Evaluation of the seismic tomography tool FMTOMO for high performance computing applications

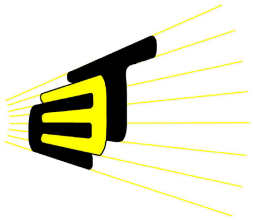
Juan S. López G.* and Sergio A. Abreo C. †

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA
Y DE TELECOMUNICACIONES

Bucaramanga, 2014

*Universidad Industrial de Santander, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Connectivity and Signal Processing Group, Bucaramanga, Colombia. Project Developer. Email:juan.lopez@correo.uis.edu.co

†Universidad Industrial de Santander, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Connectivity and Signal Processing Group, Bucaramanga, Colombia. Project Advisor. Email:sergio.abreo@correo.uis.edu.co



Evaluación de la herramienta de tomografía sísmica FMTOMO para aplicaciones de alto desempeño

Juan Sebastián López Guerra

Trabajo de grado para optar al titulo de Ingeniero Electrónico

Director
PhD.(c) Sergio Alberto Abreo Carrillo

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

Bucaramanga, 2014

Contents

List of Figures

List of Tables

RESUMEN

TÍTULO: EVALUACIÓN DE LA HERRAMIENTA DE TOMOGRAFÍA SÍSMICA FMTOMO PARA APLICACIONES DE ALTO DESEMPEÑO. *

AUTORES: Juan Sebastián López Guerra. **

PALABRAS CLAVES: Tomografía sísmica, método de marcha rápida, trazado de rayos, computación de alto desempeño.

DESCRIPCIÓN:

Colombia siempre ha sido un país reconocido por su riqueza en cuestión de recursos naturales: vegetales, animales, hídricos y minerales. Precisamente estos últimos se han tornado de gran interés en una época donde el petróleo hace girar al mundo. Debido a este interés en la explotación petrolera, las zonas de fácil acceso al petróleo comienzan a acabarse y por esta razón, los esfuerzos se están enfocando a la exploración de zonas de difícil acceso. Para la exploración petrolera en estas zonas se buscan alternativas que den buenos resultados sin ser demasiado costosas. Una de estas técnicas es la tomografía sísmica, que permite la obtención de modelos del subsuelo a partir de adquisiciones en la superficie.

Diversas herramientas para la realización de tomografía sísmica han sido desarrolladas, una de ellas es FMTOMO, que es una herramienta que resuelve la tomografía sísmica como un problema inverso que se plantea como un problema de optimización a partir de una aproximación lineal de la ecuación eikonal.

El principal enfoque del trabajo es evaluar la herramienta en términos de aplicación y en términos de desempeño computacionales.

---

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones. Director: PhD. (c) Sergio Alberto Abreo Carrillo.

ABSTRACT

TITLE: EVALUATION OF THE SEISMIC TOMOGRAPHY TOOL FMTOMO FOR HIGH PERFORMANCE COMPUTING APPLICATIONS.*

AUTHOR: Juan Sebastian López Guerra. **

KEY WORDS: Seismic tomography, fast-marching method, ray tracing, high performance computing.

DESCRIPTION:

Colombia has always been a well-known country in terms of natural resources such as plant, animal, water and mineral resources. It is precisely the mineral resources which became of high interest due to the fact that it is oil the fuel that keeps the world going around. Because of that interest in oil exploitation, oil in easy-access-zones is running low and for that reason, efforts are shifting and focusing towards the exploration of hard-access zones. In order to explore these zones looking for hydrocarbons, economic but reliable alternative techniques are required. One of these techniques is seismic tomography, which allows to obtain sub surface's velocity models from seismic acquisitions performed on the surface of the ground.

There have been several tools developed for performing seismic tomography, one of them is FMTOMO, which is a tool that solves seismic tomography as an inverse problem, dealing with it as an optimization problem proposed from a linear approximation of the eikonal equation.

The main focus of this work is to evaluate the tool in terms of application and computing performance.

---

* Bachelor Thesis

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones. Director: PhD. (c) Sergio Alberto Abreo Carrillo.

Evaluation of the seismic tomography tool FMTOMO for high performance computing applications

Juan S. López G.[*] *and Sergio A. Abreo C.* [†]

INTRODUCTION

In Colombia, a well-known country for its natural resources, exists a high interest of exploiting fossil fuels given its wide variety of terrains present in its lithosphere. However, that same variety makes it very expensive to explore some difficult-access-zones such as off-shore zones [1].

Seismic tomography shows up as an alternative for exploration as this technique helps obtaining an approximate model of the ground's subsurface at a lower cost than it would cost taking all the machinery necessary to explore and being unable to find any material of interest. This technique consist in using seismic sources (such as detonating charges on the ground's surface) to propagate energy through a media while locating seismic receivers (geophones for example) to sense disturbances reflected from the ground's subsurface and using the recorded data to estimate a possible ground model that could reproduce the observed data by propagating energy through it[2].

The geophysicist's community [3] has developed several useful tools to perform seismic tomography, one of those tools is FMTOMO, developed by PhD Nicholas Rawlinson at the Research School of Earth Sciences at ANU (Australian National University)[4] .

FMTOMO[5] is an open-source software distribution that deals with the forward problem of travel-time prediction using a multi-stage Fast Marching Method (FMM) that solves frontier problems of the Eikonal equation and the inverse problem, which consist in adjusting model parameters to satisfy observed data given any imposed regularization.

The purpose of this work is acquire knowledge in terms of not only the use of one particular tool (FMTOMO), but also seismic tomography in general. With those purposes in mind, the distribution of this paper has been sorted as follows: the first section is a brief introduction to the topic; the next section contains helpful tips for the reader to install FMTOMO (not a detailed step-by-step guide, but nevertheless is worth reading for recently-started users of FMTOMO); following that, the next three sections explain in a more detailed way the three main steps of FMTOMO: modelling, solving the forward problem and solving the inverse problem, with the latter also containing a brief summary of how to set up the inverse problem as an optimization problem; after that, the focus shifts towards high performance computing and how to improve computational performance by using parallel computation; then follows the results and discussion sections presenting and explaining some of the results obtained in each step of the process (modelling Earth models,the forward problem, some inverse problems, and times decreased by parallelism); after that, the closing remarks and conclusions about seismic tomography, FMTOMO and future work towards parallel computing; finally, section 9 contains a list of "must-read" references to better understand inverse problems, seismic tomography as an inverse problem, using FMTOMO and performing computational task using parallel computing.

[*]Universidad Industrial de Santander, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Connectivity and Signal Processing Group, Bucaramanga, Colombia. Project Developer. Email:juan.lopez@correo.uis.edu.co

[†]Universidad Industrial de Santander, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Connectivity and Signal Processing Group, Bucaramanga, Colombia. Project Advisor. Email:sergio.abreo@correo.uis.edu.co

INSTALLING FMTOMO

Fortran Compiler

Since all sources in FMTOMO are written in fortran, it is necessary to have an appropriate compiler. There are a couple of them that come by default in linux (such as f95 or gfortran), however those compilers have proven unable to generate the binaries correctly. To avoid problems during the compilation process, it is strongly recommended to use the Intel Fortran Compiler, ifort to generate the binaries.

The Intel Fortran Compiler can be obtained from their official page: http://software.intel.com/en-us/intel-compilers

Compiling FMTOMO

FMTOMO comes with an executable file called *"compileall"*, and as its name suggests it compiles all sources and saves the binaries inside the /bin folder.

The recommendation is to read the installation instruction manual [6] in order to make sure the fortran compiler is set to ifort and the ksh (or zsh) directive points to the right location. After doing that, *"compileall"* can be executed and the /bin folder can be added to the $Path environment variable.

After compiling, there will be ten binaries in the /bin folder, which will be discussed along this paper.

CREATING 3D MODELS USING FMTOMO

Subsurface models in FMTOMO are composed by three elements: interfaces at different depths, velocity layers between interfaces and the computational grid for rays to propagate through it. In addition to the subsurface model, there are two arrays that represent seismic sources and receivers.

The subsurface model and the arrays of sources and receivers allow to simulate an acquisition. However, before getting into

this process it is important to explain some details related to the implementation.

Three-dimensional subsurface models

Using grid3dg

In order to create three-dimensional models of the subsurface of the Earth, FMTOMO comes with a handy tool named *grid3dg*[6] which allows the user to construct a model composed of a propagation grid for rays to be traced (propgrid.in); a velocity grid, made layer by layer(vgrids.in); and an interface grid, constructed interface by interface (interfaces.in).

Grid3dg.in is the only one necessary input file to run correctly *grid3dg*.

Figure 1 shows the input-output diagram describing the process above.

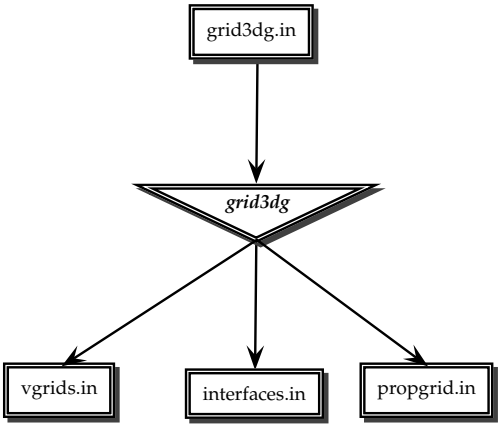


Figure 1: I/O process diagram of *grid3dg*

.

Modifying interfaces.in

As useful as *grid3dg* is, there are things that are impossible to obtain by just executing the program. There are several limitations related to the model's interfaces that need to be addressed by hand in order to overcome them. Those limitations are:

- All interfaces must completely span the horizontal dimensions of the model region difficulting the description of pinch-outs and seismic faults.
- Interfaces cannot be multi-valued in depth, making it hard to add structures like salt-domes to the models.
- Node distribution for all interfaces must be uniform. If any zone needs several nodes to be described, then the whole model must have that node distribution even if it can be described with less nodes.

Even though the difficulties above mentioned would make the reader think that model construction in FMTOMO is very limited, some of those issues can be handled by executing *grid3dg* and modifying by hand the output files to match the original model. This is done by editing the file interfaces.in.

There are some cautions the user should take into account to obtain a model correctly. One of them is a cushion of boundary nodes that are automatically located in the model so, there will be distribution of $m+2$ nodes in the longitudinal component and $n+2$ nodes in the latitudinal component, being m and n the nodes distribution originally set for each component respectively.

With these recommendations in mind, the user can develop models with a higher complexity level than those made by simply executing *grid3dg*.

Source-Receiver arrays

The next step is to set up an array of seismic sources and another array for receivers to simulate an acquisition.

The array of receivers is generated by simply executing *arraygen*[6] after editing the file arraygen.in and its output file is arraygen.out. This process is shown in figure 2.

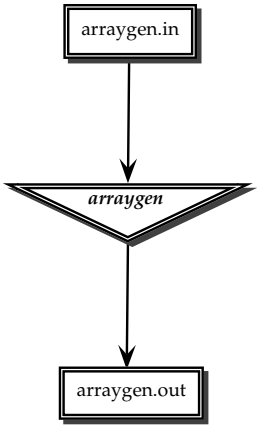


Figure 2: I/O process diagram of *arraygen*

.

The next step is to obtain an array of seismic sources. There are two options available: the programs *moddata* and *obsdata*. The difference is that the first one serves to do synthetic tests, while the latter is used when dealing with observed data, hence its name. As the sample model and the acquisition are synthetic, the focus of this paper is on *moddata*[6].

Moddata needs three input files: moddata.in, arraygen.out and "fuentes.in". The reader can notice that arraygen.out is an out-

put file resulting from the execution of *arraygen* (so it is necessary to obtain that file prior to attempting to execute *moddata*); the file moddata.in is the set up file for the process in which the user defines all parameters and conditions; the final file "fuentes.in" contains the number and location of all sources in latitude, longitude and depth with an a priori uncertainty associated to each coordinate.

Figure 3 shows both input and output files necessary to execute *moddata*.

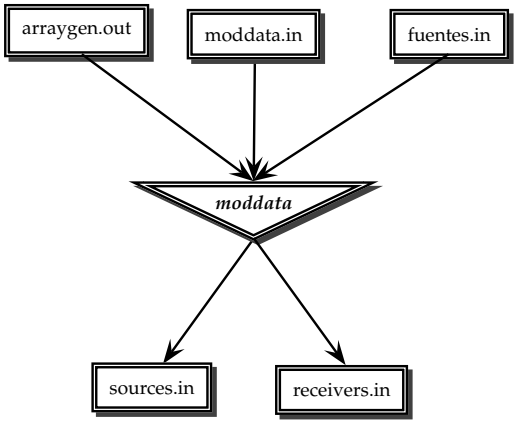


Figure 3: I/O process diagram of *moddata*

.

The format of the "fuentes.in" file is shown in listing 1:

```
# of sources
lat    lon    dep    unty-lat unty-lon unty-dep
```

Listing 1: Sources File "fuentes.in" Format

Listing 2 is an example of a few sources:

```
7
-40.2500    140.2500    0.00    2.0    2.0    2.0
-40.5000    140.2500    0.00    2.0    2.0    2.0
-40.7500    140.2500    0.00    2.0    2.0    2.0
-41.0000    140.2500    0.00    2.0    2.0    2.0
-41.2500    140.2500    0.00    2.0    2.0    2.0
-41.5000    140.2500    0.00    2.0    2.0    2.0
-41.7500    140.2500    0.00    2.0    2.0    2.0
```

Listing 2: Sources file "fuentes.in" Example

The last three columns of listing 1 are only necessary if the sources are going to be relocated —this happens with seismological sources (earthquakes), rather than with seismic sources (detonated charges, usually)—.

Before proceeding to the output files, there is an important concept that the reader should understand: "Path Signature".

The path signature describes the propagation of the wave-front among interacting interfaces starting from a source and finishing in a receiver. That means that in order to track travel-times from each source, there has to be a path starting from a source and ending in a receiver.

Figure 4 illustrate how path signatures are defined. For further explanation on "Path signatures" read Ch. 7.1 of the FMTOMO's instruction manual.[6]

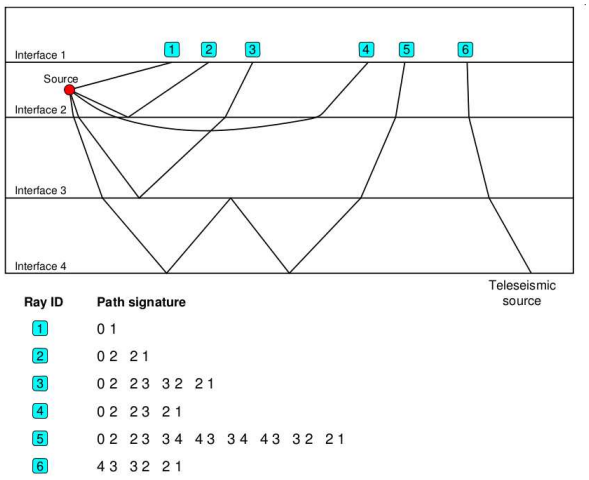


Figure 4: Signature Paths. Taken from the FMTOMO Instruction Manual [6]

As figure 4 shows, each path is composed of path segments which are formed by pairs of numbers for each segment. Those numbers represent the start and end points of each segment. All paths start from a source (number 0) and the next number is the interface reached by the ray. The following couple of numbers starts in the last interface and continues to the next. This concept is important because in that way, paths are going to be tracked down by editing moddata.in.

With that concept in mind, the editing of moddata.in can be done. The reader can refer to the FMTOMO Instruction Manual for more details. The philosophy of the file consist in defining:

- The names of input and output files
- The number of sources files —there can be multiple "fuentes.in" (with different names off course)—.
- The name of the file
- The type of source (local or teleseismic)
- Thethe number of paths (remember path signatures?)
- The number of segments for that first path of the first source file
- The path signature and each segment's velocity field (P or S).

Listing 3 is an example of how to add path signatures and segments to a sources' file:

```
fuentes1.in c: Input source file
0  c: Local (0) or teleseismic sources (1)
0  c: Compute source derivatives (0=no, 1=yes)
5  c: Number of paths from these sources
2  c: Number of path segments for path 1
c: Path seq info plus velocity fields below
0 2  2 1
1    1
4  c: Number of path segments for path 2
c: Path seq info plus velocity fields below
0 2  2 3  3 2  2 1
1    1    1    1
6  c: Number of path segments for path 3
c: Path seq info plus velocity fields below
0 2  2 3  3 4  4 3  3 2  2 1
1    1    1    1    1    1
8  c: Number of path segments for path 4
c: Path seq info plus velocity fields below
0 2  2 3  3 4  4 5  5 4  4 3  3 2  2 1
1    1    1    1    1    1    1    1
10 c: Number of path segments for path 5
c: Path seq info plus velocity fields below
0 2  2 3  3 4  4 5  5 6  6 5  5 4  4 3  3 2 2 1
1    1    1    1    1    1    1    1    1   1
```

Listing 3: Example of moddata.in

Once the editing of moddata.in is done, it is time to execute *moddata*. The output file are sources.in and receivers.in.

That concludes the creation of the seismic model and the arrays of sources and receivers.

FORWARD PROBLEM

In the previous section, a seismic model was defined with the purpose of simulating a seismic acquisition through it.

The next step is to set up the acquisition to obtain what can be called "observed travel-times" from rays propagating through the model. In order to do that, FMTOMO comes with a multi-stage fast-marching method for travel-time prediction, *fm3d* [6]. But before going into that, it is necessary to point out the execution sequence of the processes.

It is needed to define a subset model for inversion, after which the ray tracing can be done and, a synthetic dataset is obtained.

Defining a subset model for inversion

Even though this section corresponds to the forward problem, some clarifications about the inverse problem need to be done.

FMTOMO allows to invert for three variables: source location, velocity model and interface model; and we can set up the inversion program to invert for any combination of them. However, in some cases it is necessary to run an inversion for a subset of the sources, or interfaces, or layers (representing velocity). For example, as we deal with computational models of the sub-surfaces, an interface inversion should not include neither the top layer nor the bottom layers of the model because they are used only to define the boundaries of the model. In that case, one would want to invert for all interfaces except those two. For such a case, and prior to continuing, a program named *frechgen*[6] must be executed. As always, there are input/output files as shown in figure 5.

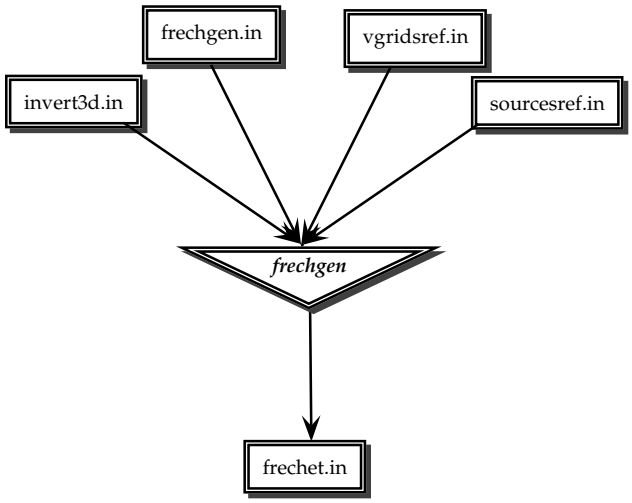


Figure 5: I/O process diagram of *frechgen*

.

There are four input files, two of them are obtained by simply copying the original, sources.in and vgrids.in and changing their names to: sourcesref.in and vgridsref.in. The other two, have to be edited by hand. One would ask why the file invert3d.in has to be provided? The answer is as simple as because one should know the type of inversion intended to do and, according to that, the acquisition that has to be done, so the recommendation is to think ahead.

Invert3d.in defines the type of inversion (source's location, layer's velocity and/or interfaces' structure). This file will be discussed in its own section.

The only file left is frechgen.in and its format is easy to understand, it just asks the names of the four files mentioned before and asks if the inversion will include all data or a subset of it. The example input files and the FMTOMO instruction manual give enough information to understand how it works.

Having defined the four input files, the execution of *frechgen* leads to the frechet.in file. The importance of doing this process first is that frechet.in is an input file for doing the ray tracing with *fm3d* defining the subset of the model that will be

inverted for.

Ray tracing

So far, there have been a lot of thing to do prior to doing the ray tracing by executing *fm3d*. But now, it is time to obtain those travel-times. The process here, is similar to previous ones. Figure 6 shows the input and output files. Almost all input files are already created and all set to run the ray tracing. We have the model files (vgrids.in, interfaces.in and propgrid.in), the instrumentation files (sources.in and receivers.in) and the subset definer file (frechet.in) —this one is so important because by defining the subset of the model to invert for, we also define which Fréchet derivatives are going to be computed—. The only new file here is mode_set.in (which is only a set of switches defined by true (T) or false (F)).

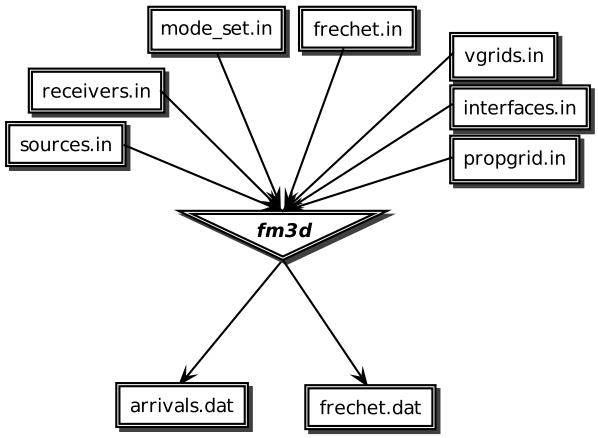


Figure 6: I/O process diagram of *fm3d*

.

The output files are two, arrivals.dat and frechet.dat respectively. The first containing the travel-times and the latter, the computed Fréchet derivatives.

With the acquisition done, it could be said that we have already our "observed travel-times"(or otimes.dat, this is the name given in FMTOMO), however there is an optional process that is worth mentioning to create synthetic datasets.

Synthetic dataset

Since any observational dataset contains noise in real-life, it is a common practice to add random noise to synthetic datasets. And, to do that, FMTOMO comes with a program called *synthdata*[6].

Figure 7 shows two input files and one output file. The first input file (syntimes.dat) is obtained by copying arrivals.dat and changing its name; the second input file must be edited by hand. This is an easy process in which can be added a data covariance without adding noise, it can be added an offset to the travel-times and finally random noise defining a standard deviation.

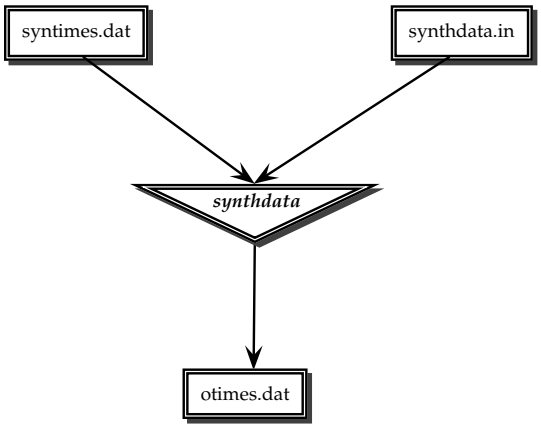


Figure 7: I/O process diagram of *synthdata*

.

The result is a file named otimes.dat containing the observed travel-times, necessary for the inversion process.

INVERSE PROBLEM

"The inverse problem involves adjusting the model parameter values (i.e. interface depth, velocity, hypocenter location) in order to try and satisfy the data observations (source- receiver travel-times), subject to any imposed regularization."[6]

The process prior to obtaining the otimes.dat file has been long; from modelling the subsurface, to propagating rays, to generating a synthetic dataset in order to perform an inversion, which consists in:

1. Constructing an initial model M_0 (interfaces.in, vgrids.in and propgrid.in).
2. Propagating rays through the initial model to obtain model's travel-times (mtimes.dat).
3. Adjusting the model parameter values (M_1).
4. Calculating time residuals.
5. Repeating steps 1. through 4. n iteration times.

There are a few concepts here that are worth explaining. For instance, the initial model M_0 is NOT the original model created before, it is a model created to possess characteristics similar to the original model or, as in the case of real seismic data, characteristics that are believed the original model has. That model is going to be modified with each iteration becoming $M_1, M_2, \ldots, M_i, M_{i+1}, \ldots M_n$. Being M_n the final model after n iterations.

As each iteration changes the model, each model's travel-times will also differ. For that reason the file arrivals.dat is renamed mtimes.dat (M_i travel-times) as it will be compared to otimes.dat (observed travel-times).

To adjust the model parameter values an inverse problem has to be solved; those problems often involve a minimization of an objective function.

According to Aki's model[7], when a seismic source transmits its energy through a media, that wave-front propagation can be represented as rays traveling from that source to receivers located on the surface of the ground.

Figure 8 shows a ray traveling from a seismic source, through some cells of a media —each cell with its own velocity— to a receiver. The time a ray takes to get to the receiver can be obtained from equation 1, rewritten as equation 2.

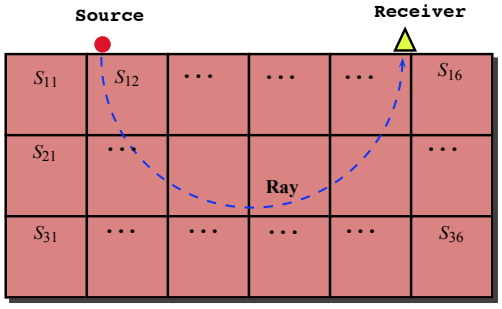


Figure 8: Aki's Model

.

$$v = x/t \tag{1}$$

$$t = x/v \tag{2}$$

Where,

t: time required for the ray to travel from source to receiver

x: distance travelled by the ray

v: velocity of the media

To avoid working with the inverse of the velocity, equation 3 defines a concept known as *slowness* (read: inverse of the velocity).

$$t = x * s \tag{3}$$

Now, in vectorial or matrix form, equation 3 turns into equation 4

$$\vec{d} = \mathbb{G} * \vec{m} \tag{4}$$

Where,

$\vec{d}$: represents the observed data vector (in this case, the times all rays took to travel to each receiver).

$\mathbb{G}$: represents the matrix containing the distances each ray travelled through each cell of the media.

$\vec{m}$: represents the slowness vector

Equation 4 is often written as equation 5 in which the matrix $\mathbb{G}$ usually contains the physics involved in the phenomenon, relating the observed data/times ($\vec{d}$) with the slowness model ($\vec{m}$)

$$\mathbb{G}\vec{m} = \vec{d} \tag{5}$$

Aki's aproach presented seismic tomography as a parameter estimation problem in which the following assumptions are commonly used:

1. The model physics ($\mathbb{G}$) is known.
2. A unique true model ($\vec{m}_{true}$) exists.
3. A true dataset ($\vec{d}_{true}$) exists.
4. Observations ($\vec{d}$) are composed by a true dataset plus noise present while acquiring data ($\vec{\eta}$).

Assumption 4. can be written as equation 6.

$$\vec{d} = \vec{d}_{true} + \vec{\eta} \tag{6}$$

Replacing equation 6 into equation 5, is obtained equation 7.

$$\mathbb{G}\vec{m} = \vec{d}_{true} + \vec{\eta}$$

$$\mathbb{G}\vec{m}_{true} + \vec{\eta} = \vec{d}_{true} + \vec{\eta} \tag{7}$$

If we can write the noise effect on the model as equation 8, equation 7 can be rewritten as equation 9.

$$\mathbb{G}(\vec{\delta}) = \vec{\eta} \tag{8}$$

$$\mathbb{G}(\vec{m}_{true} + \vec{\delta}) = \vec{d}_{true} + \vec{\eta} \tag{9}$$

The problems in equation 9 are two:

1. $\vec{m}_{true}$ and $\vec{\delta}$ are unknown
2. $\vec{d}_{true} + \vec{\eta}$ are known, but they are mixed.

A possible solution to the problem can be obtained through equation 10, in which $\tilde{\mathbf{m}}$ represents an estimator of $\vec{m}$ and the calculated data vector $\tilde{\mathbf{d}}$ is the produced by solving the forward problem with $\tilde{\mathbf{m}}$ (Eq. 11).

$$\tilde{\mathbf{m}} = Arg_m Min||\mathbb{G}\vec{m} - \vec{d}||_2^2 \tag{10}$$

$$\tilde{\mathbf{d}} = \mathbb{G}\tilde{\mathbf{m}} \tag{11}$$

As $\tilde{\mathbf{m}}$ serves as estimator of $\vec{m}$, the calculated data vector $\tilde{\mathbf{d}}$ will differ from the observed data vector $\vec{d}$ (from now on $\vec{d}_{obs}$), and that difference will result as an estimation error ($\vec{E}$) or residual.

$$\vec{E} = \mathbb{G}\tilde{\mathbf{m}} - \vec{d}_{obs} \tag{12}$$

Using the least squares minimization[8] to find the best fit model and assuming that data errors are independent and normally distributed, a least squares solution can be obtained by minimizing the prediction error of the solution (each residual inversely weighted by the standard deviation of the corresponding error distribution).

$$\Psi(\vec{m}) = \sum_{i=1}^{n} \left(\frac{\vec{E}}{\sigma_i} \right)^2 = \vec{E}^T \mathbb{C}_d^{-1} \vec{E} \tag{13}$$

Replacing equation 12 into 13 results in equation 14.

$$\Psi(\mathbf{m}) = \left(\mathbb{G}\tilde{\mathbf{m}} - \vec{d}_{obs} \right)^T \mathbb{C}_d^{-1} \left(\mathbb{G}\tilde{\mathbf{m}} - \vec{d}_{obs} \right) \tag{14}$$

Where,

$\tilde{\mathbf{m}}$: estimated model vector of slowness

$\mathbb{C}_d$: covariance matrix from data at the acquisition due to noise.

$\mathbb{G}\tilde{\mathbf{m}}$: travel-time predictions associated with model $\tilde{\mathbf{m}}$.

$\vec{d}_{obs}$: observed travel-times.

As equation 14 shows, Ψ is minimized when the model travel-times are close enough to the observed travel-times $\vec{d}_{obs}$.

Data errors are assumed uncorrelated, therefore:

$\mathbb{C}_d = [\delta_{ij}(\sigma_d^j)^2]$ with σ_d^j being the uncertainty of the j^{th} point.

One of the main issues of inverse problems is that model parameters are not well constrained by the data itself (most of the times the problem is under-determined). In order to diminish this inconvenience, there are several regularization methods that take part as additional terms added to the function to be minimized.

The first term added to equation 14: $\Phi(\vec{m})$, is defined by equation 15 and its purpose is to boost solution models $\vec{m}$ that are close to the initial model $\vec{m}_0$.

$$\Phi(\vec{m}) = (\vec{m} - \vec{m}_0)^T \mathbb{C}_m^{-1} (\vec{m} - \vec{m}_0) \tag{15}$$

Where,

$\vec{m}$: Solution model.

$\vec{m}_0$: Initial model

$\mathbb{C}_m$: *A priori* model covariance matrix (allowed variance between models).

As uncertainties in the initial model are assumed to be uncorrelated, $\mathbb{C}_m = [\delta_{ij}(\sigma_m^j)^2]$ with σ_m^j being the uncertainty of the j^{th} model parameter of the initial model.

The second term added is $\Omega(\vec{m})$, defined by equation 16.

$$\Omega(\vec{m}) = \vec{m}^T \mathbf{D}^T \vec{Dm} \tag{16}$$

Where,

$\vec{m}$: Solution model.

D: Second derivative operator.

$\vec{Dm}$: Finite difference estimate of a specified spatial derivative.

The purpose of $\Omega(\vec{m})$ is to balance the trade-off between satisfying the data, with minimum misfit models —which become very rough due to the model being adjusted to noise— and finding smooth solution models —which don't fit data— as shown in figure 9.

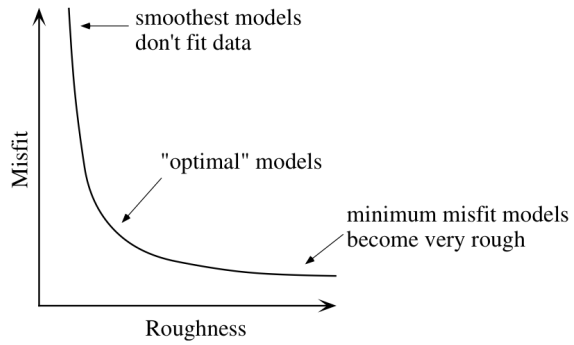


Figure 9: Regularization Trade-off. Taken from: Introduction to seismology[9]

To govern the trade-off between how well the solution model will satisfy the data, how close will it be to the initial model and how smooth will it be, there are two tuning parameters, each one multiplying one of the add terms. Those parameters are the damping factor ε and the smoothing factor η which are multiplied to $\Phi(\vec{m})$ and $\Omega(\vec{m})$ respectively.

After all the adjustments to the function, the objective function $S(\vec{m})$ is described in equation 17.

$$S(\vec{m}) = \frac{1}{2}\left[\Psi(\vec{m}) + \varepsilon\Phi(\vec{m}) + \eta\Omega(\vec{m})\right] \tag{17}$$

Which is the form of the function *invert3d* minimizes.

A lot of input files are needed in order to execute *invert3d*, the first one of them being invert3d.in lists all other needed files and then sets values to constants such as ε and η from equation 17. The format of invert3d.in is as follows in listing 4

```
 ccccccccccccccccccccccccccccccccccccccccc
c This file contains all required input
c parameters and files for the
c inversion program "invert3d.f90"
ccccccccccccccccccccccccccccccccccccccccc
vgridsref.in c: Reference velocity grid
vgrids.in    c: Current velocity grid
interfacesref.in  c: Reference interface grid
interfaces.in     c: Current interface grid
sourcesref.in     c: Reference source coords
sources.in        c: Current source coordinates
stimes.dat   c: Current source time
             c: perturbations
receivers.in c: Receiver coordinates
otimes.dat c: Observed traveltimes
mtimes.dat c: Model traveltimes
rtimes.dat c: Reference teleseismic traveltimes
frechet.in c: Frechet derivative parameters
frechet.dat c: Frechet derivatives
inviter.in  c: File indicating current inv step
propgrid.in c: File with propagation
            c:grid parameters
```

```
0.07 c: Minimum distance between interfaces(km)
0.5  c: Minimum permitted velocity (km/s)
0 c: Remove mean from predicted teleseisms
  c:(0=no,1=yes)
1 1 1  c: Velocity inv(0=no,1=yes),damp,smooth
1 1 1  c: Interface inv(0=no,1=yes),damp,smooth
0 1 1  c: Source inv(0=no,1=yes),damp1,damp2
20     c: Subspace dimension (max=50)
1.0    c: Global damping factor (epsilon)
1.0    c: Global smoothing factor (eta)
6371.0 c: Earth radius in km
```

Listing 4: Example of invert3d.in file

Lines one through six describe the initial (reference) model and the actual model —which will be rewritten after the inversion takes place—. Those six files come in pairs as three kinds of inversion can be done: interfaces inversion, velocity inversion and source location inversion. The file receiver.in contains all receiver locations. otimes.dat is the file with the observed data ($\vec{d}_{obs}$) in equation 14. The file with the current model travel-times ($\mathbb{G}(\vec{m})$) is next, its name is mtimes.dat and it simply is the file arrivals.dat renamed (this file is produced by executing *fm3d*. rtimes.dat is used with teleseismic sources, but as this is not the case it will simply be a copy of mtimes.dat renamed. The next two files are frechet.in and frechet.dat, they are input and output files for *fm3d* respectively and contain the Frèchet derivatives calculated for a subspace inversion, meaning, inverting not for the whole model but for a fragment of it. The next file, inviter.in is used to keep track of the current iteration number and finally, propgrid.in contains the propagation grid for rays to be traced.

After all those files, the lower boundaries for distance between interfaces and permitted velocities are set. Then, a switch has to set to 0 or 1 to define if the user wants to remove mean from predicted teleseisms, but again, as we are not dealing with teleseismic sources this line is left open to the reader to dig in deeper. Now come three lines which define the type of inversion that will be done along with some damping and smoothing factors; first velocity inversion, then interfaces inversion and after that, source location inversion.

It is important to note that as the focus of the research is kept in using active seismic sources, one usually knows the location of all sources so there is no point in running a inversion for source location.

Editing invert3d.in to select one or more than one type of inversion will rule over any subspace inversion set by frechgen.in, that means that if for example frechgen.in defines an inversion for interfaces 2, 3 and 4 but invert3d.in defines only to perform a velocity inversion, no change to any interface will be done at all. The next line has the subspace dimension. In a few words, this works to restrict the minimization of the quadratic approximation of the objective function $\mathbf{S}(\vec{m})$ to an n-dimensional subspace of model space. The next lines define the values of ε and η of equation 17 which simply pre-multiply the damping and smoothing factors defined individually for ve-

locity, interfaces and/or source inversion. Finally, the last line sets the Earth's radius (which must coincide with the radius set in grid3dg.in).

Even though FMTOMO comes with the inversion tool *invert3d*, in most cases it won't be necessary to execute it as a stand alone program as FMTOMO also comes with a script named *tomo3d* that executes not only the inversion process, but also the ray-propagation process.

Tomo3d[6] is a shell script that runs both the ray-propagation/trace acquisition (*fm3d*) and the inversion *invert3d* processes iteratively in order to minimize the objective function $\mathbf{S}(\vec{m})$ (eq. 17) and thereby taking into account the non-linearity of the problem in the inverse problem.

To execute *tomo3d* various files must be present:

- frechet.in
- frechgen.in
- interfacesref.in
- invert3d.in
- mode set.in
- otimes.dat
- propgrid.in
- receivers.in
- residuals.in
- sourcesref.in
- vgridsref.in
- tomo3d.in

The last line specifies the file tomo3d.in, in which (as usual) some parameters are set regarding the execution of *tomo3d*.

Tomo3d.in is composed of three lines only, in each one of them a number is set to define how will *tomo3d* be executed. For example, the first line contains the number of iterations FMTOMO is going to perform an inversion; that means that if the first line is 6, the inverse problem will be solved six times. However, the direct problem will be solved seven times as it executes *fm3d* for the initial model and also for the final model. The second line contains the option of whether to begin an all new tomographic process or to continue for another set of iterations, this is achieved by setting the number to 0 or 1 respectively. Now (if the last line is 1) the third line defines the starting point of the next set of iterations, being 0 the option to begin with *invert3d* and 1 the option to begin with *fm3d*. This last value should be usually 0 as *fm3d* is executed at the end of each iteration.

An important fact about FMTOMO is that no matter how many unknowns the inverse problem has, most of the execution time will be consumed by the forward problem, that is, the simulation of the acquisition by propagating rays through the media.

With that in mind, alternatives to improve the performance (performance can be improved by reducing the execution time) of the seismic tomography using FMTOMO are of high interest among users.

The question of how to improve the computational performance can be solved in two ways: one way is to simply improve hardware itself; the other is to improve using the available resources.

Nowadays computers have evolved, and processors have gone from being single-cored to dual-cored, quad-cored, etc. Taking advantage of that multi-cored philosophy one could find ways not to use all cores to do the same process but to use them in parallel to run several processes simultaneously.

PARALLEL MODE

As stated at the end of the previous section, parallel computing can help improving the execution performance by taking advantage of all cores resembling nodes in a cluster.

The reasoning is simple:

Why would one want to parallelize the seismic tomography process?

—Because one of the main issues with seismic-related processes is the massive amount of data they involve, that translates into high computing time, and therefore, low performance. Following that train of thoughts, parallel computing helps dealing with massive data and thereby improving performance.

Being that the case, the next question would be:

Can the seismic tomography process made by FMTOMO be parallelized?

—It can not be stated an absolute YES or NO because seismic tomography implies to solve two different problems: the forward and the inverse problem.

To talk about the forward problem means to simulate an acquisition of seismic traces. That experiment consists (in situ) in using a seismic source to propagate energy through the media and then capturing the disturbances reflected using geophones. In a real acquisition each "shot" takes place one after another and all data captured during a shot its relevant only to that shot.

That fact gives the forward problem a valuable characteristic, independence among experiments. However, in simulated acquisitions there is an advantage point, shots can be simulated almost all at once as each one of them is an independent event. That simultaneousness opens possibilities to parallelism.

On the other hand, the inverse problem needs to find a model that best fits into the data collected within certain boundaries and, to do that, it incurs into high data dependence. In simple words: to execute an inversion, all data are needed, but attempting to do it in parallel it can not be assured that all data is available at the same time. This issue makes seismic inversion a poor candidate in terms of parallelism.

Taking into account that most of the time is consumed during the forward problem and that process has high parallel capabilities, FMTOMO comes with a parallel mode used to distribute computational load into nodes to simulate acquisitions simultaneous. Say for example that an entire simulation is composed of four (4) seismic sources and the computer in which that processes is executes has a quad-core processor. The user could A) run the entire process in a sequential way, one source after another or B) run the process in parallel using each core for each source. If there are more sources than available cores, then each core will compute a group of sources until all propagation times are solved for.

Activating the parallel mode in FMTOMO is quite simple. The first step is to select the number of nodes in which the process is going to be executed; after that, the folder containing the model and the set up for the propagation must be copied the same number of times as the selected number of nodes (it does not matter if the process will be executed in different machines or in different cores of the same machine); following that, the file mode_set.in must be edited setting the switches no_pp_mode and parallel mode to true (T) and; the final step is to create a file named "myid " with two lines: the first defining the number of nodes and the second the ID of that specific node. For example, a quad-core machine would have four nodes with ID's ranging from zero to three (0,1,2,3), therefore, the second myid would contain in the first line the number 4 and in the second line the number 1. After that, the next step is to execute *fm3d* on each folder.

The previous procedure raises one question: *How are supposed to be executed the four processes at the same time?*

And that question is very important because which would be the point of parallelism if one has to manually execute each process one after another?

Fortunately, the GNU tool *parallel*[10] comes to save the day, as its main purpose is precisely to execute instructions in each core, or even in different machines.

Having reviewed the most important stages of seismic tomography, FMTOMO was challenged to show its capabilities during a set of tests whose results are presented in the following section.

TEST RESULTS

Choosing the model

To test the capabilities of seismic tomography using FMTOMO, the first step is: to choose a model that has certain characteristics of interest such as synclines, anticlines, seismic faults and high velocity intrusions. Figure 10 shows the chosen model created from synthetic data using Seismic Unix (SU) [11] and as it can be seen, it has both synclines, anticlines, a seismic fault and a high velocity intrusion.

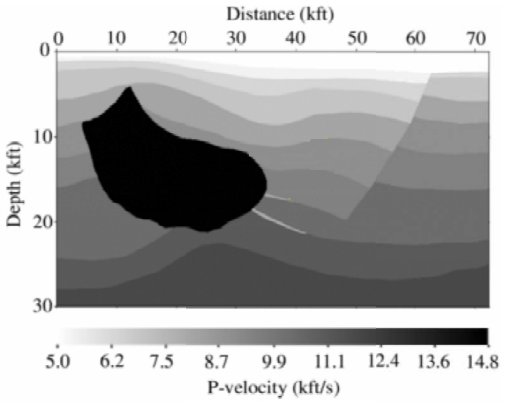


Figure 10: Choosen Earth's Model

Constructing the model in FMTOMO

Once the study model is defined, it has to be constructed using FMTOMO's tool for creating Earth models: *grid3dg*. A similar model to the one in figure 10 was constructed using *grid3dg* but edited by hand to be able to model the finest details. Figure 11 shows the model used as original model for all subsequent tests.

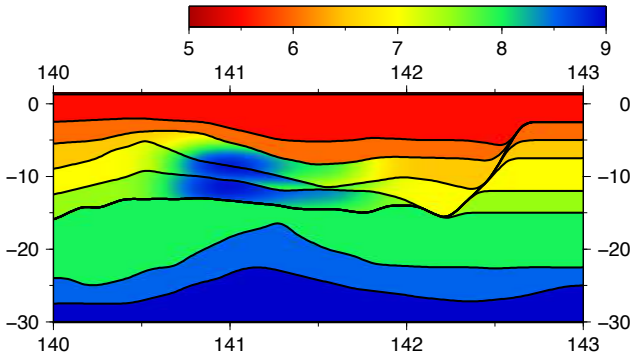


Figure 11: Original Model Constructed with FMTOMO

It is not an exact replica of the proposed model but it keeps the main characteristics of interest.

Solving the Forward Problem

The simulated acquisition show in figure 12 illustrates the rays traced through the model and as it can be seen from it, the last layer has absolutely no rays in it , that is for computational purposes only as there is no information on lower layers and therefore no reflections can be obtained.

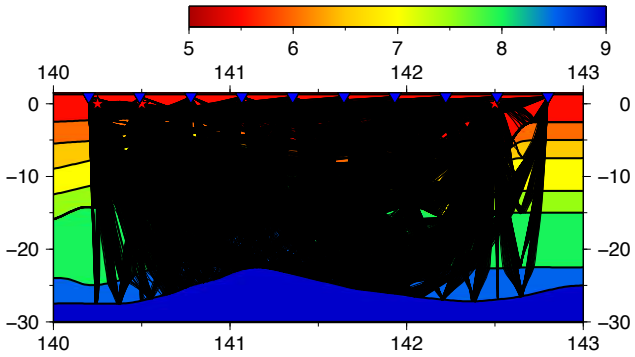


Figure 12: Original Model after simulating acquisition

One point against FMTOMO is that as the model was edited by hand, some rays presented errors an were not traced. The resulting files after running *fm3d* are arrivals.dat and frechet.dat which, as commented before, contain the rays' travel times and the Frèchet derivatives. As explained above, the objective is to obtain a file with the observed times (dobs), to achieve that, *synthdata* takes arrivals.dat and adds random noise to it and turns it into otimes.dat.

Solving the Inverse Problem

Once the forward problem has been solved and travel-times were obtained, what follows is to use that information to take an initial model and modify it iteratively as close as possible to the original model.

Several tests were done with different initial models because the start point of minimization problems is of extreme importance. If an initial model M_0 is close to the original model M_{True}, the solution model M_{Sol} will most likely converge to a closer point to M_{True}. Being said that, it is important to remember that a whole process of selecting initial models can be done through global optimization techniques as explained in[12].

The reasoning above is to say that solution models obtained are subject to not being acceptable solutions as they can be local minimums, not global ones.

The methodology used to select the initial model is quite empirical, but helps defining the strengths and weaknesses of FMTOMO.

The tests started with a single-layered model, then the number of layers was increased to three to prove if FMTOMO was able to forecast the real number of layers and add them to the model. Unfortunately, the discrepancy between the interfaces of the real model and the initial model made it impossible to obtain a model by performing an inversion process. Neither one, nor three layers were enough for FMTOMO to correctly perform an inversion.

Due to the results above, the next test was to start with a model with the same number of layers, all of them horizontally oriented. The velocities selected for this test ranged from low to high. Figure 13 shows a low-velocity model used as initial model and figure 14 shows the solution model after six iterations.

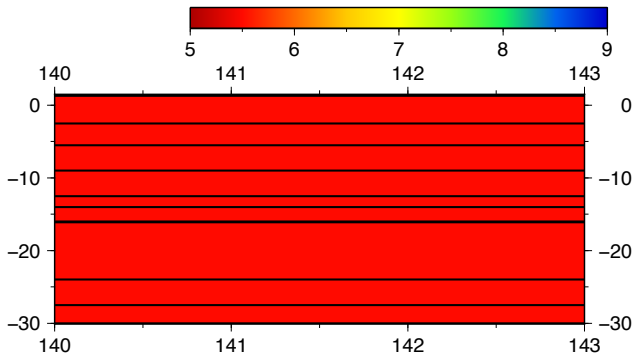


Figure 13: Low-Velocity with Horizontally Oriented Interfaces Model used as M_0

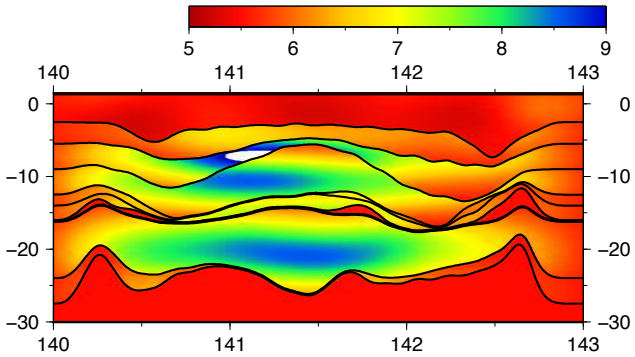


Figure 14: Solution Model obtained after six iteration steps using model in figure 13 as M_0

It is worth noting the high-velocity zone that appears in figure 14, ressembling the intrusion in the original model in figure 11 located around coordinates (141,-10). Another interesting feature is how interfaces try to pinch-out at coordinates (142.5,-15) being perhaps the effect of the seismic fault in the original model.

Finally, a similar test to the one before was done, but in this case interfaces had inclinations depending of how they were oriented in the original model. For velocities, the test was made as before: from low to high velocities and one model with the exact same velocity distribution as the original model.

Figures 15 and 16 show the initial and obtained model using low velocities respectively, while figures 17 and 18 show one test using the same velocity distribution as the original model.

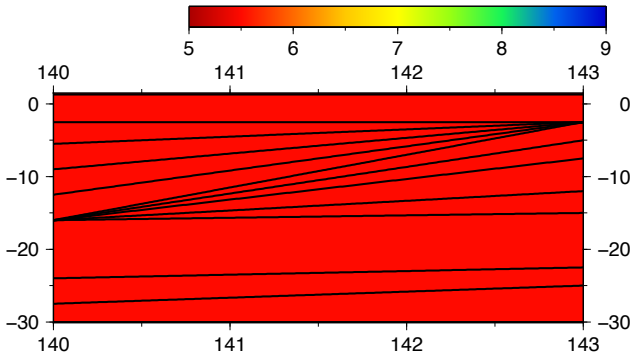


Figure 15: Low-Velocity Model with diagonally oriented interfaces used as M_0

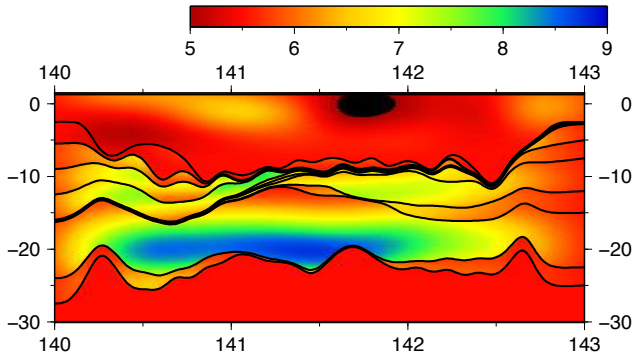


Figure 16: Solution Model obtained after six iteration steps using model in figure 15 as M_0

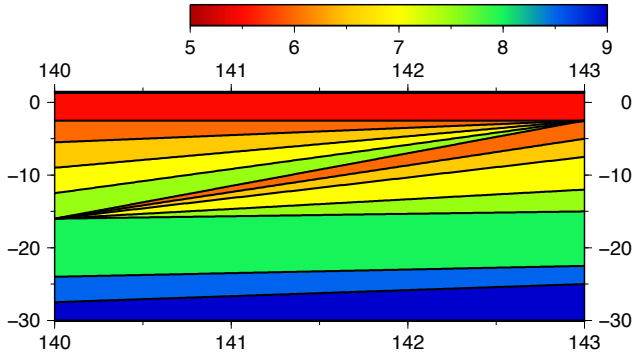


Figure 17: Original-Velocity with diagonally oriented interfaces Model used as M_0

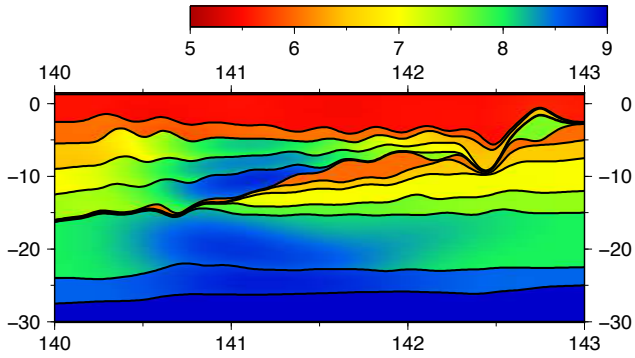


Figure 18: Solution Model obtained after six iteration steps using model in figure 17 as M_0

In both tests above, it is remarkable how the seismic fault stands out and even the high-velocity intrusion appears in both models (more clearly in the latter than the first).

Using GNU Parallel and FMTOMO's Parallel Mode

Prior to engage in the whole "parallel computing" concept, it was necessary to measure the execution time of all processes to

confirm that as the instruction manual says, the forward part of the problem is the most time-consuming part. Table 1 contains the time that each FMTOMO'S instruction took to execute a set of examples.

Instruction	Time 1[s]	Time 2[s]	Time 3[s]
arraygen	0.077	0.030	0.021
grid3dg	0.424	0.252	0.427
moddata	0.085	0.083	0.076
frechgen	0.004	0.051	0.044
fm3d	90.673	250.311	840.196
synthdata	0.069	0.229	0.640
invert3d	2.814	3.700	25.818
residuals	0.067	0.038	0.113

Table 1: Execution time.

From simpler models to more complex ones, the time it takes to execute the forward problem is by far higher than the time consumed by any other FMTOMO's tool.

Fortunately, the most time consuming instruction is also the most parallelizable one and, following that train of thoughts, consumed time can be reduced by two means: parallelizing the forward problem or using a more powerful machine.

The statement above brings up four possible scenarios: one, using the lowest-end machine and solving the forward problem in a sequential way (worst-case scenario); another scenario is using the same machine but performing the process in parallel; the next scenarios occur when using a higher-end machine, solving the sequential forward problem or the parallelized forward problem (best-case scenario).

Both used machines are compared in table 2. The first is my personal computer and the latter is a server owned by CPS research group at UIS, named Osaka.

	Author's PC	Osaka
CPU	Intel Core i5-2430M	Intel Xeon E5-2609
CPU Cores	2 (Emulates 4)	4
CPU's Frequency	2.40 GHz	2.40 GHz
CPU's DMI	5 GT/s	6.4 GT/s
CPU's Cache Memory	3 MB	10MB
RAM	4GB DDR3	8 GB DDR3
RAM's Frequency	1333MHz	1600MHz

Table 2: Computers used to execute FMTOMO.

As both machines can perform tasks using four cores (even though, mine has only two real cores and two virtual ones) the exercise was to copy the folder in which the forward problem is solved to four new folders and as seen in section 6, in each of them a file named myid is created with the first being the same for all and the second line ranging from zero to three to set each folder's ID. The next step is to execute *fm3d* in all folders, that is when *parallel* comes into play to simultaneously execute the instructions.

Table 3 shows the time it took to execute *fm3d* to propagate rays through the original model (Fig. 11) in each scenario (two different machines with and without parallelizing).

FM3D	Author's PC	CPS' PC
Not Parallel	13m57s	9m19s
Parallel	8m12s	2m50s

Table 3: Four scenarios of executing *fm3d*.

DISCUSSION

In all test run, the objective was to find a solution model similar to the original one by trial and error using different initial models that grow in complexity with each trial. The obtained results showed that not knowing the number of interfaces is a huge problem, however, taking into account that the "original" model is also generated using the computer and the whole process was simulated, it does not take to much effort to adjust the number of layers of both models to match. The issue turns critical when performing a in-situ acquisition because there is no way to know the so called "original" model.

The problem stated above is hard to completely solve, nevertheless, an alternative could be to let loose the constrains surrounding the interface inversion and redirect the efforts towards the velocity inversion. Losing (read: not gaining) information about interfaces is a major let down; on the other hand, information about the velocity model is still available and can be estimated with less issues than information about interfaces.

The answer to the question of which test came up with the closest model to the original? Is a tough one as every solution model had something to say about the original. That is perhaps the most important result of all, not the solution models viewed as single-answers, but to see them as pieces of one bigger puzzle. With each test some features appeared or hid in the model, that means that with every obtained model, new initial models M_0 could be proposed in order to reinforce an specific characteristic.

An experimental way of determining features could be similar to the exercise proposed in this work, trying different initial models in order to look for features that most of the solution models have in common. That could be a valuable clue for geophysicists to encourage or discourage fossil fuels exploration in certain zones.

It is remarkable how parallel computing reduces the computing time of a task (given that the task in question can be parallelized), the results in table 3 show that as expected, improving the machine and performing the task in parallel way, the best-case scenario is produced, while the worst case is using the slowest machine and not parallizing the task. Moreover, it is outstanding that by using parallel computing the computation time was reduced even more than improving the hardware of the machine. Even though this result depends highly on the machines used, it is worth noting that one PC is an almost-two-years-old (worth $800 US at that time) while the other is a brand new server computer (worth $ 4000 US).

CONCLUSIONS

Seismic Tomography

The main purpose of seismic tomography is to obtain velocity models that best adjust to a set of travel-times, being a complementary tool for seismic imaging processes. Proposing the seismic tomography as an inverse problem allows to find a possible solution by predicting a model that best fits a set of travel-times, however, multiple solutions can be obtained and those possible solutions highly depend on the starting point or initial model from which the process starts. For that reason, selecting an appropriate initial model could mean the difference between finding a acceptable solution to the inverse problem or not.

FMTOMO

About FMTOMO, it can be concluded that it has both pros and cons as any computational tool would. Below are listed some of the advantages and disadvantages of FMTOMO:

Advantages

- FMTOMO allows to emulate the most important parts that take place in a seismic tomography process, from modelling, to propagating rays, to inverting the data obtained and (with the proper visualization tools), to illustrate graphic results.
- The fact that FMTOMO is an open-source tool is by itself an advantage point to it as it allows to people all around the world to take advantage of the tool for seismic exploration, to make a contribution and give feedback to the author, PhD Nicholas Rawlinson.
- One particular strength of FMTOMO is its capability to solve the inverse problem for velocity grids rather than for interfaces (as seen in figure 18).
- The parallel mode embedded within FMTOMO opens up a whole new point of view for seismic problems, which are now not only geophysicist's problems, but have turned to be multidisciplinary problems that can be dealed with from different approaches: physical, mathematical and nowadays, computational.

Disadvantages

- Even though FMTOMO comes with a handy tool for creating Earth models, the most refined details have to be added by hand which can result in inconsistencies if the user is not careful enough.
- FMTOMO deals poorly with structures such as salt domes as the interfaces modelled in FMTOMO cannot fold over themselves, which kind of limits the range of applications for the tool.
- If the model from which the observations are made is created using FMTOMO, the initial model from which the tomographic process begins must have the same number of interfaces as the original model. That translates into a huge limitation for FMTOMO. In fact, the best tomographic results come from using initial models close to the real model as they are most likely to converge from the space of solutions to a local minimum which will be close the the global minimum and hence, the real model.
- Interface inversions can be done using FMTOMO, however the results obtained so far have not been as conclusive as the results obtained by performing velocity inversions.
- The parallel model has yet to be proved and improved to provide a better use of resources of these new emergent technologies —technologies not present ten or twenty years ago—.

Parallel Computing

As times goes by, computational prowess will be improved, but the real point is to really take advantage of all available resources to overcome the technology limitations in order to be able to handle bigger and tougher problems.

FMTOMO and Parallel proved to be complementing tools that can team-up to deal with a problem from various fronts in order to obtain good results while reducing computing time by taking advantage of all computational resources available.

Table 3 showed four scenarios in which the boundary scenarios were known from the beginning, but it was uncertain which of the other two scenarios consumed the lowest time. The result turned out to be quite surprising, as using parallel computing the consumed time was lower than physically improving hardware (which is by far the most expensive solution). This means that parallel computing is a real alternative to improve computational performance without having to invest a huge amount of money on hardware —the only limitation is that the process must be parallelizable in order to really take advantage of parallelism—.

With the results obtained from this work, further researches can take place to improve the performance of FMTOMO (geophysics approach) or to increase the level of parallelism and hence prove the scalability and use of *parallel*.

ACKNOWLEDGEMENTS

First of all, I would like to thank my mother, my beloved best-friend Carolina and my family for their love and support during my whole career and even my whole life. To my father: *Javy* I wouldn't have done it without you.

To my advisor for leading me not only during this process but also giving me important advises. To the CPS research group and specially to PhD. Oscar M. Reyes directing the lives of so many researchers and for letting me in to the group and be involved in this project.

To the Colombia's Petroleum Institute (ICP for its acronym in spanish) for funding projects like this one through technological cooperation agreements such as Agreement UIS-ICP No.005 of 2003.

To the author of FMTOMO, Nicholas Rawlinson, not only for developing such an useful tool for performing seismic tomography but also for taking the time to read my mails and giving answer to them solving some of my initial doubts.

To Santander's Industrial University (UIS for its acronym in spanish) for giving me a high-level education and a place to go in pursue of knowledge.

Thanks to everyone involved in the accomplishment of this achievement.

APPENDIX A

SOURCES OF THE BIBLIOGRAPHY

```
% BIBLIOGRAPHY ARTICLE FWI
% JANUARY 2014
%%%%%%%%%%%%%%%%%%%%%%% SECTION-1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

@article{Abreo2012,
author = {Abreo, Sergio},
title = {{Viability of implement the seismic tomography to determine the
velocity models of new and / or complex zones over an Alternative High
performance Computing ( AHPC ) Platform}},
year = {2012}
}

@book{lehman,
author    = {Lehman, Bodo},
title     = {Seismic traveltime tomography for engineering and exploration applications},
publisher = {EAGE Publications},
year      = {2007}
}

@Misc{orfeus,
 author = {{Observatories and Research Facilities for European Seismology}},
 title = {},
 note ={{Review: June 2013}},
 howpublished={\url{http://www.orfeus-eu.org/software/seismo_softwarelibrary.html}}
 }

@Misc{nick,
 author = {{Rawlinson, Nicholas }},
 title = {},
 note ={{Review: June 2013}},
 howpublished = {\url{http://rses.anu.edu.au/~nick}},
 }

@Misc{FMTOMO,
 author = {{Rawlinson, Nicholas}},
 title = {},
 note ={{Review: June 2013}},
 howpublished = {\url{http://rses.anu.edu.au/~nick/fmtomo.html}},
 }

@book{manual,
author    = {Rawlinson, Nicholas},
title     = {Fast Marching Tomography Package: Instruction Manual},
publisher = {Research School of Earth Sciences, Australian National University, Canberra ACT 0200},
year      = {2007},
}

@article{KeiitiAki1976,
author = {{Keiiti Aki}, W. H. K. Lee},
title = {{Determination of three-dimensional velocity anomalies under a seismic
array using first P arrival times from local earthquakes. 1. A homogeneous
initial model}},
volume = {81},
year = {1976}
}
```

```
@book{parameter,
author    = {Aster, Richard},
title     = {Parameter Estimation and Inverse Problems},
publisher = {Academic Press},
year      = {2005},
}


@book{shearer,
author    = {Shearer, Peter},
title     = {Introduction to seismology},
publisher = {Cambridge University Press},
year      = {2009},
}


@Misc{parallel,
 author = {{GNU Project}},
 title = {},
 note ={{Review: July 2013}},
 howpublished = {\url{http://www.gnu.org/software/parallel/}},
 }

@Misc{seismicunix,
 author = {{Stockwell, John}},
 title = {},
 note ={{Review: July 2013}},
 howpublished = {\url{http://www.cwp.mines.edu/cwpcodes/}},
 }

@book{sen1995,
  title={Global Optimization Methods in Geophysical Inversion},
  author={Sen, K. and Stoffa, P.L.},
  isbn={9780080532561},
  lccn={95012651},
  series={Advances in Exploration Geophysics},
  url={http://books.google.com.co/books?id=PT5MqSIvREMC},
  year={1995},
  publisher={Elsevier Science}
}
```

APPENDIX B

INSTALLATION AND MODELLING USING FMTOMO BLOG

INTRODUCCIÓN

Con el fin de estudiar el proceso de tomografía sísmica y su implementación computacional se ha decidido, tras un período de pruebas con diferentes distribuciones de software de la comunidad geofísica, utilizar la distribución FMTOMO. Esta distribución de software se puede descargar de la web de la Universidad Nacional de Australia (UAN por sus siglas en inglés): http://rses.anu.edu.au/∼nick/fmtomo.html

Características Importantes de FMTOMO

- La estructura del modelo se representa por capas sub-horizontales en coordenadas esféricas. Dentro de cada capa, se permite la variación suave de la velocidad en tres dimensiones. También es posible especificar un medio descrito únicamente por un "continuo" de velocidad variable (sin interfaces).
- La geometrá de los bordes buede variar también, permitiendo una amplia variedad estructuras representables.
- Las restricciones de velocidad, estructura de la interface y la ubicación de la fuente pueden ser restringidas por separado o en conjunto en la inversión.
- Se pueden usar una amplia variedad de conjuntos de datos, incluyendo reflexiones, *wide-angle* y eventos sismológicos locales para restringir estructuras en tres dimensiones.
- Es posible realizar inversión conjunta de multiples conjuntos de datos
- El uso de coordenadas esféricas implica que son posibles aplicaciones tanto locales como semi-globales.

Este documento sirve de guía para todo aquel interesado en realizar procesos de inversión tomográfica haciendo uso de esta herramienta computacional.

INSTALACIÓN

Software Requerido

Para la instalación de la distribución FMTOMO se hace necesario disponer de un compilador de Fortran. Por experiencia el compilador que viene por defecto en gfortran conocido como f95, presenta problemas. La recomendación que se hace en la web oficial de FMTOMO es trabajar utilizando el compilador de Fortran de Intel, conocido como ifort.

Para la visualización de resultados e imágenes sísmicas generadas se emplean las herramientas GMT y OpenDX.

Cada una de las herramientas se encuentran en sus páginas oficiales:

- ifort: http://software.intel.com/en-us/intel-compilers
- GMT: http://gmt.soest.hawaii.edu
- OpenDX: http://www.opendx.org

Intel Fortran Compiler ifort

Para descargar el compilador de Fortran de Intel se debe acceder a la página anteriormente escrita y en la zona de descargas se debe seleccionar un tipo de licencia (comercial, académica o no comercial), realizar un registro en el que piden el correo electrónico y mediante un e-mail enviado por Intel entrar al link de descarga correspondiente junto con un el número de serie asignado.

Una vez descargado y antes de ser instalado, se deben instalar ciertas librerías, para esto basta con digitar en consola las siguientes instrucciones:

$sudo apt-get install rpm build-essential

$sudo apt-get install libstdc++6

$sudo apt-get install ia32-libs

Culminada la instalación de las librerías, se puede descomprimir el archivo descargado de la página de intel. Para instalarlo, basta con ejecutar el archivo install.sh como sigue:

$ sudo ./install.sh

Las instrucciones de instalación son sencillas, basta con seleccionar el compilador y digitar el número de serie que llegó en el correo. Se acepta el acuerdo de licencia y finalmente antes de terminar, puede que el proceso de instalación muestre una advertencia de paquetes opcionales no instalados. Esta se puede ignorar.

Finalmente para usar el compilador, se debe ejecutar en consola el comando ifort:

$ ifort “codigo”.f

Se hace incapié en la relevancia del compilador, debido a que los compiladores usados diferentes del compilador de intel tuvieron errores en los binarios generados.

GMT 4.5.9

La herramienta GMT (Generic Mapping Tool) será importante para ubicacion de fuentes sísmicas y de sus receptores. Para su instalación se hace necesario tener instalados los siguientes paquetes con anterioridad:

- zlib 1.2.8
- hdf5 1.8.11
- curl 7.22.0
- netCDF 4.3.0
- gdal 1.10.0

Para su instalación se recomienda crear una carpeta /../local en un lugar conocido por el usuario, por ejemplo /home/“usuario”/local

Una vez creada la carpeta el proceso de instalación de los cuatro paquetes es sencillo.

Para zlib, dentro de su carpeta hacer:

$ make clean

$./configure - -prefix=/home/“usuario”/local

$ make check install

Un proceso similar se lleva a cabo dentro de las carpetas de los programas restantes salvo por un detalle al escoger el –prefix que llevará ahora la dirección creada con anterioridad. Esto es, para curl por ejemplo:

$ make clean

$./configure - -prefix=/home/“usuario”/local

$ make check install

Y para hdf5:

```
$ make clean
$ CFLAGS=-O0
$ ./configure - -with-zlib=/home/“usuario”/local - -prefix=/home/“usuario”/local
$ make
$ make check
$ make install
```

Se le agrega la bandera CFLAGS = -O0 para que no existan errores con la optimización a la hora de compilar. Tambin se ejecutan los comandos make uno por uno debido a que se tuvieron fallas en la compilación al hacer directamente el “make check”.

Con los tres primeros programas instalados, se procede a instalar netCDF.

Como se ha hecho anteriormente, se procede con un comando make clean, tras lo cual se deben ajustar las variables CPPFLAGS y LDFLAGS como se muestra a continuación.

```
$ CPPFLAGS=-I/home/“usuario”/local/include
$ LDFLAGS=-L/home/“usuario”/local/lib
```

Y de nuevo:

```
$./configure - -prefix=/home/“usuario”/local - -disable-netcdf-4
$ make check install
```

Se agrega la opción - - disable-netcdf-4 debido a errores en la compilación que surgieron de la no desactivación del programa.

Ahora, para gdal el procedimiento es similar, teniendo en cuenta que se deben agregar los paths para curl y hdf5. Este proceso se muestra a continuación:

```
$ make clean
$ ./configure - -prefix=/home/”usuario”/local/
              - -with-hdf5=/home/“usuario”/local/
              - -with-curl=/home/“usuario”/local/bin/
$ make install
```

Finalmente con todos los programas instalados, se deben agregar al $PATH para poder acceder a ellos desde la consola. Para esto basta con editar el archivo .bashrc y aadir a la variable PATH las carpetas donde se encuentren los binarios. Por ejemplo:

```
PATH = $PATH:/home/”usuario”/local/bin
export $PATH
```

Se puede proseguir ahora a instalar GMT mediante la ejecución del archivo install-gmt.sh. Basta con seguir las instrucciones del programa para su instalación. Preguntará por las ubicaciones de los binarios anteriormente instalados y preguntará si se desean instalar más paquetes. La instalación se deja a decisión del usuario.

OpenDX

OpenDX es un sistema de herramientas e interfaces para visualización de datos. Su utilidad se encuetra en su capacidad para representar gráficamente las capas interiores de la tierra como un "continuo". Se puede descargar directamente del centro de Software de Ubuntu o alternativamente desde consola mediante comando:

```
$sudo apt-get install dx
```

Instalación FMTOMO

La distribución FMTOMO viene con un archivo ejecutable en consola llamado compileall. Este genera los binarios que en conjunto representan a FMTOMO. Estos se guardan en el directorio /bin de la carpeta. Los programas generados son:

- arraygen: programa generador de arreglos de receptores.
- frechgen: programa generador de las derivadas de Frechet
- grid3dg: programa para construir modelos 3D en el formato requerido por fm3d. Se puede usar para contruir modelos de prueba sintéticos y modelos iniciales para la inversión tomográfica.
- moddata: genera archivos de entrada de fuentes y receptores en el formato requerido por fm3d. Su principal uso es para realizar pruebas sintéticas.
- residuals: computa los residuales de los tiempos de propagación asociados al modelo.
- tomo3d: es el programa princial, se encarga del proceso iterativo del método ejecutando fm3d e invert3d consecutivamente.
- fm3d: se encarga de resolver el problema directo mediante el método multi-etapa de marcha rápida y genera los archivos necesarios para realizar el problema inverso.
- gmtslice: es el programa encargado de generar los archivos necesarios para la visualización mediante GMT.
- invert3d: este programa resuelve el problema inverso de los tiempos de propagación para la estructura del modelo usando el método de inversión subespacial y sobreescribe archivos para realizar de nuevo el problema directo.
- obsdata: genera archivos de entrada de fuentes, receptores y tiempos de propagación en el formato requerido por fm3d. Su uso principal es para involucrar observaciones.
- synthdata: su función es generar tiempos de propagación sintéticos para FMTOMO.Toma la salida de fm3d y le agrega ruido aleatorio.

Una descripción detallada de cada uno de ellos se puede encotrar en el manual de instrucciones de FMTOMO, disponible con su distribución.

CREACIÓN DE MODELOS

1. Programas y Archivos Necesarios.
   El programa con el cual se construyen los modelos para FMTOMO se llama *grid3dg*. Para ejecutarlo se necesita un archivo de entrada de igual nombre con extensión .in (grid3dg.in). En este archivo se definen las dimensiones del modelo, sus interfaces (reflectores) y sus capas (con sus velocidades asociadas). Tras modificar grid3dg.in y ejecutar *grid3dg*, los archivos generados son tres: interfaces.in, vgrids.in y propgrid.in.
2. Modificación de Archivos de Entrada.

   En grid3dg.in lo primero que se configura es el número de capas (este es el número de interfaces menos uno), el número de tipos de velocidad (1 para P, 2 para P y S). Posteriormente se definen las dimensiones como tales del modelo (el rango longitudinal, latitudinal y de profundidad). Como tercer paso se define la malla de propagación, que no debe confundirse con la de velocidades.

   Hecho lo anterior, el archivo procede con la configuración de cada una de las capas de velocidad. Un ejemplo, a continuación:

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c Set velocity grid values for layer 2
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
12          12          c: Number of radial grid points
12          2           c: Number of grid points in theta (N-S)
12          18          c: Number of grid points in phi (E-W)
1                       c: Use model (0) or constant gradient (1)
P                       c: Use P or S velocity model
ak135.vel               c: Velocity model (option 0)
1                       c: Dimension of velocity model (1=1-D,3=3-D)
8.0          6.0        c: Velocity at origin (km/s) (option 1)
9.5          7.0        c: Velocity at maximum depth (km/s) (option 1)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Las primeras tres lineas definen la malla de velocidades en las tres dimensiones (radial, longitudinal y latitudinal) La cuarta linea define si el modelo será un gradiente constante o se obtendrá de un archivo externo. Para la primera opción, se define en las últimas dos lineas el intervalo de velociades para esa capa. Es de notar que para las primeras tres y las últimas 2 lineas hay pares de datos, esto se usa cuando se van a trabajar con dos tipos de velocidades (P y S) que definen dos mallas de velocidad superpuestas. Las lineas 5, 6 y 7 definen si se usa la malla de velocidades P o S, el nombre del archivo externo con las velocidades (en caso de no seleccionar el gradiente) y el tipo de modelo de velocidades que dependiendo si la velocidad solo varía con la profundidad o también varía en latitud y longitud, puede ser 1D o 3D respectivamente.

Después de esta configuración, se pueden agregar tres opciones a la capa. La primera es una estructura aleatoria, la segunda es una configuración de tablero de ajedrez, y la tercera para agregar picos de velocidad en puntos establecidos. As, una por una se definen las capas de velocidades del modelo.

Con las capas definidas, el último paso es definir las interfaces. Esto se puede hacer desde un archivo externo o definiendo los puntos noroeste, suroeste y noreste de la interface como se muestra a contiuación:

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c Set up interface grid for interface 2
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
0                       c: Obtain grid from external file (0=no,1=yes)
interface2.z            c: External interface grid file (option 1)
1.3                   c: Height of NW grid point (option 0)
-48.0                   c: Height of NE grid point (option 0)
1.3                   c: Height of SW grid point (option 0)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

En las últimas tres lineas, valores positivos se definen por encima del nivel del mar y negativos por debajo. De igual forma que las capas, las interfaces se definen una por una pudiendoles agregar estructuras aleatorias para no generar lineas perfectamente rectas.

Con el archivo grid3dg.in listo, ya se puede ejecutar el comando *grid3dg*.

3. Ejemplo.

En el manual de instrucciones de FMTOMO, se encuentra un ejemplo de corte transversal de un modelo de velocidades complejo que se muestra en la B-1.

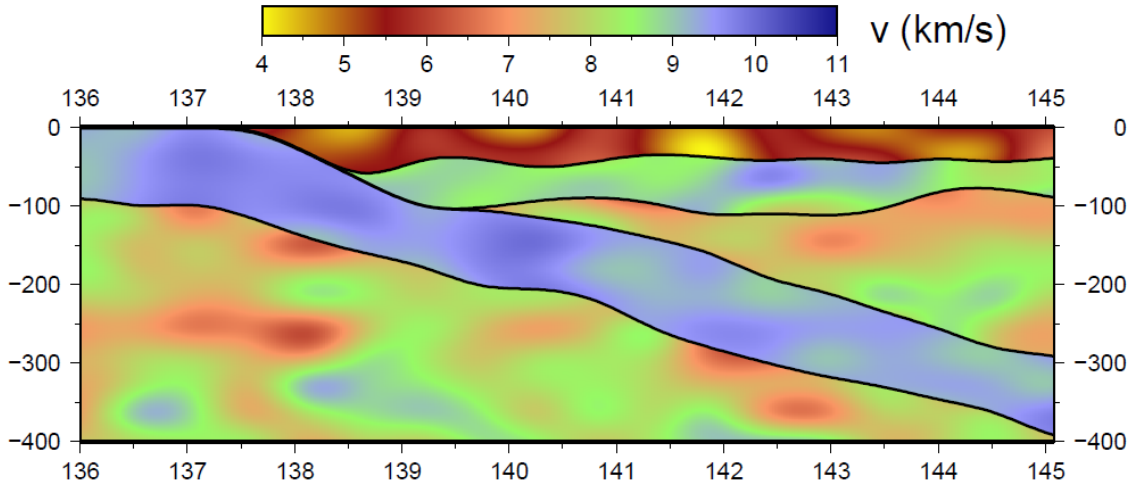


Figure B-1: Corte transversal Este-Oeste de un modelo de velocidades complejo (mostrando una posible zona de subducción sintética)

La idea es entonces reproducir el modelo observado en la Figura B-1. Para esto se observa que inicialmente, hay 6 interfaces: las dos externas que definen las dimensiones del modelo en profundidad y cuatro internas con velocidades diferentes en cada una de las capas.

Al configurar el archivo grid3dg.in se definen las dimensiones del modelo y se editan las 6 interfaces y las 5 capas. Debido a que se tiene sólo un corte del modelo, las dimensiones N-S no son relevantes.

Las dimensiones del modelo se definieron en la Tabla 4 como:

Rango	Inicio	Fin
Radial (Top-Bottom) [km]	1.5	-400
Latitudinal (N-S) [Grados]	-40	-42
Longitudinal (E-W) [Grados]	136	145

Table 4: Dimensiones del Modelo.

Posteriormente se deben definir los gradientes de velocidad de cada una de las capas, estos se muestran en la Tabla 5.

Capa	Velocidad Inicial [km/s]	Velocidad Final [km/s]
1	5	6
2	8	9.5
3	7	8.5
4	10	10.5
5	5.5	10

Table 5: Rango de velocidades de cada capa.

Finalmente las interfaces definidas se muestran en la Tabla 6. Para definir cada interface se debe establecer la profundidad de sus límites noroeste (NW), noreste (NE) y suroeste (SW)

Interface	NW [km]	NE [km]	SW [km]
1	1.3	1.3	1.3
2	1.3	-48	1.3
3	1.3	-98	1.3
4	1.3	-298	1.3
5	-98	-398	-98
6	-400	-400	-400

Table 6: Ubicación de los puntos extremos de cada interface.

Tras modificar el arhcivo grid3dg.in con los valores anteriores y deshabilitando las configuraciones opcionales, el modelo obtenido será como se observa en la Figura B-2.

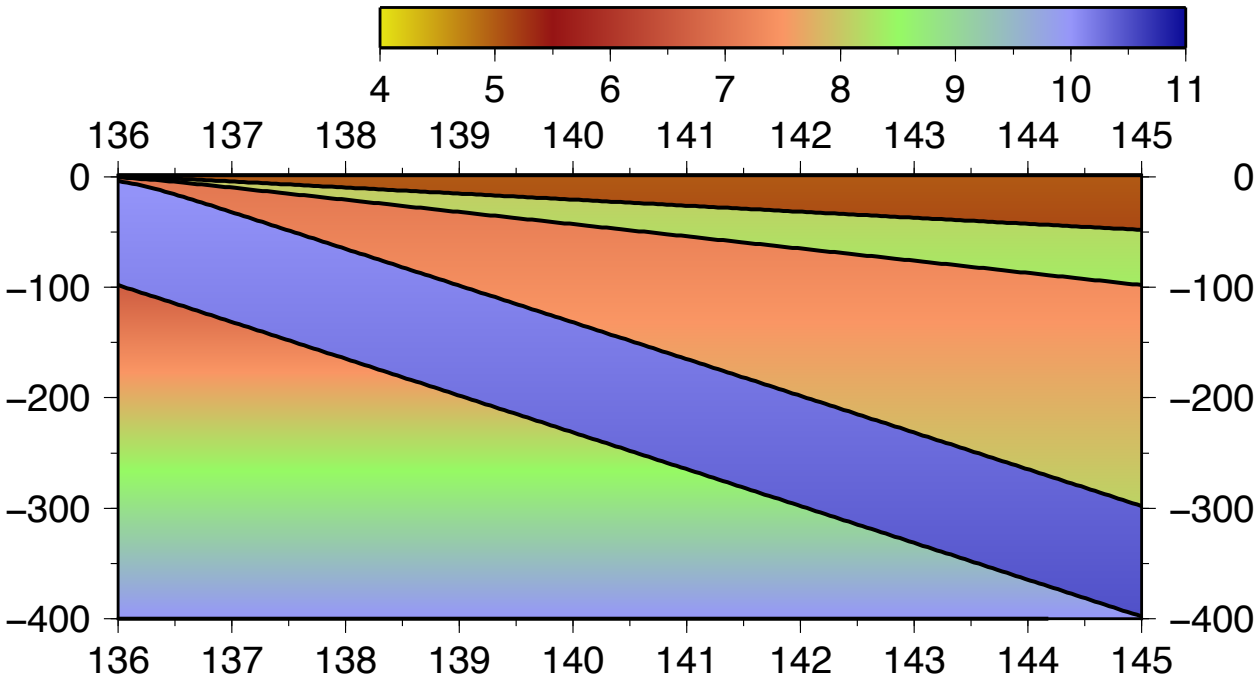


Figure B-2: Modelo obtenido mediante la edición de grid3dg.in

Como se observa en la Figura B-2, las interfaces son rectas y las velocidades varían como gradiente con la profundidad, pero lo más importante es que las primeras 4 capas estan iniciando en un punto en común, diferente a lo que se ve en la Figura B-1 en donde las primeras 3 capas no inician en el extremo superior izquierdo del modelo. Esto se resolverá más adelante.

Como segundo paso, se habilitó la opción de aleatoriedad en cada capa de velocidad. Con esta opción ya no se tendrán gradientes de velocidad tan proporcionales a la profundidad. El modelo se aprecia en la Figura B-3.

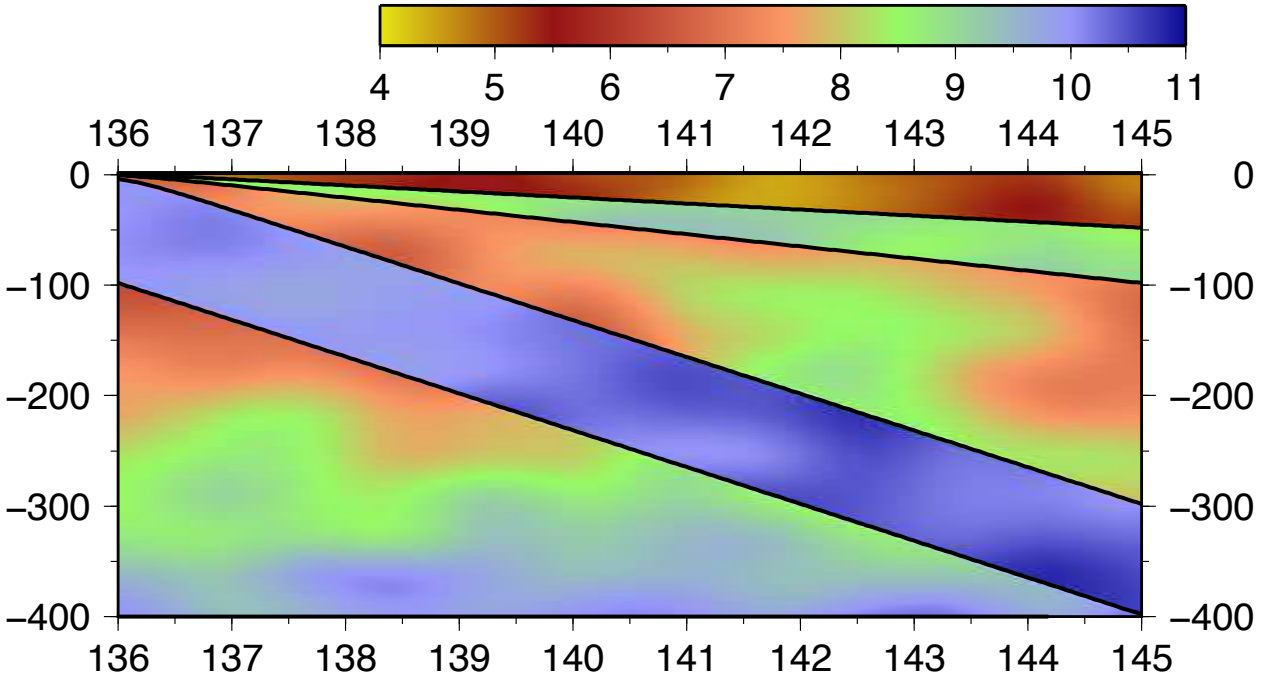


Figure B-3: Modelo obtenido con ruido aleatorio para las velocidades de cada capa.

Es de resaltar que la velocidad seguirá variando con la profundidad de acuerdo al gradiente propuesto anteriormente, pero se le adiciona una componente aleatoria de naturaleza gaussiana, tal que, su desviación estándar no ocasione que la velocidad sea menor que la mínima velocidad propuesta ni mayor que la máxima. En este caso específico, que en nungún momento la velocidad sea ni menor a 4 km/s ni mayor a 11 km/s.

De manera similar a las capas, en la Figura B-4 se muestra el modelo cuando se habilita la aleatoriedad en las interfaces interiores del modelo con el fin de no tener superficies perfectamente rectas (algo improbable en un modelo real de la tierra).

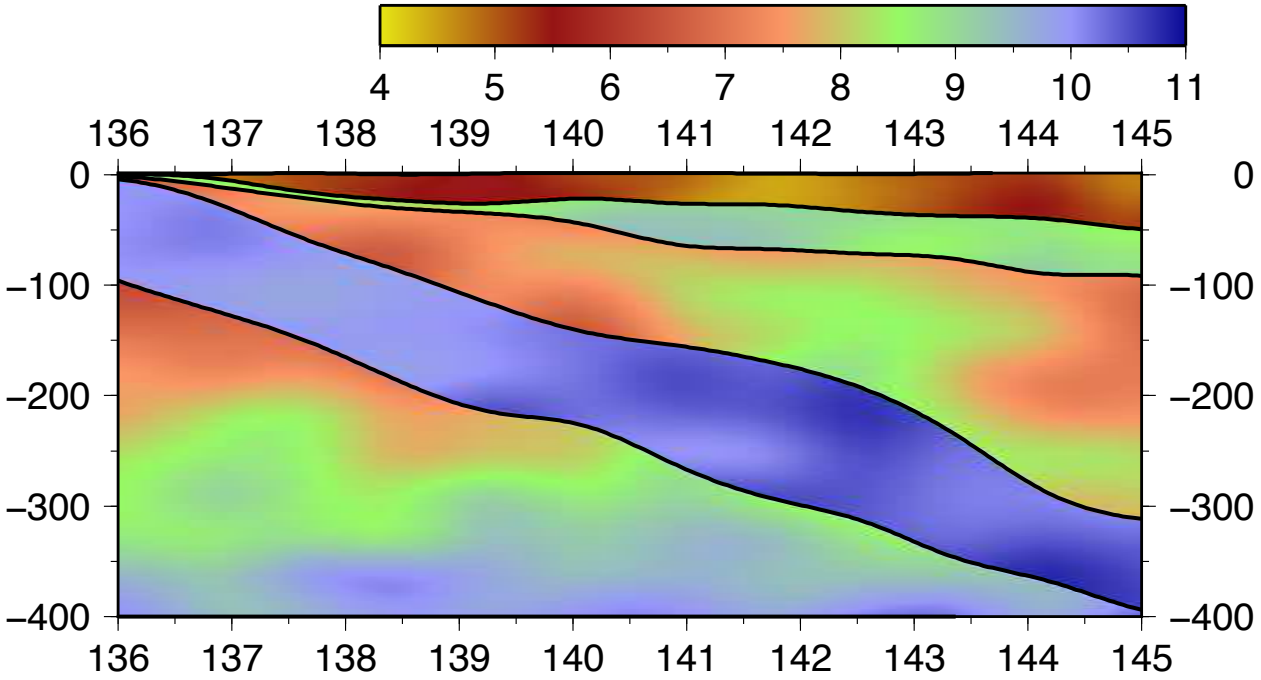


Figure B-4: Modelo obtenido con ruido aleatorio para cada interface.

Una vez en este punto, se debe solucionar el problema con el punto de inicio de las interfaces 2, 3 y 4. Para modificar el modelo se hace necesario ahora editar directamete el archivo interfaces.in. Con el fin de editarlo correctamente, se debe reconocer su estructura: en la primera línea se encuentra el número de interfaces, en la segunda se encuentran el número de nodos en latitud y en longitud respectivamente, en la tercera linea están definidos los espaciados de latitud y longitud (en radianes), en la cuarta linea se encuentra el punto de origen en radianes de la malla de interfaces.

Para el ejemplo en cuestión, el archivo interfaces.in comienza así:

```
            6
            5              21
 1.745329300562543E-002    8.726646502812704E-003
-0.750491599241893          2.36492120226224
```

Se definieron 6 interfaces y una malla de 3 puntos en latitud y 19 en longitud (FMTOMO agrega celdas exteriores automáticamente para poder realizar la interpolación en las fronteras.

Tras estas cuatro lineas, están definidas las interfaces del modelo completo en 3D siguiendo un orden sencillo, pero de cuidado. Si el modelo tiene dimensiones de M elementos en latitud y N elementos en longitud, las interfaces se describen $M+2$ bloques, donde cada bloque M_i representa una interface descrita por $N+2$ elementos, donde cada elemento N_i representa la "altura" a la que se encuentra ese nodo de la interface. Es de resaltar que se dice "altura" porque el valor de cada nodo se obtiene como la diferencia entre el radio de la tierra y la profundidad a la que se encuentra cada nodo. De esta manera un nodo ubicado a 100m de profundidad se encontraría escrito en interfaces.in como $6372.3 - 10 = 6362.3$.

Conociendo la manera en que se encuentran distribuidos los nodos de las interfaces del modelo, se puede proceder entonces a modificarlas interfaces de interés para que el modelo se acerque más a la referencia. Como ejemplo se muestra la modificación a la interface 2 para que pase de ser como en la Figura B-4 a como es en la Figura B-5.

Figura B-4	Figura B-5
6362.39932	6372.10000
6361.07571	6372.10000
6363.75008	6372.10000
6359.75601	6372.10000
6372.40401	6372.10000
6351.39432	6347.34444
6353.26159	6322.60556
6348.98636	6324.86667
6352.64663	6332.12778
6341.47705	6325.38889
6339.40861	6326.65000
6338.35822	6337.91111
6345.71314	6332.17222
6327.99275	6334.43333
6351.60379	6333.69444
6326.80815	6330.95556
6317.41385	6330.21667
6338.34588	6331.47778
6328.23185	6330.73889
6326.89751	6327.00000
6332.24494	6320.26111

Table 7: Modificación de la segunda interface.

De manera similar se realiza el ejercicio con las interfaces 3 y 4, hasta obtener el modelo observado en la Figura B-5.

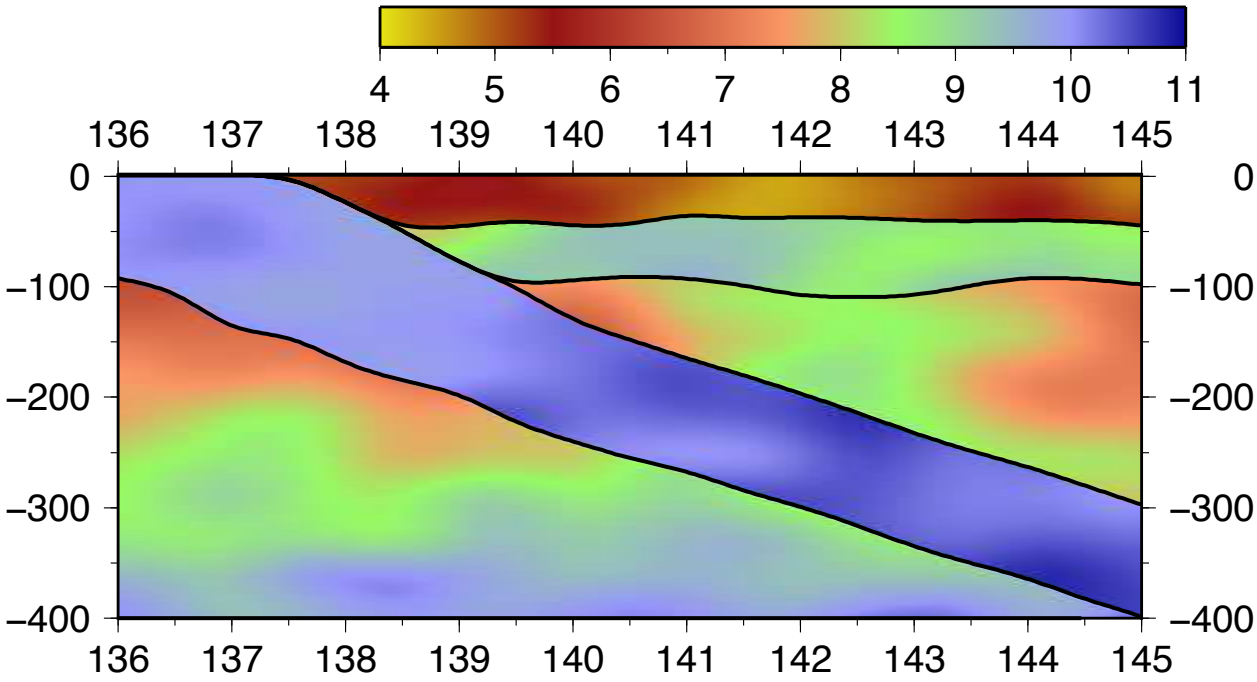


Figure B-5: Plot Este-Oeste de un modelo de velocidades complejo (posiblemente mostrando una zona de subducción sintética

Como es evidente, modelos complejos requieren un mayor cuidado, en especial si se tienen variaciones tanto en latitud como en longitud.

APPENDIX C

SCRIPT USED FOR MODELLING: "MODELAR.SH"

```
#! /bin/bash
#####################################################
# PASOS PARA CREAR UN MODELO SINTTICO
#####################################################
# 1. Crear la carpeta ../Ejemplo/mkmodel
# 2. Crear o copiar el archivo arraygen.in
# 3. Crear o copiar el archivo grid3dg.in ---> Modelo Real o Modelo inicial
# 4. Crear o copiar el archivo moddata.in
# 5. Crear o copiar el archivo "fuentes.in"
#   Que define la ubicacion de las fuentes
#####################################################
# 6. Editar archivo arraygen.in
# 7. Ejecutar arraygen
arraygen
# Genera: arraygen.out
#####################################################
# 8. Editar archivo grid3dg.in
# 9. Ejecutar grid3dg
#cp grid3dgtrue13.in grid3dg.in
grid3dg
# Genera:  vgrids.in
# interfaces.in
# propgrid.in
#cp interfacesdef5.in interfaces.in # INTERFACES MODIFICADAS A MANO
#####################################################
# 10. Editar archivo moddata.in
# 11. Ejecutar moddata
moddata
# Genera:  sources.in
# receivers.in
#####################################################
# 12. Crear copiar de referencia del modelo
# (vgrids.in, interfaces.in, propgrid.in y sources.in
cp vgrids.in vgridsref.in
cp interfaces.in  interfacesref.in
cp sources.in sourcesref.in
#####################################################
# 13. Modelo Inicial
cp interfaces.in interfacestrue.in
cp vgrids.in vgridstrue.in
```