MDSBI: Multi-Dimensional Spectral Boundary Integral Code, Version 3.9.10

Eric M. Dunham
edunham@fas.harvard.edu

17 May 2007

1 Introduction

MDSBI is a Fortran 95 boundary integral code for solving elastodynamic problems, specifically those dealing with shear ruptures or frictional instabilities. The code handles both two- and three-dimensional problems, as well as the response of a single spatial Fourier mode. The code handles the case of identical materials on either side of the fault, as well as the more general bimaterial problem, including its poroelastic correction. Various friction laws include slip-weakening, rate-and-state (with flash heating), and thermal pressurization. The section of the code for flash heating and thermal pressurization was written jointly with Hiro Noda. The elastodynamic response is calculated by a convolution over space and time of the slip or slip velocity history with appropriate kernels. The spatial convolution is performed in the Fourier domain using FFTs. The code was developed based on ideas from several papers, in particular, P. H. Geubelle and J. R. Rice (A spectral method for three-dimensional elastodynamic fracture problems, Journal of the Mechanics and Physics of Solids, 43, 1791–1824, 1995), P. H. Geubelle (A numerical method for elastic and viscoelastic dynamic fracture problems in homogeneous and bimaterial systems, Computational Mechanics, 20, 20–25, 1997), M. S. Breitenfeld and P. H. Geubelle (Numerical analysis of dynamic debonding under 2D in-plane and 3D loading, International Journal of Fracture, 93, 13–38, 1998), and N. Lapusta, J. R. Rice, Y. Ben-Zion, and G. Zheng (Elastodynamic analysis for slow tectonic loading with spontaneous rupture episodes on faults with rate- and state-dependent friction, Journal

of Geophysical Research, 105, 23,765–23,789, 2000). These papers are an excellent place to start if you wish to read more about the method.

A description of the numerical method used to solve the coupled flash heating and thermal pressurization equations in is preparation by H. Noda, E. M. Dunham, and J. R. Rice (2007).

2 Geometries

To help explain the geometry of the problems that can be solved, see Figure 1. This shows a 2D planar interface (or fault) in a 3D elastic whole-space. For a 3D problem, the fields vary in both the x and y directions. If the fields vary in only one direction, say x', and are constant in the perpendicular direction, say y', then the problem is 2D and and is considerably simplified. 2D problems come in several flavors: mode I, mode II, mode III, and mixed mode. The mode I (or tensile) problem has opening in the z direction, but is currently only implemented in the bimaterial formulation. For the mode II geometry, $x' = x$ and all loading and slip occurs in the x direction. For the mode III geometry, $x' = y$ and all loading and slip occurs in the x direction. The shear mixed mode case occurs when $x' \neq x$ or y. In this general case, the loading and slip may have components in both the x and y directions. If the angle between x and x' is ϕ, as shown in Figure 1, then $\phi = 0$ is mode II, $\phi = \pi/2$ is mode III, and $0 < \phi < \pi/2$ is mixed mode. For programming convenience, the x' axis in 2D problems is simply designated as the x axis. Components of fields, such v_x and v_y, refer to the components in the actual x and y coordinates, not x' and y'. Wavenumber components (k_x and k_y) also use the actual x and y coordinates.

Another simplification occurs when slip occurs only in one direction, which is taken to be x in the program. In this case, the memory requirements drop by a factor of two.

3 Installation

1. Download the compressed and archived source code `mdsbi-v3.9.10.tgz` from
   `http://www.people.fas.harvard.edu/~edunham/codes.html`
   to a chosen installation directory (taken to be `INSTALL` in this text).

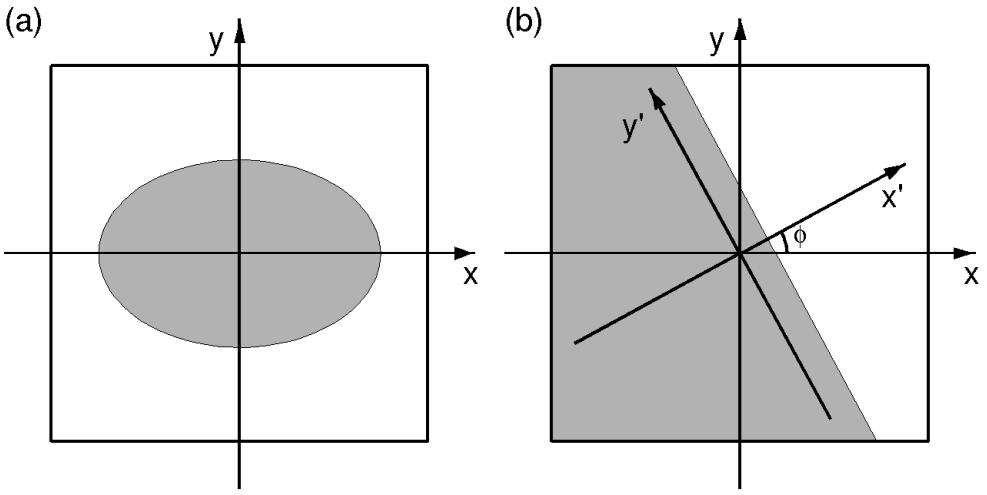

Figure 1: Model geometry, (a) 3D and (b) 2D, showing the fault plane. The gray area schematically shows the rupture. For the general 3D case (a), the rupture front has no specified symmetries. For the shear mixed mode 2D case (b), all fields vary only in the x' direction (i.e., they are independent of y'). The angle between the primed and unprimed coordinate systems is ϕ. Two special limits of the 2D case occur when slip occurs only in the x direction and $\phi = 0$ (mode II) or $\phi = \pi/2$ (mode III).

2. Unzip and extract:
   `tar xvzf mdsbi-v3.9.10.tgz`
   or:
   `gunzip mdsbi-v3.9.10.tgz` followed by `tar xvf mdsbi-v3.9.10.tar`
   This will create a directory `INSTALL/mdsbi/` that contains some files and subdirectories. The subdirectories are

   (a) `analysis`: routines to analyze the output data

   (b) `data`: where the output data will be written (you may wish to create a symbolic link of the same name if you want to store the data elsewhere)

   (c) `kernel`: convolution kernel tables

   (d) `problems`: files configuring the problems to be solved

   (e) `src`: source code

3. Download the kernel tables `kernels.tgz` from
   `http://www.people.fas.harvard.edu/~edunham/codes.html`
   to the kernel directory `INSTALL/mdsbi/kernel`.

4. Unzip and extract:
   `tar xvzf kernels.tgz`
   or:
   `gunzip kernels.tgz` followed by `tar xvf kernels.tar`
   This should produce two files containing the tabulated values of the convolution kernel; make sure they are in the `kernel` directory.

5. The code uses two libraries that are not included in this package. They are LAPACK, a package of linear algebra routines—see `www.netlib.org/lapack/` and FFTW, the Fastest Fourier Transform in the West—see `www.fftw.org`. The FFTW routines can perform FFTs on arrays of all sizes, not just powers of two.

   To set up these libraries, you need to make a few changes in the `INSTALL/mdsbi/src/` directory:

   (a) Install the LAPACK library from `www.netlib.org/lapack/`. This library is often pre-installed on Linux distributions, and is included in recent OS/X distributions (in the vecLib framework), so you might want to check if you already have it before installing.

You can actually get by with only a subset of the complete LAPACK library known as ATLAS (also available on netlib).

(b) Edit `makefile` to link to the appropriate LAPACK library (a few examples are given in the makefile).

(c) Set up the FFT routines. All FFT calls will be contained in `fft_routines.f90`. I've provided this file for both FFTW 2.x (`fft_routines_fftw2.f90`) and FFTW 3.x (`fft_routines_fftw3.f90`). I also have a version for FFTW 3.x that works with IBM's compiler `xlf95` by adding an extra underscore to subroutine names (`fft_routines_fftw3bgl.f90`). If you want to use another library, then you can use either of these files as a template. If you choose FFTW, then download and install the library from `www.fftw.org`, making sure that Fortran libraries are created. To use FFTW 3.x:

i. Make a symbolic link (or rename) the file:
`ln -s fft_routines_fftw3.f90 fft_routines.f90`

ii. Edit `makefile` to link to the FFTW3 library: `-lfftw3`.

and to use FFTW 2.x:

i. Make a symbolic link (or rename) the file:
`ln -s fft_routines_fftw2.f90 fft_routines.f90`

ii. Edit `makefile` to link to the FFTW2 libraries: `-lrfftw -lfftw`.

6. Set your compiler options in `makefile`. I have a few options in the makefile; basically uncomment your compiler or add it and do likewise with the flags. The program successfully compiles and runs with the Intel compiler `ifort`, the Sun compiler `f90`, IBM'S compiler `xlf95`, and the G95 compiler `g95` (available at `www.g95.org`). Please let me know if you experience any errors with yours.

   One item of note concerns the use of non-Fortran 95 intrinsic subroutines. I have made use of only one such subroutine: `flush`. The `flush` subroutine forces the program to automatically write all queued output to a specific file. This option is nice when you want to look at the contents of the output files (or the `status` file, discussed later, that tells you which time step you are on or any error messages), before the program is finished running. For certain compilers or optimization levels, this data is not output immediately when the program is directed

to write output. The `flush` subroutine causes this to occur. If you experience problems with this non-intrinsic, simply comment out the appropriate line in the source code. The specific subroutine is `flush_io` in `io.f90` in the `INSTALL/mdsbi/src/` directory.

7. Choose the precision with which you would like the calculations to be performed. The default installation of the FFTW library includes only double precision routines. For single precision routines, you will need to change the FFTW installation options. Within the program, edit the file `INSTALL/mdsbi/src/constants.f90`. Specifically set `pr = real4` for single precision or `pr = real8` for double precision. The default is double precision. You will also have to modify the appropriate routines in `INSTALL/mdsbi/src/fft_routines.f90`. Sometimes the solution degrades when using single precision; other times it seems fine. Use it at your own risk!

8. Compile by typing `make` in the source directory. This generates the executable `mdsbi` (or something similar, depending on what you've chosen in the makefile) in the main directory `INSTALL/mdsbi/`.

4 Configuration

Specific problems are found in the problems directory: `INSTALL/mdsbi/problems/`. All options are configured in a file named `problem_name.in`, where `problem_name` is the name of the problem. Input is primarily accomplished using Fortran namelists. Detailed instructions are included as comments in the problem `slipweak_identical_300m.in`, and several other examples are included (which are described under Included Problems). These additional files are not extensively commented. The most complete listing of all available options is within the source code itself, which is extensively commented. If you wish to create your own problem, copy one of the existing problem files, giving it a different name (e.g., `new_problem.in`).

5 Running

The program has the ability to solve several problems in succession (with each defined in a separate file in the `INSTALL/mdsbi/problems/` directory).

The code reads the file `INSTALL/mdsbi/input.in` to obtain the name of the problem list file and the name of the status file.

The problem list file contains a list of all the problems to be solved. By default, this is `INSTALL/mdsbi/problems/list.in`. However, you may use a different list located elsewhere. This is useful when you wish to run several instances of the program simultaneously. Your problem list file must have the extension `.in`.

Any messages from the program (such as the current problem, current time step, warnings, errors, etc.) are written to the status file, which will appear in the main directory. The default name is `status`, though you can change this as well in the file `INSTALL/mdsbi/input.in`.

1. Edit the problem list file, either `INSTALL/mdsbi/problems/list.in` or your chosen file, to include the names of all of the problems to be solved. Specify this list and the name of the status file to which messages will be written in `INSTALL/mdsbi/input.in`.

2. Switch to the main directory: `cd INSTALL/mdsbi/`. Run the executable by typing: `mdsbi` (or `mdsbi &` if you want it to run in the background). This will read the file `input.in` to determine which problem list and status file to use. Data files are written within the `INSTALL/mdsbi/data/` directory, which you can link to your favorite storage location.

3. Check the status of the running program by examining the status file (e.g., type `cat status`).

6 Analysis of Data

The data is output to the directory `INSTALL/mdsbi/data/`. There are two ways to output the fields, each of which is configured in the `problem_name.in` configuration file:

1. `mesh`: Output the fields on a mesh. The spatial mesh can be 2D, 1D (a line), or 0D (a point). Each field component is saved as a binary data file containing the field values on the mesh at specified time steps.

2. `front`: Output the time at which the rupture front (defined as when the value of a particular field exceeds a specified value) passes each point.

A Matlab script, `analyze.m`, is provided as an example in the analysis directory
`INSTALL/mdsbi/analysis/`. Cut and paste the commands into the Matlab window (run Matlab in that same directory) and several figures will be created. The comments in the file explain what you are seeing.

7 Included Problems

Several problems are included as examples:

1. `slipweak_identical_300m`: Sample rupture dynamics calculation with slip-weakening friction law for a fault between identical materials. This configuration file is heavily commented, and is hopefully self-documenting regarding different program options. Note that the 300 m grid spacing likely does not provide sufficient resolution of the solution; it is chosen simply for demonstration purposes so that the problem size will be small.

2. `slipweak_bimaterial_300m`: Same as `slipweak_identical_300m`, except for the bimaterial problem. Note the differences with the identical materials problem, namely that many of the parameters in the namelists are different. It is not necessary to list all parameters in each namelist; only the relevant ones are needed.

3. `pointforce_identical`: Point force on a frictionless interface between identical materials. This is a nice problem since the result can be compared to an analytical solution.

4. `heterogeneous`: Heterogeneous rupture dynamics run. This problem is based on `slipweak_identical_300m`, with the additional feature that data files are read in to introduce heterogeneity to the initial stress, slip-weakening distance, dynamic friction, and static friction fields. Examples of both ASCII and binary input data files are included. The files are created with a Matlab script `input_heterogeneous.m` that is also included in the `INSTALL/mdsbi/problems/` directory. The binary data files generated with this script are not necessarily portable if your machine has different endian. For this reason, I have not included these files in the distribution. Instead, you will need to run the Matlab script `input_heterogeneous.m` on your computer first.

5. `thermpres.in`: Thermal pressurization with flash heating. This models ruptures on faults governed by rate-and-state friction (in a form that includes strong weakening from flash heating at high slip velocities) and thermal pressurization. See H. Noda, E. M. Dunham, and J. R. Rice (in preparation, 2007) for further details, and to obtain parameters that illustrate the pulse and crack modes of rupture.

8 Miscellaneous Notes

8.1 Field Names and Sign Conventions

Let me comment on the names of the fields, which appear both in the code itself and in the `.in` problem files and the Matlab analysis scripts. I use similar names for the identical materials version and the bimaterial version, which might lead to confusion. For both the identical materials version and the bimaterial version, `Ux` and `uy` refer to slip in the x and y directions, `Vx` and `Vy` refer to slip velocity, `sx` and `sy` refer to stress, `sx0` and `sy0` refer to loads. For the identical materials version, `fx` and `fy` refer to stress transfer functionals. For the bimaterial version, the response of the two half-spaces is computed separately before boundary conditions are applied that couple them. So there are fields for each side, the upper (or "plus") side denoted by a "p", and the lower (or "minus") side denoted by an "m". Hence, the displacements are `uxm`, `uxp`, etc., the particle velocities are `vxm`, `vxp`, etc., and the stress transfer functionals are `fxm`, `fxp`, etc. The stresses are continuous so they remain the same as in the identical materials case.

The code keeps track of the stress components of traction, instead of the tractions direction (which have different signs on the different sides of the fault). Slip is right lateral. Compression is negative and tension is positive.

8.2 Units

The program performs no conversion of units, so you are free to use any self-consistent system (e.g., SI units). In several of the examples, units relevant to large-scale earthquakes are used. These are not SI units, instead I use a mixture of units for different fields that keep fields of order unity. (The code performs no rescaling of fields to prevent round-off error. This is your responsibility.) In these examples, stresses are in MPa, shear modulus is in

GPA; the extra factor of 10^3 is canceled by measuring displacements in m, but distances in km.

9 Program Details

The program is organized into a set of modules, typically one per file, containing the main data types and associated subroutines and functions. In alphabetical order, the modules are

1. `asperity`: Linked-list routines for adding heterogeneity to fields
2. `constants`: Constants and precision definitions
3. `convolution`: Convolution variables
4. `convolution_routines_bm`: Routines to perform convolution, bimaterial version
5. `convolution_routines_im`: Routines to perform convolution, identical materials version
6. `fault`: Fields on the fault
7. `fft_routines_fftw2`: FFT routines using FFTW2.x
8. `fft_routines_fftw3`: FFT routines using FFTW3.x
9. `fields`: Routines related to fields
10. `fourier`: Fourier-domain fields
11. `friction`: Friction laws
12. `friction_routines`: Routines for friction laws
13. `friction_solver`: Solver for friction laws
14. `front`: Output timing of rupture front
15. `history`: History of Fourier coefficients of slip or slip velocity
16. `init`: Routines to initialize problems

17. `integration`: Routines for integrating fields in time
18. `io`: Input/output routines
19. `kernel`: Convolution kernel
20. `linalg`: Linear algebra routines
21. `load`: Time-dependent loads
22. `main`: Main program routines
23. `mesh`: Output history of fields on a mesh
24. `model`: Basic model parameters
25. `mpi`: Dummy MPI library
26. `mpi_routines`: MPI routines
27. `problem`: Main problem data type
28. `problem_routines`: Routines to solve a single problem
29. `rates`: Routines to set rates (time-derivatives of fields)
30. `ratestate`: Rate-and-state friction
31. `rk`: Runge-Kutta coefficients
32. `slipweak`: Slip-weakening friction
33. `substep`: Routines for substepping
34. `thermpres`: Thermal pressurization
35. `timestep`: Routines for time steps
36. `utilities`: Assorted useful routines

Each of these modules comes in its own file, `module_name.f90`. The code has more features that I haven't discussed in this guide, and other modules. You are, of course, welcome to use these, but (as with everything else) I urge you to test them extensively before trusting the results.