A new Methodology for Implementing Full Waveform Inversion

Emma Pearce

Supervisor: Dr Gerard Gorman

Submission Date: 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, preventing the techniques adoption for industry use. The new methodology presented here allows FWI to be implemented with an intuitive notebook style interface that is adaptable, allowing models to be inverted with little work using the numerical acoustic framework developed by *Guasch* (2012). These are equations that traditionally require a low level language such as Fortran to implement them along side a detailed knowledge of partial differential equation discretisation. This is achieved by a symbolic maths engine (SymPy) and Devito, a domain specific language that enables geophysicists to write and compile complicated mathematical equations in only a few lines of code, where Python is used to implement the FWI, but a low level language is used as the solvers. A first order steepest descent method has been used to calculate the gradient, providing an acceptable balance between computational requirements and convergence speed. Further to this, parameters that usually require large amounts of time to find the optimum value for can be edited effortlessly and the repercussions quickly observed. Improvements are needed to ensure a more robust inversion that is not prone to cycle skipping and can prevent unrealistic reconstructions. 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 (**?**), 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 the inversion to three dimensions with elastic properties being taken into account. Although successful,

the implementation was difficult to extend further because of the complex algorithms used in FWI and the need for high computational performance. Further to this, developing a new method is a difficult and laborious task, particularly when programming in a low level language such as Fortran.

In this work, FWI has been implemented in Python, a high level language, using the domain specific language compiler Devito (*Lange et al.*, 2016) (*Warner et al.*, 2013), whilst utilising the numerical acoustic framework develop 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.

Through the use of a simple to use Jupyter Notebook [1] that makes the running of FWI on different models practical to execute whilst enabling parameters to be easily changed, a new methodology for FWI has been established. This is the first time that both FWI and reverse time migration (RTM) have been accomplished in this way, providing the possibility for partial differential equation discretisations to be written in only a few lines of code by those less proficient at programming. 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.

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 (SymPy[3]). 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

[1] jupyter.org
[2] github.com/opesci
[3] www.sympy.org

velocity models and the subsequent migration/inversion calculated (*Lange et al.*, 2016) to produce reconstructions of the Earth's subsurface.

1.1.2 Reverse time migration

RTM is the practice of re-locating diffraction events on unmigrated seismic records to points, thereby moving ("migrating") reflection events to their proper locations in time, creating a true image of structures within the earth. Traditional migration techniques only propagate data downwards through a velocity model, in comparison to RTM, which propagates data both downwards and upwards producing high resolution models of the subsurface (**?**). RTM is currently recognised as the most common method of seismic modelling as it offers an acceptable compromise between computational power and image quality, unlike current methods of FWI (*Yoon et al.*, 2004).

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, which is used by both RTM and FWI, thus 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 Earth's 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 Forward inverse problem

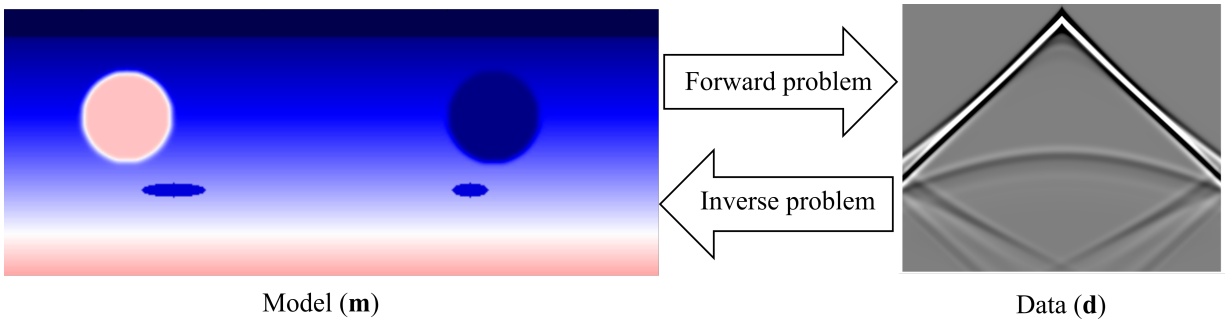

Figure 1.1: Visual representation of the relationship between physical quantities (model parameters) and real observations (data) and how they relate to the process of having a forward and an inverse problem. The model parameters are seismic velocity, and the data is a pressure seismogram.

A forward problem as shown by figure 1.1 is 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 describe 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 and 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. 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. A scalar quantity known as the objective function (f) is then defined by,

$$f = \frac{1}{2}\left\|\mathbf{p\prime} - \mathbf{d}\right\|_2^2 = \left\|G(\mathbf{m}) - \mathbf{d}\right\|_2^2, \tag{1.4}$$

that is equal to the squares of $\Delta\mathbf{d}$. A value of zero for 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 $\|\|_2^2$ is the L^2 norm squared. 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 is identified as the solution to the problem (*Han et al.*, 2014). This 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 to reduce f 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). $\mathbf{H}$ however is much harder to calculate due to its size, therefore only the diagonal elements of it are computed. The approximation of the diagonal elements of $\mathbf{H}$ are possible to calculate using spatial preconditioning (Appendix A) 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. The adjoint source 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 Step length

The step-length α is a scaling factor used to adjust the starting model by a small amount and then 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 $\Delta\mathbf{d}$ 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 complete work flow followed by FWI is shown in figure 1.3 providing a summary of the iterative process.

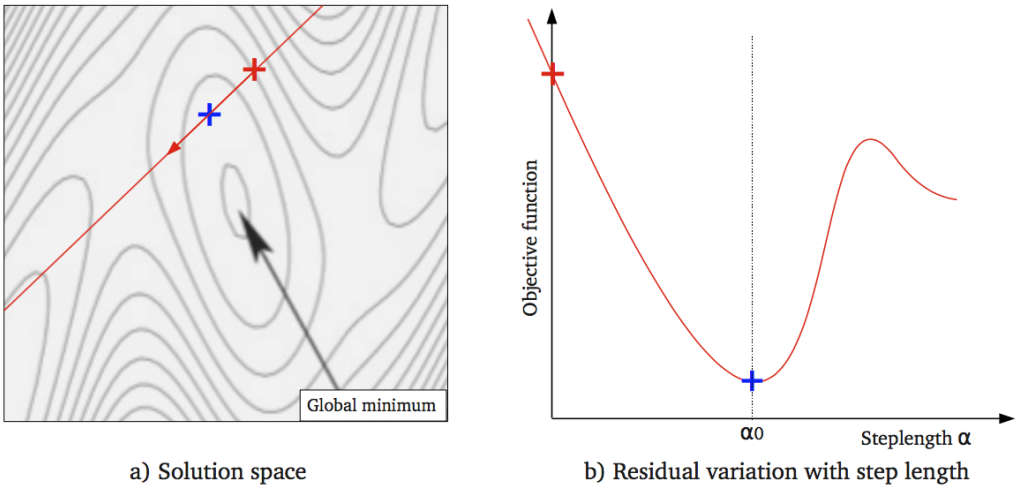

Figure 1.2: The solution space where the minimum is assumed to be the global minimum. The red cross is representative of Δd^0 and the blue cross the minimum of f along the direction of the gradient in image (a). Δd^1 would be a point between the two crosses that is able to assume the residual has a linear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

1.2.5 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.

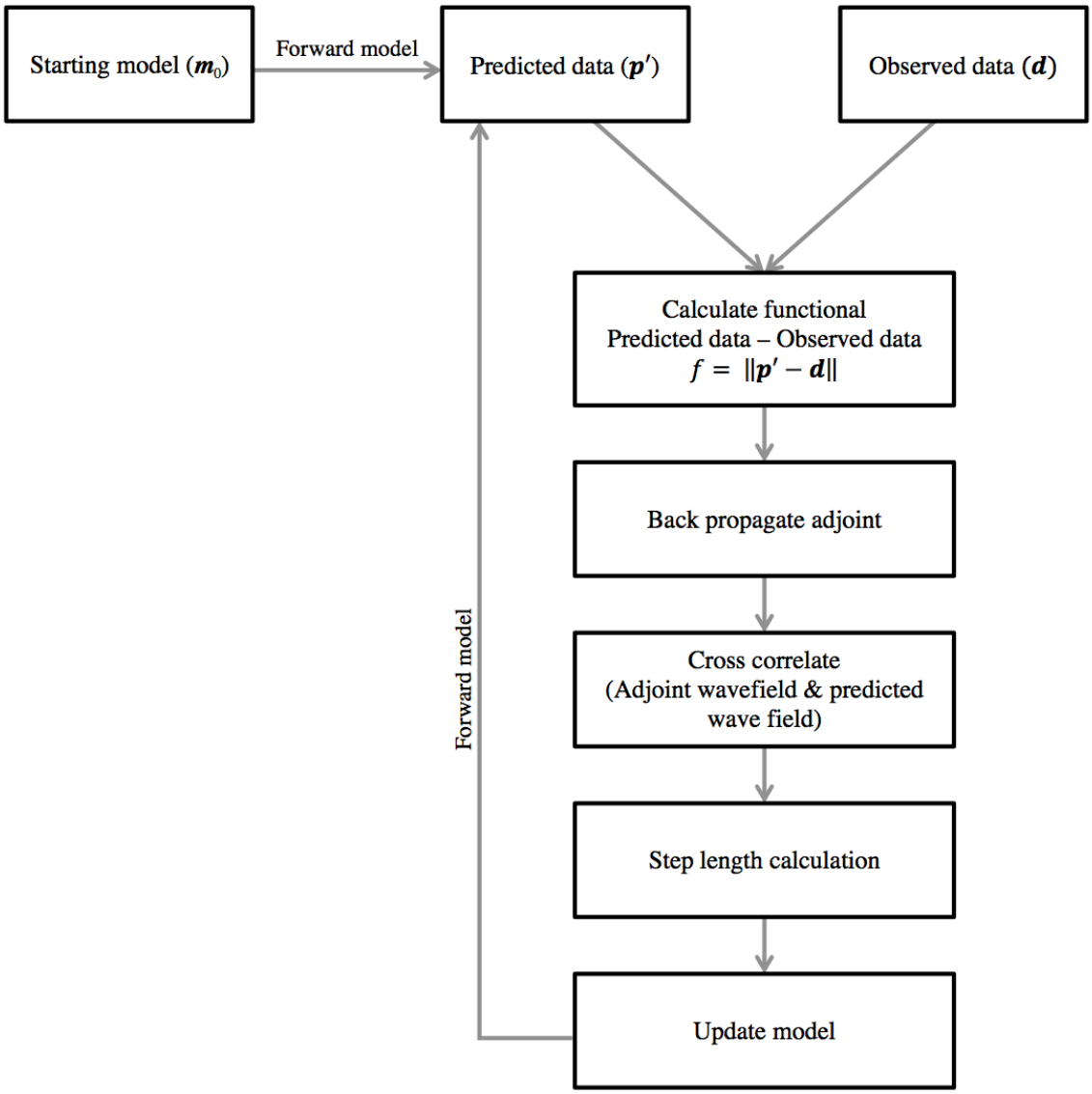

Figure 1.3: The work flow followed by FWI when undergoing iterations to reduce the objective function f.

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 provides the motivation behind the current project to allow FWI to become common practice within industry with a practical FWI methodology that uses less computational resources and can be executed using a high level language.

Chapter 2

Methodology

2.1 Jupyter notebook

All of the algorithms shown are run through Jupyter Notebook, 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). Appendix A shows how to install Devito manually.

2.3 Parallel computing

To run both RTM and FWI they have been parallelised using ipyparallel [1]. Computers have random-access memory (RAM) that cannot cope with the demands of real sized FWI problems, mainly due to the elevated number of sources present in the data. The use of parallel computing can mitigate this problem. It allows multiple calculations to be executed at the same time by accessing a large cluster of networked multicore computer nodes usually designated to run different shots across different nodes shown by figure 2.6, thus reducing the time needed to calculate RTM/FWI. The maximum size of the model is defined by the memory of the nodes (**?**).

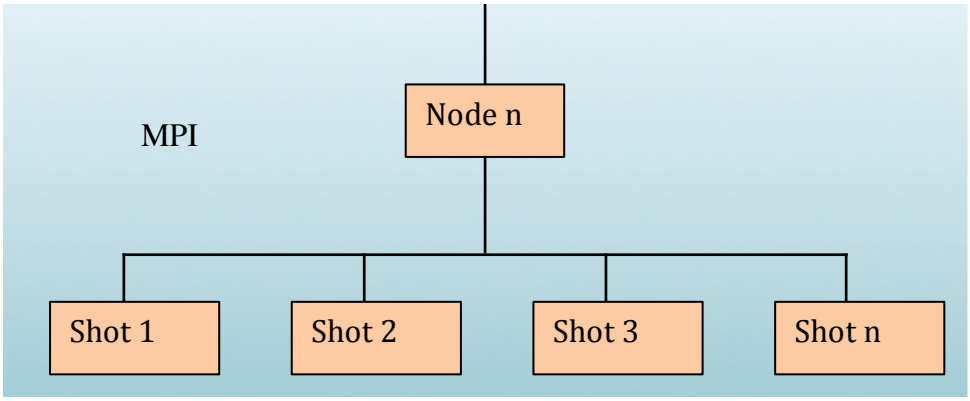

Figure 2.1: Parallel computing showing one of the nodes in the cluster with n amount of shots running from it. MPI - Message Passing Interface, an API, is used to communicate the different processes across the cluster.

Parallel computing is the reason that FWI is feasible on a reasonable time scale. This method is used on a larger scale in industry meaning if the code is required to process higher resolution/larger images, it would just be a matter of increasing the number of clusters available to run the algorithms. Instructions on how to set up the parallel environment are shown in Appendix (A).

[1] ipyparallel.readthedocs.io

2.4 Initialising 2D synthetic models

Created using the script `Synthetic Model` in Appendix C producing a phantom 2D model 2.3, representative of a true subsurface model from observed data. This has then been adjusted to create figure 2.2 that is used as the starting guess for the inversion.

Table 2.1: Parameters for both the phantom and Marmousi model

Parameter	**Phantom**	**Marmousi**
Frequency F	10Hz	15Hz
Grid spacing	15m	7.5m
Spatial order	4	4
number of sources to receivers	666	101
Receiver run time	5800 ms	4000 ms

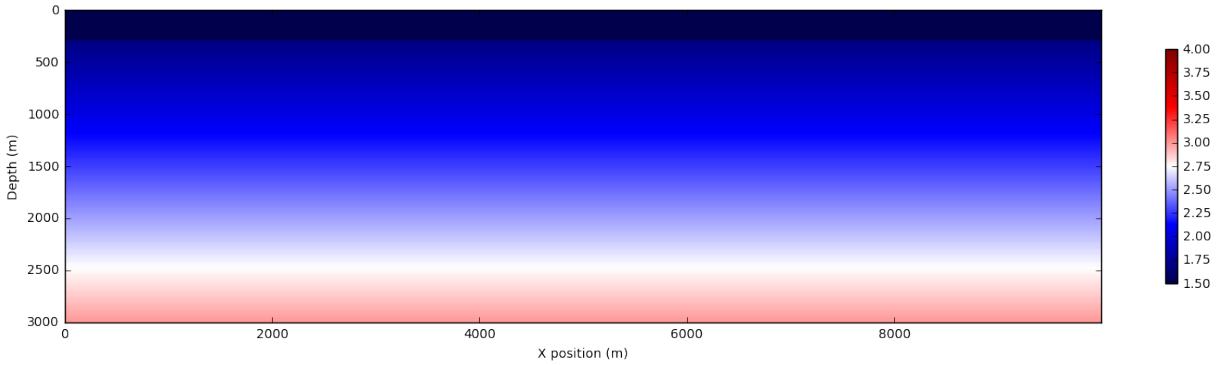

Figure 2.2: A velocity model in ms^{-1}. Initial velocities are chosen for the sea water velocity 1500 ms^{-1}). The subsurface is given a velocity gradient with the lowest velocity 1700 ms^{-1} and the highest velocity at the bottom of the mode equal to 3000 ms^{-1}. The grid spacing of the model is defined by dx which is set equal to 15m.

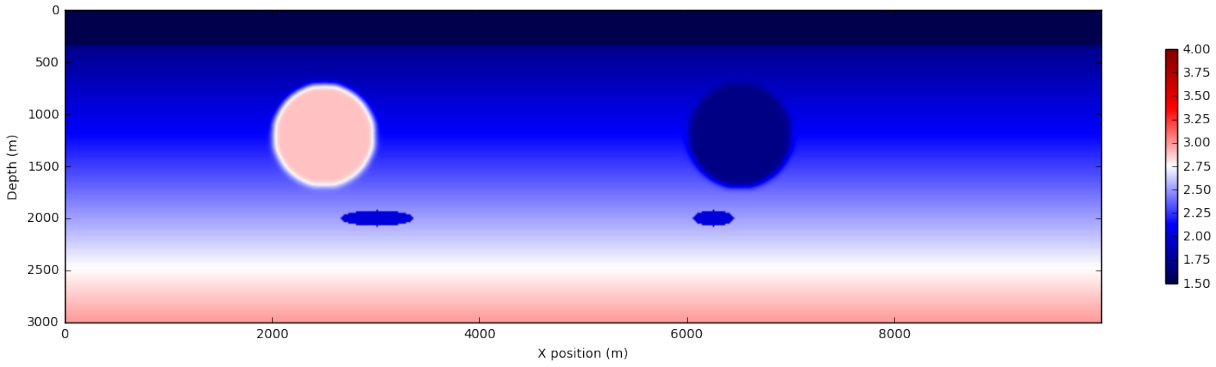

Figure 2.3: A velocity model in ms^{-1}. Velocity anomalies are added to the base model.The velocity anomalies will be what the simulated model is trying to replicate. Circular smoothed anomalies of positive and negative values (positive V = 2900 ms^{-1}, negative V = 1700 ms^{-1}) will bend rays. Reflectors 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. The model has been based on the analogy of the North Sea, with sea depths of 300m and negative refractors imitating gas seepages.

When testing geophysical processing within industry the Marmousi model is used as a control complex geological structure (*Bourgeois et al.*, 1991). Therefore the successful inversion of such model is synonymous with a working program. Built from `Synthetic Model` the true model is shown in figure 2.4 with the initial guess a smoothed version, shown in figure 2.5.

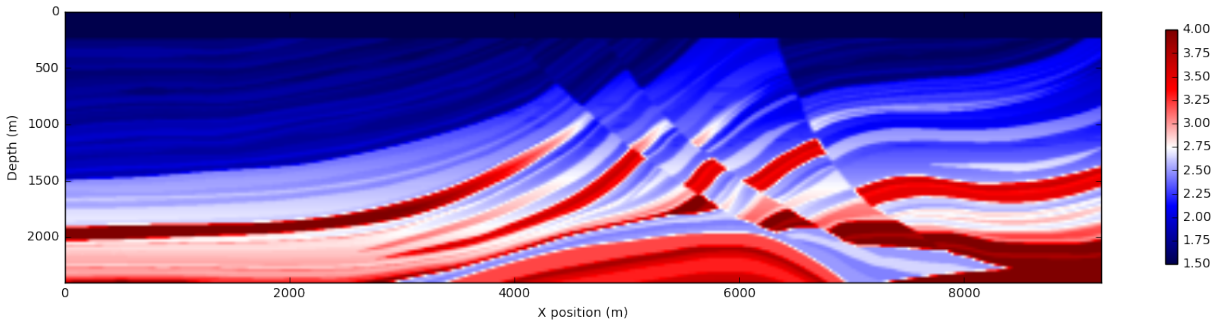

Figure 2.4: A velocity model in ms^{-1}. 2D Marmousi Model based on a profile through North Quenguela trough in the Cuanza basin, Angola. The model contains many refractors, steep dips and strong velocity gradients both vertically and horizontally (*Bourgeois et al.*, 1991).

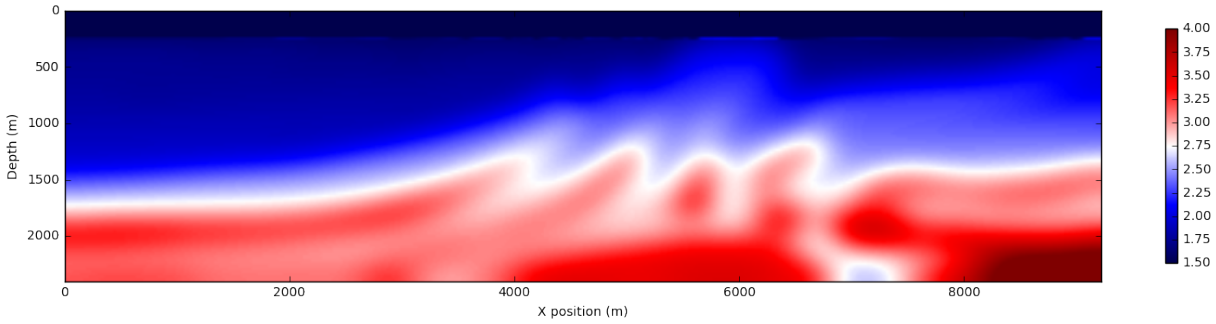

Figure 2.5: A velocity model in ms^{-1}. The 2D Marmousi model smoothed to provide a synthetic initial guess when running RTM and FWI.

2.4.1 Paramaters

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

Frequency

A frequency value should be chosen to be 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 3 - 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 executing fourth order finite differences) (*Warner et al.*, 2013), for the minimum wavelength. For both models the minimum velocity is 1500 ms^{-1}. Using U_{min} =Fλ where F is frequency and λ is wavelength, gives λ = 100m. 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 (CFL) Condition and must be met in order to retain stability when running inversion. If not, then the model can become unstable. It requires that a time step must be small enough that a wavefield crosses no more than half a node. (*Courant et al.*, 1967).

Receiver run time

This should be long enough that the wave propagated from the first source has enough time to travel to the bottom of the model and be received by the final source in the array. For the synthetic model assuming an average velocity of 2000 ms^{-1} the minimum time interval is 5.8 s.

2.5 Data

A Ricker wavelet (figure 2.7) is used as the source wavelet (*Wang*, 2015). An example of the shot record produced from this wavefield is shown in figure 2.8.

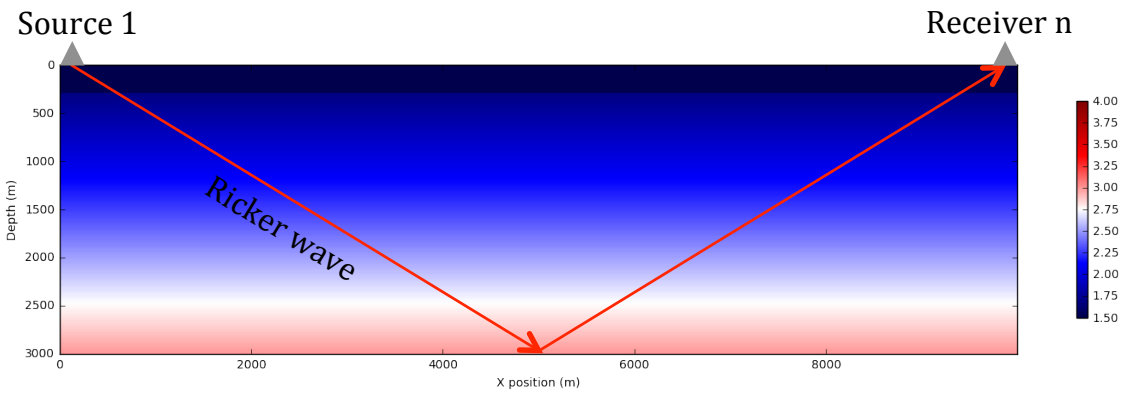

Figure 2.6: The travel path taken by a wave propagated form the first source and received by the final receiver.

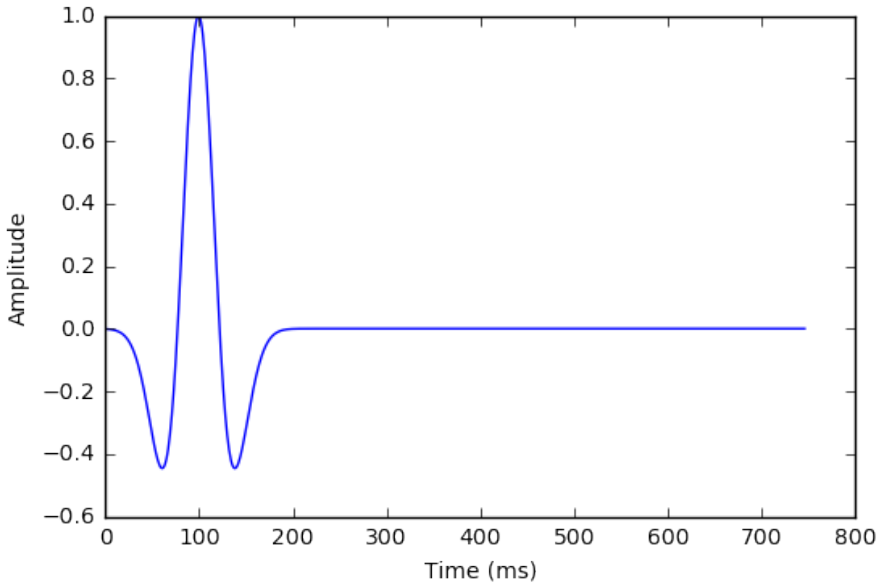

Figure 2.7: Ricker wavelet defined by $A = (1 - 2\pi^2 f^2 t^2)e^{-\pi^2 f^2 t^2}$ where A is amplitude, F frequency, and t time in seconds. With $F = 15Hz$.

2.6 FWI step length

Explained in Section 1.2.4 is the importance of the starting value of the step length, α_i. If the value is too large it can overshoot the correct minimum and settle in the wrong local minimum. A linear relationship must also be able to be assumed between the starting value of the gradient and the minimum value of f along the direction of gradient. The value must be great enough that numerical rounding cannot interfere with it. To settle this criteria α_i has been taken as 1% of the gradient normalised by the maximum value of the gradient.

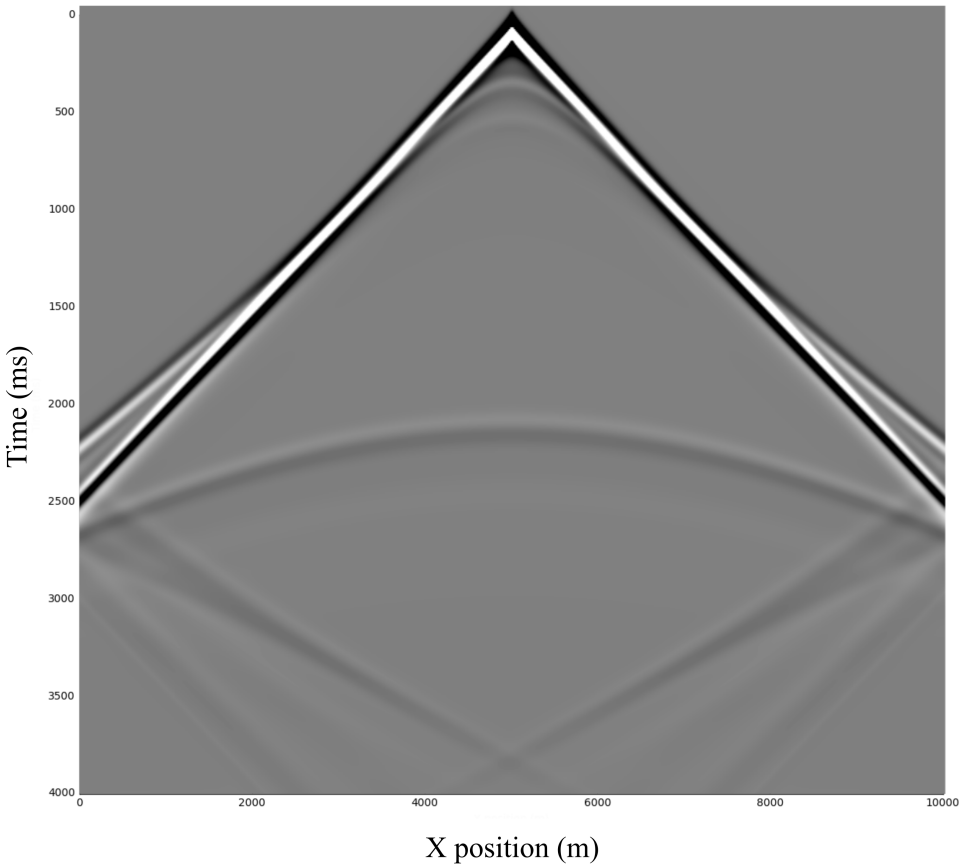

Figure 2.8: An example of the Shot record showing a pressure seismogram at the middle of the domain for the phantom 2D synthetic model.

2.7 FWI models

When running FWI the number of sources and receivers used could be selected (nsrc). This project uses a random sample of 5 for each iteration for both models, for a minimum of 30 iterations. Enabling the whole span of the model to be covered without the need to utilise all sources at once.

Chapter 3

Results

Inversions produced of both the phantom and Marmousi using the FWI notebook. The values of f and α for each inversion are shown in Appendix B. The graphs show the convergence of the objective function towards the minimum value across all iterations. All models are in velocity with dark red the maximum velocity, and dark blue the minimum.

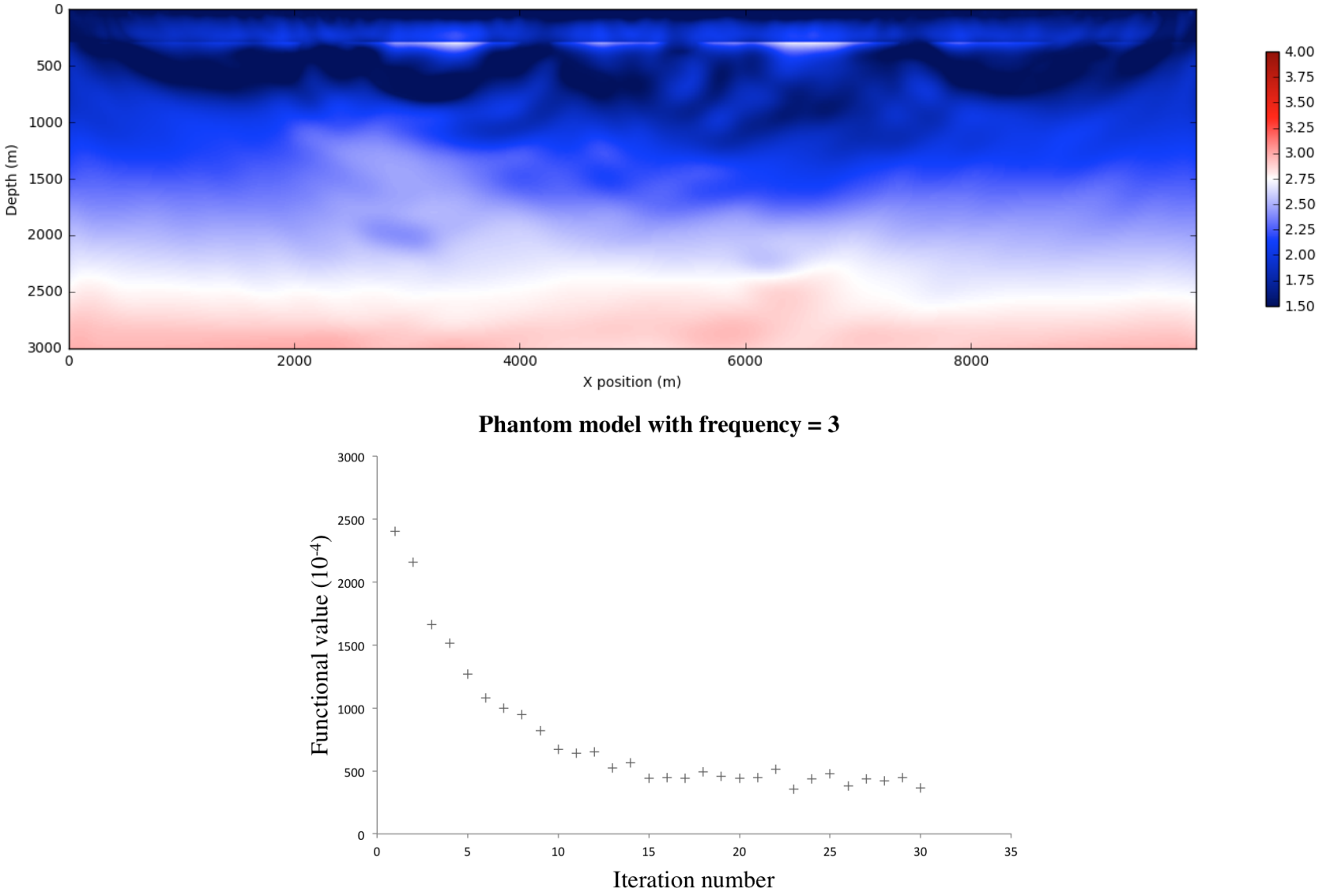

Figure 3.1: Reconstruction of Phantom 2D after 30 iterations using 5 random shots per iteration. Frequency is set at 3Hz. A mask has not been included in the inversion algorithm. Convergence of the functional towards the minimum is achieved within the first 10 iterations.Minimum functional value is 363.28 $\times 10^4$

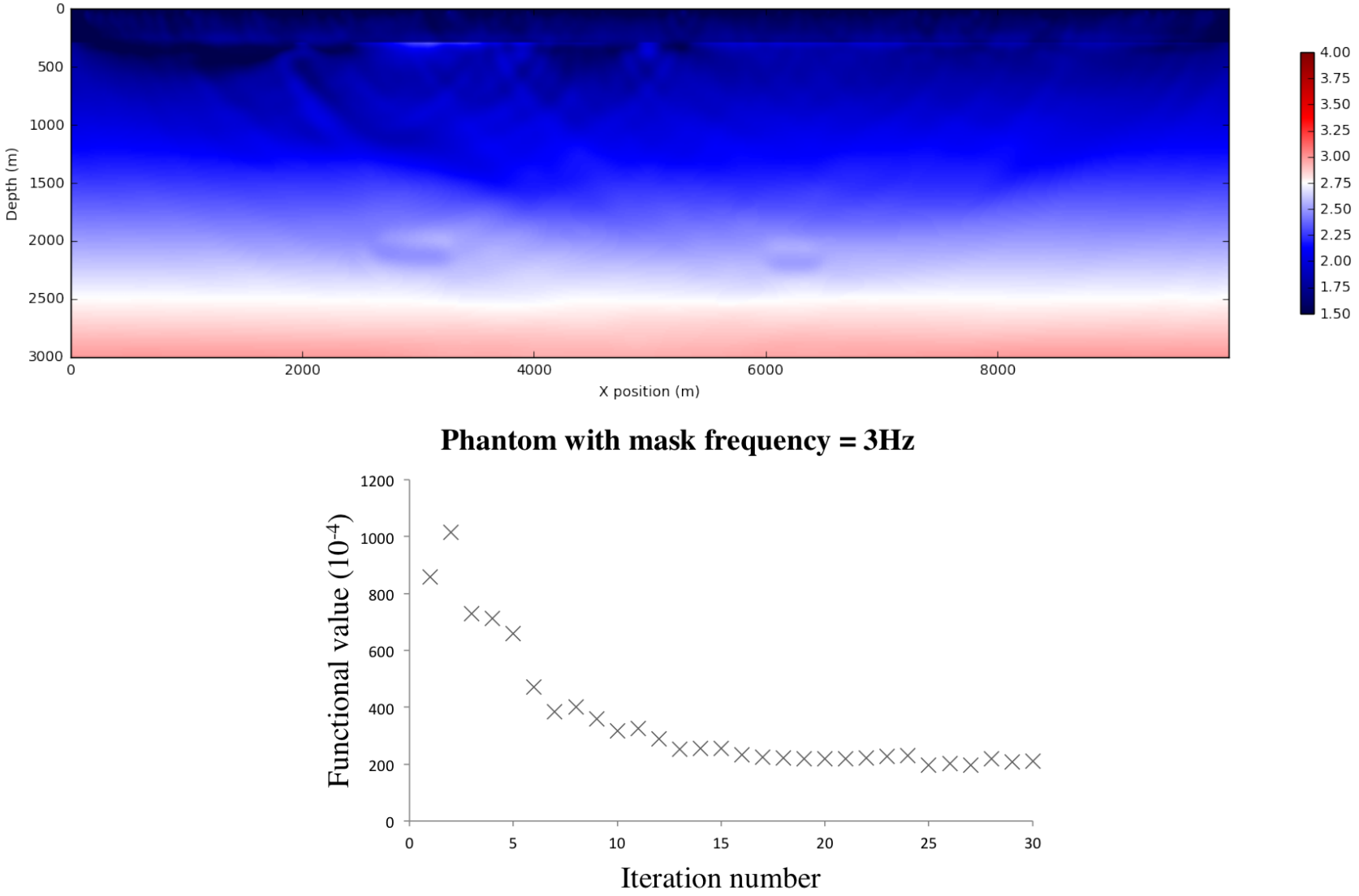

Figure 3.2: A mask has been included and the previous inversion re-run. Minimum functional value of 196.97 $\times 10^4$ is achieved

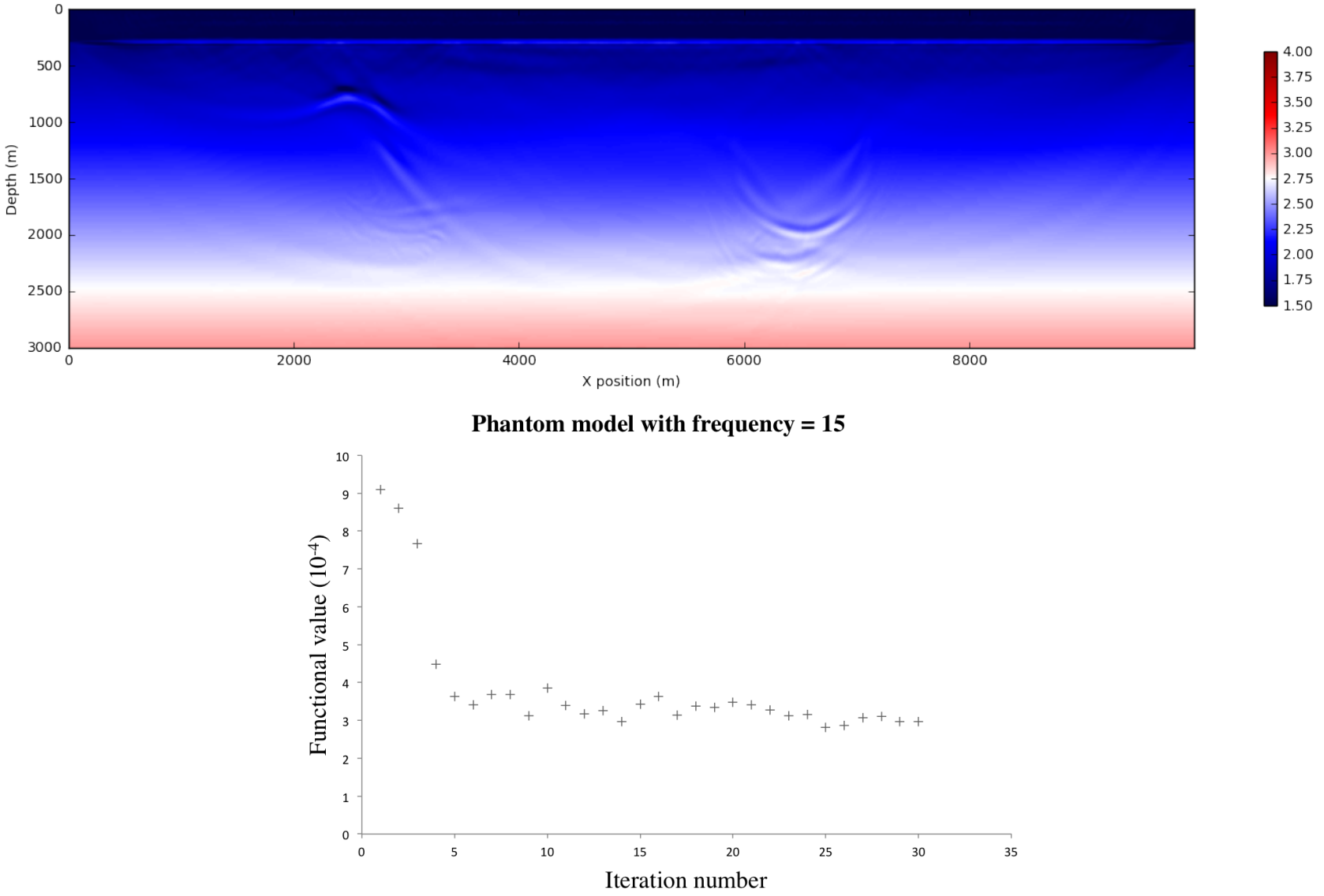

Figure 3.3: Phantom 2D model with 5 shots used per iteration for 30 iterations with frequency at 15Hz. A lack of reconstruction of the true model is seen, with the functional minimum value of 2.97 $\times 10^4$.

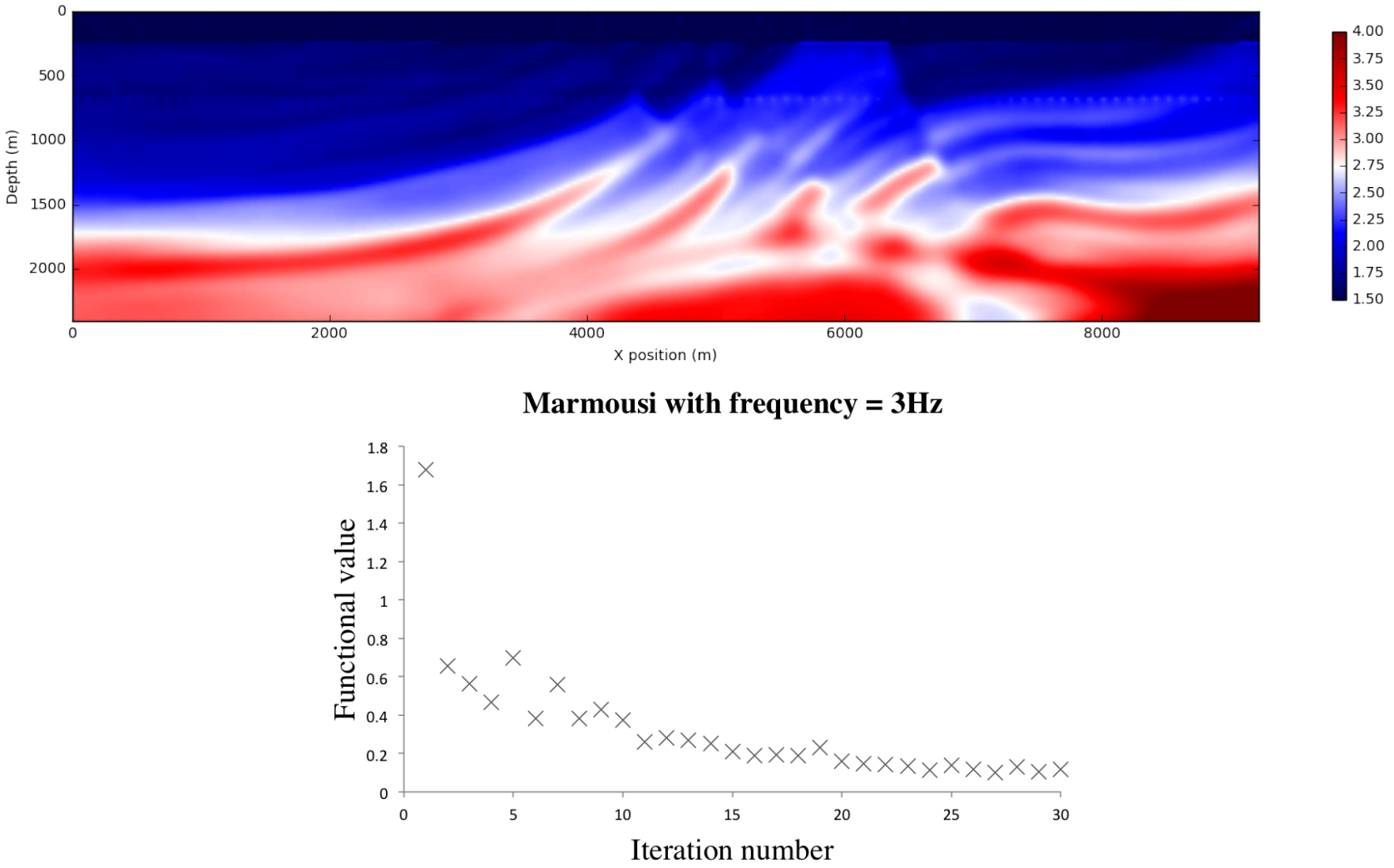

Figure 3.4: Marmousi model inverted using a frequency of 3Hz for 30 iterations with a final residual 46.13 and a minimum functional value of 0.12.

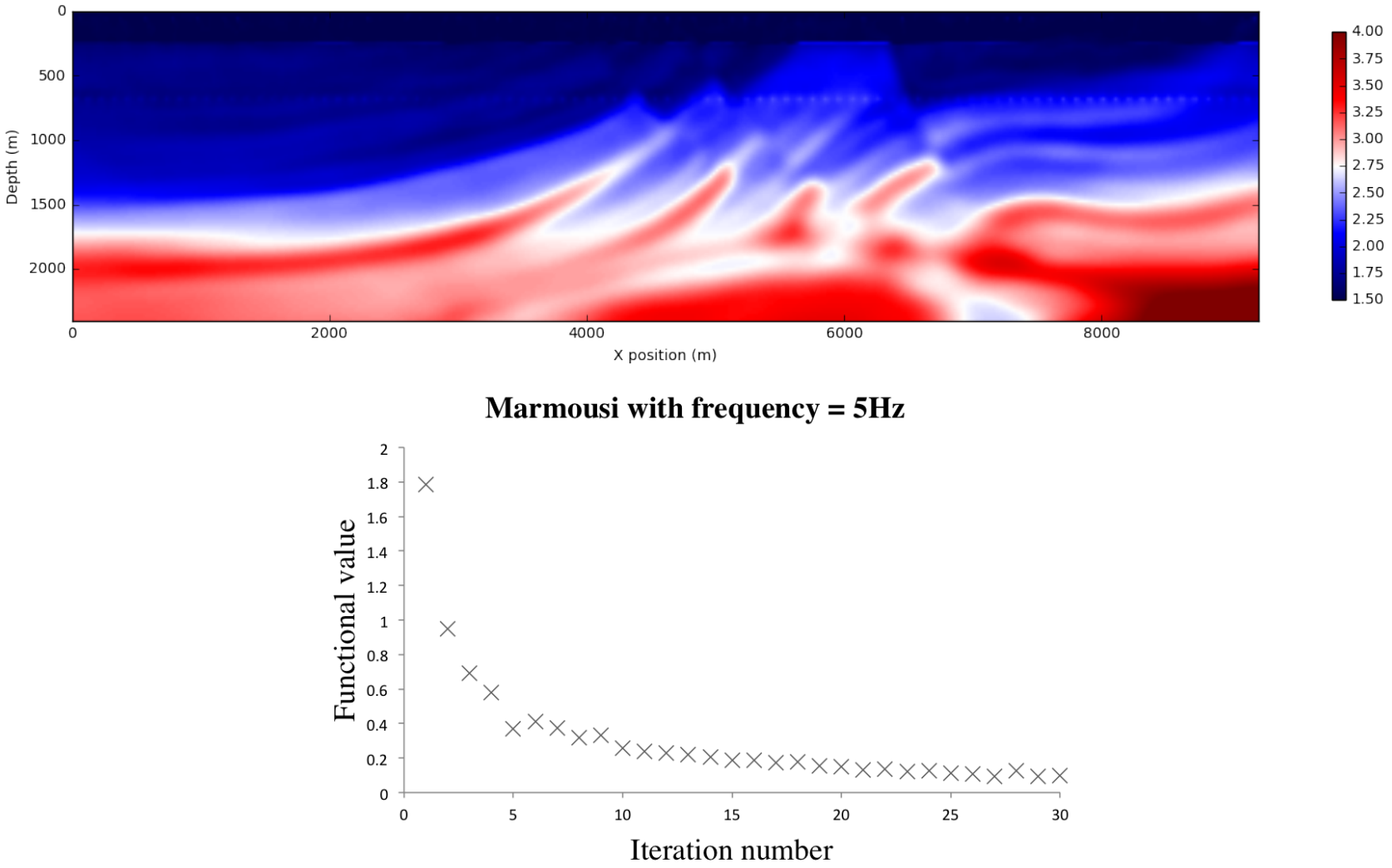

Figure 3.5: Marmousi model inverted using a frequency of 5Hz for 30 iterations with final residual of 45.95 and a minimum functional value of 0.10.

Chapter 4

Discussion

Although all methodologies of FWI have the universal aim of reducing the objective function to a minimum, the gradient can be obtained using different algorithms. In addition to this, different heuristics can be put in place to improve the resolution of the reconstruction. The chosen techniques for the new methodology are discussed and evaluated along with results produced from the FWI notebook from both the phantom and Marmousi models.

4.1 Gradient calculation used by the new methodology

When considering the method that should be used in the FWI notebook to obtain the 'optimum' value of the step length, a calculation that can find the solution to a large-scale non-linear optimization problem is required (*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 4.1. This highlights that the use of a global method for the synthetic models is implausible as dx = 15m giving the model 133,200 parameters $(\left(\frac{lx}{dx} \times \frac{lz}{dx}\right) = (666 \times 200) = 133,200)$. Local descent methods offer far less iterations than the number of parameters to provide an acceptable sampling size, meaning the computation is cheaper and inversion can be achieved.

Amongst possible local descent algorithms either first or second-order calculations can be used. The simplest are first order. These use only the gradient to define the descent direction. The two most common first order methods are the steepest descent and the non-linear conjugate gradient. Second-order methods are defined through the use of a second-order derivative used to approximate the inverse

Table 4.1: 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 computational 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.

of the Hessian matrix, consisting of the quasi-Newton l-BFGS method and the truncated Newton method. Presented below is a summary of each method utilising the work of *Métivier and Brossier* (2016). As all gradient methods ensured the decrease of the objective function and convergence towards a local minimum, it is just a matter of which method will provide the optimum reconstruction verses computational requirements.

Steepest descent

The steepest Descent (SD) is the simplest method, where if P the preconditioned (Appendix A) is not defined then Δx_i is just the opposite of the gradient of the objective function with respect to the model $\frac{\partial f}{\partial \mathbf{m}}$ as shown by,

$$\Delta x_i = -P\frac{\partial f}{\partial \mathbf{m}}. \tag{4.1}$$

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

Nonlinear conjugate gradient

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

$$\Delta x_0 = -Pf\Delta x_i = -P\frac{\partial f}{\partial \mathbf{m}} + \beta_i \Delta x_{i-1}. \tag{4.2}$$

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

Quasi-Newton *l*-BFGS

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

$$\Delta_i = -Q_i\frac{\partial f}{\partial \mathbf{m}}. \tag{4.3}$$

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 super-linear 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, allowing FWI to converge faster than Quasi-Newton but at a higher computational cost using,

$$\mathbf{H}\Delta x_i = -\frac{\partial f}{\partial \mathbf{m}}. \tag{4.4}$$

Both second order methods remain competitive on all other benchmarks (*Castellanos et al.*, 2015).

It was concluded that the SD method would be used, as it was the simplest to examine the convergence behaviour in detail, whilst providing a reliable reconstruction. It is possible that when testing 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. 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 would reconstruct to a good level of detail up to a depth of 4.5km, with the phantom and Marmousi models being 3km deep, the need for second order was not necessary.

4.2 Inversion results

From the inversions shown in Section 3 a number of observations can be made. The optimum value of frequency used during the inversion to produce the best reconstruction is dependent on a number of parameters, and establishing the best to use is one of the most time consuming parts of initiating an inversion. A lower frequency can be used if the data is a good quality, enabling the data to be recovered to a deeper depth as the wave-field experiences less attenuation and a lower amount of computational memory is required to run the inversion. For the same starting model, the lower the frequency, the fewer iterations needed to converge to the minimum functional value, and the faster the convergence, highlighted by the graphs 3.4 where 3Hz is chosen as the source frequenc, and 3.5 using 5Hz. The faster convergence is however achieved at a cost to the resolution of the final produced image. It is expected that a higher frequency will produce a better resolution reconstruction. Figure 3.4 has a final functional value of 0.12 compared to 0.10 for figure 3.5 along with a lower final residual value (45.95 versus 46.13), representative of an inversion that is closer to the true model. Highlighted by figure 3.3 is what can occur if a frequency of too high a value is used. The waves are unable to penetrate through the velocity contrasts associated with the circular anomalies in the phantom model. They are therefore only able to reconstruct the high impedance contrasts (i.e. that at the top of the positive anomaly, and the bottom of the negative anomaly), producing a poor inversion that has not been recovered below these features, as the wave-field is unable to penetrate through this layer. A high value frequency can also cause the inversion to become unstable and artefacts can be integrated into the reconstruction due to the CFL conditions not being met.

From the data in Appendix B the values of the functional are high for all inversions run on the phantom

model, in the order of 10^4. The phantom model was also poorly reconstructed as shown by figure 3.2, with the final residual value equal to 72.74, indicative of a model far from the observed data. Values as high as this are associated with cycle skipping, suggesting the starting model is not close enough to the true model to converge towards the correct minima. FWI uses a perturbed 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 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, causing either the model to converge towards the incorrect local minima, or the value of f will not reduce, getting bigger as the perpetuated guess moves further away from the true model (*Shah et al.*, 2012). To mitigate this issue the true model can be blurred to reduce any sharp boundary contrasts, producing an image that is easier to invert, and closer to the initial guess. This is however only a solution when using synthetic data. 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 robust and objective means " to mitigating this problem via the spatial continuity of windowed, low-frequency, phase differences. The implementation of such a method should be considered to ensure a more reliable inversion algorithm.

Additional reasons for the poor reconstruction of the phantom model are that absorbing boundary conditions have not been modelled. These should be included to reproduce a semi-infinite model geometry and prevent false reflections from the model boundaries from contaminating the modelled wave-fields. These false reflections can be seen in figure 2.8. Further to this, a free-surface - a surface subject to zero parallel shear - should also be implemented within the model. Solid-fluid interface phenomena such as Rayleigh or ground roll waves etc. arise naturally when running inversions with FD operators and interfere with the source wave-field. Free-surface conditions can be implemented under the staggered formulation of *Moczo et al.* (2002).

A more successful inversion is shown by the Marmousi model in figure 3.4 and figure 3.5. The functional values are both significantly closer to zero than those from the phantom model, at 0.12 and 0.10 for 3Hz and 5Hz respectively. The reconstructions however are not perfect. One issue contributing to this is that the starting guess for Marmousi is a smoothed version of the true model. When smoothing the model in velocity, travel times are altered to produce the new image. If however the model is

inverted to show slowness ($frac1v$) and smoothed, travel times are preserved leading to more accurate reconstructions.

As seismic surveys are often carried out offshore, the models that FWI are inverting for can have a water body associated with them. Shown in figure 3.1, is how the boundary between the subsurface and water has become distorted, caused due to inversion attempting to invert for the water properties rather than the surrounding rock. As the properties of water can be taken as being constant they can be set *a priori* to prevent the inversion being influenced by the values. This can be achieved by creating a mask that sets the gradient equal to zero for any velocity value below 1501 ms^{-1} (the average velocity of salt water). Figure 3.2 highlights the difference that this heuristic makes in the inversion process, where a clear boundary between water and seabed can be seen.

As the energy of the sources propagates out it is dispersed over a larger area causing spherical spreading and attenuation. In terms of FWI this means 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 waves. Figure 3.4 and figure 3.5 highlight this phenomenon, where anomalously high velocity values can be seen at the points where the sources/receivers are located, as at these points the wave has not experienced any energy decay. In 2D the ratio between energy decay per unit area is proportional to the square of the distance between the source and the wave-field ($A \approx \frac{1}{r^2}$) (*Guasch*, 2012). Therefore, when inverting these points produce anomalously high values. To account for this a Hessian preconditioned can be used, 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 SD gradient is very successful at producing a faster convergence rate and a better resolution image.

To ensure the FWI algorithm is able to reconstruct a realistic value of the subsurface, constraints have been applied to prevent seismic velocity values being included that are unrepresentative of geological properties. In this case velocities below 1500 ms^{-1}, as this would be slower than the water velocity, and above 4500 ms^{-1}, highly a metamorphosed rock, are set *a priori*.

Limitations of the notebook format

The notebook is intended to be a practical way to implement FWI such that a model can easily be placed into it, however this means the model must have certain criteria to work with the algorithm. Included in this criteria are that the model being the transpose of the true data, frequency should be

stated in kHz and velocity in mms^{-1}, the model should also be written in Python 3, as ipyparallel is not compatible with Python 2.7. Further to this, parameters are intended to be simple to change including frequency, grid spacing and receiver run time, but due to the use of Jupyter Notebook, when changes are made the cluster must be re started for them to be recognised. When running the algorithm, if connection is lost to the notebook, it is not possible to reload the current inversion despite the cluster still being active. This presents a distinctive limitation, and has been a consistent issue throughout and limited the number of inversions that have been run, and making an already extensive debugging process even longer.

4.2.1 Issues encountered and future work

Inversion is a computationally demanding algorithm, and can therefore quickly become bigger than the allocated RAM. The phantom 2D model with a grid spacing of 666 by 200 is unable to run below 125 GB of computational memory with greater than 5 sources being used per iteration (out of a total of 666). The amount of inversions that have been run was limited by this fact, as it was only possible to run one model at a time, with each taking around 3 hours to complete. The extension of the FWI algorithm to a non-isotropic, attenuating, 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 mathematics then remains the same, applying the same practices to the new equation. To include these properties a greater amount of RAM would be needed, but this is a feasible requirement.

A large proportion of the time taken for this project evolves around debugging the FWI algorithm to be able to produce a successful inversion. This is a time consuming process due to the multiple 'layers' of the new methodology, with both high and low level languages being used, and although inversions were produced, improvements are still needed as noted by the above mentioned discussion. The most notable of which are,

1. Hessian preconditioner
2. Preventer of cycle skipping
3. A second order gradient method
4. Extension to include further model parameters

Chapter 5

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.

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*.

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.

Moczo, P., J. Kristek, V. Vavryčuk, R. J. Archuleta, and L. Halada (2002), 3d heterogeneous staggered-grid finite-difference modeling of seismic motion with volume harmonic and arithmetic averaging of elastic moduli and densities, *Bulletin of the Seismological Society of America*, *92*(8), 3042–3066.

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 Adjoint

The adjoint-state method is a widely 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 untouched, (for example normalised). That is the objective 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.2 Devito setup

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

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

A.3 Parallel computing

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.

A.4 Preconditioner

A preconditioning strategy is used when conducting a second-order method for calculating the inverse of the Hessian operator. It creates an approximation of the Hessian that is computed from analytic or semi-analytic 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 preconditioned for the Hessian matrix can be defined by

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

Appendix B

Results data

Iteration	Functional (10^{-4})	Step length α
1	2401.98	42.40
2	2157.40	22.99
3	1662.40	19.14
4	1514.80	16.96
5	1269.96	12.01
6	1080.50	16.24
7	998.96	29.36
8	945.40	33.29
9	817.80	23.45
10	671.21	20.23
11	640.74	16.23
12	650.25	27.19
13	524.91	4.89
14	564.42	14.89
15	441.22	14.64
16	446.43	6.31
17	439.89	11.79
18	493.49	5.34
19	455.97	4.18
20	440.94	9.83
21	444.97	5.05
22	511.49	6.67
23	355.47	2.97
24	436.29	1.27
25	477.35	4.64
26	380.40	3.44
27	434.85	17.52
28	420.56	1.49
29	446.94	2.52
30	363.28	5.38

Table B.1: Phantom 2D inversion data with a wavelet frequency of 3Hz for 30 iterations

Iteration	Functional (10^{-4})	Step length α
1	922.87	40.99
2	786.50	15.47
3	734.63	11.42
4	733.05	11.09
5	694.75	14.04
6	442.32	6.31
7	430.00	8.85
8	368.36	17.37
9	308.78	15.57
10	327.79	7.80
11	273.58	8.93
12	283.37	5.80
13	226.48	4.82
14	233.87	8.57
15	236.67	1.50
16	238.17	10.27
17	219.37	2.26
18	227.16	3.27
19	249.41	9.84
20	224.23	0.69
21	245.68	2.04
22	225.95	4.01
23	200.82	4.74
24	223.52	5.03
25	207.05	12.98
26	211.22	21.24
27	222.57	8.93
28	208.16	2.28
29	202.28	21.08
30	196.97	1.98

Table B.2: mask with poor reconstruction

Iteration	Functional	Step length α
1	1.68	4.55
2	0.66	6.62
3	0.56	10.62
4	0.47	15.45
5	0.70	7.12
6	0.38	10.28
7	0.56	11.65
8	0.38	11.29
9	0.43	7.72
10	0.37	18.75
11	0.26	16.35
12	0.28	4.72
13	0.27	5.23
14	0.25	7.41
15	0.21	2.56
16	0.19	10.88
17	0.19	4.87
18	0.19	4.55
19	0.23	8.84
20	0.16	6.04
21	0.15	2.57
22	0.14	6.33
23	0.14	1.56
24	0.11	5.83
25	0.14	4.10
26	0.12	2.52
27	0.10	2.71
28	0.13	5.85
29	0.10	2.29
30	0.12	2.05

Table B.3: Marmousi reconstruction with frequency at 3Hz

Iteration	Functional	Step length α
1	1.78	6.87
2	0.95	5.14
3	0.69	26.16
4	0.58	13.30
5	0.37	19.91
6	0.41	12.25
7	0.37	17.97
8	0.32	10.56
9	0.33	11.30
10	0.26	8.67
11	0.24	9.26
12	0.23	4.29
13	0.22	4.06
14	0.21	6.20
15	0.19	8.76
16	0.19	5.37
17	0.18	3.29
18	0.18	9.26
19	0.15	2.50
20	0.15	3.68
21	0.13	10.49
22	0.13	3.73
23	0.12	5.27
24	0.13	4.14
25	0.11	2.61
26	0.11	3.38
27	0.09	0.92
28	0.12	2.75
29	0.09	2.70
30	0.10	3.27

Table B.4: Marmousi 5hz

Iteration	Functional (10^4)	Step length α
1.00	9.09	-7.94
2.00	8.61	8.37
3.00	7.66	6.75
4.00	4.49	5.88
5.00	3.63	3.06
6.00	3.42	5.40
7.00	3.68	3.13
8.00	3.69	4.18
9.00	3.12	1.20
10.00	3.86	1.29
11.00	3.39	0.98
12.00	3.18	5.68
13.00	3.27	3.99
14.00	2.97	1.20
15.00	3.42	4.73
16.00	3.63	4.77
17.00	3.15	2.59
18.00	3.37	3.01
19.00	3.35	5.04
20.00	3.48	0.96
21.00	3.40	2.90
22.00	3.27	3.02
23.00	3.12	8.74
24.00	3.16	1.37
25.00	2.81	3.22
26.00	2.87	9.14
27.00	3.08	0.40
28.00	3.10	0.97
29.00	2.96	3.65
30.00	2.97	0.84

Table B.5: Phantom 2D inversion data with a wavelet frequency of 15Hz for 30 iterations high frequ contrast

Appendix C

Scripts

C.1 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
```