

User Guide

Quick test

Run a simple point source explosion test and plot a 2D slice of particle velocity::

```
cd scripts/example
python sim.py
python plot.py
```

Plotting requires Matplotlib.

Scripting with Python

Every script starts by importing the ``sord`` module. SORD parameters are specified as local variables that are passed to the ``sord.run()`` function by the ``locals()`` built-in Python function. A complete list of possible SORD parameters and default values are specified in `<parameters.py>`__.

The Python language gives a powerful way to construct SORD input. For example, it may be desirable to specify the total simulation time, and divide by ``dt`` to determine the number of time steps::

```
T = 100.0          # total time temporary variable
nt = int( T / dt ) # number of time steps
```

The ``T`` variable is not a valid SORD parameter; it is only used for temporary storage. Variables with single single character names or names ending with an underscore, are ignored by ``sord.run()`` , so may be safely used for temporary storage. More elaborate example scripts can be found in the ``scripts`` directory.

Running jobs

Each time ``sord.run()`` is called, a run directory is created at the location set by the ``rundir`` parameter (``tmp/`` in this case). The directory contains the executable and scripts to run the code, and will contain all of the generated output and metadata. From the run directory, start the job interactively with the ``run.sh`` script::

tmp/run.sh

Or, submit the job to the batch system with the ``queue.sh`` script ::

tmp/queue.sh

Output (and large input, such as the material model) is stored in flat binary binary files. Statistic, such as peak acceleration and peak velocity, are computed periodically during each run and stored in the ``stats/`` directory.

Field I/O

Multidimensional field arrays can be accessed for input and output through the ``fieldio`` list. The ``<fieldnames.py>`` file specifies the list of available field variables, which are categorized in four ways: (1) static vs. dynamic, (2) settable vs. output only, (3) node vs. cell registration, and (4) volume vs. fault surface. For example, density ``rho`` is a static, settable, cell, volume variable. Slip path length ``sl`` is a dynamic, output, node, fault variable. The ``fieldio`` list is order dependent with subsequent inputs overwriting previous inputs. So, for example, a field may be assigned to one value for the entire volume, followed by a different value for a sub-region of the volume.

All field I/O operations require slice indices ``[j,k,l,t]``, which

specify a
 four-dimensional sub-volume of the array in space and time. Array
 indexing
 starts at 1 for the first node, and 1.5 for the first cell. Negative
 indices
 count inward from end of the array, starting at -1 for the last node,
 and -1.5
 for the last cell. Indices can be either a single index, a range
```(start,`  
`end)```, or a strided range ```(start, end, step)```. Empty parentheses  
```()``` are  
 shorthand for a full range. Empty brackets ```[]``` are shorthand for
 the entire
 4D volume. Some examples of slice notation:::

```

[10, 20, 1, (1,-1)]      # Single node, full time history
[10.5, 20.5, 1.5, ()]    # Single cell, full time history
[2, (), (), (1,-1,10)]   # j=2 node surface, every 10th time step
[(), (), (), -1]         # Full 3D volume, last time step
[]                        # Entire 4D volume
  
```

Each member of the ``fieldio`` list contains a mode, a field name, and
 slice
 indices, followed by mode dependent parameters. The following I/O
 modes are
 available, where ``'f'`` is the field variable name (from the list
``<fieldnames.py>``), and ``[]`` are the slice indices:::

```

('=', 'f', [], val),          # Set f to value
('+', 'f', [], val),         # Add value to f
('=s', 'f', [], val),        # Set f to random numbers in
range (0, val)
('=f', 'f', [], val, tfunc, T), # Set f to time function with
period T, scaled by val
('=r', 'f', [], filename),   # Read from filename into f
('=R', 'f', [], filename),   # Read from filename into f
with extrapolation.
('=w', 'f', [], filename),   # Write f to filename
('=wi', 'f', [], filename),  # Write weighted average of f
to filename.
  
```

A letter ``'i'`` in the mode indicates sub-cell positioning via
 weighted
 averaging. In this case the spatial indices are single logical
 coordinates
 that may vary continuously over the range. The fractional part of the
 index
 determines the weights. For example, an index of 3.2 to the 1D
 variable f
 would specify the weighted average: $0.8 * f(3) + 0.2 * f(4)$.

Reading and writing to disk uses flat binary files where j is the fastest changing index, and t is the slowest changing index. Mode 'R' extrapolates any singleton dimensions to fill the entire array. This is useful for reading 1D or 2D models into 3D simulations, obviating the need to store (possibly very large) 3D material and mesh coordinate files.

All input modes may use '+' instead of '=' to add to, rather than replace, preexisting values. For a list of available time functions, see the ``time_function`` subroutine in `<src/util.f90>`_. The routine can be easily modified to add new time functions. Time functions can be offset in time with the ``tm0`` initial time parameter.

Boundary Conditions

Boundary conditions for the six faces of the model domain are specified by the parameters ``bc1`` (near-size, x, y, and z faces) and ``bc2`` (far-side, x, y, and x faces). The symmetry boundary conditions can be used to reduce computations for problems where they are applicable. These are not used for specifying internal slip boundaries. However, for problems with symmetry across a slip surface, the fault may be placed at the boundary and combined with an anti-mirror symmetry condition. The following BC types are supported:

Type 0: Vacuum free-surface. Stress is zero in cells outside the boundary.

Type 3: Rigid surface. Displacement is zero at the boundary.

Type 1: Mirror symmetry at the node. Normal displacement is zero at the boundary. Useful for a boundary corresponding to (a) the plane orthogonal to the two nodal planes of a double-couple point source, (b) the plane normal to the mode-III axis of a symmetric rupture, or (c) the zero-width axis of a 2D plane strain problem.

Type -1: Anti-mirror symmetry at the node. Tangential displacement is zero at the boundary. Useful for a boundary corresponding to (a) the nodal planes of a double-couple point source, (b) the plane normal to the mode-II axis of a symmetric rupture, or (c) the zero-width axis of a 2D antiplane strain problem.

Type 2: Mirror symmetry at the cell. Same as type 1, but centered on the cell.

Type -2: Anti-mirror symmetry at the cell. Same as type -1, but centered on the cell. Can additionally be used when the boundary corresponds to the slip surface of a symmetric rupture.

Type 10: Perfectly match layer (PML) absorbing boundary.

Example: a 3D problem with a free surface at Z=0, and PML absorbing boundaries on all other boundary faces::

```
nn = 50, 50, 50  
bc1 = 10, 10, 0  
bc2 = 10, 10, 10
```

Example: a 2D antiplane strain problem with PML absorbing boundaries. The number of nodes is 2 for the zero-width axis::

```
nn = 50, 50, 2  
bc1 = 10, 10, -1  
bc2 = 10, 10, -1
```

Defining the fault rupture surface

Fault rupture always follows a surface of the (possibly non-planar) logical mesh. The orientation of the fault plane is defined by the ``faultnormal`` parameter. This can be either 1, 2, or 3 corresponding to surfaces normal to the j, k, or l logical mesh directions. Any other value (typically 0) disables rupture altogether. The location of the rupture plane with in the mesh

is determined by the ``ihypo`` parameter, which has a dual purpose of also defining the nucleation point. So, the indices of the collocated fault double nodes are given by ``int(ihypo(faultnormal))``, and ``int(ihypo(faultnormal)) + 1``. For example, a 3D problem of dimensions $200.0 \times 200.0 \times 200.0$, with a fault plane located at $z = 100.0$, and double nodes at $l = (21, 22)$, may be set up as such::

```
dx = 5.0
faultnormal = 3
ihypo = 21, 21, 21.5
nn = 41, 41, 42
bc1 = 0, 0, 0
bc2 = 0, 0, 0
```

For problems with symmetry across the rupture surface (where mesh and material properties are mirror images), the symmetry may be exploited for computational savings by using an appropriate boundary condition and solving the elastic equations for only one side of the fault. In this case, the fault double nodes must lie at the model boundary, and the cell-centered anti-mirror symmetry condition used. For example, reducing the size of the previous example to put the rupture surface along the far z boundary::

```
nn = 41, 41, 22
bc2 = 0, 0, -2
```

Alternatively, put the rupture surface along the near z boundary::

```
ihypo = 21, 21, 1.5
nn = 41, 41, 22
bc1 = 0, 0, -2
bc2 = 0, 0, 0
```

Further symmetries may present. If our previous problem has slip only in the x direction, then we may also use node-centered mirror symmetry along the in-plane axis, and node-centered anti-mirror symmetry along the anti-plane axis, to reduce

computations eight-fold::

ihypo = 21, 21, 21.5
nn = 21, 21, 22
bc1 = 0, 0, 0
bc2 = -1, 1, -2