Quake-DFN

version 1.1

User's Guide (draft)

Kyungjae Im

February 2025

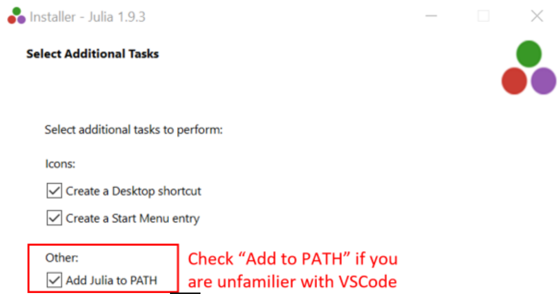
Table of Contents

Installation and Test

Installation guide

Quake-DFN is written in Julia. VSCode IDE is recommended as it is coded and tested in the VSCode. The required Julia packages are listed below. We checked the following steps work.

1) Install Julia and VSCode in the most recent versions (check "add to PATH").



2) Open the VSCode, go to extensions ( ), and install "Julia Language Support"

3) Open Julia REPL (*control* + *shift* + *p* for pc or *command* +*shift* + *p* for Mac) and select "Julia: start REPL".

4) Install the packages:
In Julia REPL, press "]", then the prompt will become "(xxx) pkg>". At the prompt,

```
add Pkg
add PyPlot                    # (this may take a while)
add PyCall
add Conda
build PyCall
add DelimitedFiles
add JLD2
add LinearAlgebra
add Printf
add SpecialFunctions
add StaticArrays
add LowRankApprox
add Distributed
add Statistics
add Clustering
```

5) Install Python-related packages
Back to REPL (press "BackSpace")

```
using PyPlot                  #(this may take a while)
using PyCall
using Conda
```

6) restart VSCode and make sure the Quake-DFN folder is the root (file → open folder → select Quake-DFN folder)

Test Simulation

Unpacking and running

Unpack the Quake-DFN and open the QuakeDFN folder in VSCode (file → open folder in VSCode). The root folder has three sub-folders, nine '*.jl' files, one txt file, and this manual.

The txt file (*Input_BulkFaultGeometry.txt*) contains the input file of BP5QD benchmark problem (Jiang et al., 2022). This file alone is sufficient to conduct simulations. The simulation can be implemented by running *RUN_BUILD_INPUT.jl* and then *RUN_QUAKEDFN.jl*. The result is automatically saved in the "results" folder. The result can be visualized by running *Results/2_3DPlot_and_Animation.jl* (ensure PlotStep is within the total recorded step).

More details of the test simulation

The input txt file (*Input_BulkFaultGeometry.txt*) contains the rock properties and fault geometry of BP5QD SEAS benchmark problem (Jiang et al., 2022). The geometry can be visualized by running *Plot_BulkFaultGeometry.jl*. Once you run it, figure 1a will pop up. The detailed geometry is embedded within large surrounding loading faults that apply a constant loading rate. Zooming-in shows the actual geometry of the BP5QD problem. Each row in the *Input_BulkFaultGeometry.txt* represents one block of the geometry shown in the plot. The color code of the fault is set to present slip orientation (blue: left lateral, red: right lateral).

The faults can be discretized by running the "*RUN_BUILD_INPUT.jl*". Once run with setting Plot3D_DiscretizedBlock = 1, it will generate a 3D geometry plot in which the fault is discretized (figure 1b). The large loading faults are not discretized since there is no benefit as they only slip at constant velocity. Running the *RUN_BUILD_INPUT.jl* generates a file name *Input_Discretized.jld2*. This file is the actual input file that contains the stiffness matrices and initial parameters. Note that the friction parameters and initial conditions can be re-adjusted anytime after building this input file. Hence, element-by-element adjustment is possible. However, if the fault geometry needs to be changed, *Input_Discretized.jld2* should be rebuilt.

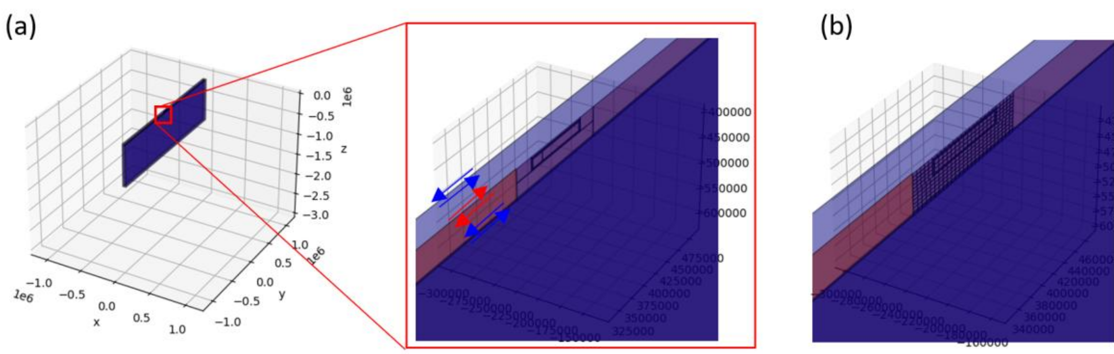


Figure 1. Test simulation geometry. (a) un-discretized fault geometry (*Plot_BulkFaultGeometry.jl*). (b) after discretization (*RUN_BUILD_INPUT.jl*)

Now, the simulation can be conducted by running the *Run_QUAKEDFN.jl*. Timestep plot may pop up if DtPlot = 1, which defines the simulation time step size at a given maximum velocity. The time step skim can be redefined in the *Run_QUAKEDFN.jl* (see section 2.5).

Once the simulation is done, the result file is automatically saved in the folder *Results*. Two files will be generated at the end of the simulation: *Result/Result.jld2* and *Result/Result_Input.jld2*. The first contains simulation results, such as velocity and displacement at each recorded step. The latter file contains input parameters. Several "*.jl" files provided in the *Results* folder generate different plots. Running *Result/2_3DPlot_and_Animation.jl* presents a 3D snapshot of the simulation result (default: velocity) at a given recorded step. Figure 2 shows the result of PlotStep =30, 50, and 100. Note that this simulation does not precisely reproduce the BP5-QD simulation as the elements are coarser. A more precise BP5QD simulation can be done by adjusting the maximum grid length in the *Input_BulkFaultGeometry.txt* (change 4000.0 to 1000.0)

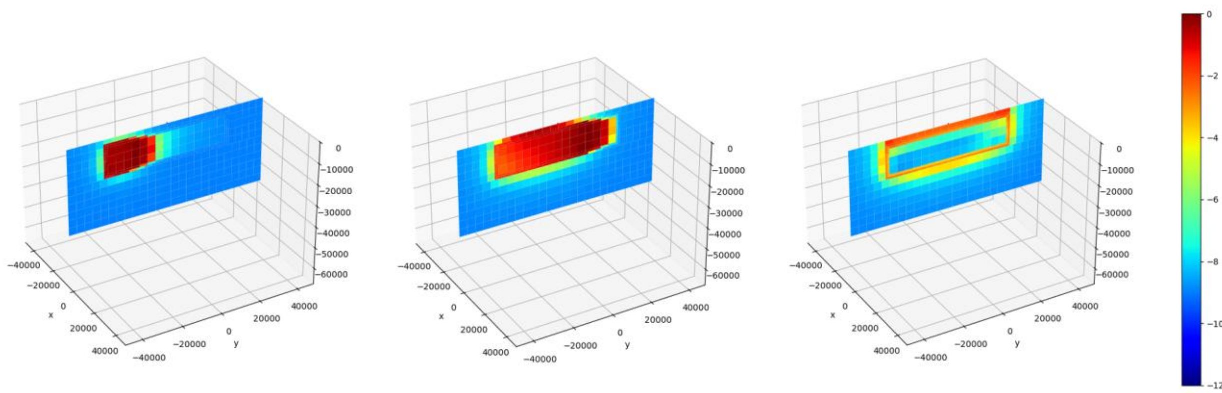


Figure 2. Simulation results plotted by *Result/2_3DPlot_and_Animation.jl* with PlotStep = 30, 50, and 100.

1. Quake-DFN Introduction

Quake-DFN is the boundary element simulator developed for earthquake rupture simulation of discretely distributed faults governed by rate and state friction law. The simulator formulation is, by default, consistent with the widely used quasi-dynamic formulation with radiation damping, but the lumped mass effect, which represents overshoot, can be enhanced if needed. The simulation can be conducted with/without H-Matrix compression.

1.1 Workflow summary

Input files

The simulator works with one essential input that defines un-discretized fault geometry and two supplementary input files that define detailed parameters in the discretized fault geometry and external stress. The three input files are listed below.

1) *Input_BulkFaultGeometry.txt*: This essential input file defines un-discretized fault geometry. It can be discretized by running *RUN_BUILD_INPUT.jl* (See section 2.1). Once discretized, this file is not necessary.

2) *QuickParameterAdjust.jl*: This file reads the discretized input (*Input_Discretized.jld2*) and change parameters before the simulations. Since it works on the discretized elements, it can apply detailed heterogeneity. (See section 2.3)

3) *Input_ExteralStressChange.jld2*: (Optional) This input defines external shear and normal stress change. It can be any external loading or injection-induced stress change. (See section 2.4)

Full matrix workflow

The full matrix simulation has two steps. (1) discretize the input file and (2) run the simulation. The two RUN flies are listed below

1) *RUN_BUILD_INPUT.jl*: Discretize the *Input_BulkFaultGeometry.txt* and generate the *Input_Discretized.jld2*, which is actual input file for the simulation.
Set `HMatrixCompress = 0` for full-matrix-only simulations.

2) *RUN_QUAKEDFN.jl*: Read *Input_Discretized.jld2*. and conduct a simulation. The total simulation steps and timestep size are defined here. (See section 2.5)

H-Matrix workflow

For H-matrix approximation, the simulation has three steps. (1) Build Hmatrix Structure. (2) discretize the input file, and (3) run the simulation. The three RUN flies are listed below

1) *RUN_BUILD_HmatrixStructure.jl*: Build Hmatrix Structure. It generates *Input_HmatrixStructure.jld2*, which is required for Hmatrix compression.

2) *RUN_BUILD_INPUT.jl*: Discretize the *Input_BulkFaultGeometry.txt* with *Input_HmatrixStructure.jld2*and generate the *Input_Discretized.jld2*, which is actual input file for the simulation.
   Set HMatrixCompress = 1 for H-matrix compression. If the element size is extremely large, this may take a very long time. In this case, one can consider a manual distributed computation implemented in the *DistributedDiscritization_beta* folder. Note that if the element size is very large, make sure to set SaveOriginalMatrix = 0 so that the original matrix will not occupy memory space

3) *RUN_QUAKEDFN_Hmatrix.jl*: Read *Input_Discretized.jld2*. and conduct a simulation. The total simulation steps and timestep size are defined here. (See section 2.5)

The simulation result is saved in the *Results* folder as a jld2 file. The folder also provides codes for 2D and 3D plots and Catalog generation.

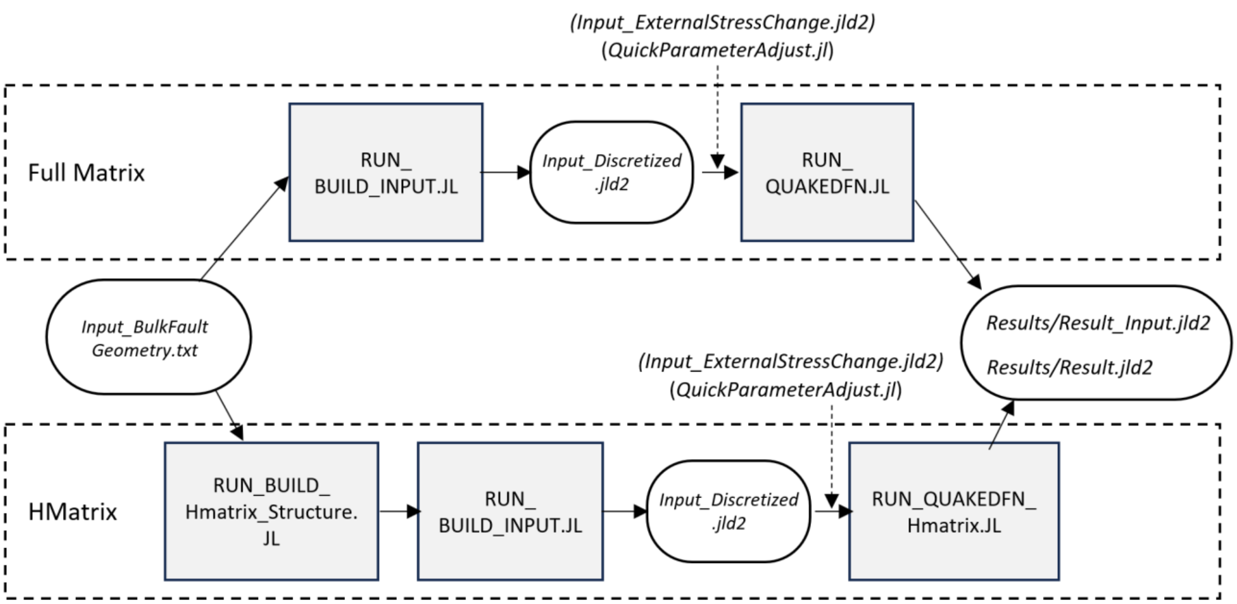


Figure 3. Simulation workflow chart.

1.2 Theoretical background

Quake-DFN considers discretely distributed boundary element faults in 3D elastic half-space with quasi-static stress transfer. A lumped mass (Im and Avouac, 2021) and radiation damping (Rice, 1993) approximate the inertia effect. With these assumptions, the momentum balance equation at i^{th} boundary element becomes

$$M_i \ddot{\delta}_i = \sum_j k_{ij}^{\tau}(\delta_{0j} - \delta_j) - \mu_i(\sigma_{0i}^{'} + \sum_j k_{ij}^{\sigma}\delta_j + \sigma_i^{'E}) - \frac{G}{2\beta}\dot{\delta}_i + \tau_i^E, \tag{1}$$

where M is the lumped mass per unit contact area for each element, δ_{0j} is the initial displacement of element j, δ_j is the shear displacement of element j, σ'_{0i} is the initial effective normal stress of element i, G is shear modulus, β is shear wave speed, and k_{ij} is a stiffness matrix that defines the elastic stress change imparted on element i due to slip of element j (k^τ and k^σ represent shear and normal stiffness matrix, respectively). The stiffness matrices are calculated by assuming quasistatic stress transfer (Okada, 1992). The τ^E and σ'^E are shear and effective normal stress changes driven by external stress, such as tectonic loading or poro-elastic stress change.

We used the rate and state friction (Dieterich, 1979; Marone, 1998)

$$\mu = \mu_0 + a \log\left(\frac{V}{V_0}\right) + b \log\left(\frac{V_0\theta}{D_c}\right) \tag{2}$$

and the aging law with the normal stress dependent evolution (Linker and Dieterich, 1992)

$$\frac{d\theta}{dt} = 1 - \frac{V\theta}{D_c} - \alpha\frac{\theta\dot{\sigma}}{b\sigma}, \tag{3}$$

where V is velocity, θ is the state variable, μ_0 is a reference friction coefficient at reference velocity V_0, Dc is a critical slip distance, and a and b are empirical constants for the magnitude of direct and evolution effects, respectively. The α term defines normal stress dependent state evolution.

The lumped mass approximation allows for inertial effects not captured by radiation damping alone, such as inertial overshoot or friction-induced vibrations (Im & Avouac 2021). The lumped mass per unit area M approximates the equivalent mass in a rupture process. If rupture size is fixed (as can happen in a repeating earthquake in a finite size of fault), M can be defined as

$$M \sim \frac{\rho L}{(1-\nu)\pi^2}, \tag{4}$$

where L is the length scale of the rupture size, ρ is rock density, and ν is Poisson's ratio. Conversely, if the rupture size is not fixed, L may be approximated by the expected rupture size in the simulations. By default, M is set to be h_{min} × density / 2, where h_{min} is the minimum size of all elements. This setting makes M/k_{ij} sufficiently low and suppresses the overshoot effect, making the simulation close to radiation damping alone.

The governing equation (equation 1) is a widely used quasi-dynamic formulation using radiation damping (e.g., Erickson et al., 2020) with an added overshoot effect. Therefore, our simulator is compatible with the other simulators that employ radiation damping (see section 3.1). We utilize two methods to solve equations 1-3: (i) a typical iterative method that is applied to a low-velocity system and (ii) the method of Im et al. (2017), which is stable at high velocity. The two solvers are automatically switched for each element based on their velocities. The timestep is dependent on the maximum velocity but automatically adapts if it fails to find a converged solution.

To resolve the earthquake rupture process, the shear stiffness of an element should be sufficiently larger than the critical stiffness (Rice, 1993). We define

$$\frac{k_{ii}^{\tau}}{K_c} = \frac{k_{ii}^{\tau} D_c}{(b-a)\sigma} \tag{5}$$

To achieve a minimum resolution, the k^{τ}_{ii}/K_c should be sufficiently larger than 1. In Quake-DFN, this can be checked before running the simulation. Once faults are discretized, run *Plot_Input.jl* with un-commenting *PlotInput=KoverKC* and comment out others (see below).

```
############################################################################
####################### Which parameter want to plot? ############
    ColorMinMax = 0
    # PlotInput = log10.(Fault_Theta_i); ColorMinMax = 0
    # PlotInput = log10.(Fault_V_i); ColorMinMax = 0
    # PlotInput =Fault_NormalStress; ColorMinMax = 0
    PlotInput =KoverKC ;ColorMinMax=[0,5]
    # PlotInput =UnderResolved ;ColorMinMax=[0,1]
    # PlotInput = Fault_a - Fault_b; ColorMinMax = 0
    # PlotInput =  Fault_BulkIndex; ColorMinMax = 0
    # PlotInput = Fault_Dc; ColorMinMax = 0
    # PlotInput = FaultLLRR; ColorMinMax = 0
    # PlotInput = FaultLLRR .* Fault_Friction_i; ColorMinMax = 0
    # PlotInput = Fault_Friction_i; ColorMinMax = 0
```

2. Workflow

2.1 Required Input File: *Input_BulkFaultGeometry.txt*

The *Input_BulkFaultGeometry.txt* file contains information on rock properties, fault geometry, frictional properties, and element size after the discretization. The first five lines of the test simulation are as follows.

```
SwitchSS/RN ShearMod    PoissonRatio    R_Density   Crit_TooClose   TooCloseNormal_Multiplier   Minimum_NS
1.0 2.0e10  0.2 2400.0  1.05    0.6 2.0e6
Ctr_X   Ctr_Y   Ctr_Z   St_L    Dip_L   StAng   DipAng  Rake    a       b       Dc      Theta_i V_i Fric_i  Sig0    SigGrad V_Const MaxLeng
-1000.0 -1000.0 1500.0  2000.0  3000.0  50.0    70.0    0.0     0.003   0.006   0.0002  1.0e10  1.0e-15 0.6 2.0e6   6000.0  0.0 200.0
```

The first and third lines (texts) are the index of input parameters for the input numbers of the second and fourth (and after) lines, respectively. The first line indicates

(1) SwitchSS/RN Input system of sense of slip:
(0): Rake Angle, (1): Strike-Slip, (2): Normal-Reverse.
(2) ShearMod Shear modulus [Pa]
(3) PoissonRatio Poisson's ratio [-]
(4) R_Density Rock Density [kg/m^3]
(5) Crit_TooClose Criteria for "too close"
(6) TooCloseNormal_Multiplier: Multiplier for normal interaction for the "too close" calculation
(7) Minimum_NS Minimum normal stress allowed in the simulation

The indexes (5) and (6) were introduced for numerical stability in very complex fault systems, but they are not being used in the current version.

Sometimes, complex geometry results in very strong interactions. This problem can be reduced by separating faults in *Input_BulkFaultGeometry.txt* at intersections, as shown in Figure 4b, so that any element center cannot be located at intersections.

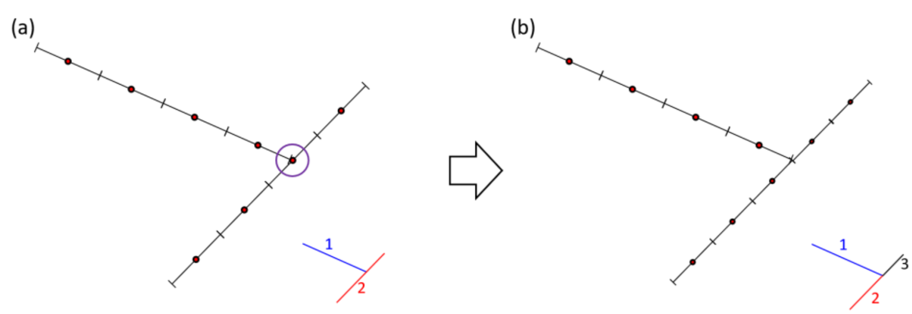


Figure 4. (a): Strong interaction example. (b) strong interaction is avoided. (b) can be achieved in *Input_BulkFaultGeometry.txt* by separating faults at the intersection (bottom right figure in b).

The indexes at the third line define fault geometry, frictional properties, and the initial state, which are denoted as follows (see figure 5 for illustration).

(1) Ctr_X x component of Fault Center [m]
(2) Ctr_Y y component of Fault Cente [m]
(3) Ctr_Z depth of Fault Center (always positive) [m]
(4) St_L fault length along strike [m]
(5) Dip_L fault length along dip [m]
(6) StAng Strike Angle (0-180) [Degree]
(7) DipAng Dip Angle (0-180) [Degree]
(8) Rake Rake angle (if SwitchSS/RN = 0) or sense of slip (if SwitchSS/RN = 1 or 2) [-]
SwitchSS/RN = 1: Left lateral (-1) right lateral (1)
SwitchSS/RN = 2: reverse (-1) normal (1)
(9) a Rate-and-State Parameter "a" [-]
(10) b Rate-and-State Parameter "b" [-]
(11) Dc Rate-and-State Parameter "D_c" [m]
(12) Theta_i Initial value of Rate-and-State Parameter θ_i [s]
(13) V_i Initial velocity V_i [m/s]
(14) Fric_i Initial Friction μ_i [-]
(15) Sig0 Normal stress at the surface [Pa]
(16) SigGrad Normal stress gradient with depth [Pa/m]. Zero for uniform normal stress
(17) V_Const Constant velocity (if non-zero, it will only slip at this velocity) [m/s]
(18) MaxLeng Discretized element length [m]

The friction parameters and initial conditions (9-16) can be adjusted after discretization. Conversely, the geometry and rake angle (1-8 and 18) cannot be adjusted once the stiffness matrix is built (i.e., after running the *RUN_BUILD_INPUT.jl*).

The geometry parameters (1-8) should be defined as illustrated in figure 5.

The initial condition of rate and state friction law is

$$\mu_i = \mu_0 + a \log\left(\frac{V_i}{V_0}\right) + b \log\left(\frac{V_0 \theta_i}{D_c}\right). \tag{6}$$

Every variable and initial condition of equation 6 is defined in the *Input_BulkFaultGeometry.txt*, except μ_0 and V_0. Note that these two parameters are interdependent, and therefore, one can set V_0 arbitrarily. In Quake-DFN, $V_0 = 10^{-9}$m/s, and μ_0 is determined by equation 6. But instead of V_0, one may want to set μ_0 as an initial value since it is a laboratory-measurable quantity. This can done by using equation 6. For example, if one wants to fix μ_0 and make V_i be calculated correspondingly, one can calculate

$$V_i = V_0 exp\left(\left(\mu_i - \mu_0 - b \log\left(\frac{V_0 \theta_i}{D_c}\right)\right) / a\right) \tag{7}$$

with the desired μ_0. Putting this initial velocity in the input will set μ_0 to the desired value.

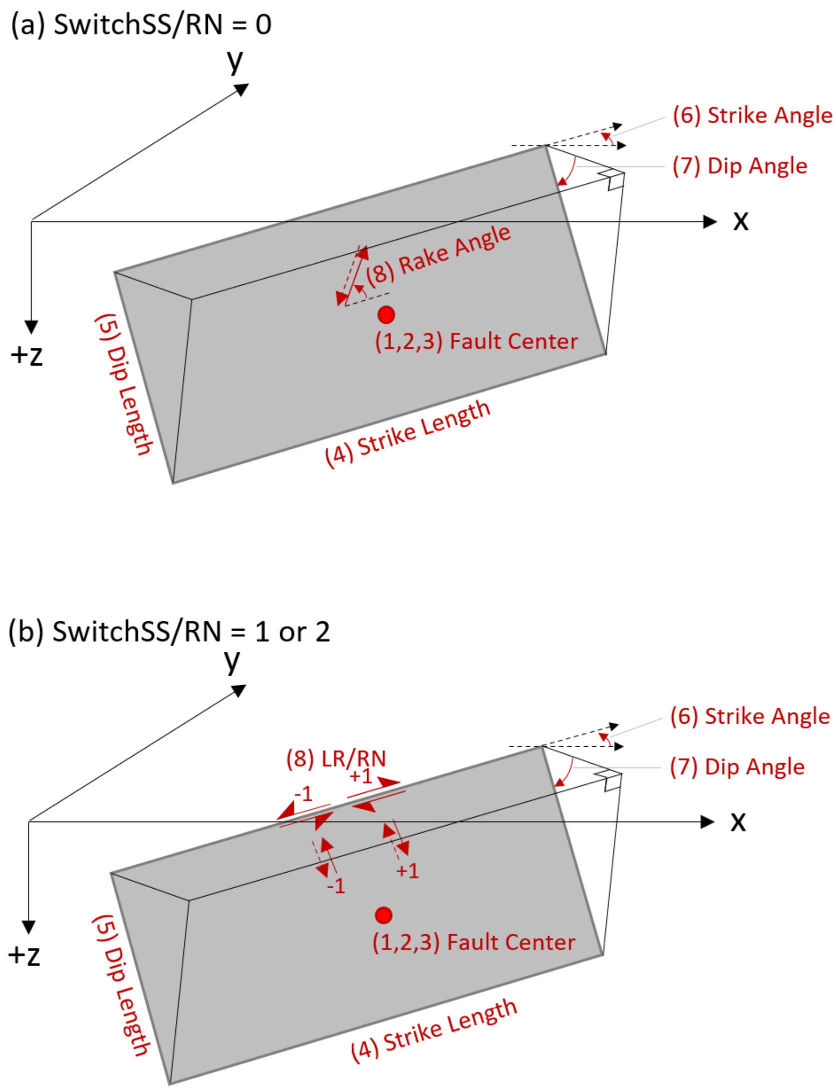


Figure 5. Definitions of the fault geometry parameters. (a) in case SwitchSS/RN = 0. (b) in case SwitchSS/RN = 1 (strike-slip mode) or 2 (normal and reverse mode)

2.1.1 Stress application to complex fault systems

Each bulk element is characterized by a distinct rake angle that corresponds to the prevailing regional stress field. Quake-DFN provides a function to compute and implement the initial stress (friction) and rake angle based on any applied stress field. This process can be executed using the *Tools/StressApplytoBulkGeometry_Homogeneous.jl* after the geometry has been built. This code reads the *Input_BulkFaultGeometry.txt* and modifies it based on the applied stress field. An illustrative example can be found in the section titled "Example Simulations 3.7 Complex Fault Geometry 2 – Complex Fault Network in Homogeneous Stress Field."

2.2. Discretize the fault and build stiffness matrices: ***RUN_BUILD_INPUT.jl*** and ***Input_Discretized.jld2***

The *RUN_BUILD_INPUT.jl* discretizes the input geometry and builds stiffness matrices based on Okada (1992) formulation. The stiffness matrix defines shear and normal stress change driven by a slip of an element. For N number of elements, one stiffness matrix build requires N×N times of Okada 1992 calculations. So, if N is large, this calculation may take a long time. However, once the stiffness matrix is built, it can be used for a wide range of parameter sets, as the input parameter and initial condition change do not require re-calculation of the stiffness matrix.

If *N* is large, the stiffness matrix will occupy large memory, which is often a critical problem for large-scale simulation. The H-Matrix is a compressed version of the full matrix. To use H-Matrix, one should run *RUN_BUILD_HMatrix_Structure.jl* first, then *RUN_BUILD_INPUT.jl* with HMatrixCompress = 1. Note that when the element size is very large, make sure to set SaveOriginalMatrix = 0 so that the original matrix will not occupy memory space.

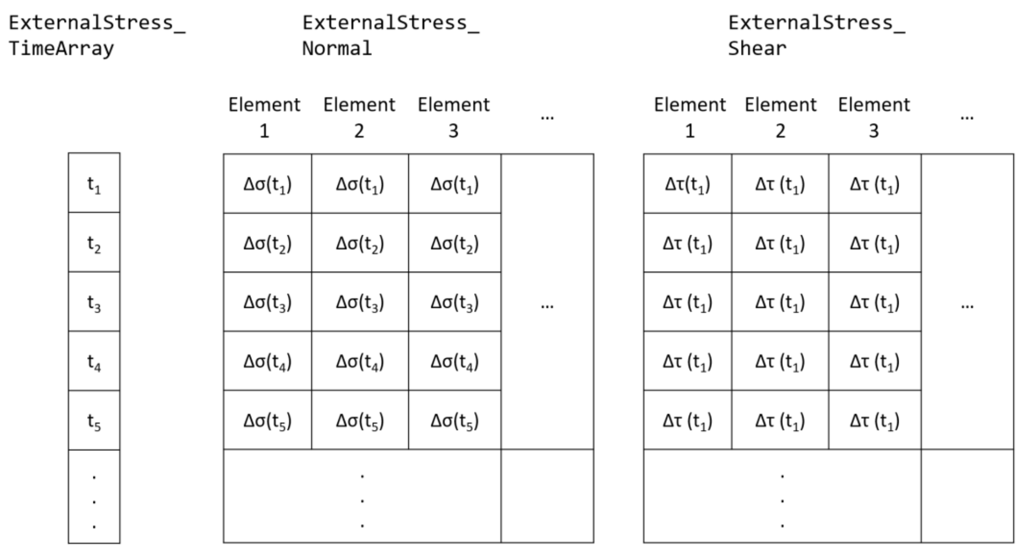
The calculated stiffness matrix and input parameters are saved as *Input_Discretized.jld2*.

2.3. Parameter adjustment for discretized elements: ***QuickParameterAdjust.jl***

QuickParameterAdjust.jl can change parameters and initial state at each individual discretized element level. Therefore, complex heterogeneity can be implemented here. This parameter adjustment does not require re-discretization. Hence, sensitivity tests can be easily implemented in this file. Once faults are discretized, one can test a wide range of parameter sets in here. This file is automatically run right before the simulation.

2.4. External Stress change: ***Input_ExternalStressChange.jld2***

This file is optional. With this file, any time-dependent external stress can be applied to each element. The element count in this file should be equal to the discretized element count in *Input_Discretized.jld2*. Otherwise, the simulation will neglect the file. The file should contain three variables (variable name should be identical to): (1) ExternalStress_TimeArray, (2) ExternalStress_Normal, and (3) ExternalStress_Shear. Each variable should contain the following information.



Once the simulation began, the simulator automatically interpolates shear and normal stress change from the variables.

2.4.1 ExternalStressCalculation/PressureCalculation_Rudnicki.jl

This is an example that generates *Input_ExternalStressChange.jld2*. This file should be run after discretization. Once run, it reads the location and orientation of each discretized element stored in the *Input_Discretized.jld2* file and calculates the normal and shear stress change from spherical pressure diffusion (Rudnicki, 1986; Segall and Lu, 2015). The calculated shear and normal stress are rotated to the fault slip orientation. In the file, we should define several parameters as below.

```
FlowRate=100 # kg/s
PressureOrigin=[0.0, 0.0,-1500]; # Custom Faults
Permeability = 1e-16;
Viscosity = 0.4e-3;
SkemptonCoeff=0.75;
PoissonRatio_Undrained=0.3;
FluidDensity_Ref = 1e3;
```

See Segal and Lu (2015) for more information.

2.5. Conduct Simulation: ***RUN_QUAKEDFN.jl*** (or ***RUN_QUAKEDFN_Hmatrix.jl***)

RUN_QUAKEDFN.jl reads *Input_Discretized.jld2* (and additionally *QuickParameterAdjust.jl* and *Input_ExternalStressChange.jld2* if defined) and run simulation. By setting DtPlot = 1, it generates the $\log_{10}$(dt) vs $\log_{10}(V)$ plot. This plot presents the reference timestep size at a given maximum velocity. Simulation time and time step strategy should be defined in this file.

```
########################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

These variables define total simulation and recording length. The simulation will be conducted for ***TotalStep***. The simulator stores the simulation results of every ***RecordStep*** in memory (others will be discarded). The recorded result will be saved to the *Result* folder every ***SaveStep***s. In the above example, the simulation will be conducted for 10000 steps, and the simulation progress is saved two times (at the step 5000 and 10000) in the *Result* folder. The SaveStep should be smaller than TotalStep (otherwise, no result will be saved) and better be a divisor of the TotalStep. The result file will contain information on a total of 1000 recorded steps (every 10 simulation steps).

```
########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based
on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
RuptureTimeStepMultiple = 3
VerticalLengthScaleforM = 0 # if 0, Mass is automatically determined based on the
fault length (radiation damping dominated). If not, M = VerticalLengthScaleforM *
density / 2

# Manually adjust time step below. No change when 0.0
TimeSteppingAdj =
    [0.0  0.0  0.0  0.0;   # Time step size
     0.0  0.0  0.0  0.0]   # Velocity
```

Time stepping logic (*dt* velocity dependence) is defined here. Simulation timestep size is mainly dependent on the maximum velocity of all the elements. If ***TimeStepOnlyBasedOnUnstablePatch*** = 1, the time step is only dependent on the unstable (*a-b* < 0) patches. ***TimeStepOnlyBasedOnUnstablePatch*** = 1 is faster and, most of the time, okay.

We set several different scenarios for time stepping. If **TimeStepPreset** = 1, the simulator uses a conservative scenario (smaller step), and if 4, it uses the most optimistic scenario (larger step).

One can manually adjust the time-step skim in the table **TimeSteppingAdj**. The first low represents the step size, and the second low represents the corresponding maximum velocity. This is illustrated in figure 6. Any element with zero uses the preset timestep.

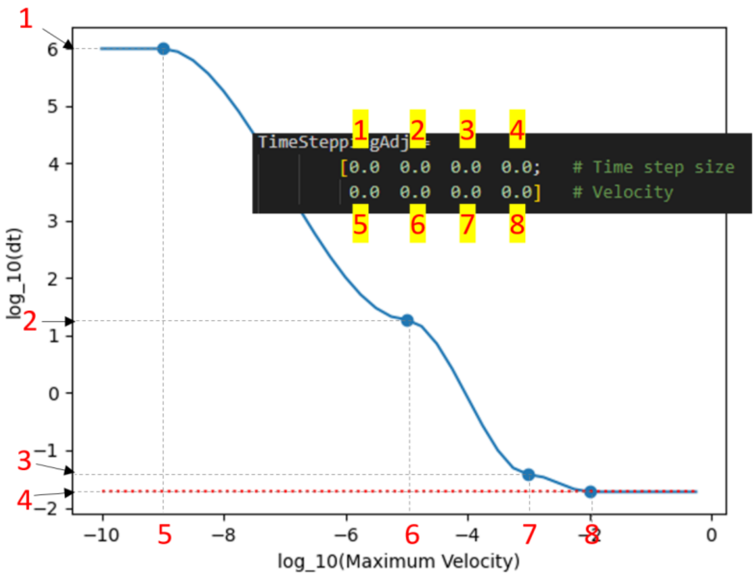


Figure 6. Illustration for ***TimeSteppingAdj***.

```
############################## Plots before run? ################################
DtPlot = 1 # 1 will plot dt vs maxV
GeometryPlot = 0 # 1 will plot a-b
```

Setting 1 will give a corresponding plot at the beginning of the simulation.

During the simulation, Julia REPL window updates the simulation status as follows.

(1)	(2)	(3)	(4)	(5)
0.00100	0.00002	1.556e-02	1.898e-02	1
0.00200	0.00002	1.673e-02	1.898e-02	1
0.00300	0.00002	1.733e-02	1.898e-02	1
0.00400	0.00002	1.785e-02	1.898e-02	1
0.00500	0.00002	1.836e-02	1.898e-02	1
0.00600	0.00003	1.889e-02	1.898e-02	1
0.00700	0.00003	1.942e-02	1.898e-02	1
0.00800	0.00003	1.997e-02	1.898e-02	1

Each column represents:

(1) Simulation progress (Simulation will be over when this becomes 1)

(2) the time in the simulation [day]

(3) maximum velocity

(4) timestep size [s]

(5) Is a high-velocity solver being used? (if 1, high-velocity solver (Im et al., 2017) is being used for high-velocity elements

2.6. Large-scale simulations using H-Matrix

Large-scale simulation requires a large number of elements N. Since memory consumption and simulation time increase with N^2, if N is large, the simulation requires an impractical size of memory and simulation time. This problem can be alleviated by using H-Matrix compression (Hackbusch, W. 1999). Quake-DFN allows the H-Matrix compression for large-scale simulations (figure 7). See Figure 7 for memory saving and simulation speed improvement. An illustrative example can be found in the section titled "Example Simulations 3.8 H-Matrix"

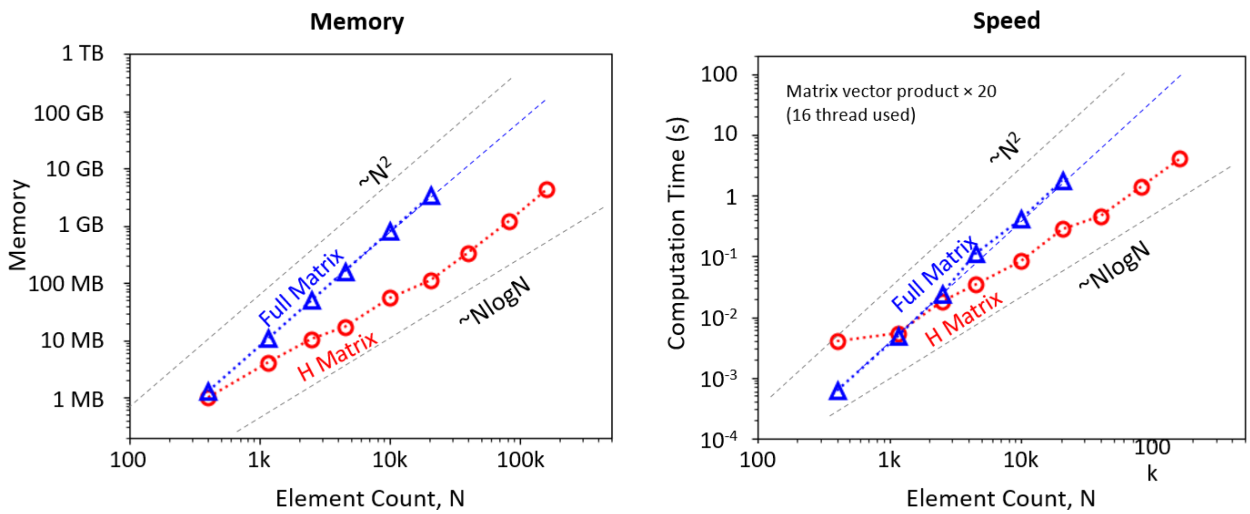
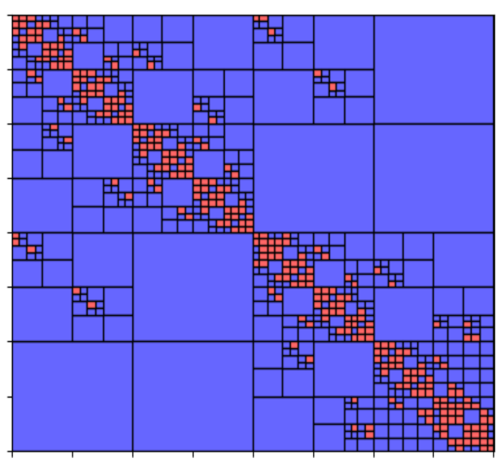


Figure 7. Memory compression and speed change by using H-Matrix approximation.

The simulation workflow to use H-Matrix is shown in figure 3. To use the H-Matrix, H-Matrix structure should be built by running *RUN_BUILD_HMatrix_Structure.jl*. This will provide the H-Matrix structure plot similar to below



The blue area is a compressed matrix, while the red area remains as a full matrix. Once run, *Input_HmatrixStructure.jld2* file will be built.

Now H-Matrix input file can be built by running *RUN_BUILD_INPUT.jl*. Make sure to set HMatrixCompress = 1. If HMatrixCompress = 1 without *Input_HmatrixStructure.jld2*, it will create error. If the element size is large, make sure to set SaveOriginalMatrix = 0.

```
##########################################################################
########################## Hmatrix compress? #############################
HMatrixCompress = 1
SaveOriginalMatrix = 1

#####----- Hmatrix compression Tolerance ----#####
Tolerance = 1e3
##########################################################################
##########################################################################
```

Once done, use *RUN_QUAKEDFN_Hmatrix.jl* to conduct simulations. The simulation parameters are identical to *RUN_QUAKEDFN.jl* (section 2.5).

3. Example Simulations

Here are some brief instructions for provided examples in the *InputGeometryExamples* folder. **All examples here are coarsely discretized for quick testing**. To resolve nucleation length correctly, one can make it finer by adjusting "MaxLeng" in the *Input_BulkFaultGeometry.txt*.

3.1 BP5QD

(1) Run *InputGeometryExamples/Example1_BuildGeometry_BP5QD.jl*

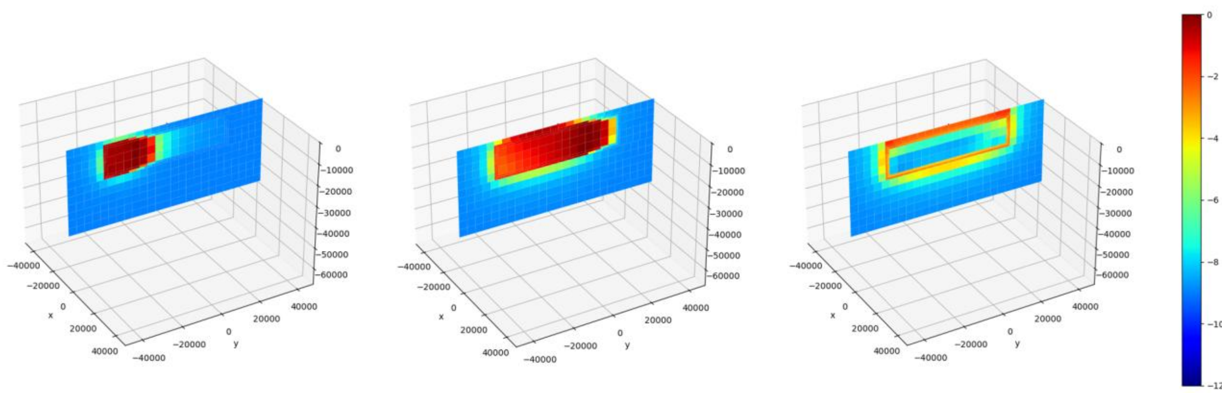
(2) Run *RUN_BUILD_INPUT.jl*

(3) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 1000 # Total simulation step
SaveStep = 1000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

Simulation results (*Results/2_3DPlot_and_Animation.jl*)

Plotstep: 30, 50, 100



Note that the result does not plot loading faults because they are typically very large. If one want to plot the loading fault, change `LoadingFaultPlot = 1` in *Results/2_3DPlot_and_Animation.jl*.

To reproduce the BP5QD benchmark test (Jiang et al., 2022) correctly, the below two parameters in *InputGeometryExamples/Example1_BuildGeometry_BP5QD.jl* should be changed to 1000.0.

```
UntableMaximumSegLength = 4000.0
StableMaximumSegLength = 4000.0
```

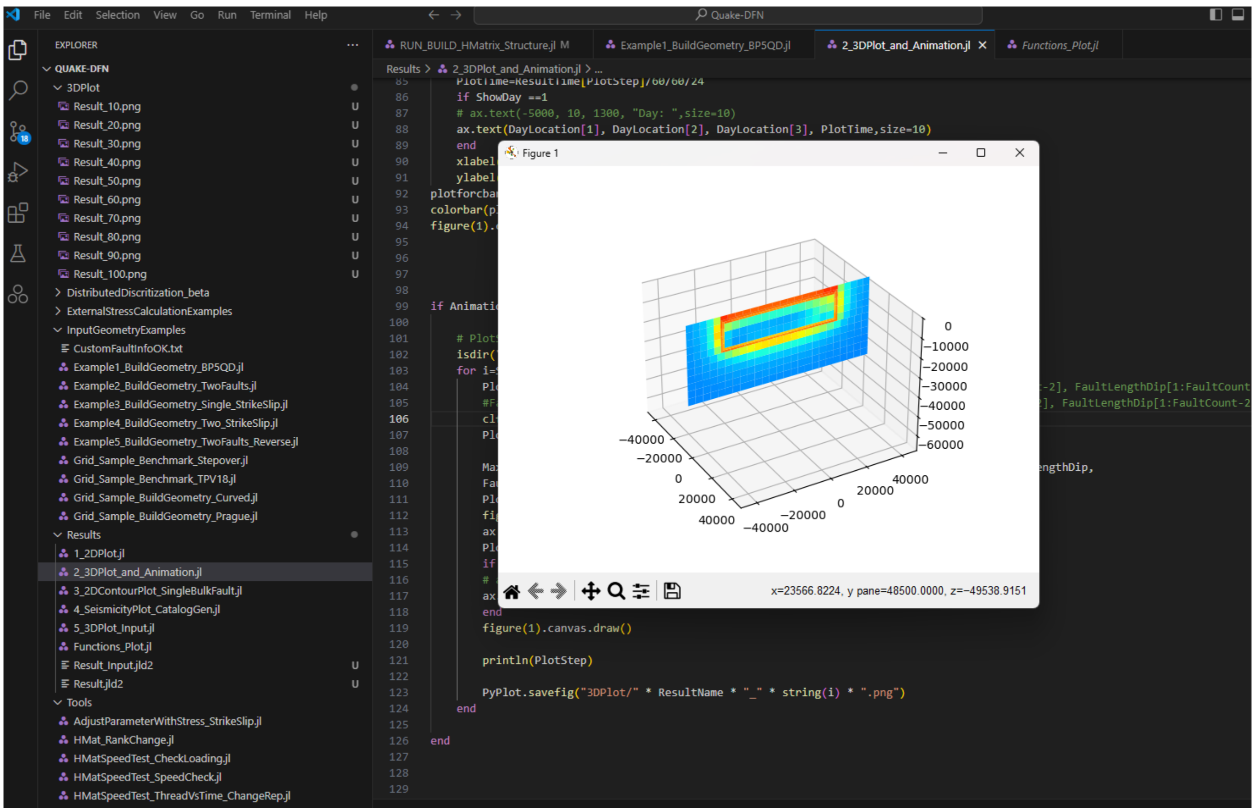
Save multiple snapshots

In the Results/2_3DPlot_and_Animation.jl,

```
############# Saving Multiple Figures ###############
Animation_Save = 1 # 1 for save
StepBegin = 10 # first record step
StepEnd = 100
StepInterval = 10
###################^^^^^^^^^^^^^^^###################
```

and run.

It will generate the folder *3DPlot*, in which the snapshots are saved as png file. This is useful for generating the rupture video files.



Extra: BP5QD Grid Size Adjustment

(1) Run *InputGeometryExamples/Example1_BuildGeometry_BP5QD.jl*

(2) Open the *Input_BulkFaultGeometry.txt* and adjust MaxLeng of first fault (4th line) to 1000.0 as below and save it.

```
SwitchSS/RN ShearMod    PoissonRatio    R_Density    Crit_TooClose    TooCloseNormal_Multiplier    Minimum_NS
1.0 3.2038e10   0.25    2670.0  1.05    0.6 2.0e6
Ctr_X   Ctr_Y   Ctr_Z   St_L    Dip_L   StAng   DipAng  LR/RN   a   b   Dc  Theta_i V_i Fric_i  Sig0    SigGrad V_Const MaxLeng
0.0 -24000.0    10000.0 12000.0 12000.0 90.0    90.0    1.0 0.004   0.03    0.13    1.3e8   0.03    0.6 2.5e7   0.0 0.0 1000.0
0.0 6000.0  10000.0 48000.0 12000.0 90.0    90.0    1.0 0.004   0.03    0.14    1.4e8   1.0e-9  0.6 2.5e7   0.0 0.0 4000.0
0.0 31500.0 10000.0 1000.0  16000.0 90.0    90.0    1.0 0.031000000000000003    0.03    0.14    1.4e8   1.0e-9  0.6 2.5e7   0.0 0.0 4000.0
0.0 -31500.0    10000.0 1000.0  16000.0 90.0    90.0    1.0 0.031000000000000003    0.03    0.14    1.4e8   1.0e-9  0.6 2.5e7   0.0 0.0 4000.0
```

This will make the grid size of the first fault 1000.0.

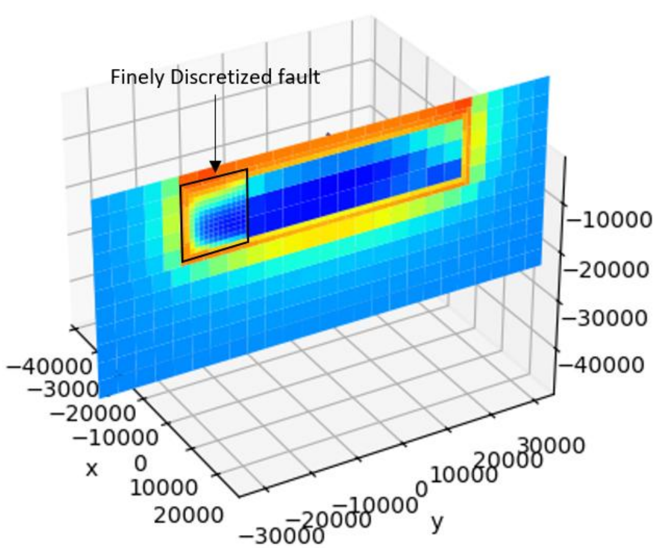
(3) Run *RUN_BUILD_INPUT.jl*

(4) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 2000 # Total simulation step
SaveStep = 2000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

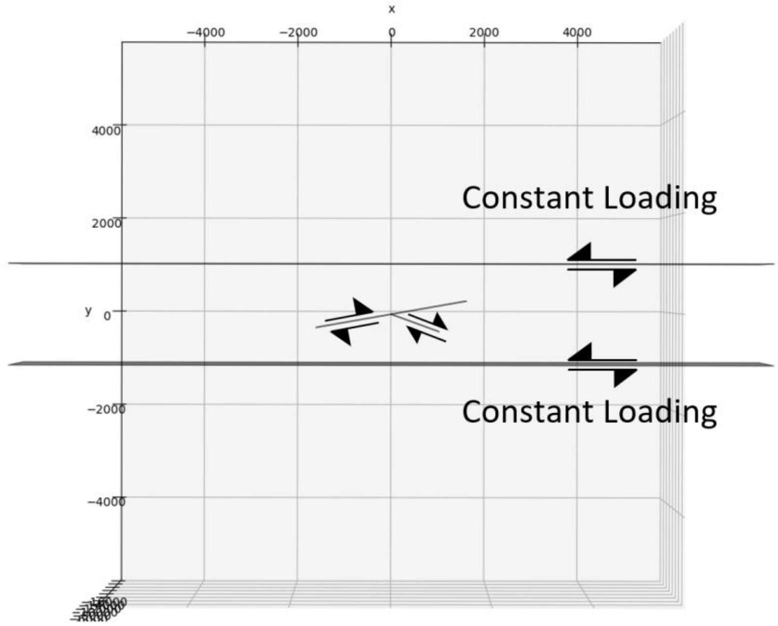
Simulation results (*Results/2_3DPlot_and_Animation.jl*)

Plotstep: 200



3.2 Two-Fault System with Constant Loading

(1) Run *InputGeometryExamples/ Example2_BuildGeometry_TwoFaults_Loaded.jl*



Input_BulkFaultGeometry.txt

```
SwitchSS/RN ShearMod    PoissonRatio    R_Density   Crit_TooClose   TooCloseNormal_Multiplier   Minimum_NS
1.0 2.0e10  0.2 2400.0  1.05    0.6 2.0e6
Ctr_X   Ctr_Y   Ctr_Z   St_L    Dip_L   StAng   DipAng  LR/RN   a   b   Dc  Theta_i V_i Fric_i  Sig0    SigGrad V_Const MaxLeng
738.605814759156    130.23613325019778  1500.0  1500.0  3000.0  10.0    90.0    1.0 0.003   0.006   0.0005  1000.0  1.0e-9  0.6 2.0e6   6000.0  0.0 200.0
-738.605814759156   -130.23613325019778 1500.0  1500.0  3000.0  10.0    90.0    1.0 0.003   0.006   0.0005  1000.0  1.0e-9  0.6 2.0e6   6000.0  0.0 200.0
469.8463103929542   -171.01007166283435 1500.0  1000.0  3000.0  160.0   90.0    1.0 0.003   0.006   0.0005  1000.0  1.0e-9  0.6 2.0e6   6000.0  0.0 200.0
0.0 1000.0  3500.0  15000.0 7000.0  0.0 90.0    -1.0    0.01    0.005   0.001   1.0e6   1.0e-9  0.6 3.0e6   0.0 1.0e-9  1.0e10
0.0 -1000.0 3500.0  15000.0 7000.0  0.0 90.0    -1.0    0.01    0.005   0.001   1.0e6   1.0e-9  0.6 3.0e6   0.0 1.0e-9  1.0e10
```

Input_BulkFaultGeometry.txt shows that the two horizontal faults have constant velocities. If constant velocity is assigned, the faults will only slip at the velocity with all the other parameters ignored. These two faults are the loading fault that applies constant loading to the two smaller faults in between. These loading faults does not need to be discretized, hence very large MaxLeng value is assigned.

While the fault geometry appears to have only two faults in between the loading faults, the non-loading faults are 3 pieces in the Input_BulkFaultGeometry.txt. This is due to the "joint separation" discussed in figure 4.

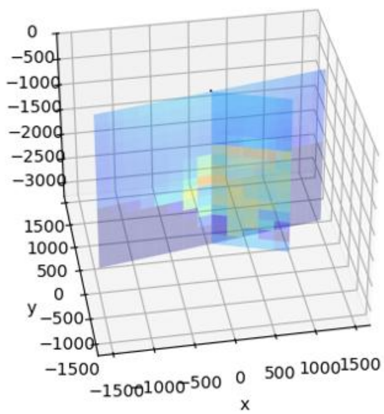
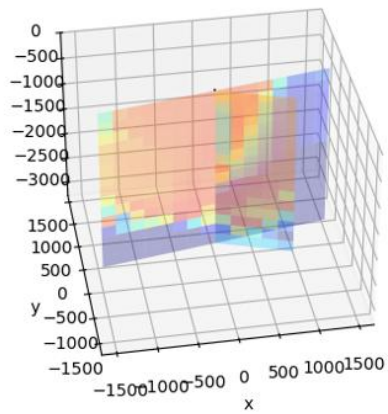
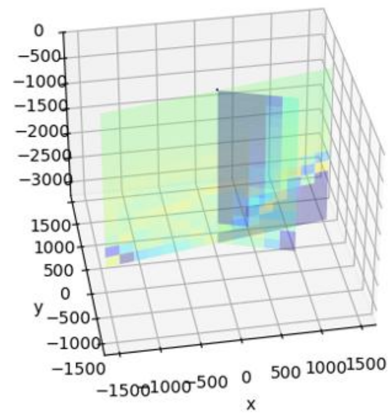
(2) Run *RUN_BUILD_INPUT.jl*

(3) Run *RUN_QUAKEDFN.jl*

```
######################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 10000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

Simulation results (*Results/2_3DPlot_and_Animation.jl*)

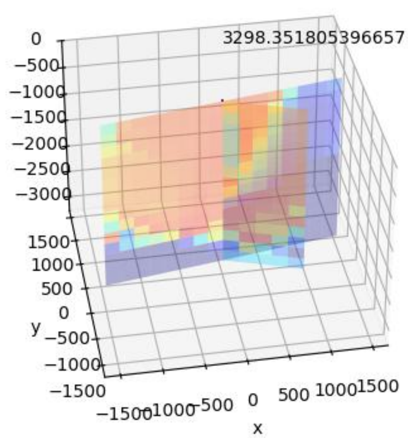
Plotstep: 700, 800, 900 (set "Transparent = 1" for transparent plot)

One can turn on the plot time by setting

```
ShowDay = 1 # If 1, day is shown in the location
DayLocation = [0,0,1000]
```

Plotstep: 800



The rupture occurred on day 3298.

3.3 Single fault with a pressure source

(1) Run *InputGeometryExamples/Example3_BuildGeometry_Single_StrikeSlip.jl*

(2) Run *RUN_BUILD_INPUT.jl*

(3) Run *ExternalStressCalculationExamples/PressureCalculation_Rudnicki.jl*

```
FlowRate=100 # kg/s
PressureOrigin=[0.0, 0.0,-2000]; # Custom Faults
Permeability = 1e-16;
Viscosity = 0.4e-3;
SkemptonCoeff=0.75;
PoissonRatio_Undrained=0.3;
FluidDensity_Ref = 1e3;
```

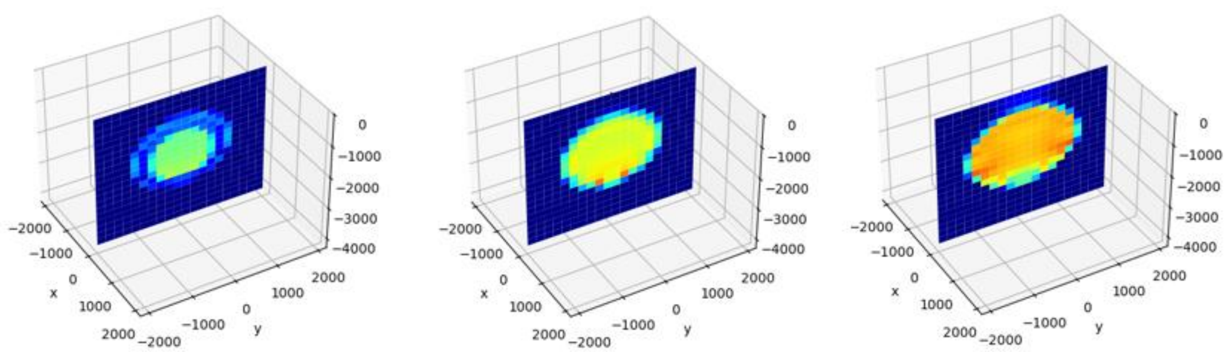
This applies a spherical pressure source to the center of the fault. Shear and normal stress change plot will show up at the end of the simulation.

(4) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 10000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

Simulation results
Velocity

Plotstep: 500, 700, 1000

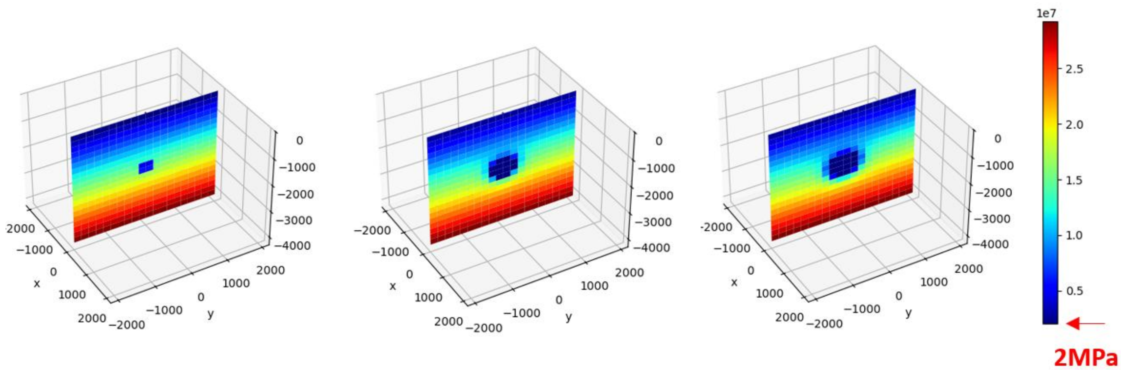


Normal Stress Plot

In *Results/2_3DPlot_and_Animation.jl*,

```
################## What to Plot ? ###################
# PlotInput=log10.(ResultV[PlotStep,:]); ColorMinMax=[-12,0]
PlotInput= Result_NormalStress[PlotStep,:] ; ColorMinMax=0
# PlotInput=log10.(ResultPressure[PlotStep,:]); ColorMinMax=[3,6]
# PlotInput= Result_NormalStress[PlotStep,:] -  Fault_NormalStress;
ColorMinMax=[-1e6,1e6]
# PlotInput=ResultDisp[PlotStep,:]; ColorMinMax=0
#############--------------------------#############
```

Plotstep: 100, 300, 1000



Effective normal stress decreases due to the pressurization. Note that the minimum normal stress is 2Mpa due to the "(7) Minimum_NS" in Input_BulkFaultGeometry.txt (section 2.1).

Extra: heterogenous fault properties

(1) Run *InputGeometryExamples/Example3_BuildGeometry_Single_StrikeSlip.jl*
(2) Run *RUN_BUILD_INPUT.jl*
(3) Run *ExternalStressCalculation/PressureCalculation_Rudnicki.jl*
(4) Open *QuickParameterAdjust.jl* and put the following and save.

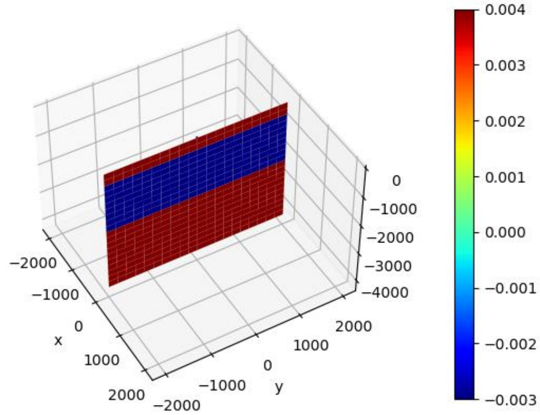
```
######################################################################
########################## Direct Adjust #############################
for i=1:FaultCount
   if 500 > FaultCenter[i,3] || FaultCenter[i,3] > 2000
     Fault_a[i] = 0.01
   end
end
###^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^###
######################################################################
```

This changes the rate and state 'a' parameter and will make the fault stable in shallow (<500m) and deep (>2000m) areas.

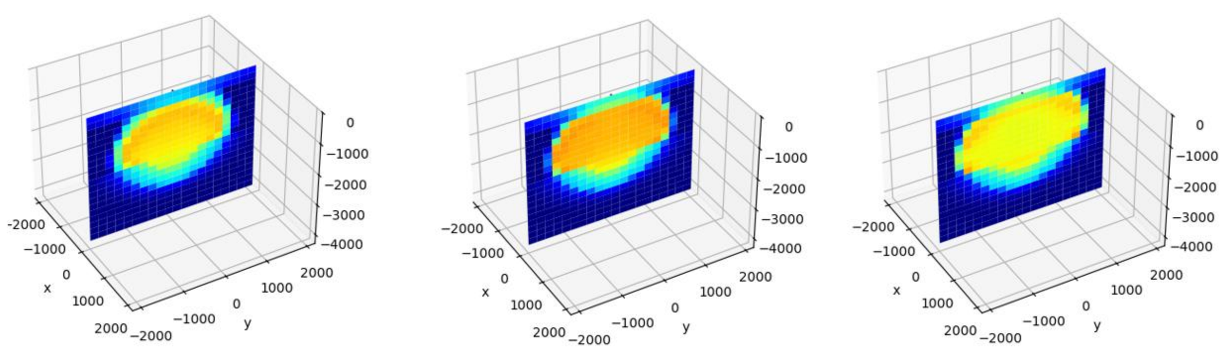
(4-1) *Optional* Run *Plot_Input.jl* to check a-b

```
################# Which parameter want to plot? #######################
    ColorMinMax = 0
    # PlotInput = log10.(Fault_Theta_i); ColorMinMax = 0
    # PlotInput = log10.(Fault_V_i); ColorMinMax = 0
    # PlotInput =Fault_NormalStress; ColorMinMax = 0
    # PlotInput =KoverKC ;ColorMinMax=[0,5]
    # PlotInput =UnderResolved ;ColorMinMax=[0,1]
    PlotInput = Fault_a - Fault_b; ColorMinMax = 0
    # PlotInput =  Fault_BulkIndex; ColorMinMax = 0
```

This will show *a-b* value as follows



(5) Run *RUN_QUAKEDFN.jl*
Simulation results
Plotstep: 500, 600, 700



Rupture propagation is blocked at shallow (<500m) and deep (>2000m) areas.

3.4 Two strike-slip faults with a pressure source

(1) Run *InputGeometryExamples/Example4_BuildGeometry_Two_StrikeSlip.jl*

(2) Run *RUN_BUILD_INPUT.jl*

(3) Run *ExternalStressCalculation/PressureCalculation_Rudnicki.jl*

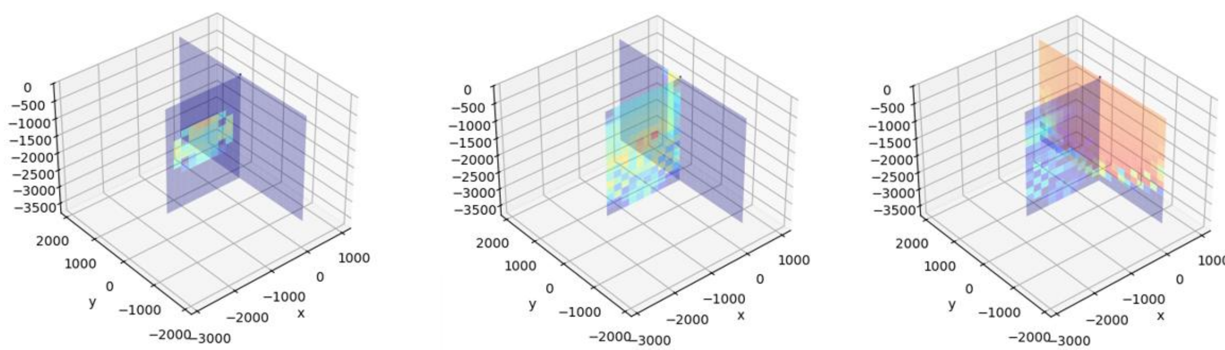
Make sure to Change Pressure origin(fault center of smaller fault)

```
FlowRate=100 # kg/s
PressureOrigin=[-1000.0, 0.0,-1500]; # Custom Faults
Permeability = 1e-16;
Viscosity = 0.4e-3;
SkemptonCoeff=0.75;
PoissonRatio_Undrained=0.3;
FluidDensity_Ref = 1e3;
```

(4) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate

########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based
on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
```

Simulation results
Plotstep: 200, 500, 1000



Extra: Stress Based Initial Condition

One can adjust the initial condition based on the local stress field. Consider rate and state friction law:

$$\mu_i = \mu_0 + a \log\left(\frac{V_i}{V_0}\right) + b \log\left(\frac{V_0 \theta_i}{D_c}\right)$$

To conduct the simulation, one should define any three of the following four: μ, μ_0, V_i, θ_i.

Practically, μ_i should be calculated based on the local stress field. The μ_0 is a measurable parameter (together with V_0). The two initial conditions, V and θ, are relatively hard to measure. However, we may estimate the initial θ_i since its maximum value is the elapsed time from the last rupture (maximum $d\theta/dt$ = 1 s/s), while the initial V has no limit. Hence, one may want to fix the initial θ and determine V correspondingly.

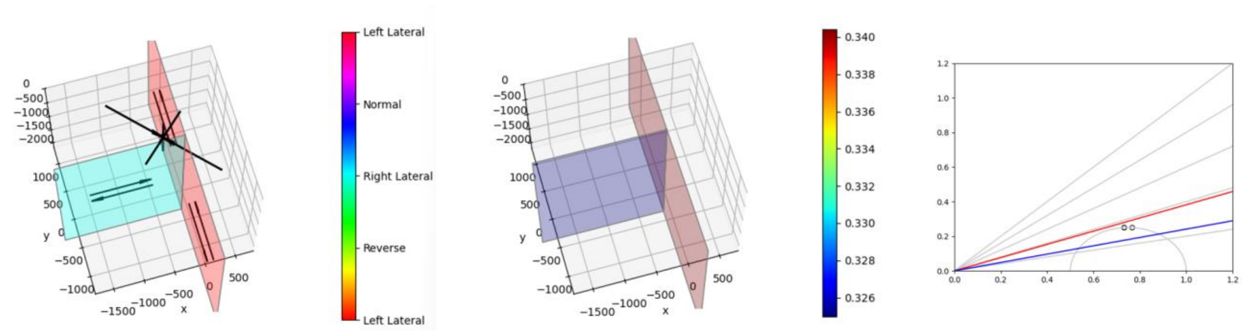
There are two different ways to implement this. (i): before the discretization and (ii) after the discritization. The first method can be implemented as follows.

(1) Run *InputGeometryExamples/Example4_BuildGeometry_Two_StrikeSlip.jl*

(2) Run *Tools/StressApplytoBulkGeometry_Homogeneous.jl*

```
    ###### build Principal Stress. Compression Positive. Only Ratio Matters!
    PrincipalStressRatioX = 0.5
    PrincipalStressRatioY = 1.0
    PrincipalStressRatioZ = 0.5
    StressRotationStrike = 43 # degree
    StressRotationDip = 0 # degree

    MaximumTargetVelocity = 1e-11
    ConstantTheta = 1e10
    Fault_a_Rev = 0.0
    Fault_b_Rev = 0.0
   Fault_Dc_Rev = 0.0
    V_p = 1e-5
    V_r = 1e-2
    MinFrictionAllowed = 0.05

    MinimumNormalStressAllowed = 1e6
    StressOnSurface_Sig1Orientation = 10e6 # pascal
    StressGredient_Sig1Orientation = 0 # pascal/m
    FaultSegmentLength = 0 # if 0, segment length will be unchanged

    LoadingFaultAdjust = 0 # if 0, Loading fault sense of slip will not be
changed
    LoadingFaultInvert = 1 # if 1, loading fault sense of slip become inverted
```

Once run, Input_BulkFaultGeometry.txt is changed according to the above input. μ is calculated by the given stress change. Three plots will pop up as below.



These plots illustrate the slip and friction characteristics derived from the applied stress fields. The first plot displays the applied principal stresses. The Mohr circle (σ1 – σ3) demonstrates the optimality of two fault orientations within this context. Both orientations are optimally oriented with the given stress field. The red and blue lines represent the equivalent static and dynamic friction, which are estimated based on the rate and state friction law. In the input, we set `MaximumTargetVelocity = 1e-11` *resulting in an initial velocity for the most optimally oriented fault of* 10^{-11} *m/s. This condition places the fault above the self-acceleration criterion outlined in the rate and state friction law (VΘ/Dc > 1), rendering the faults critically stressed.*

(3) Run *RUN_BUILD_INPUT.jl*

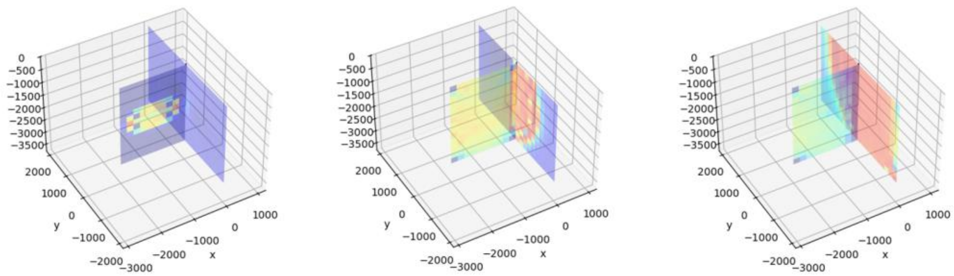
(4) Run *ExternalStressCalculation/PressureCalculation_Rudnicki.jl*

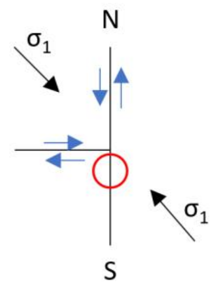
```
FlowRate=100 # kg/s
PressureOrigin=[-1000.0, 0.0,-1500]; # Custom Faults
Permeability = 1e-16;
Viscosity = 0.4e-3;
SkemptonCoeff=0.75;
PoissonRatio_Undrained=0.3;
FluidDensity_Ref = 1e3;
```

(5) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate

########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based
on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
```

Simulation results
Plotstep: 80, 110, 150





The simulation result is similar to the original simulation, except that the main fault rupture initially propagates in the opposite direction. This is due to the applied stress field. The smaller fault is right lateral now due to the maximum stress orientation of 43°W. The smaller fault slip will cause normal stress reduction at the south half of the main fault, making the rupture propagate toward the south.

The stress-based initial condition change can also be conducted by uncommenting the following lines in QuickParameterAdjust.jl.

```
    ##########################################################################
    #######  Calculation of initial state from stress orientation #######
    # MaxStressOrientation = 100. # between 0-180 degree
    # StressRatioMaxOverMin = 0.5
    # MinFrictionAllowed = 0.1 # smaller than this friction is not allowed
    # StressGradAtMaxOrientation = 10000.0
    # SurfaceStressAtMaxOrientation = 2e6
    # Fault_Theta_i .= 1e10
    # Fault_V_i .= 0.0
    # Friction_0 = ones(FaultCount) * 0.32
    # V0=1e-9;
    # Fault_Friction_i, Fault_NormalStress, Fault_V_i, Fault_Theta_i =
    #             StressDependentFrictionParameters(MaxStressOrientation,
StressRatioMaxOverMin, MinFrictionAllowed,
    #             StressGradAtMaxOrientation, SurfaceStressAtMaxOrientation,
    #             FaultStrikeAngle, FaultDipAngle, Fault_V_i, Fault_Theta_i,
Fault_Friction_i, FaultLLRR,
    #             Fault_a, Fault_b, Fault_Dc, Fault_NormalStress, Friction_0,
FaultCenter)
```

If parameters are adjusted here, one can run RUN_QUAKEDFN.jl right away.

It is convenient to change parameters here since it doesn't require discretization. *However, the* ***changes in QuickParameterAdjust.jl can not change rake angle***.

3.5 Two reverse faults with a pressure source

(1) Run *InputGeometryExamples/Example5_BuildGeometry_TwoFaults_Normal.jl*

```
Input_BulkFaultGeometry.txt
SwitchSS/RN ShearMod    PoissonRatio    R_Density   Crit_TooClose   TooCloseNormal_Multiplier   Minimum_NS
2.0 2.0e10  0.2 2400.0  1.05    0.6 2.0e6
Ctr_X   Ctr_Y   Ctr_Z   St_L    Dip_L   StAng   DipAng  LR/RN   a   b   Dc  Theta_i V_i Fric_i  Sig0    SigGrad V_Const MaxLeng
-577.3502691896258  -0.0    1000.0  2000.0  2309.401076758503   90.0    60.0    -1.0    0.003   0.006   0.0002  1.0e10  1.0e-15 0.33    2.0e6   4500.0  0.0 200.0
363.97023426620234  0.0 1000.0  2000.0  2128.355544951824   90.0    110.0   -1.0    0.003   0.006   0.0002  1.0e10  1.0e-15 0.33    2.0e6   4500.0  0.0 200.0
-363.97023426620234 0.0 3000.0  2000.0  2128.355544951824   90.0    110.0   -1.0    0.003   0.006   0.0002  1.0e10  1.0e-15 0.33    2.0e6   4500.0  0.0 200.0
```

In Input_BulkFaultGeometry.txt, SwitchSS/RN is now 2.0. If this is 1.0, the system is strike-slip, and if this is 2, the system is reverse-normal slip. Now, both fault has -1 at LR/RN. In the Reverse-Normal system, -1 denotes reverse faulting.

(2) Run *RUN_BUILD_INPUT.jl*

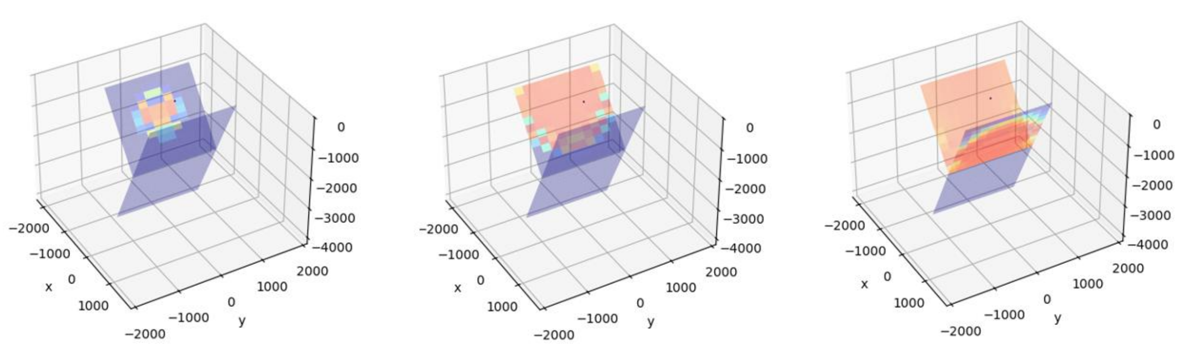
(3) Run *ExternalStressCalculation/PressureCalculation_Rudnicki.jl*

```
FlowRate=100 # kg/s
PressureOrigin=[-577.0, 0.0,-1000]; # Custom Faults
Permeability = 1e-16;
Viscosity = 0.4e-3;
SkemptonCoeff=0.75;
PoissonRatio_Undrained=0.3;
FluidDensity_Ref = 1e3;
```

(4) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 10000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate

########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based
on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
```

Simulation results
Plotstep: 100, 120, 135



3.6 Complex fault geometry 1 – Ridgecrest earthquake

We provide a simplified Ridgecrest earthquake rupture geometry in the folder *InputGeometryExamples*.

(1) Move *InputGeometryExamples/Input_BulkFaultGeometry_Ridgecrest.txt* to the root directory and change the name to *Input_BulkFaultGeometry.txt*.

(2) Run *Run_BUILD_INPUT.jl*

(3) set *QuickParameterAdjust.jl* as follows:

```
############################################################################    #
#######  Calculation of initial state from stress orientation #######
MaxStressOrientation = 85. # between 0-180 degree
StressRatioMaxOverMin = 0.5
MinFrictionAllowed = 0.1 # smaller than this friction is not allowed

StressGradAtMaxOrientation = 6000.0
SurfaceStressAtMaxOrientation = 2e6
Fault_Theta_i .= 1e10
Fault_V_i .= 0.0
Friction_0 = ones(FaultCount) * 0.30
V0=1e-9

Fault_Friction_i, Fault_NormalStress, Fault_V_i, Fault_Theta_i =
 StressDependentFrictionParameters(MaxStressOrientation, StressRatioMaxOverMin,
MinFrictionAllowed, StressGradAtMaxOrientation, SurfaceStressAtMaxOrientation,
FaultStrikeAngle, FaultDipAngle, Fault_V_i, Fault_Theta_i, Fault_Friction_i,
FaultLLRR, Fault_a, Fault_b, Fault_Dc, Fault_NormalStress, Friction_0,
FaultCenter)

    ##########^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#############
    ######################################################################
```

Unlike the example in 3.4 extra, the stress is applied to the discretized geometry. In this case, the rake angle cannot be changed. Hence, it is recommended to use Tools/StressApplytoBulkGeometry_Homogeneous.jl if the applied stress change is significant.

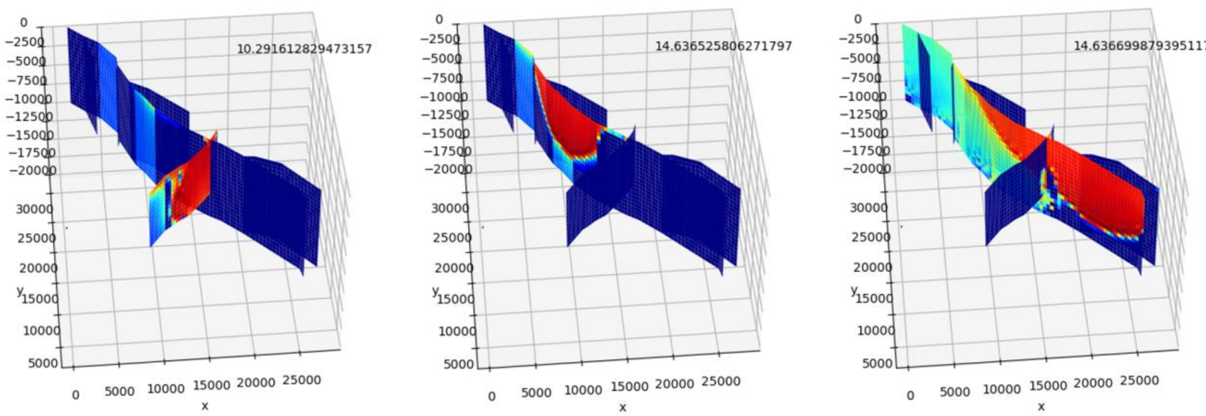
This input geometry has no loading fault. So, if no fault is critically stressed, rupture would not occur. The setting in the QuickParameterAdjust.jl adjusts the initial condition according to the stress field (σ_1 angle 85° defined as fault geometry in Figure 5, $\sigma_3 = 0.5\sigma_1$). The stress is depth-dependent with 2MPa + 6kPa × depth along the maximum stress angle. This setting makes some part of the fault critically stressed (i.e., self-accelerates).

(4) Run *RUN_QUAKEDFN.jl*

```
########################## Simulation Time Set ################################
TotalStep = 5000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
```

Simulation results

PlotStep 50, 160, 260

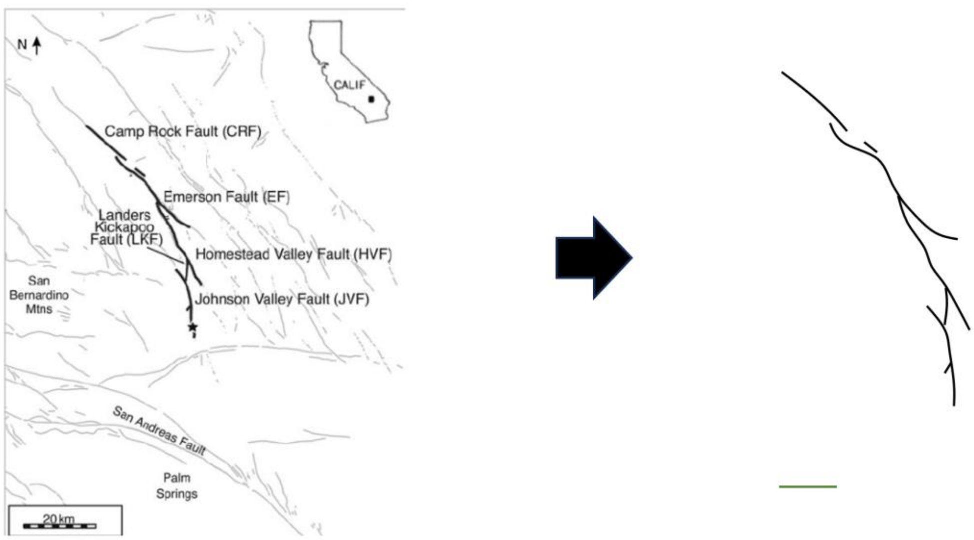


Note that there is a ~4 days gap between the foreshock (left) and the mainshock (middle and right). In actual sequence, it was ~1 day. This is just a simple example of how Quake-DFN can reproduce a realistic earthquake sequence. More tuning is needed for more precise fitting to the actual sequence and its magnitude.

3.7 Complex fault geometry 2 – Complex fault network in homogeneous stress field

Quake-DFN offers a useful tool for simulating earthquake rupture in actual fault geometries by converting JPG images into input files for vertical faults. While we are using PowerPoint for this process, any graphic tool can suffice for generating images.

(1) The initial step involves creating a JPG file that depicts the fault geometry. Utilizing graphic tools in PowerPoint, one can trace an image of the surface rupture from the Landers earthquake (Olsen et al., 1997), as demonstrated in the right image below. Note that the green horizontal line will be used as scale initially set as 10 km. The right image then should be saved as "*Faultimage.jpg*" and stored in *ImageReader* folder in Quake-DFN.



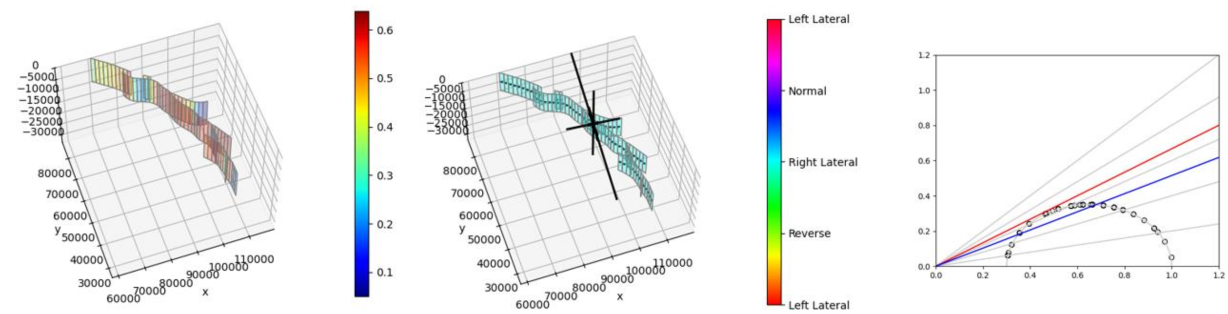
When the image is ready, run *Fault_segmentation_entire_process.py* first. Next, execute BuildGeometry_givencoordinates_shallowfault.jl. An input file, *InputFaultGeometry.jl*, will be created in the Quake-DFN root folder. You can verify the built geometry by running *Plot_BulkFaultGeometry.jl*.

(2) Run Tools/StressApplytoBulkGeometry_Homogeneous.jl.

```
###### build Principal Stress. Compression Positive. Only Ratio Matters!
PrincipalStressRatioX = 0.3
PrincipalStressRatioY = 1.0
PrincipalStressRatioZ = 0.6
StressRotationStrike = -5 # degree
StressRotationDip = 0 # degree


MaximumTargetVelocity = 1e-11
ConstantTheta = 1e10
Fault_a_Rev = 0.0
```

```
    Fault_b_Rev = 0.0
    Fault_Dc_Rev = 0.0
    V_p = 1e-5
    V_r = 1e-2
    MinFrictionAllowed = 0.05

    MinimumNormalStressAllowed = 1e6
    StressOnSurface_Sig1Orientation = 10e6 # pascal
    StressGredient_Sig1Orientation = 0 # pascal/m
```

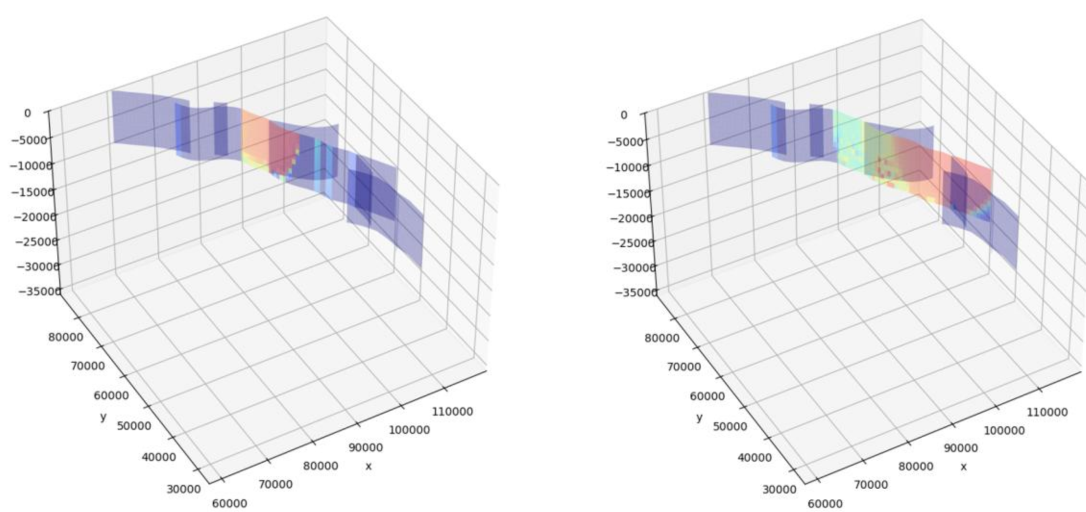
Initial friction and a sense of slip is calculated by the applied stress field.



(3) Run *Run_BUILD_INPUT.jl*

(4) Run *RUN_QUAKEDFN.jl*

PlotStep 50, 100



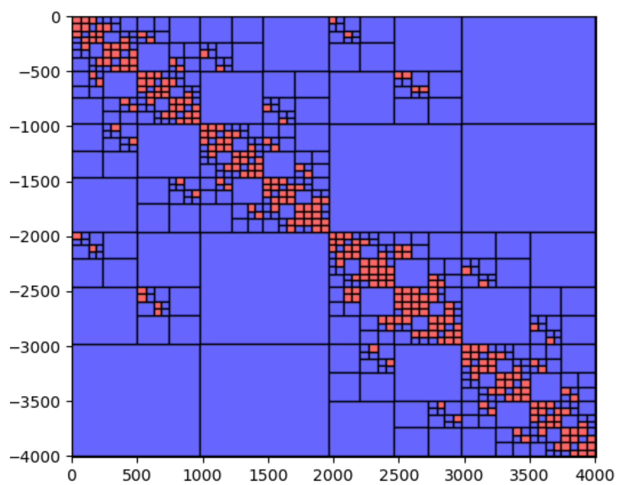
3.8 Hmatrix

(1) Run *InputGeometryExamples/Example1_BuildGeometry_BP5QD.jl* with element size 1000.

```
Unstable_a=0.004
Unstable_b=0.03
Unstable_Dc=0.14
Unstable_Theta=1e9
UntableMaximumSegLength = 1000.0

Stable_a=0.04
Stable_b=0.03
Stable_Dc=0.14
StableMaximumSegLength = 1000.0;
```

(2) Run *RUN_BUILD_HMatrix_Structure.jl*

A H-Matrix structure will be pop-up.



(3) Run *Run_BUILD_INPUT.jl*

```
##########################################################################
########################## Hmatrix compress? ##############################
HMatrixCompress = 1 #
SaveOriginalMatrix = 1  #
#####----- Hmatrix compression Tolerance ----#####
Tolerance = 1e3 # pascal
##########################################################################
##########################################################################
```

Both full matrix stiffness and Hmatrix stiffness will be built.

(4-1) Run *RUN_QUAKEDFN_Hmatrix.jl*

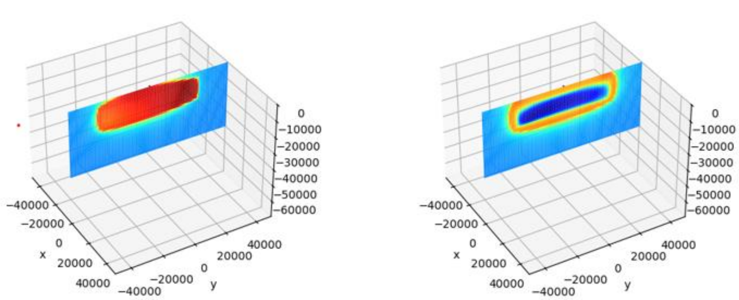
```
ThreadCount = 10 # if zero, it uses current thread count opened in REPL

########################## Simulation Time Set ################################
TotalStep = 5000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate !! should be a factor of SaveStep !!
########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based
on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
RuptureTimeStepMultiple = 3
VerticalLengthScaleforM = 0 # if 0, Mass is automatically determined based on the
fault length (radiation damping dominated). If not, M = VerticalLengthScaleforM *
density / 2
# Manually adjust time step below. No change when 0.0
TimeSteppingAdj =
    [0.0  0.0  0.0  0.0;   # Time step size
     0.0  0.0  0.0  0.0]   # Velocity
```

Simulation Time is 27 s

```
0.97800 0.00339   1.676e-03   5.975e-02    1
0.98000 0.00339   1.671e-03   5.975e-02    1
0.98200 0.00340   1.665e-03   5.975e-02    1
0.98400 0.00341   1.660e-03   5.975e-02    1
0.98600 0.00342   1.655e-03   5.975e-02    1
0.98800 0.00342   1.650e-03   5.975e-02    1
0.99000 0.00343   1.645e-03   5.975e-02    1
0.99200 0.00344   1.640e-03   5.975e-02    1
0.99400 0.00344   1.635e-03   5.975e-02    1
0.99600 0.00345   1.630e-03   5.975e-02    1
0.99800 0.00346   1.625e-03   5.975e-02    1
1.00000 0.00346   1.620e-03   5.975e-02    1
Saved Upto Here
 27.225829 seconds (349.84 M allocations: 29.995 GiB, 7.43% gc time, 1.50% compilation time: 2% of which was recompilation)
```

Simulation results

Plotstep 100, 300



(4-2) Run *RUN_QUAKEDFN.jl*

Since we also saved the original matrix (step 3), we can conduct a simulation with the full matrix.

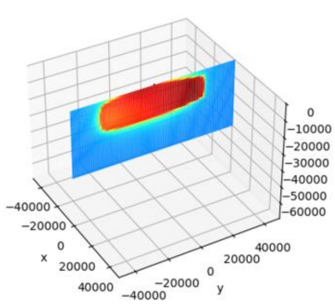
```
########################## Simulation Time Set ################################
TotalStep = 5000 # Total simulation step
SaveStep = 5000 # Automatically saved every this step
RecordStep = 10 # Simulation sampling rate
########################## Time Stepping Setup ################################
TimeStepOnlyBasedOnUnstablePatch = 1 # if 1, time step is calculated only based on the unstable patch
TimeStepPreset = 3 # 1: conservative --> 4: optimistic
RuptureTimeStepMultiple = 3
VerticalLengthScaleforM = 0 # if 0, Mass is automatically determined based on the fault length (radiation damping dominated). If not, M = VerticalLengthScaleforM * density / 2
```

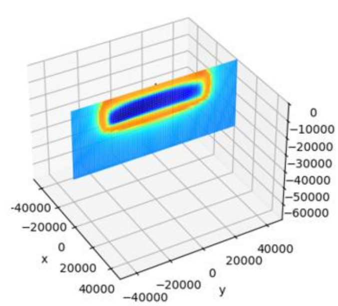
Simulation Time is 39 s

```
0.99000 0.00343   1.642e-03   5.975e-02    1
0.99200 0.00344   1.637e-03   5.975e-02    1
0.99400 0.00344   1.632e-03   5.975e-02    1
0.99600 0.00345   1.627e-03   5.975e-02    1
0.99800 0.00346   1.622e-03   5.975e-02    1
1.00000 0.00346   1.617e-03   5.975e-02    1
Saved Upto Here
 39.939115 seconds (296.89 M allocations: 8.576 GiB, 1.87% gc time, 0.56% compilation time)
```

Simulation results

Plotstep 100, 300





The two simulation results are almost identical, but using H-Matrix is faster. Note that the speed gap increases as element size increases (figure 7).

References

Dieterich, J. H. (1979). Modeling of rock friction: 1. Experimental results and constitutive equations. *J. Geophys. Res.*, *84*(9), 2161–2168.

Hackbusch, W. (1999). A sparse matrix arithmetic based on-matrices. Part I: Introduction to-matrices. *Computing*, *62*(2), 89–108

Im, K., & Avouac, J.-P. (2021). Tectonic tremor as friction-induced inertial vibration. *Earth and Planetary Science Letters*, *576*, 117238. https://doi.org/10.1016/j.epsl.2021.117238

Im, K., Avouac, J.-P. (2024). Quake-DFN: A Software for Simulating Sequences of Induced Earthquakes in a Discrete Fault Network. https://doi.org/10.1785/0120230299

Jiang, J., Erickson, B. A., Lambert, V. R., Ampuero, J., Ando, R., Barbot, S. D., Cattania, C., Zilio, L. D., Duan, B., & Dunham, E. M. (2022). Community-driven code comparisons for three-dimensional dynamic modeling of sequences of earthquakes and aseismic slip. *Journal of Geophysical Research: Solid Earth*, *127*(3), e2021JB023519.

Marone, C. (1998). Laboratory-Derived Friction Laws and Their Application To Seismic Faulting. *Annual Review of Earth and Planetary Sciences*, *26*(1), 643–696. https://doi.org/10.1146/annurev.earth.26.1.643

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

Rice, J. R. (1993). Spatio-temporal complexity of slip on a fault. *J. Geophys. Res.*, *98*(B6), 9885. https://doi.org/10.1029/93JB00191

Rudnicki, J. W. (1986). Fluid mass sources and point forces in linear elastic diffusive solids. *Mechanics of Materials*, *5*(4), 383–393.

Segall, P., & Lu, S. (2015). Injection-induced seismicity: Poroelastic and earthquake nucleation effects. *Journal of Geophysical Research: Solid Earth*, *120*(7), 5082–5103.