Notes on setting up and running `RSQSim` and plotting the results

October 6, 2011

1 Other software needed

1.1 C compiler - gcc

I believe the code would compile with nearly any C compiler, but I have only tested it with `gcc`. With other compilers, some of the compiler options in `Makefile` might need to be modified or eliminated. `gcc` is available for nearly every workstation class computing platform. On Mac OS X, `gcc` comes with the Developer Tools which should be an optional install on the system CDs/DVDs and is also available for download from `http://connect.apple.com`.

1.2 R

Most of the processing/plotting of the results is done in `R`. `R` is the free, open-source version of AT&T Bell Labs' `S+`. It is a statistical computing environment and has some similarities to `MATLAB`. It is available from `http://www.r-project.org`. It can also be installed through `fink` (`http://fink.sf.net`). The `fink` package name is `r-base`. There are thousands of other open-source software packages available through `fink`. The easiest way to run `fink` is through the GUI interface `FinkCommander` (which appears to now come with `fink` or see `http://finkcommander.sf.net`).

1.3 Subversion

To retrive the model source code and post-processing and plotting routines, you will need the source-code revision software `subversion`. It is available through `fink` (package name `svn`) or from the `subversion` home page (`http://subversion.tigris.org`).

1.4 Gifsicle

The `R` routines that make animations of the ruptures uses a piece of software with the somewhat silly name `gifsicle` to stitch the individual frames together into an animated gif. It is available through `fink` (package name `gifsicle`), or from the gifsicle homepage: `http://www.lcdf.org/gifsicle`.

2 Getting the source code

The source code is in a Subversion repository on my workstation (`rsqsim.ucr.edu`). To check the code out, `cd` to the directory to which you want to the code to be extracted and execute the following command:
`svn co svn://rsqsim.ucr.edu/RSQSim-mirror/branches/old-write-style/ .`
Note the '`.`' at the end which refers to the current directory.

Once you have the source code, you can update to the latest version by executing `svn update` from that same directory. You can also do `svn log` from that directory to get the log files for each revision of the source code. If you need other than the current version, replace `svn co` above with `svn co -r` *`rev`*, where *`rev`* is the number of the revision you want to checkout.

3 Compiling the source code

This should be as easy as executing `make` from within the directory to which you extracted the code. If your CPU is not a 64-bit x86, then you'll need to modify the `Makefile` to delete the `-arch x86_64` before running `make`. The `Makefile` includes a number of separate targets - see `Makefile` for those if you wish to only `make` some of them.

The default is to compile for use with `OpenMPI` for parallization. If you do not have `OpenMPI` on your system, a serial version of the code should compile if you make two changes to the `Makefile`: 1) change the definition of `CC` from `mpicc` to whatever compiler you wish to use (*e.g.* `gcc`); and 2) delete `-DUSE_MPI` from the definition of `CFLAGS`.

4 R scripts

Almost all of the processing/plotting and some building of input files is done with `R` scripts (`http://www.r-project.org`). To get `R` to load up all of these scripts, you'll need to add the contents of the `.Rprofile` file that is generated by the `make` process to the `.Rprofile` file in your home directory.

5 Running the code

The code is run with the following syntax:

`runRSQSim [`*`parameterFile ...`*`]`

where *`parameterFile`* is either the name of a file containing parameter definitions or '–'. In the latter case the parameter definitions are read from `stdin`. Each line of the *`parameterFile`*s (or `stdin`) should be of the form:

`parameterName = value`

The '=' *must* be surrounded by spaces. If supplying parameter values on `stdin`, enter `cntl-D` after the `<cr>` of your final line. A current list of the `parameterName`s and their default values appears in Tables 1 and 2. These can also be found in the file `RSQSim_defaultParams.h`. Explanations of the parameters are in the following section.

5.1 Running in parallel

If you have `OpenMPI` and compiled with the default `Makefile` you should be able to run `runRSQSim` in parallel a command line like

`mpirun -np 8 runRSQSim [`*`parameterFile ...`*`]`

The `-np` arg specifies the number of processes to use. See the `OpenMPI` docs for how to, *e.g.* run processes on multiple nodes.

parameterName	type	default	units
A_1	float	0.001	
fA	float	1.0	
B_1	float	0.015	
Dc_1	float	10^{-5}	m
mu0_1	float	0.6	
ddotStar_1	float	10^{-6}	m/s
ddotAB_1	float	10^{-6}	m/s
alpha_1	float	0.25	
theta0_1	float	$2 \cdot 10^{8}$	s
tau0_1	float	60.0	MPa
sigma0_1	float	100.0	MPa
sigmaFracPin	float	0.0	
maxThetaPin	float	10^{10}	s
ddotEQ_1	float	1.0	m/s
ddotEQFname	string		
stressOvershootFactor	float	0.10	
lameLambda	float	10^{4}	MPa
lameMu	float	10^{4}	MPa
slowSlip_1	integer	0	
nEq	integer	1000	
tStart	float	0.0	s
maxT	float	10^{100}	s
faultFname	string	faults.in	
outFnameInfix	string		
writeTau	integer	0	
writeSigma	integer	0	
writeSlip	integer	0	
writeSlipSpeed	integer	0	
writeState	integer	0	
writeTheta	integer	0	
writeTransitions	integer	1	

Table 1: Parameter names, types, and default values

paramterName	type	default	units
minDtWrite	float	0.0	s
minDtWriteCoseismic	float	0.0	s
minDtWriteInterseismic	float	0.0	s
minMagWrite	float	0.0	
writeStiffness	integer	0	
stressRateSpecification	integer	1	
dMu3	float	0.001	
initTauFname	string		
initSigmaFname	string		
initThetaFname	string		
initSlipSpeedFname	string		
AFname	string		
BFname	string		
DcFname	string		
mu0Fname	string		
ddotStarFname	string		
ddotABFname	string		
alphaFname	string		
KTauFname	string		
KSigmaFname	string		
tFailFname	string		
tauFailFname	string		
tauDotFname	string		
sigmaDotFname	string		
pinnedFname	string		
neighborFname	string		
stressRateFanme	string		
slowSlipFname	string		
DEBUG	integer	0	

Table 2: More parameter names, types, and default values

6 Parameter explanations

For reference, note that the constutive relation in `RSQSim` is

$$\tau = \sigma \left(\mu_0 + a \ln \left(\frac{V}{V^*} \right) + b \ln \left(\frac{\theta V^*}{D_c} \right) \right), \tag{1}$$

and the aging form of the state evolution equation (with the normal stress dependence from **?**) is used:

$$\dot{\theta} = 1 - \frac{\theta V}{D_c} - \alpha \frac{\theta \dot{\sigma}}{b\sigma}. \tag{2}$$

6.1 A_1

The rate coefficient a in the rate and state constitutive law, if one value is to be used for all fault elements. Ignored if a value is supplied for `AFile`.

6.2 fA

Factor by which a on nucleating patches is multiplied during rupture.

6.3 B_1

The state coefficient b in the rate and state constitutive law, if one value is to be used for all fault elements. Ignored if a value is supplied for `BFile`.

6.4 Dc_1

The value of the characteristic distance for slip evolution in the aging law, D_c, (in m) if one value is to be used for all fault elements. Ignored if a value is supplied for `DcFile`.

6.5 mu0_1

The value of μ_0 in the constitutive law, if one value is to be used for all fault elements. Ignored if a value is supplied for `mu0File`.

6.6 ddotStar_1

The value of the reference slip speed, V^*, in the constitutive law (in m/s), if one value is to be used for all fault elements. Ignored if a value is supplied for `ddotStarFname`.

6.7 ddotAB_1

The value of the slip speed for transition between states `2a` and `2b` V^{AB}, (in m/s), if one value is to be used for all fault elements. Ignored if a value is supplied for `ddotStarFname`. This has to do with slow slip patches and its usage is not yet documented.

6.8 alpha_1

The value of the coefficient α relating changes in normal stress to changes in the state variable in the rate and state constitutive law, if one value is to be used for all fault elements. Ignored if a value is supplied for `alphaFile`.

6.9 theta0_1

The initial value of the state variable θ in the rate and state constitutive law (in seconds), if one value is to be used for all fault elements. Ignored if a value is supplied for `initThetaFname`.

6.10 tau0_1

The initial value of the shear stress τ (in the specified slip direction) for the patches (in MPa), if one value is to be used for all fault elements. Ignored if a value is supplied for `initTauFname`.

6.11 sigma0_1

The initial value of the normal stress σ for the patches (in MPa), if one value is to be used for all fault elements. Ignored if a value is supplied for `initSigmaFname`.

6.12 sigmaFracPin

If the normal stress on any patch drops to the product of `sigmaFracPin` and that patch's initial normal stress, that patch is locked for the remainder of the simulation.

6.13 maxThetaPin

If the state variable on any patch ever exceeds `maxThetaPin` + $1000t$, that patch is locked for the remainder of the simulation. In seconds.

6.14 ddotEQ_1

The slip speed during state 2 (seismic slip) V^{EQ} is given by `ddotEQ_1` (in m/s), if one value is to be used for all fault elements. Ignored if a value is supplied for `ddotEQFname`.

6.15 ddotEQFname

If separate V^{EQ} values are to be assigned to each patch, give the name of a file as `ddotEQFname`. This file should be a one-column ascii file where each line gives V^{EQ} (in m/s) for the corresponding patch in the `faultFname`.

6.16 stressOvershootFactor

State 2 (seismic slip) nominally ends when the shear stress drops to the "steady-state" stress (which is in quotes because when normal stress is time-varying the concept of steady-state is not quite valid) at the seismic slip speed V^{EQ}. Actually it ends when the stress drops somewhat below this "steady-state" stress with the extra stress drop given as `stressOvershootFactor` times the stress drop from the peak stress (upon entering state 2) to the "steady-state" stress.

6.17 lameLambda and lameMu

The Lamé elastic moduli λ and μ (in MPa).

6.18 slowSlip_1

`slowSlip_1` should be `0` if all the patches are not slow slip patches and `1` if they all are. Ignored if a value is supplied for `slowSlipFname`. Slow slip patches are not yet documented, but briefly: any patches designated as slow slip patches will use an alternate constitutive law with a low-state cutoff:

$$\tau = \sigma \left(\mu_0 + a \ln \left(\frac{V}{V^*} \right) + b \ln \left(\frac{\theta V^*}{D_c} + 1 \right) \right). \tag{3}$$

6.19 nEq and maxT

The simulator will run until `nEq` events have been produced or `maxT` seconds of simulated time have passed, whichever comes first.

6.20 tStart

Upon startup, the time in the simulation is set to tStart. Possibly useful if you are continuing a previous run.

6.21 faultFname

The name of the file containing the definition of the fault system geometry. This file is a 9-, 10-, or 11-column, whitespace separated ascii file, with each line of the file referring to one fault element and where the columns are:

```
x y z l w strike dip rake slip_rate [section_number section_name]
```

`x`, `y`, `z` The cooridinates of the center of the patch (in meters)

`l`, `w` The along-strike and down-dip dimensions of the fault patch (in meters)

`strike`, `dip` The orientation of the fault patch (in degrees and using the convention of Aki and Richards, 2002)

`rake` The direction of motion of the hanging wall relative to the footwall (in degrees and using the convention of Aki and Richards, 2002)

`slip_rate` The long-term average slip rate (in m/s) if `stressRateSpecification` is 1. If `stressRateSpecification` is 2 or 3, then this 9^{th} column is ignored but must still be present.

`section_number`, `section_name` These optional last two columns allow the patches to be grouped into fault sections, if so desired. Note that `section_name` should not contain whitespace. Also, `section_name` must not be interpretable as a numeric value. `runRSQSim` ignores these section numbers and names, but some of the post-processing rountines make use of them.

Note that the coordinate system used has the free surface of the half-space being the xy-plane with the elastic half-space in the region $z \leq 0$. That is z is positive up, and so all the z coordinates of the patches should be negative.

6.22 outFnameInfix

The output files will contain `outFnameInfix` as part of their file name. For example `eqs.`*`outFnameInfix`*`.out`.

6.23 writeTau, writeSigma, writeSlip, writeSlipSpeed, writeState, and writeTheta

The values of these parameters determine how often the respective quantities are written out. Possible values are:

0 never

1 at every transition

2 before and after every event

4 at every transition during an event

any sum of the last three write out multiple files according to which of the last three went into the sum

The files will be named something like `tau.`*`outFnameInfix`*`.out.`*`writeTau`* (where *`writeTau`* will be 1, 2, or 4. For example, if the parameter `writeTau` is 3, then two shear stress output files will be written `tau.`*`outFnameInfix`*`.out.1` and `tau.`*`outFnameInfix`*`.out.2`) Note that how often things get written out may be modified by `minDtWrite` and `minMagWrite`. These output files are binary files (and therefore cannot be transferred between machines of different endianness unless bytes are swapped). The format is simply 8 bytes for the time as a `double` and then `np*8 bytes` for the values of the relevant quantities (as `double`s and where `np` is the number of patches in the model) repeated for each time sample. Except that the `state.*.out.?` files use 4 byte `int`egers for the values of the state at each time step instead of 8 byte `double`s.

6.24 writePED

If `writePED` is non-zero, then five files named
`eqs.`*`outFnameInfix`*`.pList`
`eqs.`*`outFnameInfix`*`.eList`
`eqs.`*`outFnameInfix`*`.dList`

`eqs.`*`outFnameInfix`*`.tList`
`eqs.`*`outFnameInfix`*`.dtauList`
are written out. These are one-column ascii files (and would more logically be combined into a single, five-column ascii file). For each patch the ruptures during an event, a line is written to each of these files giving 1) the patch number, 2) the event number, 3) the distance slipped during the event, 4) the time of first rupture for that patch during that event, and 5) the change in shear stress for that patch during that event (that is, the difference between the shear stress at the end of the event relative to that at the beginning of the event), respectively. Note that these event and patch numbers are 1-based (yes, this is inconsistent with, *e.g.* that in `neighborFname`).

6.25 writeTransitions

If `writeTransitions` is non-zero will write out a files named like `trans.`*`outFnameInfix`*`.out`. This will be binary file (and therefore cannot be transferred between machines of different endianness unless bytes are swapped), with three values written to it for every transition: an 8-byte `double` giving the time of the transition (in seconds) and two 4-byte `int`s giving the number of the element which transitioned and which state it transitioned to.

6.26 minDtWrite

Write out tau, sigma, theta, slip, slip speed and/or state at most every `minDtWrite` (in seconds). Affects both `*.1` and `*.4` output files.

6.27 minDtWriteCoseismic and minDtWriteInterseismic

Allows separate values to be given for `minDtWrite` during and between ruptures, respectively. `minDtWriteCoseismic` will affect `*.1` and `*.4` output files, while `minDtWriteInterseismic` will affect only `*.1` output files. If these are not given and `minDtWrite` is, `minDtWrite` will be used for both. If these are given, any value given for `minDtWrite` will be ignored.

6.28 minMagWrite

Only write out tau, sigma, theta, slip, slip speeed, and/or state for events with $M \geq$ `minMagWrite`. Affects `*.2` and `*.4` output file.

6.29 writeStiffness

Which stiffness matrices to write out. Possible values are:

0 none

1 K^τ

2 K^σ

3 K^τ and K^σ

The output files will be named like `Ktau.`*`outFnameInfix`*`.out` and `Ksigma.`*`outFnameInfix`*`.out`. They will be ascii files with n_p lines and each line with n_p whitespace separated entries, where n_p is the number of patches in the fault model, and the j^{th} field on the i^{th} line is K_{ij} (in MPa/m).

6.30 stressRateSpecification

The value of `stressRateSpecification` indicates how the tectonic stressing rates are to be determined for each patch. Possible values are:

1 The 9^{th} column of `faultFname` is interpreted as a long-term average slip rate (in m/s), and the stressing rates (both shear and normal) are determined via a backslip calculation.

2 Stressing rates (in MPa/s) are provided in `tauDotFname` and/or `sigmaDotFname`. If one (or both) of these file names is not supplied, the corresponding stressing rates are set to zero.

3 A constant stressing rate tensor is provided in `stressRateFname` and this rate tensor is projected onto each fault patch to give the shear and normal stressing rates.

6.31 dMu3

Any patches for which $a > b$ (and are therefore rate strengthening) are assigned a state of `3` (which never changes) and are approximated as always being in steady-state. As such, their slip speed is uniquely determined by the stresses acting on them. This continuous variation of slip speed is approximated by a piece-wise constant function of time: the speed is only updated when μ (the ratio of shear to normal stress τ/σ) deviates by an amount `dMu3` from that μ which was used to set the current slip speed.

6.32 initTauFname, initSigmaFname, initThetaFname, initSlipSpeedFname

If values are supplied for these, the initial values of the respective quantities are read in from the file whose name is supplied (and any relevant parameters supplied for `tau0_1`,`sigma0_1`, and `theta0_1` will be ignored). The files should be single-column ascii files where the i^{th} line gives the value of the relevant quantity for the i^{th} patch (in MPa for the stresses, seconds for the state variable θ, and m/s for the slip speeds).

6.33 AFname, BFname, DcFname, mu0Fname, ddotStar, ddotAB, alphaFname

If values are supplied for these, the respective material paramters will be read from the file whose name is supplied (and any relevant parameters supplied for `A_1`, etc. will be ignored). File formats should be similar to that described for `initTauFname` with the values for a, b, μ_0, and α being dimensionless, the values for D_c being in meters, and the speed values being in m/s.

6.34 KTauFname, KSigmaFname

If values are supplied for these, the respective stiffness matrices are read in from the file whose name is supplied rather than being calculated. The file format is as is described for `writeStiffness`.

6.35 tFailFname, tauFailFname

If a value is supplied for `tFailFname`, the patches are artifically forced to fail at the times given in the file whose name is supplied. The format of the file is a single column ascii file with the i^{th} line giving the time (in seconds) at which to force the i^{th} patch to fail. Any negative time indicates a patch which is not forced to fail at any *a priori* time. How a given patch is forced to fail depends on whether a `tauFailFname` is supplied, and if so, upon the value for the given patch in that file. The file whose name is given by `tauFailFname` should also be a single column ascii file with the i^{th} line giving the failure shear stress (in MPa) for the i^{th} patch. Any negative value indicates that the shear stress at failure is unspecified. If no value is supplied for `tauFailFname` or if a given patch that is being forced to fail has a negative entry in `tauFailFname`, then the shear stress is left unchanged and

the state variable θ is reduced sufficiently to cause the patch to fail immediately. If there is a non-negative value in `tauFailFname` for a given patch that is being forced to fail, the shear stress is first instantaneously changed to that value and then the state variable θ is reduced sufficiently to make the patch fail immediately.

6.36 tauDotFname, sigmaDotFname

If `stressRateSpecification` is `2`, then the stressing rates for each patch are read from these files. Each should be a single-column ascii file, with each line giving the stressing rate (in MPa/s) for the corresponding patch in `faultFname`. If one (or both!) of these files is not supplied, the corresponding stressing rates are set to zero.

6.37 pinnedFname

If a value is supplied for this then some of the patches may be pinned (not allowed to slip). The file whose name is supplied should be a single-column numeric ascii file where if the i^{th} line is non-zero then the i^{th} patch is not allowed to slip.

6.38 neighborFname

As an *ad hoc* way of accounting for the fact that with our oversized patches we are unable to resolve the high stresses at the edge of a rupture front, we reduce the rate coefficient a on state `1` patches when other patches are rupturing. This should logically be done only on those patches which are neighbors to those that are rupturing. If nothing is supplied for `neighborFname` then a is reduced on all state `1` patches during rupture (which works ok in some circumstances). It is better to supply a value for `neighborFname` which should be an ascii file with one line per fault patch. Each line should be a white-space separated list of the patch numbers of the neighbors of the corresponding fault patch. Note that these patch numbers are 0-based.

7 Output files

The only file that is always output is one containing basic "catalog" information. It is named `eqs.`*`outFnameInfix`*`.out`. It is an ascii file. The first line gives the revision number of the code that was used in the run. Then there are a number of lines giving all the control parameters used (in the same format that is used for the

paramterFile, so that this part of the file can be cut out, possibly modified, and used as an input parameterFile). Then follows one line for each earthquake with eight fields whose meanings are: t_0, M_0, M_w, x, y, z, area, dt. The meanings of the fields are:

t_0 origin time in seconds

M_0 seismic moment in N-m

M_w moment magnitude ($M_w = (2/3)(\log_{10}(M_0) - 9.1)$)

x, y, z hypocentral coordinates in meters

area total area which ruptured in m^2

dt duration in seconds

Other optional files are output depending on various input parameters defined in the parameterFile and their formats are described along with the descriptions of the input parameters in the previous section.

8 Outline of typical steps for a run

8.1 Build a parameterFile

You can type this in by hand, or copy and modify one from a previous run, or just running runRSQSim with no command line arguments will result in an aborted run but a file named eqs..out will be written out with all of the default input paramters in the correct format for a parameterFile, from which the appropriate lines can be cut, pasted, and modified.

8.2 Build the fault geometry file faultFname

First an important note: some of the plotting routines depend on faults running subparallel to the y-axis. The format of this file is described with the description of the faultFname parameter. You can build this however you want: typing in by hand, writing a Fortran program to build the file, etc. There are a few R scripts in the BuildFaultModels subdirectory of the source code directory that do things like build random fractal faults and a few specific fault systems, that I may or may not document at some point. There is also a C program, singleStrikeSlip that

builds a `faultFname` file with a single vertical left-lateral strike-slip fault. It can be compiled by running `make singleStrikeSlip` from within the `BuildFaultModels` directory. It is run like:
`singleStrikeSlip x0 y0 zmin zmax L nl nz strike [dDotDrive] > faultFname`
The command line arguments are:

x0, y0 give the coordinates (in meters) of one end of the fault

zmin, zmax give the vertical extent of the fault (in meters). Note that `zmin` and `zmax` should be negative.

L gives the length of the fault (in meters)

nl, nz give the number of patches in the along-strike and down-dip directions, respectively.

strike gives the strike of the fault (in degrees and following the conventions of Aki and Richards, 2002)

dDotDrive the long term average slip velocity (in m/s) or the tectonic stressing rate (in MPa/s) depending on the value of the `stressRateSpecification` input paramter. This one value will be used for all patches.

Note that `singleStrikeSlip` writes its output to `stdout` and so should be redirected into a file as shown above.

8.3 Build any desired optional input files

These will be required if you want to specify a certain spatial distribution of initial conditions or material properties, if you want to pin part of the fault, or if you want to artifically nucleate an event. There are R scripts for building some of these.

8.3.1 Pinning part of the fault

If you have set up your `.Rprofile` as described previously, you should have access to a function named `mkPinnedFile` from within R (you start up R by simply executing `R` from the Unix prompt). Its calling sequence is `mkPinnedFile(faultFname, xmin, xmax, ymin, ymax, zmin, zmax, outFname)`. This will read in `faultFname` and write out a file to `outFname` which will cause any patches with their centers outside of the volume defined by `xmax, ymin, ymax, zmin, zmax` (in meters, as usual) to be pinned. If you leave out the `outFname` argument it will write to a file with a name derived from `faultFname` by appending `.pinned`.

8.3.2 Artifically nucleating an event

There is an R function named `mkFailCircle` which will write out a file suitable for use as a `tFailFile` (and optionally a `tauFailFile` as well) which will cause an event to be nucleated by forcing patches to fail within a circular area with a given center and which expands at a given rate out to a given maxmimum radius (beyond which the rupture may or may not spontaneously propagate). Its calling sequence is `mkFailCircle(faultFname, outFname, x0, t0, v, rmax, tauFail0, tauOutFname)`. The `faultFname` argument should be obvious. `x0` is a vector of length 3 whose elements give the x, y, and z coordinates (in meters) of the center of the circle. `t0` gives the time (in seconds) of the start of the nucleation. `v` gives the rate at which the radius of the nucleation circle expands (in m/s). `rmax` gives the maximum radius (in meters) of the nucleation circle. `tauFail0` and `tauOutFname` are optional arguments if you want the patches to fail with a particular value of shear stress. `tauFail0` is the (single) value of shear stress (in MPa) to which failing patches will be artifically brought before being forced to fail, and `tauOutFname` is the name of a file suitable for use as a `tauFailFname` input parameter.

8.3.3 Spatially varying initial conditions or material parameters

I give here a simple example of how to make a file suitable for use as `initSigmaFname` with a block of higher normal stress (from within R)

```
faultFname <- "80km.fault"
sigma0 <- 120
sigma1 <- 180
ymin <- 40e3; ymax <- 43e3; zmin <- -5.5e3; zmax <- -2.5e3
initSigmaFile <- "initSigma"
flt <- readFault(faultFname)
sigma <- rep(sigma0, flt$np)
sigma[flt$y >= ymin & flt$y <= ymax &
      flt$z >= zmin & flt$z <= zmax] <- sigma1
write(sigma, initSigmaFile, 1)
```

The first five lines just set up the file names, the normal stresses for the background and within the block, the extent of the block, and the name of the file to which to write the results. Note that <- is the assignment operator in R. The sixth line uses the `readFault` function to read in all the information about the fault into a variable `flt`. The seventh line makes `sigma` a vector with a length equal to the number of patches in the fault model (`flt$np`), with all elements equal to `sigma0`

(`rep` is a built in R function which makes a vector by repeating a given value a given number of times). The eighth line (which continues to the ninth line) sets those elements of `sigma` that correspond to fault elements that have their centers within the block defined by `ymin, ymax, zmin, zmax` to be `sigma1`. The final line uses the builtin R function `write` to write out the values in the vector `sigma` to the file whose name is `initSigmaFile` as a single column (the final argument, 1).

8.4 Run the model

Just execute `runRSQSim` *`parameterFile`* at your shell or OS prompt. It will write out to the screen the parameters it is using (so you can check to see if it interpreted your `parameterFile` properly). Then keeps you more or less informed on what it is doing, eventually telling you as it finishes and begins each event. Note that if you are doing a long run, you can use all of the R functions in the next section at any time to make plots for the events that have finished, even while `runRSQSim` is continuing to add new events.

9 Plotting the results

The plotting of results is all done from within R. There are a number of R functions in the source code directory that should be available if you have your `.Rprofile` set up as described earlier. Several of these plotting functions are described below.

9.1 Reading in the basic catalog information

All of the plotting routines depend upon the catalog information so this is the first thing that must be done. This is done with the `readEqs` function. The following line would be a typical use

```
eqs <- readEqs("eqs.sample.out")
```

You can of course use any variable name you want in place of `eqs` and would need to replace the file name with that from your run. `readEqs` returns a data structure with all of the information from the catalog file, the fault geometry, and the input parameters. You can get a list of the names of elements of the data structure with the following R command: **names**(eqs). You should get something like the following:

```
[1] "baseDir"                "ProgramVersion"
[3] "AMin"                   "AMax"
```

```
 [5] "fA"                    "BMin"
 [7] "BMax"                  "DcMin"
 [9] "DcMax"                 "mu0Min"
[11] "mu0Max"                "alphaMin"
[13] "alphaMax"              "theta0Min"
[15] "theta0Max"             "tau0Min"
[17] "tau0Max"               "sigmaMin"
[19] "sigmaMax"              "sigmaZGrad"
[21] "ddotEQ"                "stressOvershootFactor"
[23] "lameLambda"            "lameMu"
[25] "nEq"                   "maxT"
[27] "faultFname"            "outFnameInfix"
[29] "writeTau"              "writeSigma"
[31] "writeSlip"             "writeState"
[33] "writeTheta"            "writeTransitions"
[35] "minDtWrite"            "minMagWrite"
[37] "writeStiffness"        "stressRateSpecification"
[39] "initTauFname"          "initSigmaFname"
[41] "initThetaFname"        "AFname"
[43] "BFname"                "DcFname"
[45] "mu0Fname"              "alphaFname"
[47] "KTauFname"             "KSigmaFname"
[49] "tFailFname"            "tauFailFname"
[51] "pinnedFname"           "t0"
[53] "M0"                    "M"
[55] "x"                     "y"
[57] "z"                     "area"
[59] "dt"                    "fault"
[61] "patch"                 "hypoDensity"
[63] "tauDot"                "sigmaDot"
```

I may document these fully later, but for now: the 3rd through 51st of these are the control parameters, the 52nd through 59th are vectors of the quantities in the catalog file, `fault` is data structure in itself with all the info from the `faultFname` file, `patch` is the number of the fault element upon which each event nucleated, `hypoDensity` is the number of events which nucleated on each fault element, `tauDot` and `sigmaDot` are the tectonic stressing rates for each fault element. In R, you access the elements of a data structure with the dollar sign ($). For example, in the above example

`eqs$M` would be a vector with the moment magnitudes of all the events.

9.2 Basic catalog plots

If you have done a run generating a catalog for a large number of events (versus a run for a single large artificially nucleated event), the function `plotEqs` may be of interest. This function produces a number of plots of things like magnitude vs. time, frequency-magnitude distribution, etc. The first (and only required) argument is the data structure read in with `readEqs`. Thus the simplest call would be `plotEqs(eqs)` (with `eqs` replaced by whatever variable name you are using). There are some futher arguments, but you'll have to look at the code or wait for me to update this documentation for those.

9.3 Event stress change and slip distributions

`stressDropSlipImage` is a function which produces images of of the shear stress change and total slip in a chosen set of events. Its full calling sequence is:
`stressDropSlipImage(eqs, use=1:eqs$nEq, profileZ=NULL, tau=NULL, d=NULL, mkPdf=FALSE, writeTau=2, writeSlip=2)`
The arguments that are followed by '=' have defaults (the values on the right side of the '=') and so can be omitted if you are happy with the default value. Thus the only required argument is the first, which is, as usual, the data structure read in by `readEqs`. The function will plot two windows, one with images of the stress change and slip distributions in one event and the other with profiles of stress change and slip at a given depth level. It will then prompt on whether to continue to the next event (in order of time), switch and go the other direction in time, or quit. Note that this is one of the functions that depends on the fault being subparallel to the y-axis, in the sense that the view of fault is looking in the negative-x direction.

9.3.1 Optional arguments

First, note that in R any arguments you give in their proper order do not need to be named. For example, `slipDropStressImage(eqs1, 1:10)` passes in `eqs1` as `eqs` and 1:10 (a vector with the integers from 1 to 10 inclusive, MATLAB and R share this syntax) as `use`. Arguments can be given out of order but then must be named. For example if you wanted to pass in `TRUE` for `mkPdf` but wanted to use the defaults (and therefore not have to type in) all the arguments between `eqs` and `mkPdf` you

could use: `stressDropSlipImage(eqs, mkPdf=TRUE)`

use This argument determines which events will have their stress change and slip distributions plotted. The default (`1:eqs$nEq`) is to make plots for all of the events. If you have a few specific events you want, you can type the numbers in by hand. Note that to construct a vector in R from some specific numbers, you use the builtin R function `c`. For example, `c(2,3,5,7,11)` gives a vector whose elements are the first five prime numbers. Another common situation would be to want to look at all the events bigger than some magnitude or rupture area. For this, use the standard R function `which`. For example, `stressDropSlipImage(eqs, use=which(eqs$M >= 7))` would make plots for those events with magnitudes equal to or greater than 7. Notes for those new to R: `eqs$M` is a vector with the magnitudes of all the events; so `eqs$M >= 7` is a vector of the same length as `eqs$M` with elements equal to `TRUE` if the corresponding element of `eqs$M` is greater than or equal to 7 and `FALSE` otherwise; and `which` is a function that returns a vector of the indices of its vector argument that are `TRUE`; so that `which(eqs$M >= 7)` is a list of those events whose magnitude is greater than or equal to 7.

profileZ First note that profiles will only work properly for regular geometries. This argument specifies at which depth the profile will be taken as an integer ranging from 1 (the deepest row of patches) to `eqs$fault$nz` (the shallowest row). Giving a negative value for `profileZ` will plot the depth-averaged stress change and slip.

tau, d To be documented later

mkPdf Eventually, if this is `TRUE` then a pdf file of the plots will be written out, but it is not yet implemented. I'll either implement this soon, or at least explain here how to make a pdf of any plot window in R.

writeTau, writeSlip These should integers, either 1, 2, or 4. The function will try to read the stresses and slips from files with that extension, so the appropriate file must have been written out during the model run. Which files are written out by the model is controlled by the parameters of the same name as these arguments. The default values (of 2) make the most sense here since this function only needs the values at the start and end of the events, but doesn't need those from any times during the events.

9.4 ruptureAnimation

This function also depends on the fault(s) being subparallel to the y-axis. It makes an animation of the state (0, 1, or 2) of each patch during (and, if desired, between) ruptures. Patches in state 0 are colored white, those in state 1 blue, and those in state 2 red. It makes a frame every 0.1 s (currently hardwired, needs to be changed to be an argument). It plots each frame on the screen and makes a .gif and .png of each frame and then calls an external program (with the somewhat silly name of `gifsicle`) to put the frames together into an animated gif movie file. If it runs to completion, the individual frame files get deleted, but if it crashes or you interrupt it these files are left around and need to be deleted by hand. The final animated gif file will be named like `state.`*`outFnameInfix`*`.`*`eventNumber`*`.all.gif` and will be in the subdirectory `Plots` of whatever directory the output files all live in.

The full calling sequence is: `ruptureAnimation(eqs, ieq, ieqf=ieq, nmax=10000, writeState=4)`. Here there are two required arguments. The first is the usual data structure that was read in by `readEqs`, and the second is the number of the event you wish to animate. That is, the animation will start at the beginning of the `ieq`$^{\text{th}}$ event.

9.4.1 Optional arguments

ieqf The animation will stop at the end of the `ieqf`$^{\text{th}}$ event. The default is to stop at the end of the `ieq`$^{\text{th}}$ event so that only that event is animated. If `ieqf` is larger than `ieq` then several events and the time periods between them will be animated. There is currently a bug in multi-event animations such that the last patch that stops sliding in one event is plotted as sliding during the entire interseismic period until the next event.

nmax Depending on the size of the model and the number of time samples in the output file corresponding to the event(s) in question, it may take too much memory to read all the relevant data for an event in at once. `ruptureAnimation` will read in `nmax` time samples at a time. Having this argument is a dumb way to deal with this memory problem. Instead `ruptureAnimation` should figure out how much memory the machine has (via `gc`) and based on that and the number of fault elements in the model (`eqs$fault$np`), figure out how many time samples it can safely read at a time.

writeState As with the `writeTau` and `writeSlip` arguments to `stressDropSlipImage` this is used in constructing the name of the output file from which the function will try to read the state data. As with those, the corresponding file must have been written out during the model run. Sensible values are either 1 or 4.

9.5 Further plotting functions

I'll document these as I get time. `slipAnimation` and `stressAnimation` are similar to `ruptureAnimation`. `stressProfileAnimation` and `slipProfileAnimation` do what their names imply. An additional argument to these is `iz`, an integer which gives at which depth to make the profiles, similar to the `profileZ` argument to `stressDropSlipImage`. The default is to make the profile at the shallowest level of the fault. Other plotting functions include `plotRuptureTimeAndSpeed` and `slipImages`, and there are others as well.