A new Methodology for Implementing Full Waveform Inversion

Emma Pearce

Supervisor: Dr Gerard Gorman

Submission Date: 10^{th} January 2017

This report is submitted as part requirement for the MSci Degree in Geophysics at Imperial College London. It is substantially the result of my own work except where explicitly indicated in the text.

Imperial College London

Department of Earth Science and Engineering

The main body of this dissertation has XXXX words.

Abstract

Full waveform inversion (FWI) is a numerical method that provides high resolution quantitative images of the subsurface. It is a computationally expensive technique and practical software implementations are typically complex and extensive. The new methodology presented here allows FWI to be implemented with a user friendly notebook style interface that is adaptable allowing parameters to be changed and different models easily inverted utilising the inversion algorithms such as those described in *Guasch* (2012), in contrast to current method that require a detailed knowledge of partial differential equation discretisations and how to implement them. This is done using a symbolic maths engine (SymPy) through Devito, a domain specific language that enables geophysicists to write and compiles complicated mathematical equations in only a few lines of code whilst being run through a Jupyter Notebook interface. Allowing a high level language to be ran fast. The new methodology is run using parallel computing, meaning it is simple to adapt the technique to industry use through increasing the number of engines available. This methodology therefore provides a step in the right direction for FWI to become common practice within industries such as the oil and gas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Full Waveform Inversion (FWI) is a numerical method whose goal is to create high resolution quantitative images from seismic data. It is a mix of the most successful parts of seismic reflection and seismic refraction, providing data that can specify information of the geometrical distribution of geological features along with quantitative images of the Earth's physical properties such as velocity and density (*Tarantola*, 2005).

The wave equation, an equation that accurately describes wave propagation in a complex medium (*Mulder and Plessix*, 2004), is used to predict seismic data from a best-guess starting model and aims to find a solution that matches the raw seismic data trace-by-trace. Achieved by improving the subsurface model through iteratively solving a minimisation problem, where it is the difference between the observed data and the simulation that is being minimised (*Warner et al.*, 2013).

Historically, seismic techniques have been used as a method of imaging the Earth's interior, with the concept first being introduced by Viktor Ambartsumian in the latter part of the 1920s (*Guasch*, 2012). Although at the time the paper was published presenting the idea that "a homogeneous string is uniquely determined by its set of oscillation frequencies " (*Ambartsumian*, 1998) was largely ignored, 15 years later it begun to attract the attention of scientists. Since these early attempts, FWI has evolved and is now the preferred seismic method amongst geophysicists due to its high degree of spatial resolution and cost effectiveness (*Pratt*, 1999). FWI is now being expanded to dimensions that better represent real life examples and in principle, it can be used to recover any physical property that is a parameter of the seismic wavefield (*Warner et al.*, 2013).

Guasch (2012) expanded FWI to three dimensions with elastic properties being taken into account,

however because of the complex algorithms used in FWI and the need for computational performance, developing a new method is a difficult and laborious task, particularly when programming in a low level language such as Fortran. Although successful, the implimentation is complex, computationally expensive and difficult to extend further. In this work, FWI has been implemented in Python using the domain specific language compiler Devito (*Lange et al.*, 2016) (*Warner et al.*, 2013). This work follows the numerical acoustic framework to develop and apply wave formulation by *Guasch* (2012). This significantly reduces the complexity of the FWI implementation whilst still parallelising and optimising the software achieving the same or better performance as carefully hand tuned codes, in essence creating a new methodology to run FWI. Utilising a simple to use Jupyter Notebook [1] that aims to make the running of FWI on different models easier to execute and enabling parameters to be accessible and easily changed.

This is the first time that both FWI and reverse time migration (RTM) have been executed in this way. The implementation of a new methodology is a laborious task that encounters many unexpected errors, therefore the examples of FWI in this work apply to synthetic 2D small-amplitude pressure waves propagating within an inhomogeneous, isotropic, non-attenuating, non-dispersive, stationary, fluid medium. Although possible to expand this model to include 3D, anisotropy, attenuation and elastic effects, due to the difficulty of creating a new methodology these have been ignored to simplify the process as much as possible.

All codes utilising Devito in this dissertation can be found via the OPESCI github website[2].

1.1 Background

1.1.1 Devito

Devito utilises a finite difference Domain Specific Language (DSL) to allow solvers to be implemented at a high-level using symbolic python. Python and NumPy provide the glue for implementing FWI while complied C code is generated for the actual solvers. Devito has been used to write programs to execute FWI and RTM. They both allow the integration of external synthetic velocity models and the subsequent migration/inversion calculated (*Lange et al.*, 2016) to produce reconstructions of the Earth's subsurface.

[1] http://jupyter.org/
[2] https://github.com/opesci

1.1.2 Reverse Time Migration

RTM is seen as the precursor to FWI. The wave equation is used to solve for velocity models of complex geological features (*Yoon et al.*, 2004). RTM is currently recognised as the most common method of seismic modelling as it offers an acceptable compromise between computational power and image quality, as it is only required to sum the gradients, (compared to FWI due to it being an iterative method).

To aid in the development of FWI in Devito, the synthetic model will first be tested using RTM as this is less computationally expensive, therefore highlighting implementation errors particularly those associated with the gradient calculation, enabling changes to be made more rapidly as it is written in a high level language. Once the initial model has been converted to the Devito format and successfully integrated into the RTM code, it can then be modified to create a basic FWI implementation.

1.1.3 Full Waveform Inversion

FWI is based on the principle that seismic wave energy propagation through the Earths surface can be approximated by the a wave equation that has oscillatory solutions, such as that shown in Equation 1.1. This equation has analytical solutions for relatively simple models, where the model is able to describe the medium in which the wave is travelling in through the medium's physical properties. The condition being that to be able to model the seismic waves the model's physical parameters must be able to be represented by a mathematical analytical function with some properties that are dependent on the wave equation media parameters.
In its most simple form the wave equation used by FWI can be represented by

$$\frac{1}{c^2}\frac{\partial^2 \rho}{\partial t^2} - \bigtriangledown . \left(\frac{1}{\rho^2} \bigtriangledown p\right) = s \tag{1.1}$$

Where $p(x,t)$ is acoustic pressure, $s(x,t)$ is a source term, and $c(x)$ the speed of sound in the medium and $\rho(x)$ the density.

The problem is that the geometrical distribution of media parameters in the real world cannot be described by an analytical function, removing the possibility of solving the wave equation analytically,

resulting in the need for using a numerical method to model seismic waves (*Guasch*, 2012).

1.2 Inverse Theory

1.2.1 Inverse Method

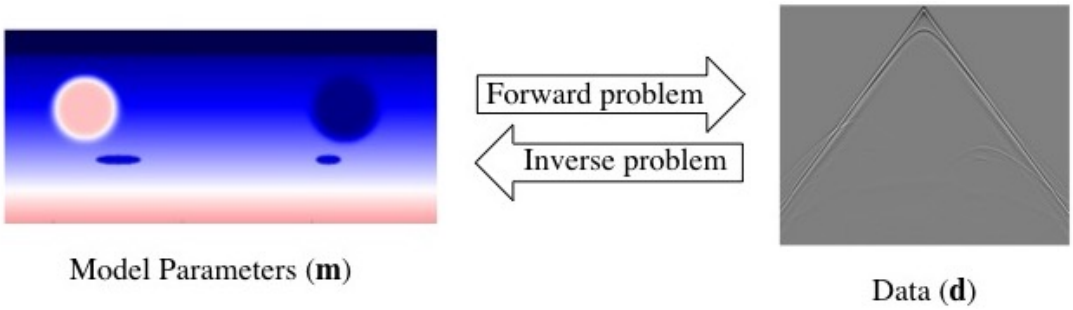

Figure 1.1: Visual representation of the relationship between physical quantities (model parameters) and real observations (data) and how they relate to the process of having a forward and an inverse problem. The model parameters are seismic velocity, and the data is a pressure seismogram.

A forward problem as shown by Figure 1.1is defined where a starting model made of physical quantities is used to generate the predicted wave-field (observed data). This observed data model is a straightforward application of the governing laws of physics applied to a set of pre-defined variables that define the system being studied(*Guasch*, 2012). If inversion were performed on this, a unique solution to the wave equation is obtained through the use of the laws of physics, as all the required parameters are pre defined. This is usually represented by the discrete wave equation,

$$\mathbf{p} = G(\mathbf{m}) \tag{1.2}$$

Although the wave equation represents a linear relationship (Equation 1.1), it also represents the non-linear relationship shown in Equation 1.2 where G is an operator representing physical laws (e.g the wave equation), that relates $\mathbf{p}$ the data (the predicted seismic wave-field) to the model parameters $\mathbf{m}$ that describes the subsurface (*Warner et al.*, 2013).

Unlike the forward problem, the inverse problem will not usually have a unique solution (non-deterministic).

The related non-linear inverse problem corresponding to equation 1.2 is

$$\hat{\mathbf{m}} = G^{-1}\mathbf{d} \tag{1.3}$$

Where $\mathbf{d}$ now represents the observed field data and $\hat{\mathbf{m}}$ is the model for which we are trying to solve.

1.2.2 Objective Function

The difficulty of inversion arises as G is not a matrix, it is a non-linear function that describes how to calculate the wavefield $\mathbf{p}$ given the model $\mathbf{m}$ and is dependent on the source. Therefore it is not possible to invert G without knowing all the contributions of the various terms (which is what we are trying to calculate in the first place), furthermore being non-linear prevents a system of linear equations being able to solve the forward problem using numerical methods. To overcome this, an approximate solution to Equation 1.3 is obtained which is improved using iteration. A starting model $\mathbf{m_0}$ is chosen that is reasonably close to the true model. It is required that it is close to the true model in order for the iterations to converge towards the correct solution to the equation. A residual data set $\Delta\mathbf{d}$ is then defined as the difference between the predicted and observed data $\Delta\mathbf{d} = \mathbf{p\prime} - \mathbf{d}$ where $\Delta\mathbf{d}$ is a function of the starting model, $\mathbf{p\prime}$ is the data set that Equation 1.2 predicts and $\mathbf{d}$ is the observed data set. This will then be used to create a new model that minimises the difference between the observed and simulated data. An objective function (f) is defined by,

$$f = \|\mathbf{p\prime} - \mathbf{d}\|_p = \|G(\mathbf{m}) - \mathbf{d}\|_p \,. \tag{1.4}$$

It is a scalar quantity that is equal to the sum of the squares of $\Delta\mathbf{d}$. A value of zero of f would indicate that the observed data and the predicted data are identical, thus the inversion has been successful. Because this is an undeterministic problem, the goal of the inversion is to find a model that minimises f where $\|\|_p$ is the L^2 norm. If we consider the set of f values for every possible subsurface model a'surface' can be created, where the global minimum of this hyper-surface f is identified as the solution to the problem (*Han et al.*, 2014). The solution space defined by f will often have more than one minima, therefore, it is important to have a reasonable initial guess to minimise the risk of falling into a poor local minima when methods such as the steepest descent are used, explaining why $\mathbf{m_0}$ must be reasonable close to the true model.

1.2.3 Gradient

If a linear relationship between changes in the model and the corresponding residual is assumed, then the change that should be made to the starting model are represented by,

$$\Delta \mathbf{d} = -\mathbf{H^{-1}} \frac{\partial f}{\partial \mathbf{m}} \tag{1.5}$$

Where $\mathbf{H}$ is the Hessian matrix containing all second-order differentials of f and $\frac{\partial f}{\partial \mathbf{m}}$ is the gradient of f (both with respect to all model parameters). As $\frac{\partial f}{\partial \mathbf{m}}$ cannot be calculated directly the adjoint method is used, provided by *Tarantola* (1984) (Appendix A) whereas $\mathbf{H}$ is much harder to calculate due to its size, therefore only the diagonal elements of it are retained. The approximation of the diagonal elements of $\mathbf{H}$ are now possible to calculate using spatial preconditioning where h_{ii} corresponds to model parameter m_i. A final equation representing the model update is then represented by,

$$\Delta m_i = -\frac{\alpha}{h_{ii}} \frac{\partial f}{\partial m_i} \tag{1.6}$$

where α is the step-length, a scaling factor.

When using FWI with a real dataset a starting model is needed along with a source wavelet. To do this the adjoint is back propagated from the receivers by treating them as sources and producing a residual wavefield. This is then cross-correlated with the source wavefield (used to generate the simulated receiver data) in the time domain for each point in the model producing a gradient for each source. These gradients are then stacked (i.e. summed) together to produce the global gradient $\frac{\partial f}{\partial m_i}$ from Equation 1.6 which tells us the direction of steepest decent that is assumed to be the direction the model should be adjusted in (*Pratt et al.*, 1998).

1.2.4 RTM as a precursor to FWI

When running RTM the methods shown above are utilised to calculate a gradient with the key difference being that the residual wavefield is back propagated in FWI in comparison to RTM where it is the entire wavefield. Additionally RTM generates a reflectivity image from the cross-correlation whereas FWI the cross-correlation imaging conditions are in velocity space (*Warner et al.*, 2013).

1.2.5 Step length

The step-length is the final step needed for FWI, calculated by adjusting the starting model by a small amount of α and observing the effect this adjustment has on the residual (*Pratt et al.*, 1998). From this the optimum factor of α by which to adjust the model in order to minimise $\mathbf{\Delta d}$ is obtained using,

$$\alpha = \frac{(\Delta d^0)^\dagger(\Delta d^0 - \Delta d^1)}{(\Delta d - \Delta d^1)^\dagger(\Delta d^0 - \Delta d^1)} \tag{1.7}$$

where Δd^0 and Δd^1 are the residual from the starting guess model and the first perturbation model respectively. Showing that α only depends on the already calculated residuals at the two different points and therefore its optimum value can be directly computed using only one extra forward model to determine Δd^1 (*Guasch*, 2012). Shown visually in Figure 1.2. Therefore adjusting the model by this calculated value of α will scale it closer to the minimum. The work flow followed by FWI is shown in Figure 1.3 giving a summary of how each part fits into the process of obtaining a reconstruction of the subsurface.

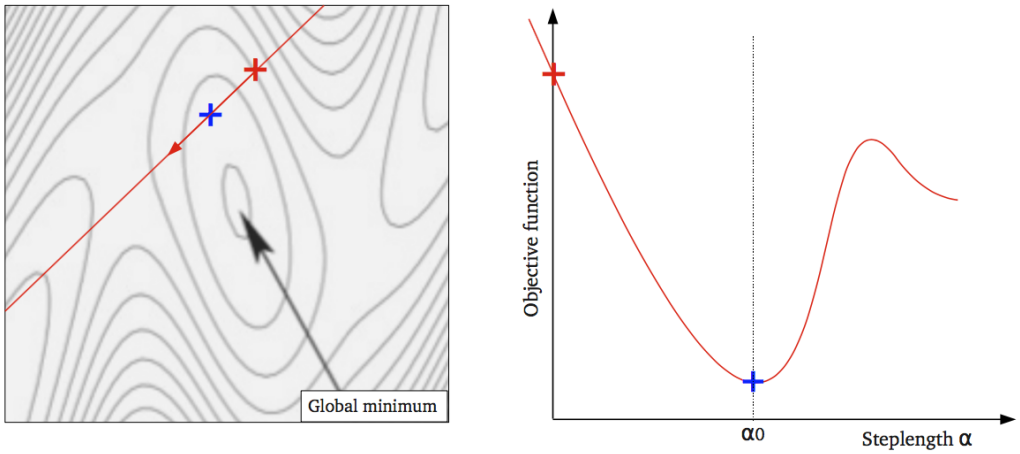

Figure 1.2: The solution space where the minimum is assumed to be the global minimum. The red cross is representative of Δd^0 and the blue cross the minimum of f along the direction of the gradient. Δd^1 would be a point between the two crosses that is able to assume the residual has a linear relationship. Image b shows the residual values along the gradient direction where the minimum value of f is reached when the model has been scaled by a total step length value equal to α_0. Image taken from *Guasch* (2012).

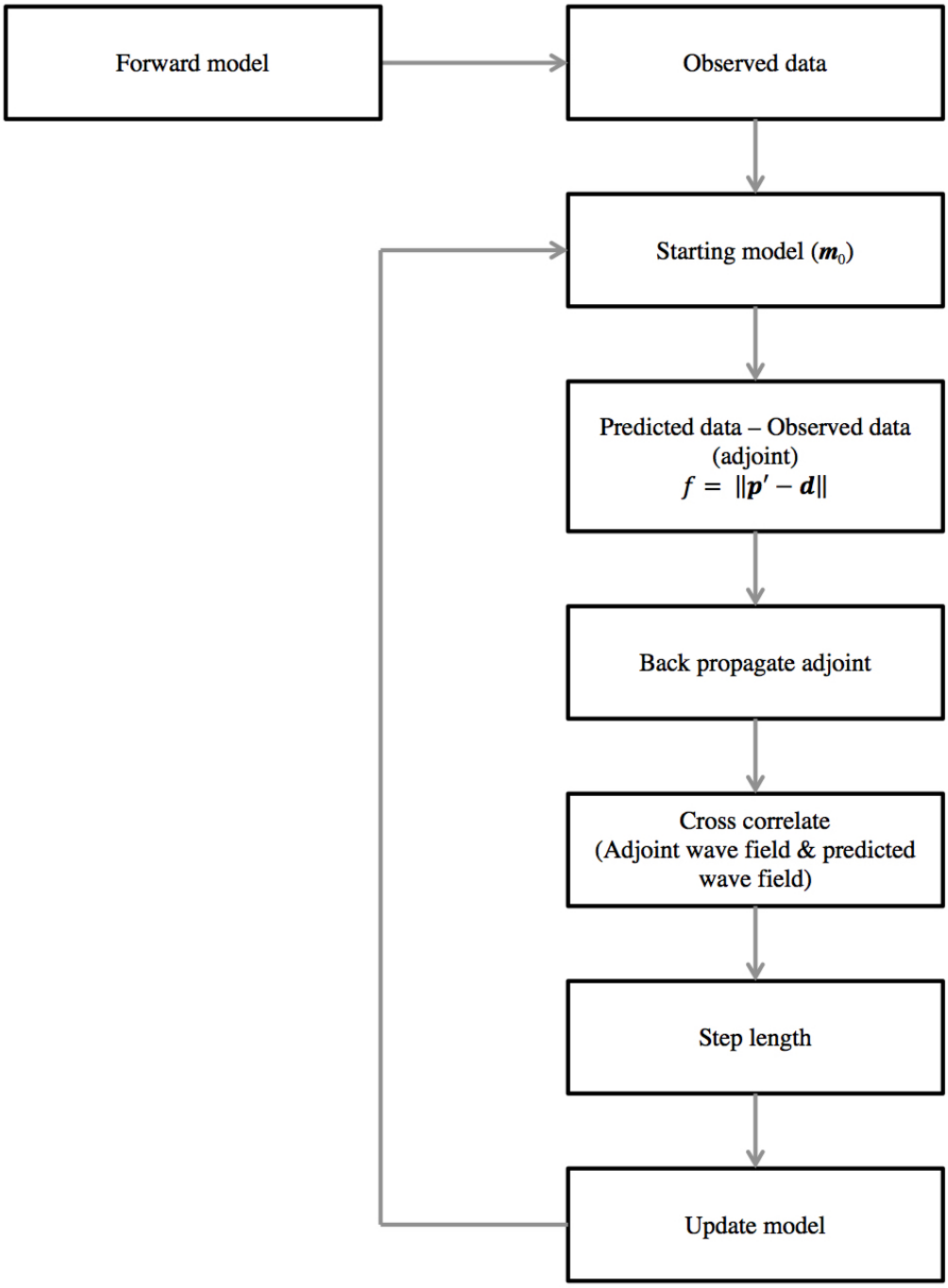

Figure 1.3: The work flow followed by FWI when undergoing iterations to reduce the objective function f.

1.2.6 Summary

Exploration seismology is by far the most important geophysical technique used in industry due to its higher accuracy, resolution, and penetration than other methods (*Sheriff and Geldart*, 1995). Currently within industries such as the oil and gas methods such as RTM are used to map the geological structures rather than identifying the hydrocarbons directly in the material properties.

FWI is desirable as it is able to take this method further and recover subsurface properties including density, from only one parameter; in this case, velocity. Thus allowing the detailed reconstruction of the subsurface without the need to drill wells or boreholes, which is currently what is needed if you want to be able to relate the lithology to the structure. This therefore provides the motivation behind the current project to allow FWI to become common practice within industry with a simpler methodology that uses less computational resources to perform inversion on real life data sets.

Chapter 2

Methodology

2.1 Jupyter notebook

All of the algorithms shown are run through Jupyter Notebook. This is a web application that allows code to be run and edited 'live' in a notebook style interface, intended to be a user friendly application.

2.2 Devito

Equations used have been written using Devito, a domain specific language (DSL) designed to make the software implementation look similar to the equations being solved. As an example, the forward wave equation with absorbing boundary conditions can be written as:

$$\eta \frac{du(x,t)}{dt} + m\frac{d^2u(x,t)}{dt^2} - \Delta^2 u(x,t) = q \tag{2.1}$$

where u denotes the pressure wave field, m is the square slowness, q is the source term and η denotes the spatially varying dampening factor used to put an absorbing boundary condition in place. which can then be represented in Devito by:

```
wave_equation_forward = m*u.dt2 - u.laplace + damp * u.dt
stencil_forward = solve(wave_equation,u,forward)[0]
```

where Devito's dt, dt^2 and laplace operators are used to perform the stencil expansion according to the spatial and temporal discretization order and the damping factor is dealt with via a separate expression. The resulting expression is then rearranged using the SymPy solve routine to show forward propagation, from which the corresponding operator can be created and executed (*Lange et al.*, 2016).

Devito can be installed manually using

```
git clone https://github.com/opesci/devito.git
cd devito && pip install --user -r requirements.txt
```

2.3 Initialising Model

A synthetic model is created using the script `Phantom Synthetic Model` in Appendix B producing Figure 2.1 with the model's parameters shown in Table 2.3. From this velocity anomalies are added producing Figure 2.2. The model has been based on the analogy of the North Sea, with sea depths of 300m and negative refractors imitating gas seepages. Figure 2.1 will be set as the starting guess ($\mathbf{p}\prime$) when running inversion. The inversion will be run until the final solution replicates 2.2 ($\mathbf{d}$ as closely as possible and f is at a minimum.

Paramater	**Value**
Frequency	10 Hz
Grid Spacing	15 m
Spatial order	4
number of sources and receivers (nsrc)	666
Time interval (tn)	5800 ms
Critical Time-step	2.01 ms

Table 2.1: Phantom 2D model parameters

2.3.1 Paramaters

when creating a synthetic model to run inversion on certain criteria is required to be met in order to produce a stable inversion.

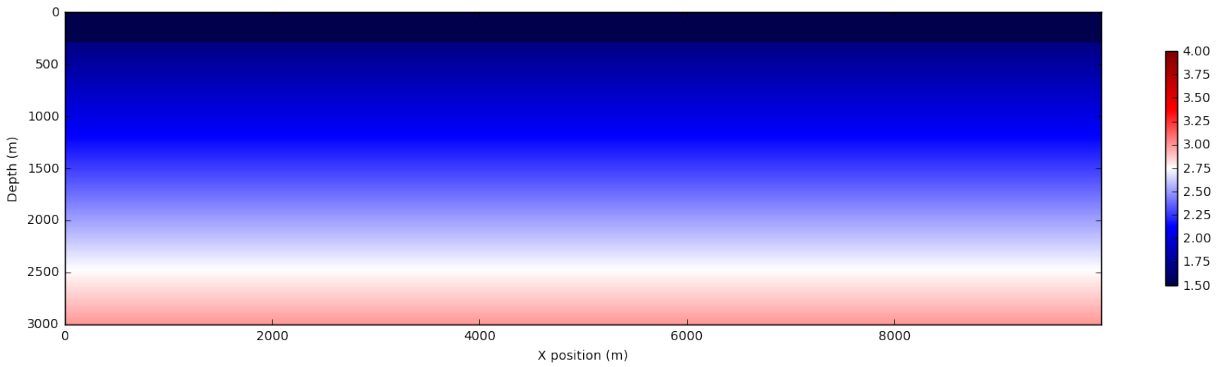

Figure 2.1: A velocity model in ms^{-1}. Initial velocities are chosen for the sea water velocity 1500 ms^{-1}). The subsurface is given a velocity gradient with the lowest velocity 1700 ms^{-1} and the highest velocity at the bottom of the mode equal to 3000 ms^{-1}. The grid spacing of the model is defined by dx which is set equal to 15m.

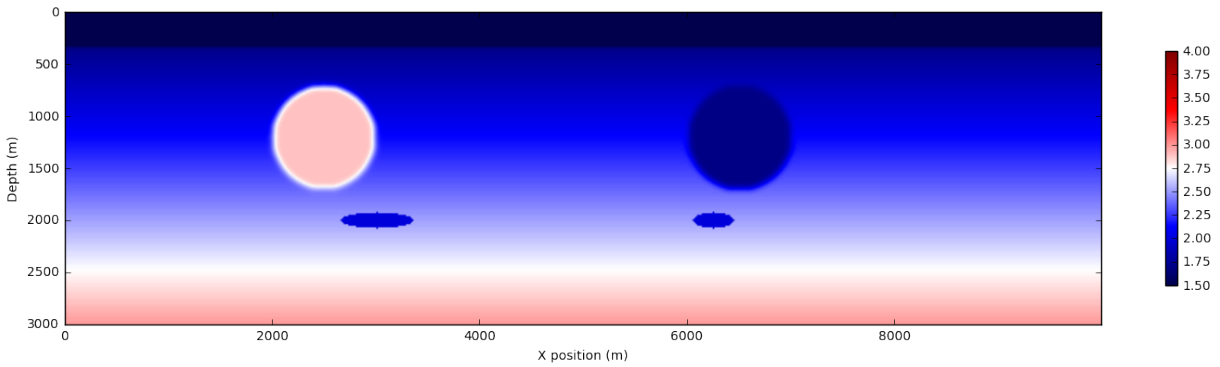

Figure 2.2: A velocity model in ms^{-1}. Velocity anomalies are added to the base model.The velocity anomalies will be what the simulated model is trying to replicate. Circular anomalies represent reflectors of positive and negative values (positive V = 2900 ms^{-1}, negative V = 1700 ms^{-1}). These have then been smoothed. Refractors with sharp edges are represented by the elliptical negative anomalies (V = 2000 ms^{-1}). Both are of different sizes and positions, again to see how well these can be replicated during inversion.

Frequency

Selected as 10Hz, a realistic value able to image to a depth of 3km but also maintain reasonable resolution, without experiencing significant attenuation. It is always aimed that the lowest value of frequency is used in order to allow the iterations to converge to the minimum quicker. The better the starting model provided (when not using synthetic data, the quality of the collected data is key in deciding the frequency) the lower the frequency can be. It is usual to aim for a frequency between 7 - 15Hz (*Guasch*, 2012).

Grid Spacing

To avoid dispersion there must be a minimum of five grid points per wavelength for a fourth-order system (a system represented by a quartic equation) (*Levander*, 1988), for the minimum wavelength. For this model the minimum velocity is 1500 ms^{-1}. Using $U_{min} = f\lambda$ where f in this instance is frequency and λ is wavelength, gives $\lambda = 100$m. Therefore the maximum grid spacing possible to avoid dispersion is 20m. 15m is chosen as a 'safe' grid spacing, low enough below the maximum to ensure dispersion does not occur. The synthetic phantom 2D model is fourth order as the lower the spatial order, the more grid points per λ required, requiring a finer grid spacing and in turn larger computational resources to run the inversion.

Critical time-step

The critical time-step is given by a function in Devito that estimates $dt_{crit} < 0.495\frac{\Delta x}{U_c}$, this condition is known as the Courant-Friedrichs-Lewy Condition and must be met in order to retain stability when running inversion. If not, then the model can become unstable (*Graves*, 1996). It requires that a time step must be small enough that a wavefield crosses no more than half a node (*Courant et al.*, 1967).

tn

The time the receivers run for, it should be long enough that the wave propogated from the first source has enough time to reach the final source, i.e. the time taken for a wave to travel from the first receiver to the last in the model after bouncing off the mid point of the bottom of the model. For the synthetic model assuming an average velocity of 2000 ms^{-1} the minimum time interval is 5.8 s.

nsrc

The number of sources to receivers. This is set at 666 to get one source/receiver per node. The amount of receivers utilised can be adjusted for each inversion. A random selection of sources can be chosen at each iteration and then not used again, this ensures all sources are utilised and the reconstruction is detailed across the whole width of the model.

tn

The time the receivers run for, chosen to be equal to the time taken for a wave to travel from the first receiver to the last in the model after bouncing off the mid point of the bottom of the model. Assuming an average velocity of 2000 ms^{-1}.This sets the minimum time interval to 5.8 s.

2.4 Marmousi

A 2D Marmousi model is also used as a more complex structure. When testing geophysical processing in industry the Marmousi model is often used, therefore it is a valued control model (*Bourgeois et al.*, 1991). The true model is shown in Figure 2.3 with the initial guess a smoothed version, shown in Figure 2.4.

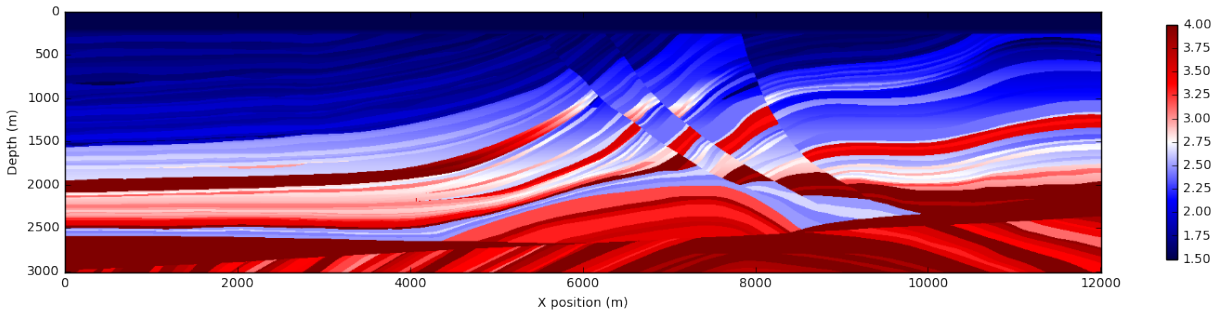

Figure 2.3: A velocity model in ms^{-1}. 2D Marmousi Model based on a profile through North Quenguela trough in the Cuanza basin, Angola. The model contains many refractors, steep dips and strong velocity gradients both vertically and horizontally (*Bourgeois et al.*, 1991)

.

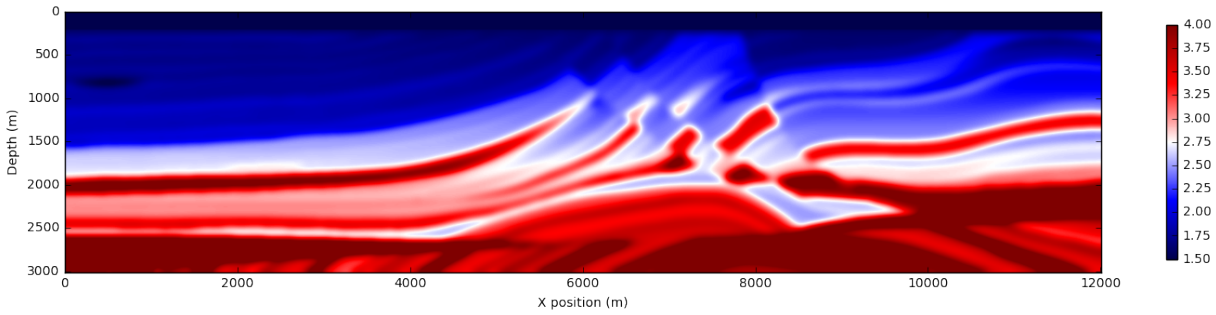

Figure 2.4: A velocity model in ms^{-1}. The 2D Marmousi model smoothed to provide a synthetic initial guess when running RTM and FWI.

2.5 Shot

A Ricker wavelet (Figure 2.6) is used as the source wavelet defined by $A = (1 - 2\pi^2 f^2 t^2)e^{-\pi^2 f^2 t^2}$ with $f = 15Hz$ as it produces a wavefield representative of a seismic wave propagating through a viscoelastic homogeneous medium (*Wang*, 2015). An example of the shot record is shown in Figure 2.5 showing a pressure seismogram from a source at the middle of the domain.

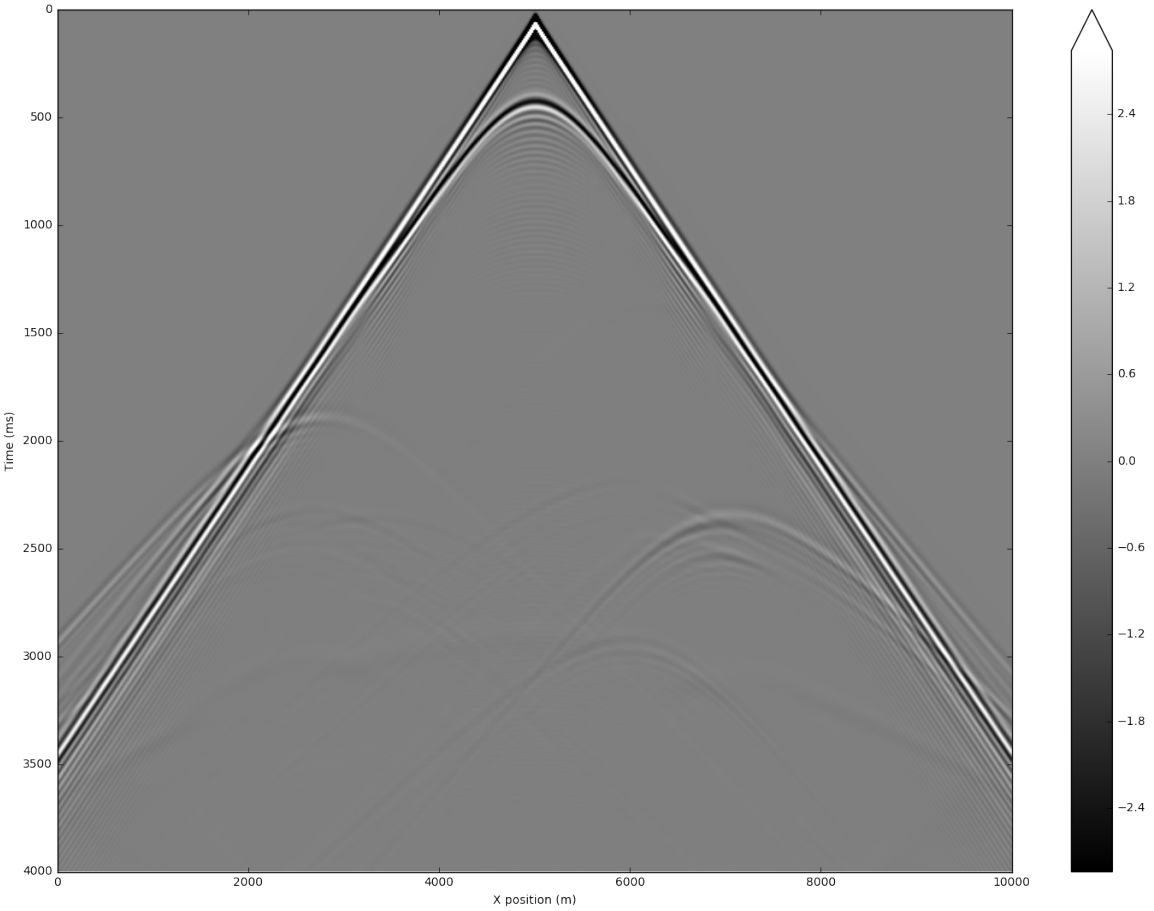

Figure 2.5: An example of the Shot record at source 330.

2.6 Ipyparallel

To run both RTM and FWI they have been parallelised using ipyparallel [1]. This allows multiple calculations to be executed at the same time by accessing a large cluster of networked multicore computer nodes usually designated to run one source per node thus reducing the time needed to calculate RTM/FWI. This method is used on a larger scale in industry meaning if the code is required

[1] ipyparallel.readthedocs.io

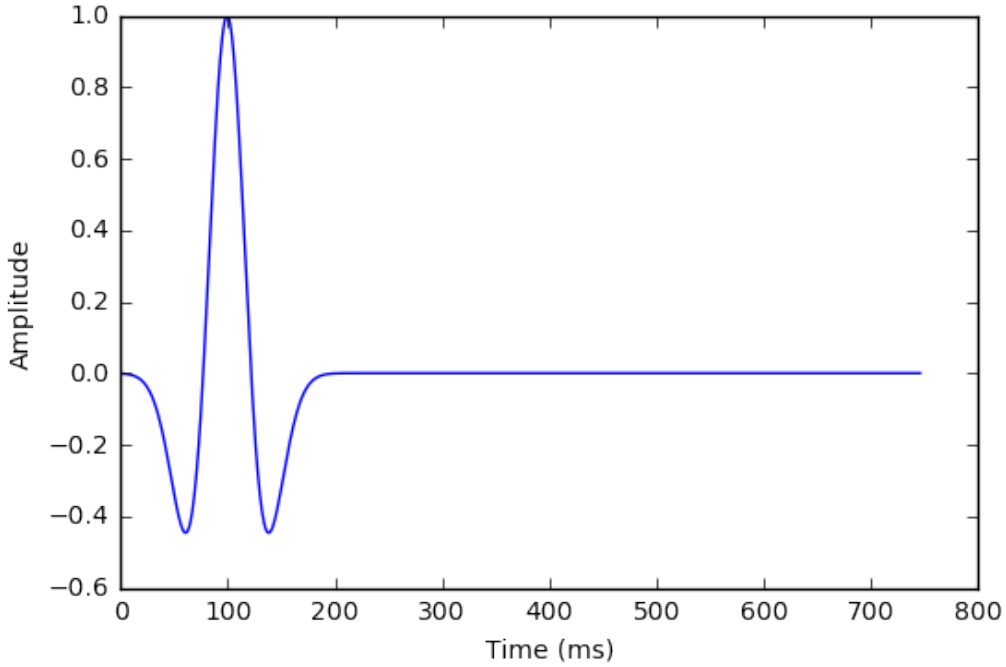

Figure 2.6: Ricker wavelet used in RTM with a frequency of 15Hz

to process higher resolution/larger images, it would just be a matter of increasing the number of engines available to run the algorithms.

To set up the parallel environment within the command window type:

```
pip install ipyparallel
ipython profile create --parallel --profile=mpi
```

Edit the file ipcluster_config.py and add the lines:

```
c.IPClusterEngines.engine_launcher_class = 'MPIEngineSetLauncher'
c.MPI.use = 'mpi4py'
```

Start the cluster with the command:

```
ipcluster start -n 12 --profile=mpi
```

where 12 dictates the number of engines being used.

2.7 RTM as a precursor

The synthetic models (2D phantom and marmousi) are used in the RTM code, once a successful migration was achieved and a velocity model produced, the models were integrated into in the FWI code.

2.8 FWI Step Lenght

Explained in Section 1.2.5 is the importance of the starting value of alpha, α_i. If the value is too large it can overshoot the minimum and settle in the wrong local minimum. It must also not be too large that a linear relationship between the starting value of the gradient and the minimum value of f along the direction of gradient cannot be assumed. Finally the value must be great enough that numerical rounding cannot interfere with the value. To settle this criteria the value has been taken as 1% of the gradient normalised by the maximum value of the gradient.

2.9 FWI models

When running FWI the number of sources used could be selected. This project uses a random sample of 5 sources for each iteration for the phantom 2D model. The number of iterations had to be larger as fewer sources were used. This was chosen to be a minimum of 100 and a maximum of 300. This enabled the whole array of sources to be used when running the algorithm without the need to utilise them all at once.

Chapter 3

Discussion

Although FWI has the universal aim to reduce the objective function, this can be achieved using a variety of different methodologies. Here, the chosen practice is discussed and evaluated.

3.1 Gradient Methods

When considering the method used to obtain the 'optimum' value of the step length as discussed in Section 1.2.5 when calculating the steepest descent, either first or second-order methods can be used to find the solution of a large-scale non-linear optimization problem (*Métivier and Brossier*, 2016).

Global or local descent algorithms are the two types of method available to solve this exact problem with the differences shown in Table 3.1. With the Phantom 2D model having grid spacing of dx = 15m this gives the model 133,200 parameters $(\left(\frac{lx}{dx} \times \frac{lz}{dx}\right) = (666 \times 200) = 133,200)$ making using a global method implausible. Therefore local descent methods offer far fewer iterations than the number of parameters to provide an acceptable sampling size meaning the computation is far cheaper.

Amongst possible local descent algorithms the simplest are first order. These use only the gradient to define the descent direction, such as in RTM where the final image is created through only the summation of the gradient. The two most common first order methods are the Steepest descent and the Nonlinear Conjugate gradient. Second-order methods are defined through the use of a second-order derivative used to approximate the inverse of the Hessian matrix, consisting of the quasi-Newton l-BFGS method and the Truncated Newton method.

Table 3.1: Comparison between global and local methods used to find the solution to a large-scale non-linear optimization problem (*Métivier and Brossier*, 2016).

Method	Description	Advantage	Disadvantage
Global	Find the global minimum of f from any given starting point through global convergence. Achieved through a guided-random sampling of the parameter space.	Guaranteed to find the global minimum of the objective function.	Number of discrete parameters cannot exceed a few tenths to hundreds as even modern high performance computing (HPC). platforms cannot evaluate f in a reasonable time. In FWI the number of unknown parameters is at a minimum an order of magnitude greater rendering global descent unfeasible.
Local	From a chosen starting point the closest minimum of f is converged towards. Following an initial guess a series of iterations are run successively updating the model until the minimum is found.	Requires less computationa resources as it is only running a monotonic decrease of the objective function	The global minimum is not always found. If the initial starting guess is not close enough to the, true model it is possible to converge into local minimum.

As all methods ensured the decrease of the objective function and guaranteed convergence towards the local minimum it is simply a decision of which method will provide the optimum result whilst using the minimum amount of computational resources. Presented below is a summary of each method utilising the work of *Métivier and Brossier* (2016).

Steepest Descent

The steepest Descent (SD) is the simplest method that can be used to calculate the gradient, where if P the preconditioned (Abstract A) is not defined then Δx_i is just the opposite of the gradient of the objective function with respect to the model $\frac{\partial f}{\partial \mathbf{m}}$.

$$\Delta x_i = -P\frac{\partial f}{\partial \mathbf{m}} \tag{3.1}$$

Advantages of this method are P is not required and it is computationally inexpensive. However a high number of iterations can be required to reach the minimum value of f. When using this method with the Marmousi model and P obtained using *Shin et al.* (2001) (Appendix **??**) it provides good results, converging relatively quickly but providing a maximum depth of approximately 4.5km for detailed reconstruction.

Nonlinear Conjugate Gradient

The nonlinear conjugate gradient (NCG) method is a linear convergence minimization algorithm for quadratic functions where the update to the model is computed as a the opposite of the gradient as in SD but in addition to this it also uses the descent direction computed in the previous iteration

$$\Delta x_0 = -Pf\Delta x_i = -P\frac{\partial f}{\partial \mathbf{m}} + \beta_i \Delta x_{i-1} \tag{3.2}$$

where β is a scale parameter calculated by *Dai and Yuan* (1999). Although able to reduce the number of iterations required in comparison to SD the method is unpredictable and can often cause erratic behaviour and for cases such as the Marmousi it is less efficient than the steepest descent.

Quasi-Newton l -BFGS

A second order method that is utilised due to its simplicity and efficiency (*Nocedal*, 1980) is the Quasi-Newton.

$$\Delta_i = -Q_i\frac{\partial f}{\partial \mathbf{m}} \tag{3.3}$$

Where Q_i is the approximation of the inverse Hessian calculated using finite difference. The similarity between this and the SD method is notable as Q now substitutes as the preconditioner P. When calculating FWI this method can achieve a superlinear convergence rate meaning it is faster than SD and NCG. It can combine a diagonal P and Q to provide the best computational efficiency amongst the different optimisation methods, supported by its use on the 2D Marmousi model. It is also able to reconstruct higher resolution images to a depth greater than the first order methods.

Truncated Newton

This second order method does not calculate an approximation of the Hessian (Q) such as in Quasi-Newton, instead it computes an inexact (truncated) solution for the linear system associated with the Newton equation. This allows FWI problems to be solved at a reasonable computational cost using,

$$\mathbf{H}\Delta x_i = -\frac{\partial f}{\partial \mathbf{m}} \tag{3.4}$$

This enables the system to converge faster than Quasi-Newton but at a higher computational cost, but both second order methods remain competitive on all other benchmark problems *Castellanos et al.* (2015).

After comparing the different minimization algorithms above it was concluded that the steepest descent method would be used for FWI calculations on both the Phantom and Marmousi model. It is possible that when testing the methods on Marmousi that the second order methods were seen to out perform first order due to the initial model being sufficiently close to the true model (as it is just a blurred version of the true Marmousi). When running inversion on the Phantom model this was not the case and therefore the difference between first and second order algorithms is less notable and does not justify the additional computational cost. As both first and second order methods would reconstruct to a good level of detail up to a depth of 4.5km, with the Phantom model only being 3km deep again, the need for second order was not necessary.

3.1.1 Mask

FWI is unable to tell the difference between water in a model that has a set value and does not need to be updated, and the subsurface that needs to be constantly updated. Therefore when running iterations a mask is put in to prevent the water velocity being changed. To do this any velocity value below 1501 ms^{-1} has the gradient set equal to zero.

3.1.2 Box constraints

Perturbations can occasionally cause anomalously high or low velocity updates when trying to reduce f. In order to prevent this a mask can be used which limits any values that are unrepresentative of realistic geophysical constraints being included into the model. In this case velocities below 1500 ms^{-1} as this would be slower than the water velocity and above 4500 ms^{-1} are set a priori.

3.2 Inversions

Using the Phantom 2D model with 5 random shots utilised for 300 iterations Figure 3.2 was produced with a value of α equal to 1.66 and $f = 130605$. From the value of the functional it is clear that the

reconstruction is still a long way from being close to the true model but the shapes of the positive and negative anomalies are beginning to show. The high value of the functional is usually associated with cycle skipping, suggesting the starting model is not close enough to the true model to converge towards the correct minima. This can be compared to Figure **??** that was run for only 100 iterations with f = 163660 and α =0.16 again reinforcing the fact that although f was decreasing, it is significantly far from a value of zero.

Figure 3.3 shows a simple Marmousi 2D model with f = 1.8819e+07 and α =1.31842 for 20 iterations using 100 shots. The value of the functional is much smaller with f converging towards zero. This also highlights the need for a preconditioner as the values close to the source/receiver are much higher than those further away.

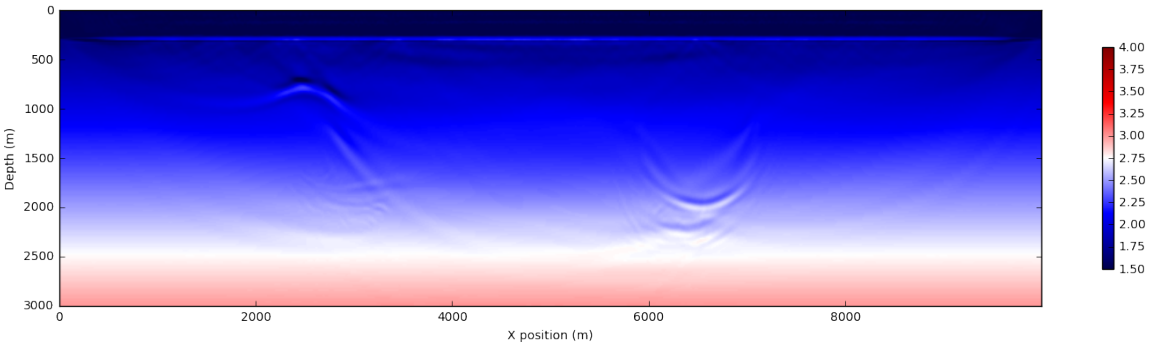

Figure 3.1: Phantom 2D model with 5 shots used per iteration for 300 iterations

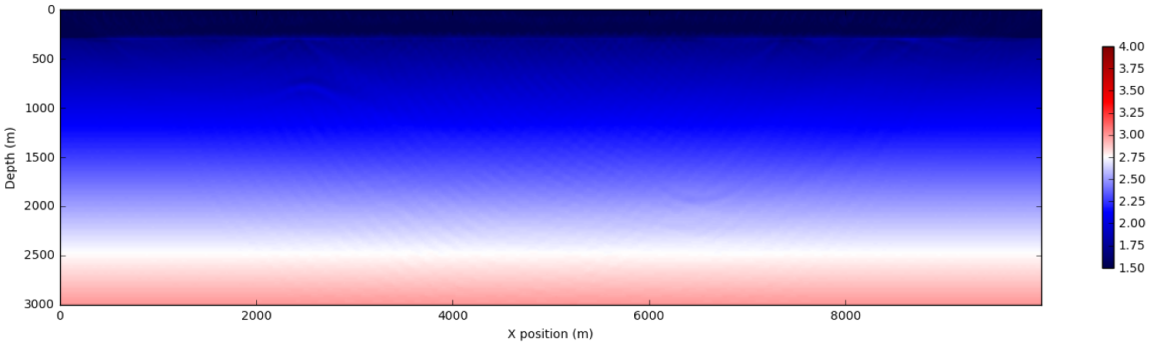

Figure 3.2: Phantom 2D model with 5 shots used per iteration for 100 iterations. It is clear to see how the velocity anomalies have been reconstructed to a lower standar than those in the model run for 300 iterations

3.3 Limitations and Improvements

Although through the use of the new methodology that uses Devito, FWI is faster to implement and simpler to run, limitations are still present that are a common occurrence in other methodologies.

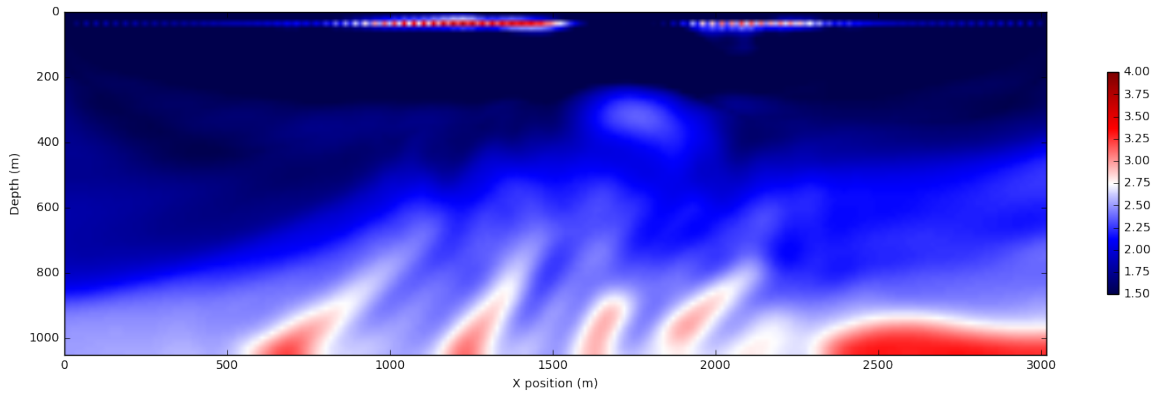

Figure 3.3: A simple marmousi 2D model run for 20 iterations and using all 101 shots

3.3.1 Heuristics

When running FWI heuristics are put in place in order to produce the optimum reconstruction, below are those that were not included in the current version of the FWI algorithm, but are recommended as future adaptations.

Cycle Skipping

If the starting model used as the initial guess for FWI is too far form the original data set then a phenomena known as cycle skipping can occur as seismic data is oscillatory and FWI is a local inversion scheme. FWI uses a perturbs model that immediately reduces the mismatch between observed and predicted seismic data. Assuming the starting guess model is good enough, and the frequencies used are low enough then the following iterations will move the model in the right direction, towards the global correct minima. However if the initial guess is more than 180° out of phase with the true data then the first arrival of the model data can be matched to the wrong arrival of the true data. This can either cause the model to converge towards a local minima or the value of f will not reduce, and possibly get bigger as the perpetuated guess moves further away from the true model (*Shah et al.*, 2012). Currently cycle skipping is one of the main limitations of FWI preventing its wider adoption within industry use and although the process can be spotted manually by comparing predicted and observed data, this is an ambiguous subjective method. The work of *Shah et al.* (2012) provides a "a robust and objective means " to mitigating this problem via the spatial continuity of windowed, low-frequency, phase differences. The implication of such a method should be mandatory especially when using Devito to ensure a reliable inversion algorithm.

Hessian Preconditioning

As the energy of the sources propagates out it is dispersed over a larger area. The models run using FWI in this paper are exempt of attenuation, as density has been assumed to be constant. However spherical spreading occurs regardless of the model properties. In terms of FWI what this means is that the parts of the model further away from the source are imaged worse than those close to the sources as they are being imaged with lower amplitude. In 2D the ratio between energy decay per unit area is proportional to the square of the distance between the source and the wavefield ($A \approx \frac{1}{r^2}$) (*Guasch*, 2012). To account for this a Hessian preconditioned can be used. This is a matrix that is able to boost frequencies that have lost energy through spherical spreading and damp those close to the source. This is an optional method that when combined with the Steepest Descent gradient is very successful.

Gradient Function

The Gradient function used was the step-length, a first order linearization method. A simple yet effective improvement that can be made to the FWI program is the implementation of a second order gradient method. The first order method currently used is comparable to a second order when a Hessian preconditioner is used in conjunction. The current FWI code does not have a preconditioner attached to it, therefore the use of a second order method would improve the convergence of the objective function.

3.3.2 Computational

Computational memory

Inversion is a computationally demanding algorithm, and can therefore quickly become too big to deal with when using parallel computing. The 2D Marmousi model has a grid of approximately 200 by 50. This is sufficient to enable FWI to run utilising all the shots and receiver (101) for a sufficient number of iterations to provide an inversion. The Phantom 2D model with a grid spacing of 666 by 200 is unable to run below 125GB of computational memory with greater than 5 sources being used (out of a total of 666). Specifically when parallelising the code using the Devito function `dview.map()` although enabling it to run quicker, the computational cost is far greater. To mitigate this issue only

the `map()` function was used with the Phantom 2D model, although increasing the time taken to run, more shots per iteration could be used. A simple solution to this problem is allowing access to more engines when paralyzing the code.

Limitations of the notebook

When loading the synthetic model that the inversion will be run on, issues can be encountered spanning from the format. The FWI program is aimed at being an easy to implement structure that any model can be placed into, however this means the model must have certain criteria to fit the pre defined inversion structure such as being the transpose of the true model. In addition to this, parameters are intended to be simple to change including frequency, grid spacing and time step. However due to the use of Jupyter Notebook, when changes are made the cluster must be stopped and re started for Jupyter to recognise the new version. It is also useful to take note that units used within the program are not conventional. Frequency is stated in KHz and Velocity is in mms^{-1}. Ipyparallel is also not compatible with Python 2.7. The models and codes have all been converted to python 3. Therefore any model that is to be implemented into the FWI program must be written in Python 3.

Extension of model properties

The extension of the FWI algorithm to a non-isotropic, atteuating, elastic, 3D model is a relatively simple step from the current method. The extension simply requires the starting seismic equation shown by Equation 1.1 to be extended to include the above properties. The rest of the mathamatics stays the same applying the same pracices to the new equation.

Chapter 4

Conclusion

The new methodology of FWI enables the simple implementation of a model into a user friendly interface. The model can be stored in a separate data file and loaded into the FWI algorithm. The number of iterations run and shots utilised can be easiliy changed allowing the user to dictate the level of resolution and time given to getting an inversion. Other parameters related to the model are also easily changed. The variation of the grid spacing, frequency and the time step can be adjusted and the impact quickly observed, allowing the optimum values for each to be found.

Further to this, the methodology can easly be upgraded to run much larger and higher resolution models by simply increasing the number of engines available with the parallel computing, providing the opportunity for the program to be integrated into industries, specifically the oil and gas, where FWI would play a valued role in improving the interpretations and understanding of the subsurface.

Improvements are required to the new methodology to provide a robust algorithm, including the integration of a second order gradient method, a preventer of cycle skipping and a Hessian Pre conditioner to provide a more reliable reconstruction that better represents the true subsurface and does not run the risk of converging towards a local rather than global minimum of the objective function.

To improve the reconstruction speed and resolution a second order gradient function should be incorporated. Although with the current synthetic models that have been tested this will have limited impact, with a real life dataset where the initial guess model is larger and further from the true data a second order gradient method will help the inversion to converge towards the true solution.

The new methodology has been successful at running inversions and is a quicker and easier to ma-

nipulate interface in comparison to that used by *Guasch* (2012). It is a step in the right direction to FWI being implemented as common practice within industries.

Bibliography

Ambartsumian, R. V. (1998), *A life in astrophysics: Selected papers of Viktor A. Ambartsumian.*

Bourgeois, A., M. Bourget, P. Lailly, M. Poulet, P. Ricarte, and R. Versteeg (1991), Marmousi, model and data, *The Marmousi experience: EAGE*, pp. 5–16.

Castellanos, C., L. Métivier, S. Operto, R. Brossier, and J. Virieux (2015), Fast full waveform inversion with source encoding and second-order optimization methods, *Geophysical Journal International*, *200*(2), 718–742.

Courant, R., K. Friedrichs, and H. Lewy (1967), On the partial difference equations of mathematical physics, *IBM journal*, *11*(2), 215–234.

Dai, Y.-H., and Y. Yuan (1999), A nonlinear conjugate gradient method with a strong global convergence property, *SIAM Journal on Optimization*, *10*(1), 177–182.

Graves, R. W. (1996), Simulating seismic wave propagation in 3d elastic media using staggered-grid finite differences, *Bulletin of the Seismological Society of America*, *86*(4), 1091–1106.

Guasch, L. (2012), 3d elastic full-waveform inversion, Ph.D. thesis, Imperial College London.

Han, B., Q. He, Y. Chen, and Y. Dou (2014), Seismic waveform inversion using the finite-difference contrast source inversion method, *Journal of Applied Mathematics*, *2014*.

Lange, M., N. Kukreja, M. Louboutin, F. Luporini, F. Vieira, V. Pandolfo, P. Velesko, P. Kazakas, and G. Gorman (2016), Devito: Towards a generic finite difference dsl using symbolic python, *arXiv preprint arXiv:1609.03361*.

Levander, A. R. (1988), Fourth-order finite-difference p-sv seismograms, *Geophysics*, *53*(11), 1425–1436.

Métivier, L., and R. Brossier (2016), The seiscope optimization toolbox: A large-scale nonlinear optimization library based on reverse communication, *Geophysics*, *81*(2), F1–F15.

Mulder, W. A., and R.-E. Plessix (2004), A comparison between one-way and two-way wave-equation migration, *Geophysics*, *69*(6), 1491–1504.

Nocedal, J. (1980), Updating quasi-newton matrices with limited storage, *Mathematics of computation*, *35*(151), 773–782.

Pratt, R. G. (1999), Seismic waveform inversion in the frequency domain, part 1: Theory and verification in a physical scale model, *Geophysics*, *64*(3), 888–901.

Pratt, R. G., C. Shin, and G. Hick (1998), Gauss–newton and full newton methods in frequency–space seismic waveform inversion, *Geophysical Journal International*, *133*(2), 341–362.

Shah, N., M. Warner, T. Nangoo, A. Umpleby, I. Stekl, J. Morgan, L. Guasch, et al. (2012), Quality assured full-waveform inversion: Ensuring starting model adequacy, in *Proceedings of the 82nd Annual International Meeting*.

Sheriff, R. E., and L. P. Geldart (1995), *Exploration seismology*, Cambridge university press.

Shin, C., S. Jang, and D.-J. Min (2001), Improved amplitude preservation for prestack depth migration by inverse scattering theory, *Geophysical prospecting*, *49*(5), 592–606.

Tarantola, A. (1984), Inversion of seismic reflection data in the acoustic approximation, *Geophysics*, *49*(8), 1259–1266.

Tarantola, A. (2005), *Inverse problem theory and methods for model parameter estimation*, siam.

Wang, Y. (2015), Frequencies of the ricker wavelet, *Geophysics*, *80*(2), A31–A37.

Warner, M., A. Ratcliffe, T. Nangoo, J. Morgan, A. Umpleby, N. Shah, V. Vinje, I. Štekl, L. Guasch, C. Win, et al. (2013), Anisotropic 3d full-waveform inversion, *Geophysics*, *78*(2), R59–R80.

Yoon, K., K. J. Marfurt, W. Starr, et al. (2004), Challenges in reverse-time migration, in *2004 SEG Annual Meeting*, Society of Exploration Geophysicists.

Appendix A

Notation and Definitions

A.1 Acronyms

AWE	Acoustic Wave Equation
DSL	Domain Specific Language
FWI	Full Waveform Inversion
FD	Finite Difference
RTM	Reverse Time Migration

A.2 Adjoint

The Adjoint-State method is a widlely used method used to compute the gradient of a functional with respect to the model parameters. WIth FWI the adjoint is the same as the residual providing the data is not normalised. That is the objectiv function is $f = \frac{1}{2} \|\mathbf{p}' - \mathbf{d}\|$ not $f = \frac{1}{2} \left\| \frac{\mathbf{p}'}{|p|} - \frac{\mathbf{d}}{|d|} \right\|$

A.3 Preconditioner

A preconditioning stratergy is used when conducting a second-order method for calculating the inverse of the Hessian operator. It creates an appoximation of the Hessian that is computed from analyitic or semianalytic formulas, e.g a diagonal approximation. Such methods are utilised by *Shin et al.* (2001)

in which an approximate inverse of the Hessian operator which is then integrated within first-order algorithms. A preconditioner for the Hessian matrix can be defined by

$$P \simeq \mathbf{H}^{-1} \tag{A.1}$$

Appendix B

Scripts

B.1 Phantom Synthetic Model

The Python Class script used to initiate the base model with a uniform velocity gradient and then add velocity anomalies. Refractors are added as smoothed circles of positive and negative velocity anomalies. Reflectors are added as negative velocity ellipse anomalies.

```
# coding: utf-8
from __future__ import print_function

import os
from scipy import ndimage
import numpy

from examples.containers import IShot, IGrid
from examples.acoustic.Acoustic_codegen import Acoustic_cg

# Plotting modules.
import matplotlib.pyplot as plt
from matplotlib import cm

# Setup figure size
fig_size = [0, 0]
fig_size[0] = 18
fig_size[1] = 13
plt.rcParams["figure.figsize"] = fig_size

class demo:
    origin = None
    spacing = None
    dimensions = None
```

```
t0 = None
tn = None

# Source function: Set up the source as Ricker wavelet for f0
def _source(self, t, f0):
    r = (numpy.pi * f0 * (t - 1./f0))
    return (1-2.*r**2)*numpy.exp(-r**2)

# Plot velocity
def plot_velocity(self, vp, vmin=1.5, vmax=4, cmap=cm.seismic):
    l = plt.imshow(numpy.transpose(vp), vmin=1.5, vmax=4, cmap=cm.seismic,
                extent=[self.origin[0], self.origin[0]+self.dimensions[0]*self.spacing[0],
                self.origin[1]+self.dimensions[1]*self.spacing[1], self.origin[1]])
    plt.xlabel('X position (m)')
    plt.ylabel('Depth (m)')
    plt.colorbar(l, shrink=.25)
    plt.show()

# Show the shot record at the receivers.
def plot_record(self, rec):
    limit = 0.05*max(abs(numpy.min(rec)), abs(numpy.max(rec)))
    print (limit)
    print (rec.shape)
    l = plt.imshow(rec, vmin=-limit, vmax=limit,
            cmap=cm.gray)
    plt.axis('auto')
    plt.xlabel('X position (m)')
    plt.ylabel('Time (ms)')
    plt.colorbar(l, extend='max')
    plt.show()

# Show the RTM image.
def plot_rtm(self, grad):
    diff = numpy.diff(numpy.diff(numpy.transpose(grad[40:-40, 40:-40]), 1, 0), 1)
    print ("this was fine")
    vmin = 0.001*numpy.min(diff)
    vmax = 0.001*numpy.max(diff)
    print("vmin", vmin)
    print("vmax", vmax)
    l = plt.imshow(diff,
                   vmin=vmin, vmax=vmax, aspect=1, cmap=cm.gray)
    plt.show()

def _init_receiver_coords(self, nrec):
    receiver_coords = numpy.zeros((nrec, 2))

    start = self.origin[0]
    finish = self.origin[0] + self.dimensions[0] * self.spacing[0]

    receiver_coords[:, 0] = numpy.linspace(start, finish, num=nrec)
    receiver_coords[:, 1] = self.origin[1] + 28 * self.spacing[1]
```

```
        return receiver_coords

class marmousi2D(demo):
    """
    Class to setup 2D marmousi demo.
    """
    def __init__(self):
        filename = os.environ.get("DEVITO_DATA", None)
        if filename is None:
            raise ValueError("Set DEVITO_DATA")
        else:
            filename = filename+"/Simple2D/vp_marmousi_bi"
        self.dimensions = dimensions = (1601, 401)
        self.origin = origin = (0., 0.)
        self.spacing = spacing = (7.5, 7.5)
        self.nsrc = 1001
        self.spc_order = 10

        # Read velocity
        vp = numpy.fromfile(filename, dtype='float32', sep="")
        vp = vp.reshape(self.dimensions)

        self.model = IGrid(self.origin, self.spacing, vp)

        # Smooth true model to create starting model.
        smooth_vp = ndimage.gaussian_filter(vp, sigma=(6, 6), order=0)

        # Inforce the minimum and maximum velocity to be the same as the
        # true model to insure both mdelling solver will use the same
        # value for the time step dt.
        smooth_vp = numpy.max(vp)/numpy.max(smooth_vp)*smooth_vp

        # Inforce water layer velocity
        smooth_vp[:,1:29] = vp[:,1:29]

        self.model0 = model0 = IGrid(origin, spacing, smooth_vp)

        # Set up receivers
        self.data = data = IShot()

        f0 = .025
        self.dt = dt = self.model.get_critical_dt()
        self.t0 = t0 = 0.0
        self.tn = tn = 4000
        self.nt = nt = int(1+(tn-t0)/dt)

        self.time_series = 1.0e-3*self._source(numpy.linspace(t0, tn, nt), f0)

        receiver_coords = self._init_receiver_coords(self.nsrc)
        data.set_receiver_pos(receiver_coords)
```

```
        data.set_shape(nt, self.nsrc)

        start = 2 * self.spacing[0]
        finish = self.origin[0] + (self.dimensions[0] - 2) * self.spacing[0]
        self.sources = numpy.linspace(start, finish, num=self.nsrc)

    def get_true_model(self):
        return self.model

    def get_initial_model(self):
        return self.model0

    def get_shot(self, i):
        location = numpy.zeros((1, 2))
        location[0, 0] = self.sources[i]
        location[0, 1] = self.origin[1] + 2 * self.spacing[1]

        src = IShot()
        src.set_receiver_pos(location)
        src.set_shape(self.nt, 1)
        src.set_traces(self.time_series)

        Acoustic = Acoustic_cg(self.model, self.data, src, t_order=2, s_order=self.spc_order)
        rec, u, gflopss, oi, timings = Acoustic.Forward(save=False, dse=True)

        return self.data, rec, src


class small_marmousi2D(demo):
    """
    Class to setup a small 2D marmousi demo.
    """
    def __init__(self):
        filename = os.environ.get("DEVITO_DATA", None)
        if filename is None:
            raise ValueError("Set DEVITO_DATA")
        else:
            filename = filename+"/marmousi3D/MarmousiVP.raw"
        self.dimensions = (201, 201, 70)
        self.origin = (0., 0.)
        self.spacing = (15., 15.)
        self.nsrc = 101
        self.spc_order = 4

        # Read velocity
        vp = 1e-3*numpy.fromfile(filename, dtype='float32', sep="")
        vp = vp.reshape(self.dimensions)

        # This is a 3D model - extract a 2D slice.
        vp = vp[101, :, :]
        self.dimensions = self.dimensions[1:]
```

```
        self.model = IGrid(self.origin, self.spacing, vp)

        # Smooth true model to create starting model.
        slowness = 1.0/self.model.vp
        smooth_slowness = ndimage.gaussian_filter(slowness, sigma=(3, 3), order=0)
        smooth_vp = 1.0/smooth_slowness

        # smooth_vp = numpy.max(self.model.vp)/numpy.max(smooth_vp) * smooth_vp

        truc = (self.model.vp <= (numpy.min(self.model.vp)+.01))
        smooth_vp[truc] = self.model.vp[truc]

        self.model0 = IGrid(self.origin, self.spacing, smooth_vp)

        # Set up receivers
        self.data = IShot()

        f0 = .007
        self.dt = dt = self.model.get_critical_dt()
        t0 = 0.0
        ly = self.dimensions[1]*self.spacing[1]
        lx = self.dimensions[0]*self.spacing[0]/2
        tn = 2*numpy.sqrt(lx*lx+ly*ly)/2.0 # ie the mean time
        print ("tn is %f"%tn)
        self.nt = nt = int(1+(tn-t0)/dt)

        self.time_series = self._source(numpy.linspace(t0, tn, nt), f0)

        receiver_coords = numpy.zeros((self.nsrc, 2))
        start = 2 * self.spacing[0]
        finish = self.origin[0] + (self.dimensions[0] - 2) * self.spacing[0]
        receiver_coords[:, 0] = numpy.linspace(start, finish,
                                               num=self.nsrc)
        receiver_coords[:, 1] = self.origin[1] + 2 * self.spacing[1]
        self.data.set_receiver_pos(receiver_coords)
        self.data.set_shape(nt, self.nsrc)

        start = 2 * self.spacing[0]
        finish = self.origin[0] + (self.dimensions[0] - 2) * self.spacing[0]
        self.sources = numpy.linspace(start,
                                      finish,
                                      num=self.nsrc)

    def get_true_model(self):
        return self.model

    def get_initial_model(self):
        return self.model0

    def get_shot(self, i):
```

```
        location = numpy.zeros((1, 2))
        location[0, 0] = self.sources[i]
        location[0, 1] = self.origin[1] + 2 * self.spacing[1]

        src = IShot()
        src.set_receiver_pos(location)
        src.set_shape(self.nt, 1)
        src.set_traces(self.time_series)

        Acoustic = Acoustic_cg(self.model, self.data, src, t_order=2, s_order=self.spc_order)
        rec, u, gflopss, oi, timings = Acoustic.Forward(save=False, dse=True)

        return self.data, rec, src


class small_phantoms2D(demo):
    """
    Class to setup a small 2D demo with phantoms.
    """
    def __init__(self):
        f0 = .015
        #dx = 2500.0*f0    # wave_speed = frequency * wavelength
        #use slowest velocity as this is where the code is most dispursive
        dx = 15
        self.origin = origin = (0, 0)
        self.spacing = spacing = (dx, dx)
        self.dimensions = dimensions = (int(10000/spacing[0]), int(3000/spacing[1]))
        self.sd = sd = 300  # Sea depth in meters

        self.nsrc = nsrc = 666  # Number of source/receivers
        self.spc_order = 4 # Spacial order.

        model_true = numpy.ones(dimensions)
        #transpose array to work with toolkit
        model_true = numpy.transpose(model_true)

        #Puts depth of sea floor into grid spacing defined by dx
        sf_grid_depth = int(sd/spacing[1])
        print ("the seafloor depth for the initial guess model is", sf_grid_depth)
        max_v = 3000.  # m/s  velocity at bottom of sea bed
        seabed_v = 1700.  # m/s velocity at top of seabed

        #Velocity gradient of seabed
        m = (max_v-seabed_v)/(dimensions[1]-1-sf_grid_depth)

        #Set velocity of seabed (uses velocity gradient m)
        for i in range(sf_grid_depth, dimensions[1]):
            model_true[i][:] = (m*(i-sf_grid_depth)) + seabed_v


        #We are going to use the background velocity profile as the initial
```

```
#solution.
smooth_vp = numpy.copy(model_true)

#Set velocity of water
for i in range(sf_grid_depth):
    smooth_vp[i][:] = 1500.  # m/s water velocity

#Convert to km/s
smooth_vp = smooth_vp*1.0e-3

#transpose smooth model
smooth_vp = numpy.transpose(smooth_vp)

self.model0 = IGrid(origin, spacing, smooth_vp)

#Reflectors: Add circular positive velocity anomaly.
radius = int(500./spacing[0])
cx1, cy1 = int(2500./spacing[0]), int(1200./spacing[1])  # Center
yc1, xc1 = numpy.ogrid[-radius:radius, -radius:radius]
index = xc1**2 + yc1**2 <= radius**2
model_true[cy1-radius:cy1+radius, cx1-radius:cx1+radius][index] = 2900.

#Reflectors: Add circular negative velocity anomaly.
cx2, cy2 = int(6500./spacing[0]), int(1200./spacing[1])
yc2, xc2 = numpy.ogrid[-radius:radius, -radius:radius]
index = xc2**2 + yc2**2 <= radius**2
model_true[cy2-radius:cy2+radius, cx2-radius:cx2+radius][index] = 1700.

# Smoothen the transition between regions.
blended_model = ndimage.gaussian_filter(model_true, sigma=4)
#when i blurr the model it changes the sea floor depth?
#blended_model = model_true

# Add reflectors - negative anomalies
ey1, ex1 = int(2000./spacing[0]), int(3000./spacing[1])
rx, ry = int(350./spacing[0]), int(75./spacing[1])
ye, xe = numpy.ogrid[-radius:radius, -radius:radius]
index = (xe**2/rx**2) + (ye**2/ry**2) <= 1
blended_model[ey1-radius:ey1+radius, ex1-radius:ex1+radius][index] = 2000.

ey2, ex2 = int(2000./spacing[0]), int(6250./spacing[1])
rx, ry = int(200./spacing[0]), int(75./spacing[1])
ye2, xe2 = numpy.ogrid[-radius:radius, -radius:radius]
index = (xe2**2/rx**2) + (ye2**2/ry**2) <= 1
blended_model[ey2-radius:ey2+radius, ex2-radius:ex2+radius][index] = 2000.

#set the water velocity of the true model
print ("the seafloor depth for the true model is", sf_grid_depth)
#Set velocity of water
for i in range(sf_grid_depth):
    blended_model[i][:] = 1500.  # m/s water velocity
```

```
        # Convert to km/s
        vp = blended_model * 1.0e-3  # Convert to km/s

        #transpose true model
        vp = numpy.transpose(vp)

        self.model = IGrid(origin, spacing, vp)

        # Define seismic data.
        self.data = data = IShot()

        self.dt = dt = self.model.get_critical_dt()
        self.t0 = t0 = 0.0
        self.tn = tn = 5800
        self.nt = nt = int(1+(tn-t0)/dt)

        self.time_series = self._source(numpy.linspace(t0, tn, nt), f0)

        self.receiver_coords = numpy.zeros((nsrc, 2))
        start = 2 * spacing[0]
        finish = origin[0] + (dimensions[0] - 2) * spacing[0]
        self.receiver_coords[:, 0] = numpy.linspace(start,
                                                    finish,
                                                    num=nsrc)
        self.receiver_coords[:, 1] = origin[1] + 2 * spacing[1]
        data.set_receiver_pos(self.receiver_coords)
        data.set_shape(nt, nsrc)

        start = 2 * spacing[0]
        finish = origin[0] + (dimensions[0] - 2) * spacing[0]
        self.sources = numpy.linspace(start, finish, num=nsrc)

    def get_true_model(self):
        return self.model

    def get_initial_model(self):
        return self.model0

    def get_shot(self, i):
        location = numpy.zeros((1, 2))
        location[0, 0] = self.sources[i]
        location[0, 1] = self.origin[1] + 2 * self.spacing[1]

        src = IShot()
        src.set_receiver_pos(location)
        src.set_shape(self.nt, 1)
        src.set_traces(self.time_series)

        Acoustic = Acoustic_cg(self.model, self.data, src, t_order=2, s_order=self.spc_order)
        rec, u, gflopss, oi, timings = Acoustic.Forward(save=False, dse=True)
```

```
        return self.data, rec, src
```