FMTT: Fast Marching Teleseismic 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 FMTT for performing teleseismic arrival time tomography using the fast marching method (FMM) for the forward prediction step and a subspace inversion scheme 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 perturbations to be reconciled.

Traveltimes from distant events are computed to the base of a local 3-D model beneath a receiver array using *ak*135 predictions (i.e. the earth is assumed to be spherically symmetric outside the local model). FMM is then initiated from the base of the model to compute traveltimes to receivers located anywhere within the model. Additional software is provided to generate 3-D models in spherical coordinates. A variety of structures can be imposed including checkerboards, spikes and random (Gaussian) variations. These models are defined by a grid of nodes with tri-cubic B-spline interpolation, which produces a continuous, smooth and locally controlled velocity medium. This continuous velocity field is "diced up" to produce a computational grid for FMM - the level of dicing (and hence traveltime accuracy) is controlled by the user.

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.

A recently added feature of this software package is the ability to combine data sets from more than one teleseismic array in a single inversion. This feature can be useful, particularly if a sequential roll-out of seismic arrays is performed to cumulatively cover a region of the Earth. However, it should be noted that if relative arrival time residuals are used in such circumstances, long wavelength features of the underlying structure will not be recovered. In other words, the solution model will represent a high-pass filtered version of the true structure, with the degree of filtering dependent on the size of the instrument arrays used. To overcome this limitation, one could use a background starting model derived by other means (e.g. regional/global tomography) that contains this information. I have had a number of requests for this 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 fmtt_v1.0.tar.gz from the web page located at http://rses.anu.edu.au/∼nick/teletomo.html into an empty directory. To unpack the contents of the file, use either:

- tar -xvzf fmtt_v1.0.tar.gz
- gunzip -c fmtt_v1.0.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.
- Move to the subdirectory source/aktimes/ and edit line 5 of Makefile so that it points to an appropriate Fortran 77 compiler. The code which makes the *ak*135 traveltime predictions is written in Fortran 77. The default setting for the Fortran 77 compiler is ifort, the Intel fortran compiler. However, most Fortran compilers (e.g. g77) should work. Note that it is generally advisable to use the same fortran compiler to build the ak135 traveltime tables and FMST, since binary files are exchanged between programs.

While every effort has been made to use standard Fortran 77 and 90, 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. The only Fortran 77 compiler tested was g77.
- Linux PC with Intel P4 processor running a 32 bit operating system. Fortran 90 compilers tested include g95 and ifort (Intel fortran compiler). The only fortran 77 compiler tested was g77.
- Sun Blade 100 running Solaris. Fortran 90 compilers tested include frt (Fujitsu Fortran 90 Compiler) and f90 (Sun Fortran 90 compiler). Fortran 77 compilers tested include g77 and f77 (Fortran 77 SC4.2).

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). [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, it will produce screen output related to the Fortran 77 Makefile and creation of the *ak*135 traveltime tables. Successful compilation will be marked by the words Compilation complete. The compilation time on a 64 bit AMD platform with a 1.6 GHz Opteron processor using ifort is about 6 seconds. All of the 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 tomo3dt, 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 fm3dt has an input parameter file called fm3dt.in. Example input parameter files can be found in the subdirectory source/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 Synopsis

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.

- aktsurf: This program computes *ak*135 traveltimes from each teleseismic source to all grid nodes (within a specified region) that reside on the surface of the local 3-D propagation mesh. These times are back-tracked to the base of the model using the program itimes.

- fm3dt: Solves the forward problem of traveltime prediction by applying the so-called Fast Marching Method (FMM), a grid based eikonal solver. fm3dt can also generate ray paths and Fréchet derivatives, the latter of which are required by the inversion routine. Note that this code requires traveltimes to grid points at the base of the model to be specified, as it is designed to track an impinging teleseismic wavefront through a local 3-D velocity structure.

- gmtslicet: A translation code that converts output from the tomography code into a format suitable for input into GMT (Generic Mapping Tools) scripts. This allows the velocity field to be visualized as colour-contoured cross-sections, with ray paths, wavefronts, sources and receivers projected if required. GMT scripts are provided (see the examples that come with the distribution) to generate these plots.

- grid3dtg: A program for constructing 3-D models in the format required by fm3dt. It can be used to generate checkerboard and other synthetic test models (e.g. random structure), and initial models for tomographic inversion.

- itimes: Takes the output from aktsurf and backtracks the traveltimes to the base of the 3-D model. The output from this program is read in by fm3dt.

- misfitt: A simple program for computing model roughness and variance. Can be useful when trying to justify the most appropriate damping and smoothing regularisation parameters.

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

- resplott: 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.

- subinv: 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 fm3dt.

- syntht: 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.

- tomo3dt: 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 grid3dtg, and can be used to produce a 3-D grid of velocity nodes in spherical coordinates (essentially a curved 3-D box). The values of the velocity nodes can be set to:

- Arbitrary depth dependent background values (i.e. a model in which velocity varies with depth only - often appropriate for a tomographic starting model).

- Background model with Gaussian noise of a specified standard deviation superimposed.
- Background model with a checkerboard pattern superimposed.
- Background model with random spikes superimposed.

All input parameters are set using grid3dtg.in, which is more-or-less self-explanatory (see examples). The minimum scale-length of structure permitted in a model is dictated by the number of grid points in r, θ and ϕ. At this stage, only depth varying background models can be produced - 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.

It is probably worth giving a brief explanation of several of the input parameter settings in grid3dtg.in. On line 16, there is a switch that states Use model (0) or constant gradient (1). Setting this to 0 means that a user-supplied list of velocity versus depth values will be read in to build the background model. A setting of 1 indicates that velocity will vary linearly with depth only. The following line (17) is only relevant if option 0 is chosen on line 16. The filename supplied on line 18 is the user input file (used if option 0 is chosen on line 16) which provides a list of depths and velocities. For example, if the $ak135$ velocity model is used, then the first 10 lines of the file would look like:

```
0.00000    5.8000    3.4600
20.0000    5.8000    3.4600
20.0000    6.5000    3.8500
35.0000    6.5000    3.8500
35.0000    8.0400    4.4800
77.5000    8.0450    4.4900
120.000    8.0500    4.5000
165.000    8.1750    4.5090
210.000    8.3000    4.5180
210.000    8.3000    4.5230
......
```

The first entry on each line is the depth, the second entry is the P-wavespeed, and the third entry is the S-wavespeed. Provided this basic format is retained, any depth dependent velocity model can be specified. Note that the actual variation of velocity with depth in the 3-D model will be a smoothed version of what is specified in the file, due to the use of a cubic B-spline parameterization to specify the continuous velocity field. In addition, the full detail of the input 1-D model may not be properly captured by the velocity grid, depending on the node spacing that is chosen in depth. Finally, it it worth noting that grid3dtg automatically creates a cushion layer of boundary nodes around the specified velocity model to facilitate the use of cubic B-splines; therefore, it is necessary to make sure that the input 1-D velocity model extends above and below the actual model region specified near the top of grid3dtg.in

Lines 19 and 20 of grid3dtg.in are only relevant if option 1 is chosen on line 16. In this case, velocity varies linearly with depth, and the value at the top of the model is given by the entry on line 19, and the value at the base of the model is given by the entry on line 20.

The output file is named grid3d.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 tomo3dt) and renamed gridi.vtx.

5 Computing Traveltimes to Base of 3-D Model

Two programs are used to calculate traveltimes from a distant (teleseismic) source to all points at the base of the 3-D computational grid. The first program, aktsurf, uses the ttimes software to compute the traveltimes of any global phase through a spherically symmetric Earth (defined using $ak135$ velocities) to all grid points on the *surface* of the 3-D model. The second program, itimes, uses ray tracing to compute the traveltime from the surface of the model back to the base of the model and subtracts this value from the global times provided by aktsurf. Because these backprojected traveltimes can be irregularly distributed at the base of the 3-D model, itimes performs interpolation to obtain the traveltimes at the regularly distributed grid points. **Note:** Depending on the geometry of the impinging teleseismic wavefront, not all grid points at the base of the model will be assigned traveltimes. Therefore, it is important that the horizontal bounds of the computational grid are significantly larger than the horizontal bounds of the receiver array.

5.1 aktsurf

The program aktsurf has a default input parameter file called aktsurf.in, which simply points to a number of input and output files. In order for this program to operate, the binary files ak135.hed and ak135.tbl **must** be in the working directory from which you run aktsurf. These two files were created when you ran the script compileall to build the executables, and can be found in the subdirectory source/aktimes/. Note that these binary files are machine dependent (e.g. if they were built on a Sun workstation, they are unlikely to work on a Linux PC) and can be compiler dependent; therefore, it is recommended that they be constructed each time the distribution is installed.

aktsurf **only** needs to be rerun when **either** of the following three situations arise:

- The sources.dat file is edited.
- The grid spacing and/or dimensions of the computational grid are changed in any way.
- You switch from a P-wave local 3-D model (default) to an S-wave local 3-D model.

5.2 itimes

itimes has a default input file called itimes.in, which lists a number of input and output files as well as variables used by the program. Most of these entries are self-explanatory (see examples), but it is still worth examining several of them. On line 1, the 1-D reference velocity model should be the original *ak*135 velocity model, as this is what is used by aktsurf by default (if the binary traveltime tables are built using a variant of *ak*135, then the same model should be used by itimes). On line 8, the entry reads Depth discretization of 1-D model (km). Snell's law is used to back project traveltimes from the surface to the base of the model. This parameter sets the depth interval of the stratification used to approximate the depth-varying model. The default value should be fine in most circumstances.

itimes only needs to be rerun when **either** of the following situations arise:

- The sources.dat file is edited.
- The grid spacing and/or dimensions of the computational grid are changed in any way.

- You switch from a P-wave local 3-D model (default) to an S-wave local 3-D model.

6 Traveltimes Through the 3-D Model with FMM

Traveltimes through the local 3-D model are computed using FMM with the starting narrow band encapsulating all points at the base of the grid. The program which applies FMM is fm3dt and has an input parameter file called fm3dt.in, which can be used to adjust various options. The list below describes several of those whose meaning may not appear obvious on a perusal of the file. See the subdirectories example1 and example2 for examples of fm3dt.in.

- On line 8 of the file, the comment reads Bspline dicing in r, theta, phi. This entry sets the grid spacing of the propagation grid through which FMM computes traveltimes. The three integers specify the radial, latitudinal and longitudinal sub-sampling 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 $13 \times 35 \times 37 = 16,835$ velocity nodes (see gridi.vtx), excluding the boundary of cushion nodes. Cubic B-spline patches are used to define a smoothly varying velocity continuum between these control nodes, which 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 fm3dt, 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 $2 \times 2 \times 2$, which means that there will be a total of $[(13-1)\times 2+1]\times[(35-1)\times 2+1]\times[(37-1)\times 2+1] = 125,925$ nodes defining the propagation grid.

- On line 10 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.

- On line 11 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).
- On line 12, there is an option that reads limit traveltime field to receivers. This means that fm3dt will terminate when all grid points at the surface within a specified boundary around a receiver array have an associated traveltime. This measure can save CPU time, particularly when multiple receiver arrays are specified that together span a large region.
- On line 13, the boundary around the receiver is set, but only used if the option on line 12 is set to 1. In both example1 and example2, the cushion is set to 0.4°, which means that fm3dt will terminate as soon as all points within a distance of 0.4° from the edge of each array (only one in the case of example1) have associated traveltimes.

fm3dt 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 tomo3dt.

7 Inversion Routine

A locally linearized traveltime inversion iteration is carried out using the program subinv, which has an input parameter file called subinv.in, the entries of which are reasonably self-explanatory. The important parameters that may need to be changed include the Damping factor (set to 5.0 in example1) and the Smoothing factor (set to 10.0 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.

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 not significant compared to 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 5% full, but you may need to increase this number if the program complains during execution.

Finally, an option exists for including station terms as unknowns in the inversion. Due to ray path geometry, teleseismic data do not constrain shallow structure (the upper limit is largely controlled by station spacing), which often contributes significantly to arrival time residual patterns. One means of trying to mitigate the influence of these contributions in the reconstruction of well resolved deeper structure is to allocate a traveltime residual parameter (or station term) to each receiver station. By including these as unknowns in the inversion, the hope is that shallow structural contributions to arrival time residual patterns will be absorbed by the station term rather than get erroneously mapped into deeper structure. Although commonly used, I personally find the inclusion of station terms to be somewhat dubious, as they are generally poorly constrained, and can be very sensitive to the imposed regularization. Nevertheless, I have included them as an option for completeness, but would encourage the user to exercise caution in their use. At the very least, I suggest plotting the values produced by the output file (stationres.dat) to see if the pattern makes any sense. [**NOTE:** If you decide to include station terms, it is worth increasing the subspace dimension from the default value of 10 (e.g. to 20). This is because the velocity and station terms represent different parameter classes and are minimized along orthogonal search directions.]

8 Performing a Tomographic Inversion

A complete tomographic inversion run can be carried out simply by using the shell script tomo3dt. If tomo3dt is used, manual execution of all programs related to computing traveltimes and running an inversion step, as specified above, is not required. tomo3dt is a Korn shell script, so you will need ksh installed (see first line of code). Note that the authentic ksh is a commercial product 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. zsh may also be used.

The default parameter file for tomo3dt is called tomo3dt.in, and contains only two entries. The first entry asks whether traveltimes to the base of the model need to be computed. If the source input file and diced velocity grid are unchanged from the previous run of tomo3dt, then this can be set to zero, which will speed up the inversion slightly as *ak*135 traveltimes do not need to be re-computed in this case. The second entry prompts the user to specify the number of iterations of sequential forward and inversion steps. In example1, this is set to 6, but in general should depend on your convergence criteria.

When you run tomo3dt with the default input files and parameters for example1, you should get something like the following output to the screen:

```
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
Program fm3dt has finished successfully!
```

Each time the FMM program is successfully executed, the line Program fm3dt 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. The total run time of tomo3dt on a 1.6 GHz Opteron PC with 4 Gb memory and running 64 bit Suse Linux 9.1 is 12 minutes. In this case, the code was compiled using ifort (the Intel compiler in 32 bit mode).

8.1 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 tomo3dt 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 8 lines of the example1 source file are:

  ```
  1
  163
  -6.260000 150.5800 50.00000
  P
  8.150000 94.14000 17.00000
  ```

```
P
-31.64000 -177.9900 9.000000
PcP
```

The first line specifies the number of receiver arrays present. This would normally be set to 1, unless there are multiple adjacent arrays present that have been deployed at different times (i.e. a rolling array type scenario), and hence have detected different sources. The second line nominally specifies the total number of sources $ns(i)$ for array i. The third line gives the source coordinates in (latitude, longitude, depth). Southern hemisphere latitudes are negative, and western hemisphere longitudes are also negative. Depth is in km and is positive (i.e. 50.0 means 50.0 km below sea level). The third line indicates the phase type associated with the source. The remaining lines simply repeat lines three and four for the remaining sources and phases. All phase types should end in P if a P-wave model is specified, and S if an S-wave model is specified. Note that if there is more than one phase type associated with a single source, then the source location and phase type entries need to be repeated for each phase. For example, if one picked P, PcP and PKiKP from one event, then six lines would be added to the file, with the source coordinates repeated three times, one for each phase. $ns(i)$ would also need to be increased by 3. Thus, $ns(i)$ doesn't really represent the total number of sources, but the total number of phases picked from all sources. If additional arrays are present, then multiple source lists must be appended.

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

```
1
20 SEAL ARRAY STATIONS
0.292 -35.4463 147.0653 se01
0.101 -35.4618 146.2128 se02
```

The first line specifies the number of receiver arrays present, and should be identical to the first line of sources.dat. The second line contains the number of receivers $nr(i)$ and the name of array i. This name can be omitted as it is not read in by any program. The third line specifies the location of the station in (height, latitude, longitude). Height is in km above sea level and latitude is negative for the southern hemisphere. The last entry is the station name, but again this can be omitted as it is not read in by any program. The remaining lines contain the coordinates of the other stations. This is repeated for additional arrays if they are present.

- **gridi.vtx:** This file describes the initial or starting model velocity field. The first 7 lines of the example1 input file are:

  ```
  13 35 37
  1.500000 -31.500000 138.000000
  25.000000 0.239000 0.355000

  4.90000000 0.30000000
  4.90000000 0.30000000
  6.58750000 0.30000000
  ```

  The first line specifies the number of velocity vertices in (depth, latitude, longitude). The second line contains the origin of the 3-D grid as (height, latitude, longitude). Height is in km above sea level, and should be set at or above the elevation of the highest station in your array. The third line contains the grid spacing in (depth, latitude, longitude). Depth spacing is in km, while 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 depth, latitude, longitude. Important: The number of nodes specified on the first line doesn't include a cushion of boundary nodes that surrounds the computational grid, although the file actually contains these nodes. 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 grid3dtg, 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 grid3dtg.

- **otimes.dat:** This file contains the observed arrival time residuals. 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 total number of phases that have been picked (given at the top of sources.dat). If more than one receiver array is present (as in example2), then the total number of entries is equal to $\sum_{i=1}^{n} ns(i) \times nr(i)$, where n is the number of receiver arrays. For example1, $n = 1$, $ns = 163$ and $nr = 20$, so the number of entries is $163 \times 20 = 3260$. The first three lines of the example1 file are:

```
1 -0.4131579 6.0899999E-02
1 -0.5131579 5.2299999E-02
1 0.1368421 3.7500001E-02
```

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 arrival time residual 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,n
   DO j=1,ns(i)
      DO k=1,nr(i)
         switch(k,j,i),tres(k,j,i),trerr(k,j,i)
      ENDDO
   ENDDO
ENDDO
```

(switch, tres,trerr) simply refer to the switch value, the arrival time residual, and the associated picking error respectively.

- **rtimes.dat:** This file contains reference traveltimes for the initial model. In most cases, you will not need to worry about this file as it is automatically generated by tomo3dt. However, if your relative arrival time picks have not been corrected for topography, and station elevations vary, then you may want to generate your own reference traveltimes. To do this, run fm3dt with all station elevations set to zero with the initial model. Then rename the output file rtravel.out to something like rtimesf.dat. Finally, edit the input file subinv.in (and residualst.in) so that the file containing the reference traveltimes is correctly named. It is important that you do not simply call this new file rtimes.dat, as tomo3dt will by default copy rtravel.out to this filename for the initial model, which, when the inversion is run, will have non-zero station elevations.

- **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) tomo3dt.in to set the number of non-linear iterations; (2) fm3dt.in to set the resolution of the computational grid and to adjust other FMM parameters (in general, I wouldn't change these except for the output file parameters); (3) subinv.in to control damping and smoothing regularisation.

8.2 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. In order to obtain arrival time residuals through the solution model, simply take the difference between these values and rtimes.dat.
- **raypath.out:** This file contains raypath geometries generated by fm3dt, if they are turned on in fm3dt.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. It is in binary format. It is essential that fm3dt.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 arrival time residuals at each iteration. The example1 file that comes with the

distribution contains:

```
197.97 0.03920
105.39 0.01111
97.46 0.00950
95.07 0.00904
94.08 0.00885
93.52 0.00875
93.14 0.00868
```

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.

9 Plotting the Results

A suite of plotting programs are provided for plotting various components of the output from the FMM code. These plotting programs are based on GMT scripts, and 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 `gmtslicet` and has an input parameter file called `gmtslicet.in`, which is largely self-explanatory. It allows the user to take depth, N-S and/or E-W slices through the velocity model and/or traveltime field. In addition, it allows all ray paths to be projected onto any of these three planes. The resolution of the velocity plot is controlled by user-supplied inputs.

Three GMT plotting scripts are provided (see subdirectories `example1/gmtplot/` and `example2/gmtplot/`):

- **plotd** Plots a depth slice. Insert or remove # at the beginning of the relevant command line to include or exclude wavefronts and rays.

- **plotns** Plots a N-S slice. Insert or remove # at the beginning of the relevant command line to include or exclude wavefronts and rays.

- **plotew** Plots an E-W slice. Insert or remove # at the beginning of the relevant command line to include or exclude wavefronts and rays.

The default output postscript file in all three cases is gmtslice.ps, but this can be easily modified. Some knowledge of GMT programs/scripts is required in order to obtain satisfactory plots - see http://gmt.soest.hawaii.edu/.

Other plotting programs that may be useful are resplott and gmthist. These programs allow frequency histograms of the arrival time residuals for a particular model to be plotted. resplott is a Fortran 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.The input parameter file for resplott (called resplott.in) is self explanatory. The first entry sets a switch that generates arrival time residual files for the initial model or the solution model (see example1/gmtplot/ or example2/gmtplot/).

10 Examples

10.1 Example 1

The first example provided with this distribution (see subdirectory example1) images the upper mantle beneath the Murray Basin in southeast Australia by inverting arrival time residuals recorded by an array of 20 short period stations. Refer to the paper seaustralia.pdf located in the subdirectory docs for more detail about this experiment, data analysis, results and interpretation. Prior to running this example, ensure that the binary files ak135.hed and ak135.tbl are copied to this directory (they should be located in source/aktimes after compilation). Once this has been done, simply execute tomo3dt.

In order to visualize the results using the Generic Mapping Tools (GMT - freely available from http://gmt.soest.hawaii.edu/), enter the subdirectory gmtplot and execute gmtslicet. The three GMT scripts plotd, plotew and plotns contained in this directory allow depth, E-W and N-S slices to be plotted. By default, each of these scripts produces a postscript file called gmtslice.ps. Figure 1 shows the default output of the three scripts if the above procedure is carried out.

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 resplott and then gmthist to generate gmthist.ps. The input file resplott.in can be edited in order to toggle between

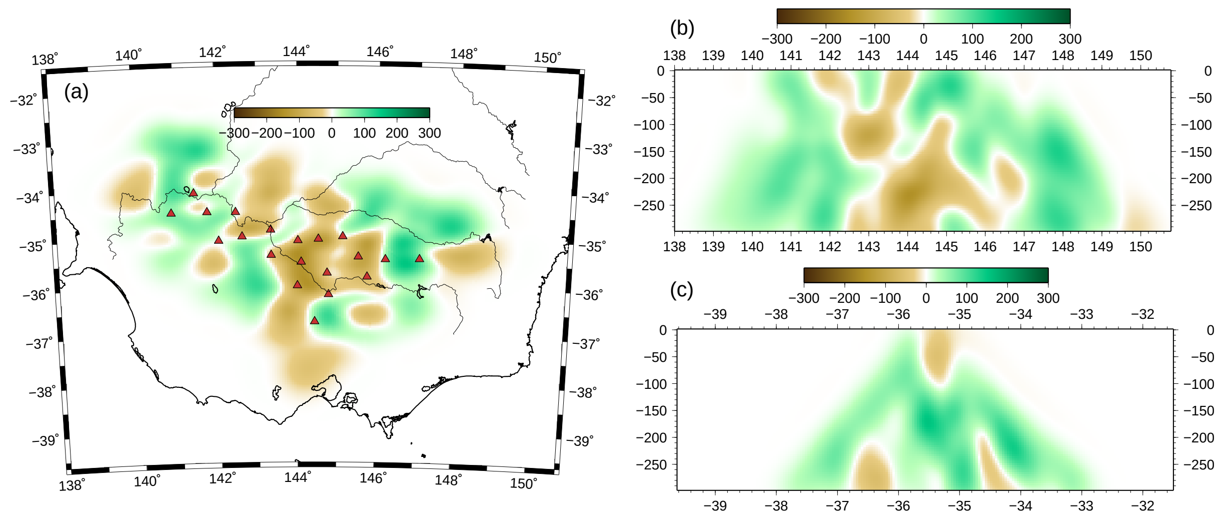

Figure 1: Example output from gmtslicet using example1 provided with the distribution. (a) Depth slice; (b) E-W slice; (c) N-S slice.

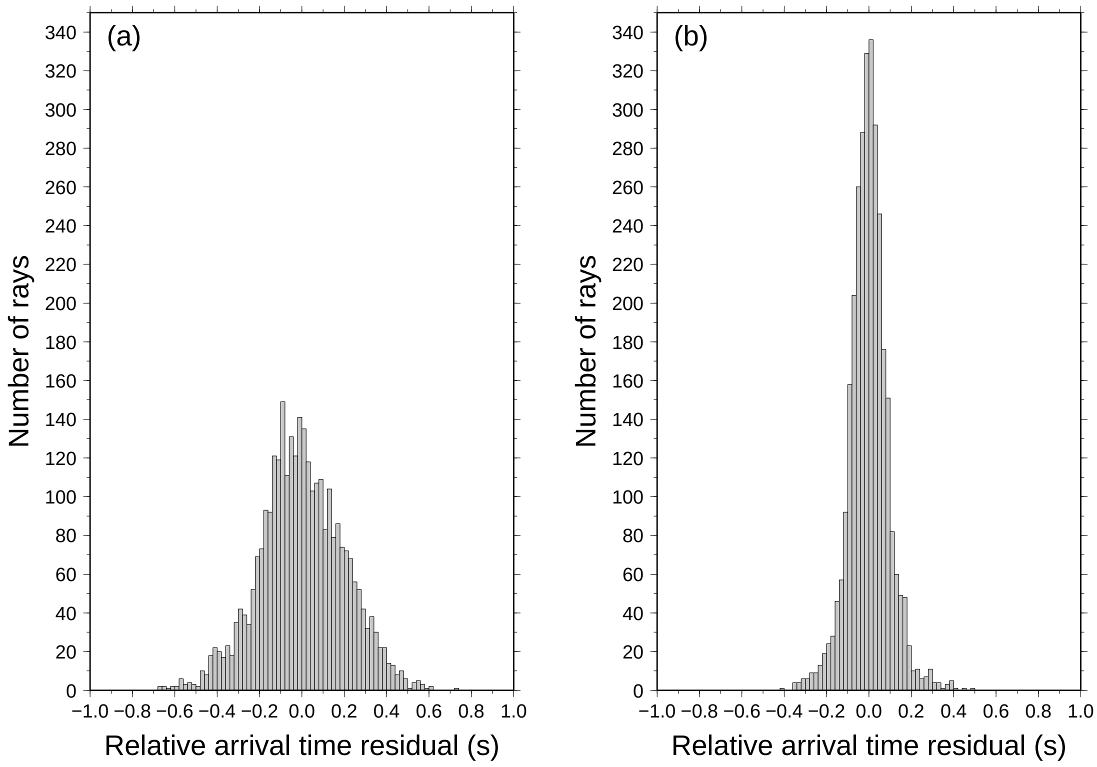

Figure 2: Frequency histograms showing fit to observed arrival time residual data of (a) initial model, and (b) solution model.

the initial and solution models. Figure 2a shows the frequency histogram for the initial (constant velocity) model, and Figure 2b is the equivalent plot for the solution model obtained after six iterations.

Other things to note about this example are that the initial model was constructed using the input file located in the subdirectory mkmodel (simply enter this directory and execute grid3dtg), and that appropriate damping and smoothing factors can be chosen with the aid of the program misfitt. Simply enter the subdirectory analysis and execute misfitt to obtain the variance and roughness of the current model.

10.2 Example2

The second example provided with this distribution (see subdirectory example2) is a simple synthetic test involving a relatively small number of sources. It has been included mainly to demonstrate how data from multiple arrays can be simultaneously inverted. In this case, there are two receiver arrays, each comprising 49 instruments, which have been deployed at different times. Correspondingly, each have detected a separate set of events. Therefore, the first line in both sources.dat and receivers.dat is "2", indicating that two arrays are present. In sources.dat, there are two consecutive lists of event locations and phase types, and in receivers.dat, there are two lists of receiver locations. Note that in theory, one could in fact exploit the flexible switching options allowed for in otimes.dat to have only one array present, by "switching off" half of the receivers for the first set of sources, and then reversing the switches for the second set of sources. However, it turns out that this is computationally much less efficient than specifying separate receiver arrays.

As in the previous example, the binary files ak135.hed and ak135.tbl must be copied to the current working directory (they should be located in source/aktimes after compilation) in order to run tomo3dt. Due to the relatively small number of sources, this example should execute in around 90 s or less on a relatively recent PC. The output residuals.dat file should look something like:

```
151.91 0.02310
63.89 0.00409
53.88 0.00291
50.83 0.00259
49.90 0.00249
49.13 0.00242
```

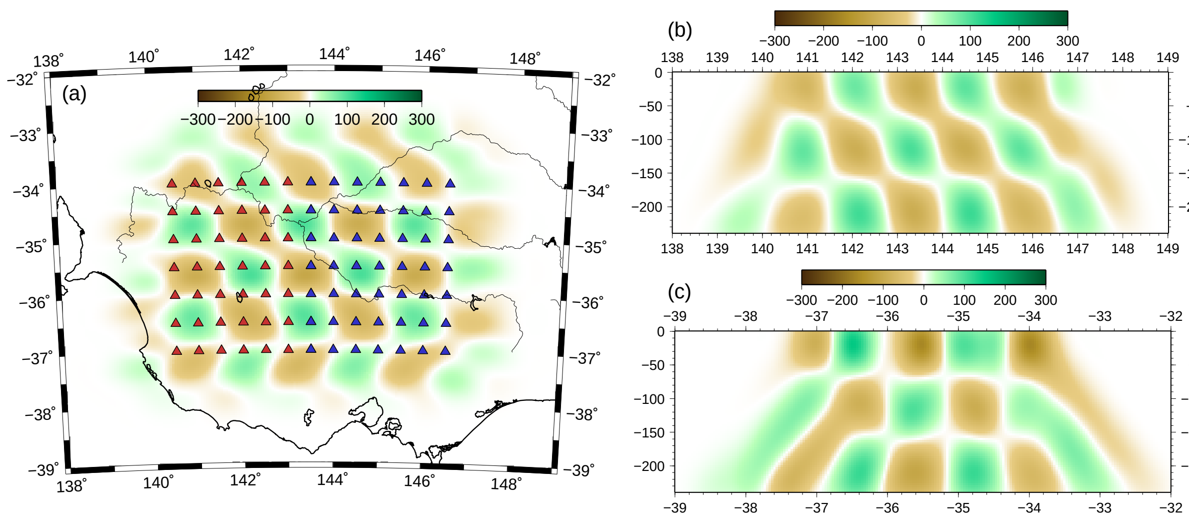

Figure 3: Example output from gmtslicet using example2 provided with the distribution. (a) Depth slice (red and blue triangles denote receivers from different arrays); (b) E-W slice; (c) N-S slice.

48.81 0.00238

Output from the three GMT scripts plotd, plotew and plotns contained in gmtplot are shown in Figure 3 (see previous example for more details). The recovered checkerboard pattern does not exhibit any sign of a "suture" between the two adjacent arrays and quite accurately replicates the true structure. This is because the velocity field contains no long wavelength features that might otherwise have been filtered out of the image. For instance, if the true model included a constant velocity gradient in the E-W direction, then it is likely that this feature would not be properly recovered, unless it was included as *a priori* information in the initial model. It is important to consider the limitations of using relative arrival time residuals when data from multiple arrays are simultaneously inverted.

The initial and synthetic models for this example were created using grid3dtg with the input file located in the subdirectory mkmodel. The initial model was constructed by setting the switch at the start of the "Optionally apply checkerboard" section (line 31) to zero. The synthetic model was created by setting this switch to 1. In order to construct the synthetic dataset contained in otimes.dat, the first step was to run fm3dt with line 7 of fm3dt.in set to gridt.vtx, where gridt.vtx is the renamed checkerboard file produced by grid3dtg. The rtravel.out file produced by fm3dt should then be copied to the subdirectory mkdata, and renamed observed.dat. The next step is to run fm3dt with line 7 of fm3dt.in set to gridi.vtx, where gridi.vtx is the renamed background file produced by grid3dtg. The traveltime output file rtravel.out should in this case be copied to rtimes.dat. Execution of syntht in the subdirectory mkdata will then produce otimes.dat, which can be copied to the main working directory. Note that the two traveltime initialization programs aktsurf and itimes need only be run once in sequence prior to this whole process (they only need to be rerun if the propagation grid geometry or source distribution changes).

11 FAQs

1. **Q.** *I would like more details on how the code can be used in practice.*

   A. In the directory docs, a complete copy of a paper that uses the code is provided. seaustralia.pdf is a paper published in AJES which describes a teleseismic tomography experiment in south east Australia. The solution model generated in example1 provided with this distribution should be approximately the same as the SEAL solution model in the AJES paper.

2. **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 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, the computing time will increase by a factor of at least 8. In general, I find that something like half a million nodes is usually adequate, but of coarse this will vary depending on the problem you are dealing with. My approach is to use a coarse dicing factor (like 1 or 2) 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.

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.** *Can teleseismic wavefronts pass through the sides of the 3-D model region?*
   A. No. As it currently stands, teleseismic wavefronts are computed to the base of the 3-D model only before FMM is applied. This means that you must be careful to make sure that the horizontal bounds of the velocity grid are sufficiently large so that wavefronts impinging from various angles will pass through the base of the model before hitting any of the stations. This can be checked by using the plotting routines that visualize the raypaths.

5. **Q.** *Will new features be added to the code in future?*
   A. I hope to fix any obvious bugs, but beyond this, I do not anticipate changing anything significantly.