FMST: Fast Marching Surface Tomography Package - Instructions

by Nick Rawlinson

Research School of Earth Sciences, Australian National University, Canberra ACT 0200

Contents

1 Introduction

This document describes how to use the Fortran 90 software package FMST for performing travel time tomography. The fast marching method (FMM) is used for the forward prediction step and a subspace inversion scheme is used for the inversion step. The method is iterative non-linear in that the inversion step assumes local linearity, but repeated application of FMM and subspace inversion allows the non-linear relationship between velocity and traveltime to be accounted for.

Traveltimes from point sources to receivers are computed in 2-D spherical shell coordinates (θ, ϕ). Software is also provided to generate 2-D models in the appropriate format for input into the FMM code. A variety of structures can be imposed on a constant velocity background model, including checkerboards, spikes and random (Gaussian) variations. These models are defined by a grid of nodes with bi-cubic B-spline interpolation, which produces a continuous, smooth and locally controlled velocity medium.

The inversion step allows both smoothing and damping regularization to be imposed in order to address the problem of solution non-uniqueness. A shell script has been written to couple the FMM and subspace codes into the complete iterative non-linear tomography routine.

Please be aware that the software has not been exhaustively tested, and probably contains a few bugs that are yet to be detected. Feel free to contact me for assistance if you believe that you have encountered a coding error. I have had a number of requests for the code, and have therefore put it on the web together with this document, which hopefully will help you negotiate some of its more idiosyncratic elements.

2 Unpacking and Compiling the Code

Download the UNIX tarred and gzipped file `fmst_v1.1.tar.gz` from the web page located at http://rses.anu.edu.au/∼nick/surftomo.html into an empty directory. To unpack the contents of the file, use either:

- tar -xvzf fmst_v1.1.tar.gz
- gunzip -c fmst_v1.1.tar.gz | tar xvof -

The contents of the tar archive will be placed in the current working directory, and the UNIX ls command should produce something like:

```
bin/    compileall*    docs/    example1/    example2/    source/
```

The next step is to begin the compilation procedure.

All of the F90 source files are contained in the subdirectory source/, and must be compiled with a Fortran 90 compiler. It is strongly recommended that you use the same compiler to create each separate executable, as data is exchanged between several programs using binary files. Rather than compile all the programs separately, the simple shell script compileall, located in the root directory of the distribution, can be used to automatically compile the entire package and place the executables in the directory bin/. If you wish to use this script, it is **essential** that you do not move it from its current relative location. Before executing compileall, make sure that you do the following:

- edit line 22 of compileall so that the code is compiled using your favourite Fortran 90 compiler.

Note that every effort has been made to use standard Fortran 90, but it is always difficult to know how different compilers will behave. The software package has been tested on the following platforms:

- Linux PC with AMD Opteron processor running a 64 bit operating system. Fortran 90 compilers tested include g95, ifort (Intel fortran compiler), Pathscale, NAG and Portland.
- Linux PC with Intel P4 processor running a 32 bit operating system. Fortran 90 compilers tested include g95 and ifort (Intel fortran compiler).
- Sun Blade 100 running Solaris. Fortran 90 compilers tested include frt (Fujitsu Fortran 90 Compiler) and f90 (Sun Fortran 90 compiler).

My tests indicate that the code runs much faster (4-5 times) on AMD or Intel based platforms compared to the Sun Blade. If you use an Intel platform, I strongly recommend using the Intel compiler as it is much faster than others I've tested. On a 64 bit AMD Opteron platform, Pathscale is about 30% faster than the best of the rest (which is actually the Intel compiler in 32 bit mode). [While I have not tested the 64 bit version of the Intel compiler, I believe that its performance is on par with the Pathscale compiler]. The g95 compiler is still in its beta stage, but my tests indicate that the latest version is quite competitive.

When you run compileall, successful compilation should write the following information to the screen:

```
Compilation of fm2dss complete

Compilation of grid2dss complete

Compilation of misfitss complete

Compilation of residualss complete

Compilation of resplotss complete

Compilation of subinvss complete

Compilation of synthtss complete

Compilation of tslicess complete

Compilation complete
```

The compilation time on a 64 bit AMD platform with a 1.6 GHz Opteron processor using ifort was about 3 seconds. All of the 8 executables will be placed in the subdirectory bin/. Note that the distribution already has an executable in this subdirectory prior to the compilation process. It is called ttomoss, and is actually a shell script, so does not require compilation.

All of the executables require an input parameter file, which by default has the same name as the executable, but ends with .in For example, fm2dss has an input parameter file called fm2dss.in. Example input parameter files can be found in a subdirectory of source/ called inputfiles/.

In order to run the executables from any directory, you will need to make sure that the path to the bin/ directory is correctly set in your startup shell script. Alternatively, you could copy the executables to a directory that you know is already specified in the path list. In the former case, if you use a .cshrc file, then you could add a line like:

- set path = ($path /directories/bin)

after the path is initially set. In the above example, directories refers to the directory structure above the bin/ directory (e.g. ~/tomography/fmst).

If you encounter any bugs, or have trouble getting the code to compile, then please send me an email at **nick@rses.anu.edu.au**.

3 Program Descriptions

A very brief description of the function of each program contained in bin/ is given below. For more detail, refer to the sections that follow.

- fm2dss: Solves the forward problem of traveltime prediction by applying the so-called Fast Marching Method (FMM), a grid-based eikonal solver. fm2dss can also output ray paths and Fréchet derivatives, the latter of which are required by the inversion routine.

- grid2dss: A program for constructing 2-D models in the format required by fm2dss. It can be used to generate checkerboard and other synthetic test models (e.g. random structure), and initial models for tomographic inversion.

- misfitss: Simply prints to screen a measure of model roughness and smoothness. It can be useful when working out the most appropriate regularization parameters for "tuning" the solution.

- residualss: Computes summary traveltime residuals (RMS and variance) associated with a given model.

- resplotss: This program takes traveltime information generated by the tomographic inversion process and converts it into a format suitable for input into a GMT (Generic Mapping Tools) script that plots a frequency histogram of the initial or final data fit.

- subinvss: Uses a subspace inversion method to perform a linearised inversion of the data. In order to address the non-linearity of the problem, this program is applied iteratively in sequence with fm2dss.

- synthtss: A simple program for adding Gaussian noise to a synthetic dataset in order to simulate the effects of picking error that is present in observational data.

- tslicess: A translation code that converts output from the tomography software into an input format that can be used by GMT (Generic Mapping Tools). This allows the velocity field to be visualized as a colour contour map, with ray paths, wavefronts and source/receiver locations superimposed. GMT scripts are provided (see the examples that come with the distribution) to generate these plots.
- ttomoss: A simple k-shell script that calls the necessary executables listed above in order to perform iterative non-linear tomography.

4 Model Generation

The program supplied for generating models is called grid2dss, and can be used to produce a 2-D grid of velocity nodes in spherical shell coordinates (essentially a curved box). The values of the velocity nodes can be set to:

1. constant
2. constant with Gaussian noise of a specified standard deviation superimposed.
3. constant with a checkerboard pattern superimposed
4. constant with random spikes superimposed

All input parameters are set using grid2dss.in, which is more-or-less self-explanatory (see examples). The minimum scalelength of structure permitted in a model is dictated by the number of grid points in θ and ϕ. At this stage, the only background model that can be used is a constant velocity field. If you require something more complicated, then you will need to generate the grid file by some other means. Note that the reference to model covariance is only relevant for subsequent use of the model in the associated inversion routine.

The output file is named grid2d.vtx in the distribution. If you create a starting model for the inversion using this code, this file should be copied to the working directory (i.e. the directory from which you execute ttomoss) and renamed gridi.vtx.

5 Traveltimes Through the 2-D Model with FMM

Traveltimes through the 2-D model are computed using FMM from specified source points. The program which applies FMM is called fm2dss and has an input parameter file called fm2dss.in, which can be used to adjust various options including the order of the upwind finite difference approximation used, whether ray paths are found etc. Two important input data files are sources.dat and receivers.dat, which define the source and receiver locations respectively. The format of these files are relatively straightforward to understand. In the case of sources.dat, the first line contains the number of sources, and all subsequent line pairs give the coordinates (latitude (+N, -S), longitude E)°E of each source. In the case of receivers.dat, the first line gives the number of receivers, and all subsequent lines give the location (latitude (+N, -S), longitude E)° of each receiver.

The simple format used for the sources and receivers implies that traveltimes are found for every source receiver pair. While this is not necessarily the case, it should be noted that once FMM computes a traveltime field, calculating two-point traveltimes takes an insignificant amount of CPU time in comparison. Thus, computing traveltimes for 10 receivers or 1000 receivers makes little difference. However, if we computed all ray paths and Fréchet derivatives as well (an *a posteriori* operation), then the additional compute time may become significant (e.g. for ambient noise tomography), which is why it is not done. Thus, another important input file is called otimes.dat, which contains a list of all the valid two-point paths. This file has a very simple format: for ns sources and nr receivers, otimes.dat contains ns $\times$ nr lines. Each line contains either a 1 or a 0, to indicate whether that particular source-receiver combination exists (1) or not (0) respectively. The order of otimes.dat must strictly conform to the following structure:

```
do i=1,ns
    do j=1,nr
        switch(j,i)
   enddo
enddo
```

where i=1,ns and j=1,nr is the entry order of sources.dat and receivers.dat respectively. This allows any arbitrary combination of sources and receivers to be specified. Note that null sources (sources which do not have any associated receivers) are not permitted.

The file is called otimes.dat because when observational data is available, it doubles as the observed traveltime input file (by placing traveltimes and picking errors after the switch(j,i) entry on each line).

The purpose of several of the parameters contained in fm2dss.in are not particularly self-explanatory. On line 8, the grid dicing in latitude and longitude is specified. These values refer to the subsampling of the continuous velocity field as defined by the cubic B-spline velocity patches, which interpolate the velocity grid. In example1, the velocity field is defined by $24 \times 28 = 672$ velocity nodes (see gridi.vtx) which are smoothly interpolated with the spline patches to form the continuous velocity field. These 672 nodes constitute the *inversion grid* - i.e. the velocity values of these nodes are adjusted by the inversion scheme in order to satisfy the data. The *propagation grid*, which is required by FMM in order to solve the eikonal equation, can be any arbitrary (but regular) resampling of the continuous velocity field, and therefore need not be related to the inversion grid. However, in fm2dss, we specify the propagation grid point separation as a function of the inversion grid point separation via a dicing factor. This dicing factor simply specifies the number of propagation grid cells that spans the same distance as one inversion grid cell. One argument for using this approach is that the node separation of the propagation grid will always be less than the minimum wavelength of structure defined by the inversion grid. In example1, the dicing specified is 8×8, which means that there will be a total of $[(24-1) \times 8 + 1] \times [(28-1) \times 8 + 1] = 40,145$ nodes defining the propagation grid.

On line 9 of the input parameter file, there is a switch to optionally apply source grid refinement. This refers to the use of a local denser mesh of propagation grid points in the vicinity of the source. One of the dominant sources of error in FMM that employs a uniformly regular grid is due to high wavefront curvature in the vicinity of a source point. This high curvature results in the correct shape of the wavefront being poorly approximated with the propagation grid. One way of dealing with this problem is to locally refine the mesh in the source neighbourhood so that the high-curvature portion of the wavefront is better represented. Thus, on line 10, one has the option of imposing a square region about the source with a higher node density. The spacing between the refined grid nodes is specified by a dicing factor - in example1, this is 5 i.e. five refined grid cells span each global propagation grid cell in both latitude and longitude (a total of 25 refined cells per global propagation cell). The extent of the refined grid refers to the extent to which the refined grid extends outwards from the source in latitude and longitude as a function of the global propagation grid cells. In example1, this value is 10, so the maximum number of local propagation grid nodes will be $(2\times10\times5+1)^2 = 10,201$. As soon as the first-arrival wavefront from the point source impinges on the edge of the local propagation grid, it is mapped back onto the coarser global propagation grid, before FMM is continued. This ensures that causality and hence the stability of the scheme is retained. The parameters of the refined grid in the default example were set using trial and error, and for most examples tested appear to produce good results; therefore, I would not recommend changing them unless you are convinced that different values will produce a superior outcome (although they have been changed in example2).

On line 12 of the input file, you can set the finite difference scheme to be first order or mixed order. This refers to the accuracy of the upwind finite difference scheme used to solve the eikonal equation. The first-order scheme has been proven to be unconditionally stable. The mixed order scheme is nominally second-order accurate, but switches back to first-order approximations when the required traveltimes for a second-order approximation are unavailable. It has not been proven that this scheme is unconditionally stable, but all of my testing has shown it to be robust. Therefore, I would recommend using the mixed-order scheme.

Finally, on line 13 there is a parameter that sets the narrow band size. This parameter is included as a memory conservation measure - we do not know the size of the narrow band (which encapsulates the first-arrival wavefront) in advance, so we set it as a fraction of the total number of propagation grid points. This is an ad hoc measure and wouldn't be required if a linked list was used to store the binary tree. The default value of 0.5 is unlikely to be exceeded, but if you want to guarantee that it doesn't, set it to 1.0 (but at the expense of more memory). For 2-D problems, the memory required to hold the propagation grid is relatively small anyway, so to some extent this memory conservation measure is unnecessary. However, the program was originally developed in 3-D, where such considerations are more important.

fm2dss can be executed to give the source-receiver traveltimes (by default put in a file called rtravel.out). Ray paths and wavefronts can be written to file if required. In the latter case, the traveltime field of only one source can be dumped in order to save disk space. Wavefronts are simply iso-contours of the traveltime field. Ray paths are computed *a posteriori* by following the gradient of the traveltime field (∇T) from each receiver, back to the source. At coarse grid resolutions, these rays can be a bit jagged, due to limited traveltime accuracy. However, they are sufficiently smooth and accurate at the type of grid spacings necessary to tackle realistic problems. The Fréchet derivatives are computed using these ray trajectories. **Note:** The switches controlling traveltime and Fréchet derivative output **must** be turned on in order to use ttomoss.

6 Inversion Routine

A locally linearized traveltime inversion is carried out using the program subinvss, which has an input parameter file called subinvss.in, the entries of which are reasonably self-explanatory. The important parameters that may need to be changed include the damping factor (set to 1.0 in example1) and the smoothing factor (set to 2.0 in example1). The damping factor effectively prevents the solution model from straying too far from the initial model, while the smoothing factor constrains the smoothness of the solution model. In practice, they are both *ad hoc* variables controlled by the user, but the idea is to get a smooth model that is not greatly perturbed from the initial model, yet satisfies the data. In version 1.1 of the software, a switch has been added that allows the user to control whether or not latitude is accounted for in the smoothing. Thus, if this switch is set to zero, the smoothing is spatially variable with latitude, in that no account is taken of the fact that with increasing latitude, the nodes become more closely spaced together along lines of longitude. If the switch is set to one, then an attempt is made to ensure that the smoothing scalength is uniform irrespective of the node location in latitude. In practice, it appears that both methods seem to produce fairly similar results.

The other variable that can be changed at the user's discretion is the size of the subspace dimension. The inversion method that is used is called subspace inversion, because it projects the full linearized inverse problem onto a much smaller n-dimensional model space. The advantage of this approach is that the solution to the inverse problem only requires the inversion of an $n \times n$ matrix. In example1, the subspace dimension is set to 10. If $n = 1$, then the solution is equivalent to that obtained by the steepest descent method. Increasing n increases the time it takes to solve the inverse problem. However, this increase in time is usually not significant when compared to the CPU time required for the solution of the forward problem. The set of vectors which span the n-dimensional subspace are computed based on the gradient vector and Hessian matrix in model space. It turns out that as n increases, the vectors become less linearly independent. Singular

Value Decomposition is used to orthogonalize the resultant set of n vectors, and throws away those vectors that are redundant. From experience, there is little point in setting n to a value greater than 10.

The variable that is rather mysteriously labelled "Fraction of max. G size for sparse matrix" is simply there as a result of the way that the sparse matrix manipulation is set up in the code. It indicates the expected maximum number of non-zero elements in the Fréchet matrix as a fraction of the total number of elements. Setting this to a larger value increases the memory requirements of the program. This value could actually be computed during the FMM step and automatically inserted into the inversion program, but I haven't got around to making this change yet. In practice, I've found that G is rarely more than 20% full, but you may need to increase this number if the program complains during execution.

7 Performing a Tomographic Inversion

A complete tomographic inversion run can be carried out simply by using the shell script ttomoss. If ttomoss is used, manual execution of all programs related to computing traveltimes and running an inversion step, as specified above, is not required. Korn shell scripting is used, so you will need ksh installed (see first line of code). Note that the authentic ksh is a commercial program not often included in Linux distributions. However, most come with the Public Domain Korn Shell, which contains several known bugs. I have tested the script with this version of ksh, and it works fine. An alternative which will also work is zsh.

The default parameter file for ttomoss is called ttomoss.in, and contains only one entry, which is simply the number of nonlinear iterations that will be performed. In example1, this is set to 6, but should be modified in accordance with your convergence criteria.

When you run ttomoss with example1, you should get something like the following output to the screen:

```
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 6
Message from SVD orthogonalization algorithm
```

```
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 6
Message from SVD orthogonalization algorithm
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 6
Message from SVD orthogonalization algorithm
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 7
Message from SVD orthogonalization algorithm
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 7
Message from SVD orthogonalization algorithm
Program fm2dss has finished successfully!
Due to redundancy, the subspace dimension
is being reduced from 10 to 7
Message from SVD orthogonalization algorithm
Program fm2dss has finished successfully!
```

Each time the FMM program is successfully executed, the line Program fm2dss has finished successfully! appears. When the inversion program is run, and it finds redundancy in the subspace vectors, a message specifying the actual number of subspace dimensions that are used is produced. From the above output, one could set the subspace dimension to 7, but there really is no need, as the program automatically does the reduction anyway. The total run time of ttomoss on a 1.6 GHz Opteron PC with 4 Gb memory and running 64 bit Suse Linux 9.1 is 11 seconds. In this case, the code was compiled with ifort (the Intel compiler in 32 bit mode).

7.1 Required input files

In order to set up an inversion using your own data, you will need the following input files. Once they are in place, all you need to do is run `ttomoss` to carry out the full tomographic inversion.

- **sources.dat**: This file specifies the location of all sources and the associated phase types that have been picked. The first 7 lines of the `example1` source file are:

  ```
  14
  -7.0 120.0
  -8.0 130.0
  -10.0 140.0
  -7.0 150.0
  -15.0 114.0
  -30.0 114.0
  ```

  The first line specifies the total number of sources `ns`. The second line gives the source coordinates in (latitude, longitude). Southern hemisphere latitudes are negative, and western hemisphere longitudes are also negative. The remaining lines simply contain the coordinates of the remaining sources.

- **receivers.dat** This file specifies the location of all receivers in the array. The first 7 lines of the `example1` receiver file are:

  ```
  35
  -35.0 120.0
  -35.0 125.0
  -35.0 130.0
  -35.0 135.0
  -35.0 140.0
  -35.0 145.0
  ```

  The first line contains the number of receivers `nr`. The second line specifies the location of the station in (latitude, longitude). Latitude is negative for the southern hemisphere, and western hemisphere longitudes are also negative. The remaining lines contain the coordinates of the other receivers.

- **gridi.vtx:** This file describes the initial or starting model velocity field. The first 7 lines of the example input file are:

  ```
  24 28
  -5.00000000 110.00000000
  1.73913043 1.85185185

  5.00000000 0.30000000
  5.00000000 0.30000000
  5.00000000 0.30000000
  ```

  The first line specifies the number of velocity vertices in (latitude, longitude). The second line contains the origin of the 2-D grid as (latitude, longitude). The third line contains the grid spacing in (latitude, longitude). Latitude and longitude spacing is in degrees. The remaining lines specify the velocity and associated *a priori* error estimate for each velocity vertex, beginning from the grid origin and looping in the order latitude, longitude. Important: The number of nodes specified on the first line include a cushion of boundary nodes that surround the computational grid. This is necessary in order to describe a smoothly varying cubic spline velocity field. The origin of the grid specified on line two defines the origin of the computational grid. For most tomographic applications, you should be able to utilize the model generation program grid2dss, in which case you don't have to worry about the concept of a boundary of cushion nodes, as it is automatically taken care of by grid2dss.

- **otimes.dat:** This file contains the observed traveltimes and source-receiver association information. The total number of entries in this file should be equal to the product of the number of receivers (given at the top of receivers.dat) and the number of sources (given at the top of sources.dat). For example1, this number is $14 \times 35 = 490$. The first 7 lines of the example file are:

  ```
  1 626.03450 0.1000
  1 629.77640 0.1000
  1 645.18100 0.1000
  1 685.74410 0.1000
  1 740.10890 0.1000
  ```

```
1 779.47130 0.1000
1 829.06100 0.1000
```

The first line contains three numbers; the first is either 1 or 0. A 1 indicates that the following traveltime residual is to be included in the inversion; a 0 indicates that the following traveltime is to be ignored in the inversion. For example, if you are unable to identify a pick at a particular station, then you can simply supply a dummy value and set the switch to 0. The second entry is the traveltime in seconds. The third entry is the error associated with the pick (also in seconds), which is subsequently used to weight the relative importance of picks in the inversion. Note that this value cannot be zero. The following lines repeat this information for all source-receiver pairs. The order of the entries is:

```
DO i=1,ns
    DO j=1,nr
        switch(j,i),tres(j,i),trerr(j,i)
    ENDDO
ENDDO
```

where ns is the number of source points, and nr is the number of receivers. (switch, tres, trerr) simply refer to the switch value, the traveltime residual, and the associated picking error.

- **Input parameter files:** As mentioned previously, you may want to edit some of the input parameter files that feed into the component programs of the tomographic inversion routine. In particular: (1) ttomoss.in to set the number of non-linear iterations; (2) fm2dss.in to adjust FMM parameters, such as propagation grid resolution and source refinement; (3) subinvss.in to control damping and smoothing regularisation.

7.2 Useful output files

When you perform an inversion, the code generates output files that you will need in order to visualize the solution model and see how well it satisfies the data. These files include:

- **gridc.vtx:** This file describes the velocity field of the solution model. It has exactly the same format as the reference velocity field file gridi.vtx
- **rtravel.out:** This file contains the source-receiver traveltimes through the solution model as predicted by fm2dss.
- **itimes.dat:** This file contains the source-receiver traveltimes through the reference model.
- **raypath.out:** This file contains raypath geometries generated by fm2dss, if they are turned on in fm2dss.in. This file is in binary format. Refer to the next section on plotting for more information.
- **frechet.out:** This file contains the Fréchet matrix in binary format. It is essential that fm2dss.in is set up so that this file is generated, as it is required by the inversion routine.
- **residuals.dat:** This file contains the RMS value and variance of the current model traveltime residuals at each iteration. In the case of example1 provided with the distribution, this file should contain:

  ```
  10989.16 121.00856
  3625.61 13.17196
  1499.60 2.25340
  957.51 0.91870
  587.29 0.34561
  455.32 0.20774
  380.26 0.14489
  ```

  The first line corresponds to the residual for the starting model, and each subsequent line corresponds to the residual for the model produced by each of the six iterations. For each line, the first value is the RMS data residual in ms, and the second value is the data variance in s^2.

8 Plotting the Results

Plotting programs are provided for visualising various components of the output from the FMM code. These plotting programs are based on GMT scripts. They allow wavefronts and rays computed by FMM to be superimposed on the velocity field.

The program that takes output from the FMM code and converts it into a format suitable for input into GMT scripts is tslicess. The input parameter file is tslicess.in, which is largely self-explanatory. In addition to plotting velocity perturbations, it allows wavefronts and rays to be superimposed on the projection. The resolution of the velocity plot is controlled by user-supplied inputs.

A GMT plotting script called plotgmt is provided for both example1 and example2 (see the subdirectory gmtplot in each case). Insert or remove # at the beginning of the relevant command line to include or exclude wavefronts and rays. The default output postscript file is plotgmt.ps, but this can be easily modified. Some knowledge of GMT programs/scripts is required in order to obtain satisfactory plots.

Other plotting programs that may be useful include resplotss and gmthist. These programs allow a frequency histogram of the traveltime residuals for a particular model to be plotted. resplotss, which is located in bin/ is a Fortran 90 program that simply converts output from the inversion programs into a format suitable for input into GMT. gmthist is the GMT script that performs the actual plotting, and can be found in the subdirectory gmtplot in both example1 and example2. The input parameter file for resplotss (called resplotss.in) is self explanatory. The first entry sets a switch that generates traveltime residual files for the initial model or the solution model.

9 Examples

Two examples are provided with the code. The first is contained in the directory example1/, and the second in example2/.

9.1 Example 1

This example is purely synthetic, and uses a contrived source-receiver geometry to recover a checkerboard pattern. In order to run the example and produce output data, simply enter the subdirectory example1/ and execute ttomoss at the command line. The execution time on a recent PC should only be about 10 s or less.

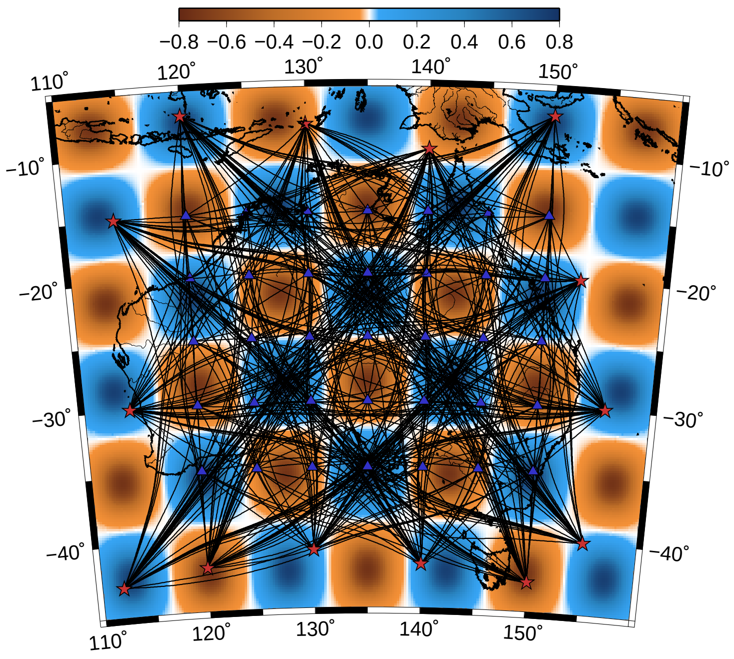

Figure 1: Example output from plotgmt using example1 provided with the distribution. In this case, ray paths through the synthetic test model are plotted along with the source (red stars) and receiver (blue triangles) locations.

All of the input files necessary to adjust the forward prediction and inverse step are present, as are the source, receiver and "observed traveltime" files. These can be edited if you wish to see how the inversion responds to changes.

In order to plot the results using the Generic Mapping Tools (GMT - freely available from http://gmt.soest.hawaii.edu/), enter the subdirectory gmtplot/. Simply execute tslicess and then plotgmt in order to produce plotgmt.ps. This can be visualized using ghostview or some other postscript viewer. Figure 1 shows the result of this action for the true model (i.e. the synthetic checkerboard) and Figure 2 shows the result for the recovered model. In both cases, ray paths are superimposed on the velocity model. Note that Figure 1 was plotted by running fm2dss with fm2dss.in edited so that gridt.vtx (the true model) generated the ray paths that are observed. Similarly, the first line of tslicess.in can be edited to read either the true model or the inversion model gridc.vtx

Within the same subdirectory, it is possible to generate a plot of a frequency histogram that shows the traveltime misfit of the current model. Simply run resplotss and then

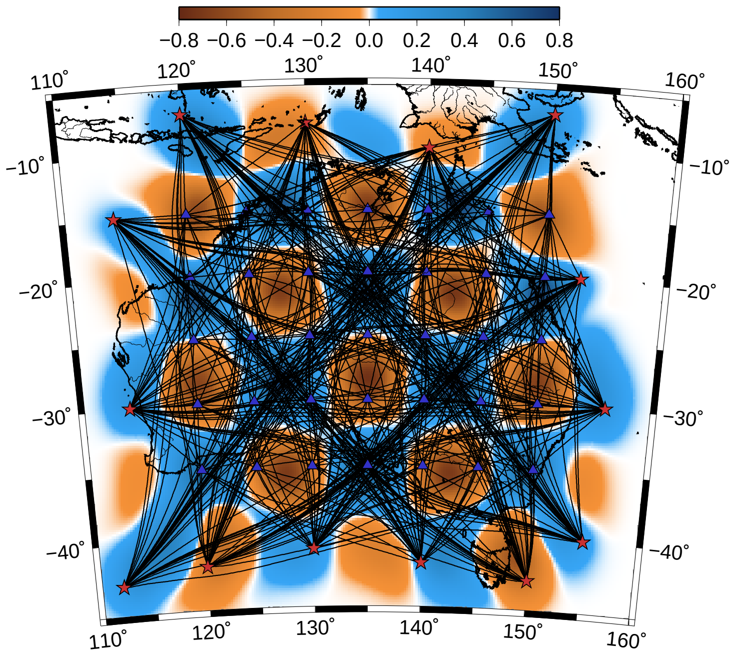

Figure 2: Similar to Figure 1, but now showing the recovered model obtained after six iterations of the tomographic method.

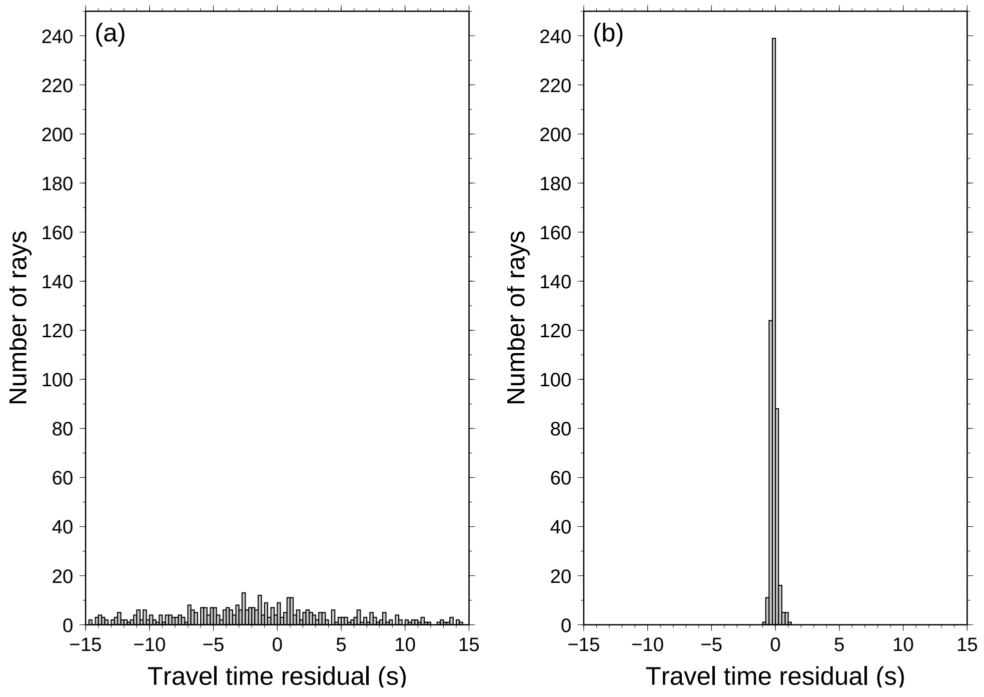

Figure 3: Frequency histograms showing fit to "observed" traveltime data of (a) initial model, and (b) solution model.

gmthist to generate gmthist.ps. The input file resplotss.in can be edited in order to toggle between the initial and solution models. Figure 3a shows the frequency histogram for the initial (constant velocity) model, and Figure 3b is the equivalent plot for the solution model obtained after six iterations.

Other things to note about this example are that the synthetic and initial models were constructed using the input file located in the subdirectory mkmodel (simply enter this directory and execute grid2dss), and the synthetic "observed" dataset was generated by the input file in mkdata (simply enter this directory and execute synthtss). In the latter case, you must of course first generate the correct rtravel.out file using fm2dss with gridt.vtx.

9.2 Example 2

Example 2 is a real data example that inverts traveltimes derived from long term inter-

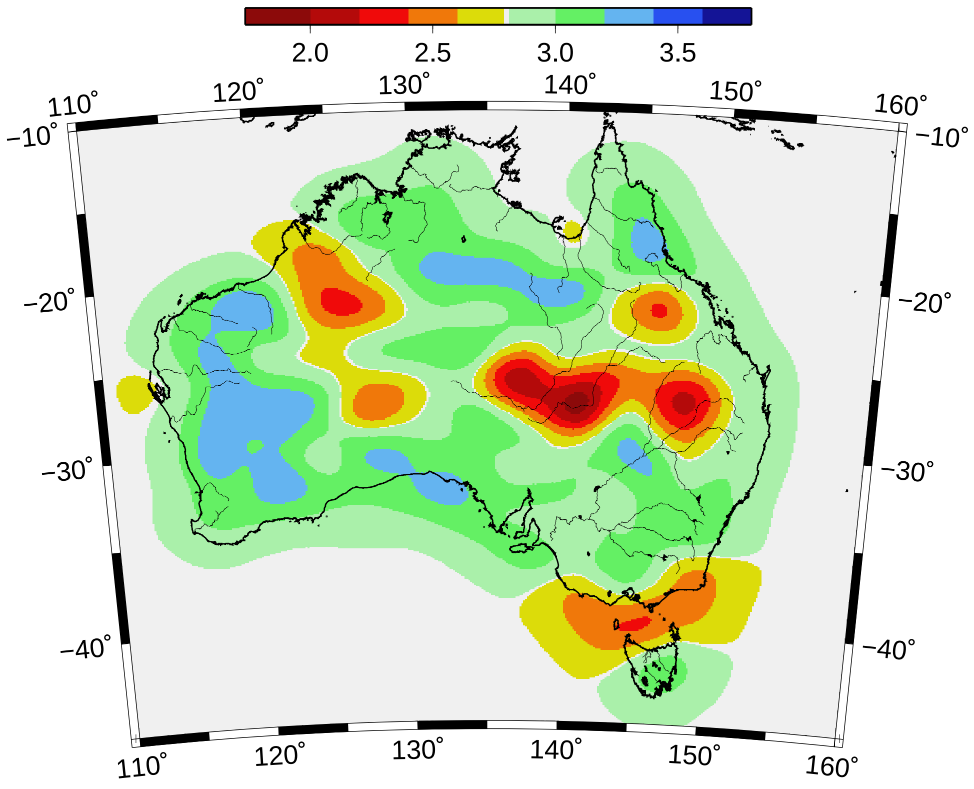

Figure 4: Ambient noise tomography inversion result showing significant variations in Rayleigh wave group velocity throughout the Australian continent.

station cross-correlations of the ambient noise field recorded by seismic arrays located on the Australian continent. The aim of this ambient noise tomography example is to constrain the mid-upper crustal Rayleigh wave group velocity of the Australian plate. More detail about this example is contained in Saygin, E. 2007. Seismic receiver and noise correlation based studies in Australia, PhD thesis, Australian National University.

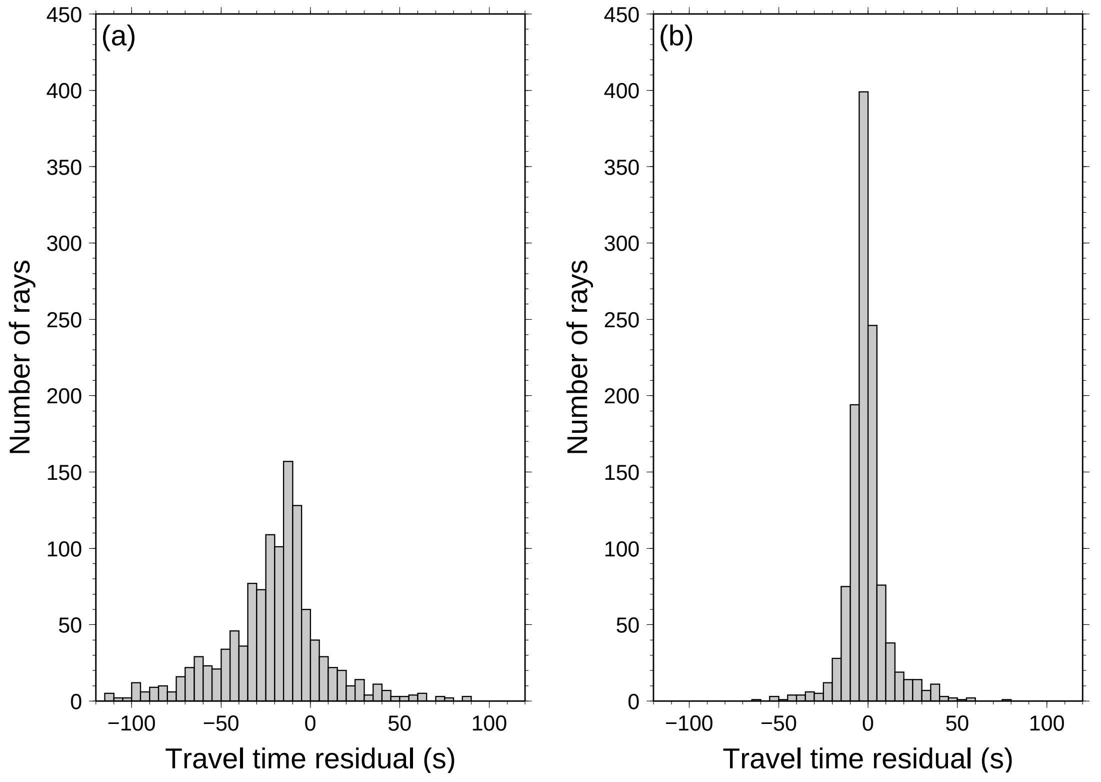

Figure 5: Frequency histograms showing fit to observed traveltime data of (a) initial model, and (b) solution model.

Apart from dealing with observational data, example2 differs from example1 in several important ways. First, the source location file sources.dat is identical to receivers.dat. This is because all of the receivers are virtual sources. Second, the input traveltime file otimes.dat, despite being very large ($208 \times 208 = 43,264$ lines long, since there are 208 receivers), is quite sparse. In example1, all switches were set to one because every source receiver pair has an associated traveltime. Here, due to only a relatively small proportion of the total number of receivers being deployed at once, and an intrinsic redundancy of sources and receivers (source i and receiver j is the same combination as source j and receiver i), most of the switches are set to zero.

As in the previous example, this example can be run by simply entering the directory example2 and executing ttomoss. On a relatively recent PC, the execution time should be around 30 s or less. This is slower than the previous example, due to the large number of sources. However, I have made both the propagation grid and refined source grid used by FMM a bit smaller to compensate for this, which can be justified because the data errors are quite large, and decreasing the grid size makes little difference to the result. Figure 4 shows the result of the inversion, plotted in exactly the same manner as the previous example. The only differences are that now, absolute rather than relative velocity is plotted (this setting can be changed in the resplotss.in file), and ray paths, sources and receivers are omitted (this can be changed in plotgmt). Figure 5 shows a frequency histogram comparison between the traveltime fit of the initial and solution models. Again these plots were generated in exactly the same manner as in the previous example.

10 FAQs

1. **Q.** *How can I optimize the speed of the tomographic inversion?*
   A. The key to getting the maximum performance from the code is to specify the coarsest grid spacing that still produces traveltimes with sufficient accuracy. This is because the traveltime prediction step is usually much slower than the inversion step, and FMM scales as O(NlogN), where N is the total number of points on the computational grid. Therefore, if you halve the grid spacing (by editing fm2dss.in), the computing time will increase by a factor of at least 4. In general, I find that something like 50,000 nodes is usually adequate, but of course this will vary depending on the problem you are dealing with. My approach is to use a coarse dicing factor (like 2 or 3) and then increase it and see whether the solution model changes significantly. If not, then there is no point in increasing the dicing factor any further.

2. **Q.** *What happens when a ray path intersects the boundary of the medium?*
   A. When this occurs, the ray simply tracks along the edge of the medium and will form a two point ray path. It will be the shortest path for the defined medium, but since the boundary locations are arbitrarily defined, it will most likely not be a true ray path. fm2dss will notify you if any of these paths occur - if they do, then it is strongly recommended that you extend the boundary of the medium.

3. **Q.** *Is the code very memory hungry, particularly for large problems involving many*

unknowns and paths?

A. In short, no. The largest problem I have tried involved over 100,000 unknowns and 6,000 ray paths, and even then the memory used was in the 10s of Mb rather than 100s of Mb.

4. **Q.** *Is there an easy way to set up synthetic tests like a checkerboard resolution test?*

   A. Checkerboard or spiking resolution tests are quite easy to set up. The grid generation program grid2dss has a facility for producing 2-D checkerboards, spikes or random patterns. Copy the output .vtx file from this program into the appropriate location for input into fm2dss. The next step is to run fm2dss to generate traveltimes for this model. Once this is complete, copy the file rtravel.out to the subdirectory containing synthtss.in. Then execute the program synthtss which will generate otimes.dat. This can be copied to the directory from which you execute ttomoss, where it can then be used as the "observed" traveltime values.

5. **Q.** *Will new features be added to the code in future?*

   A. I hope to fix any obvious bugs, and perhaps include station terms as unknowns in the inversion. In the longer term, I may allow the use of teleseismic sources (so that wavefronts from a distant source are initiated at the edge of the model), include weak anisotropy, compute geometric spreading, and allow for finite frequency effects.

6. **Q.** *I want to know more about FMM and subspace inversion.*

   A. A number of papers have been published on FMM and its applications. Several useful references are listed below:

   - Rawlinson, N. and Sambridge M., 2005. The fast marching method: An effective tool for tomographic imaging and tracking multiple phases in complex layered media, *Explor. Geophys.,* **36,** 341-350.
   - Rawlinson, N. and Sambridge, M., 2004. Multiple reflection and transmission phases in complex layered media using a multistage fast marching method, *Geophysics,* **69,** 1338-1350.
   - Rawlinson, N. and Sambridge, M., 2004. Wavefront evolution in strongly heterogeneous layered media using the fast marching method, *Geophys. J. Int.,* **156,** 631-647.

- Rawlinson, N. and Sambridge, M., 2003. Seismic traveltime tomography of the crust and lithosphere, *Advances in Geophysics,* **46,** 81-198.

These and other potentially useful papers can be downloaded from:
http://rses.anu.edu.au/∼nick/