# tandem

Carsten Uphoff

Sep 15, 2023

# CONTENTS

Tandem is a scientific software for SEAS ( Sequences of Earthquakes and Aseismic Slip) modelling and for solving Poisson and linear elasticity problems. It implements the Symmetric Interior Penalty Galerkin (SIPG) method using unstructured simplicial meshes (triangle meshes in 2D, tetrahedral meshes in 3D).

For a recent overview see our vEGU21 display.

CHAPTER

ONE

# LICENSE

tandem is made available under the BSD 3-Clause License.

CHAPTER

TWO

# TABLE OF CONTENTS

## 2.1 Getting started

### 2.1.1 Quick start with Docker

**Attention:** Please install Docker to follow the quick start procedure. You may use a drop-in replacement such as podman, too.

#### Step 1: Enter the Docker container

Open a terminal and pull the tandem-env image:

```
$ docker pull uphoffc/tandem-env
```

Activate the Docker container with

```
$ docker run -it -v $(pwd):/home -u $(id -u):$(id -g) uphoffc/tandem-env
```

You should now see something like

```
I have no name!@<random string>:/home$
```

**Note:** In the docker run command, the option `-it` opens an interactive terminal, `-v $(pwd):/home` maps your current working directory to the home folder inside the container, and `-u $(id -u):$(id -g)` fixes the file permissions for files you create inside the container.

#### Step 2: Compile tandem

Inside the Docker container, clone the tandem repository and load submodules:

```
$ git clone https://github.com/TEAR-ERC/tandem.git
$ cd tandem/
$ git submodule update --init
```

Tandem uses CMake, which is contained in the Docker image. We do not run CMake from the tandem directory but create a build directory to not pollute our workspace. From the build directory we run CMake followed by make:

```
$ mkdir build
$ cd build
$ cmake .. -DPOLYNOMIAL_DEGREE=6
$ make -j
```

Note that we specified the polynomial degree of the finite element spaces with the `POLYNOMIAL_DEGREE` variable. Another important variable is `DOMAIN_DIMENSION`, which should be set to 2 or 3, depending on whether you want to run 2D or 3D models. You can also use `ccmake ..` to enter a GUI which shows all available compilation variables.

#### Step 3: Run tests

To check that everything works, run

```
$ make test
```

from the build folder. At the end of the tests, you should see

```
100% tests passed, 0 tests failed out of 21
```

You are now set to run the *examples*.

### 2.1.2 Installation

Tandem and its dependencies can be installed automatically with Spack, or manually.

#### Spack installation

Spack is an HPC software package manager. It automates the process of installing, upgrading, configuring, and removing computer programs. In particular, the spack package `tandem` allows automatically installing tandem and all its dependencies, and creating environment modules. First, install spack with, e.g.

```
cd $HOME
git clone --depth 1 https://github.com/spack/spack.git
cd spack
echo "export SPACK_ROOT=$PWD" >> $HOME/.bashrc
echo "export PATH=\$SPACK_ROOT/bin:\$PATH" >> $HOME/.bashrc
```

Then install tandem with:

```
spack install tandem@main polynomial_degree=3 domain_dimension=2
```

tandem can then be loaded with `spack load tandem`. Alternatively, we might prefer loading tandem from environment modules. We therefore now detail the procedure to generate such module(s). You may want to update `~/.spack/modules.yaml`, to specify the path where the module file(s) should be installed (if e.g. if want to share your installation with other users and they cannot access your $HOME), and to generate module files with more readable names:

```
modules:
  default:
    roots:
      tcl: your_custom_path_2_modules
  default:
```

(continues on next page)

(continued from previous page)

```
    tcl:
      all:
        suffixes:
          domain_dimension=2: 'd2'
          domain_dimension=3: 'd3'
          polynomial_degree=1: 'p1'
          polynomial_degree=2: 'p2'
          polynomial_degree=3: 'p3'
          polynomial_degree=4: 'p4'
          polynomial_degree=5: 'p5'
          polynomial_degree=6: 'p6'
```

Note that a custom install directory for spack packages can also be set, by changing `~/.spack/config.yaml`:

```
config:
  install_tree: path_2_packages
```

We can then generate a tandem module file with:

```
spack module tcl refresh tandem
```

to access the module at start up, add to your `~/.bashrc`:

```
module use your_custom_path_2_modules/your_spack_arch_string
```

e.g.:

```
module use $HOME/spack/modules/x86_avx512/linux-sles15-skylake_avx512/
```

### SuperMUC-NG installation

First, have a look at this page to best configure git on SuperMUC-NG.

The software stack on SuperMUC-NG has been installed with spack. Yet, spack on SuperMUC-NG is not recent enough to natively know how to compile tandem. The recipe for compiling spack should then be added from a repository:

```
# load spack
module load user_spack
# clone seissol-spack-aid and add the repository
git clone --branch supermuc_NG https://github.com/SeisSol/seissol-spack-aid.git
cd seissol-spack-aid
spack repo add ./spack
```

tandem can be then installed, e.g. with:

```
spack install tandem@main polynomial_degree=3 domain_dimension=2 target=skylake_avx512
```

The procedure to create an environment module is the same as detailed above.

### Manual installation

The following dependencies are likely available via your package manager:

- A recent C++-17 capable compiler (we recommend GCC  8.0 or clang  8)
- MPI (e.g. OpenMPI)
- zlib ( 1.2)
- Eigen ( 3.3)
- Python ( 3.5) with NumPy ( 1.12.0)
- Lua ( 5.3)
- CMake ( 3.18)

The following dependencies likely need to be installed manually:

- METIS ( 5.1) and ParMETIS ( 4.0)
- PETSc ( 3.13)
- (Optional) libxsmm (= 1.16.1)

### Dependencies via package manager

The following instructions are valid for Debian buster and might also work for Ubuntu. Consult your package manager's documentation for other operating systems.

```
# apt-get install -y gcc g++ gfortran libgomp1 \
      make cmake libopenblas-dev libopenblas-base \
      libopenmpi-dev libopenmpi3 git libeigen3-dev \
      python3 python3-distutils python3-numpy \
      liblua5.3-0 liblua5.3-dev zlib1g zlib1g-dev
```

### Install METIS and ParMETIS

```
# wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/metis-5.1.0.tar.gz
# wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/parmetis-4.0.3.tar.gz
# tar -xvf metis-5.1.0.tar.gz
# tar -xvf parmetis-4.0.3.tar.gz
# cd metis-5.1.0
# make config && make && make install
# cd ../parmetis-4.0.3
# make config && make && make install
# cd ..
```

### Install PETSc

```
# wget http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-lite-3.14.6.tar.gz
# tar -xvf petsc-lite-3.14.6.tar.gz
# cd petsc-3.14.6
# ./configure --with-fortran-bindings=0 --with-debugging=0 \
    --with-memalign=32 --with-64-bit-indices \
    CC=mpicc CXX=mpicxx FC=mpif90 --prefix=/usr/local/ \
    --download-mumps --download-scalapack \
    COPTFLAGS="-g -O3" CXXOPTFLAGS="-g -O3"
# make PETSC_DIR=`pwd` PETSC_ARCH=arch-linux-c-opt -j
# make PETSC_DIR=`pwd` PETSC_ARCH=arch-linux-c-opt install
# cd ..
```

### (Optional) Install libxsmm

```
# wget https://github.com/hfp/libxsmm/archive/refs/tags/1.16.1.tar.gz
# tar -xvf 1.16.1.tar.gz
# cd libxsmm-1.16.1
# make -j generator
# cp bin/libxsmm_gemm_generator /usr/local/bin/
# cd ..
```

### Compile tandem

```
$ git clone https://github.com/TEAR-ERC/tandem.git
$ cd tandem/
$ git submodule update --init
$ mkdir build
$ cd build
$ cmake .. -DPOLYNOMIAL_DEGREE=6
$ make -j
```

If you installed libraries to a folder different from `/usr` or `/usr/local` and CMake cannot find your libraries, try to set the CMAKE_PREFIX_PATH, e.g.

```
$ cmake .. -DPOLYNOMIAL_DEGREE=6 -DCMAKE_PREFIX_PATH=/path/to/your/libs
```

If you require multiple paths to CMake, the syntax is as follows `-DCMAKE_PREFIX_PATH="/usr/local/path_1;/usr/local/path_2"`

### 2.1.3 Examples

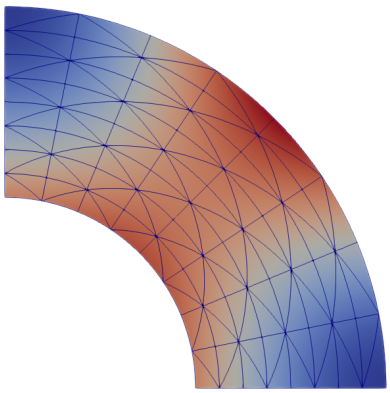
In the tandem repository you find many examples. The folders `poisson` and `elasticity` contain test problems for the Poisson and Elasticity solvers. In the folder `tandem` you find SEAS models. The folder `options` contains PETSc solver configurations.

#### Elasticity problem

From the build directory, run

```
$ ./app/static ../examples/elasticity/2d/cosine.toml --output cosine
```

After a successful run, there should be the file `cosine.pvtu` in your build directory. The pvtu-file can be visualized with ParaView. Cf. the following visualization of the cosine example:



Along with the pvtu-file, you also get console output, e.g.

```
Solver warmup: 0.120276 s
Solve: 0.310153 s
Residual norm: 1.20671e-11
Iterations: 127
L2 error: 2.88579e-09
```

We see that we need quite a number of iterations to solve this problem. Let's use a LU-decomposition instead of an iterative solver:

```
$ ./app/static ../examples/elasticity/2d/cosine.toml --output cosine \
      --petsc -options_file ../examples/options/lu_mumps.cfg
```

The console output should be similar to the following:

```
Solver warmup: 0.194319 s
Solve: 0.00834613 s
Residual norm: 0
Iterations: 1
L2 error: 2.88579e-09
```

We see that the warm-up time increased but the solve time decreased a lot. Moreover, we only need 1 "iteration" as we used a direct solver.

Now open up the parameter file, `cosine.toml`:

```
resolution = 0.125

[elasticity]
lib = "cosine.lua"
...
```

The cosine example uses a generated mesh, therefore we can adjust the mesh resolution in the parameter with the `resolution` parameter. You could now edit the parameter file to adjust the resolution. Alternatively, you can override top-level parameters from the command line:

```
$ ./app/static ../examples/elasticity/2d/cosine.toml --output cosine \
      --resolution 0.0625 --petsc -options_file ../examples/options/lu_mumps.cfg
```

You should now see

```
Solver warmup: 0.760638 s
Solve: 0.0306711 s
Residual norm: 0
Iterations: 1
L2 error: 2.42624e-11
```

The solve and warm-up time increased considerably, but also the error is lower. Indeed, comparing the errors with

$$\log_2\left(\frac{2.88579\cdot 10^{-9}}{2.42624\cdot 10^{-11}}\right)\approx 6.9$$

shows that the empirical convergence order is close to the theoretical convergence order 7. (Assuming that you compiled tandem with `POLYNOMIAL_DEGREE=6`.)

### SEAS problem

**Attention:** Please install Gmsh for this section.

On your local machine, go to the folder `examples/tandem/2d` and run

```
$ gmsh -2 bp1_sym.geo -setnumber Lf 0.5
```

You have now created a mesh with an on-fault resolution of 0.5 km. Now go to your build folder (inside the Docker container, if you used Docker) and run:

```
$ ./app/tandem ../examples/tandem/2d/bp1_sym.toml \
      --petsc -options_file ../examples/options/lu_mumps.cfg \
      -options_file ../examples/options/rk45.cfg -ts_monitor
```

In comparison to the Elasticity example, we added the `rk45.cfg` options file which selects an adaptive Runge-Kutta time-stepping scheme. The option `-ts_monitor` enables monitoring of time and time-step size.

Time to fetch a coffee, as this is going to take a while. In order to speed things up, add `--mode QDGreen`:

```
$ ./app/tandem ../examples/tandem/2d/bp1_sym.toml --mode QDGreen \
      --petsc -options_file ../examples/options/lu_mumps.cfg \
      -options_file ../examples/options/rk45.cfg -ts_monitor
```

Tandem now spends some time in a pre-computation step, but the time-stepping itself will be much faster.

The code logs the slip rate and other quantities at certain points and saves those in the `fltst_*` files. You can view these files using the viewrec tool from the SeisSol project – even when tandem is still running.

Welcome to tandem!

Tandem may be used in 2D and 3D and supports low-order as well as high-order spaces. The specific configuration is selected at compile time, which is why tandem needs to compiled from source. If you have Docker installed, you might want to try the *quick start* procedure. Here, all required dependencies are already contained in a Docker image. Otherwise, follow the *regular installation* procedure.

Once tandem is installed, try to run the *example problems*.

## 2.2 My first model

In this first tutorial, we are going to build SEAS model for normal and reverse faulting on a planar fault.

We create the CAD model of the fault and generate the mesh in the *first step*. The material and friction parameters are set in a *Lua-script*. Simulation parameters are set in the *parameter file*.

**Tip:** If you dislike copying text snippets, you can copy the complete tutorial files from the example folder.

### 2.2.1 Mesh creation with Gmsh

Gmsh allows CAD modelling as well as mesh generation. It comes with its own scripting language that we use to build the geometry.

Create a file called `tutorial.geo` and open it with your favourite text editor.

We first define a few parameters. These parameters can be either set from the Gmsh GUI or from the command line using `-setnumber`.

```
DefineConstant[ res = {20.0, Min 0, Max 10, Name "Domain resolution" } ];
DefineConstant[ res_f = {0.25, Min 0, Max 10, Name "Fault resolution" } ];
DefineConstant[ dip = {60, Min 0, Max 90, Name "Dipping angle" } ];

SetFactory("OpenCASCADE");
```

The last line enables the OpenCASCADE CAD kernel that we use to create our geometry. The dip angle is converted from degrees to radians and a few constants to define the bounding box are set:

```
dip_rad = dip * Pi / 180.0;
W = 40.0;
H = 100.0;
dX = 100.0;
X0 = -dX;
X1 = H * Cos(dip_rad) / Sin(dip_rad) + dX;
Y0 = -H;
```

We create our domain $[X_0, X_1] \times [Y_0, 0]$:

```
box = news; Rectangle(box) = {X0, Y0, 0.0, X1-X0, -Y0};
```

**See also:**

The domain dimensions are given in kilometres. Thus, we are going to *scale the Lamé parameters* accordingly.

We then insert a fault. As we are going to vary the $a$-parameter from 0 km to 40 km depth, we split the fault to later set a higher resolution in the upper part of the fault.

```
p1 = newp; Point(p1) = {0.0, 0.0, 0.0, res_f};
p2 = newp; Point(p2) = {W * Cos(dip_rad) / Sin(dip_rad), -W, 0.0, res_f};
p3 = newp; Point(p3) = {H * Cos(dip_rad) / Sin(dip_rad), -H, 0.0, res_f};

fault1 = newl; Line(fault1) = {p1,p2};
fault2 = newl; Line(fault2) = {p2,p3};
```

The mesh generator is currently unaware of the fault. Hence, we intersect the fault with the domain:

```
v[] = BooleanFragments{ Surface{box}; Delete; }{ Line{fault1, fault2}; Delete; };
```

The Line-IDs have changed in the above boolean operation. We recover the individual lines by searching them inside bounding boxes:

```
eps = 1e-3;
top[] = Curve In BoundingBox{X0-eps, -eps, -eps, X1+eps, eps, eps};
bottom[] = Curve In BoundingBox{X0-eps, Y0-eps, -eps, X1+eps, Y0+eps, eps};
left[] = Curve In BoundingBox{X0-eps, Y0-eps, -eps, X0+eps, eps, eps};
right[] = Curve In BoundingBox{X1-eps, Y0-eps, -eps, X1+eps, eps, eps};
```

Finally, we set resolution parameters, assign boundary conditions, and set the mesh format to version 2.2.

```
MeshSize{ PointsOf{Surface{:};} } = res;
MeshSize{ PointsOf{Line{fault1};} } = res_f;

Physical Curve(1) = {bottom(),top()};
Physical Curve(3) = {fault1,fault2};
Physical Curve(5) = {left[],right[]};
Physical Surface(1) = {v[]};

Mesh.MshFileVersion = 2.2;
```

The argument of `Physical Curve` must be set to 1, 3, or 5. A 1 stands for free surface, 3 for fault, and 5 for Dirichlet boundary condition.

We can now generate the mesh and adjust the resolution and dip angle from the command line. E.g.

```
$ gmsh -2 tutorial.geo -setnumber res_f 0.5
```

### 2.2.2 Lua scripting

**Warning:** This page is under construction.

```
local Tutorial = {}
Tutorial.__index = Tutorial

-- constant parameters
Tutorial.b = 0.010
Tutorial.V0 = 1.0e-6
Tutorial.f0 = 0.6

-- internal parameters
Tutorial.rho = 2.670
Tutorial.cs = 3.464
Tutorial.nu = 0.25

function Tutorial.new(params)
    local self = setmetatable({}, Tutorial)
    self.dip = params.dip
    self.Vp = params.Vp
    return self
end

function Tutorial:boundary(x, y, t)
    local Vh = self.Vp * t / 2.0
    if x < 0 then
        Vh = -Vh
    end
    return Vh, 0.0
end

function Tutorial:mu(x, y)
    return self.cs^2 * self.rho
end

function Tutorial:lam(x, y)
    return 2 * self.nu * self:mu(x,y) / (1 - 2 * self.nu)
end

function Tutorial:eta(x, y)
    return self.cs * self.rho / 2.0
end

function Tutorial:L(x, y)
    return 0.008
end

function Tutorial:Sinit(x, y)
    return 0.0
end
```

(continues on next page)

(continued from previous page)

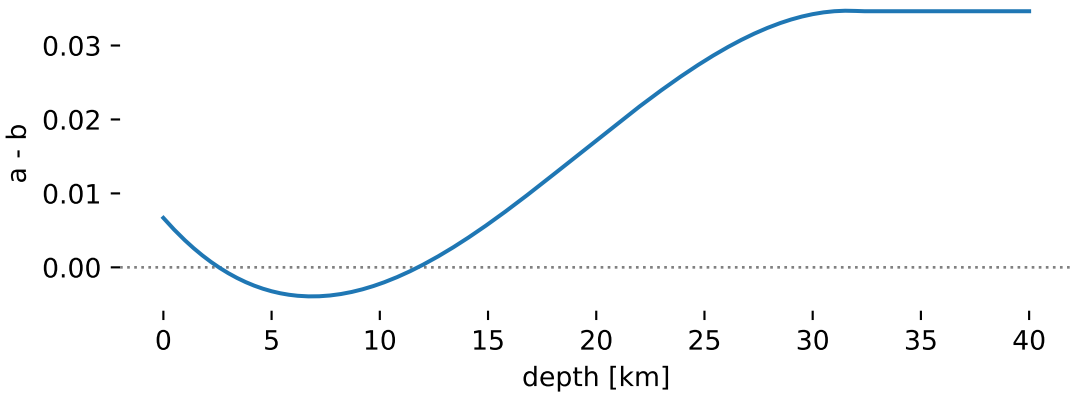
```
function Tutorial:Vinit(x, y)
    return self.Vp * math.cos(self.dip * math.pi / 180.0)
end

function Tutorial:a(x, y)
    local d = math.min(math.abs(y), 32.2)
    return self.b + -5.1115922342571294e-6*d^3 + 0.00029499040079464792*d^2 - 0.
↪00333076172038043 3*d + 0.006685594326305008
end

function Tutorial:sn_pre(x, y)
    return 50.0
end

function Tutorial:tau_pre(x, y)
    local Vi = self:Vinit(x, y)
    local sn = self:sn_pre(x, y)
    local amax = self:a(0, -40)
    local e = math.exp((self.f0 + self.b * math.log(self.V0 / math.abs(Vi))) / amax)
    return -(sn * amax * math.asinh((Vi / (2.0 * self.V0)) * e) + self:eta(x, y) * Vi)
end

normal = Tutorial.new{dip=60, Vp=1e-9}
reverse = Tutorial.new{dip=60, Vp=-1e-9}
```




### 2.2.3 Parameter file

**Warning:** This page is under construction.

```
final_time = 47304000000
mesh_file = "tutorial.msh"
lib = "tutorial.lua"
```

(continues on next page)

(continued from previous page)

```
scenario = "normal"
type = "elasticity"
ref_normal = [1, 0]
boundary_linear = true

[fault_probe_output]
prefix = "fltst_"
probes = [
    { name = "dp000", x = [0.0, -0.0] },
    { name = "dp025", x = [1.2500000000000002, -2.1650635094610964] },
    { name = "dp050", x = [2.5000000000000004, -4.330127018922193] },
    { name = "dp075", x = [3.7500000000000001, -6.495190528383289] },
    { name = "dp100", x = [5.0000000000000001, -8.660254037844386] },
    { name = "dp125", x = [6.2500000000000002, -10.825317547305483] },
    { name = "dp150", x = [7.5000000000000002, -12.990381056766578] },
    { name = "dp175", x = [8.7500000000000002, -15.155444566227676] },
    { name = "dp200", x = [10.000000000000002, -17.32050807568877] },
    { name = "dp250", x = [12.500000000000004, -21.650635094610966] },
    { name = "dp300", x = [15.000000000000004, -25.980762113533157] },
    { name = "dp350", x = [17.500000000000004, -30.31088913245535] },
]

[fault_output]
prefix = "output/fault"
rtol = 0.1

[domain_output]
prefix = "output/domain"
rtol = 0.1
```

### 2.2.4 Run model

**Warning:** This page is under construction.

Petsc options

```
-ksp_type preonly
-pc_type lu
-pc_factor_mat_solver_type mumps

-ts_type rk
-ts_rk_type 5dp
-ts_rtol 1e-8
-ts_atol 1e-50
-ts_adapt_wnormtype infinity

-ts_dt 0.0001
-ts_monitor
```

```
$ ./tandem tutorial.toml --discrete_green yes --petsc -options_file solver.cfg
```

### 2.2.5 Post processing

**Warning:** This page is under construction.

## 2.3 Reference

### 2.3.1 Equation scaling

When working with SI units in SEAS models numbers might get very large. Rescaling the equations might be advantageous to avoid large round-off errors in finite precision. In this section, we show how to properly scale the elasticity equations.

The linear elasticity equations in first order form are given by

$$\begin{aligned}\sigma_{ij} &= \lambda\delta_{ij}\frac{\partial u_k}{\partial x_k} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\\ -\frac{\partial \sigma_{ij}}{\partial x_j} &= f_i\end{aligned}$$

We define scaled quantities

$$\bar{x}_i = \frac{x_i}{L}, \quad \bar{u}_i = \frac{u_i}{u_c}, \quad \bar{\sigma}_{ij} = \frac{\sigma_{ij}}{\sigma_c},$$

where $L, u_c, \sigma_c$ are *scaling constants*. Inserting these into the linear elasticity equations gives

$$\begin{aligned}\sigma_c\bar{\sigma}_{ij} &= L^{-1}u_c\lambda\delta_{ij}\frac{\partial \bar{u}_k}{\partial \bar{x}_k} + L^{-1}u_c\mu\left(\frac{\partial \bar{u}_i}{\partial \bar{x}_j} + \frac{\partial \bar{u}_j}{\partial \bar{x}_i}\right)\\ -L^{-1}\sigma_c\frac{\partial \bar{\sigma}_{ij}}{\partial \bar{x}_j} &= f_i\end{aligned}$$

Multiplying the first equation with $\sigma_c^{-1}$, multiplying the second equation with $L\sigma_c^{-1}$, and defining

$$\bar{\lambda} = \sigma_c^{-1}u_cL^{-1}\lambda, \quad \bar{\mu} = \sigma_c^{-1}u_cL^{-1}\mu, \quad \bar{f}_i = L\sigma_c^{-1}f_i$$

leads to

$$\begin{aligned}\bar{\sigma}_{ij} &= \bar{\lambda}\delta_{ij}\frac{\partial \bar{u}_k}{\partial \bar{x}_k} + \bar{\mu}\left(\frac{\partial \bar{u}_i}{\partial \bar{x}_j} + \frac{\partial \bar{u}_j}{\partial \bar{x}_i}\right)\\ -\frac{\partial \bar{\sigma}_{ij}}{\partial \bar{x}_j} &= \bar{f}_i\end{aligned}$$

That is, we recovered the original equations and we only need to scale the mesh and the parameters.

#### Example

We change units with the scaling constants

$$L = 10^3, \quad u_c = 1, \quad \sigma_c = 10^6$$

In the rescaled equations, the spatial dimension of the mesh is [km], velocities are in [m/s], and stresses are in [MPa]. Parameters and source terms are scaled with

$$\bar{\lambda} = 10^{-9}\lambda, \quad \bar{\mu} = 10^{-9}\mu, \quad \bar{f}_i = 10^{-3} f_i$$

i.e. the Lamé parameters are given in [GPa] and force in [$10^{-3}$ N/m$^{-3}$].

### 2.3.2 Sign conventions

Slip is defined as

$$\boldsymbol{S} = \boldsymbol{u}^- - \boldsymbol{u}^+$$

Let the orthogonal basis $\boldsymbol{n}, \boldsymbol{d}, \boldsymbol{s}$ be given, where normal $\boldsymbol{n}$ points from the "-"-side to the "+"-side, $\boldsymbol{d}$ is the dip direction, and $\boldsymbol{s}$ is the strike direction. The slip-rate vector is defined as

$$\boldsymbol{V} = [\dot{\boldsymbol{S}} \cdot \boldsymbol{d}, \dot{\boldsymbol{S}} \cdot \boldsymbol{s}],$$

the shear traction vector is

$$\boldsymbol{\tau} = [\boldsymbol{d} \cdot \sigma\boldsymbol{n}, \boldsymbol{s} \cdot \sigma\boldsymbol{n}],$$

and the normal stress is given by

$$\sigma_n = \boldsymbol{n} \cdot \sigma\boldsymbol{n}.$$

Note that in 2D we drop the second component of the slip-rate and shear traction vector.

The friction law is given by

$$-(\boldsymbol{\tau}^0 + \boldsymbol{\tau}) = (\sigma_n^0 - \sigma_n) f(|\boldsymbol{V}|, \psi) \frac{\boldsymbol{V}}{|\boldsymbol{V}|} + \eta \boldsymbol{V},$$

where $\boldsymbol{\tau}^0$ and $\sigma_n^0$ are pre-stresses. We take $\sigma_n^0$ to be positive in compression, thus the sign is different to $\sigma_n$.

### 2.3.3 Fault basis

Slip and slip-rate are defined with respect to a local fault basis. In this document the conventions for the fault basis are introduced. The direction of movement is defined in terms of the hanging wall and the foot wall:

"The foot wall (hanging wall) is defined as the block below (above) the fault plane. (...) the hanging wall moves up with respect to the foot wall and the fault is known as **reverse**. (...) the opposite happens and the fault is said to be **normal**." [J. Pujol, Elastic Wave Propagation and Generation in Seismology]

The sign of the fault normal is chosen such that

$$n \cdot n_{\text{ref}} > 0.$$

We define that the fault normal points from the foot wall to the hanging wall. In this way the reference normal $n_{\text{ref}}$ selects the foot and the hanging wall.

The first component of the slip or slip-rate vector is defined w.r.t. to the normal direction of the fault. Due to the no-opening condition the first component is zero.

The third component of the slip or slip-rate vector is defined w.r.t. to the strike direction. The latter is defined such that a hypothetical observer standing on the fault looking in strike direction sees the hanging wall on his right. Thus, the strike direction is

$$s := u \times n,$$

where $u$ is the direction of "up", given in the configuration file. E.g. using the **enu** convention, up would be the vector $u = (0, 0, 1)$.

The second component of the slip or slip-rate vector is defined w.r.t. to the dip direction, which we define to point "down". That is, the dip direction is

$$d := s \times n$$

### Left-lateral, right-lateral, normal, reverse

The slip vector is given by $u = [u_n]n + [u_d]d + [u_s]s$, where the square bracket operator for a scalar field $q$ is defined as

$$[q] := q^- - q^+ = \lim_{\epsilon \to 0} q(x - \epsilon n) - q(x + \epsilon n)$$

Recall that the normal points from the foot wall to the hanging wall. Thus, if $[u_d] > 0$ we have a **reverse** fault. Conversely, if $[u_d] < 0$ we have a **normal** fault.

For strike slip fault, i.e. $[u_s] \neq 0$, we have to distinguish two cases:

"In a left-lateral (right-lateral) fault, an observer on one of the walls will see the other wall moving to the left (right)." [J. Pujol, Elastic Wave Propagation and Generation in Seismology]

If $[u_s] > 0$ then we have a right-lateral fault and if $[u_s] < 0$ then we have a left-lateral fault.

### Special-case: Flat fault

Don't do that.