-compiled within a controlled container environment. Users interested in making modifications to the source of VQ should compile from source, while the Docker image is more likely to run without issue on a wide variety of systems, with no system requirements other than Docker itself. For most users, the Docker image is the recommended usage method.

The following sections will lead you through the installation process.

3.2 Getting Help

For help, send e-mail to the Short Term Crustal Dynamics Mailing List (`cig-short@geodynamics.org`). You can subscribe to the Mailing List and view archived discussion at the Geodynamics Mail Lists web page (`http://geodynamics.org/cig/about/mailing-lists/`). For bugs found in the manual or source code, or to make feature requests please use the Github issue tracker at (`https://github.com/geodynamics/vq/issues`).

3.3 Compiling from Source

3.3.1 System Requirements

Virtual Quake and QuakeLib have been tested on Linux, Mac OS X and several other UNIX based platforms. Virtual Quake has also been successfully run in parallel on several XSEDE systems and commodity cluster systems.

Installation of Virtual Quake requires a C++ compiler. Other requirements are the headers and development libraries for

- OpenMPI
- HDF5 (OpenMPI version)
- SWIG
- CMake

You must also have Python 2.7 or greater installed, with h5py.

3.3.2 Obtaining Source

To obtain the latest official release of Virtual Quake, go to the Geodynamics software package web page (`http://geodynamics.org/cig/software/vq`), download the source archive and unpack it using the `tar` command:

```
$ tar xzf vq-1.1.0.tar.gz
```

To get the latest development version of Virtual Quake, use git to make a copy of the repository:

```
$ git clone --recursive https://github.com/geodynamics/vq
```

3.3.3 Installation Procedure

After unpacking the source, use the following procedure to install the Virtual Quake executable as well as the QuakeLib library and the mesher program:

1. Navigate to the directory containing the Virtual Quake source.

   ```
   $ cd vq
   ```

2. Make the build directory and navigate to it.

   ```
   $ mkdir build
   $ cd build
   ```

3. Use CMake to configure before compiling VQ.

   ```
   $ cmake ..
   ```

4. Use make to build QuakeLib and the VQ binaries.

   ```
   $ make
   ```

5. The final step is required only if the user intends to use the quakelib python module
   `$ sudo make install`

If you are content to run Virtual Quake from the build directory, then you are done. Upon successful completion, the `make` command creates two executables "mesher" and "vq" in the `/build/src/` subdirectory. "vq" is the binary you will use to run a Virtual Quake simulation, and the mesher program can create fault models as described in Section 5.2.

3.3.3.1 Mac OS X

If you have a third party installation of python (e.g. from Homebrew or MacPorts) Virtual Quake will build and install, but the Python QuakeLib module may not work (failure will generally manifest itself as a segmentation fault). This is because CMake may build against a different installation of Python than it runs Python scripts with. To fix this you can specify the Python installation to use when compiling QuakeLib with:

```
cmake -DPYTHON_EXECUTABLE=/.../python \
    -DPYTHON_LIBRARY=/.../libpython2.7.dylib \
    -DPYTHON_INCLUDE_DIR=/.../include/python2.7/ ..
```

Be sure to change the paths to whatever is appropriate on your system.

3.3.3.2 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 Quake binary is at `build/src/vq`.

3.3.3.3 Selecting a Compiler — Multiprocessing

Depending on the machine used to run VQ, you may need to change which compiler CMake uses to compile VQ. For example, if the user wants to compile VQ with gcc 3.3, execute cmake as shown below.

```
$ CC=gcc-3.3 CXX=g++-3.3 cmake ..
$ make
```

To compile Virtual Quake and deploy it across multiple processors using MPI, execute cmake instead as:

```
$ CC=mpicc CXX=mpicxx cmake ..
$ make
```

3.3.3.4 HDFView

HDFView is a visual tool written in Java for browsing and editing HDF5 files. While it is not necessary for the normal operation of Virtual Quake, you may find it useful for accessing Virtual Quake data in HDF5 files. You may download it from the HDFView home page (`hdf.ncsa.uiuc.edu/hdf-java-html/hdfview`).

3.4 Docker Image

3.4.1 Initial Setup

All instructions for Virtual Quake Docker image setup can be found at the VQ Docker Hub, (`https://hub.docker.com/r/geodynamics/virtualquake/`).

The only requirement of the host system for running the Docker image of Virtual Quake is Docker itself, which can be downloaded from (`https://www.docker.com/products/docker`).

Once the Docker client is running on your machine, download the Virtual Quake image with

```
$ docker pull geodynamics/virtualquake:3.1.1
```

3.4.2 Running a Container

First, determine a directory on your computer (The ”host” computer) in which to store all simulation files. These will include any fault model files, simulation outputs, and PyVQ plots.

To allow the VQ container to access these files, include the full path of the chosen directory in the indicated spot in the `docker run` command. Since Docker works slightly differently on Mac and Linux, use the appropriate command for your machine:

- For Linux:

```
$ docker run --rm -v <full path to host directory>:/home/virtualquake/external_volume
         -e HOST_UID=$(id -u) -e HOST_GID=$(id -g) -it geodynamics/virtualquake:3.1.1
```

- For Mac:

```
$ docker run --rm -v <full path to host directory>:/home/virtualquake/external_volume
         -it geodynamics/virtualquake:3.1.1
```

3.4.3 Running Simulations

After executing the `docker run` command, you’ll be dropped into a terminal running inside the container, in a directory with access to all the files in the host directory you chose. The VQ fault mesher, simulator, and PyVQ analysis tools are located in the `~/vq-3.1.0` directory inside the container, and can be accessed and used normally while in the container console. Upon exiting the container terminal, the container will shut down, but all files stored and modified in the host directory will remain.

Chapter 4

Running VQ

4.1 Introduction

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

4.2 Basic Usage and Tests

The main procedure for running a VQ simulation is to create the fault model (see Sections 5.3 and 5.4 for examples), compile Virtual Quake following Section 3.3.3 to generate the executable, place the executable in the same folder as the parameter files, and finally execute the program.

4.2.1 CMake Tests

If you installed Virtual Quake according to Section 3.3.3, then you successfully compiled the QuakeLib library and the VQ and mesher executables. CMake is used to configure the simulation program, and also provides a test framework to ensure all parts of VQ are working as expected. To run the suite of tests use the following commands.

```
$ cd build/
$ make test
Running tests...
Test project /.../build
        Start   1: CondUnitTest
  1/230 Test   #1: CondUnitTest .............................   Passed    0.08 sec
        Start   2: FricUnitTest
  2/230 Test   #2: FricUnitTest .............................   Passed    0.09 sec
        Start   3: GreenUnitTest
  3/230 Test   #3: GreenUnitTest ............................   Passed    0.09 sec
        Start   4: OctreeTest
  4/230 Test   #4: OctreeTest ...............................   Passed    0.06 sec
        Start   5: UtilUnitTest
  5/230 Test   #5: UtilUnitTest .............................   Passed    0.15 sec
        Start   6: EventUnitTest
  6/230 Test   #6: EventUnitTest ............................   Passed    0.05 sec
        Start   7: GeomUnitTest
  7/230 Test   #7: GeomUnitTest .............................   Passed    0.05 sec
        Start   8: MetadataUnitTest
  8/230 Test   #8: MetadataUnitTest .........................   Passed    0.04 sec
        Start   9: RectBoundTest
```

```
  9/230 Test   #9: RectBoundTest ............................   Passed    0.04 sec
        Start  10: mesh_P1_none_12000
 10/230 Test  #10: mesh_P1_none_12000 .......................   Passed    0.01 sec
        Start  11: param_P1_none_12000
 11/230 Test  #11: param_P1_none_12000 ......................   Passed    0.01 sec
        Start  12: run_P1_none_12000
 12/230 Test  #12: run_P1_none_12000 ........................   Passed    0.05 sec
        Start  13: test_consistent_P1_none_12000
 13/230 Test  #13: test_consistent_P1_none_12000 ............   Passed    0.32 sec
        Start  14: test_slip_P1_none_12000
 14/230 Test  #14: test_slip_P1_none_12000 ..................   Passed    0.33 sec
        Start  15: test_interevent_P1_none_12000
 15/230 Test  #15: test_interevent_P1_none_12000 ............   Passed    0.31 sec
...
        Start 230: test_two_consistent_taper_renorm_3000
230/230 Test #230: test_two_consistent_taper_renorm_3000 ....   Passed    0.72 sec

100% tests passed, 0 tests failed out of 230

Total Test time (real) =  54.90 sec
```

The final lines of the output summarize the results from the unit tests and test simulations.

4.2.2 Explicit Test Simulation

If you would rather explicitly follow the steps of building your own fault model, generating the parameter files and running the simulation, then you should consult the tutorial in Section 5.3.

4.3 Advanced Usage

Since the simulation physics and event model will work with any arbitrarily complex fault model, advanced users can make Virtual Quake generate simulated seismic histories for any fault system. The only requirement for using VQ to simulate dynamics on an arbitrary fault network is to prepare the input files. Chapter 4 contains examples that illustrate this procedure of defining fault geometry and properties.

4.3.1 Tuning Parameters

Fault simulations must initially be tuned to correctly simulate actual earthquakes. The primary tuning parameters in Virtual Quake are the dynamic triggering factor η, defined in Section 2.3, and the stress drop factor defined in Section 2.3.1. The dynamic triggering factor is used to encourage rupture propagation during a simulated earthquake. This parameter acts to tune the rupture and slipping properties of faults without requiring field measurements of each fault's physical properties, a result of VQ's abstraction and generality. Currently, this parameter is set globally for the fault model. For parameter usage, see Section A.1.3.

Figure 4.1 shows the effect of a changing dynamic trigger factor η on the frequency-magnitude relation for simulations of the UCERF3 fault model with a fixed stress drop factor. Figure 4.2 shows the effect of a changing stress drop factor ΔM on the frequency-magnitude relation for simulations of the UCERF3 fault model with a dynamic trigger factor.

4.3.2 Simulation Performance and Scaling

Virtual Quake is designed to support parallel computing with OpenMP or MPI and this section gives some quantitative results of deploying VQ in multiprocessing environments. The key factors determining the scope of the output and the simulation performance are the number and size of the fault elements in the fault model. The size of the meshed elements will affect both simulation accuracy and computational resource requirements. The appropriate size

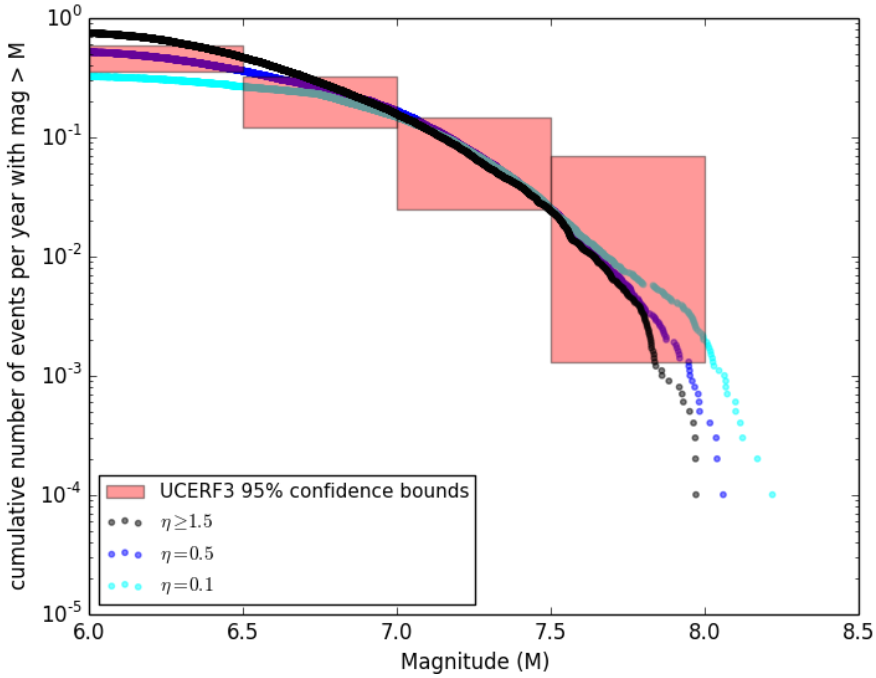


Figure 4.1: With the stress drop factor set to 0.7, this is the effect of tuning the dynamic trigger factor η on the frequency-magnitude distribution. These are 10,000 year simulations of the UCERF3 fault model compared to observed California earthquake rates and 95% confidence bounds in red.

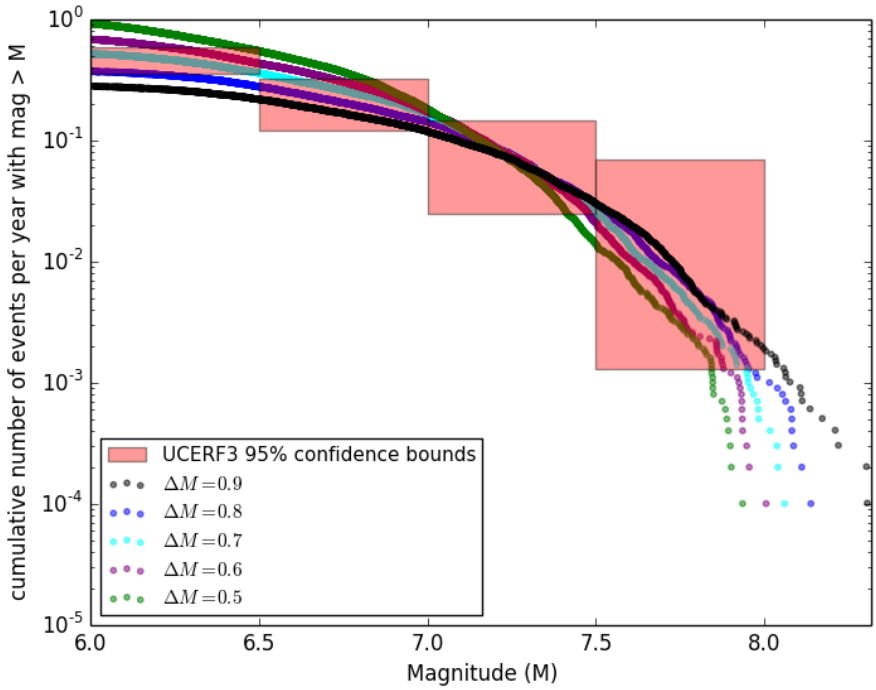
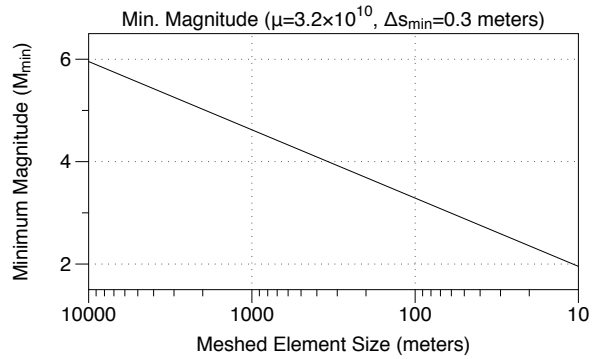


Figure 4.2: With the dynamic trigger factor set to $\eta = 0.5$, this is the effect of tuning the stress drop factor ΔM on the frequency-magnitude distribution. These are 10,000 year simulations of the UCERF3 fault model compared to observed California earthquake rates and 95% confidence bounds in red.

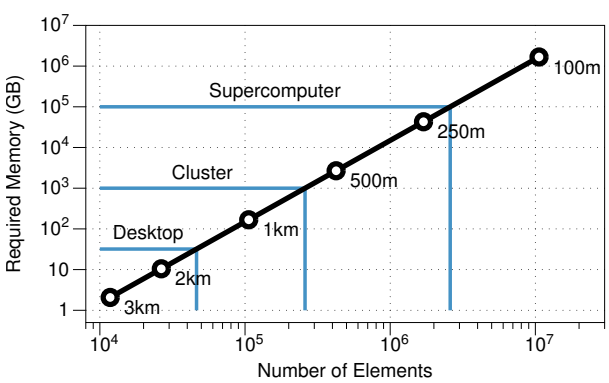
Operation	Minimum	Maximum
Addition	3,622	10,804
Multiplication	8,708	25,566
Square Root	260	780
Branch	1,148	3,924
Other	571	1,923
Total	14,309	42,997

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

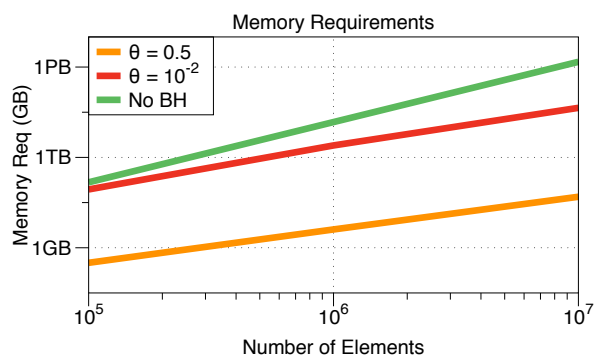


(b) Element size versus minimum event magnitude.

Figure 4.3: Number of elements determines computational cost - this will vary depending on the relative rake and dip. Element size governs simulated earthquake magnitude range.



(a) Element size determines computational regime.



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

Figure 4.4: Tradeoffs in element size and resource requirements.

of fault elements for a given simulation depends on the desired minimum earthquake magnitude and the available computational resources.

4.3.3 Element Size and Minimum Magnitude

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

$$M_{min} = \frac{2}{3} log_{10}(\mu s L^2) - 6.0 \tag{4.1}$$

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 2.1.3 the minimum slip distance is 0.3 meters. Figure 4.3b shows the relationship in VQ between element size and minimum earthquake magnitude.

The number of elements in a model will also affect the required memory. For non-approximated fault interaction in a model with N elements, the memory requirements are:

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

Chapter 5

Tutorials

5.1 Overview

These tutorials are meant to serve as a guide to some of the types of simulations VQ can perform. These cookbook examples are distributed with the package under the `examples` directory. Each tutorial is a self-contained lesson in how to use Virtual Quake. The tutorials increase in degree of complexity from one to the next. In the last section, we demonstrate how to construct the entire California fault system for large scale simulations.

5.1.1 Prerequisites

Before you begin any of the tutorials, you will need to install Virtual Quake following the instructions in Chapter 3 on page 31. If you do not wish to create your own mesh, the meshes are also provided as part of the tutorial.

5.2 Building a Fault Model

VQ uses a mesher program for fault model file manipulation. The mesher imports one or more trace files, manipulates them based on user arguments and exports one or more files to be used as input files for a VQ simulation. After following install instructions in Section 3.3.3, the mesher program is compiled to an executable located at `build/src/mesher`. This program is called by the shell scripts in the examples below to generate a fault model for VQ. See Section C on page 71 for more information on the options for the mesher program.

The next sections give examples and describe the model files that the mesher creates. For both files, each commented line describes the parameter in the corresponding column of the uncommented line immediately following the comments. See Section A on page 65 and Section B.1 on page 69 for more information on the input fault model files and parameters.

5.2.1 Trace File

The basic outline for a fault trace file is given in Section 2.1 on page 17. The trace file specifies the fault geometry and physical parameters. An example is printed below.

```
# fault_id: ID number of the parent fault of this section
# sec_id: ID number of this section
# num_points: Number of trace points comprising this section
# section_name: Name of the section
0 0 2 One_Fault_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
```

5.2.2 Example Traces

The `examples/fault_traces/` folder contains the fault traces used in the examples below. In addition, we provide trace files for all of California's major fault sections in the subfolder `fault_traces/ca_traces/`.

5.2.3 Friction Parameters

Virtual Quake 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 that 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 VQ to match established physical laws and empirical observations.

5.2.4 Simulation Parameter File

The main simulation input file is the parameter file. This file tells the simulator where the fault model files are located, sets various simulation variables, and specifies the output of the simulation. An example parameter file template is located in `examples/`. For all possible simulation parameters, see Appendix A.

5.2.5 Producing a Fault Model

The first step is to use `setup_mesh.sh` to generate the fault trace file, and the second step is to edit the simulation parameter file "params.d". This method is illustrated in the tutorials in the following sections.

5.2.6 Using the Mesher

The mesher is called on the command line. See Appendix C for all supported runtime options.

```
./mesher [options]
-s FILE, --print_statistics=FILE
Print statistics regarding final model to the specified file.
-m, --merge_duplicate_verts
Merge duplicate vertices after importing files.
-d, --delete_unused
Delete unused vertices after importing files.
-t METHOD, --taper_trace_method=METHOD
Specify the how to taper the imported faults when meshing.

FILE IMPORT
-i FILE, --import_file=FILE
Specify a model file to import and merge. Must have a paired import_file_type.
-j TYPE, --import_file_type=TYPE
Specify a model file type for importing. Must have a paired import_file.
-l SIZE, --import_trace_element_size=SIZE
Specify the element size (in meters) to use for trace file meshing. Must have a paired trace type file
```

```
-C FILE, --import_eqsim_condition=FILE
Specify an EQSim condition file to import for the model.
-F FILE, --import_eqsim_friction=FILE
Specify an EQSim friction file to import for the model.
-G FILE, --import_eqsim_geometry=FILE
Specify an EQSim geometry file to import for the model.

FILE EXPORT
-e FILE, --export_file=FILE
Specify a file to export the completed model to. Must have a paired export_file_type.
-f TYPE, --export_file_type=TYPE
Specify a file type to export the completed model. Must have a paired export_file.
-D FILE, --export_eqsim_condition=FILE
Specify an EQSim condition file to export for the model.
-R FILE, --export_eqsim_friction=FILE
Specify an EQSim friction file to export for the model.
-M FILE, --export_eqsim_geometry=FILE
Specify an EQSim geometry file to export for the model.
```

5.2.6.1 Taper Functionality

Specifying any tapering method other than `none` will result in element slip rates being reduced near the edges of faults. Slip rates are tapered both vertically, as the square root of the element's distance from the bottom of the fault normalized by the total fault depth, and horizontally for those elements within twelve kilometers of the ends of faults, as the square root of the element's distance from the fault end normalized by twelve kilometers. The `taper_renorm` tapering method will perform this tapering with an additional slip rate normalization factor to maintain total moment rate over the fault.

5.2.7 Parameter File

Copy the `example_params.d` file into the same folder as the mesher generated model. Edit the parameter file so the simulation parameters are set correctly for the current run.

5.2.8 Bounding the Greens Functions

In some cases, the mesher produces elements that have some small percentage of overlap. This overlap can cause the Greens functions (interaction coefficients) to take on extreme/non-physical values. If your simulation is producing anomalous earthquakes, or behaving oddly, try the following.

After using a test simulation to save your Greens functions to a file (e.g. "greens_values.h5"), you can use the script in vq/PyVQ/pyvq/betas/ called greens.py. Open a **python environment** from this directory, or in a python script execute the following to fit a Gaussian profile to the Greens functions, plot the distribution of values, and return the 1 standard deviation bounds. The values shown in example output below are the 1 standard deviation bounds (max, min) for the Greens functions. Issue the plt.show() command to reveal the plot of the Greens functions and the best fit Gaussian, as shown in Figure 5.1.

```
$ python
Python 2.7.9 (default, Dec 13 2014, 15:13:49)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from matplotlib import pyplot as plt
>>> import greens
>>> greens.plot_greens_hists(greens_fname="path/to/greens_values.h5", shear_normal="shear")
       [....some output omitted...]
       Greens range for 1 sigma ... x=[ 3661623.85  -831572.04]  ...
>>> plt.show()
```

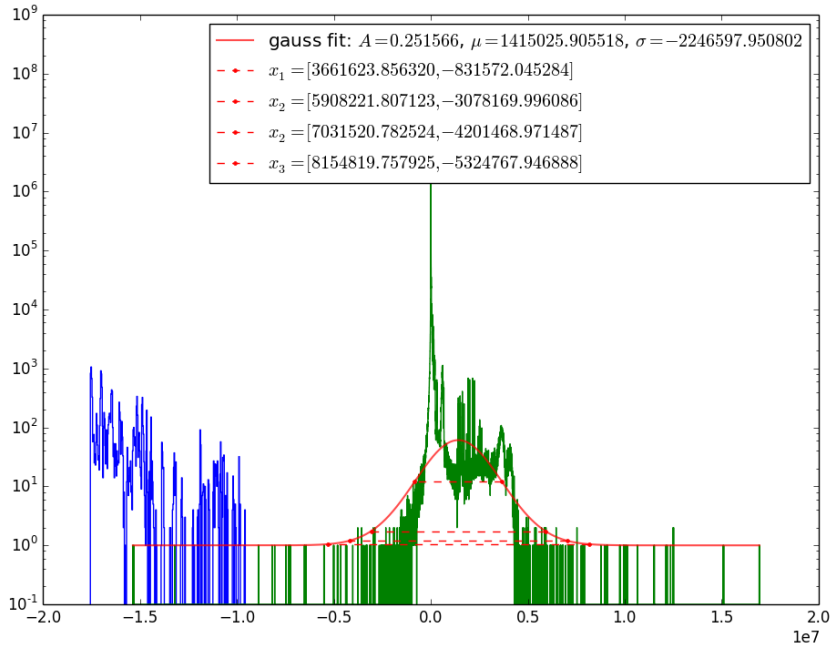


Figure 5.1: Example histogram of shear stress Greens function values with best fit Gaussian and bounds.

```
>>> greens.plot_greens_hists(greens_fname="path/to/greens_values.h5", shear_normal="normal")
       [....some output omitted...]
       Greens range for 1 sigma ... x=[ 61623.85  -31572.04] ...
>>> plt.show()
```

To implement these bounds for the shear stress Greens functions, in your simulation parameter file set the `sim.greens.shear_offdiag_max` equal the the 1 standard deviation maximum value given by greens.py (3661623.85 in the example above), and set `sim.greens.shear_offdiag_min` equal the the 1 standard deviation minimum value (-831572.04 in the example above). You can do the same for the normal stress bounds, setting `sim.greens.normal_offdiag_max` and `sim.greens.normal_offdiag_min`. All parameters are given in Appendix A.1.4.

You can also use the simulation parameter "sim.greens.offdiag_multiplier" to reduce the interactions in the Green's function matrices. The value can range from 0 to 1.

5.2.9 Periodically Saving the Simulation State

The Virtual Quake simulation can crash mid-simulation for a number of reasons, and while we do not anticipate your simulation to crash, it does happen. To prevent the loss of simulation data, and to avoid the situation of a simulation crashing 50,000 years in, leaving you to start your large simulation from the beginning, we have the ability to output the state of simulation.

In order to save the simulation state to an HDF5 file, for example (you can also save to text file), after every 10,000 earthquakes, simply add the following to your simulation parameter file:

```
sim.file.output_stress             = file_to_save_sim_state.h5
sim.file.output_stress_type         = hdf5
sim.file.output_stress_num_events = 10000
```

If you want to start a simulation after the last checkpoint was written for the previous simulation, simply run a new simulation with the following parameters:

```
sim.file.input_stress             = file_to_save_sim_state.h5
sim.file.input_stress_type        = hdf5
```

Lets say this process gives you two simulation files: one initial simulation that saved it's state before failing (let's call it initial_simulation.h5), and a second simulation file that continued the initial simulation by loading the saved state of the initial simulation (continued_simulation.h5). We can combine the events from these two simulation files with PyVQ and plot the results of the complete simulation by running the following example (to plot the magnitude-area distribution):

```
$ python path/to/vq/pyvq/pyvq.py --event_file initial_simulation.h5  --combine_file
continued_simulation.h5 --stress_file file_to_save_sim_state.h5
--plot_mag_rupt_area
```

More PyVQ examples can be found in Section 5.7, and a detailed list of stress simulation parameters (including text file I/O) can be found in Section A.1.5.

5.3 Single Element Tutorial

5.3.1 Overview

This tutorial is the simplest possible implementation of Virtual Quake. In this tutorial we will build the trace file for a single vertical strike-slip fault element, then use this to build a fault model with the mesher program and run this simulation for 10,000 years.

5.3.2 Creating Input Files

We are going to create a new fault trace file for this single element. The only data we need for this are the latitude and longitude of the endpoints of our element. For this example we will use the following coordinates (37.83,-122.4797) and (37.803,-122.4765). These endpoints give a 3km fault element that runs parallel to and directly under the Golden Gate Bridge in San Francisco. The trace file is printed below and the fault element is shown above ground in Figure 5.2.

```
# fault_id: ID number of the parent fault of this section
# sec_id: ID number of this section
# num_points: Number of trace points comprising this section
# section_name: Name of the section
0 0 2 One_Fault_GG_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)
37.83 -122.4797 0 3000 1 0 180 90 3e+10 3.2e+10
37.803 -122.4765 0 3000 1 0 180 90 3e+10 3.2e+10
```

To create the input files, we use the mesher program as follows (arguments explained in Section 5.2).

```
$ cd examples
$ mkdir golden_gate
$ cd golden_gate
```

Figure 5.2: Single 3km x 3km fault element running under the Golden Gate Bridge in San Francisco, shown above ground. This plot was generated by Google Earth from the mesher generated KML file.

```
$ ../../build/src/mesher \
--import_file=../fault_traces/golden_gate.txt \
--import_file_type=trace \
--import_trace_element_size=3000 \
--taper_fault_method=none \
--export_file=golden_gate_3000.txt \
--export_file_type=text \
--export_file=golden_gate_3000.kml \
--export_file_type=kml \
--print_statistics=statistics_3000.txt
```

This generates several output files as listed by the command line output shown below.

```
*** Summary of edits ***
File import ../fault_traces/golden_gate.txt with type trace... done.
File export golden_gate_3000.txt with type text... done.
File export golden_gate_3000.kml with type kml... done.
Print statistics to statistics_3000.txt
```

Trace values for the faults in California are taken from Ward's ALLCAL2 model [10], more information is available at http://scec.usc.edu/research/eqsims/documentation.html.

5.3.3 Edit Parameter File

The parameter file must be in the same directory (`examples/golden_gate/`). Make sure the parameter file `params.d` is as shown below:

```
sim.version                        = 2.0
sim.time.end_year                  = 10000
```

```
sim.greens.method                 = standard
sim.greens.use_normal             = false
sim.friction.dynamic              = 0.5
sim.file.input                    = golden_gate_3000.txt
sim.file.input_type               = text
sim.file.output_event             = events_3000.txt
sim.file.output_sweep             = sweeps_3000.txt
sim.file.output_event_type        = text
```

5.3.4 Running VQ

With the generated fault mesh and and parameter file we can run the simulation. To do so, call the vq executable (single processor) and pass the parameter file as a command line argument:

```
$ ../../build/src/vq ./params.d
```

You should see output similar to:

```
# *** MPI CPU count           : 1
# Initializing blocks.
# To gracefully quit, create the file quit_vq in the run directory.
# Calculating Greens function with the standard Okada class.
# Greens function took 0.00187707 seconds.
# Greens shear matrix takes 128 bytes.
# Greens normal matrix takes 128 bytes.
# To access the event output file during the simulation, pause
# by creating the file pause_vq.  Delete the file to resume.
# Writing events in format text to file events_3000.txt
# Plugin requested simulation end: Block stress calculation
 #                   Timer Name  AvgCounts        Min        Max       Mean     StdDev
 0                   Total Time          1      0.053      0.053      0.053      0.000
 1             Init and Cleanup          2      0.010      0.010      0.010      0.000
 2                 Comm Barrier          2      0.000      0.000      0.000      0.000
 3     Block stress calculation        748      0.003      0.003      0.003      0.000
 4     Propagate event ruptures        748      0.019      0.019      0.019      0.000
 5                  Data Writer        748      0.018      0.018      0.018      0.000
```

5.3.5 Results

The output data will be in `events_3000.txt` and `sweeps_3000.txt`. For more details about the file contents, see Appendix D. The results are not very exciting, there is one earthquake that occurs regularly due to constant backslip. But this illustrates setting up a Virtual Quake simulation start to finish.

5.4 Tutorial Using Multiple Elements

5.4.1 Overview

Now we will repeat the single vertical strike-slip fault tutorial from the previous section, but we will break up the fault into multiple elements.

5.4.2 Creating Input Files

For this tutorial, changing the element size to 1000 meters instead of 3000 meters will divide the fault into a 3 by 3 set of elements. The command to do so is shown below:

```
$ cd examples/golden_gate
$ ../../build/src/mesher \
--import_file=../fault_traces/golden_gate.txt \
--import_file_type=trace \
--import_trace_element_size=1000 \
--taper_fault_method=none \
--export_file=golden_gate_1000.txt \
--export_file_type=text \
--export_file=golden_gate_1000.kml \
--export_file_type=kml \
--print_statistics=statistics_1000.txt
```

Which should result in the following output:

```
*** Summary of edits ***
File import ../fault_traces/golden_gate.txt with type trace... done.
File export golden_gate_1000.txt with type text... done.
File export golden_gate_1000.kml with type kml... done.
Print statistics to statistics_1000.txt
```

We will copy the previous `params.d` file to `params_multiple.d` and edit it to match the new file names, as shown below:

```
sim.version                       = 2.0
sim.time.end_year                 = 10000
sim.greens.method                 = standard
sim.greens.use_normal             = false
sim.friction.dynamic              = 0.5
sim.file.input                    = golden_gate_3000.txt
sim.file.input_type               = text
sim.file.output_event             = events_3000.txt
sim.file.output_sweep             = sweeps_3000.txt
sim.file.output_event_type        = text
```

5.4.3 Running VQ

Again we run the simulation by calling the vq executable:

```
$ ../../build/src/vq ./params_multiple.d
```

Output should be similar to the previous simulation:

```
# *** MPI CPU count           : 1
# Initializing blocks.
# To gracefully quit, create the file quit_vq in the run directory.
# Calculating Greens function with the standard Okada class.
# Greens function took 0.00388789 seconds.
# Greens shear matrix takes 1.125 kilobytes.
# Greens normal matrix takes 1.125 kilobytes.
# To access the event output file during the simulation, pause
# by creating the file pause_vq.  Delete the file to resume.
# Writing events in format text to file events_1000.txt
# Plugin requested simulation end: Block stress calculation
 #                   Timer Name  AvgCounts        Min        Max       Mean     StdDev
 0                   Total Time          1      0.228      0.228      0.228      0.000
 1             Init and Cleanup          2      0.005      0.005      0.005      0.000
```

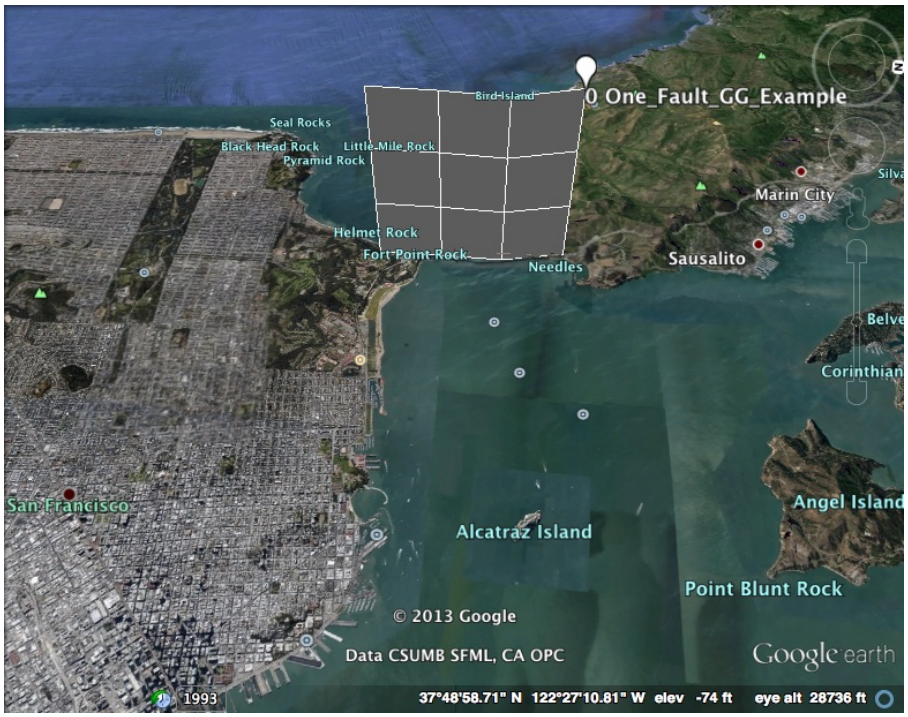


Figure 5.3: The same 3km x 3km fault section from figure 5.2 but meshed into 1km x 1km elements. This plot was generated by Google Earth from the mesher output KML file.

```
 2                Comm Barrier          2      0.000      0.000      0.000      0.000
 3     Block stress calculation        748      0.009      0.009      0.009      0.000
 4     Propagate event ruptures        748      0.124      0.124      0.124      0.000
 5                  Data Writer        748      0.089      0.089      0.089      0.000
```

Note the Greens shear and normal matrices now require 1.125 KB of memory instead of 128 bytes of memory as in the previous simulation. This is because the simulation now has 9 elements and a correspondingly larger Green's matrix.

5.4.4 Results

The output data is stored in `events_1000.txt` and `sweeps_1000.txt`. For more details about the file contents, see Appendix D.

5.5 Multiple Fault Trace Points

5.5.1 Overview

This tutorial explores a VQ simulation involving a fault section defined by three trace points — each segment 15km long and 12km deep meshed into 3km by 3km elements. We will also use larger fault elements than the previous example, which raises the lower bound for magnitudes in the output data set. Furthermore, we now will be simulating the interaction between 40 elements so the computational resource requirements grow as well (see Section 4.3.2 for a detailed discussion).

5.5.2 Input Files

We will use another trace file that is provided with the VQ package. This trace file specifies two neighboring vertical strike slip fault segments that intersect at the Earth and Physical Sciences building on campus at the University of

California, Davis, shown in Figure 5.4. We generate the mesh with the following commands.

```
$ cd examples
$ mkdir two_fault
$ cd two_fault
$ ../../build/src/mesher \
--import_file=../fault_traces/multiple_fault_trace.txt \
--import_file_type=trace \
--import_trace_element_size=3000 \
--taper_fault_method=none \
--export_file=two_fault_3000.txt \
--export_file_type=text \
--export_file=two_fault_3000.kml \
--export_file_type=kml \
--print_statistics=statistics_3000.txt
```

Which should output the following.

```
*** Summary of edits ***
File import ../fault_traces/multiple_fault_trace.txt with type trace... done.
Computing stress drops with stress_drop_factor=0.3
File export two_fault_3000.txt with type text... done.
File export two_fault_3000.kml with type kml... done.
Print statistics to statistics_3000.txt
```

Next, create the `params.d` file as below:

```
sim.version                      = 2.0
sim.time.end_year                = 10000
sim.greens.method                = standard
sim.greens.use_normal            = false
sim.friction.dynamic             = 0.5
sim.file.input                   = two_fault_3000.txt
sim.file.input_type              = text
sim.file.output_event            = events_3000.txt
sim.file.output_sweep            = sweeps_3000.txt
sim.file.output_event_type       = text
```

5.5.3 Running VQ

Again we run the simulation on a single processor by simply executing the vq executable:

```
$ ../../build/src/vq ./params.d
```

To instead run VQ in parallel on 2 or more processors/cores with MPI, use:

```
$ mpirun -np 2 ../../build/src/vq ./params.d
```

The output from an MPI simulation should be identical that from a single processor simulation.

5.5.4 Results

The output data is stored in `events_3000.txt` and `sweeps_3000.txt`. For more details about the file contents, see Appendix D.

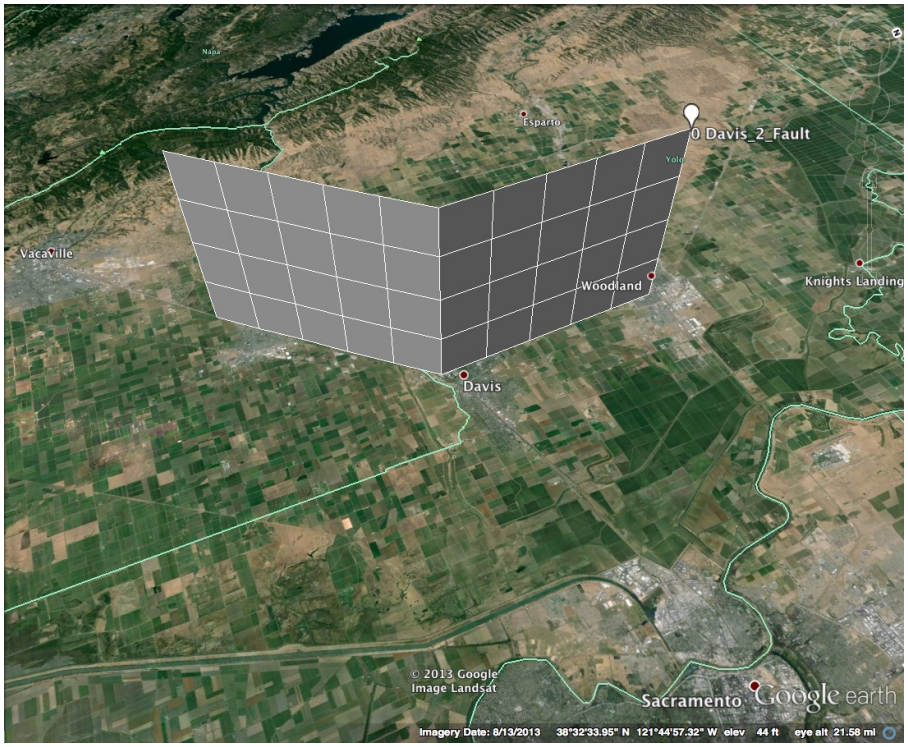


Figure 5.4: Two fault sections that meet at the UC Davis campus. The fault sections are 15km x 12km and meshed into 3km x 3km elements. This plot was generated by Google Earth from the mesher output KML file.

5.6 Building the full California Fault Model

5.6.1 Overview

In the examples folder we also provide all the major fault traces in California and an executable script that can use them to build a full fault model. To build the full California fault model, execute the following commands to mesh the model into 3km elements:

```
$ cd examples/ca_model/
$ ./gen_ca_model.sh 3000
```

Next, create an appropriate parameter file and run VQ as shown in the previous sections. Note that the full CA simulation requires a fair amount of memory and may take several hours to finish. Progress will be reported during the simulation if the `sim.system.progress_period` parameter is set to a nonzero value (unit is seconds).

5.7 Virtual Quake Data Analysis Tutorial (PyVQ)

5.7.1 Overview

Section 2.5 showed plots of gravity changes and an InSAR interferogram for a selected simulated earthquake. These plots that analyze simulation data utilize the python interface to the QuakeLib library, specifically a python module named quakelib that is generated during the final step of compiling Virtual Quake (when using the command "sudo make install", see section 3.3.3). In the following sections we describe how to make plots with Virtual Quake data using the pyvq script in the pyvq directory.

The set of simulation data analysis plotting scripts are located in the pyvq folder, and accessing by calling the python script pyvq.py with various command line arguments. The functionality is briefly explained in the README_PYVQ file and discussed in further detail here. **The full list of possible PyVQ parameters is given in Section 5.8**.

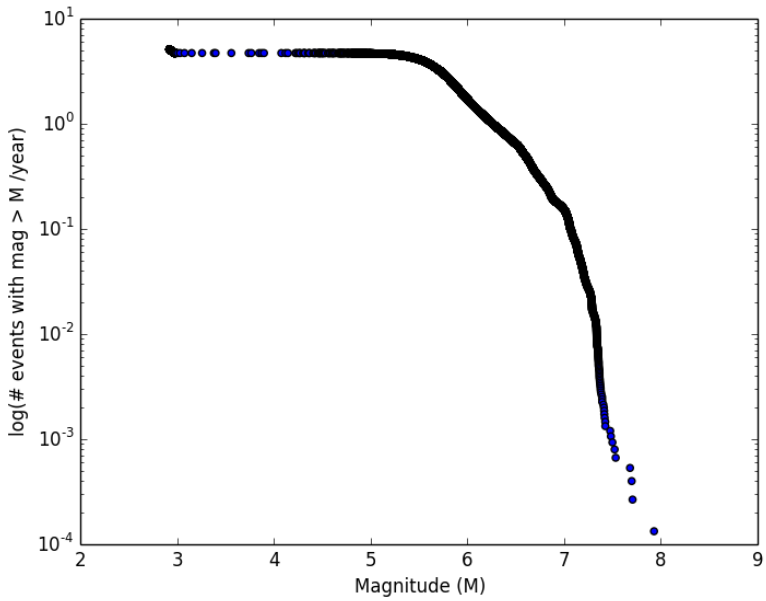


Figure 5.5: Example frequency-magnitude distribution.

5.7.2 Simulation Statistics Plots

To generate a frequency magnitude plot from your simulation file (hdf5), located at "path/to/sim_file.h5", simply execute the shell command (in one line):

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5   --plot_freq_mag 1
```

If you generated a text file from your simulation, simply use instead:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.txt  --plot_freq_mag 1
```

This saves the plot to a file with the filename chosen from the name of the simulation file and the type of plot being drawn, Figure 5.5.

To plot both the magnitude-rupture area and magnitude-mean slip distributions (Figure 5.6) simply execute:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5
      --plot_mag_rupt_area --plot_mag_mean_slip
```

5.7.3 Overview Plots/Functions

To list the largest N (an integer) earthquakes (by magnitude) and print their properties, execute the following:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --summary N
```

To plot all the statistical plots for a simulation, e.g. the frequency-magnitude, the mean slip vs .magnitude, rupture area vs. magnitude, and other distributions, execute the following:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --all_stat_plots
```

or you can execute the following to plot all the statistical plots and some additional time series plots:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --diagnostics
```

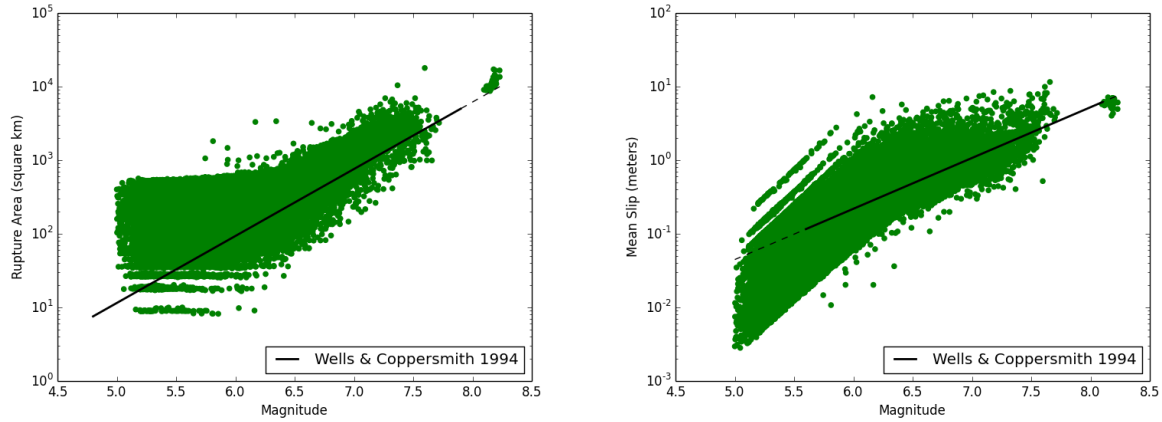


Figure 5.6: **Left**: Example magnitude vs rupture area distribution. **Right**: Example magnitude vs mean slip distribution. Compare these to Wells and Coppersmith 1994 relations.

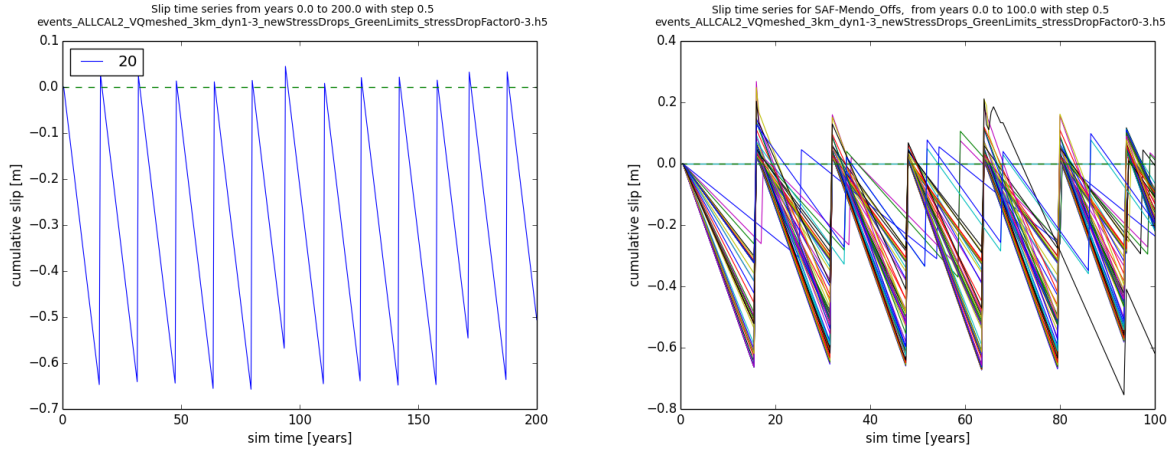


Figure 5.7: **Left**: Slip time series for element #20 **Right**: Slip time series for all elements in section 0, SAF-Mendo_Offs.

5.7.4 Time Series Plots

To plot the slip time series of element 20 from year 0 to year 100 shown in figure 5.7 (use the –dt parameter to set the time step in units of years):

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --slip_time_series
--elements 20 --min_year 0 --max_year 200 --dt 0.5
```

And to plot the slip time series for all elements in section 0 (a.k.a. ”fault 0”) from year 0 to year 100 shown in figure 5.7:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --slip_time_series
--use_sections 0 --min_year 0 --max_year 100  --dt 0.5
```

5.7.5 Space-Time Plots

To visualize when and where earthquake sequences occur during the simulation on a particular fault, we will generate space-time plots. The vertical axis is the simulation time in years, the horizontal axis is the distance along strike, the

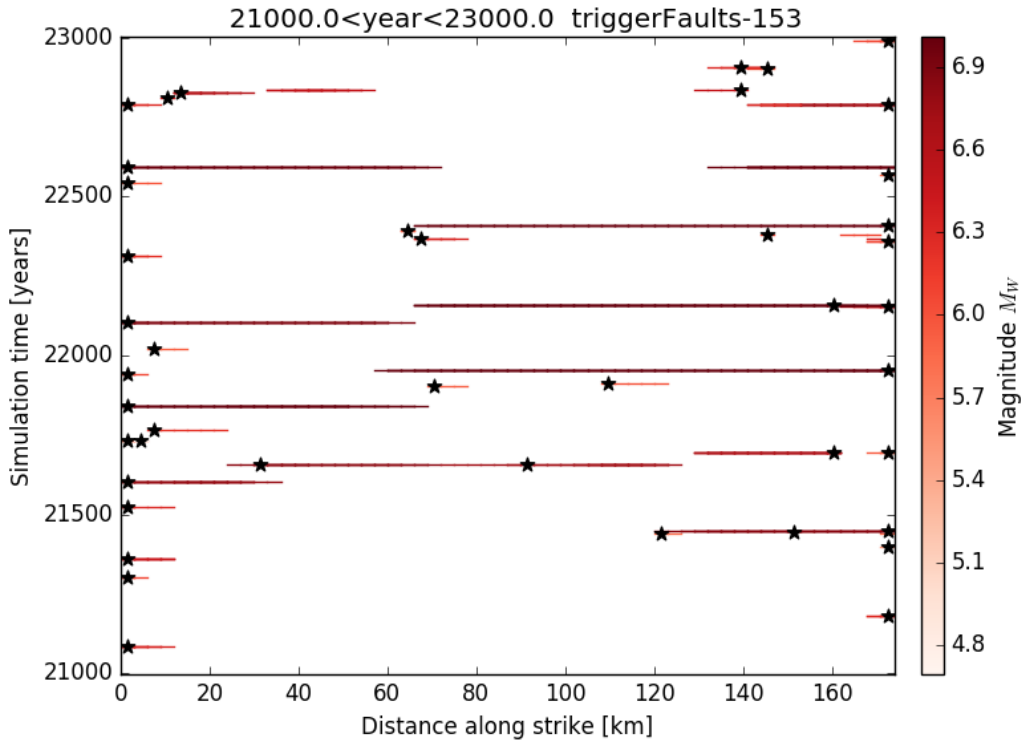


Figure 5.8: Space-time plot for simulated earthquakes from year 21000 to year 23000 on fault 153.

color will indicate event magnitude, lines are plotted to indicate which fault elements slipped during each earthquake, and black stars denote the rupture initiation location for each event. To generate a space-time plot like Figure 5.8, execute the following command that includes which fault (153) to plot and the simulation time range to plot (21000-23000 years):

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --model_file
path/to/model_file.txt   --use_faults 153 --min_year 21000 --max_year 23000
--spacetime
```

5.7.6 Earthquake Probability Plots

To plot the conditional probabilities (Figure 5.9) of an earthquake with magnitude ≥ 7 occurring on sections [4,5,6,7,8,9,10,11,12,13] (as defined in your fault model) as a function of time since the last earthquake on those sections:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5
     --plot_cond_prob_vs_t  --model_file path/to/model.txt --model_file_type
     'text' --min_magnitude 7.0 --use_sections 4 5 6 7 8 9 10 11 12 13
```

To plot the expected waiting times until the next earthquake with magnitude ≥ 7 occurring on sections [4,5,6,7,8,9,10,11,12,13] as a function of time since the last earthquake on those sections:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5
     --plot_waiting_times  --model_file path/to/model.txt --model_file_type
     'text' --min_magnitude 7.0 --use_sections 4 5 6 7 8 9 10 11 12 13
```

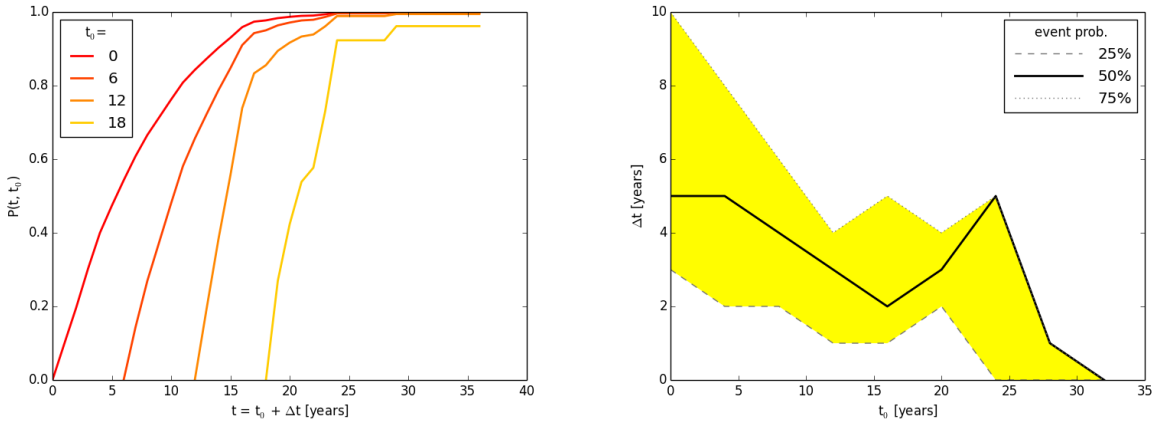


Figure 5.9: **Left**: Conditional probability of an earthquake $M \geq 7$ on the specified sections as a function of time since the last earthquake $M \geq 7$. **Right**: Distribution of waiting times until the next earthquake $M \geq 7$ for waiting times with 25%, 50% and 75% probability.

		1 year	5 years	20 years
N=25475	$M > 5$	0.41	0.95	1.00
N=23774	$M > 6$	0.39	0.93	1.00
N=7495	$M > 7$	0.14	0.53	0.96

You can also print out a table of conditional earthquake probabilities if you know the observed time since the last earthquakes on your faults. To print out conditional probability table for $M > 5$, $M > 6$ and $M > 7$ for the next 1yr, 5yr and 20yr intervals execute the following command (here we are using observed time since the last California earthquakes for the –t0 values). The results are shown in the table above.

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5
     --probability_table --t0 1.0 1.4 5.8 --t 1 5 20 --magnitudes 5 6 7
```

5.7.7 Plotting Data on a Map and Event Field Plots

To plot the traces of your faults on a simple map:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --model_file
     path/to/model.txt --traces
```

To create a Google Earth file (KML) of the elements involved in an event, colored by their event slips (shown in Figure 5.10):

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --model_file
     path/to/model.txt --event_kml --event_id 0
```

QuakeLib uses Green's functions to compute co-seismic fields given a fault geometry and a slip distribution. For displacements we use Okada's equations and for gravity, potential, and geoid height changes we use Okubo's equations.

To compute the gravity changes for 5 meters of uniform slip on your fault model (example shown in Figure 5.11 for single 10km by 10km faults):

```
$ python path/to/vq/pyvq/pyvq.py --model_file path/to/model.txt --model_file_type
     'text' --uniform_slip 5 --field_plot --field_type 'gravity'
```

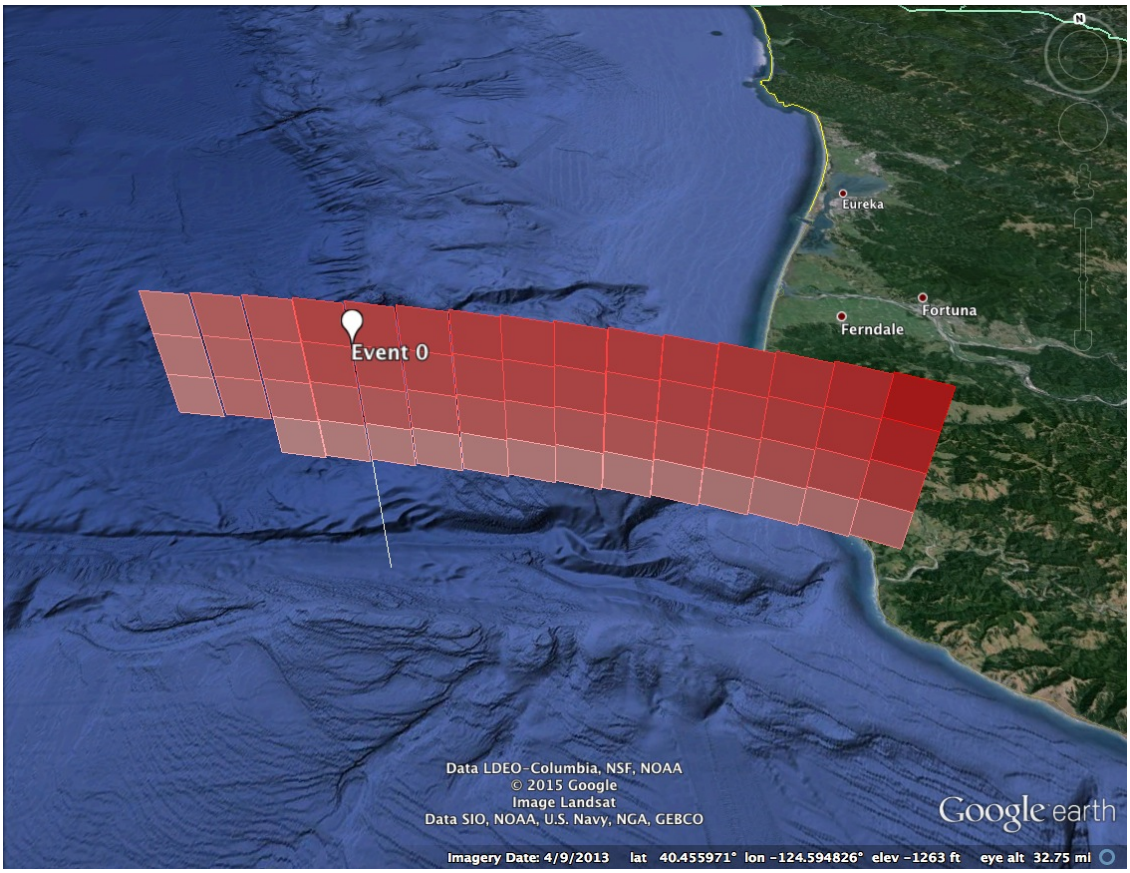


Figure 5.10: Slips for event 0. The triggering element is shown with the white place marker. Color scale is white (almost zero slip) to red (max slip).

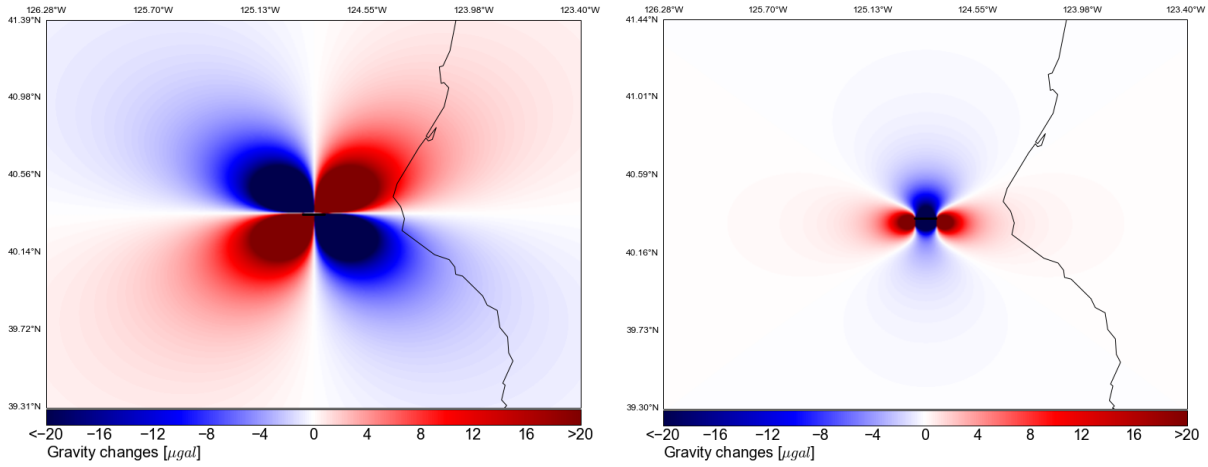


Figure 5.11: **Left**: Gravity changes for 5m slip on a 10km by 10km vertical strikeslip fault. **Right**: Gravity changes for 1m slip on a 10km by 10km normal fault with dip angle 30 degrees.

To compute the gravity changes and InSAR interferogram (observing with L-band 21cm) for event #210 from your simulation (Figures 5.12 and 5.13):

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --model_file
     path/to/model.txt --model_file_type 'text' --field_plot
     --field_type 'gravity' --event_id 210

$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5   --model_file
     path/to/model.txt --model_file_type 'text' --field_plot
     --field_type 'insar' --event_id 210 --wavelength 0.21
```

To evaluate an event field at specified lat/lon points, you must provide the file containing lines of lat/lon pairs (–lld_file), the result is written to a file:

```
$ python path/to/vq/pyvq/pyvq.py --event_file path/to/sim_file.h5  --model_file
     path/to/model.txt --model_file_type 'text' --field_eval
     --field_type 'gravity' --event_id 210 --lld_file path/to/lld_file.txt
```

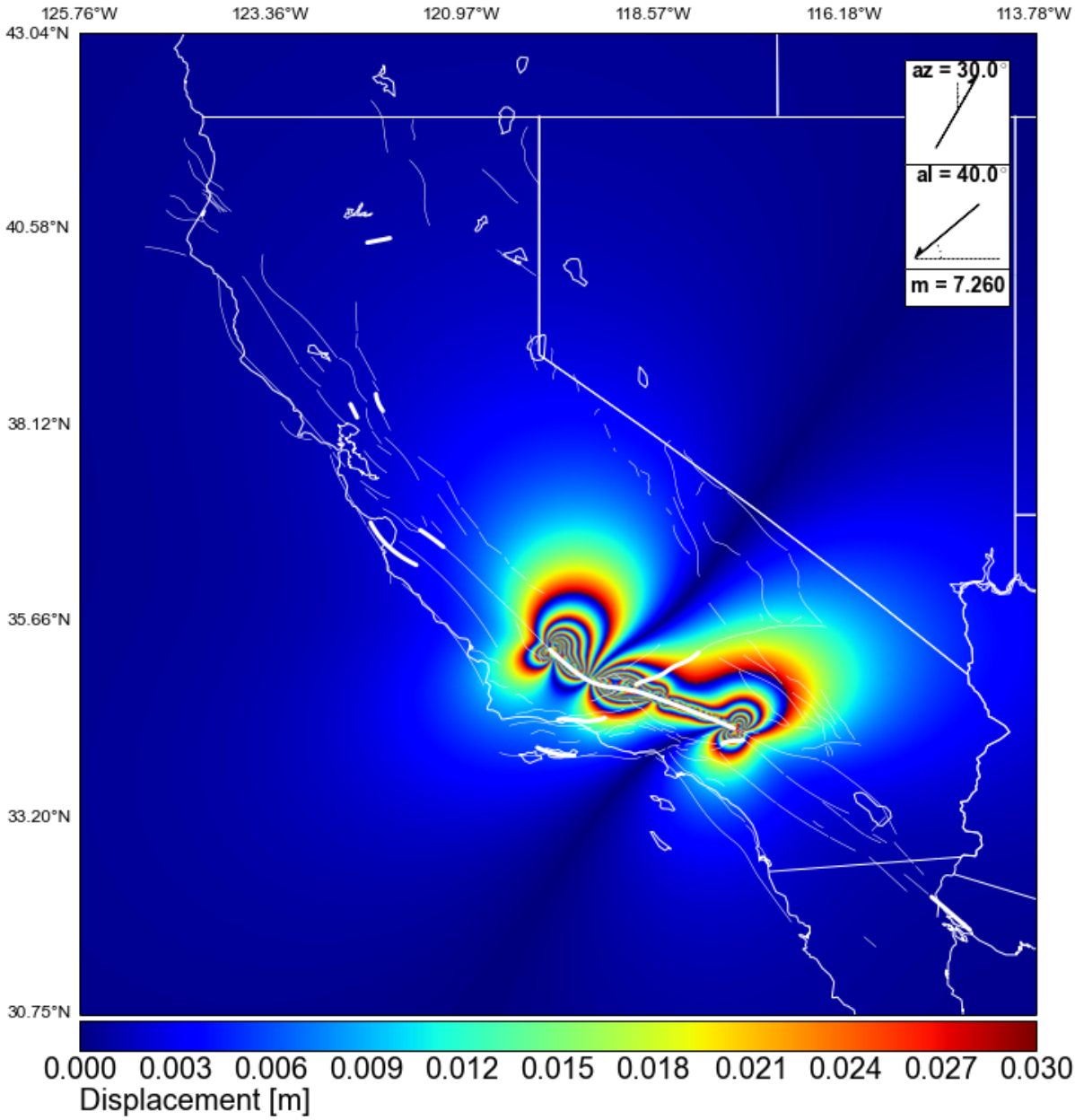


Figure 5.12: Simulated InSAR interferogram for magnitude 7.26 earthquake on the San Andreas Fault.

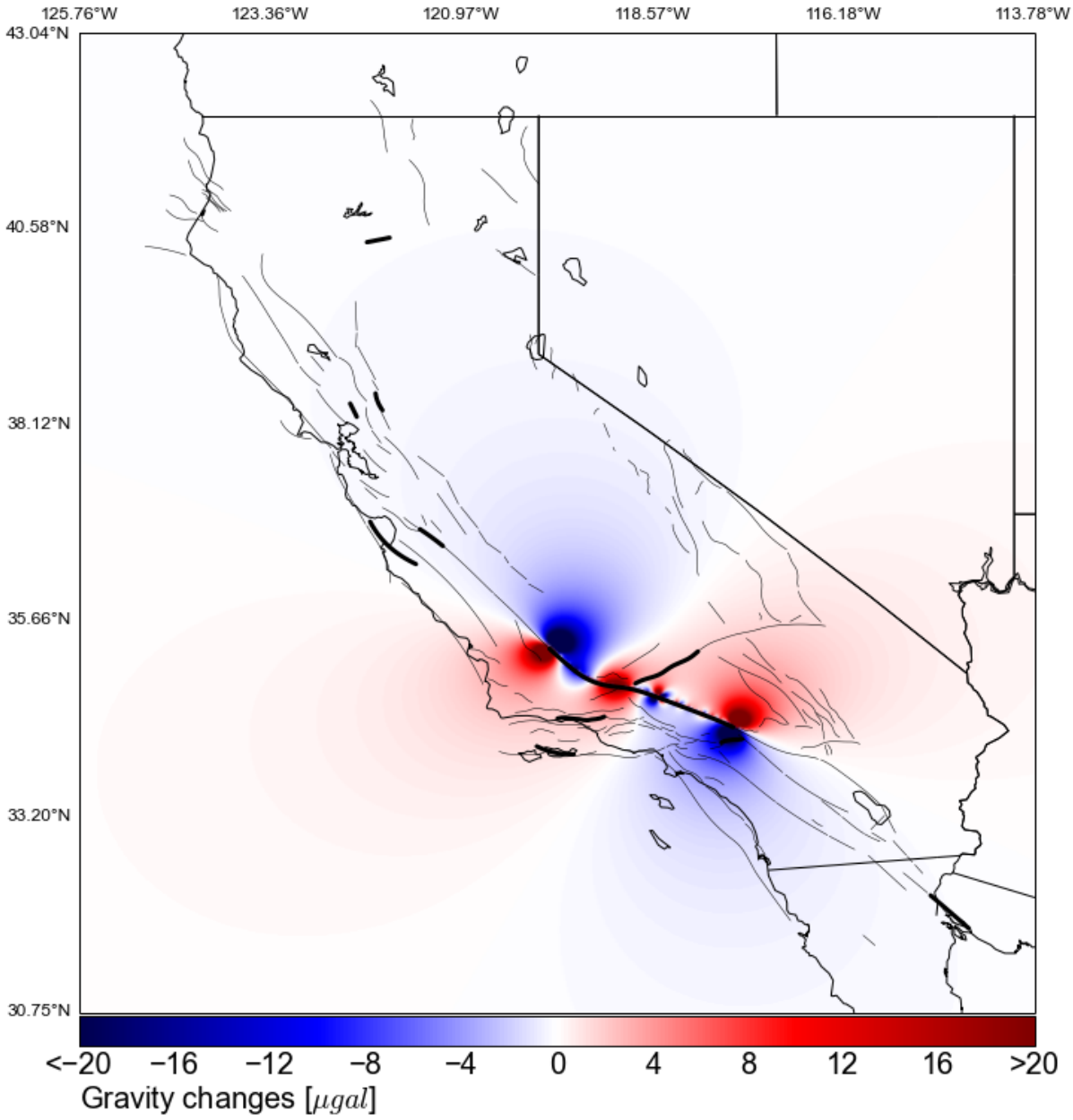


Figure 5.13: Simulated gravity changes for magnitude 7.26 earthquake on the San Andreas Fault.

5.8 PyVQ Plotting Arguments Grouped by Functionality

This section summarizes the current functionality of the pyvq script (section 5.7). These parameters do not have default values, and are not used unless specified.

5.8.1 Model File Parameters

`--event_file`	The simulation output file.
`--sweep_file`	The simulation sweep file, required only if simulation file is text.
`--model_file`	The fault model file.
`--model_file_type`	Either 'hdf5' or 'text'.
`--stress_file`	hdf5 file containing simulation stress state information.
`--combine_file`	An hdf5 simulation file whose events will be combined with those from –event_file. Must also specify –stress_file.

5.8.2 Subsetting/Filtering Parameters

These parameters are used to select subsets of the faults or subsets of events.

`--use_faults`	Select a subset of events that are triggered on the specified faults from the model file.
`--use_sections`	Select a subset of events that are triggered on the specified fault sections from the model file. To select sections 23, 43, and 101 it would be "–use_sections 23 43 101".
`--min_magnitude`	Select a minimum magnitude for earthquakes to be analyzed. To select earthquakes only above magnitude 5.5 it would be "–min_magnitude 5.5".
`--max_magnitude`	Select a maximum magnitude for earthquakes to be analyzed. Can be used in conjunction with –min_magnitude to specify a range of magnitudes.
`--min_year`	Select a minimum year for a time interval subset. Can be used in conjunction with –max_year to specify a simulation time range.
`--max_year`	Select a maximum year for a time interval subset. Can be used in conjunction with –min_year to specify a simulation time range.
`--min_slip`	Select a minimum mean slip threshold for plotting event data, units are meters.
`--max_slip`	Select a maximum mean slip threshold for plotting event data, units are meters.

5.8.3 Statistical Plotting Parameters

`--summary n`	List the n earthquakes with the largest magnitude, also prints their event mean slip, rupture area, etc.
`--plot_freq_mag n`	Frequency-Magnitude. n=1 Normal plot, n=2 adds a Gutenberg-Richter line with b=1, n=3 adds UCERF2 observed seismicity rates in California, n=4 adds both the b=1 line and observed California rates.
`--plot_mag_rupt_area`	Magnitude vs Rupture Area scaling. Compare to Wells and Coppersmith 1994.
`--plot_mag_mean_slip`	Magnitude vs Mean Slip scaling. Compare to Wells and Coppersmith 1994.
`--all_stat_plots`	Plot all statistical plots.
`--wc94`	Add Wells and Coppersmith 1994 scaling relation to Mean Slip or Rupture Area plots.
`--diagnostics`	Plot many different statistical plots and time series plots to characterize a simulation.

5.8.4 Time Series Plotting Parameters

`--slip_time_series`	Plot the slip time series for a single element up to an entire fault, use with –elements or –use_sections.
`--dt`	Time step in decimal years for slip time series plots.
`--fault_time_series`	Plot the slip time series for an entire fault, use with –use_faults. Ex. "–use_faults 1" to plot fault time series for fault #1. You can specify multiple faults, and it saves one file for each fault time series.
`--fault_group_time_series`	After using –fault_time_series to save fault time series data for each fault, make a plot of time series averaged over groups of faults by specifying the fault time series files for each group of faults with –group1_files and –group2_files.
`--group1_files`	"List of files containing the fault slip time series. Must also specify –group2_files. These subsets are used for plotting fault group average time series.
`--group2_files`	List of files containing the fault slip time series. Must also specify –group1_files. These subsets are used for plotting fault group average time series.
`--num_sweeps`	Plot, for each event, the number of event sweeps as a function of simulation time.
`--event_mean_slip`	Plot, for each event, the mean slip as a function of simulation time.
`--event_shear_stress`	Plot, for each event, the fractional change in shear stress as a function of simulation time.
`--event_normal_stress`	Plot, for each event, the fractional change in normal stress as a function of simulation time.

5.8.5 Probability Plotting Parameters

These parameters can be used in combination with "–use_sections", "–min_magnitude", and "–max_magnitude" to select subsets of your fault model and events for analysis.

`--plot_prob_vs_t`	Plot probability of earthquake as a function of time since the last earthquake.
`--plot_prob_vs_t_fixed_dt DT`	Plot probability of earthquake in the next DT years as a function of time since the last earthquake. E.g. "–plot_prob_vs_t_fixed_dt 30" plots the probability of an earthquake in the next 30 years as a function of time.
`--plot_cond_prob_vs_t`	Plot conditional probability of earthquake as a function of time since the last earthquake plotted at various times after the last earthquake to show the evolution of the distribution.
`--plot_waiting_times`	Plot the waiting times until the next earthquake as a function of time since the last earthquake, for waiting times with 25%, 50% and 75% probability.
`--beta`	Beta parameter for the Weibull distribution, must also specify tau. If beta and tau are specified, the corresponding Weibull distribution is drawn in the "–plot_cond_prob_vs_t" plot.
`--tau`	Tau parameter for the Weibull distribution, must also specify beta.
`--probability_table`	Prints out probabilities of $M > 5$, $M > 6$ and $M > 7$ earthquakes. Must also specify –t0.
`--t0`	List of times [rounded to the nearest 0.1 years] since the last observed earthquakes of a given magnitude on the modeled faults. Must also specify the same number of –magnitudes arguments. Example "–t0 4.1 12.5 35.9 –magnitudes 5 6 6.5" for the probabilities of $M > 5$, $M > 6$ and $M > 6.5$. Can request specific number of years in the future to evaluate the probabilities with –t.
`--magnitudes`	List of magnitudes to compute conditional probability table for. Must also specify the same number of –t0 arguments, where the i-th –t0 value is the time elapsed since the last observed earthquake with magnitude M = i-th –magnitudes value. Example "–t0 4.1 12.5 35.9 –magnitudes 5 6 6.5" for the probabilities of $M > 5$, $M > 6$ and $M > 6.5$. Can request specific number of years in the future to evaluate the probabilities with –t.
`--t`	Number of years into the future to evaluate conditional probabilities. E.g. "–t 1 5 10" will evaluate the conditional probabilities for earthquakes with magnitudes given by –magnitudes, with the last-observed earthquake of each magnitude occurring at each value of –t0. Example "–t0 4.1 12.5 35.9 –magnitudes 5 6 6.5" for the conditional probabilities of $M > 5$, $M > 6$ and $M > 6.5$ at 1, 5 and 10 years from present if the last earthquakes of each magnitude were –t0 years ago.
`--fit_weibull`	Fit the weibull distribution to the probability plots and plot the best fitting Weibull distribution.
`--plot_recurrence`	Plot a histogram of recurrence times. This is best used with subsetting commands, selecting a particular fault section(s) with –use_sections or a particular fault(s) with –use_faults.

5.8.6 Field Plotting Parameters

For any field plot you must specify "–model_file" and "–model_file_type". To plot co-seismic fields, you must also specify an "–event_file", "–event_file_type", and "–event_id". To plot the field resulting from uniform slip across the entire fault model you must also specify "–uniform_slip".

`--field_plot`	Required to plot any fields
`--field_type`	The type of co-seismic field to plot. Options are: “gravity”, “satellite_gravity”,, “dilat_gravity” (dilatational), “displacement”, “insar”, “potential” (gravitational potential), “geoid” (geoid height changes).
`--wavelength`	The observing wavelength for the InSAR interferogram, default is 21cm.
`--no_mask`	This option turns off the ocean-masking for displacement plots.
`--event_id`	The number of the event to plot.
`--uniform_slip`	The slip to apply to every fault element in the fault model, in meters.
`--colorbar_max`	The maximum value for the colorbar scale on field plots (excluding insar and displacement). E.g. to limit the gravity field plots to the color range -20 microGal to 20 microGal you would specify “–colorbar_max 20”.
`--angles`	The observing angles for displacements and InSAR, the field is projected along this direction. Must specify azimuth and elevation angles in degrees. E.g. “–angles 20.5 66.2”.
`--field_eval`	Evaluate an event field at given lat/lon points. Must specify –field_type, –lld_file. Results are saved to file
`--lld_file`	Text file containing lat/lon pairs in columns. Used for field evaluations.

5.8.7 Geometry/Model Plotting Parameters

For any geometry plot you must specify “–model_file”.

`--spacetime`	Create and save a Space-Time plot for a particular fault over a particular simulation time range. Must also specify –min_year, –max_year, and –use_faults.
`--event_kml`	Create and save a Google Earth (KML) file that includes elements that slip during the specified event, colored by the amount of event slip. Must also specify –event_id.
`--block_area_hist`	Create and save a histogram of fault element areas. To present numbers relative to a reference value, must specify –reference .
`--block_length_hist`	Create and save a histogram of fault element lengths (square root of area). To present numbers relative to a reference value, must specify –reference .
`--block_aseismic_hist`	Create and save a histogram of fault element aseismic fractions (percent of fault slip that is aseismic).
`--fault_length_hist`	Create and save a histogram of fault lengths . To present numbers relative to a reference value, must specify –reference .
`--fault_length_distribution`	Create and save the cumulative distribution of fault lengths.
`--reference`	A reference number or unit value for `--block_length_hist` or `--block_area_hist` plots.
`--traces`	Plot the fault traces on a simple map.

5.8.8 Miscellaneous Plotting Parameters

`--dpi`	The dots-per-inch resolution of the figures to be saved. E.g. –dpi 300.
`--pdf`	Save figures as PDF instead of the default PNG.
`--eps`	Save figures as EPS instead of the default PNG.
`--no_titles`	Save figures without titles.

Part III

Appendices

Appendix A

Input Parameters for Virtual Quake

A.1 Input Parameters Grouped by Functionality

This section explains the meaning of the input parameters for Virtual Quake. 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.sanity_check = false`	Whether to perform sanity checks on simulation values each time step 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 comparative performance profiling.
`sim.system.progress_period = 0`	How frequently (in wall time seconds) to display simulation progress. If undefined or <= 0, simulation progress will not be displayed.

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.

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. If unspecified, Green's function values are not written to a file after being calculated.
`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 elements greater than this distance (in km) apart. If undefined or <= 0, all interactions will remain the same.
`sim.greens.sample_distance = 1000.0`	When calculating the Green's function, take samples at this minimum distance between samples. This allows better convergence between models with few large elements and models with many small elements. If the element size is smaller than this value, it has no effect.
`sim.greens.offdiag_multiplier = 1.0`	If specified, this multiplies the interaction Greens function values by a factor between 0 and 1.
`sim.greens.shear_offdiag_min`	Double, no default value. If specified, this truncates the shear interaction Greens function values to some minimum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.shear_offdiag_max`	Double, no default value. If specified, this truncates the shear interaction Greens function values to some maximum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.shear_diag_max`	Double, no default value. If specified, this truncates the shear self-stress Greens function values to some maximum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.shear_diag_min`	Double, no default value. If specified, this truncates the shear self-stress Greens function values to some minimum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.normal_offdiag_min`	Double, no default value. If specified, this truncates the normal interaction Greens function values to some minimum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.normal_offdiag_max`	Double, no default value. If specified, this truncates the normal interaction Greens function values to some maximum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.normal_diag_max`	Double, no default value. If specified, this truncates the normal self-stress Greens function values to some maximum. Sometimes required if the mesher puts fault elements too close together.
`sim.greens.normal_diag_min`	Double, no default value. If specified, this truncates the normal self-stress Greens function values to some minimum. Sometimes required if the mesher puts fault elements too close together.

A.1.5 File I/O parameters

`sim.file.input`	The input model file of the simulation.
`sim.file.input_type`	The input model file format - must be one of text or hdf5.
`sim.file.output_event`	The file which events will be written to. If undefined, events will not be recorded.
`sim.file.output_sweep`	The file which sweeps will be written to. If undefined, sweeps will not be recorded.
`sim.file.output_event_type`	The file format to output events - must be one of text or hdf5. If hdf5, sweep and event information will be printed to the same file specified by `sim.file.output_event`.
`sim.file.output_stress`	The file to output the stress state after a specified number of events, specified by `sim.file.output_stress_num_events`.
`sim.file.output_stress_type`	The file type for stress output, either text or hdf5. If text, must also specify `sim.file.output_stress_index`.
`sim.file.output_stress_index`	The text file name to write the stress state information. Must be used with `sim.file.output_stress`.
`sim.file.output_stress_num_events`	The number of events before each stress state output.
`sim.file.input_stress`	The stress state file used to set initial stresses.
`sim.file.input_stress_type`	The file type for stress input, either text or hdf5. If text, must also specify `sim.file.input_stress_index`.
`sim.file.input_stress_index`	The text file name to load stress state information. Must be used with `sim.file.input_stress`.

A.1.6 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

Appendix B

Virtual Quake Input Model Format

B.1 Trace File Format

The initial fault geometry for VQ 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 VQ model for a simulation can be generated by combining multiple fault traces using the mesher program (see Section 5.4 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.
`sec_id`	ID number of this section.
`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. In the simulation they are converted to SI units though this is hidden from the user.

`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 δ the actual depth of the fault will be depth_along_dip*sin δ.
`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

Mesher Program Options

C.1 Mesher Options Grouped by Functionality

This section explains the meaning of the options used by the mesher program for Virtual Quake model file manipulation. These options are grouped by their functionality.

C.1.1 General Options

-s FILE, --print_statistics=FILE	Print statistics regarding final model to the specified file.
-m, --merge_duplicate_verts	Merge duplicate vertices after importing files.
-d, --delete_unused	Delete unused vertices after importing files.
-q --stress_drop_factor	Default is 0.3, this factor is a stress drop multiplier on a log scale. Increasing by 0.1 implies a global stress drop increase of 40%. See section 2.3.1.
-t METHOD, --taper_fault_method=METHOD	Specify the how to taper the imported fault model when meshing. Choices for taper method are: none, taper, taper_full, taper_renorm. See 5.2.6.1 for a description of tapering.

C.1.2 File Import Options

-i FILE, --import_file=FILE	Specify a model file to import and merge. Must have a paired import_file_type.
-j TYPE, --import_file_type=TYPE	Specify a model file type for importing. Must have a paired import_file. Must be one of trace, text, or hdf5.
-l FILE, --import_trace_element_size=FILE	Specify the element size to use for trace file meshing. Must have a paired trace type file import.
-C FILE, --import_eqsim_condition=FILE	Specify an EQSim condition file to import for the model.
-F FILE, --import_eqsim_friction=FILE	Specify an EQSim friction file to import for the model.
-G FILE, --import_eqsim_geometry=FILE	Specify an EQSim geometry file to import for the model.
-v, --vertex_das_by_section	Use if imported vertex distances along strikes (from EQSim file) are given with respect to section instead of whole fault..

C.1.3 File Export Options

-e FILE, --export_file=FILE	Specify a file to export the completed model to. Must have a paired export_file_type.
-f TYPE, --export_file_type=TYPE	Specify a file type to export the completed model. Must have a paired export_file. Must be one of trace, trace_faultwise, text, or hdf5.
-D FILE, --export_eqsim_condition=FILE	Specify an EQSim condition file to export for the model.
-R FILE, --export_eqsim_friction=FILE	Specify an EQSim friction file to export for the model.
-M FILE, --export_eqsim_geometry=FILE	Specify an EQSim geometry file to export for the model.

Appendix D

Virtual Quake Output File Format

D.1 Introduction

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

D.2 HDF5 Output

If Virtual Quake is compiled with the HDF5 library it is possible to output simulation results in this format. 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.

D.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.

D.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.

D.3 Events

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`. Corresponding data is instead written to ASCII text files if the appropriate parameter is set in the simulation parameter file.

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.

D.4 Event Sweeps

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. Corresponding data is instead written to ASCII text files if the appropriate parameter is set in the simulation parameter file.

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).

Appendix E

License

Bibliography

[1] Michael Barall. Data transfer file formats for earthquake simulators. *Seismological Research Letters*, 83(6):991–993, 2012.

[2] 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.

[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] Steven N. Ward. Allcal earthquake simulator. *Seismological Research Letters*, 83(6):964–972, 2012.

[11] 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.

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

[13] 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.