Geol 483.3

Lab Project #1: Cross-well travel-time modelling and tomography

This project consists of two parts. First, you will set up a synthetic cross-well model and use Matlab or Octave to compute the first-arrival travel times in it. After that, you will use these synthetic travel times to invert for seismic velocities in a procedure called seismic velocity tomography. For both of these tasks, you will use matrix operations, which are very convenient in Matlab/Octave.

To aid with your work, I provide a "ray tracing" program called `trace_ray.m`, program `test_model.m` which you can use as a seed for starting your work, and three auxiliary subroutines. These codes are provided in the following zipped archive. Download this file and unpack the codes in your work directory.

Look into code `test_model.m` and try executing it in Matlab or Octave. I only tested this code in Octave but hope it should run in Matlab as well. If you still find some problems and have to make corrections, please provide a copy of the corrected codes with this lab report.

Code `test_model.m` creates and plots a 10×10 grid of 20×20-m velocity blocks, which will be the study area of this lab. Thus, you will be modelling a square section of 200×200 m size.

One straight ray with arbitrary source and receiver positions (they can be anywhere inside or outside the model area) is also created and plotted in green. The `trace_ray()` function determines which model blocks are intersected by this ray and returns parameters useful for plotting and constructing the forward and inverse problems. In particular, it determines the entry and exit points and the lengths of ray segment in each model cell. These entry and exit points are shown by blue circles and red pluses in the plot.

The approach to forward modeling and inversion in seismic tomography is based on:

1) Representing the velocity structure by a single vector (column) of "slownesses" $s_k = 1/V_k$. In the numbering convention of the code `trace_ray()`, model cells are counted in a horizontal scanline order – first all cells along the top row, then cells in the second row, etc. The total number of model parameters will be denoted M in equations below.
2) Similarly, representing all travel times (modeled or measured) $\{t_j\}$ by a single "data" vector. This vector was denoted **d** in the lectures, but here, we denote it **t** for clearer association with travel times. The number of travel-time points will be denoted N below.
3) Representing the relation between model velocities (slowness) and travel times as a matrix multiplication. With a given some velocity (slowness) distribution the travel time at any source-receiver pair i equals:

$$t_i = \sum_{k=1}^{M} L_{ik} s_k \qquad (1)$$

where k is the index of model cell, L_{ik} is the ray path of i-th ray within this cell. In matrix form, this relation for all source-receiver pairs together is

$$\begin{pmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{pmatrix} = \mathbf{L} \begin{pmatrix} s_1 \\ s_2 \\ \dots \\ s_M \end{pmatrix}, \qquad \text{or simply} \qquad \mathbf{t} = \mathbf{L}\mathbf{s}, \tag{2}$$

where **L** is a matrix with columns corresponding to model cells, and rows corresponding to each travel-time measurement.

Assignments

1) In a copy of `test_model.m` (rename it as you like), **initialize the slowness array** of `model.ncells` elements (this variable is denoted *N* above and equals 100, but better still use the name). Initially, set the velocities equal 2 m/ms everywhere, i.e. slownesses equal 0.5 ms/m (use meters to measure distances and milliseconds for times).

When coding, try using functions to perform localized and repetitive tasks, not simply replicating the code. Use functions `distance()`, `checkerboard()`, `grid_block()`, `trace_ray()` that I have supplied, and maybe add others.

For transforming model grids into vectors for inversion and back, I have added functions `grid2vect()` and `vect2grid()`.

2) **Plot the model** using the provided function `plot_model()`. Note that if the second parameter (slowness vector) is given in the call to this function, it will plot the corresponding velocities within the model by color.

If you are good at Matlab, you could try exploring better options for color palette used in function `colorbar` in `plot_model.m`.

3) **Implement a of cross-well recording geometry.** To do this, in the code, create a matrix with source positions and another matrix for receiver positions. In these matrices (as in other parts of this code), the first column should contain coordinates *x* and the second column – coordinates *z* (depth), and rows correspond to different sources (or receivers).

   Let us assume that the left edge of the model area is the borehole containing 11 sources within the depth range from 70 to 170 m. The corresponding matrix of source coordinates can be easily filled by using function `linspace()`. Similarly place 21 receivers along the right edge, at depths 0–200 m.

Do not "hardwire" constants like 11 and 21 into your code. Instead, define variables for these constants and use them. In case you do not like the results and want to explore them better, you might want top change the number of sources, receivers, ranges of their depths, etc.

4) **Trace and plot all rays** similarly to what is shown by green line in `test_model.m`. Overlay the traces on the plot of velocity model and grid. See whether the ray geometry is as intended.

   In the loop modelling and plotting rays, include code for accumulating the matrix **L** as shown in the example.

5) **Verify that matrix L is correct.** The correctness can be checked by the following condition for each travel-rime point number *i*:

$$\sum_{k=1}^{M} L_{ik} = \text{total\_length\_of\_ray\_}i\ .$$

Evaluate this sum from matrix **L** (by using the Matlab/Octave function `sum()`) and compare to the source-receiver distance determined directly from the endpoints. This distance can be calculated by function `distance()` included in my code.

6) Plot matrix **L** in a separate figure (`figure(2)`) using function `imagesc()`. Add axes labels and `colorbar()`. Explain the banded pattern of the matrix and its general appearance (many zeros, variable values of nonzero values in different areas, shapes of patterns representing rays, range of values).

In the following steps, you will perform forward travel-time modelling:

7) **Using a single matrix multiplication** by equation (2) above, **compute the travel times for each ray**. For source positions at the top, middle, and bottom of the source spread, **plot the travel times as functions of receiver depth**. Comment on the shapes of the curves. Are the smallest values and moveouts as expected? Where do the minima of the travel times occur?

8) Now apply a +10% velocity anomaly within a 40×40-m area (four model cells) somewhere in the middle of the model and repeat the calculation of travel times. Add these travel times to the plot in the preceding step. How are the times different? Why? Can you locate the velocity anomaly from these plots and state whether it is positive or negative?

9) **Save** the structure `model`, slowness, rays, matrix **L**, and modeled travel times by using '`save`'.

10) Create and save another travel-time dataset, with velocity anomaly shifted to the edge of ray coverage.

Next, try tomographic inversion. Note that matrix **L** is not invertible for two reasons: a) there are more equations than unknowns, and b) (which is more important!) some of its columns are identically zero. These columns correspond to model cells not crossed by rays. Therefore, these cells cannot be inverted in principle. To resolve this problem, we need to provide additional equations constraining these cells without rays. Let us use the simple "regularization" or "whitening" approach discussed in the lecture.

Start from constructing the Least Squares sense by multiplying equation (2) by the transposed matrix $\mathbf{L}^{\mathbf{T}}$ on the left:

$$\mathbf{L}^{T}\begin{pmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{pmatrix} = \mathbf{L}^{\mathbf{T}}\mathbf{L}\begin{pmatrix} s_1 \\ s_2 \\ \dots \\ s_M \end{pmatrix},$$

which gives the Least Squares solution:

$$\begin{pmatrix} s_1 \\ s_2 \\ \dots \\ s_M \end{pmatrix} = \left(\mathbf{L}^T\mathbf{L}\right)^{-1}\mathbf{L}^T \begin{pmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{pmatrix}. \quad (3)$$

This matrix equation is readily programmable in Matlab or Octave. As mentioned above, the inverse of $\mathbf{L}^T\mathbf{L}$ still does not exist because of some cells not covered with rays calculated. We can "regularize" this solution by adding a small diagonal matrix to $\mathbf{L}^T\mathbf{L}$ before inverting it:

$$\begin{pmatrix} s_1 \\ s_2 \\ \dots \\ s_M \end{pmatrix} = \left(\mathbf{L}^T\mathbf{L} + \varepsilon\mathbf{I}\right)^{-1}\mathbf{L}^T \begin{pmatrix} t_1 \\ t_2 \\ \dots \\ t_N \end{pmatrix}. \quad (4)$$

where $\mathbf{I}$ is the identity matrix. Parameter ε can be selected based on the norm of matrix $\mathbf{L}^T\mathbf{L}$:

$$\varepsilon = \lambda \cdot norm\left(\mathbf{L}^T\mathbf{L}\right),$$

where λ is another small dimensionless parameter. Try several values for it, such as 0.0001, 0.001, and 0.01.

For each of the two travel-time datasets you created, do the following:

11) **Implement matrix inversion** (4) for both models you saved. Plot the inverted velocity cross-sections.

12) **Compare the inverted models to those used as inputs for travel-time modeling**. Discuss the difference in the shapes of the anomalies recovered by the inversion in the two cases.

Look at the resulting images carefully, and also carefully think about what you expect to see in the images. Do the images match your expectations? Which anomaly is recovered better, at the center or corner of the model? What could be the reason for this?

You will see that the inversion is (obviously) unable to constrain the slowness variation in areas not covered by rays. Also, because of predominantly horizontal rays directed left to right, the velocities are strongly "smeared" horizontally. In which part of the model velocities are overestimated, and in which they are underestimated by the inverse? Can you explain this trend?

Also, note that we used the simple regularization scheme in eq. (4). Would you think that smoothness constraints might be better for this case? Does the model look "rough" at the edges of ray coverage?

Finally, try the checkerboard resolution test:

13) In a new copy of the code, **generate another synthetic model with positive and negative** ±10% anomalies alternating in a checkerboard pattern. The anomalies can consist of a single cell, 2×2, and 3×3 cell blocks.

To create checkerboard patterns, I provide code `checkerboard.m`. To obtain, for example a 2×2 pattern of slowness variation `ds` ms/m, you can write `p = ds*checkerboard(model,2,2)`. This will give you a matrix which you can plot using `imagesc`. To transform this matrix into a model vector for inversion, use function `grid2vect(p)`.

14) **Simulate travel times** through the model using matrix equations (2) and **invert them as above**.

The resulting inverted model should correspond to the original checkerboard model. Are all the anomalies equally well recovered? Why? How does this change if you increase the sizes of checkerboard anomalies?

Hand in:

Codes, report in a Word file, screen captures or PostScript/PDF plots, by email.