Imperial College London
Department of Earth Science and Engineering

Computational and Numerical Aspects of Full Waveform Seismic Inversion

Timothy James Burgess

October 2018

Supervised by Prof. Mike Warner and Prof. Joanna Morgan

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Petroleum Geophysics of Imperial College London and the
Diploma of Imperial College London

Declaration of Originality

I herewith certify that all material within this thesis that is not my own work has been duly acknowledged.

Timothy Burgess

Copyright Declaration

Abstract

Full-waveform inversion (FWI) is a nonlinear optimisation procedure, seeking to match synthetically-generated seismograms with those observed in field data by iteratively updating a model of the subsurface seismic parameters, typically compressional wave (P-wave) velocity.

Advances in high-performance computing have made FWI of 3-dimensional models feasible, but the low sensitivity of the objective function to deeper, low-wavenumber components of velocity makes these difficult to recover using FWI relative to more traditional, less automated, techniques.

While the use of inadequate physics during the synthetic modelling stage is a contributing factor, I propose that this weakness is substantially one of ill-conditioning, and that efforts to remedy it should focus on the development of both more efficient seismic modelling techniques, and more sophisticated preconditioners for the optimisation iterations. I demonstrate that the problem of poor low-wavenumber velocity recovery can be reproduced in an analogous one-dimensional inversion problem, and that in this case it can be remedied by making full use of the available curvature information, in the form of the Hessian matrix.

In two or three dimensions, this curvature information is prohibitively expensive to obtain and store as part of an inversion procedure. I obtain the complete Hessian matrices for a realistically-sized, two-dimensional, towed-streamer inversion problem at several stages during the inversion and link properties of these matrices to the behaviour of the inversion. Based on these observations, I propose a method for approximating the action of the Hessian and suggest it as a path forward for more sophisticated preconditioning of the inversion process.

Acknowledgements

The last four years have been quite a journey, and fortunately not one I have had to take alone. There are many who deserve my thanks.

I would like to sincerely thank my supervisors, Profs Mike Warner and Jo Morgan, for their guidance, patience, and feedback. You have given me the freedom to explore my own ideas and steer my own course, yet were always there to offer assistance when I needed it. Thanks must also go to Prof. Felix Herrmann and Dr Lluis Guasch, for countless thought-provoking chats about anything and everything inversion-related, and to Adrian Umpleby, for his boundless patience for my FULLWAVE3D questions.

I also extend special thanks to Dr Matt Lamont and Downunder Geosolutions, both for encouraging me to do this in the first place and for their generous support, moral and financial.

To my Imperial colleagues, both within the FULLWAVE group and without, I owe much of my remaining sanity. You have always been there to offer both encouragement and distraction as needed. I will miss working, and not working, with all of you.

To Mum and Dad: your love, sacrifice and encouragement over the years are what put me on this amazing path and provided me with this opportunity.

To Stu, for your generosity, encouragement, wisdom and support over nearly 20 years.

And finally, to Helen. Without all of your cheerleading, love, and understanding, I would not be where I am today.

Contents

List of Tables

List of Figures

To Helen

"A complex system that works is invariably found to have evolved from a simple system that works."

— John Gaule

1 Introduction

The science of using seismic waves to infer information about the properties of the Earth through which they have travelled is now well more than a century old. The term "seismology" itself is credited to Robert Mallet, and it is his paper "The dynamics of earthquakes" (1848) that is considered to be one of the foundational publications of modern seismology. Mallet spent several years measuring the velocity of seismic waves through sand and rock, before applying his discoveries to the great Neapolitan earthquake of 1857 to estimate the location of the energy source, coining the now-ubiquitous term "epicenter" in the process (Mallet, 1862).

Since then, observations of seismic energy from earthquakes have allowed seismologists to infer much of what we know of the interior of the Earth: the existence of an outer (Oldham, 1906) and inner (Lehmann, 1936) core, and the Moho discontinuity (Mohorovičić, 1910). Meanwhile, the burgeoning hydrocarbon industry also saw potential in this technique, with refraction seismology being used as early as 1921 to detect subsurface salt domes (Keppner, 1991). These studies were also notable for their use of controlled sources, rather than naturally-occurring seismic events. The first commercial discovery of oil from this technique came not long after, in 1924 (Sheriff and Geldart, 1995).

Around the same time, other practitioners started to explore the usefulness of controlled-source signals reflected from interfaces deeper below the surface, rather than those scattered at wide angles in the upper layers. However, data obtained from near-normal-incidence reflections can be more difficult to work with than that from transmitted arrivals: the high sensitivity of the signal to the unknown position of the reflector complicates the disambiguation of the velocity and density parts of the model. Thus, the use of reflection data was, and still largely is, considered a separate technique from refraction seismology, with its own algorithms and target applications, and images generated from reflection seismic data are typically of "reflectivity" (Bleistein et al., 1985) rather than seismic velocity.

Full Waveform Inversion (FWI) elegantly brings these two branches of seismology together, starting from the observation that if it were possible to compute a wave-equation simulation of a seismic experiment performed in the real world, then it might be possible to adjust the parameters of the simulation until the simulated data matched the observed data - both the refracted and reflected arrivals. At that point, it could be (somewhat incorrectly, as I shall later demonstrate) concluded that the model used in the simulation was an accurate representation of the physical properties of the subsurface.

In its current form, FWI is most frequently attributed to Tarantola (1984), although the underlying adjoint-state technique which makes it feasible arose at least a decade earlier, in e.g. Lions and Magenes (1972) and Chavent (1974). The adjoint-state technique allowed efficient gradient-based optimisation schemes, which are more or less essential when minimising functions of so many variables, to be applied to the matching of the simulated and observed data. All that is required is a single objective function, mapping the model parameters to something to be minimised, and constraints to limit the space of feasible models. A great deal of the published work (e.g. Gauthier et al. (1986); Sirgue and Pratt (2004); Virieux and Operto (2009); Li and Demanet (2016)), discusses the simplest formulation of the misfit function, which I additionally restrict here to real-valued vectors, as subsequent discussion will only be in the time domain:

$$\begin{aligned} f &= \sum_i f_i \\ &= \frac{1}{2}\mathbf{r}_i^T \mathbf{r}_i \\ \mathbf{r}_i &= \mathbf{d}_{\text{modelled},i} - \mathbf{d}_{\text{observed},i} \end{aligned} \tag{1.1}$$

With i denoting the index of a given seismic experiment, or shot.

Especially for field datasets, this simple formula can omit a considerable amount of complexity, including muting out of seismic arrivals that are not expected to be adequately modelled, matching of amplitudes between observed and predicted data, and filtering to low acoustic frequencies (Warner et al., 2013). Relating it back to the formulation originally presented in Tarantola (1984), Pratt et al. (1998) notes that the above formula, in effect "assumes an identity data covariance matrix and infinite *a priori* model variances". Nonetheless, it is sufficient for introductory purposes.

The elegance of the adjoint-state method can be demonstrated in just a few lines of derivation, which also serves to introduce the reader to notation I will use throughout this thesis. The above equation (1.1) defines the misfit function in terms of the modelled data for each shot, $\mathbf{d}_{\text{modelled},i}$. It is now necessary to define the relationship between this modelled data and the vector of model parameters $\mathbf{m}$ from which it is generated. To do this, I introduce two new objects: a generic wave equation $\mathbf{A}_i(\mathbf{m})\mathbf{u}_i = \mathbf{s}_i$, which is solved to obtain the full seismic wavefield $\mathbf{u}_i$; and a "picking matrix" $\mathbf{P}_i$, which defines how the modelled data is sampled from this full wavefield for each shot. Note that this form of the wave equation assumes nothing other than linearity between the source term $\mathbf{s}_i$ and the resulting wavefield, and there are many possible forms for the operator $\mathbf{A}(\mathbf{m})$: acoustic, elastic, constant- or variable-density, with or without attenutation, with or without anisotropy.

To see how to construct the gradient of the misfit function in terms of the model parameters, differentiate this generic wave equation with respect to any single model parameter m_k, and note that the source term does not depend on the model parameters:

$$\begin{aligned} \mathbf{A}_i \mathbf{u}_i &= \mathbf{s}_i \\ \frac{\partial}{\partial m_k} \mathbf{A}_i \mathbf{u}_i &= \mathbf{0} \\ \frac{\partial \mathbf{A}_i}{\partial m_k} \mathbf{u}_i + \mathbf{A}_i \frac{\partial \mathbf{u}_i}{\partial m_k} &= \mathbf{0} \\ \frac{\partial \mathbf{u}_i}{\partial m_k} &= -\mathbf{A}_i^{-1} \frac{\partial \mathbf{A}_i}{\partial m_k} \mathbf{u}_i \end{aligned} \tag{1.2, 1.3}$$

Then, use the definition of the picking matrix $\mathbf{P}$ and the misfit function (1.1):

$$\begin{aligned} \mathbf{r}_i &= \mathbf{d}_{\text{modelled},i} - \mathbf{d}_{\text{observed},i} \\ &= \mathbf{P}_i \mathbf{u}_i - \mathbf{d}_{\text{observed},i} \\ \frac{\partial f_i}{\partial m_k} &= \mathbf{r}_i^T \frac{\partial \mathbf{r}_i}{\partial m_k} \\ &= \mathbf{r}_i^T \mathbf{P}_i \frac{\partial \mathbf{u}_i}{\partial m_k} \\ &= -\mathbf{r}_i^T \mathbf{P}_i \mathbf{A}_i^{-1} \frac{\partial \mathbf{A}_i}{\partial m_k} \mathbf{u}_i \end{aligned}$$

At this point, it is helpful to introduce the adjoint wave equation matrix, representable in the time domain as a transpose $\mathbf{A}^T$. The physical interpretation of this transposition is primarily a time-reversal of the original matrix $\mathbf{A}$, although depending on the exact form of $\mathbf{A}$ other effects may also be introduced. Making use of the matrix identity $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$, one obtains:

$$\frac{\partial f_i}{\partial m_k} = -(\mathbf{A}_i^{-T} \mathbf{P}_i^T \mathbf{r}_i)^T \frac{\partial \mathbf{A}_i}{\partial m_k} \mathbf{u}_i \tag{1.4}$$

The form of the final equation (1.4) is informative, and in it lies the key that makes FWI affordable. The computationally expensive parts of the right hand side of (1.4) are the evaluation of the wavefield $\mathbf{u}_i = \mathbf{A}_i^{-1} \mathbf{s}_i$, and the evaluation of the backpropagated residual $(\mathbf{A}_i^{-T} \mathbf{P}_i^T \mathbf{r}_i)$. Since neither of these depend on which model parameter k we wish to obtain the gradient with respect to, these must only be evaluated once for each shot i to obtain all the gradients $\partial f_i/\partial m_k$.

This breakthrough, followed by advances in computer technology that allowed for more efficient solutions of the wave equation and its adjoint, have thrust FWI into the limelight over the past two decades: Virieux and Operto (2009) present a comprehensive review of this development, with an extensive bibliography.

Many early applications, such as Pratt and Shipp (1999), performed the modelling and inversion in the frequency domain, in two dimensions. This allowed for considerably reduced memory usage, as the wave equation for each frequency could be solved separately, with the size of the solution vector $\mathbf{u}_i$ being only the number of spatial points. Furthermore, for many sources, the wave equation matrix $\mathbf{A}$ could be factorised once for each frequency, and this factorisation used cheaply to compute the wavefield for many sources. As inversions pushed into three dimensions and to higher frequencies, however, the increased domain sizes meant that such factorisations were no longer efficient (Umpleby et al., 2010). While research continues into frequency domain methods (e.g. van Leeuwen and Herrmann (2014); Wang et al. (2016)), and they do offer some unique capabilities (e.g. van Leeuwen et al. (2014)), the comparable efficiency and extra flexibility afforded by time-domain solvers - in particular, the ability to selectively mute and weight parts of the wavefield - has meant that the FULLWAVE research group at Imperial College London now tends to use them exclusively.

For all the surge in research interest, however, FWI has yet to live up to the hope that many, including myself, have for it: single-process, simultaneous macromodel and reflectivity image construction directly from raw seismic data. For many years, a significant barrier was the requirement for the starting velocity model to be close enough to the true model that the modelled traveltimes of transmitted arrivals matched the observed traveltimes to within half a cycle at the frequencies used in the inversion. Failure to satisfy this criterion could lead to spurious models, where velocities could be updated in the opposite direction to that needed, pushing the objective function into a local minimum, where predicted and observed events appeared mostly aligned, but were actually out of phase by a multiple of 2π, often referred to as “cycle-skipping” (Virieux and Operto, 2009; Warner and Guasch, 2014). This either incurred a requirement for significant velocity analysis effort before FWI could be applied, or restricted applications of FWI to datasets which contained sufficiently low noise levels at the lowest acoustic frequencies (Shah et al., 2012; Warner et al., 2013).

Warner and Guasch (2014) presented significant progress against this problem by constructing a different objective function based on matching filters, improving the convexity of the misfit surface and allowing for a far broader range of starting models to be used. While other research taking different approaches to this problem continues, e.g. Li and Demanet (2016); Métivier et al. (2016), it appears to me that mainstream success in this area is imminent, and thus this problem is not the focus of this work.

A more resilient challenge for FWI researchers has proven to be the recovery of low wavenumber velocity information from short-spread reflected arrivals (Virieux and Operto, 2009). Many techniques have been proposed to work around this problem (Xu et al., 2012; Brossier et al., 2014; Ramos-Martinez et al., 2016; Irabor and Warner, 2016), and although the uptake of these within the petroleum industry was initially slow (Morgan et al., 2013), there is now considerable enthusiasm around obtaining these low-wavenumber components from reflection data using waveform inversion. Nonetheless, the most successful applications of FWI to date have targeted the recovery of high-resolution models of the near-surface, down to the maximum depth well-sampled by transmitted arrivals - often at most one-third of the maximum source-receiver offset. Where the desired imaging targets have been deeper than this, these high-resolution models of the overburden have still improved the quality of migrated images of the reflections from the deeper areas (e.g. Warner et al. (2013); Houbiers et al. (2012)).

In this thesis, I will argue that the lack of effectiveness of FWI in recovering these deeper low-wavenumber velocity components is at least partially due to poor numerical conditioning of the FWI problem, and that addressing that conditioning is going to be an important step towards any remedy. I will also argue that many existing approaches to this problem indirectly attack the conditioning, and that goes some way to explaining their success.

I also propose that this observation inextricably links together the performance of FWI with its applicability: recovery of these ill-conditioned model components will likely require vastly more inversion steps, or perhaps more expensive steps, than are necessary to recover detailed images of the region nearer the surface. Therefore, I attack this problem from two sides: through the development of improved preconditioners, to reduce the number of optimisation iterations required, as well as the implementation of techniques to reduce the cost of those iterations. I should note that my aim is not to present an easy solution to this issue - the field seems crowded with researchers and practitioners alike sharing that goal, and I am not convinced it is a realistic one. Rather, it is to shed light on some of the underlying reasons for why inversions behave the way they do, and hope that this helps to point the arrow of future research in a productive direction.

My contributions to the field are fivefold:

- I have integrated a high-performance commercial wave equation solver into the group's flagship software package FULLWAVE3D, including a novel self-tuning algorithm, allowing FWI to be performed faster, on larger models and to higher frequencies than was previously possible.
- I have developed a simple, modular FWI code in a high-level language that is taught to earth science students, to facilitate faster research into FWI techniques.
- Using the above, I have devised an inexpensive improvement to the diagonal preconditioning formula traditionally used in FWI, taking into account both source and receiver placement and giving a 20-60% speedup in the convergence of test problems.
- I have analysed the Hessian matrix of a 1.5D synthetic FWI problem, connected this analysis to several published observations about the behaviour of the inversion, and demonstrated that in this test problem, off-diagonal preconditioning is able to extract significant low-wavenumber model updates from reflection data using FWI.
- For the first time, I have obtained and analysed the complete Hessian matrix for a realistically sized 2D FWI problem.

These contributions are dealt with individually in the forthcoming chapters.

"Machines take me by surprise with great frequency."

— Alan Turing

2

Improving FWI efficiency with high-order modelling kernels

2.1 Introduction

Since solution of the wave equation is the overwhelmingly dominant computational cost involved in Full Waveform Inversion, it is obviously the prime target for efficiency improvements. While the overall number of wave equation solves required can be reduced by composite encoded shots (Ben-Hadj-Ali et al., 2011), or subsampling of shots at each iteration (Díaz and Guitton, 2011; van Leeuwen and Herrmann, 2013), it is also worthwhile to consider how the cost of each solution might be reduced. For a time-domain wave equation solver, to which I restrict this discussion for reasons outlined in the introductory chapter, the computational cost of a single solution of the wave equation can be expressed as the product of three factors:

- The number of space-time grid points for which the wavefield must be computed
- The number of arithmetic operations required to update each of these space-time points
- The (reciprocal) speed at which these operations can be computed by the available hardware

The number of grid points that must be solved for clearly depends on the size of the volume for which the wave equation must be solved, the total time we are solving for, and the density of the space-time grid points. So to reduce the number of grid points required, one or more of these must be reduced also. The total simulation time required is typutionically fixed by the length of the data we are trying to match, leaving only total volume and point density as targets for improvement.

There are a handful of optimisations which reduce the volume to be solved. If sources and receivers for a particular shot are confined to one section of the model - as is the case with data acquired in a towed-streamer configuration - it is relatively common to make the approximation of confining the solution to the wave equation for that shot to an aperture around that section, while using an appropriate absorbing boundary condition. This approximation is equivalent to assuming that waves originating in the simulation area will not be scattered back in once they have left it. Another technique for volume reduction is based on the velocity of propagation in the continuous medium: spacetime points that are not physically reachable by a wave starting at the source point at the earliest time need not be solved for. This is often referred to as the "expanding box" optimisation.

The primary focus of this chapter, however, is in reducing the density of the grid points. I will demonstrate that using more sophisticated finite difference schemes, despite often incurring a higher number of arithmetic operations per point, can allow an overall reduction in the time required for wavefield computation, both through a reduction in the overall point count and more efficient mapping onto modern computer hardware.

In the next few sections I review the theory behind these schemes, and propose a quantitative criteria for determining the minimum grid density to use for a particular problem. I will then describe the work I undertook to integrate into FULLWAVE3D a high-performance commercial wave equation solver which implements some of the discussed techniques.

Original work

Note that while some of the dispersion analysis is novel, in particular the focus on group velocity and the proposal of a quantitative dispersion criterion, this chapter contains very little in the way of original research. Rather, it is mostly a review, and a setup for the subsequent developments. Nonetheless, this was significant work that now forms part of the distributed FULLWAVE3D package, and is routinely used by at least two of the group's commercial sponsors, allowing for higher-frequency inversions than would previously have been possible within hardware constraints.

2.2 Lower limits on the spatial grid density

Since the number of grid points is one of the important factors in the runtime of any wave equation solver, it is obviously desirable to have as few points as possible while still obtaining a sufficiently accurate result. But just how large can the spacing be?

One important limit is described in Shannon (1949), which states that a signal with bandwidth B requires a sample rate of at least $2B$ to be precisely represented - this statement is also known as the "Nyquist-Shannon sampling theorem". In the context of the wave equation, this places an absolute upper limit on both the spatial and temporal sampling - for a discretisation to faithfully represent a signal with acoustic frequencies in the range $[0, f]$, the time sampling interval must be smaller than $1/2f$ and the spatial sampling interval less than $c_{\min}/2f$, where $c_{\min}$ is the lowest velocity in the medium. However, this is rarely the strictest limitation on the sample rates.

A stricter requirement than Shannon's stems from the requirement that the solution of the discretised equations is sufficiently close to the solution of the continuous system being approximated. Consider the simplest case of the acoustic wave equation in a constant-density medium:

$$\frac{1}{c(\mathbf{x})^2}\frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla^2 \mathbf{u} + \mathbf{s} \tag{2.1}$$

Expressing the above equation in a discretised grid requires a discrete analogue of the derivative operators $\partial^2/\partial t^2$ and ∇^2. When discretising using the finite difference method, these derivatives are typically replaced with approximations derived from the Taylor expansions, which express them in terms of the function values at nearby points:

$$\begin{aligned}
f(x \pm \Delta x) &= f(x) \pm \frac{\partial f}{\partial x}\Delta x + \frac{\Delta x^2}{2}\frac{\partial^2 f}{\partial x^2} \pm \frac{\Delta x^3}{6}\frac{\partial^3 f}{\partial x^3} + O(\Delta x^4) \\
\Delta x^2 \frac{\partial^2 f}{\partial x^2} &= f(x+\Delta x) + f(x-\Delta x) - 2f(x) + O(\Delta x^4) \\
\frac{\partial^2 f}{\partial x^2} &= \frac{1}{\Delta x^2}(f(x+\Delta x) - 2f(x) + f(x-\Delta x)) + O(\Delta x^2) \\
&= D_{2x} f(x) + O(\Delta x^2)
\end{aligned}$$

The operator D_{2x} is equivalent to convolution with a short, spatially invariant sequence $(\Delta x^{-2}, -2\Delta x^{-2}, \Delta x^{-2})$, sometimes referred to as a finite difference *stencil*. The approximation $D_{2x} \approxeq \partial^2 f/\partial x^2$ is referred to as second-order because the error in the approximation scales with the square of the grid spacing. Higher order approximations are possible using more nearby points; a table of several such approximations is shown in Table 2.1.

Name	Order	Coefficients
2nd-order Taylor (D_2)	2	[1, -2, 1]
4th-order Taylor (D_4)	4	[-1/12, 4/3, -5/2, 4/3, -1/12]
8th-order Taylor (D_8)	8	[-1/560, 8/315, -1/5, 8/5, -205/72, 8/5, -1/5, 8/315, -1/560]

Table 2.1: Coefficients of several Taylor-series-derived second-derivative finite difference stencils
Coefficients of several second-derivative finite difference stencils derived from Taylor series. The 4th-order Taylor stencil is the one used by most kernels in FULLWAVE3D.

To examine the nature of the errors introduced by these approximations, I first consider the effect of applying these finite difference operators to a discretely-sampled sinusoidal signal. Using the example of the 2nd-order spatial Taylor operator D_{2x} defined above:

$$\begin{aligned}
f(x) &= \cos(kx) \\
D_{2x}f(x) &= (\Delta x^{-2})(\cos(k(x-\Delta x)) - 2\cos(kx) + \cos(k(x+\Delta x))) \\
&= (\Delta x^{-2})2\cos(kx)(\cos(k\Delta x) - 1) \\
\partial^2 f/\partial x^2 &= -k^2\cos(kx) \\
D_{2x}f(x)/(\partial^2 f/\partial x^2) &= 2(k\Delta x)^{-2}(1-\cos(k\Delta x)) \\
&= 1 - \frac{(k\Delta x)^2}{12} + O((k\Delta x)^4)
\end{aligned}$$

The above derivation demonstrates a couple of important points:

1. that sinusoidal signals are eigenfunctions of the finite difference operator (this is true of any convolution)
2. that the effect of the approximation, in the case of the 2nd-order Taylor stencil at least, is to systematically underestimate the amplitude of the derivative, with the effect worsening as the wavenumber increases

The effect of this approximation on the modelled wavefields can be understood by substituting a plane wave solution into the homogenous (i.e. source-free) discretised wave equation, and deriving a relationship between the wavenumber k and the angular frequency ω. For illustration I will use the 2nd-order Taylor stencil in time, and the 4th-order Taylor stencil in space. A 1-D system suffices for this example:

$$\begin{aligned}
u(x,t) &= A\cos(kx - \omega t) \\
D_{2t}u &= c^2 D_{4x}u \\
\frac{2\cos\omega\Delta t - 2}{\Delta t^2}u &= c^2\frac{(-1/6)\cos 2k\Delta x + (8/3)\cos k\Delta x + (-5/2)}{\Delta x^2}u \\
\cos\omega\Delta t &= \frac{c^2\Delta t^2}{\Delta x^2}\left(-\frac{1}{12}\cos 2k\Delta x + \frac{4}{3}\cos k\Delta x - \frac{5}{4}\right) + 1 \qquad (2.2)
\end{aligned}$$

A plot of the resulting curve is shown in Figure 2.1 (top), for parameters $\Delta x = 25$m, $\Delta t = 4$ms, $c = 1500$m/s. Observe that the desired linear dependence $\omega = ck$ is only true

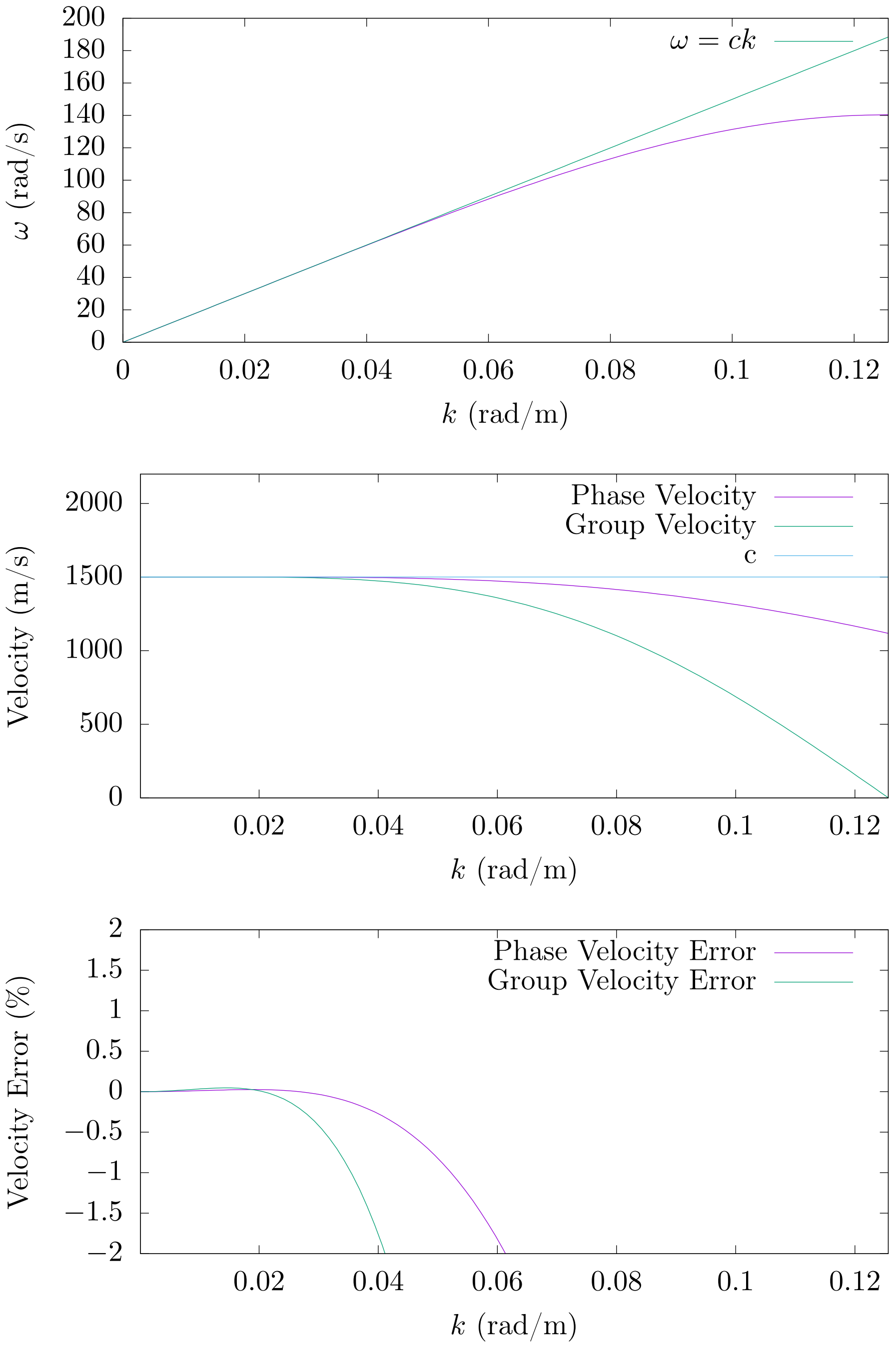


Figure 2.1: Plots showing the nonlinear dependence of frequency vs wavenumber in the computational medium, and the dispersive effect on the phase and group velocity with a 4th-order Taylor stencil in space and a 2nd-order Taylor stencil in time. This is the same configuration as the low-order acoustic anisotropic kernel in FULLWAVE3D. Key parameters: Δx=25m, Δt=4ms, c=1500m/s. Starting from the lowest wavenumbers and moving upwards, the velocity first increases slightly due to the temporal dispersion, then decreases due to the spatial dispersion.

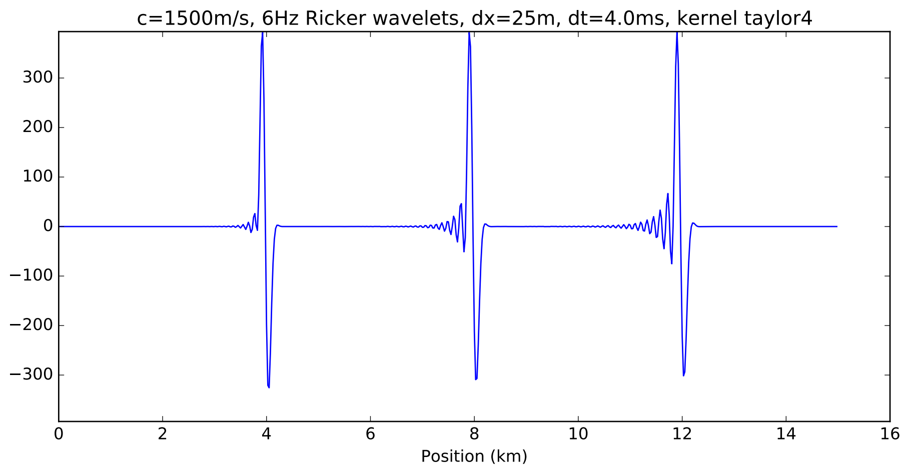
for the smaller wavenumbers. This in turn has an effect on the velocity of propagation in the discretised medium: shorter wavelengths generally propagate slower. The effects on the phase velocity $v_p = \omega/k$ and the group velocity $v_g = d\omega/dk$ (Lighthill, 1965) are shown in the centre and lower plots. The group velocity is important to consider in an error analysis - as shown in the lower plots of Figure 2.1, the group velocity errors can be far greater than those in the phase velocity. This manifests itself as the wave envelope dispersing far more than one might expect from looking at the phase velocities alone: at the highest wavenumbers in this example the phase velocity is only 25% slower than c, but Figure 2.2b clearly shows some wavenumber components propagating more slowly than this.

This dispersion can be seen in the waveforms themselves, as shown in Figure 2.2a - as the pulses move through the medium, they distort and spread, with shorter wavelengths propagating more slowly. As the grid spacing is increased, these effects become significantly worse, as can be seen in Figure 2.2b - the equivalent figure with both space and time grid spacing doubled.

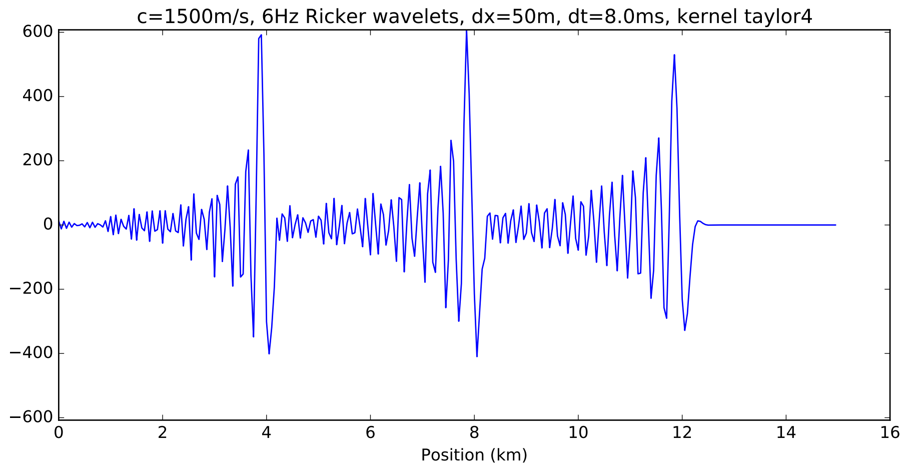
All of this begs the practical question: what is an acceptable level of dispersion? While the answer to this probably depends on the application, I would argue that one sensible criterion is that the velocity error should not introduce more than a half-cycle difference in arrival times of the generated data. If this criterion was violated, some data would be cycle-skipped even in a perfect model. For small errors such as these, this leads to a quantitative limit on the group-velocity error:

$$\begin{aligned}
\Delta T/T &= \Delta v_g/v_g \\
\Delta T &= T\Delta v_g/v_g \\
\Delta T &< 1/2f \\
\Delta v_g/v_g &< 1/(2fT) \qquad (2.3)
\end{aligned}$$

This shows that the maximum acceptable error is a function of the frequency-propagation-time product. I believe that this is an important observation: it is often assumed that the required grid spacing will scale as $\Delta x \propto f^{-1}$, but the dependence can be stronger than this,



(a) 4th-order Taylor stencil, $\Delta x = 25\text{m}$, $\Delta t = 4.0\text{ms}$



(b) 4th-order Taylor stencil, $\Delta x = 50\text{m}$, $\Delta t = 8.0\text{ms}$

Figure 2.2: Wavelets propagating to the right through a 1-D medium from a periodic source at x=0, using a 4th-order Taylor stencil in space and a 2nd-order Taylor stencil in time. As the pulses move to the right they spread, with the shorter wavelengths propagating more slowly. This effect gets dramatically worse as the grid spacing is increased.

depending on the stencil. For the reader's reference, I have used the criterion of equation (2.3) to construct a table (Table 2.2) of the grid spacings for various frequencies and total propagation times, for the common case of $c_{\text{min}} = 1500\text{m/s}$ and the D_4 4th-order Taylor stencil used in the low-order kernels within FULLWAVE3D. Note that these spacings are on the conservative side for field data inversions, both due to there being more than one dimension and because of velocity variations in the medium. The extra dimensions mean that the wavenumber projected along each principal axis tends to be lower than along the direction of propagation, by up to a factor of $\sqrt{2}$ in two dimensions and $\sqrt{3}$ in three. The velocity variations mean that the wavelengths are longer for much of the propagation, thus allowing more accurate derivatives with the same grid spacing. Nonetheless, the values in table 2.2 are valid if accurate modelling of direct arrivals is desired.

f, T	4 sec	5 sec	6 sec	7 sec	8 sec	9 sec	10 sec
6 Hz	37.7m	35.8m	33.9m	32.7m	31.4m	30.8m	29.5m
12 Hz	15.7m	14.8m	14.1m	13.5m	13.2m	12.6m	12.2m
18 Hz	9.4m	8.8m	8.4m	8.2m	8.0m	7.5m	7.3m
24 Hz	6.6m	6.1m	6.0m	5.7m	5.5m	5.3m	5.2m
30 Hz	4.9m	4.6m	4.4m	4.3m	4.1m	4.0m	3.9m
36 Hz	4.0m	3.7m	3.6m	3.5m	3.2m	3.1m	3.1m
42 Hz	3.2m	3.1m	3.0m	2.8m	2.7m	2.6m	2.6m
48 Hz	2.7m	2.6m	2.4m	2.4m	2.3m	2.2m	2.2m
54 Hz	2.4m	2.2m	2.1m	2.0m	2.0m	1.9m	1.9m
60 Hz	2.1m	1.9m	1.9m	1.8m	1.8m	1.7m	1.6m

Table 2.2: A table of maximum grid spacings for the 4th-order Taylor stencil used in FULLWAVE3D, for various frequencies and total propagation times, using the criterion of equation (2.3). The medium is assumed to have minimum velocity 1500m/s. Note that doubling the frequency typically requires more than double the grid density, since a fixed velocity error generates more noticeable dispersion at higher frequencies. These numbers are on the conservative side: both velocity variations and off-axis propagation in 2 or 3 dimensions reduce the errors from numerical dispersion. However, the values above are valid if accurate modelling of direct arrivals propagating along a grid axis is desired.

2.3 Reducing the spatial grid density using improved stencils

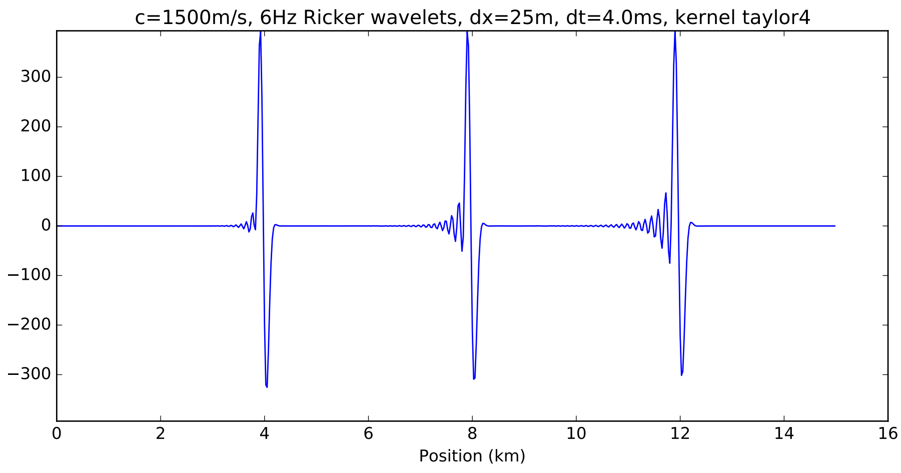
The maximum spacings obtained using the criterion of equation (2.3) are a crucial determinant of FWI performance - as I shall demonstrate shortly, in three dimensions, the runtime scales with Δx^{-4}. It is therefore usually worthwhile to use more accurate stencils than D_{4x}. There are two complementary approaches to improving the accuracy of the approximation: to sample a larger number of nearby points, and balancing the desired accuracy across the largest possible range of wavenumbers. These approaches are illustrated by comparing Figure 2.3a with Figures 2.3c, which uses the 8th-order Taylor-derived stencil, and 2.3b, which uses a 5-point optimised stencil from Zhang and Yao (2013).

Using longer stencils exploits a favourable tradeoff between reducing the number of grid points to be solved for and increasing the number of computations required per point. For instance, the minimum grid spacing for 12 Hz propagation at a duration of 4s shown in Table 2.2 is 15.7m. For the 8th-order Taylor stencil, this goes up to 27m. In three dimensions, this results in 5x fewer spatial grid points and around 1.5x fewer time intervals, at a cost of doubling the number of operations per space-time point - a clear win. It is worth noting that the longer stencils are able to give more accurate derivatives only under the assumption that the wavefield itself is continuous, and will thus introduce larger errors if this assumption is violated, such as would occur near strong medium contrasts such as e.g. salt bodies or the sea floor.

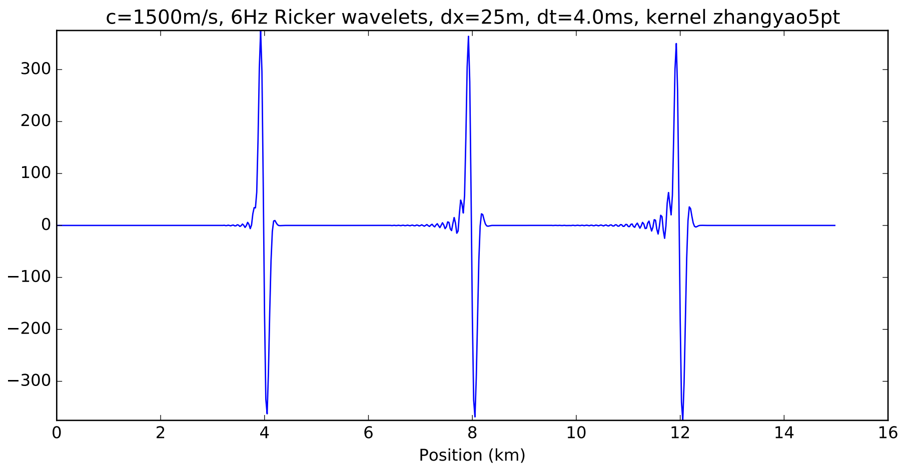
Optimised stencils take a different approach to accuracy improvement. Noting that the velocity errors for the smallest wavenumbers are often tiny, optimised stencils sacrifice some accuracy at the lowest wavenumbers to extend the range of wavenumbers for which the finite difference approximation is within the desired error bound. This has the advantage of requiring no additional computations compared to a Taylor-derived stencil of the same length - only the constant coefficients change. For 12Hz propagation for a duration of 4s, the 5 point stencil of Zhang and Yao (2013) allows for a spacing of 18.8m, giving a 1.8x reduction in runtime compared to D_{4x}.

2.4 Lower limits on the temporal grid density

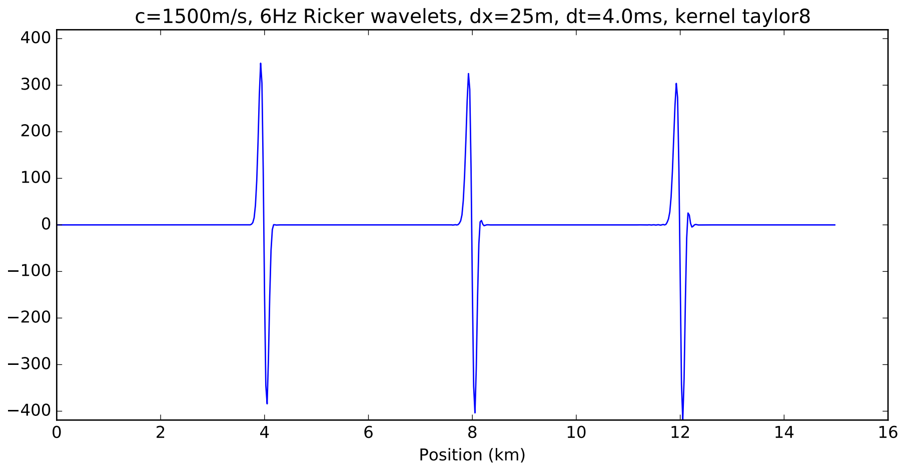
The accuracy of the time derivatives is also important. Figure 2.4 shows waveforms generated using a variety of spatial stencils, for two different timesteps. As the accuracy of the space derivatives increases, the remaining dispersion effects are due to a lack of precision in



(a) 4th-order Taylor stencil, $\Delta x = 25\text{m}$, $\Delta t = 4.0\text{ms}$



(b) Zhang & Yao's 5 point stencil, $\Delta x = 25\text{m}$, $\Delta t = 4.0\text{ms}$, cost is same as 2.3a



(c) 8th-order Taylor stencil, $\Delta x = 25\text{m}$, $\Delta t = 4.0\text{ms}$, cost is twice that of 2.3a

Figure 2.3: Two approaches to improving the accuracy of finite difference modelling. By sampling a large number of nearby points, the 8th-order Taylor finite difference operator is able to better estimate the derivatives over a wider range of wavenumbers compared to the 4th order, at the expense of more calculations per time step. By contrast, Zhang and Yao (2013) adjust the coefficients of the stencil, sacrificing accuracy at the very lowest wavenumbers to improve the approximation at the higher ones without incurring any extra computational expense.

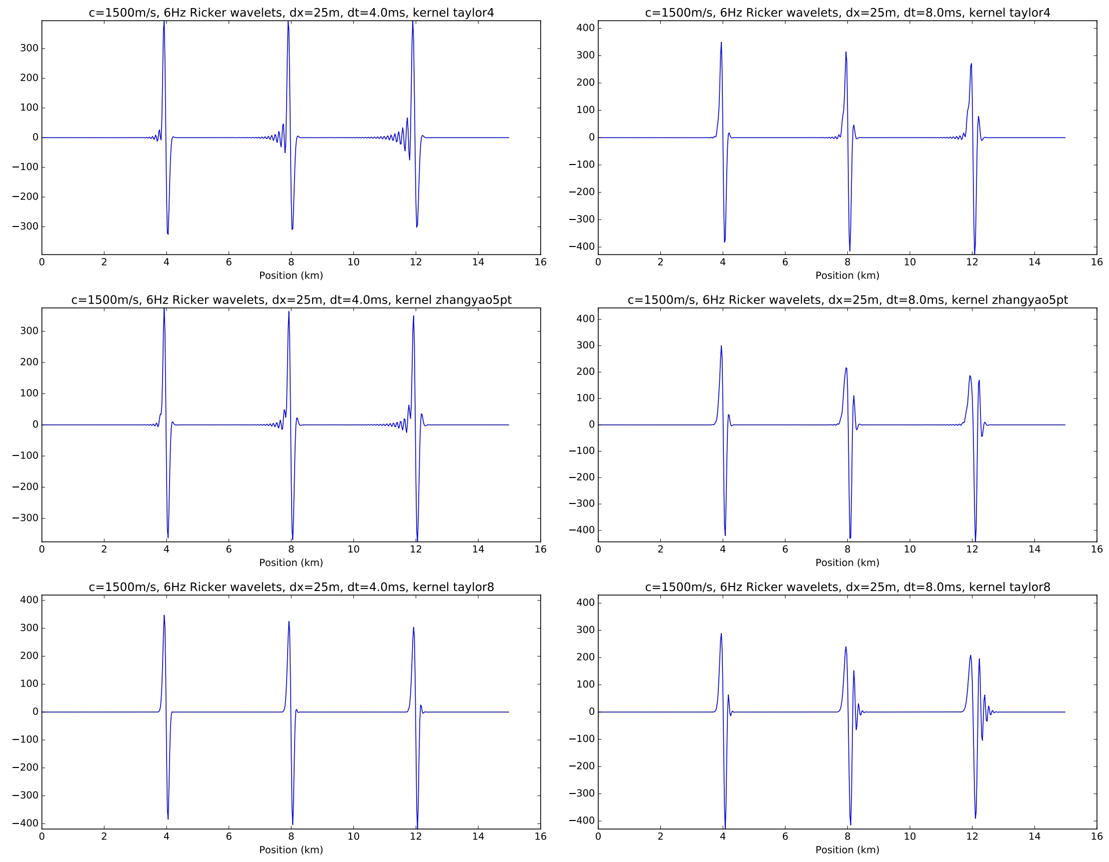


Figure 2.4: Waveforms generated using a variety of spatial stencils, for $\Delta x = 25$m, and $\Delta t = 4$ms (left) or 8ms (right), with increasing quality of the spatial stencils from top to bottom. With a highly accurate spatial stencil such as taylor8 (8th-Order Taylor), most of the visible dispersion effects are due to the 2nd-order approximation in time, which manifests as shorter wavelengths travelling slightly faster than longer ones.

the 2nd-order finite difference operator used in time. These manifest in the waveforms as shorter wavelengths travelling faster than c, rather than slower as is typically the case with dispersion from the spatial operators. This effect is actually visible in Figure 2.1 as the slight positive velocity error for the very small wavenumbers.

While Figure 2.1 depicts the errors for a specific set of parameters, it turns out that the fractional velocity errors arising from particular choices of spatial (and temporal) stencil can be expressed for all parameters in terms of the dimensionless values $k\Delta x$ (in radians), which can range from 0 to π while obeying Shannon's rule, and the "crossing factor" $c\Delta t/\Delta x$, which is the fraction of a grid cell traversed by a wave each time step. This parameterisation is illustrated neatly in equation (2.2).

For a given stencil, then, we can show the dispersion error for all combinations of spatial and temporal sampling in a single plot. Example group velocity error surfaces are shown in Figure 2.5, with the $k\Delta x$ scale being expressed more intuitively as the number of grid points per wavelength $2\pi/(k\Delta x)$. Waveforms simulated using these stencils are shown in Figure 2.4: the two columns correspond to crossing factors of 0.24 and 0.48, with a central wavelength of 10 grid points per wavelength, and the dispersive behaviour qualitatively matches the error surfaces shown.

The grey sections in the top-right of each surface of Figure 2.5 are noteworthy. These correspond to wavenumber/crossing-factor pairs for which the estimated spatial derivative on the right hand side of the wave equation is too high to correspond to any (approximated) temporal derivative. In other words, no plane-wave solution to the wave equation exists in this region. Unfortunately, as discussed in Graves (1996), this phenomenon leads to instability when using explicit time stepping to solve the wave equation, and as I will now demonstrate this often turns out to be a stronger limit on the spacing of the temporal grid than accuracy concerns.

To see this, consider a fairly general recurrence relation for explicit time stepping, where the second time derivative is equal to some kind of linear spatial operator $\mathbf{D}$ acting on the

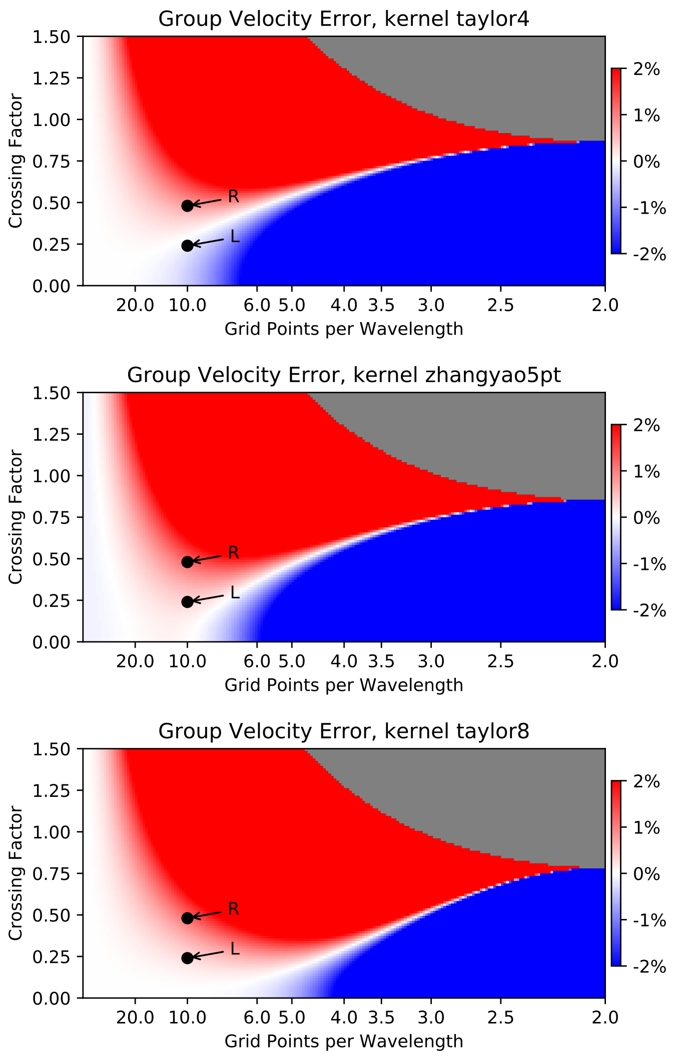


Figure 2.5: Group velocity error surfaces for various spatial stencils, when combined with a second-order Taylor stencil for the time derivatives. The grey regions correspond to wavenumbers for which there is no plane wave solution in the discretised system, with the lowermost edge of the grey region indicating the stability limit on the time step size (in one dimension). The points marked L and R on the images indicate the parameter combinations for the corresponding plots on the left and right of Figure 2.4, respectively.

wavefield at the current time step:

$$\begin{aligned} D_{2t}\mathbf{u}(t) &= \mathbf{D}\mathbf{u}(t) \qquad (2.4) \\ \mathbf{u}(t+\Delta t) &= 2\mathbf{u}(t) - \mathbf{u}(t-\Delta t) + \Delta t^2 \mathbf{D}\mathbf{u}(t) \end{aligned}$$

Rewriting this time step in matrix form:

$$\begin{bmatrix} \mathbf{u}(t+\Delta t) \\ \mathbf{u}(t) \end{bmatrix} = \begin{bmatrix} 2\mathbf{I} + \Delta t^2 \mathbf{D} & -\mathbf{I} \\ \mathbf{I} & \end{bmatrix} \begin{bmatrix} \mathbf{u}(t) \\ \mathbf{u}(t-\Delta t) \end{bmatrix}$$

This would then mean, for n timesteps:

$$\begin{bmatrix} \mathbf{u}(t+n\Delta t) \\ \mathbf{u}(t+n\Delta t-\Delta t) \end{bmatrix} = \begin{bmatrix} 2\mathbf{I} + \Delta t^2 \mathbf{D} & -\mathbf{I} \\ \mathbf{I} & \end{bmatrix}^n \begin{bmatrix} \mathbf{u}(t) \\ \mathbf{u}(t-\Delta t) \end{bmatrix}$$

Since n can be large (many thousands), this recurrence relation is only stable if all of the eigenvalues of this 2x2 block matrix have magnitude less than or equal to unity. Otherwise, some infinitessimal component of the starting wavefield which has a component in the direction of the corresponding eigenvector will grow uncontrollably as the simulation progresses. I therefore seek to find bounds on the eigenvalues of this time stepping operator in terms of the matrix $\mathbf{D}$ and the timestep Δt.

Let λ be an eigenvalue of the time-stepping matrix:

$$\begin{bmatrix} 2\mathbf{I} + \Delta t^2 \mathbf{D} & -\mathbf{I} \\ \mathbf{I} & \end{bmatrix} \begin{bmatrix} \lambda\mathbf{v} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \lambda^2\mathbf{v} \\ \lambda\mathbf{v} \end{bmatrix}$$

The top row of the above equation is the so-called quadratic eigenvalue problem:

$$\lambda^2\mathbf{v} - (2 + \Delta t^2\mathbf{D})\lambda\mathbf{v} + \mathbf{v} = \mathbf{0}$$

Now suppose $\mathbf{v}$ is an eigenvector of the operator $\mathbf{D}$, with eigenvalue γ, i.e. $\mathbf{D}\mathbf{v} = \gamma\mathbf{v}$. I now show that $\mathbf{v}$ is also a solution to the above problem, mapping into a pair of eigenvectors

$\lambda_\pm$:

$$
\begin{aligned}
\lambda^2\mathbf{v} - (2 + \Delta t^2\gamma)\lambda\mathbf{v} + \mathbf{v} = \mathbf{0} \\
\lambda^2 - (2 + \Delta t^2\gamma)\lambda + 1 = 0 \\
\lambda_\pm = \frac{1}{2}(2 + \Delta t^2\gamma \pm \sqrt{(2 + \Delta t^2\gamma)^2 - 4})
\end{aligned}
$$

If the term under the square root is negative, then the magnitude of these two complex numbers can be shown to be equal to 1:

$$|\lambda_\pm|^2 = \frac{1}{4}((2 + \Delta t^2\gamma)^2 + (4 - (2 + \Delta t^2\gamma)^2)) = 1$$

This shows that the system conserves energy, as desired: timestepping any eigenvector of $\mathbf{D}$ does not change its amplitude. (Note that this changes when absorbing boundary conditions are added, as one would expect). The phase of these complex numbers corresponds to $\omega\Delta t$ in the corresponding solutions.

If, however, the term under the square root is positive, i.e. $\Delta t^2\gamma \notin [-4, 0]$, then both of the eigenvalues are real and one has a value with magnitude greater than 1:

$$\lambda_- = \frac{1}{2}\left(2 + \Delta t^2\gamma - \sqrt{(2 + \Delta t^2\gamma)^2 - 4}\right) < -1$$

Since the presence of any eigenvalues with $|\lambda| > 1$ will lead to numerical instability, this gives a hard limit on the maximum timestep $\Delta t^2 < -4/\gamma_{\max}$, where $\gamma_{\max}$ is the largest (negative) eigenvalue of $\mathbf{D}$. The largest eigenvalues of $\mathbf{D}$ depend on the particular medium parameters and spatial finite difference operators used. However, since the eigenvalues tend to scale proportionally with $c^2/\Delta x^2$, the physical interpretation of this stability criterion is generally as a limit on the proportion of a grid cell which a wavefront is allowed to cross in a single timestep. This is known as the Courant–Friedrichs–Lewy (CFL) condition (Courant et al., 1928). A table of typical cases for an acoustic, isotropic, constant density medium is shown in Table 2.3. The limits in the one dimensional cases correspond to the lower boundary of the grey region in the corresponding plot in Figure 2.5, and are stricter in

Spatial Stencil	Dimensions	$\gamma_{\max}$	Max Crossing Factor
4th Order Taylor	1	$5.333c^2/\Delta x^2$	86 %
	2	$10.666c^2/\Delta x^2$	61 %
	3	$16c^2/\Delta x^2$	50 %
Zhang and Yao 5pt	1	$5.486c^2/\Delta x^2$	85 %
	2	$10.971c^2/\Delta x^2$	60 %
	3	$16.456c^2/\Delta x^2$	49 %
8th Order Taylor	1	$6.501c^2/\Delta x^2$	78 %
	2	$13.003c^2/\Delta x^2$	55 %
	3	$19.504c^2/\Delta x^2$	45 %

Table 2.3: Eigenvalues of the three second-derivative finite difference operators exhibited so far, and the associated maximum crossing factors when using them in tandem with a 2nd-order time-stepping scheme.

higher dimensions because larger wavenumbers can be represented in the off-axis directions.

2.5 Improving accuracy with higher-order time schemes

Because of the stability constraints described in the previous section, it is of limited value to talk about the accuracy order of the spatial discretisation by itself. The asymptotic behaviour of the errors as the spatial step size $\Delta x \to 0$ is undefined, since if $\Delta x \to 0$ while Δt is held constant the stability condition is violated (note that the reverse is not true).

The time accuracy can be improved using the method described in Kim and Lim (2007). Starting with the Taylor expansion of the wavefield at a given time t, using subscripts to indicate derivatives, and defining the second-order second-derivative finite difference operator in time D_{2t} as before:

$$\begin{aligned}\mathbf{u}(t \pm \Delta t) &= \mathbf{u}(t) \pm \Delta t \mathbf{u}_t(t) + \frac{\Delta t^2}{2}\mathbf{u}_{tt} \\ &\pm \frac{\Delta t^3}{6}\mathbf{u}_{ttt} + \frac{\Delta t^4}{24}\mathbf{u}_{tttt} + ... \\ D_{2t}\mathbf{u}(t) &= (\Delta t)^{-2}(\mathbf{u}(t+\Delta t) - 2\mathbf{u}(t) + \mathbf{u}(t-\Delta t)) \\ &= \mathbf{u}_{tt} + \frac{\Delta t^2}{12}\mathbf{u}_{tttt} + O(\Delta t^4)\end{aligned}$$

Then, starting with the original wave equation and differentiating:

$$\begin{aligned}
\mathbf{u}_{tt} &= \mathbf{D}\mathbf{u} + \mathbf{s} \\
\mathbf{u}_{tttt} &= \mathbf{D}\mathbf{u}_{tt} + \mathbf{s}_{tt} \\
&= \mathbf{D}(\mathbf{D}\mathbf{u} + \mathbf{s}) + \mathbf{s}_{tt} \\
D_{2t}\mathbf{u} &= \mathbf{D}\mathbf{u} + \mathbf{s} + \frac{\Delta t^2}{12}(\mathbf{D}(\mathbf{D}\mathbf{u} + \mathbf{s}) + \mathbf{s}_{tt}) + O(\Delta t^4) \\
&= (\mathbf{I} + \frac{\Delta t^2}{12}\mathbf{D})(\mathbf{D}\mathbf{u} + \mathbf{s}) + \mathbf{s}_{tt} + O(\Delta t^4) \qquad (2.5)
\end{aligned}$$

Thus, at the cost of one extra timestep-sized temporary buffer (to hold $\mathbf{D}\mathbf{u} + \mathbf{s}$) and an extra matrix-vector product with $\mathbf{D}$, the time accuracy of the algorithm has been substantially improved, well beyond the benefit gained by simply halving the timestep (which would incur approximately the same computational cost).

The cost is even less than it seems, however, after taking into account the improvement in the stable range of timesteps that this new scheme permits. For the purposes of stability analysis, where the source term may be ignored, equation (2.5) is the same as the original discretised wave equation (2.4), with the expression $(\mathbf{I} + \frac{\Delta t^2}{12}\mathbf{D})\mathbf{D}$ taking the place of $\mathbf{D}$ on the right hand side. Again letting γ be an eigenvalue of $\mathbf{D}$, equation (2.5) shows that $(1 + \gamma\Delta t^2/12)\gamma$ is an eigenvalue of this new spatial operator. Following through the same mathematics as the last section, the stability criterion becomes:

$$(1 + \gamma\Delta t^2/12)\gamma\Delta t^2 \in [-4, 0]$$

Making the substitution $b = \gamma\Delta t^2$, the left hand side of the above condition is equal to $b + b^2/12$, an upwards-facing quadratic with a minimum at $b = -6$. The value at this minimum is -3, so half of the required inequality is automatically satisfied for any value of b. This allows the stability criterion to be rewritten as:

$$b + \frac{b^2}{12} < 0$$

from which it can very quickly be seen that the requirement is:

$$\gamma \Delta t^2 \in [-12, 0]$$

This is a $\sqrt{3}$ increase in the range of stable timesteps over the standard second-order scheme - almost completely compensating for the doubling of the computational cost per timestep. Figure 2.6 shows the dramatic reduction in dispersion that can be obtained from using this technique.

2.6 Efficient Computer Implementations

When applied together, the techniques described in the preceding sections allow the number of spacetime points in the computation to be dramatically reduced, for a modest increase in the number of arithmetic operations required per point. There are also efficiency gains to be had in the mapping of these arithmetic operations onto computer hardware.

While FWI is typically run on a cluster of computers, the vast majority of the computational work is done separately for each shot. Notwithstanding memory constraints, typically this means that the most efficient way to do the work in parallel is to treat each shot separately within its own compute node. I therefore focus on intra-compute-node issues.

Figure 2.7 shows a diagram of one of the most common configurations for HPC cluster nodes: a single system with two processors in separate sockets, each having multiple cores and some associated memory. The two sockets are connected by a high-speed interconnect. Using the specific example of the nodes in the Imperial HPC cluster cx1, this diagram is annotated with the expected total performance of each CPU socket, measured in single-precision floating point operations per second, and the expected number of gigabytes per second that may be transferred to/from each bank of memory, and across the high-speed interconnect between the CPUs.

Most important to note in this diagram is that the number of the floating point operations the processors are capable of performing vastly exceeds the available memory bandwidth. Each single-precision floating point number is 4 bytes, so for a simple operation such as

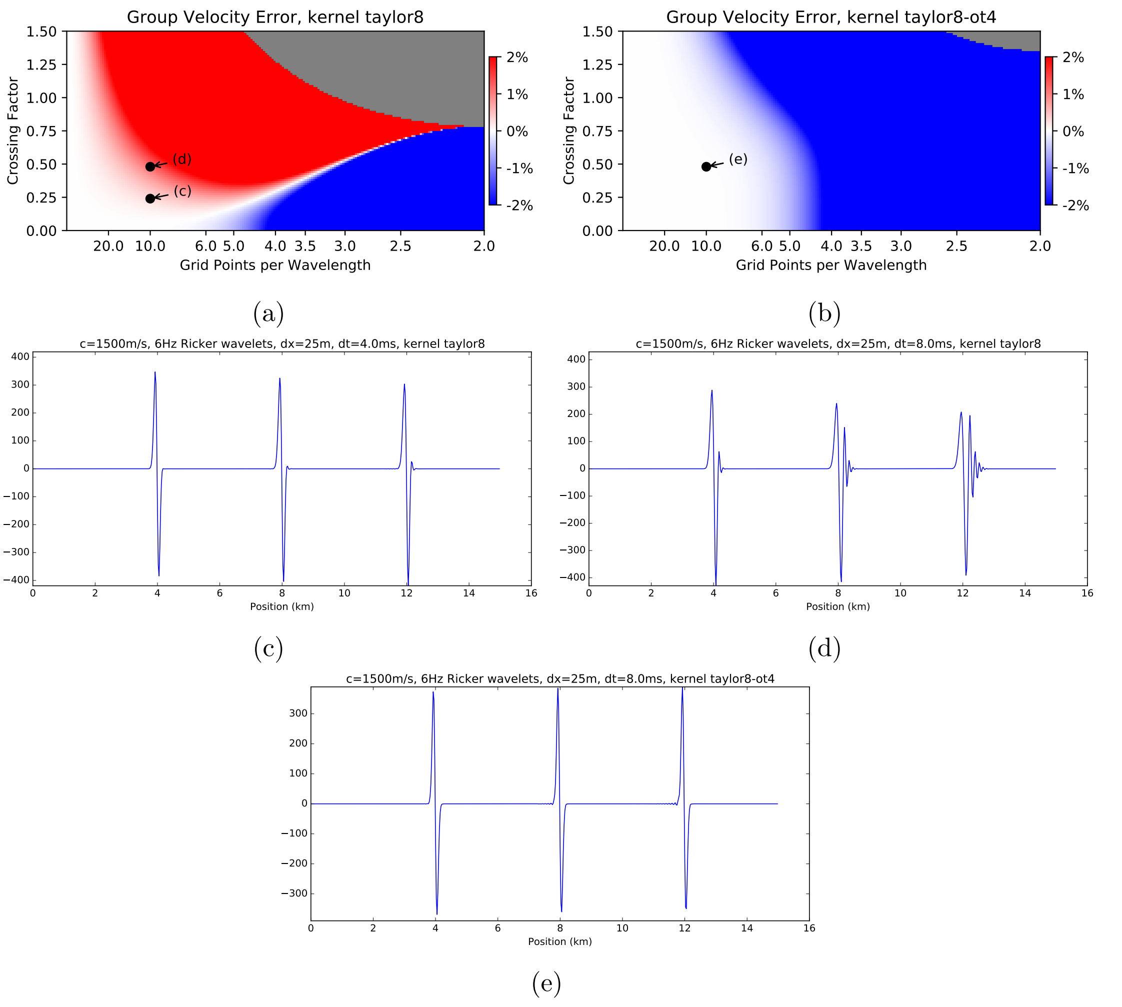


Figure 2.6: Reduction in dispersion from using the 4th-order time stepping scheme with the 8th-order Taylor stencil. By expending twice the computational cost per time step, accuracy is vastly improved, and the range of stable time step sizes is extended by 1.7x, almost completely compensating for the extra computation. (d) and (e) have the same time step, while (c) and (e) involve the same amount of computational effort. The points corresponding to the waveforms (c), (d) and (e) are marked on the group velocity plots (a) and (b).

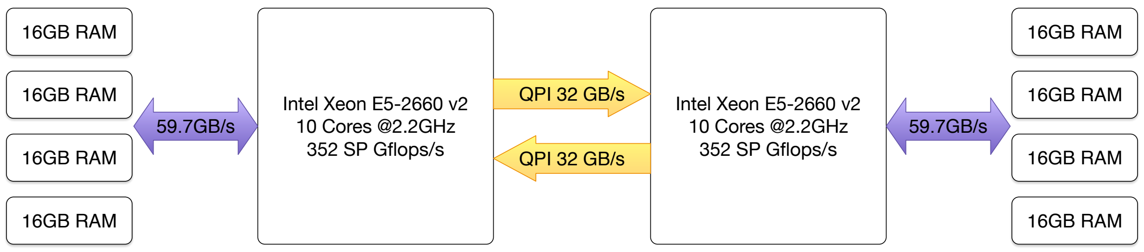


Figure 2.7: Block diagram of a typical HPC cluster node used for running FWI

c=a+b, the requirement to fetch the inputs from and store the output to memory restricts the CPU to less than 1/70th of its theoretical peak performance. This is particularly relevant to the wave-equation problem: to make effective use of the vast computing power of modern multi-core processors, a significant number of floating point operations must be performed for each operand loaded from main memory. The key to being able to do this lies in the processor's *cache* - CPUs have a complex, multi-level system of caching to retain the most recently-used elements of main memory in a much-faster block of storage on the CPU itself. This is illustrated in Figure 2.8.

To see why the cache is important, consider the problem of evaluating $\nabla^2\mathbf{u}$ for a small, simple 3D problem, using the 2nd-order Taylor spatial derivative operator. This is illustrated by the C code snippet in Listing 2.1.

Listing 2.1: Illustrative C code to evaluate the laplacian of an array with a 2nd-order stencil

```c
void laplacian() {
    float u[200][200][200];
    float delu[200][200][200];

    // obviously we expect u to be populated with data already,
    // such as the current wavefield
    // doing the edge cells is unimportant for this example, also
    int i, j, k;
    for (i = 1; i < 199; i++) {
        for (j = 1; j < 199; j++) {
            for (k = 1; k < 199; k++) {
                delu[i][j][k] =
                    u[i][j][k-1] + u[i][j][k+1] +
                    u[i][j-1][k] + u[i][j+1][k] +
                    u[i-1][j][k] + u[i+1][j][k] -
                    6*u[i][j][k];
            }
        }
    }
}
```

As the above nested loop proceeds through the array, the array element `u[i+1][j][k]` is consistently one that has not been accessed before, and thus causes a *cache miss* to load the value from main memory. It can readily be observed that for all the other values to be still in the cache from their previous usage, the CPU must have a cache large enough to hold 2 complete planes of the volume - i.e. 2x200x200 floats, or 320kB, in this example. In a multi-threaded environment, where the outermost of the above loops was parallelised with e.g. OpenMP, this amount would be required per thread of execution to avoid incurring more than one cache miss per array entry computed. [1]

Most current processors certainly have this much cache - those depicted in Figure 2.7 have 32kB of L1D, 256kB of L2 and 2560kB of L3 cache per core. However, larger models and more accurate finite-difference operators cause this to be exhausted very quickly. An 8th-order Taylor stencil would require 4x more cache, and the 21-point stencils described in the previous section 10x more, than this example. The move to longer stencils, therefore, must also be accompanied by a more sophisticated method for making optimal use of the available cache.

One simple and very well established (e.g. Wolfe (1989)) way of improving the cache-friendliness of stencil algorithms like these is known as *cache blocking*. Listing 2.2 shows a modification to the previous code to illustrate this technique.

Listing 2.2: Illustrative C code to evaluate the laplacian of an array with a 2nd order stencil, using loop blocking

```c
void laplacian() {
    float u[200][200][200];
    float delu[200][200][200];

    // obviously we expect u to be populated with data already,
    // such as the current wavefield
    // doing the edge cells is unimportant for this example, also
    int bj;
    int tilesize_j = 20;
    int i, j, k;
    for (bj = 1; bj < 199; bj += tilesize_j) {
        for (i = 1; i < 199; i++) {
            for (j = bj; j < bj+tilesize_j && j < 199; j++) {
                for (k = 1; k < 199; k++) {
                    delu[i][j][k] =
                        u[i][j][k-1] + u[i][j][k+1] +
                        u[i][j-1][k] + u[i][j+1][k] +
                        u[i-1][j][k] + u[i+1][j][k] -
                        6*u[i][j][k];
                }
            }
        }
    }
}
```

[1]The term 'cache miss' is usually taken to mean a single load of a contiguous, usually 64 bytes, block from main memory into the cache, rather than a single float. The point here is to determine how many times each entry in the input array must be loaded from main memory during the execution of the nested loops, which is unaffected by this semantic detail.

It is fairly simple to verify that this performs the same operation as the previous code snippet. However, it traverses the array in a different order, reusing values loaded from memory sooner, and thus requires a smaller amount of cache (2 x 200 x `tilesize_j`, for moderately large tile sizes) to achieve good performance. The value of `tilesize_j` needs to be chosen appropriately for the overall problem dimensions and available cache sizes. Other approaches to this problem include cache-oblivious algorithms (e.g. Frigo and Strumpen (2005)), which are more sophisticated but do not require tuning for different cache sizes; polyhedral optimisers (e.g. Bondhugula et al. (2008)), which use more sophisticated dependence analysis to find cache-optimal order of operations; and vector folding (Yount, 2015), which rearranges the data elements into more geometrically-compact blocks.

2.7 Integrating an optimised commercial propagator into FULLWAVE3D

The new FULLWAVE3D high-order kernel is a finite difference, acoustic anisotropic, time-domain wave-equation solver that was shared with the FULLWAVE3D group by one of the commercial sponsors, who requested not to be identified. It provides an optimised implementation of many of the features described in this chapter: a highly-accurate non-Taylor stencil, 4th-order time accuracy, and cache tiling. Large parts of the code are manually optimised to run efficiently on modern 64-bit x86 processors, and make extensive use of vector instructions, which are required to obtain close to the peak performance of the processor. Separate implementations of VTI and TTI-anisotropic solvers were provided, with subtly different optimisations.

While it is difficult to do it justice in this thesis, this code took considerable effort to integrate into FULLWAVE3D. There were simple software challenges such as the interopability between FORTRAN and C, the transposed ordering of the model array indices (the new kernel has depth as the most slowly-varying value in the array ordering, while FULLWAVE has it as the fastest), and the fact that FULLWAVE was hard-coded in several places to expect a 2-cell padding around the domain - 2 cells being sufficient to ensure that a 5-point stencil does not access uninitialised parts of memory. The rest of this section will describe some of the more interesting parts of this work: the creation of a novel self-tuning algorithm to optimise the performance of the propagator as it runs, and the vastly improved dispersion characteristics of the new implementation.

2.7.1 Self-tuning algorithm

As I mentioned in the previous section, cache blocking can be an effective way of improving the performance of stencil kernels. By breaking the domain into tiles, and iterating over a whole tile before moving onto the next, the working set size of an individual thread is less than if it were iterating over the entire domain in the conventional order. Furthermore, these tiles are a natural unit for dividing the work between multiple threads of execution - essential for making use of modern multi-core processors on problems with large grids. As such, the interface to the new kernel - a function call to perform a time step - allowed the tile size to be selected. The performance can vary strongly depending on the tile size used, however selection of the best tile size depends on the dimensions of the domain, the available cache size in the processor, and how many threads of execution are sharing that cache.

Given that FULLWAVE is run on a large variety of hardware, and sometimes even across heterogeneous nodes within a single execution, I decided that automatic tuning at runtime would be an effective way of getting the extra performance offered by these tunable parameters. This automatic tuning is based on the "simulated annealing" algorithm (Kirkpatrick et al., 1983) - an global optimisation algorithm suitable for nonconvex functions such as this. At the start of the run, a large population of possible tile sizes are generated, and performance statistics (total compute time, and total number of grid cells evaluated with that configuration) initialised for each one. Additionally an annealing "temperature" is initialised. This temperature indicates the willingness of the algorithm to move to a state (in this case, a tile size) for which the estimated performance is slower than the current state. Then, at each timestep of the simulation, the self-tuner follows the following steps:

1. Choose a candidate tile size from a probability distribution, biased towards sizes similar the current tile size. This is done using the Metropolis-Hastings algorithm

(Hastings, 1970).

2. If there is no performance data for the candidate tile size, set the current tile size to the candidate tile size, and return.

3. Otherwise, if the average performance (in seconds per million cells) $t_{\text{candidate}}$ of the candidate tile size is better than the average performance t_{current} of the current tile size, then set the current tile size to the candidate tile size, and return.

4. Otherwise, evaluate $p = e^{10^6(t_{\text{current}} - t_{\text{candidate}})/T}$, and choose a uniformly distributed random value $r \in [0, 1]$.

5. Reduce the temperature T according to a somewhat arbitrary schedule.

6. If $r < p$, then set the current tile size to the candidate tile size, and return.

7. Otherwise, leave the tile size as-is, and return.

Later, this optimising algorithm was also augmented to include dynamic selection of the number of execution threads (up to the number of logical cores available), and to choose between two alternative implementations of the kernel (one using SSE, the other using AVX - two generations of vector instruction sets). This allows the new kernel to make better use of processors with multiple logical threads per core, and also to dynamically select the older SSE-optimised kernel in the cases where it was faster, as was observed on the Imperial College HPC cluster.

Figure 2.9 shows this algorithm in operation, with the propagation performance increasing after a few shots once the annealing procedure finds the optimal runtime configuration. For reference, the performance of the existing FULLWAVE3D propagator is overlaid.

2.7.2 Dispersion characteristics of the high-order kernel

Figure 2.9 shows that, in terms of grid points evaluated per second, the new kernel is slightly faster than the existing FULLWAVE3D kernel for VTI, and slightly slower for TTI. However, as discussed earlier in the chapter, the real boost in performance is due to the improved dispersion characteristics of the longer, optimised stencil. Table 2.4 shows the maximum grid spacings required to meet the dispersion criterion outlined earlier in the chapter when using the new kernel. For most problems, the spacings permitted by the new kernel are at least double that of the existing one. Combined with the larger permitted range of stable timesteps (a maximum crossing factor of 66% vs 50% for the existing propagator), this means that the new kernel requires approximately 20x fewer space-time grid points than the existing one for the same quality of result. Figures 2.10 and 2.11 show a side-by-side comparison of the dispersion surfaces and results of propagation using the stencils from the existing and new kernels.

f, T	4 sec	5 sec	6 sec	7 sec	8 sec	9 sec	10 sec
6 Hz	92.3m	92.3m	91.7m	79.8m	77.9m	77.3m	76.6m
12 Hz	38.9m	38.3m	37.7m	37.4m	29.5m	28.9m	28.3m
18 Hz	25.1m	24.9m	19.3m	18.8m	18.4m	12.6m	11.9m
24 Hz	14.8m	14.1m	13.8m	9.1m	8.6m	8.5m	8.2m
30 Hz	11.3m	11.1m	7.2m	6.8m	6.5m	6.4m	6.3m
36 Hz	9.2m	6.0m	5.7m	5.4m	5.2m	5.1m	5.1m
42 Hz	5.2m	4.8m	4.7m	4.5m	4.4m	4.3m	2.2m
48 Hz	4.3m	4.1m	3.9m	3.8m	3.8m	1.9m	1.7m
54 Hz	3.8m	3.6m	3.4m	3.4m	1.7m	1.5m	1.4m
60 Hz	3.3m	3.1m	3.1m	1.5m	1.4m	1.3m	1.2m

Table 2.4: A table of maximum grid spacings for the new optimised stencil incorporated into FULLWAVE3D, for various frequencies and total propagation times, using the criterion of equation (2.3). The medium is assumed to have minimum velocity 1500m/s. Note that doubling the frequency typically requires more than double the grid density, since a fixed velocity error generates more noticeable dispersion at higher frequencies. Compare this with Table 2.2 - in most cases the maximum spacing with the new kernel is more than double that of the existing implementation.

2.8 Conclusion

In this chapter, I have described several key techniques for improving the performance of time-domain wave-equation solvers - one of the key determinants in the performance of Full-Waveform Inversion. I have reviewed and quantified the numerical dispersion caused by spatial and temporal discretisations and finite difference approximations, and proposed a criterion for acceptable levels of dispersion in the modelling. Finally, I described some of the challenges involved with integrating an externally-provided wave-propagation kernel into the group's flagship FWI software, FULLWAVE3D.

FULLWAVE3D is a comprehensive package with a proven track record in inverting field data in both academic and commercial settings. However, the experience of integrating the new kernel into it led me to an uncomfortable realisation: FULLWAVE3D is not (or at least was not, at the time) ideally suited for the kind of research I aimed to conduct. Prototyping of algorithmic ideas is difficult, not only because of my relative unfamiliarity with FORTRAN but because seemingly small changes require modifications to unexpectedly many parts of the code. Additionally, the input and output formats require conversion before they can be processed or viewed, so even simple tests can take considerable time to set up for the inexperienced user. Finally, some of the algorithms I wished to try, and experiments I wished to do, require the gradients to be computed more accurately than many parts of FULLWAVE currently do. It is with these reasons in mind that I subsequently set out to write a simpler, lighter-weight inversion code that I believe is more suitable for algorithmic research. This is described in the following chapter.

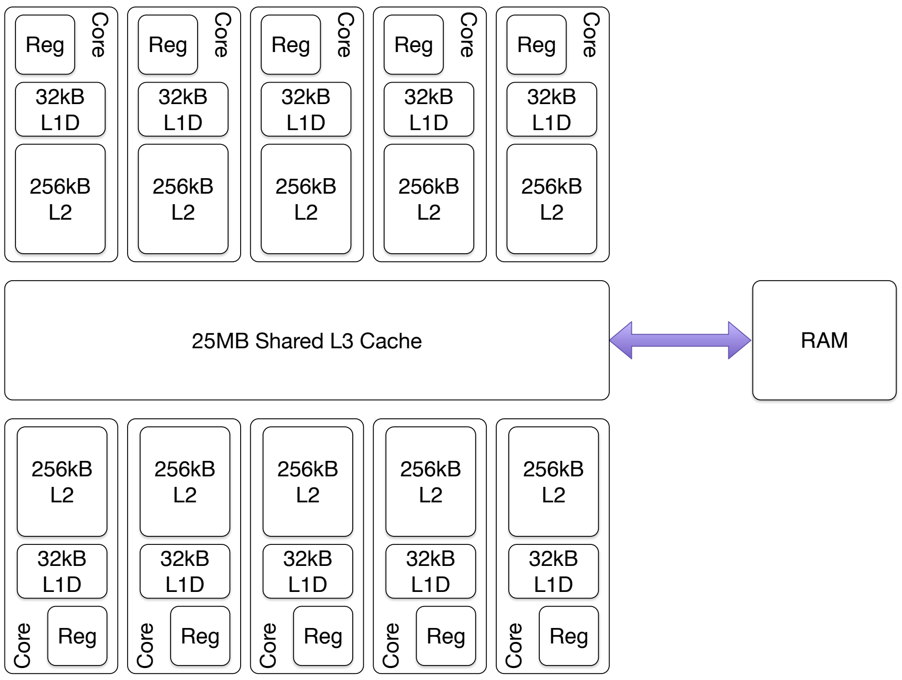


Figure 2.8: Data cache hierarchy within a single CPU from Figure 2.7, showing registers, two levels of cache dedicated to each core, and a third, much-larger, cache shared between all cores. The smaller, lower-level caches are much faster to access than the larger, higher-level ones.

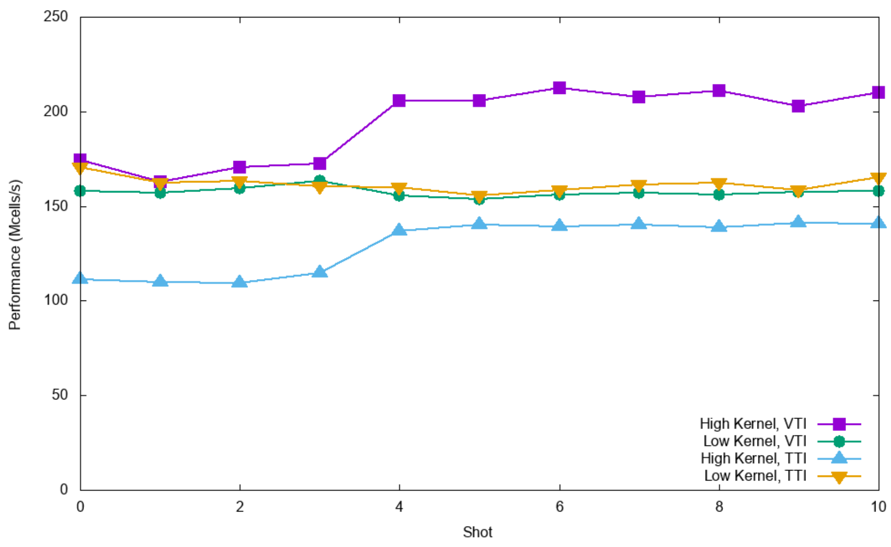


Figure 2.9: Performance of the existing FULLWAVE3D propagator and the new high-order propagator, in millions of grid cells per second, plotted for the first few shots of an inversion run. The increase in the performance of the high kernel after 3-4 shots demonstrates the boost obtained from the auto-tuning algorithm I added to the new propagator. The domain size here was 371 x 311 x 103 cells, running on a compute node with the specification of Figure 2.7. In terms of grid points evaluated per second, the performance of the two propagators is comparable, however the new propagator typically requires about 20x fewer grid points to obtain the same quality of result.

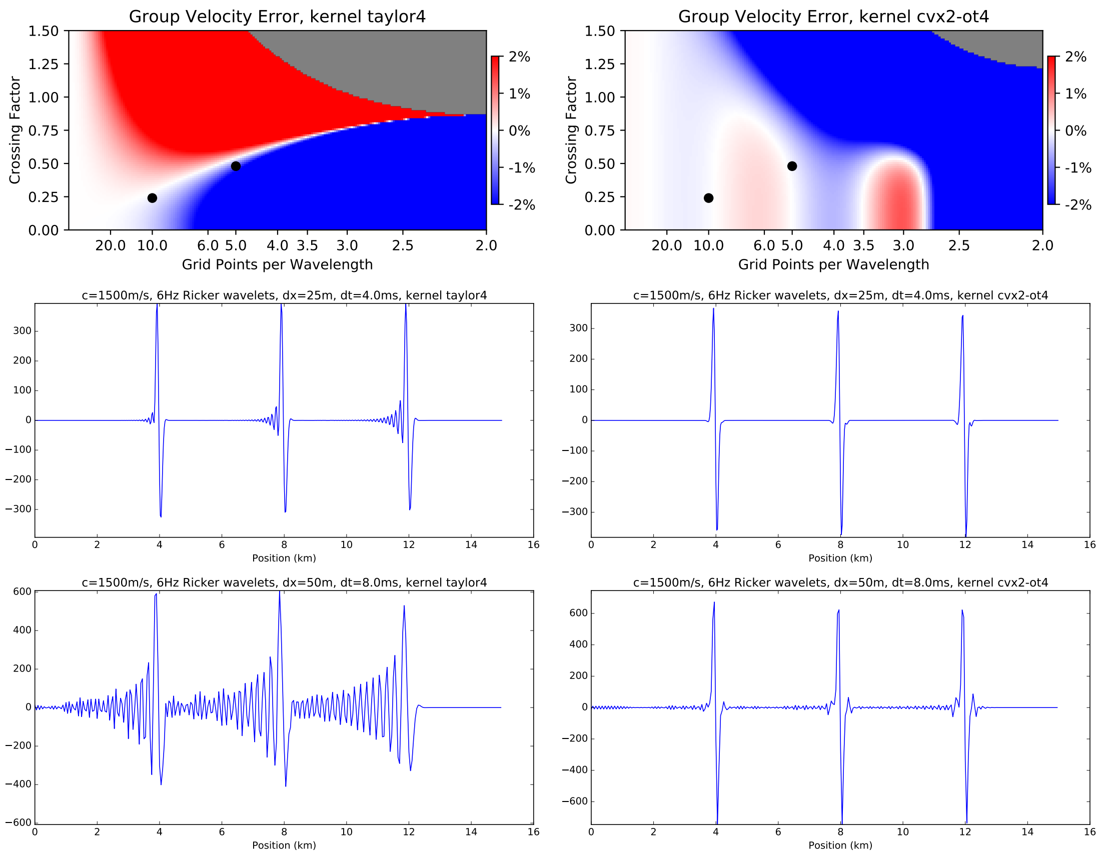
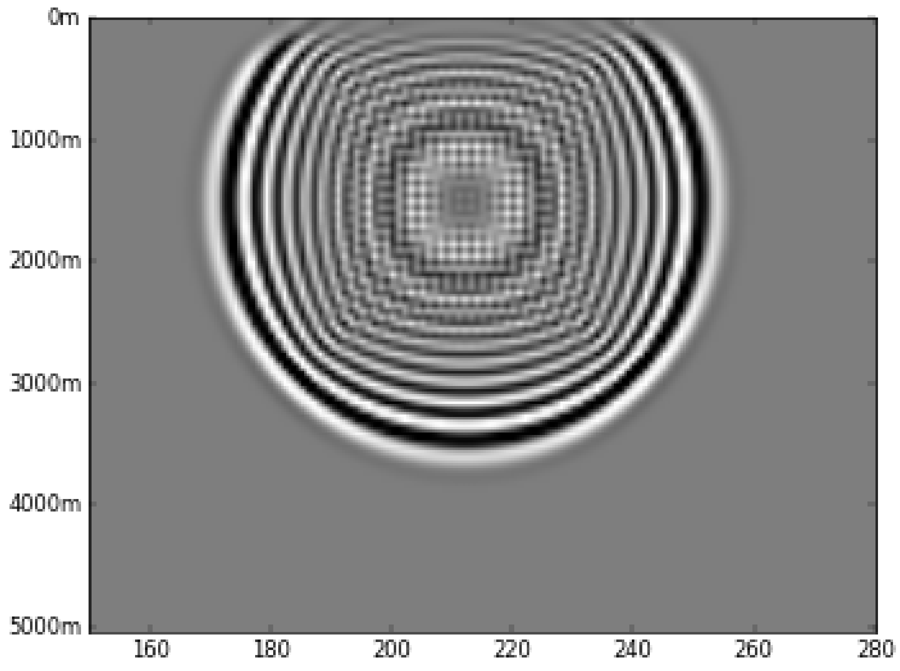


Figure 2.10: A comparison of dispersion surfaces and the resulting effects on propagation between the old (left) and new (right) kernels. Top: group velocity error. Centre: propagation of 6Hz wavelets with $\Delta x = 25\text{m}, \Delta t = 4.0\text{ms}$. Bottom: propagation of 6Hz wavelets with $\Delta x = 50\text{m}, \Delta t = 8.0\text{ms}$.



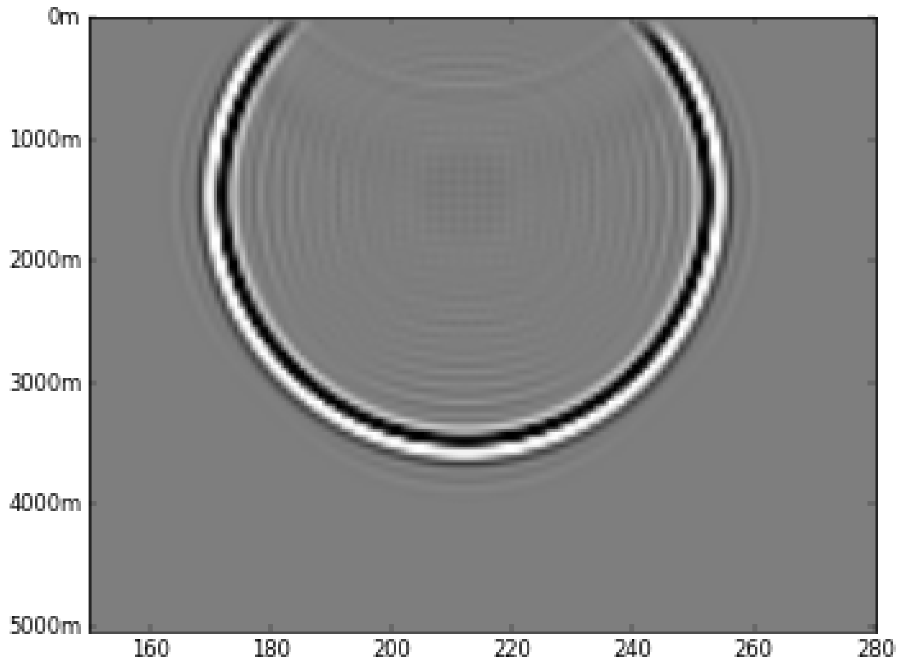


Figure 2.11: Wavefield snapshots with the existing (top) and new (bottom) kernel. Source is a wavelet with peak frequency of 10Hz, propagating in a medium with $c = 1500\text{ms}^{-1}$, with $\Delta x = 36\text{m}, \Delta t = 1.8\text{ms}$

"All things are difficult before they are easy."

— Dr Thomas Fuller

3 A research-friendly FWI implementation

3.1 Introduction

The research group's flagship software, FULLWAVE3D, is a full-featured package with a proven ability to compete at the highest level in the field of commercial FWI implementations. However, as mentioned in the conclusion of the previous chapter, it suffers from a number of downsides when used for experimentation and development of FWI at an algorithmic level.

The first of these is that the gradient computed by FULLWAVE3D, while pointing in the direction of desirable updates, is not the true gradient of the objective function. It ignores a number of aspects such as normalisation, scale factors on the time integration of the cross-correlation, and changes to the density model. More sophisticated techniques such as conjugate gradient and L-BFGS (Nocedal, 1980) tend to rely on being able to obtain the true gradient - especially as a quality control on the line search.

Furthermore, due to its large feature set, the code is relatively complex and difficult to change, even more so because it is already manually optimised for good performance.

Finally, the FORTRAN implementation makes input and output in arbitrary data formats more difficult, due to a lack of high-level abstractions. This means that visualising the output of inversions involves using (and in some cases, developing) a suite of companion utilities.

After experiencing these difficulties first-hand while integrating a new propagator, I decided that future research would be more efficient with a developer-friendly FWI implementation, and so I set out to create one from scratch.

This chapter briefly describes the resulting implementation, which I have named PyFWI. It contains no original "research", but provides the software base upon which the remainder of the work in this thesis is built. It is also my hope that this description of the features will encourage readers to try the software for themselves.

3.2 Goals and design decisions

There were a few high-level requirements that I had in mind when creating this software:

1. For maintainability, it must be implemented almost-entirely in a high level language. There are relatively few performance-critical parts of any FWI implementation: these can be implemented in C or FORTRAN if necessary, leaving the remainder accessible to non-expert developers.

2. It needed to be distributed-memory parallel. That is, to be able to run across several nodes in a standard cluster environment, such as that provided by the College.

3. The configuration should make for simple, efficient A-B testing of parameters and algorithms. In other words, a single run of the software should be able to perform multiple inversions simultaneously.

4. Results should be able to be visualised instantly, as the software is running. This allows for early termination of inversions which are performing poorly and, in concert

with the previous point, allows for rapid comparisons to be made when trying different algorithmic or parameter tweaks.

The choice of programming language is an important decision in any software project, and especially in this case there are many competing issues to weigh. Since a high level language was desired, I restricted my decision to MATLAB, Julia and Python. In the end, despite its imperfections, I settled on Python for several reasons:

1. even though the performance of native Python is not necessarily good, it is fairly simple to write performance-critical modules in C that interface relatively seamlessly with a higher-level program written in Python. In my brief evaluation, I concluded that MATLAB places several barriers in the way in this respect. In particular, an insistence on double-precision floating point, and not allowing the lower-level modules to allocate objects that may be returned to MATLAB without copying from one region of memory to another.

2. Julia, while very promising as a future high-level language for large scale computing problems, lacks - or lacked, at the time - the maturity of 3rd-party libraries that is available to Python, including visualisation (matplotlib), travel time computations (scikit-fmm), eigenvalue computations (slepc4py), and importantly in this case, MPI.

3. Python, with its history as a scripting language, allows very rapid development of ancillary, non-performance critical, code such as parsing of command line parameters and parameter files.

4. MATLAB licensing costs for large systems are relatively prohibitive, meaning that most university-class HPC systems are unable to run large MATLAB clusters.

5. Python seems to be a popular teaching language, both at Imperial College and outside, and so future students may be able to make better use of this software.

For distributed-memory parallelism, MPI is still the de-facto standard on almost all large shared computing systems. In Python, this support is provided by a simple, easy-to-use module mpi4py. This allows instances of the software running on separate compute nodes to send and receive data between each other, with optimised primitives for certain key operations such as summation of an array across all compute nodes.

3.3 Implementation

With the above goals in mind, one of the central aims behind the creation of yet another FWI implementation was the ability to rapidly prototype and evaluate new ideas. To this end, each inversion "experiment" within a run can be constructed from a number of pluggable components, able to be selected and configured in the parameter file.

The elements that may be selected by the user include:

1. The model. This module provides not only the initial model parameters, but a mapping between the model grid and the x/y/z coordinates used for source and receiver locations in the data. There are several implementations for the model module:

    - `marmousi` - since the Marmousi model (Versteeg, 1993) is so ubiquitous, it is hard coded as an option, with no additional files or parameters required. A smoothing length may be provided to apply a gaussian smoother to the true Marmousi model.
    - `vtr` - this is the unique data file format used by FULLWAVE3D. Since vtr files do not contain any information about the spacing or x/y orientation, extra parameters are required to supply this information.
    - `su` - seismic unix (Cohen and Stockwell, 2015) provides lots of simple tools for manipulating and visualising models from the command line, and is a convenient format for data exchange. The spacing and mapping between x/y location and position along the line is embedded in the trace headers of the file. Additionally, this is the format used for PyFWI's output - so the second and subsequent stages of multi-stage inversions must use this as the input. For convenience, simple parameters are also offered by this module to add checkerboards to the model.

2. The propagator. Currently, there is only one meaningful implementation of the wave propagator in PyFWI. It is a 2D-only, acoustic, constant-density implementation, using the 5-point optimised stencil from Zhang and Yao (2013) as discussed in the previous chapter. The kernel is coded in C for performance and multi-threading. There are many parameters here, including absorbing boundary thicknesses, how to construct the preconditioner from the source wavefield, decimation of the wavefield for gradient computation, etc.

3. The observed data. This module provides source/receiver locations, wavelet, observed data and weights for each shot. The weights are primarily there to facilitate muting of data that is not expected to be matched by the inversion. Available implementations for this component are:

    - `segy` - read a SEG-Y (SEG Technical Standards Committee, 2002) file with the observed data, with the source and receiver location data embedded in the SEG-Y trace headers, and another with the wavelet. In this module, weights can be generated by providing a saved numpy array with (offset, time) pairs describing a mute. For performance, the SEG-Y file is read in parallel from all nodes involved in the computation.

    - `synthetic` - one of the most convenient features of PyFWI is the ability to generate synthetic data on the fly, and then use it for inversion. This module can instantiate its own model and wave propagator component, and generate synthetic data using either a supplied wavelet or a Ricker wavelet of a nominated frequency, and a geometry specified using numpy array syntax. The synthetic data is generated as needed, and cached for future use.

4. The objective function itself, selected by choosing a module known as the "trace processor". This module is called after forward propagation is performed, and is passed the observed data and weights from the "observed data" module, as well as the predicted data from the "propagator" module. It returns an array of residuals, which can be any size, and an adjoint source to be used in the backpropagation,

to generate the gradient. There are two implementations with different kinds of normalisation: `fit_to_modelled`, which scales each trace of the observed data to obtain a minimum squared residual; and `normalise_to_modelled`, which scales each trace of the observed data to match the RMS amplitude of the corresponding modelled data; as well as an implementation for Adaptive Waveform Inversion (Warner and Guasch, 2014).

5. A projection of the update onto a reduced space. This is most commonly used to mask out the waterbody or other parts of the model that we do not wish to update, but can also be used to e.g. force updates to be one-dimensional, as will be used in Chapter 5.

6. A “direction generator” plugin, which takes preconditioned and unpreconditioned projected gradients, is able to keep history, and returns a step direction. This is currently used to implement various conjugate gradient formulae.

7. An extra matrix preconditioning step, which will also be used in Chapter 5.

3.4 Example

An example runfile for PyFWI, based on one used later in this thesis, is below.

```
# The marmousi model is a hard coded i/o type
startmodeltype marmousi

# It allows gaussian smoothing of any length - but never touches the first 12 cells (the waterbody)
smoothlength 5

# Use the original Marmousi grid spacing
dx 18.75

# This is really the only choice of propagator for now
propagator cprop_v1

# This turns on live display of the wavefields as they are simulated, every 100 timesteps
wfdebug true
wfdebug_decimation 100

# Boundary thicknesses - simple sponge boundaries
bleft 20
btop 0
bright 20
bbottom 20

# Two concurrent experiments
nexperiments 2
```

```
numiterblocks 1

# Both use synthetic data
datasourcetype synthetic

# Normalisation is enabled
trace_processor normalise_to_modelled

# Only use 10% of the shots each iteration - randomly selected
shot_fraction 0.1

# This matches the smoothing in the marmousi module - masking off updates in the top 12 cells
update_projector zmask
zmask_ncells 12

# We set conjugate gradient, using the Polak-Ribiere formula, clipped to positive
gradient_type cg-prp+

# This is a section delimiter - these next settings only apply to the first experiment
[expt0001]
# This name is used in the functional plot
exptname SourceHessian-CG

# This is used to name the output models, gradients, which are saved as .su files
save_prefix norcvrhess_cg

# The first experiment uses the traditional preconditioner, from Shin (2001)
receiverhessian no

[expt0002]
exptname BothHessian-CG
save_prefix rcvrhess_cg

# This experiment uses the preconditioner developed in the next chapter
receiverhessian yes

# Iteration blocks not really supported yet, but this is to give the idea
[iterblock0001]
iterations 100

### And this section is used by the "synthetic" data source module
[synthetic]

# Synthetic data is generated on the unsmoothed marmousi model
truemodeltype marmousi
propagator cprop_v1
smoothlength 0

# We could set different boundary thicknesses here if we wanted to not commit an inversion crime, e.g.
# bbottom 40

# The time sampling and number of time samples comes from the data source, so they are set here
nt 500
dt 0.008

# Since we are generating synthetic data, we need a convenient way to specify the geometry
# The "eval" geometry type allows us to specify a python expression
# It must return a sequence of shots, each with a list of receivers, i.e. (srcloc, [rcvrloc])
# Note the locations are in metres
geometrytype eval
geometry [((srcx,0,15), [(rcvx,0,15) for rcvx in range(100, 8800, 50)]) for srcx in range(200,8800,50)]

# The source wavelet comes from the data source module too
# In this case it also happens to be used to generate the data
# This is separate from the filters which can be applied to the observed and predicted data, on a per-experiment basis
sourcetype ricker
ricker_hz 5
```

"In mathematics, you don't understand things. You just get used to them."
— John von Neumann

4 Improved Diagonal Preconditioning

It is generally accepted that FWI, even when inverting for a single parameter, is an ill-posed problem – meaning that there is an infinite space of models which all cause the misfit function to reach its optimum value (Virieux and Operto, 2009). This is not typically attributed to an infinite number of distinct but equally-optimal local minima; rather, it refers to the observation that from an optimal point, there is a subspace of update directions in which the functional is constant to greater than first order (constant to first order in every direction is a criterion for being an optimal point).

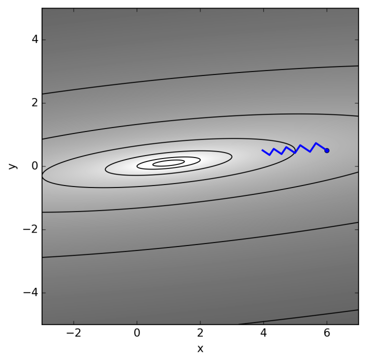
Given that there are directions to which the functional is essentially insensitive, it is therefore unsurprising that among the remaining directions of model update, there is also a wide variation in this sensitivity. Unless special care is taken, this variation in sensitivity can lead to extremely poor performance of optimisation algorithms.

This effect is most easily demonstrated by considering the case of a steepest descent algorithm minimising a quadratic form, as shown in Figure 4.1a. Observe how, due to the different curvatures in the function surface with respect to x and y, that the gradient rarely points in the direction of the optimal point - the components of the gradient in the more-sensitive directions are exaggerated relative to those in the less-sensitive directions. This causes a steepest descent solver, which always moves in the direction of the gradient at each iteration, to perform extremely poorly. Some directions are systematically overcorrected at each iteration, while others are undercorrected.

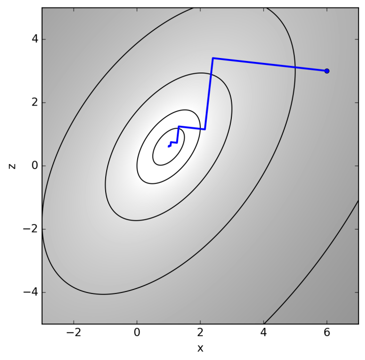
It is important to note that this phenomenon is not inherent to a given optimisation problem. Rather, it strongly depends on the representations chosen for the physical parameters. This can be illustrated by applying a very simple change of variables to the problem: instead of solving for (x,y), I will write z=6y and solve for (x,z). The effect of this change is shown in Figure 4.1b, with the mapping of the iterations onto the (x,y) space shown in Figure 4.1c. It can readily be observed that the modified system converges much more quickly. By encoding some knowledge of the sensitivities into the choice of problem variables, the solution is obtained much more efficiently. In the context of FWI, this effect has been noted by various authors, such as da Silva et al. (2014), who observe improved convergence when using different representations for the model parameters. To first order, an arbitrary change of variables can also be thought of as a mapping from the gradient of the function to a step direction – this perspective also has the advantage that it more naturally allows the mapping to change as iterations progress. This mapping process is known as preconditioning.

This chapter is broken down into five sections. In Section 4.1, I will discuss the connection between preconditioning and the matrix of second derivatives (the Hessian), demonstrate uplift from preconditioning even in problems that are not quadratic in nature like the one above, and explain what is meant by diagonal preconditioning. In Section 4.2, I will derive an expression for the elements of the Hessian matrix for the FWI problem, and show ways in which this is traditionally used to construct an approximate diagonal preconditioner for FWI. In Sections 4.3 and 4.4, I will propose an improvement to this construction, and show how it may be efficiently computed - work I first presented in Burgess and Warner (2015).

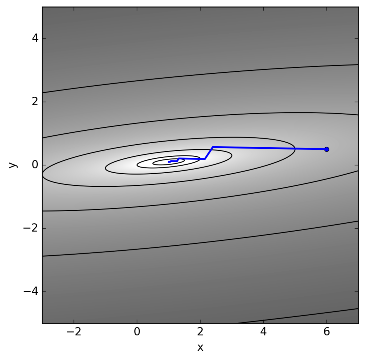
In Section 4.5, I will apply this new technique to two popular FWI benchmark problems, and demonstrate improved convergence using my technique.



(a) A contour plot of the quadratic function in (4.1), and the path taken by the first 9 steps of a steepest descent algorithm. Note that the contours are logarithmically spaced.



(b) The function of (4.1) after a change of variables $z = 6y$, and the path taken by a steepest descent algorithm in the transformed space.



(c) The function of (4.1) in the original (x,y) space, but showing the path from Figure 4.1b. Note the dramatically improved convergence compared to Figure 4.1a.

Figure 4.1: A simple quadratic objective function, from (4.1), being minimised by a steepest descent algorithm. Applying a simple change of variables and computing the steepest descent directions in the transformed space dramatically accelerates the convergence of the descent.

4.1 Preconditioning in linear and nonlinear problems

Figure 4.1 illustrates that a very simple change of variables can dramatically speed up the convergence of a steepest-descent solver. In this section, I will explain why the above change of variables improves the convergence of the solver, show how this technique can be applied to nonquadratic problems, and finally some of the compromises required to scale it up to very large problems such as FWI.

4.1.1 Linear Problems

The objective function illustrated in Figure 4.1a is a quadratic function in two variables:

$$f = (x - 1)^2 + 36(y - 0.1x)^2 \tag{4.1}$$

It is apparent from looking at this figure that the (negative) gradients of the function (perpendicular to the contour lines), even though they by definition point downhill, do not point directly towards the optimal point. This is characteristic of all problems where the objective function is more sensitive to some update directions than others: the contour plot resembles that of a narrow, sloping valley, and gradients have a strong component towards the centreline of the valley, but not along it towards the overall minimum. As a result, a descent algorithm often gets stuck "zig-zagging" down the valley, making relatively slow progress towards the solution.

The transformed problem in Figure 4.1b, on the other hand, is far more symmetric about the optimal point. The gradients, while still not pointing directly to the solution, are a dramatic improvement. For a quadratic function, this relationship between the gradient and the direction to the optimal point is encapsultated by the matrix of second derivatives - also known as the Hessian matrix. This can be seen by rewriting the function in the matrix form:

$$\begin{aligned}
f &= 1 + \begin{pmatrix} -2 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \frac{1}{2} \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} 2.72 & -7.2 \\ -7.2 & 72 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\
&= \frac{1}{2} \begin{pmatrix} x-1 & y-0.1 \end{pmatrix} \begin{pmatrix} 2.72 & -7.2 \\ -7.2 & 72 \end{pmatrix} \begin{pmatrix} x-1 \\ y-0.1 \end{pmatrix} \\
\nabla_{x,y} f &= \begin{pmatrix} 2.72 & -7.2 \\ -7.2 & 72 \end{pmatrix} \begin{pmatrix} x-1 \\ y-0.1 \end{pmatrix}
\end{aligned}$$

where the factor of one half is included to invite the connection to a Taylor series, with the square matrix being the Hessian matrix of the function. Just like other quadratic functions - in either one or more variables - the second derivative is a constant, independent of the values of x and y.

From the above equations, observe that if the Hessian matrix was a (positive) multiple of the identity matrix, then the gradients ∇f would point along the direction of vector from the optimal point $(x-1, y-0.1)$. Therefore, problems where the Hessian matrix more closely resembles a multiple of the identity are likely to be more easily solved by a simple descent-based solver.

The above polynomial can also be written in terms of the modified variables x and z:

$$\begin{aligned}
f &= 1 + \begin{pmatrix} -2 & 0 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} + \frac{1}{2} \begin{pmatrix} x & z \end{pmatrix} \begin{pmatrix} 2.72 & -1.2 \\ -1.2 & 2 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \\
&= \frac{1}{2} \begin{pmatrix} x-1 & z-0.6 \end{pmatrix} \begin{pmatrix} 2.72 & -1.2 \\ -1.2 & 2 \end{pmatrix} \begin{pmatrix} x-1 \\ z-0.6 \end{pmatrix} \\
\nabla_{x,z} f &= \begin{pmatrix} 2.72 & -1.2 \\ -1.2 & 2 \end{pmatrix} \begin{pmatrix} x-1 \\ z-0.6 \end{pmatrix}
\end{aligned}$$

Notice how the Hessian matrix is markedly different in the transformed space, and much closer to a multiple of the identity matrix. This implies that gradient vectors will be more aligned with the direction towards the optimal point, and the descent algorithm will therefore converge more quickly. Note, though, the presence of significant off-diagonal elements of the Hessian - these come from the rotated axis of symmetry of the objective function contours with respect to the coordinate axes. These off-diagonal elements mean that the problem can never be made completely symmetric, that is to say circular contour lines and gradients pointing directly towards the optimal point, with a simple axis-stretching variable change such as the one used in this case.

Given that the Hessian matrix has such a strong effect on the speed of convergence, it is informative to examine exactly what effect a given change of variables will have on it. Consider a general change of variables $f(\mathbf{x}) \to f(\mathbf{u})$. The relationship between the second derivatives of f with respect to the two sets of variables can then be expressed as:

$$\frac{\partial^2 f}{\partial u_i \partial u_j} = \sum_{k,l} \frac{\partial x_k}{\partial u_i} \frac{\partial x_l}{\partial u_j} \frac{\partial^2 f}{\partial x_k \partial x_l}$$

In the case that the change of variables is linear, $\mathbf{u} = \mathbf{M}\mathbf{x}, \mathbf{x} = \mathbf{M}^{-1}\mathbf{u}$, and therefore:

$$\begin{aligned} \frac{\partial^2 f}{\partial u_i \partial u_j} &= \sum_{k,l} (\mathbf{M}^{-1})_{ki} (\mathbf{M}^{-1})_{lj} \frac{\partial^2 f}{\partial x_k \partial x_l} \\ \mathbf{H}_u &= \mathbf{M}^{-T} \mathbf{H}_x \mathbf{M}^{-1} \end{aligned}$$

This starts to hint at a method for choosing suitable variable changes: how can $\mathbf{M}$ be chosen such that $\mathbf{M}^{-T}\mathbf{H}\mathbf{M}^{-1}$ more closely resembles the identity matrix? The ideal case would be that $\mathbf{M}^{-T}\mathbf{H}\mathbf{M}^{-1} = c\mathbf{I}$, which can be rearranged to $c\mathbf{M}^T\mathbf{M} = \mathbf{H}$. In this case, the transformed Hessian matrix is the identity, which means that the contour map of the transformed problem is symmetric about the optimal point, gradients always point towards the centre, and a steepest descent algorithm converges to the solution in a single iteration. Needless to say, this kind of an improvement in performance would be more than welcome in the case of FWI, where a single iteration can take more than 24 hours even on a powerful supercomputer. However, unlike the example above, the FWI objective function does not vary quadratically with respect to the unknowns.

4.1.2 Nonlinear problems

For a problem, such as that above, with a quadratic objective function, the change of variables represented by the matrix $\mathbf{M}$ is chosen so as to compensate for the curvature of the functional surface, and thus make the problem more symmetric about the optimal point (this change is visible in Figure 4.1). When dealing with a nonquadratic problem such as FWI, however, the curvature of the functional surface is no longer constant, and therefore the optimal change of variables is dependent on the current values of the unknowns. In this case, it is more intuitive to think of preconditioning not as a change of variables, but as a black box which transforms the gradient, as evaluated in the original coordinates, into a more optimal direction in which to take a step. To see this connection, consider the steepest descent iteration in the transformed variables $\mathbf{u}$, and map this back to the original variables $\mathbf{x}$:

$$\begin{aligned}
\mathbf{u}^{(k+1)} &= \mathbf{u}^{(k)} + \alpha \nabla_u f \\
&= \mathbf{u}^{(k)} + \alpha \mathbf{M}^{-T} \nabla_x f \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha \mathbf{M}^{-1} \mathbf{M}^{-T} \nabla_x f \\
&= \mathbf{x}^{(k)} + \alpha (\mathbf{M}^T \mathbf{M})^{-1} \nabla_x f
\end{aligned}$$

This shows that, in the case of a simple solver at least, the action of the linear-change-of-variables preconditioner can also be thought of as an operator $(\mathbf{M}^T\mathbf{M})^{-1}$, which is applied to the gradient to obtain the step direction. Since it is only used at each step, it is then easy to imagine the operator being a function of the unknowns themselves, changing its behaviour as the solver updates each iteration. This observation is key to obtaining the benefits of preconditioning even on nonlinear problems.

To demonstrate how this works, it is instructive to look at what has become one of the canonical examples in nonlinear optimisation, the Rosenbrock banana function (Rosenbrock, 1960). It is constructed very similarly to the quadratic test problem above, but with the additional feature that the valley itself is curved: rather than being along the line $y = 0.1x$

it is along the line $y = x^2$. The function can be written as:

$$f = (x - a)^2 + b(y - x^2)^2$$

where the variables a and b are constants, and the global minimum of the function occurs at (a, a^2).

The Rosenbrock function with $(a, b) = (1, 100)$ is illustrated in Figure 4.2a, with the iterations of an unpreconditioned steepest descent optimisation shown on a contour map in Figure 4.2b. As before, the narrow valleys slow down the convergence severely, with the occasional large step being taken when the curvature of the valley causes it to line up with the step direction (this is noteworthy because the same effect is seen in FWI). It would obviously be desirable to fix this problem with preconditioning, as in the earlier, quadratic case. Above, I showed how applying preconditioning to a nonlinear problem can be expressed as finding an appropriate transformation $(\mathbf{M}^T\mathbf{M})^{-1}$ to apply to the gradient at each iteration, but the question remains how to do this most effectively.

Following the observations from earlier that the "ideal" change of variables satisfies $\mathbf{M}^T\mathbf{M} = \mathbf{H}$, one obvious way to do this is to examine the curvature of the functional surface at each iterate, and determine a step direction using the above construction $\mathbf{d} = (\mathbf{M}^T\mathbf{M})^{-1}\nabla f = \mathbf{H}^{-1}\nabla f$. This is the same as constructing a direction which points towards the minimum of a *local quadratic approximation* to the function f, and is vanishingly similar to Newton's method - the only difference being that in Newton's method the direction $\mathbf{H}^{-1}\nabla f$ is used as the actual step, rather than the direction for a line search. One problem with using the local quadratic approximation, however, is that sometimes it has no minimum at all. This occurs, for example, in the upper-central region of the plot in Figure 4.2b, where the quadratic approximation forms a saddle-type function. This happens when the Hessian matrix $\mathbf{H}$ is not positive semidefinite, which also means that there is no matrix $\mathbf{M}$ which satisfies $\mathbf{M}^T\mathbf{M} = \mathbf{H}$. There are several ways to handle this situation, but for the special case where the objective function can be written as the sum of squares - $f = \frac{1}{2}\mathbf{r}^T\mathbf{r}$, a particularly effective option is available. Rather than a local quadratic approximation to the function itself, instead approximate the neighbourhood of the function by forming a *linear* approximation to each of the residual components individually. The first derivatives of the residual components with respect to the model components is now a matrix, called the Jacobian matrix, and is typically written $\mathbf{J}$. The second derivative of the implied quadratic approximation from this construction then becomes $\mathbf{J}^T\mathbf{J}$ - sometimes referred to as the *Gauss-Newton Hessian* due to its connection with the Gauss-Newton method. Because of the way it is constructed, this matrix is always positive semidefinite, and thus the resultant approximation can never curve downwards, and always has a minimum (though it may not be unique, as I shall show later). Thankfully, both the Rosenbrock function above and most formulations of FWI have an objective function compatible with this technique.

For the case of the Rosenbrock function, the matrix $\mathbf{J}$ can be written:

$$\mathbf{J} = \sqrt{2}\begin{pmatrix} 1 & 0 \\ -2\sqrt{b}x & \sqrt{b} \end{pmatrix}$$

yielding a Gauss-Newton Hessian:

$$\mathbf{J}^T\mathbf{J} = \begin{pmatrix} 2+8bx^2 & -4bx \\ -4bx & 2b \end{pmatrix}$$

The preconditioned descent algorithm, then, constructs a step direction at each iteration using the inverse of this matrix:

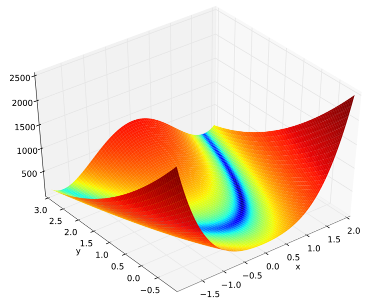
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha(\mathbf{J}^T\mathbf{J})^{-1}\nabla_x f$$

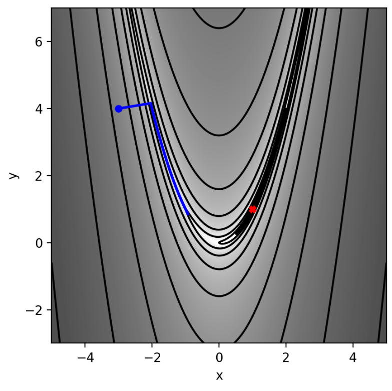
The results of this method are shown in Figure 4.2c. Even though seemingly-odd steps are taken sometimes, the convergence of the algorithm has been dramatically accelerated.

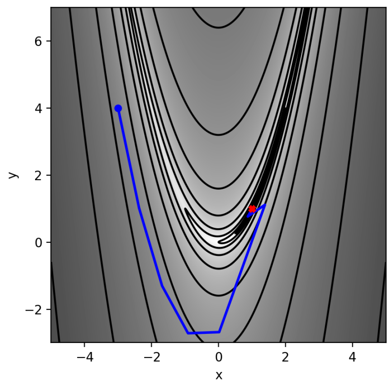
4.1.3 Large problems

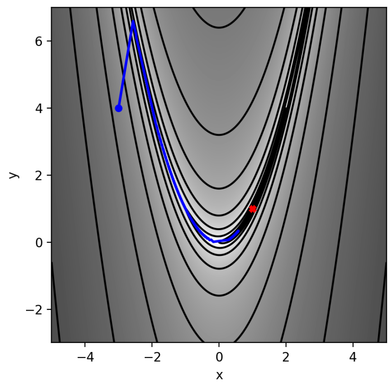
One last significant barrier stands in the way of applying this technique to FWI: the extremely large number of unknowns being solved for. In the case of the above problem, there are 2 unknown variables, and so the Hessian or Gauss-Newton Hessian matrix is only 2x2. Even a moderately-sized 3D FWI problem has more than 10^7 unknowns, and the operations required to find a gradient can take many hours on a large supercomputer. Construction of the Hessian or Gauss-Newton Hessian matrices requires around 10^7 times *more* computation than this - many thousands of years - and having done so, then occupies 10^{14} elements, or at least 400TB, of storage. The above method requires that this this matrix be computed, stored, and inverted at every iteration. Clearly, this is not practical.

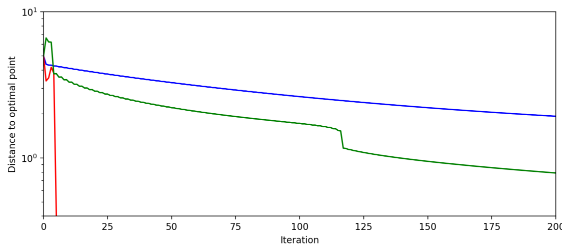
A very common method of sidestepping this issue is to approximate the matrix $\mathbf{J}^T\mathbf{J}$ by only its diagonal entries. While not being able to fully compensate for the interactions between model parameters, as represented by the non-axis-aligned valleys in the various contour plots above, this still compensates well for the overall relative sensitivities. As shown in Figure 4.1 at the beginning of this chapter, and applied to the Rosenbrock function in Figure 4.2d, this can still offer substantial improvements to convergence. I refer to this as *diagonal preconditioning.*


(a) A 3D view of the Rosenbrock function with (a,b)=(1,100)


(b) Path taken by 200 steepest-descent iterations


(c) Path taken by 9 JtJ-preconditioned descent iterations


(d) Path taken by 200 diag(JtJ)-preconditioned descent iterations



(e) Convergence graph of (b) (blue), (c) (red), (d) (green), showing distance to the optimal point.

Figure 4.2: The effect of various types of preconditioning when finding the minimum of the Rosenbrock function, which forms a curved valley with an almost-flat floor. With a steepest-descent solver (b), the optimisation proceeds very slowly along the narrow valley as expected. When preconditioned with the Gauss-Newton Hessian (c), the optimal point is reached after only 9 iterations. When preconditioned with the diagonal of the Gauss-Newton Hessian (d), the valley is traversed more quickly when it is aligned along the coordinate axes, but limited gains are observed where the orientation is close to 45 degrees. Note that in both of the preconditioned cases, the very early iterations do not reduce the functional as quickly as the steepest descent method, even though the convergence is much faster later on. Also note the large step taken in (d) at around iteration 115 - this occurs near the apex of the curve, when the step direction coincidentally points along the valley floor due to the curvature of the valley itself. A similar effect is observed in FWI inversions, possibly for the same reason.

4.2 Construction of a diagonal preconditioner for FWI

To construct the diagonal of the Gauss-Newton Hessian for FWI, I start with the definition of the canonical FWI functional, and the wave equation itself, from the introduction:

$$\begin{aligned}
f &= \frac{1}{2}\mathbf{r}^T\mathbf{r} \\
\mathbf{r} &= \mathbf{P}\mathbf{u} - \mathbf{d}_{\text{observed}} \\
\mathbf{u} &= \mathbf{A}^{-1}\mathbf{s}
\end{aligned}$$

Introducing the probing vectors $\mathbf{e}_i$, defined as having all elements equal to zero except for a 1 in the i'th element, one can then write an expression for an arbitrary column i of the Jacobian of the FWI problem for a single shot k. This describes the first derivatives of the residuals with respect to model parameter i:

$$\mathbf{J}_k\mathbf{e}_i = -\mathbf{P}_k\mathbf{A}^{-1}\left(\frac{\partial \mathbf{A}}{\partial m_i}\right)\mathbf{u}_k$$

An element i, j of the Gauss-Newton Hessian for the whole problem can then be written:

$$\begin{aligned}
(\mathbf{J}^T\mathbf{J})_{ij} &= \sum_k (\mathbf{e}_i\mathbf{J}_k^T\mathbf{J}_k\mathbf{e}_j) \\
&= \sum_k \left(\mathbf{u}_k^T\left(\frac{\partial \mathbf{A}^T}{\partial m_i}\right)\mathbf{A}^{-T}\mathbf{P}_k^T\mathbf{P}_k\mathbf{A}^{-1}\left(\frac{\partial \mathbf{A}}{\partial m_j}\right)\mathbf{u}_k\right) \qquad (4.2)
\end{aligned}$$

Consider the contribution of a single shot k to the expression above. There are several ways to think about computing it:

- From right-to-left:
  1. a virtual source based on the original forward wavefield at location j is used as a source in the wave equation
  2. the wavefield from this virtual source is sampled at each receiver location
  3. these received waveforms are re-injected as virtual sources in a reverse-time wave

equation

4. this new wavefield is sampled at location i and cross-correlated with the original forward wavefield

- From the outside in:
  1. two virtual sources, one based on the original forward wavefield at location i and the other at location j, are used as sources in two separate wave equation solutions
  2. the "i wavefield" and the "j wavefield" are each sampled at the receiver locations, and the traces cross-correlated and summed
- From the inside out:
  1. From each receiver location, propagate a single delta function in the last time sample as a virtual source in the reverse-time wave equation
  2. At each time step, cross-correlate with the forward wavefield at i and j at both the current time and all earlier times, taking the product at each time shift and summing over all time shifts. This makes use of the time-invariance of the wave equation.

To compute the N diagonal elements of the Gauss-Newton Hessian, the first scheme requires $2N$ wave-equation solves per shot, the second requires N per shot, and the last requires as many wave equation solves as there are receiver locations in each shot. This last one, while probably the cheapest by several orders of magnitude, is still too inefficient to compute without further sophisticated approximations that bring down the effective receiver count. To obtain a practical preconditioner, a cheaper approximation is therefore necessary.

4.3 Approximations

A very common choice of approximation, described in Shin et al. (2001) in the context of Reverse Time Migration, is to simply ignore the middle part $\mathbf{A}^{-T}\mathbf{P}_k^T\mathbf{P}_k\mathbf{A}^{-1}$ of equation

(4.2). Plessix and Mulder (2004) show that this is equivalent to assuming a uniform distribution of receivers throughout the medium for each shot.

However, by considering the first item in the above list more carefully, and restricting focus to the common case where $\mathbf{P}$ is a time-invariant operator which simply samples the wavefield at a desired location, an important symmetry can be exploited: the phase component of the solution of the wave equation between i and each receiver location is cancelled out by the combination of forward and backward propagation. This opens up the possibility of using some simple approximations for the amplitude.

Second to "no amplitude decay" as assumed by the scheme of Shin et al. (2001), the next simplest approximation to be used here is spherical divergence - in other words, assuming a constant background velocity. In 2D, this gives a $\frac{1}{\sqrt{r}}$ dependence of amplitude on the distance of each point from the receiver, or $\frac{1}{r}$ in 3D. This gives the first expression for my new approximation of the diagonal, for dimension d:

$$(\mathbf{J}^T\mathbf{J})_{ii} = \sum_k \left(\mathbf{u}_k^T \left(\frac{\partial \mathbf{A}^T}{\partial m_i} \right) \left(\sum_r |\mathbf{x}_i - \mathbf{x}_r|^{-(d-1)} \right) \left(\frac{\partial \mathbf{A}}{\partial m_i} \right) \mathbf{u}_k \right) \tag{4.3}$$

Better approximations are possible at the cost of more complexity. For instance, there is an analytic solution for the acoustic Green's function in a medium with a linear gradient of slowness squared (Li et al., 1990), or approximate Greens functions could be computed by dynamic ray tracing (Cerveny, 2005; Yao et al., 2017). However, these are not considered here.

4.4 Efficient Implementation

The central expression in equation (4.3), while several orders of magnitude less costly than solving the wave equation for each receiver, is still not a particularly cheap operation computationally speaking. A typical 3D towed-streamer survey has several thousand receiver locations per shot, which makes the cost of computing the central part of the approximation ($O(NN_r)$) comparable to 1 solve of the wave equation ($O(NN_t)$).

To make the approximation cheaply computable on the fly, I make use of the fact that for model points i sufficiently distant from a group of receivers, the contribution from that group can be approximated through the use of a multipole expansion of their contributions. This allows the construction of a recursive scheme where the contributions from single receiver locations are only computed in relatively small boxes around the locations themselves.

To do this, I build a binary tree where each leaf node is a receiver location and each non-leaf node can be thought of as representing all of the receiver locations below it. A simple example with 5 receivers is illustrated in Figure 4.3. Each non-leaf node carries:

- a mean position for the receivers it represents.
- the coefficients of an approximation which allow the contribution for those receivers to be computed. These will be described in the following section.
- the distance from the mean position beyond which the approximation is sufficiently accurate.

The tree is an augmented variation of the k-D tree (Bentley, 1975), which is designed to group geometrically nearby elements together. It is constructed from the top down, since the top node represents the entire set of receiver locations for a given shot. First, the approximation coefficients are computed, and then the set of receivers is split into two groups by splitting the set of receivers along the longest dimension through the mean receiver location. The two child nodes are then initialised by recursing the procedure for each of the two subgroups.

The approximation itself is constructed from a Cartesian multipole expansion around the mean receiver position in each group. Consider the contribution from a group of receivers at offsets $\mathbf{x}_k, k \in 1, 2, ..., n_r$ from this mean position, at a distant point $\mathbf{x}$, also relative.

$$\begin{aligned} f(\mathbf{x}) &= \sum_k f_k(\mathbf{x}) \\ &= \sum_k |\mathbf{x} - \mathbf{x}_k|^{-(d-1)} \\ &= \sum_k \left((\mathbf{x} - \mathbf{x}_k)^T (\mathbf{x} - \mathbf{x}_k)\right)^{-0.5(d-1)} \end{aligned} \tag{4.4}$$

Then, letting $x_{k,i}$ denote the ith component of the vector $\mathbf{x}_k$, write the Taylor expansion around $\mathbf{x}_k = \mathbf{0}$:

$$f_k(\mathbf{x}) = |\mathbf{x}|^{-(d-1)} + \sum_i \left(x_{k,i} \left. \frac{df_k}{dx_{k,i}} \right|_{\mathbf{x}_k=\mathbf{0}} \right) + \frac{1}{2} \sum_{i,j} \left(x_{k,i}x_{k,j} \left. \frac{d^2 f_k}{dx_{k,i}dx_{k,j}} \right|_{\mathbf{x}_k=\mathbf{0}} \right) + O(|\mathbf{x}_k|^3)$$

Note that this expansion is in terms of $\mathbf{x}_k$, not $\mathbf{x}$. The accuracy therefore depends on having a tight cluster of receivers relative to the distance $|\mathbf{x}|$ to the model point.

Expanding the derivative terms in the above expression:

$$\begin{aligned}
\frac{df_k}{dx_{k,i}} &= (-2(x_i - x_{k,i}))(-0.5(d-1)) \left((\mathbf{x}-\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k)\right)^{-0.5(d+1)} \\
&= (x_i - x_{k,i})(d-1) \left((\mathbf{x}-\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k)\right)^{-0.5(d+1)} \\
\frac{d^2 f_k}{dx_{k,i}dx_{k,j}} &= (x_i - x_{k,i})(x_j - x_{k,j})(d-1)(d+1) \left((\mathbf{x}-\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k)\right)^{-0.5(d+3)} \\
&- \delta_{ij}(d-1) \left((\mathbf{x}-\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k)\right)^{-0.5(d+1)}
\end{aligned}$$

And then evaluating them at the origin:

$$\begin{aligned}
\left. \frac{df_k}{dx_{k,i}} \right|_{\mathbf{x}_k=\mathbf{0}} &= x_i(d-1)|\mathbf{x}|^{-(d+1)} \\
\left. \frac{d^2 f_k}{dx_{k,i}dx_{k,j}} \right|_{\mathbf{x}_k=\mathbf{0}} &= x_i x_j (d-1)(d+1)|\mathbf{x}|^{-(d+3)} - \delta_{ij}(d-1)|\mathbf{x}|^{-(d+1)}
\end{aligned}$$

Noting that the last two expressions have no dependence on k, the original Taylor expansion reduces to:

$$\begin{aligned}
f(\mathbf{x}) = n_r|\mathbf{x}|^{-(d-1)} + (d-1)|\mathbf{x}|^{-(d+1)} \sum_i (\sum_k x_{k,i}) x_i \\
+ \frac{1}{2}(d-1)|\mathbf{x}|^{-(d+3)} \sum_{i,j} (\sum_k x_{k,i}x_{k,j})((d+1)x_i x_j - \delta_{ij}|\mathbf{x}|^2)
\end{aligned}$$

Since all the coordinates are relative to the mean receiver position in the group, the second term is zero since $\sum_k x_{k,i} = 0 \, \forall i$.

Defining $\mathbf{Q} = \sum_k \mathbf{x}_k \mathbf{x}_k^T$ for brevity, I now write the remaining terms as:

$$f(\mathbf{x}) = n_r|\mathbf{x}|^{-(d-1)} + \frac{1}{2}(d-1)|\mathbf{x}|^{-(d+3)}((d+1)\mathbf{x}^T\mathbf{Q}\mathbf{x} - (\mathrm{tr}\mathbf{Q})\mathbf{x}^T\mathbf{x}) + O(|\mathbf{x}_k|^3)$$

The coefficients required to represent the contribution from the combined group of receiver locations are simply the mean location $\mathbf{x}_0$, the number of locations n_r, and the symmetric 3x3 or 2x2 matrix $\mathbf{Q}' = (d-1)(d+1)\mathbf{Q} - (d-1)\mathrm{tr}(\mathbf{Q})\mathbf{I}$.

The error in the approximation can be estimated by comparing the relative size of the two terms:

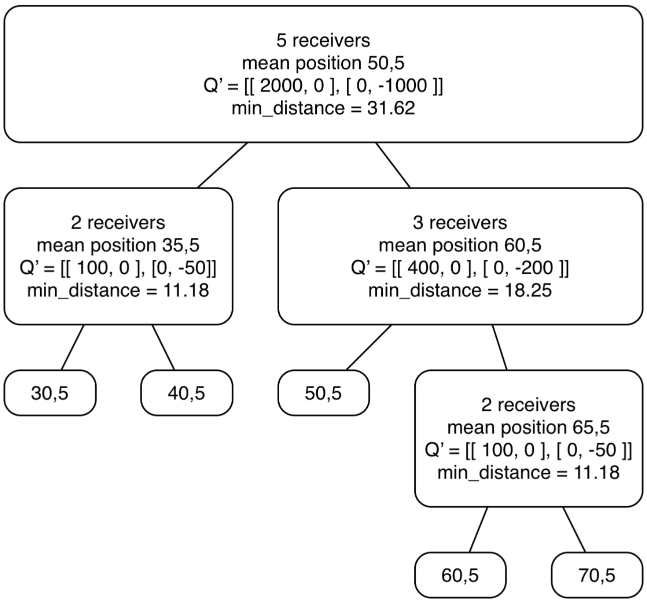
$$E(\mathbf{x}) = \frac{1}{2n_r}|\mathbf{x}|^{-4}\mathbf{x}^T\mathbf{Q}'\mathbf{x}$$

which satisfies

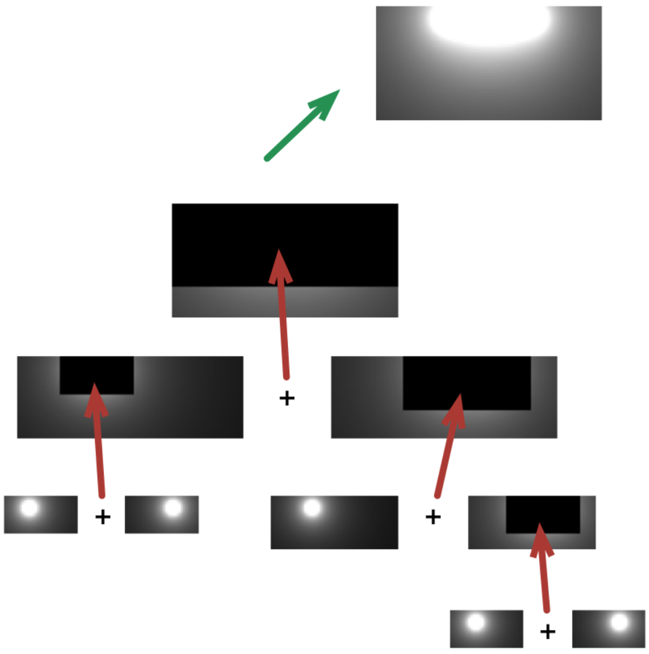
$$E(\mathbf{x}) < \frac{1}{2n_r}|\mathbf{x}|^{-2}||\mathbf{Q}'||$$

where the matrix norm refers to the largest eigenvalue, which can be simply calculated since $\mathbf{Q}'$ is only a 2x2 or 3x3 matrix at each node of the tree. Given an error tolerance ϵ, this permits the calculation of a minimum distance $|\mathbf{x}|$ for each node in the tree, beyond which the approximation can be safely used:

$$\begin{aligned}\frac{1}{2n_r}|\mathbf{x}|^{-2}||\mathbf{Q}'|| &< \epsilon \\ |\mathbf{x}|^{-2} &< \frac{2n_r\epsilon}{||\mathbf{Q}'||} \\ |\mathbf{x}| &> \sqrt{\frac{||\mathbf{Q}'||}{2n_r\epsilon}}\end{aligned}$$



(a) The division of the 5 receivers into a binary tree, grouping nearby receivers together and forming the terms of an approximation for their contribution to the sum $f(\mathbf{x})$.



(b) Recursive process used to construct the approximated sum: exact contributions are computed for each receiver in small boxes near to the location, and successively cheaper approximations are used further away.

Figure 4.3: A simple example, showing the construction of the tree and computation of the approximation for 5 receiver locations.

4.5 Performance and Results

4.5.1 Computation Performance

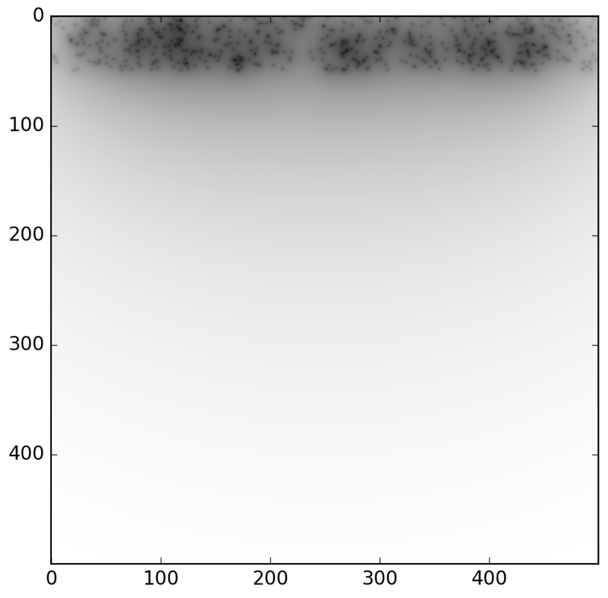


Figure 4.4: The function $f(\mathbf{x})$ evaluated for 500 randomly-placed points in the upper region of a 500x500 space

The algorithm above was used to compute an approximation to the sum (4.4) for various numbers of the receiver location points $\mathbf{x}_k$ in a 500x500 grid. The points were distributed randomly in the top 10% of the space. While this is not representative of a realistic receiver configuration in a seismic experiment, it was chosen to illustrate the generality of the above method with regards to arbitrary receiver positions. The exact computation of the function $f(\mathbf{x})$ is shown in Figure 4.4. The computation time for the approximation is shown in Table 4.1. The compute time for the approximation is broken into two parts: the time to build the tree of receivers, which scales more-or-less linearly with receiver count, and the time to compute the approximation, which scales linearly with the domain size but logarithmically with the receiver count. The observed relative errors are typically far smaller than the error threshold ϵ. This is because ϵ is expressed as the maximum difference between the first- and second- order approximations: the accuracy of the second-order approximation itself

Number of points	ϵ	Maximum Relative Error	Time (s)		
			Tree Build	Approximation	Total
100	0.050	6.298e-03	0.0149	0.2661	0.2809
	0.040	4.258e-03	0.0149	0.2994	0.3142
	0.030	2.410e-03	0.0149	0.3434	0.3583
	0.020	1.132e-03	0.0149	0.3717	0.3865
	0.010	3.559e-04	0.0149	0.5303	0.5451
	exact				1.0796
500	0.050	4.757e-03	0.0845	0.4172	0.5017
	0.040	3.521e-03	0.0845	0.4751	0.5596
	0.030	1.705e-03	0.0845	0.5500	0.6345
	0.020	7.640e-04	0.0845	0.6269	0.7114
	0.010	2.013e-04	0.0845	0.9871	1.0716
	exact				5.4138
1000	0.050	5.477e-03	0.1493	0.5523	0.7016
	0.040	2.562e-03	0.1493	0.6091	0.7584
	0.030	1.380e-03	0.1493	0.6926	0.8419
	0.020	6.736e-04	0.1493	0.8315	0.9808
	0.010	1.987e-04	0.1493	1.1860	1.3353
	exact				10.8717

Table 4.1: Timing for the computation of the sum $f(\mathbf{x})$ for various numbers of receiver points in a 500x500 domain. Tree build time scales approximately linearly with number of receiver points, but is independent of domain size. Time to compute the approximation increases very slowly with the number of receiver points, but is approximately linear with the domain size. Since ϵ is the ratio between the first- and second-order terms, the observed error between the second-order approximation and the true result tends to go as ϵ^2.

scales approximately as ϵ^2. For typical receiver counts between 100 and 500, and relative errors of $< 10^{-3}$, the approximation results in a speedup of between 3-10x compared to a naive summation.

4.5.2 Chevron/SEG 2014 Dataset

After I incorporated this new preconditioner into PyFWI, I proceeded to test it on two popular inversion problems: the Marmousi model, and the Chevron/SEG 2014 blind test dataset (Chevron/SEG, 2014).

The Chevron/SEG 2014 dataset is created from a highly realistic 50km x 6km seismic model, based upon field seismic data, well logs and rock physics. Using this model, Chevron generated synthetic data simulating a conventional 2D towed- streamer marine acquisition geometry with a maximum offset of 8000m. Reportedly with structural features inspired by real surveys in North Western Australia, the model has a non-deterministic, realistic relationship between P-wave velocity, S-wave velocity and density. It is locally isotropic, has limited bandwidth, ambient noise, surface ghosts and surface multiples, and the provided data was modelled using the full visco-elastic two way wave equation. Chevron created this dataset as a benchmark to test FWI and related technology, and have not released the true model. It therefore presents a realistic inversion challenge against which algorithmic changes can be tested, and allows experiments to avoid the so-called "inversion crime" of inverting with the same modelling assumptions that were used to generate the original data.

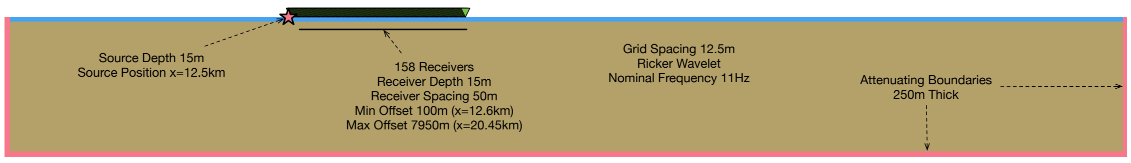


Figure 4.5: Acquisition geometry and experimental setup for inversion of the Chevron/SEG 2015 blind test dataset, showing position of source and receivers for a single shot.

In addition to the synthetic seismic data, Chevron also provided a source signature, a sonic profile in one well with a depth range of 1000-2450m, a 1-D starting P-wave velocity model, the velocity of the water column, and the geometry of the seabed. Since the provided starting model is badly cycle-skipped at the lowest usable frequencies in the data, and addressing cycle-skipping is not the goal of this work, for demonstration purposes I used a starting model derived using Adaptive Waveform Inversion (AWI) (Warner and Guasch, 2014). By using a objective function based on matching filters, AWI is able to perform inversion even when the starting model predicts traces which are cycle-skipped compared to the observed data (Guasch and Warner, 2014). A gaussian smoother was applied to the final AWI output model before starting this inversion test.

The results of two inversions are shown in Figures 4.6 and 4.7. Both were performed using conjugate gradient, in the time domain, with data filtered to below 11Hz. The first inversion was performed using a preconditioner based only on the source wavefield, using the assumption of Shin et al. (2001). The second inversion incorporated the approximate receiver Green's functions described above. The convergence rate of the inversion with the approximate receiver Green's functions in the preconditioner is visibly higher than that for the inversion preconditioned by the source wavefield alone. Significantly larger updates are made in the deeper part of the model, with only limited adverse effects in the near-surface section, and the resolution of the recovered model is significantly higher.

The functional convergence plot in Figure 4.8 shows the evolution of the functional over time for the two tests. On this chart, the relative performance of the two preconditioners appears somewhat comparable, and the new preconditioner even seems to perform a little worse. However, in ill-conditioned problems such as FWI, I believe that comparing the functional values is a misleading measure of improvement: while the preconditioner itself seeks to compensate for the imbalance in sensitivity between the shallow and deep regions of the model, the definition of the functional has not changed, and it remains relatively insensitive to deeper model updates. For example, this would mean that even if the deeper update were improved substantially, a very slightly poorer shallow result would lead to a larger value of the objective function.

I therefore seek a measure of misfit that is in the model domain. In this case, the true model is not known, but even if it were this would not necessarily be a good indicator of convergence performance either - this is not an inverse crime problem. The inversion is being computed on a different grid, with a different wave equation, and it is not necessarily true that the best-fitting acoustic, constant-density, isotropic model on a given grid is going to be a naive resampling of the true P-wave velocity used in fully-elastic modelling.

One way of evaluating convergence in the model space without knowing the true answer is to compare the results of iterations with/without the preconditioner. For example, if the preconditioned result after 15 iterations looks qualitatively very similar to the unpreconditioned result after 40 iterations, one might say that the preconditioner has helped. In Figure 4.9, I have attempted to make a quantitative equivalent to this kind of a statement. After computing 100 iterations both with and without the receiver-location aware preconditioner, I have then plotted, for every pair (i, j), the RMS model difference between the models at iteration i of one inversion, and iteration j of another. The colour scale is chosen to highlight the region in the i, j plane where outputs from the two inversions are most similar. A vertical line through this plot corresponds to a model at a single iteration of the receiver-location aware inversion. By looking along a vertical line, it is possible to see which iteration of the other inversion is most quantitatively similar, in the model space, to this one. Thus, even without knowing the true answer, I can assert that inversion with the receiver-location-aware preconditioner tends to require fewer iterations to reach a similar model compared to the regular inversion. In this example, the the observed speedup is around 60%.

4.5.3 Marmousi

The Marmousi model is almost certainly the most frequently seen model in the FWI literature. It was created in 1988 by the Institut Francais du Petrole (IFP), as a blind test for use at the 1990 EAGE workshop on Seismic Data Inversion (Bourgeois et al., 1990). In the same way as the Chevron/SEG model was based on real data obtained in North Western Australia, the structure of the Marmousi model is intended to be reminiscent of the North Quenguela trough in the Cuanza Basin (Versteeg, 1993). Unlike the previous dataset, however, and perhaps because of its age, the true Marmousi model is known. To facilitate the use of coarse grids and a slightly different acquisition geometry than that chosen in the original paper, the synthetic data was generated using my own propagator. Additionally, I am using a variation of the Marmousi model with slightly deeper water to ensure that the non-grid-aligned source/receiver scheme of Hicks (2002) does not lead to sources being injected below the waterbottom. In contrast to the previous example, inversions with the Marmousi model were performed with all receiver locations active for each source position, to replicate something an ocean-bottom-seismometer type survey.

Analagous results to the above are shown for the Marmousi model in Figures 4.10, 4.12 and 4.13. Since I also have the true model in this case, I can explicitly measure the model errors for each iteration as well, and this is shown in Figure 4.11. The benefit of the receiver-location-aware preconditioning is minor in this example, barely noticeable when looking at the models visually, though around a 20% speedup is visible on the figures showing model error. However, the extra computation required to compute the preconditioner is still even smaller than this at around 10%, as shown in Figure 4.14, so it is still a net benefit.

4.6 Conclusion

In this chapter, I have given an introduction to how and why preconditioning helps in ill-conditioned optimisation problems such as FWI. I have then used the FWI code developed in the previous chapter to develop and demonstrate a very low cost, general purpose, diagonal preconditioner for FWI. The new preconditioner outperforms the commonly used diagonal preconditioner of Shin et al. (2001) by up to 60% in a towed-streamer example, and 20% in an ocean-bottom-seismometer example, as demonstrated on two common inversion benchmarks. The computational overhead of the new preconditioner is less than 10%, even without an optimised implementation in C or FORTRAN.

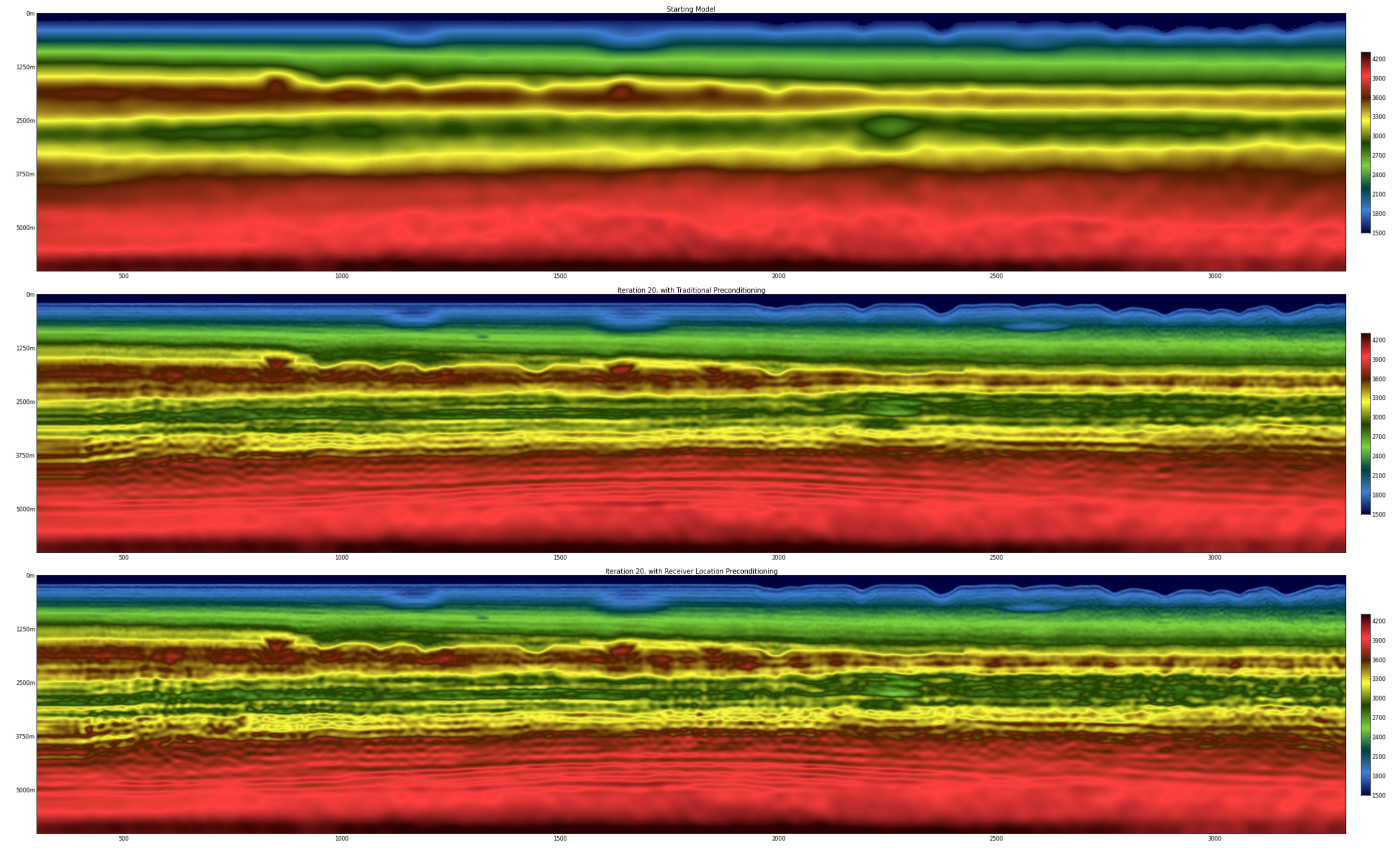


Figure 4.6: Chevron/SEG 2014: Centre section of the starting model (top), and the recovered models after 20 iterations with the traditional (centre) and receiver-location-aware (bottom) preconditioners.

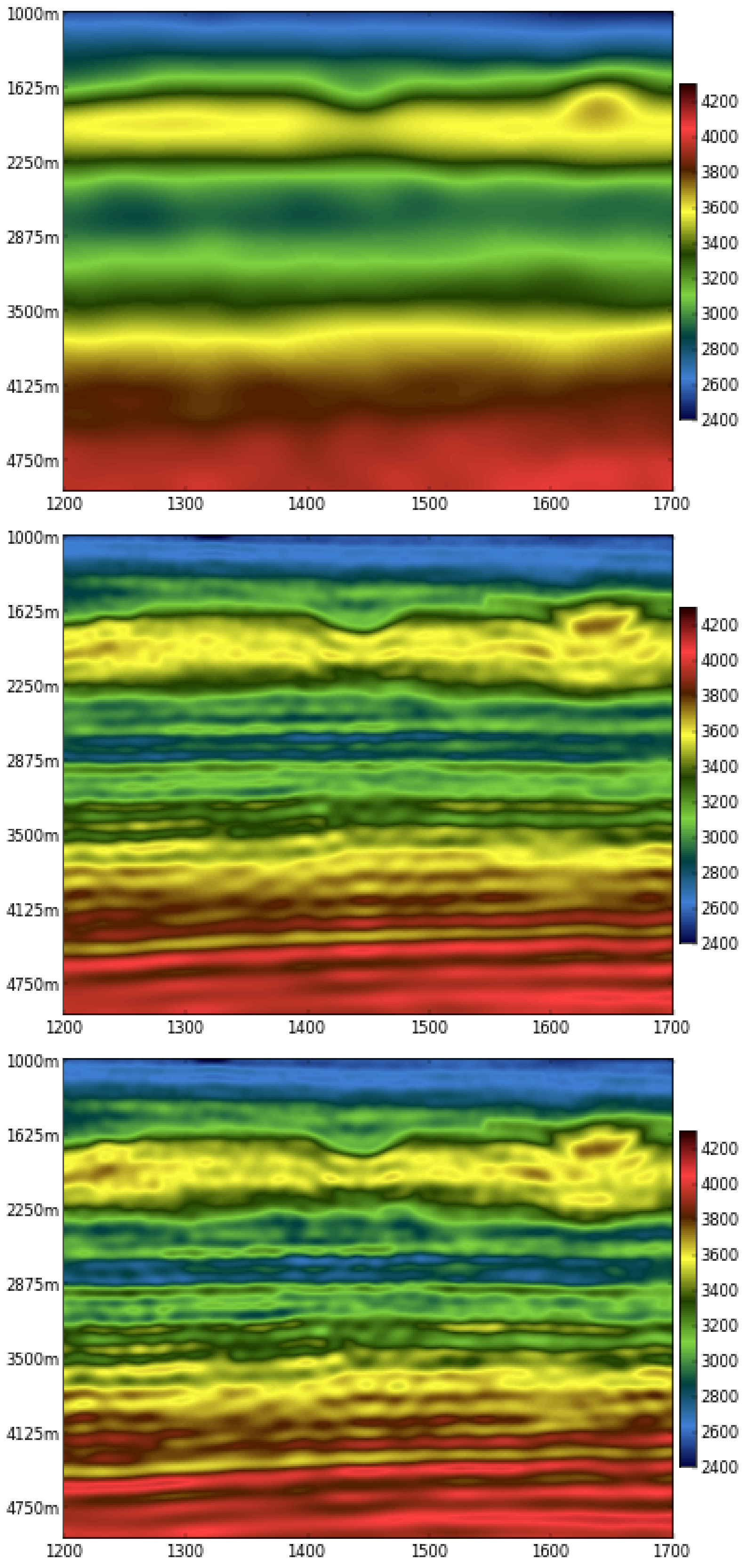


Figure 4.7: Chevron/SEG 2014: Zoomed in region of the models in Figure 4.6, with an adjusted colour scale to show the improved recovery of the low-velocity zone with the receiver-location preconditioning. Note the very slightly reduced amplitude of the shallow feature on the right hand side.

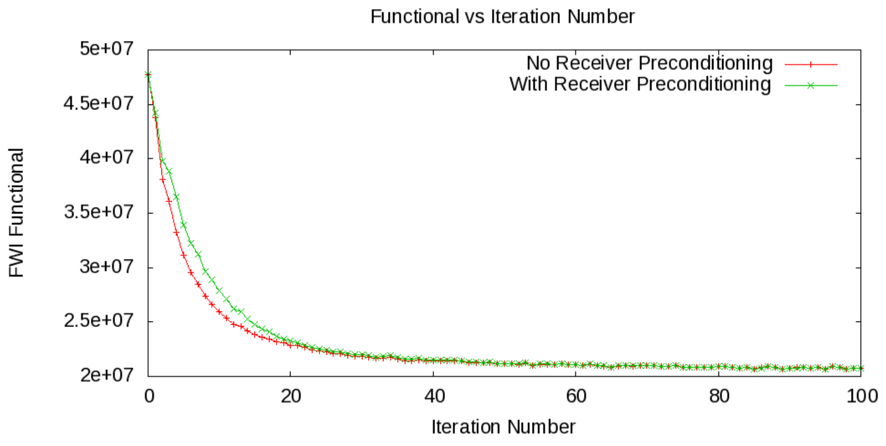


Figure 4.8: Chevron/SEG 2014: Evolution of functional during inversions with and without the receiver-location-aware preconditioning. The early iterations appear to converge slower with the new preconditioner, although they catch up at around iteration 20. This behaviour is not unexpected: the preconditioning increases the relative step sizes of model components which contribute less to the functional, so even though the model may be converging faster towards the optimal point when measured in the model-misfit domain, the functional may not decrease as quickly in the early iterations.

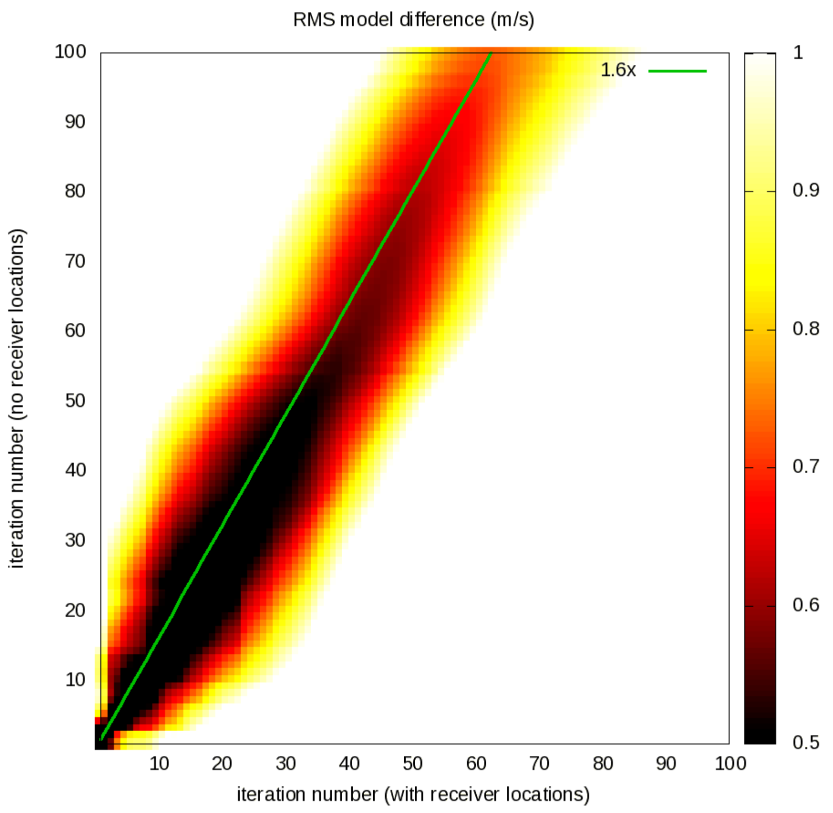


Figure 4.9: Chevron/SEG 2014 Test: Plot showing the RMS difference between every pair of models from the receiver-location-preconditioned inversion and an inversion performed with the traditional preconditioner. The colour scale is chosen to highlight the region where the models are most similar, and the line is chosen manually to give an estimate of the relative convergence rate. It shows that to obtain a similar model, the traditionally-preconditioned inversion requires about 60% more iterations than the receiver-location-aware inversion. Inversion was performed with conjugate gradient.

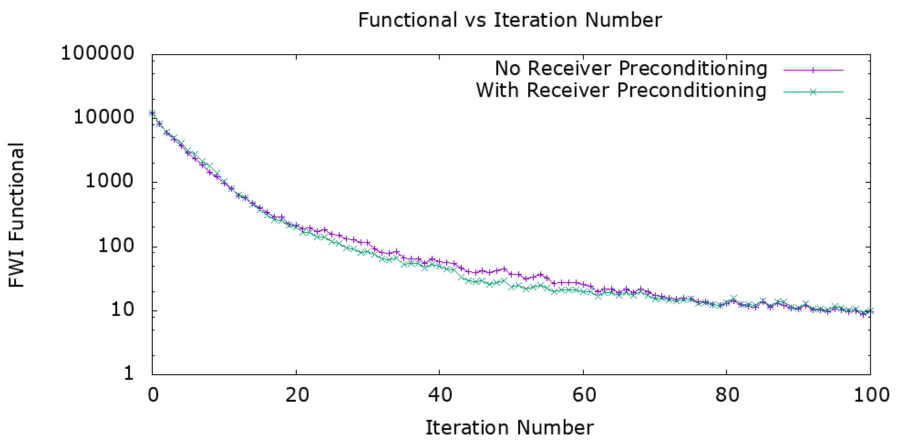


Figure 4.10: Marmousi: Evolution of functional during inversions with and without the receiver-location-aware preconditioning. Note that the preconditioned results appear to converge slightly (10%) more slowly for the first few iterations, similar to Figure 4.8.

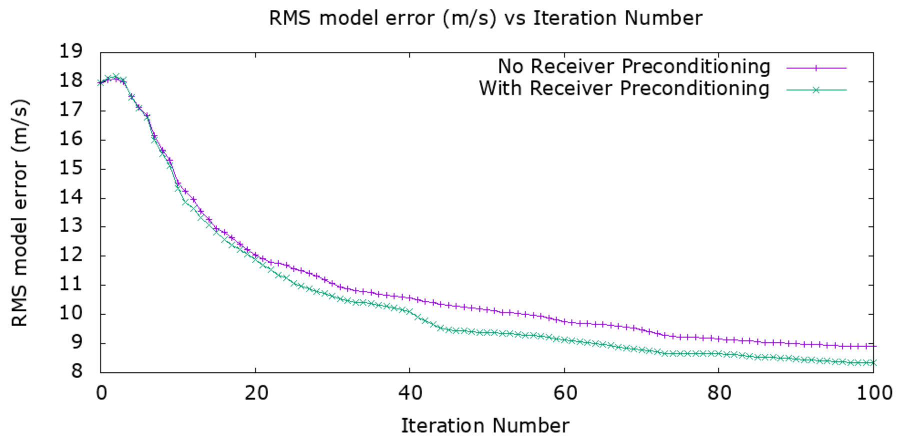


Figure 4.11: Marmousi: Evolution of the RMS model error during inversions with and without the receiver-location-aware preconditioning.

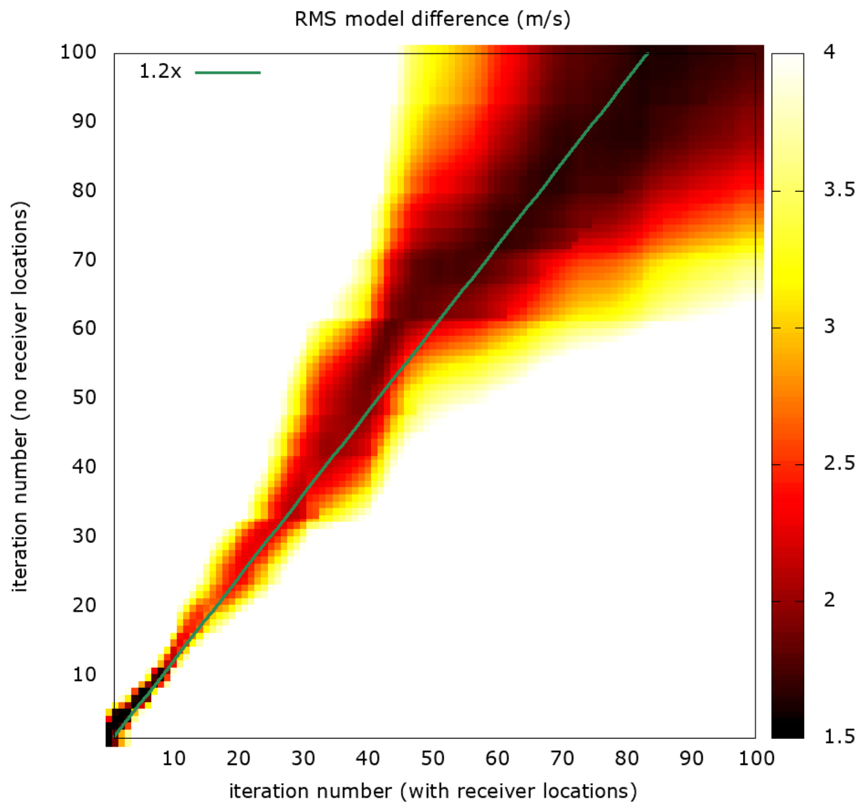


Figure 4.12: Marmousi: Plot showing the RMS difference between every pair of models from the receiver-location-preconditioned inversion and an inversion performed with the traditional preconditioner. The colour scale is chosen to highlight the region where the models are most similar, and the line is chosen manually to estimate the relative convergence rate. It shows that to obtain a similar model, the traditionally-preconditioned inversion requires about 20% more iterations than the receiver-location-aware inversion. Inversion was performed with conjugate gradient.

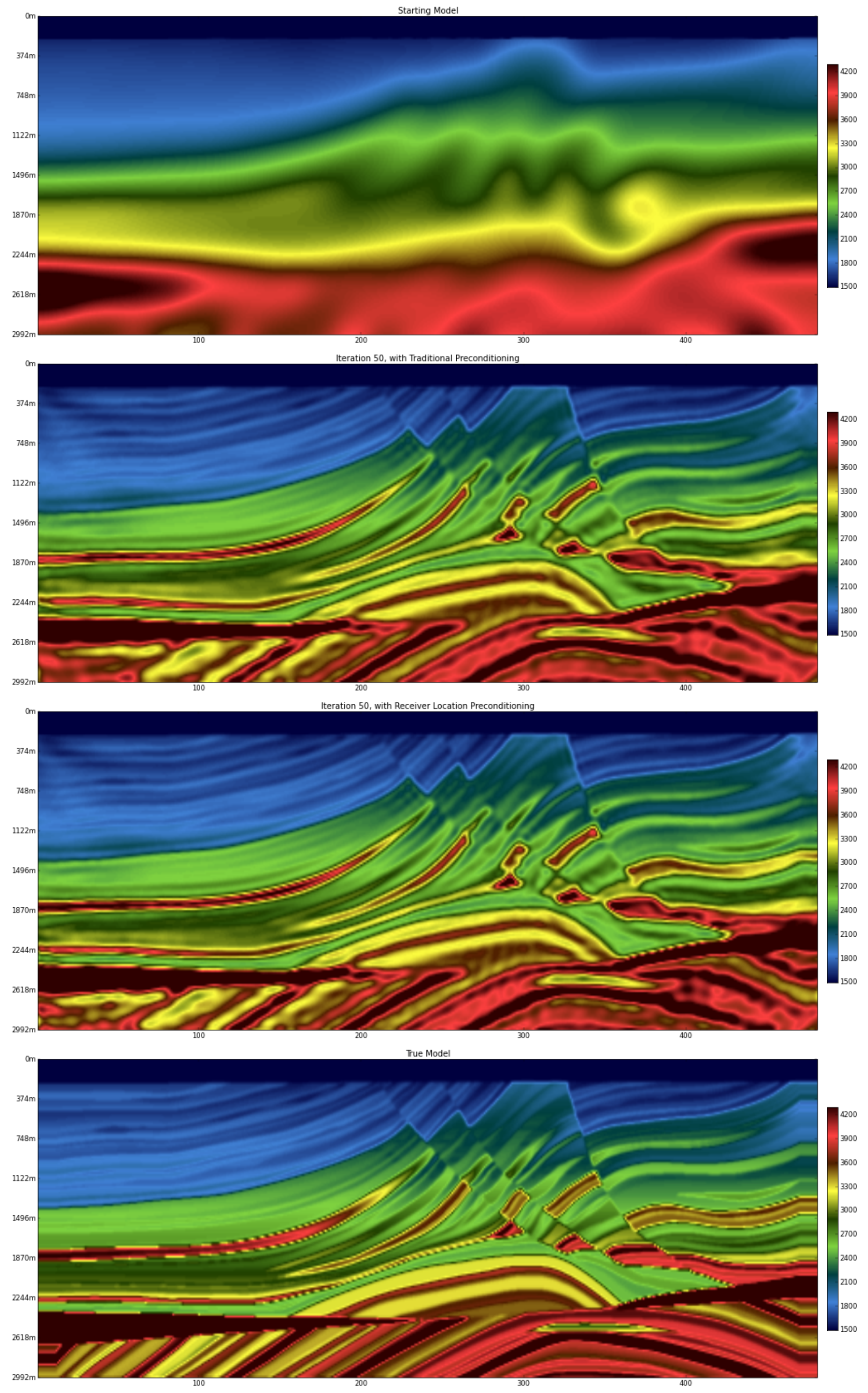


Figure 4.13: Inversion of the Marmousi model. From top: the starting model; model after 50 iterations of conjugate gradient with traditional preconditioning; model after 50 iterations of conjugate gradient with receiver-location-aware preconditioning; true model.

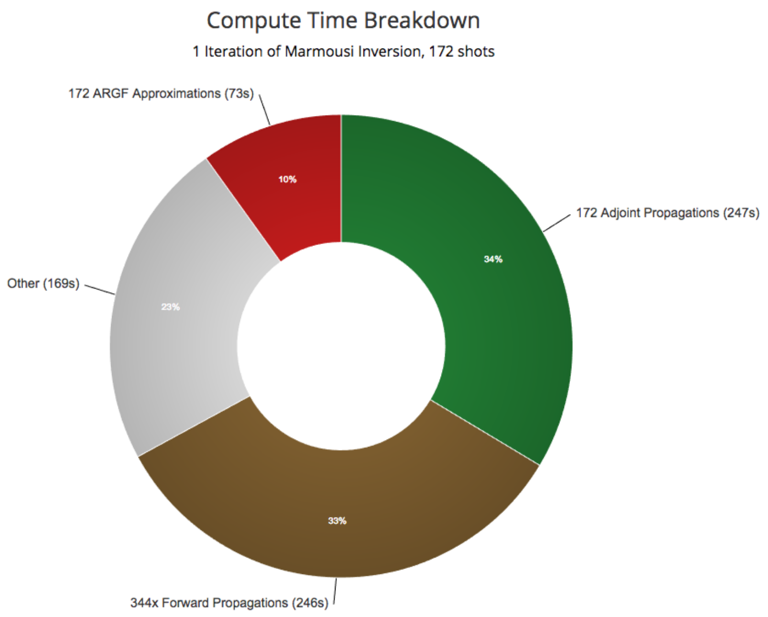


Figure 4.14: A breakdown of the compute time for a single iteration of the Marmousi Inversion of section 4.5.3. The computation of the extra preconditioner term, with $\epsilon = 0.01$, takes only 10% of the total compute time for an iteration. "Other" is all time not spent propagating wavefields or computing the approximage receiver Green's functions, and includes communication overhead, wasted time due to load imbalance, model extrapolation into boundaries, computation of the adjoint sources and model updates.

"For a successful technology, reality must take precedence over public relations, for nature cannot be fooled."

— Richard Feynman

5 Macromodel Recovery

5.1 Introduction

With the vast increases in available computational power, and the promise of automated model-building with relatively little manual involvement, FWI has risen to prominence in the petroleum industry over the past decade or so. Over this time, however, it has become apparent that beyond the depths well-sampled by transmitted arrivals, conventional FWI implementations struggle to recover information about the low-wavenumber velocity information (Xu et al., 2012; Warner et al., 2013). This is especially true with the limited availability of low acoustic frequencies in the marine towed-streamer acquisition environment. Thus, most successful applications of FWI to date target fairly shallow regions, up to around a third to a sixth of the maximum source-receiver offset (Morgan et al., 2013). At depths beyond this, where only reflection data is available, FWI has effectively been limited to imprinting high-resolution reflectivity images onto what are assumed to be already highly-accurate background velocity models, in a process strongly analagous to least-squares reverse time migration (LSRTM).

Various explanations have been offered for this, and numerous solutions proposed. Virieux and Operto (2009) point out that the mapping from acoustic frequency to wavenumber depends on the opening angle of the scattering, via the relationship $|k| \propto f \cos(\theta/2)$. Thus, the wide scattering angles ($\theta \approxeq \pi$) in the shallow permit high acoustic frequencies to be used to recover low wavenumbers, while the smaller range of observable scattering angles in the deeper section would require the presence of lower acoustic frequencies that, due to both instrument limitations or destructive interference from the sea-surface reflection, are often only weakly present in the data. Brossier et al. (2014) and Zhou et al. (2015) argue, among many other important points, that this poor recovery is inevitable without decoupling the reflectivity from the velocity, since otherwise it is impossible to simultaneously match both the required kinematics and the reflection amplitudes, upon which the FWI objective function places a relatively high importance. Other proposed solutions range from deghosting of FWI input data (Ramos-Martinez et al., 2013), to alternating inversion for high- and low-wavenumber components (Xu et al., 2012; Yao et al., 2014; Zhou et al., 2015), to attempts to linearise the problem somewhat by representing the high-wavenumber components in a different domain (Brossier et al., 2014), to sophisticated combinations of multiparameter gradients (Ramos-Martinez et al., 2016; Whitmore and Crawley, 2012).

In this chapter I will argue that there is a common underlying theme to all these explanations and solutions, and that the poor conditioning of the FWI problem, as discussed in the previous chapter, is a central issue when it comes to recovery of deeper low-wavenumber velocity information.

First, I will introduce a novel quality control technique for FWI, and then use it to demonstrate the lack of update to the deeper macromodel using one of the inversion examples from the previous chapter. Secondly, I will show that using the same data, existing petroleum-industry technology is able to make improvements to the model in this region. Then, after demonstrating that this problem can be reproduced in a one-dimensional inverse crime setting, I will show how to obtain and decompose the Hessian matrix for that problem, the results of which will show that poor conditioning is at least partially responsible for the lack of deep low-wavenumber updates. Finally, I will construct a preconditioner by applying a simple damping to the true Hessian, and show that in this one-dimensional problem it allows for meaningful updates to deep macromodel components which were previously unrecoverable.

5.2 A simple QC for FWI

I begin this discussion by introducing a simple, but I believe novel, QC for the FWI process, based on the checkerboard recovery experiments such as those described in Morgan et al. (2013). It starts from the observation that if FWI results are to be trusted, then small perturbations in the starting model should not make a substantial difference to the recovered model.

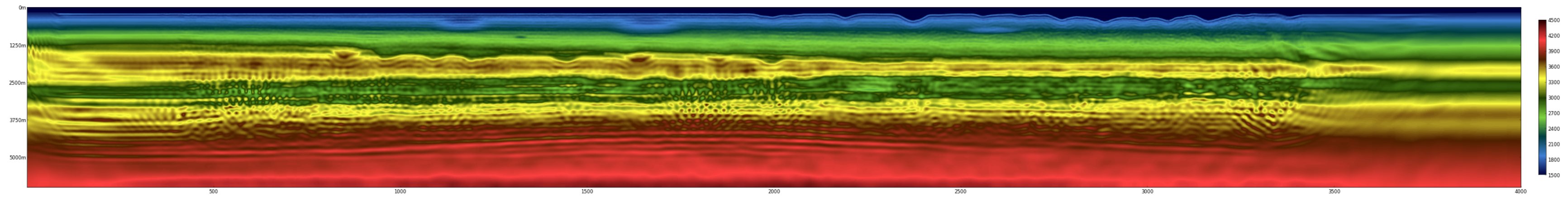


Figure 5.1: Result of inverting the Chevron/SEG 2015 blind test model after 200 iterations. Note the presence of artefacts at depths between 2500m and 3750m at several locations along the line.

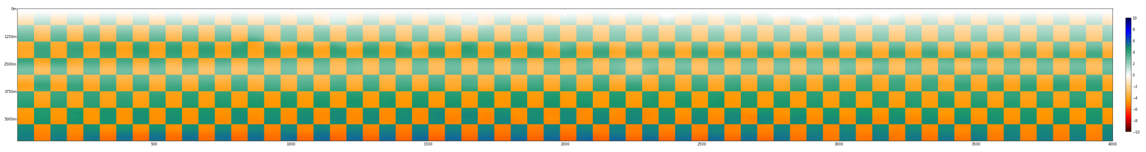


Figure 5.2: The difference between the two starting models used in the test inversion.

Consider the results from the previous chapter, but after many more iterations than one would typically perform, as shown in Figure 5.1. Along with the edge effects on the left and right, some high-wavenumber artefacts are beginning to appear between depths of 2500m and 3750m at several locations through the model. Since I am inverting "real-world" data with an acoustic approximation, it is conceivable that these artefacts - while almost certainly not geophysically realistic - are being introduced to match some aspect of the data that isn't being modelled correctly. In other words, it is possible that they would be present in the best-fitting acoustic, isotropic model for this data.

This hypothesis - that these artefacts are legitimate by-products of using inadequate physics in the modelling - would be somewhat supported if the artefacts themselves were insensitive to the starting model. To test this, I perform the same inversion, but this time do it twice. Each of the two inversions starts with with a very slightly different starting model. The two starting models are generated by taking the original starting model $c(\mathbf{x})$ and perturbing it with a checkerboard $d(\mathbf{x})$, which takes values ± 1 in a regular pattern but is zero inside the waterbody. The function I used to generate the two oppositely-perturbed starting models is:

$$c_{\pm}(\mathbf{x}) = c \pm 0.001(c(\mathbf{x}) - 1515)d(\mathbf{x}) \tag{5.1}$$

These two models are indistinguishable by eye on a full colourscale, and should be kinematically similar enough that they are not cycle-skipped with respect to each other. Figure 5.2 shows the difference between them.

As expected, since the two models are extremely similar, the same is generally true of their FWI iterates. However, showing how the difference between the two evolves as the iterations progress turns out to be quite informative. The function that I will plot is:

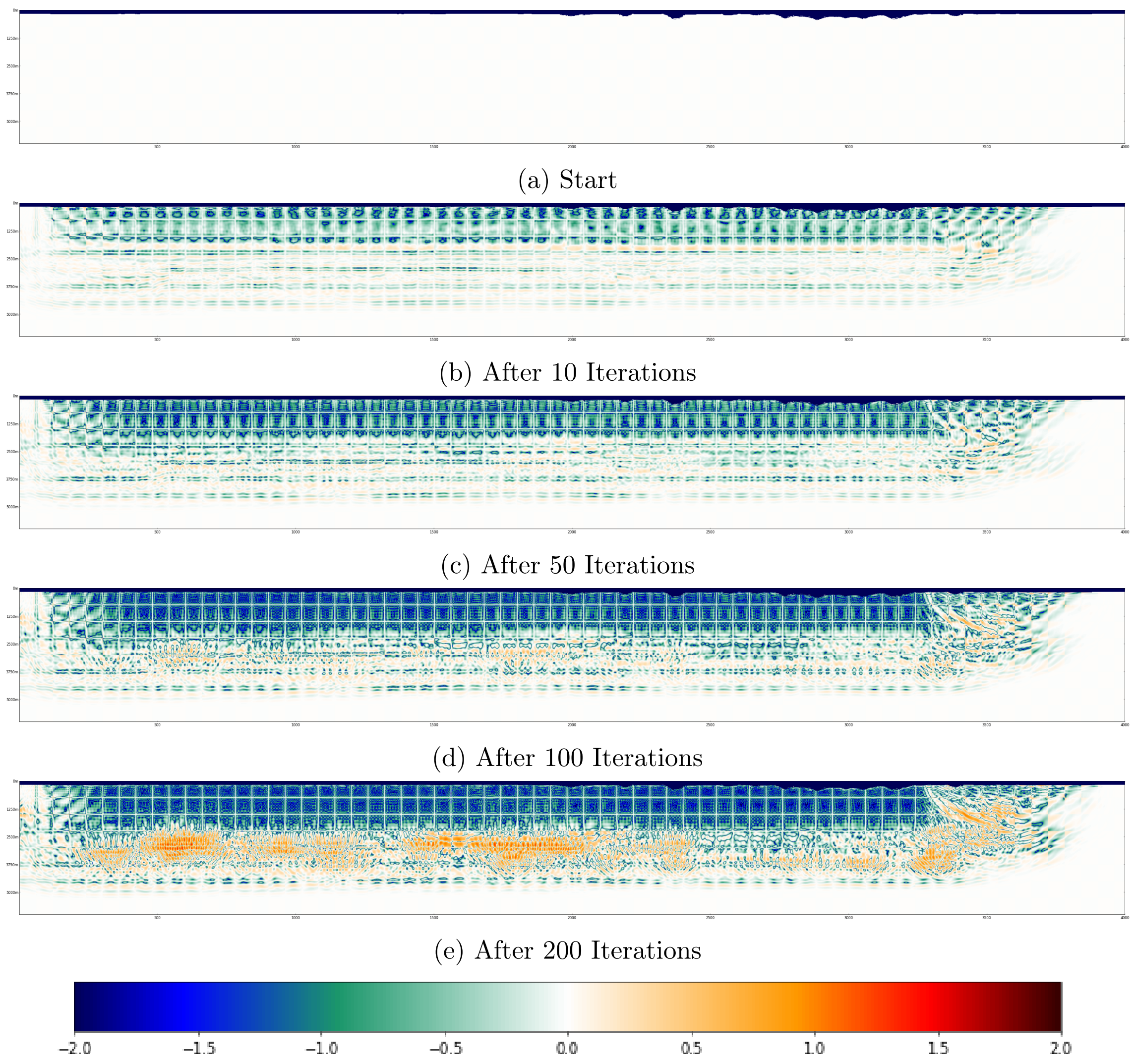
$$e(\mathbf{x}) = \log_{10} \frac{|c_+(\mathbf{x}) - c_-(\mathbf{x})|}{0.002(c(\mathbf{x}) - 1515)} \tag{5.2}$$

This function has the property that for regions where the difference shrinks or grows, $e(\mathbf{x})$ is less than 0 or greater than 0 respectively. Figure 5.3 shows this function evaluated at the starting models, and also after 10, 50, 100 and 200 iterations. In regions coloured white, the difference between the iterated models is equal to the difference between the starting models - these are generally regions where very little update has been made at all. Blue indicates regions where the difference has shrunk (the most intense blue representing 100x smaller than the starting difference), and red indicates regions where the difference has grown (the most intense red representing 100x larger than the starting difference). In the early iterations, FWI proceeds as expected - rapidly improving (blue) the model in the shallowest region, and "migrating in" corrections nearest to the boundaries of the checkers that make up the checkerboard. However, somewhere between 50 and 100 iterations the difference between the two inversions starts to increase (red) in certain regions of the model - the same regions where the artefacts were visible in Figure 5.1. By the 200th iteration, the differences between the inversion results for the two nearly-identical starting models have grown to more than 24x the starting difference in some regions.

In this particular case, this suggests that the artefacts are not necessarily by-products of the inadequate physics, but rather instabilities in the FWI process. I hypothesise that these correspond to directions along which the functional surface is not convex, and so the gradient descent pushes nearby models apart instead of together.

This technique also suggests a possible quantitative stopping criteria for FWI: when the RMS difference between corresponding iterations of the two inversions starts to increase, further iterations are, in a way, doing damage. Figure 5.4 shows how the root-mean-square difference between the models evolves as the iterations proceed. This suggests a stopping point at the following model, iteration 73, which - as shown in Figure 5.5 - is largely free from artefacts.

This QC technique is expensive in that it requires an inversion to be performed twice. I think that in practise this could be mitigated, for example by using half the number of



(a) Start

(b) After 10 Iterations

(c) After 50 Iterations

(d) After 100 Iterations

(e) After 200 Iterations

Figure 5.3: The function of equation (5.2) showing how the difference between two very-similar starting models evolves after various numbers of FWI iterations. Blue indicates areas where the difference between the models has been reduced, with the most intense blue representing 100x smaller than the starting difference. Red indicates areas where the difference has grown, the most intense red representing 100x larger than the starting difference.

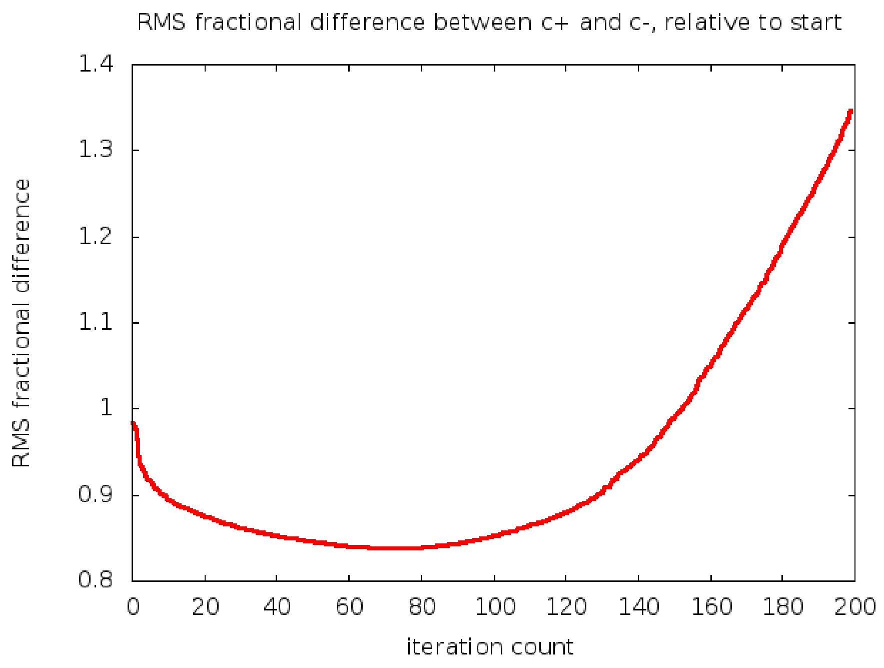


Figure 5.4: Plot showing how the normalised root-mean-square difference between the two models evolves as iterations proceed.

shots per iteration in each of the two inversions and using the mean of the two inversion results as the final output. When being used as a stopping criteria, it might also allow us to find out early that fewer iterations are justified. However, these investigations are beyond the scope of this work.

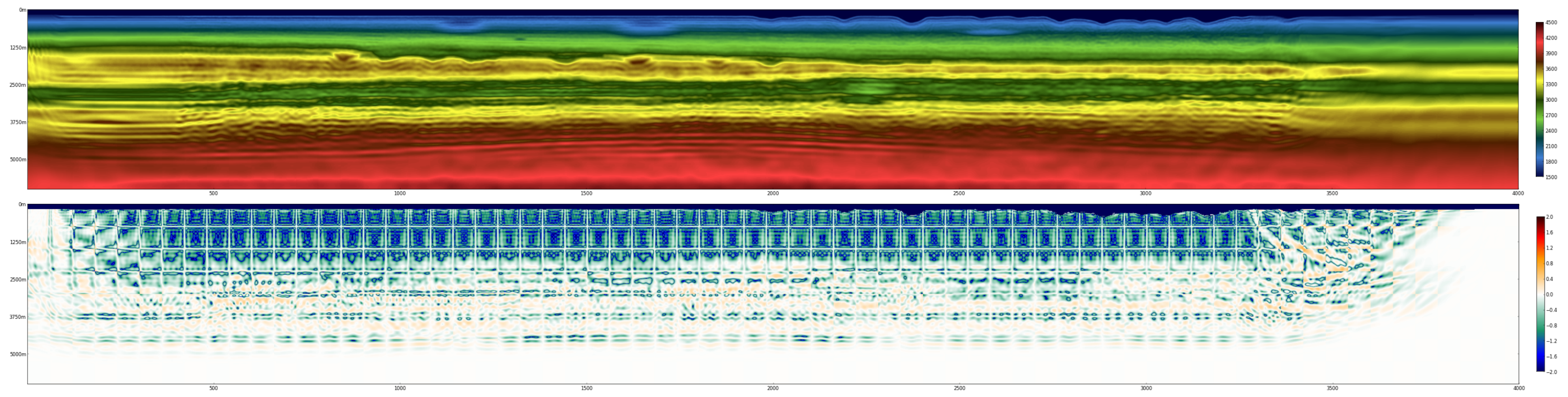


Figure 5.5: The average of the two models, and relative difference between them, after 73 iterations - the minimum of the plot in Figure 5.4

5.3 Deep Macromodel Recovery

As well as identifying regions with untrustworthy inversion output, the results above exhibit a well-known weakness of FWI discussed in the introduction to this chapter: below the region well-sampled by turning rays and wide-angle scattering, low-wavenumber components of the model see very limited improvement.

Beyond a depth of about 2500m in this model, and certainly beyond 3000m, the differences near the top and bottom boundaries of each square of the the checkerboard have been reduced, but often at the expense of the overall convergence - with the difference in other parts of each checkerboard square having been increased. This pattern corresponds with the observation of Xu et al. (2012), Morgan et al. (2013), and many others, that below the region where turning rays can be sampled, FWI updates are essentially restricted to higher wavenumbers, causing it to take on a more migration-like role. Typically this means simply creating (potentially mispositioned and/or badly focused) reflectors, rather than making significant updates to the low-wavenumber parts of the model.

Even though the FWI objective function might not be particularly sensitive to it, however, the input data does have sufficient information to constrain the macromodel in these deeper sections. We know this because traditional velocity model-building techniques work well in this case. To show that the information required for deeper macromodel updates is present in the example above, I applied a more traditional velocity model-building technique - reflection tomography - to the same dataset.

Starting with the same model as the inversions above, this was done by repeated application of a series of steps:

1. Kirchhoff depth migration on the data

2. On the resulting gathers, application of a combination of manual and automatic picking techniques to pick residual moveout, i.e. the apparent depths of events and how they vary with offset.

3. Computation of an update to the velocity model which minimises the variation in these apparent depths versus offset

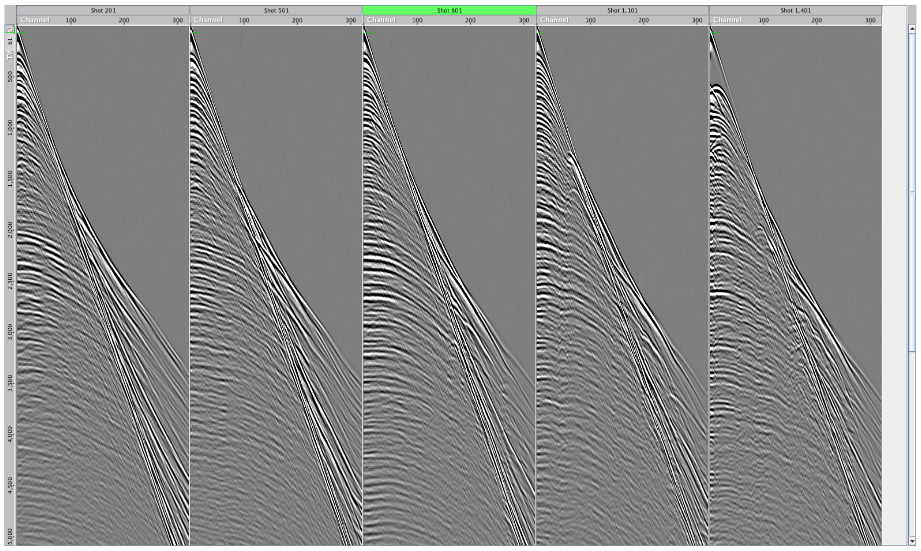


Figure 5.6: Some raw shot gathers from the 2014 SEG/Chevron blind test data.

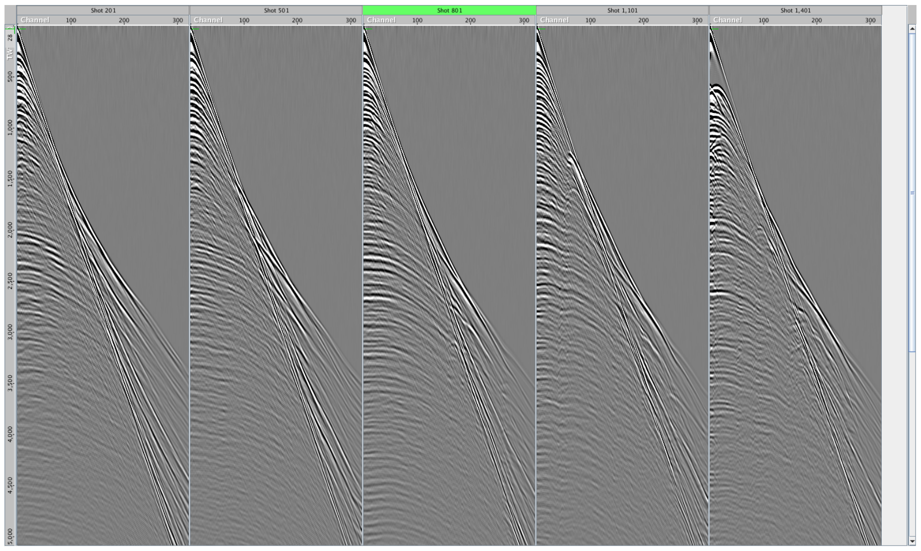


Figure 5.7: Step 1: Change amplitude decay from cylindrical to spherical.

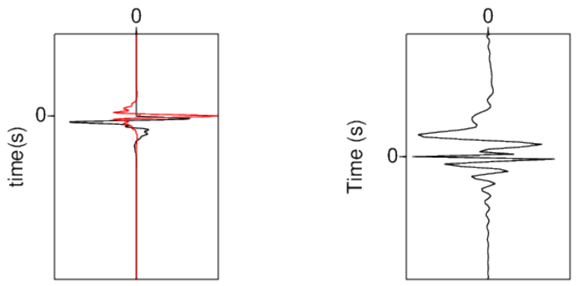


Figure 5.8: Step 2: From the source wavelet (left, black), develop a zero-phasing filter for the data (right) which transforms the wavelet to its zero-phased equivalent (left, red).

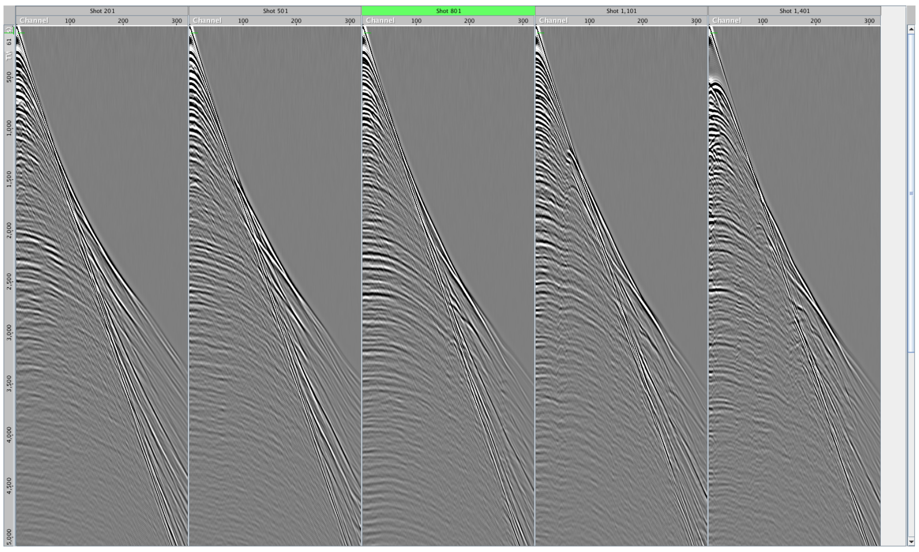


Figure 5.9: Step 3: Apply zero-phasing filter from Figure 5.8.

Before this could begin, some preprocessing of the data was required to prepare it for Kirchhoff migration. Since the synthetic data originated from a 2D model rather than a 2.5D one, the amplitude decay with time is less than would be expected in field data. For a constant-velocity model, in 3D, amplitude decays as the reciprocal of the distance from the source $1/r$, whereas in 2D this is $1/\sqrt{r}$ (remember that wave power is proportional to the square of amplitude). As the Kirchhoff migration code I was using is written to expect field data, the synthetic traces were scaled by a factor of $1/\sqrt{t}$ before migration.

Next, the data needed to be convolved with a filter to bring it to zero phase. The supplied traces represent seismic data received from the moment the (simulated) source begins to

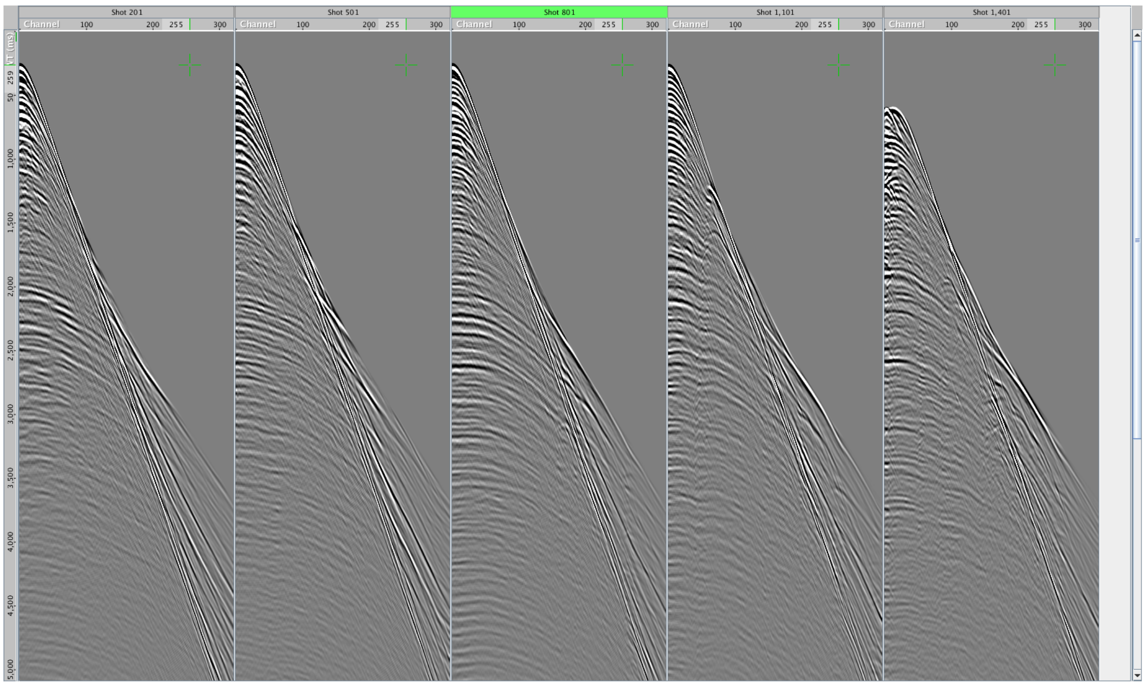


Figure 5.10: Step 4: Mute out direct arrivals.

fire, but as can be seen from the supplied source wavelet in Figure 5.8, the peak of the source amplitude occurs some time after that. This has the net effect that the reflection from a point in the subsurface with a given source→reflector→receiver traveltime appears slightly later than that in the seismic data. Before Kirchhoff migration can be used to map reflected pulses to points in the subsurface, this needs to be fixed. We do this in four steps, illustrated in Figures 5.8 and 5.9:

1. Compute the amplitude spectrum of the supplied wavelet.

2. Generate a wavelet with the same amplitude spectrum, but with all phases set to 0 - this *zero phased* wavelet is by definition symmetric about zero time.

3. Compute a matching filter which, when convolved with the original wavelet, results in the zero-phased one from step 2.

4. Convolve this matching filter with the seismic data itself to obtain phase-shifted data.

Next, a mute was also calculated to remove the direct arrival from the input data, using a simple formula relating the source-receiver offset and seismic velocity in the water body.

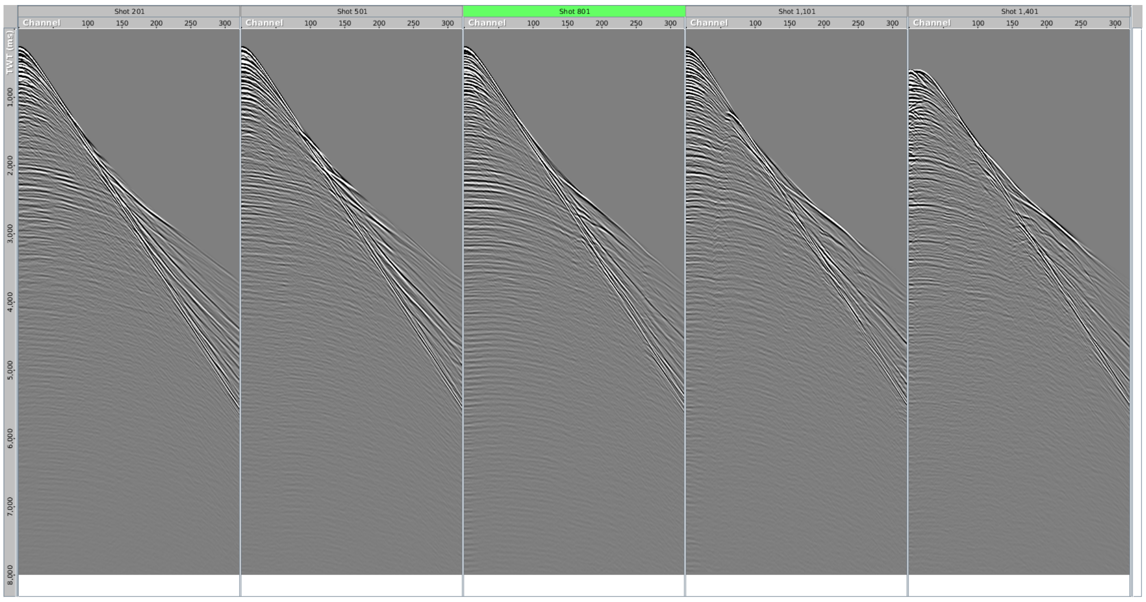
This signal does not map to reflection events, and only introduces shallow artefacts into the migrated result. Figure 5.10 shows some shot records of the data after all of these steps.

Finally, a Surface-related multiple elimination (SRME) (Verschuur et al., 1992) process was applied to the data. This involves a sophisticated convolution of the data with itself, in two dimensions, to obtain a synthetic model of reflection events that are present in the data only due to the reflectivity of the sea surface. These are then adaptively subtracted away from the data using a sliding window matching-filter procedure. An image of some shot records before and after this step is shown in Figure 5.11.

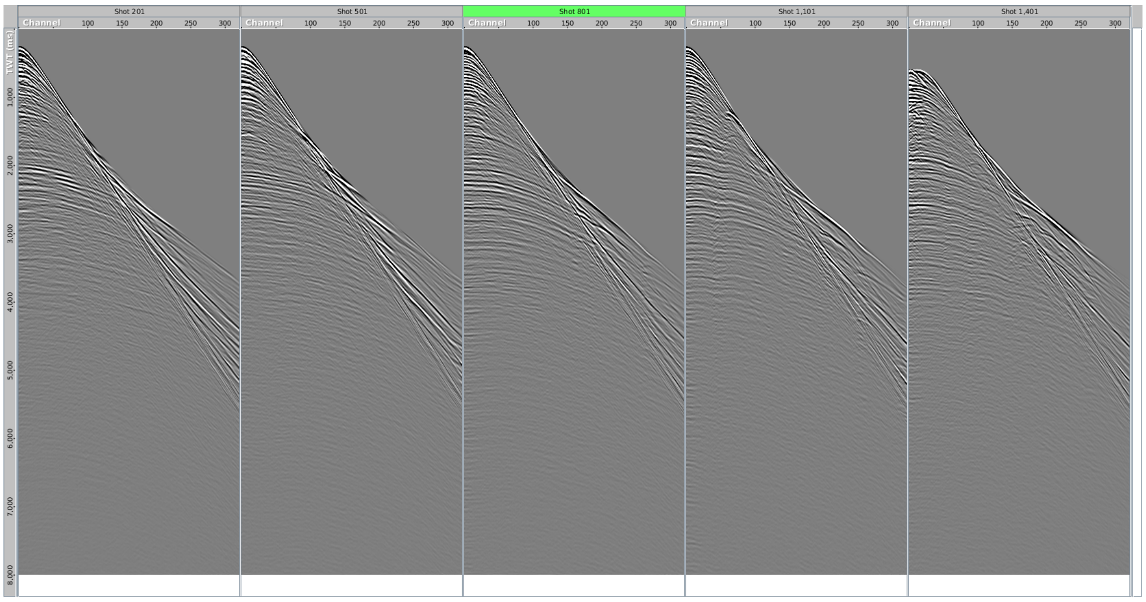
Having prepared the data, the next step is to run 2D Prestack Kirchhoff Depth Migration (KDM) to generate an image of the reflectivity in the subsurface. In this workflow, a separate reflectivity image is generated for each of many bands of surface offset, and then *common image point gathers* are constructed. These gathers, shown in Figure 5.12, show the reflectivity at each location from the whole range of available surface offsets. Since the true positions of reflectors should not change depending on the positions of the sources and receivers, the relative variation of apparent reflector depth vs surface offset gives us an indication of the errors in the velocity model used for Kirchhoff migration.

Then, a proprietary automatic interpretation algorithm from Downunder Geosolutions (DUG) is used to pick the variation of apparent reflector depth vs surface offset. Figure 5.13 shows a zoomed-in view of some of the offset gathers, overlaid with some of the output from this algorithm. These lines are referred to as *moveout picks*, and they form the input to step 3 of the reflection tomography workflow: updating the migration velocity model with the aim of flattening them. Obviously, the data is noisy and still contaminated with some multiply-reflected energy, which can cause the picking algorithm to make occasional bad picks. However, the tomography algorithm itself is relatively robust to small numbers of these.

This 3rd part of the workflow - transforming the moveout picks into a velocity update - is also a proprietary algorithm from DUG, and so I am unable to discuss it here. However, the velocity update resulting from the first tomography iteration is shown in Figure 5.14, alongside the best FWI result (iteration 73) from above. The reflection tomography

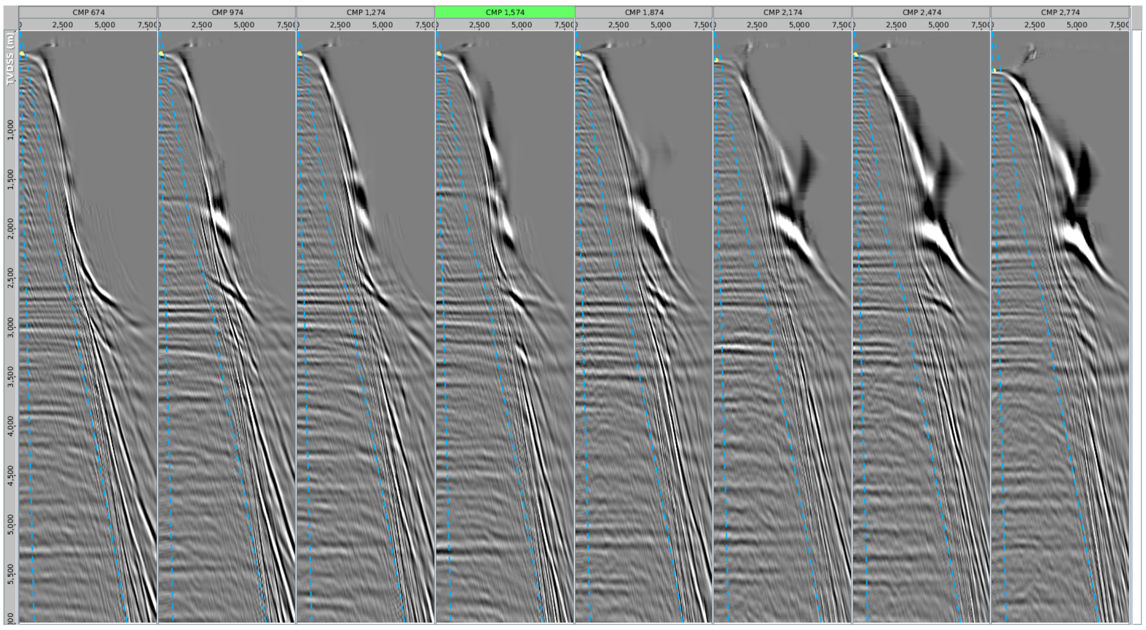


(a) Shot records before SRME.

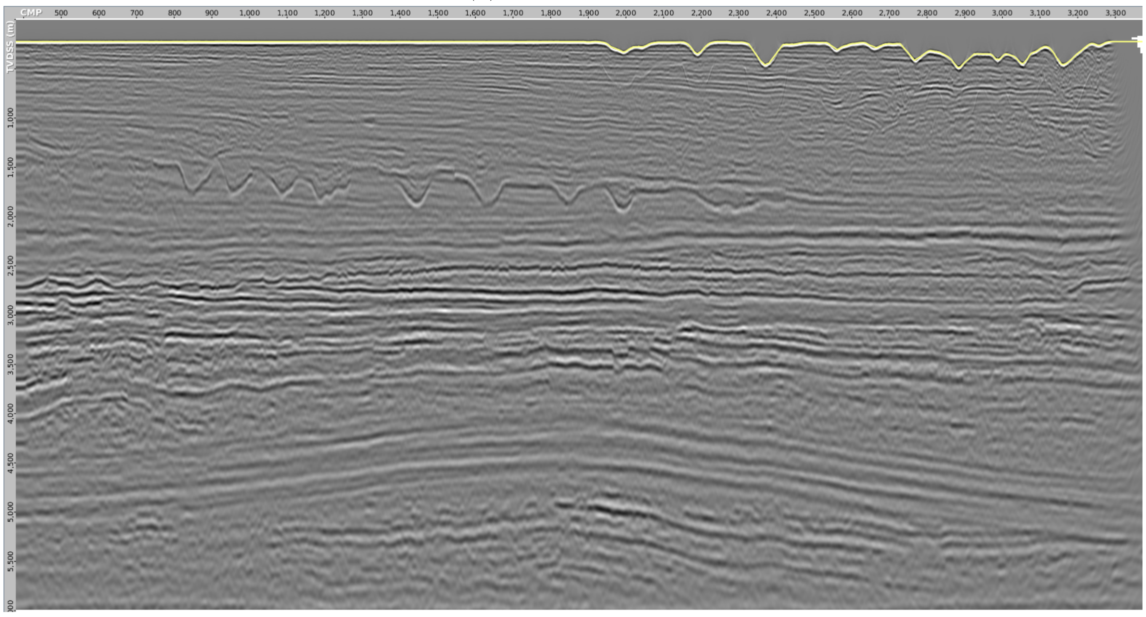


(b) Shot records after SRME.

Figure 5.11: Step 5: Surface-related Multiple Elimination (SRME). The removal of the first water-bottom multiple is most clearly visible in the centre record.



(a) Offset Gathers



(b) Stack 5-35 degrees

Figure 5.12: Kirchhoff Prestack Depth Migration, using the starting velocity model (smoothed output from AWI).

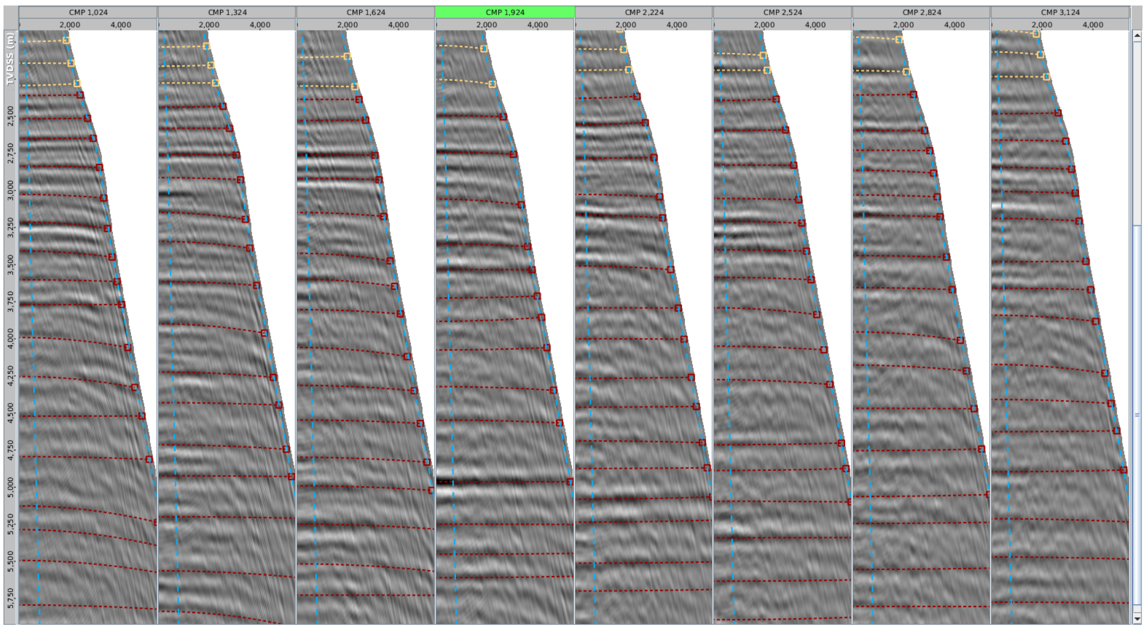
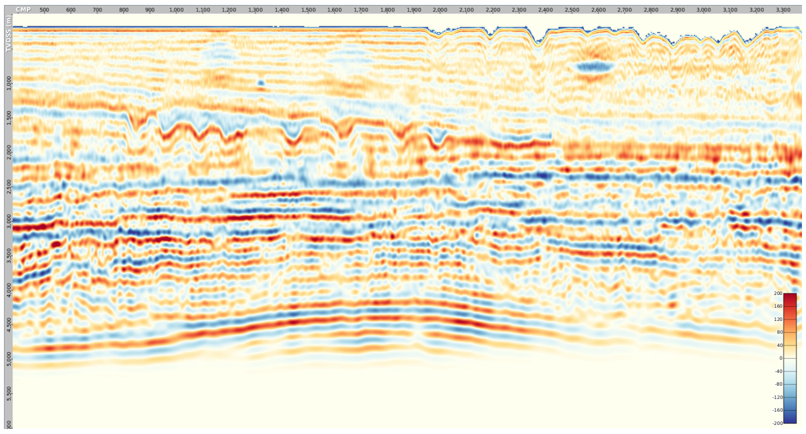
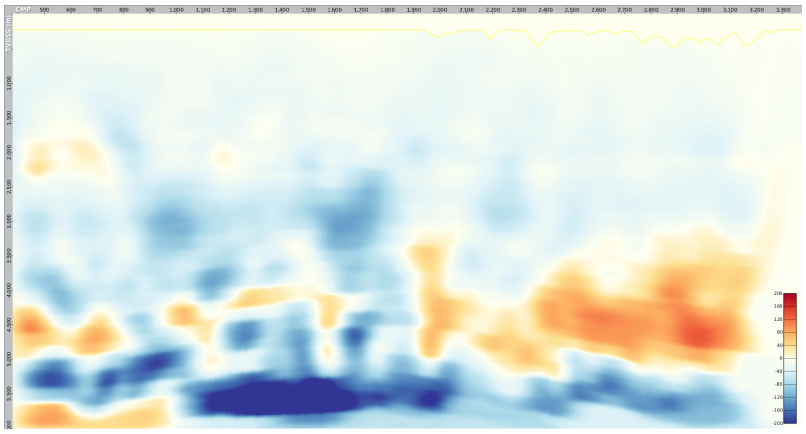


Figure 5.13: A zoomed-in view of the deeper part of the offset gathers for the migration of Figure 5.12, showing the residual moveout (RMO) picks.

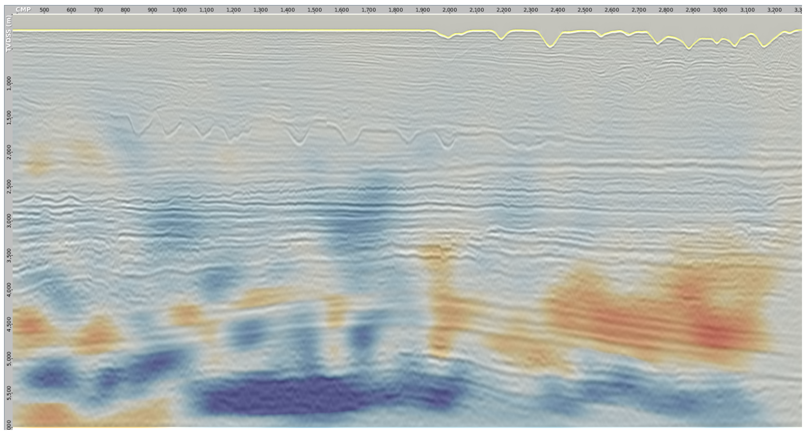
result, while lacking the high-wavenumber detail of the FWI output, has made substantial corrections to the deeper low-wavenumber components. Figures 5.15 and 5.16 show some of the migrated offset gathers in the deeper section, both before and after the tomography update. Figures 5.17 and 5.18 show a comparison of the stack. This shows that there is clearly sufficient information in the input data to update these deep, low-wavenumber velocity components, and strongly suggests that the reason for FWI's relative failure to do so must be algorithmic rather than a fundamental limitation.



(a) Velocity update ($c_{\text{after}} - c_{\text{before}}$) from 73 iterations of FWI, as described in the previous chapter



(b) Velocity update ($c_{\text{after}} - c_{\text{before}}$) from first iteration of reflection tomography



(c) Velocity update ($c_{\text{after}} - c_{\text{before}}$) from first iteration of reflection tomography, overlaid with 5-35 degree stack from Figure 5.12

Figure 5.14: Comparison of velocity updates from FWI (top) with those from reflection tomography (center and bottom). The tomography is able to make significant low-wavenumber updates in the deeper region, while FWI is only imprinting reflectors into the model.

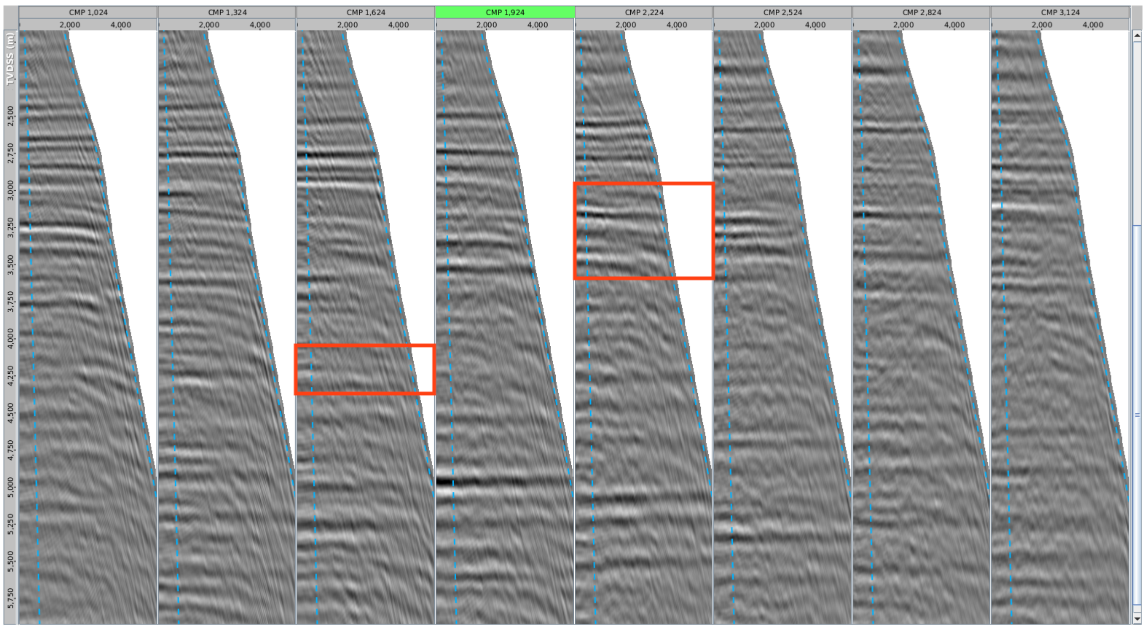


Figure 5.15: Kirchhoff-migrated offset gathers before tomography update

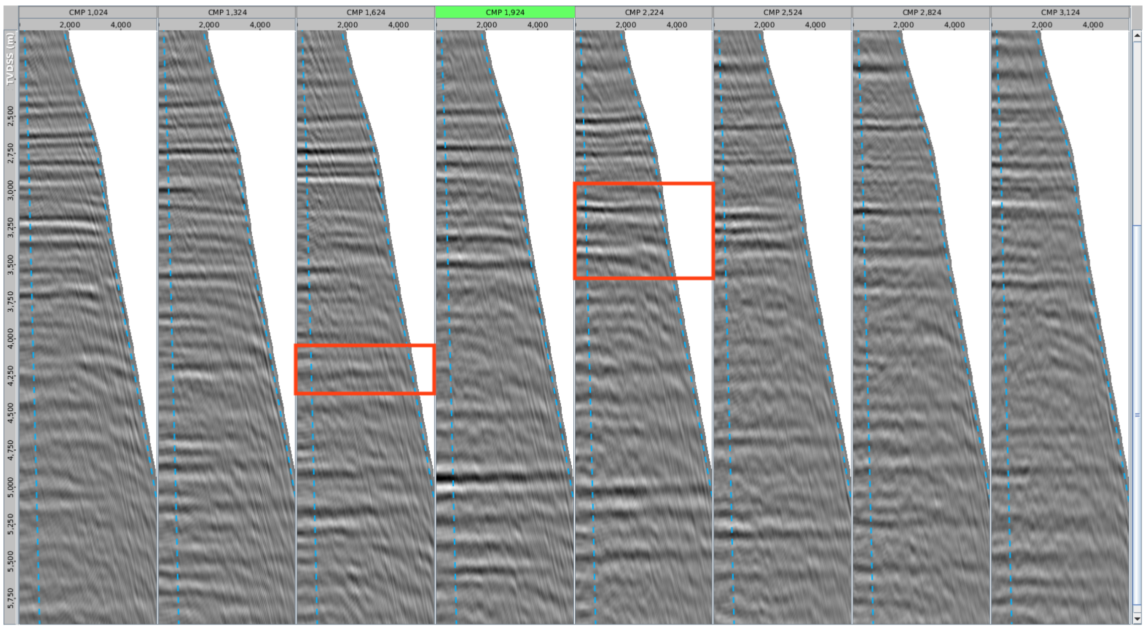


Figure 5.16: Kirchhoff-migrated offset gathers after tomography update

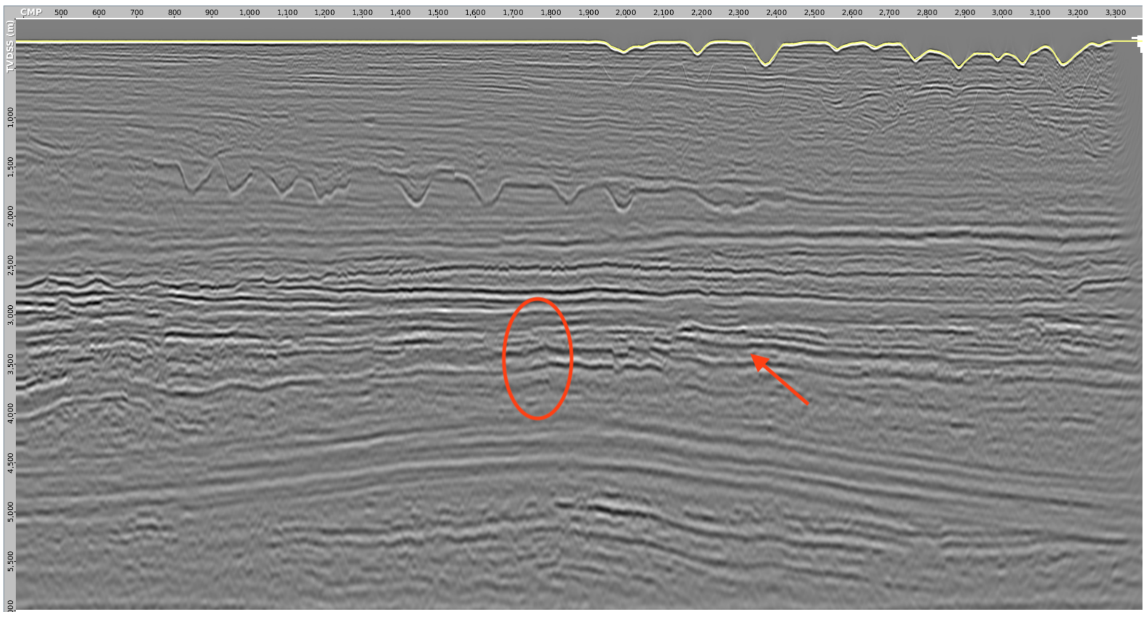


Figure 5.17: Kirchhoff-migrated stack before tomography update

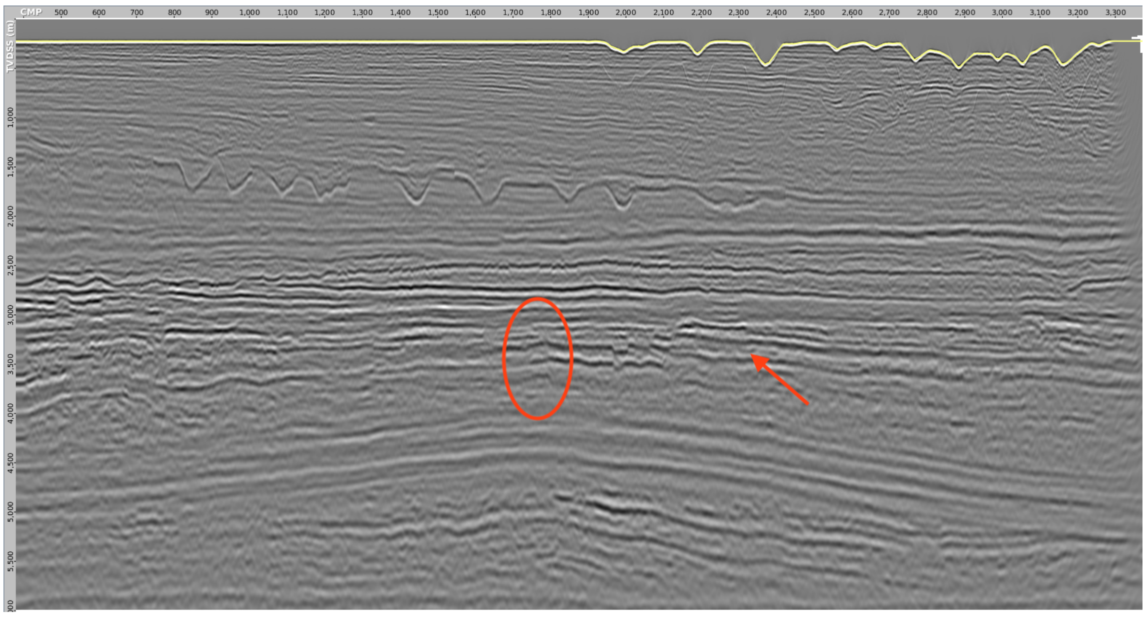


Figure 5.18: Kirchhoff-migrated stack after tomography update

5.4 A simpler problem

To begin to understand some of the causes of this effect, I study a very simple problem: FWI in a model which has no lateral variation. I will construct this model by extracting a trace from near the centre of the result in Figure 5.5 above, and generating synthetic data with an isotropic, constant-density, acoustic propagator. Note that while the velocity does not vary laterally, and can thus be represented by a one-dimensional column of values, the modelling itself is done in a two-dimensional domain in order to allow for nonzero offsets between source and receivers. The source signature used when generating the synthetic data is identical to that used in all the blind test inversions so far, as is the mute applied to avoid considering boundary reflections when matching the data. A plot and density display of the 1-D model are shown in Figure 5.19.

Just like the velocity model, the towed-streamer acquisition geometry used for this one-dimensional synthetic will be based on that from the Chevron/SEG 2014 dataset, as shown in Figure 5.20. However, this synthetic dataset has two important additional properties. Firstly, the shot spacing is taken to be equal to the grid spacing used for the modelling. Since the velocity varies only with depth, this means that the data from every shot is identical, with the consequence that only one wave equation solve is needed to generate all the synthetic data. Secondly, I assume that the sail line is infinitely long. This means that although the gradient for a single shot exhibits lateral variation (having being generated in a two-dimensional domain), the sum over shots removes that variation and allows the total gradient, and therefore the model for later iterations, to also be represented as a single column of values.

The astute reader will have recognised that I am now committing what is traditionally referred to as an *inversion crime.* The synthetic data used in this case was generated with the same code, with the same acoustic isotropic constant-density wave equation, on the same grid, with the same time spacing and boundary conditions as are being used to invert it. It is theoretically possible, therefore, that the observed and predicted data can be made to match exactly in this case - something that is essentially never true on field data. The perturbations are also too small to have induced any cycle-skipping effects. This is intentional: the aim of this section is not to demonstrate a scheme which is able to resolve deeper macromodel updates in field data, but rather to explore why the existing techniques fail to do so in even the most trivially simple case.

To show that this is the case, I performed a one-dimensional analogue of the same checkerboard test above. Using the same perturbation formula as above (equation 5.1), this time applied to the (now known) true model, FWI iterations were performed to minimise the functional. Since in this synthetic test the true model is known, only one checkerboard needs to be iterated. Furthermore, since this problem is so simple, I can afford to perform a far greater number of iterations than would ordinarily be economical in an FWI context. The difference at the starting model, as well as 50, 800 and 3600 iterations are shown in Figure 5.21.

The figure exhibits many, but not all, of the same properties seen when performing FWI on the full 2D problem. In particular, the shallow part of the model - down to about 2km - converges very quickly, while even after thousands of iterations the error in the region below 3km has yet to reduce by half. As in the 2D case earlier, the errors near interfaces are reduced quite quickly even in the deeper region, but the low-wavenumber macromodel errors are not. In contrast to the earlier experiment, however, the errors do not appear to grow in an unstable fashion. I believe this is because the starting model is now extremely close to the true model, and the functional surface - while still extremely ill-conditioned - is convex in all directions.

What these experiments show is that even in a perfect, non-cycle-skipped, inversion crime scenario, recovery of the macromodel below the region sampled by turning rays is not economically viable with the simple diagonally-preconditioned FWI algorithm being employed here, and in FULLWAVE3D. Any potential solution to the so-called “reflection FWI” problem should at least be able to tackle this case.

More insight into this problem can be obtained by examining the synthetic shot records themselves. Since these velocity differences are quite small, visually comparing the records in a printed document will not illustrate the differences in the shot records. Instead, the synthetic data from the true model (the “observed” data in this context) is shown in Figure

5.22, along with the residuals both at the start of the inversion and after 3600 iterations. The evolution of the functional - for best viewing shown on a log plot - and the RMS model error is also shown in Figure 5.23.

As can be surmised from the very large (nearly 100,000x) drop in the functional value, the difference between the observed and predicted data after 3600 iterations has been reduced dramatically - note the 10x boost in colour scale required to make the residual visible. However, as shown in Figure 5.21, significant differences to the true velocity model remain in the deeper section.

Remembering from our earlier experiments with tomography that the macromodel information is clearly present in the data, there are at least two (not mutually exclusive) explanations for this:

1. That the changes in slope of reflection events below a macromodel shift - the effect reflection tomography is effectively setup to track - only have a very small effect on the FWI functional compared to other model changes.

2. These changes in slope can also be generated, or at least well-approximated, by other - possibly more complex - model changes, and FWI is prone to generating these other less-desirable updates.

Given that FWI is typically performed using relatively low-frequency data, meaning that slight time shifts in otherwise well-matched data do not generate significant differences between signals compared to amplitude effects, I think the first explanation is especially believable. This low-frequency effect also could explain why both reflection tomography and AWI (Guasch and Warner, 2014) solve this problem better - because they both effectively involve a deconvolution step which can magnify the differences between low-frequency signals with small time shifts.

Given the dramatic reduction in the amplitudes of the residuals in the experiment above, the second observation also appears to have merit. This may indicate the need to use some kind of regularisation scheme (e.g. Asnaashari et al. (2013); Guitton (2011); Esser et al. (2016)) when performing FWI, to discourage creation of these spurious alternative structures.

To study the convergence and sensitivity characteristics of small problems like this, especially where the true model is known, it is interesting to look at the local behaviour of the functional near the true model. As discussed in the previous chapter, this information is captured in the matrix of second derivatives of the functional, i.e. the Hessian matrix **H**.

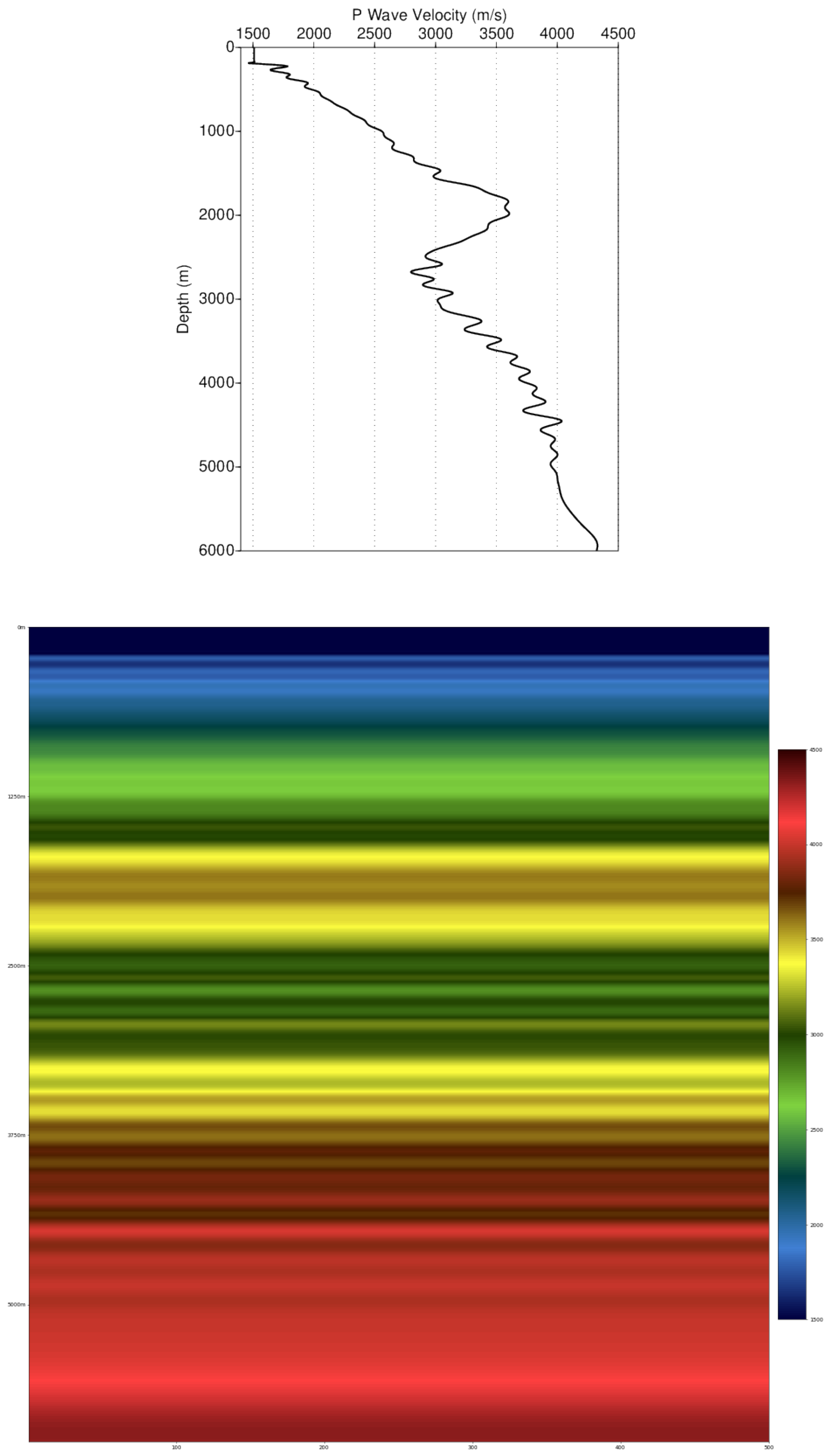


Figure 5.19: The one-dimensional velocity model used in the inversions for the remainder of this chapter, as a graph and also on the colourscale used throughout.

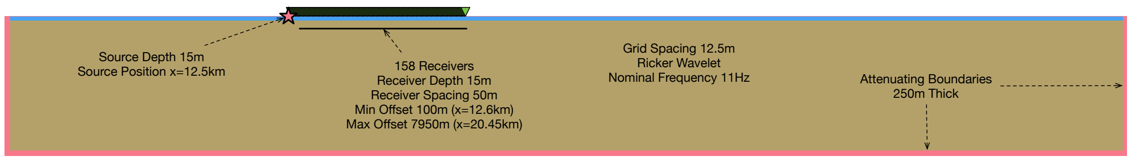
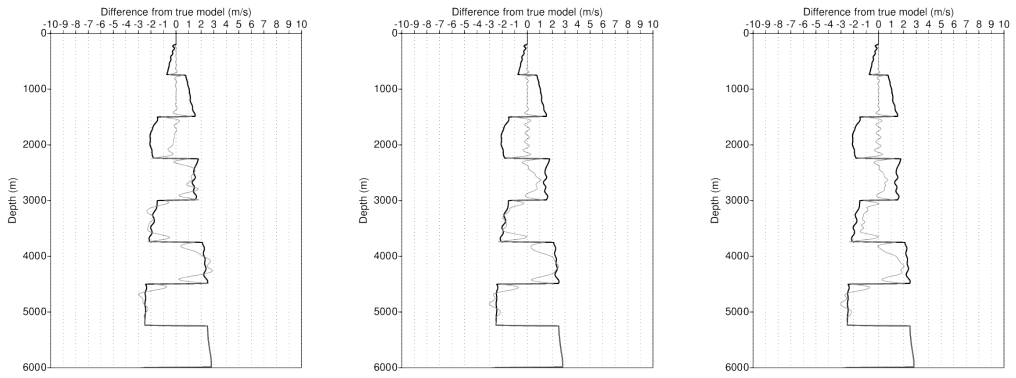


Figure 5.20: Acquisition geometry and experimental setup for the one-dimensional experiments in this chapter. The position of the shot is chosen to be far enough from the left and right edges of the domain so that reflections from the imperfect absorbing boundaries cannot reach the receiver array during the simulation time.



(a) After 50 iterations (b) After 800 iterations (c) After 3600 iterations

Figure 5.21: Results of inversion in the one-dimensional model of Figure 5.19, with a starting model constructed by adding a small perturbation to the true model. The figures show the initial perturbation from the true model (black line), and the perturbation after 50, 800 and 3600 iterations (grey line). Note that, just as in the 2D case, the high-wavenumber components of deep reflectors are recovered, but the deeper macromodel is only very slightly updated even after this extremely large number of iterations.

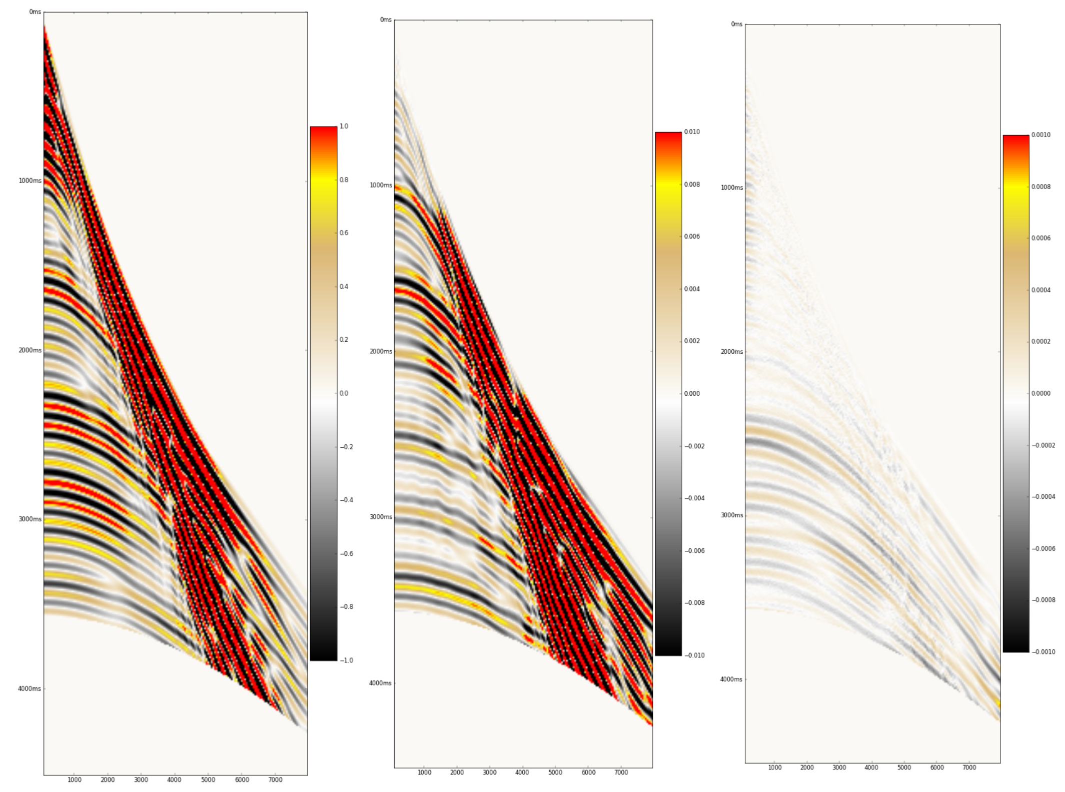


Figure 5.22: The data used in the one-dimensional inversions of this chapter, the residual at the start of the inversion and the corresponding residual after 3600 FWI iterations. Note the significantly (10x) amplified colourscale for the last figure relative to the center, required to make the residual visible. Even though the model match is relatively poor in the deeper region, the residual is still very small, as also reflected in the vastly-reduced functional values shown in Figure 5.23.

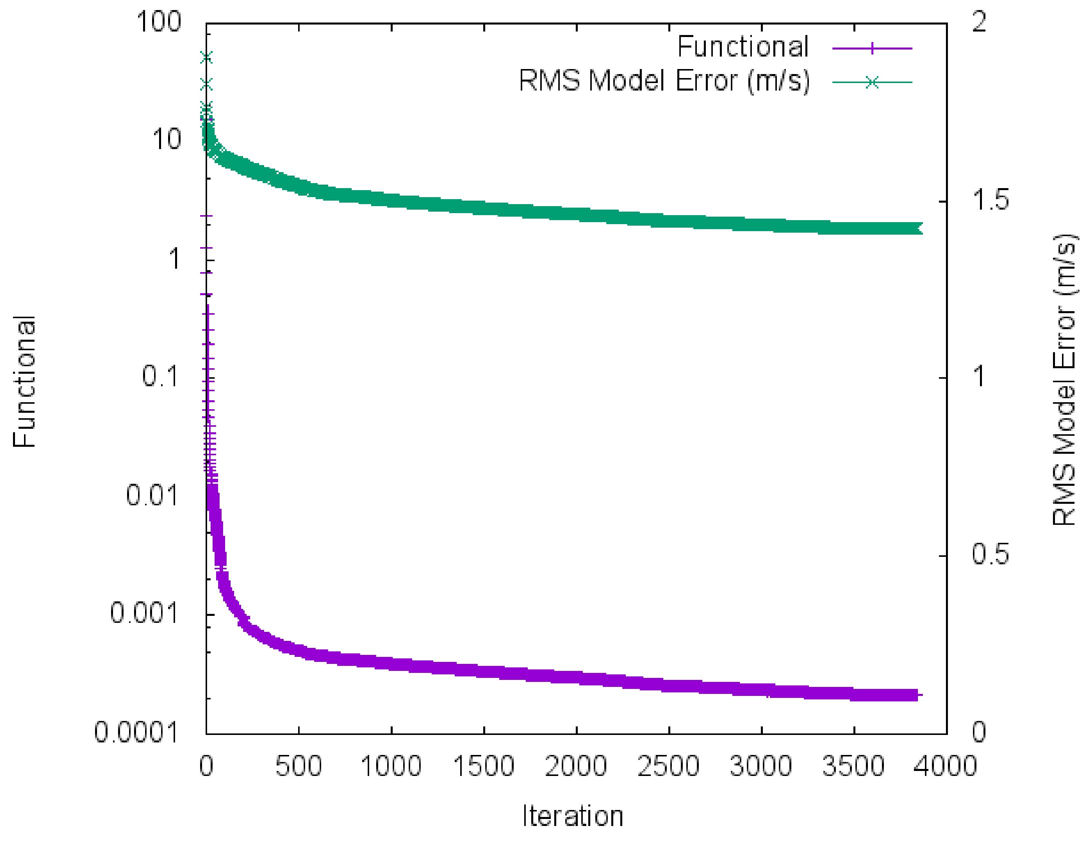


Figure 5.23: Evolution of the functional (left axis, log scale) and RMS model error (right axis) in 3600 iterations of FWI corresponding to the small starting perturbation illustrated in Figure 5.21. Note that while the functional has become vanishingly tiny, the model error has only been improved by around 25%. This is characteristic of poorly-conditioned optimisation problems such as FWI.

5.5 Hessian of a simple FWI problem

Revisiting the discussion from Chapter 4, the elements of the Hessian matrix and its decomposition into two terms for least squares problems is expressed as:

$$\begin{aligned} H_{ij} &= \frac{\partial^2 f}{\partial m_i \partial m_j} = \frac{\partial}{\partial m_i}\left(\mathbf{r}^T \frac{\partial \mathbf{r}}{\partial m_j}\right) = \frac{\partial \mathbf{r}^T}{\partial m_i}\frac{\partial \mathbf{r}}{\partial m_j} + \mathbf{r}^T \frac{\partial^2 \mathbf{r}}{\partial m_i \partial m_j} \\ \mathbf{H} &= \mathbf{J}^T\mathbf{J} + \mathbf{r}^T \frac{\partial^2 \mathbf{r}}{\partial m_i \partial m_j} \end{aligned} \tag{5.3}$$

where $\mathbf{J}^T\mathbf{J}$ is often referred to the "Gauss-Newton Hessian", because of its connection to the optimisation method.

Of course, since the number of model parameters is large, it is not practical to use this expression as a way of obtaining values for the elements H_{ij}. Instead, using the same adjoint-state trick that makes FWI possible in the first place, one can rapidly compute so-called Hessian-vector products $\mathbf{Hv}$ for arbitrary vectors $\mathbf{v}$ - in this case, products with the elementary vectors $\mathbf{e}_i$ which take the value 1 at model parameter i and 0 for all others.

Métivier et al. (2013) describes a technique for computing these $\mathbf{Hv}$ products using a second-order adjoint state method, but for the sake of simplicity I instead chose to use a finite-difference approximation, which simply perturbs the model by $\pm\epsilon\mathbf{v}$ and obtains the difference between the gradients at these two points:

$$H_{ij} = \left.\frac{\partial^2 f}{\partial m_i \partial m_j}\right|_{\mathbf{m}} \approx \frac{1}{2\Delta m_i}\left(\left.\frac{\partial f}{\partial m_j}\right|_{\mathbf{m}+\Delta m_i \mathbf{e}_i} - \left.\frac{\partial f}{\partial m_j}\right|_{\mathbf{m}-\Delta m_i \mathbf{e}_i}\right) \tag{5.4}$$

or in vector form:

$$\mathbf{He}_i \approx \frac{1}{2\Delta m_i}\left((\nabla f)(\mathbf{m} + \Delta m_i \mathbf{e}_i) - (\nabla f)(\mathbf{m} - \Delta m_i \mathbf{e}_i)\right) \tag{5.5}$$

The Gauss-Newton Hessian $\mathbf{J}^T\mathbf{J}$ can be approximated in the same way, by again perturbing each model parameter up and down, but then backpropagating the difference between the residuals through the original model, in the same manner as calculating the gradient

$\nabla f = \mathbf{J}^T\mathbf{r}$:

$$(\mathbf{J}^T\mathbf{J})\mathbf{e}_i \approx \frac{1}{2\Delta m_i}\mathbf{J}^T\left(\mathbf{r}(\mathbf{m}+\Delta m_i\mathbf{e}_i) - \mathbf{r}(\mathbf{m}-\Delta m_i\mathbf{e}_i)\right) \tag{5.6}$$

When evaluated at the true model of an inverse crime problem, of course, $\mathbf{r} = \mathbf{0}$ and so from equation (5.3), $\mathbf{H} = \mathbf{J}^T\mathbf{J}$.

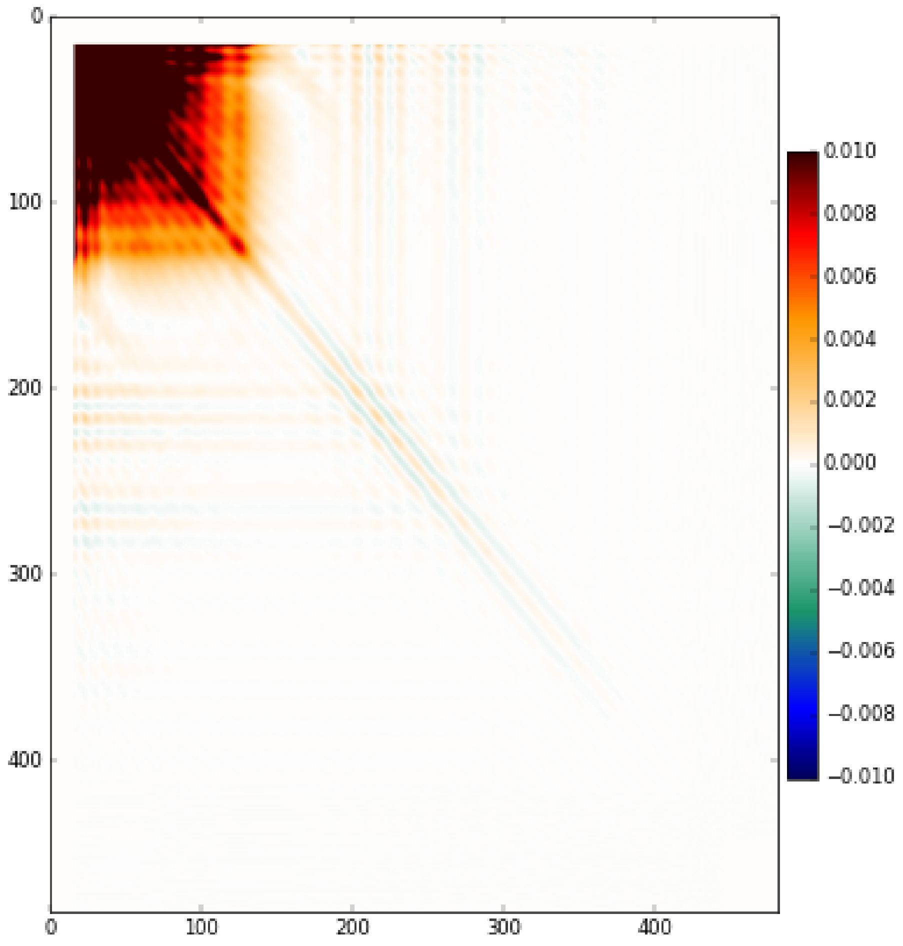


Figure 5.24: A visualisation of the Hessian matrix of the one-dimensional FWI problem, showing the second derivative of the functional $\partial^2 f/\partial m_i \partial m_j$ for all pairs (i, j). Derivatives involving model parameters inside the waterbody are not shown. Note the extreme sensitivity of the functional to the shallow part of the model - this is why the functional in this synthetic problem can be reduced by 100,000x even though the deeper part of the model has improved very little.

Figure 5.24 shows a visualisation of the Hessian matrix at the true model. Since this is a 1D system, there are only 481 parameters, which makes the matrix practical to visualise. However, the huge range of the values in the matrix makes the visualisation difficult to interpret. Figure 5.25 shows the matrix with the rows and columns scaled by the approximate

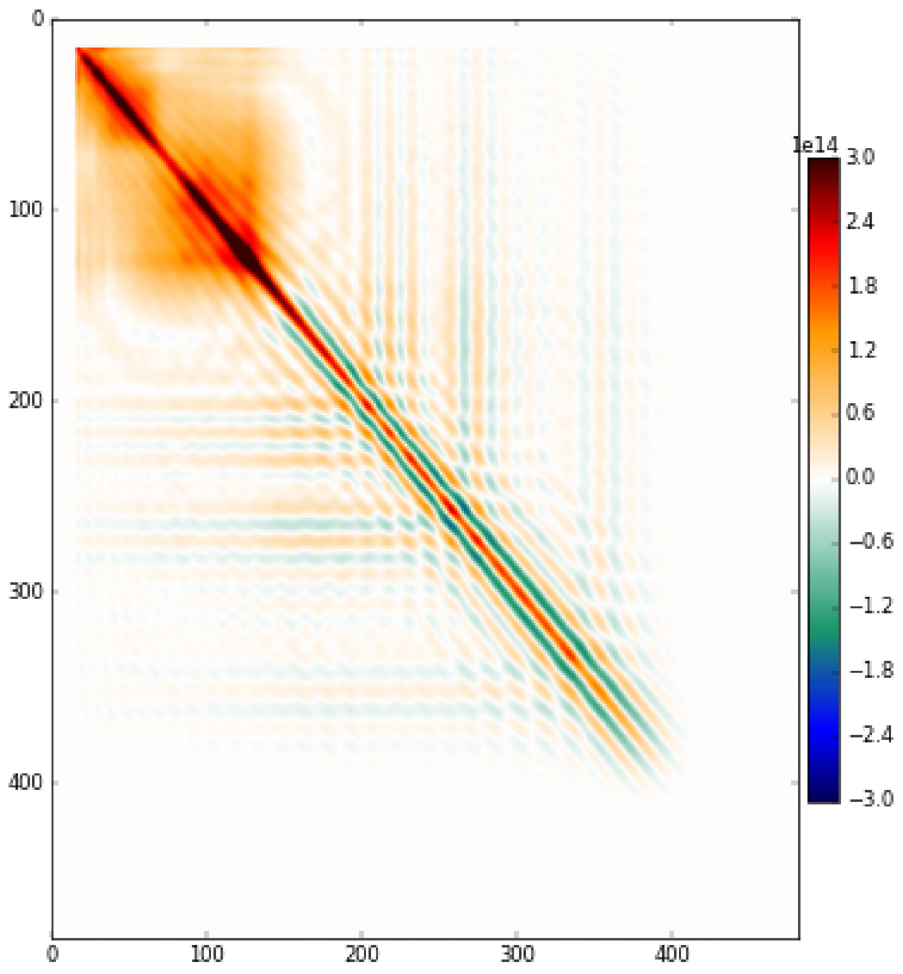


Figure 5.25: The Hessian matrix of the one-dimensional FWI problem, after applying the diagonal preconditioner derived in Chapter 4 by dividing the rows and columns by the square root of the approximate diagonal. Most important to note is the change in the nature of the Hessian below the turning-ray region.

diagonal preconditioner described in the previous chapter.

A row of this matrix describes how the gradient for all model parameters changes when the value of one such parameter is increased slightly. Positive values indicate that when the first parameter is changed, it amplifies the effect of a similarly-signed change in the other parameter, while negative values indicate that the two changes of the same sign will somewhat mitigate each other's effects. It is important that the diagonal is positive; this indicates that when a parameter is increased, the gradient in that parameter increases also - indicating upward curvature, i.e. convexity, of the functional surface.

There are several noteworthy features visible in this figure. The most obvious is the distinct change in the nature of the matrix below the top 150 or so cells - i.e. down to around 2km. This corresponds precisely to the region well-sampled by turning rays, which suggests that it might be a strong indicator of why the convergence characteristics of FWI are so different in this region compared to the deeper part. Turning rays serve to explain why the values in this block are all positive; the functional contribution from turning ray arrivals is dominated by kinematics, and increasing the value any velocity in this region will have a similar kinematic effect on many of the turning rays to increasing any other velocity. Put another way, the functional contribution from a single turning ray arrival could be expressed in terms of a weighted sum of the velocities it samples. Starting from the true model and increasing one of the velocities, the gradient will increase for all of the others as well - suggesting that they all be reduced to compensate.

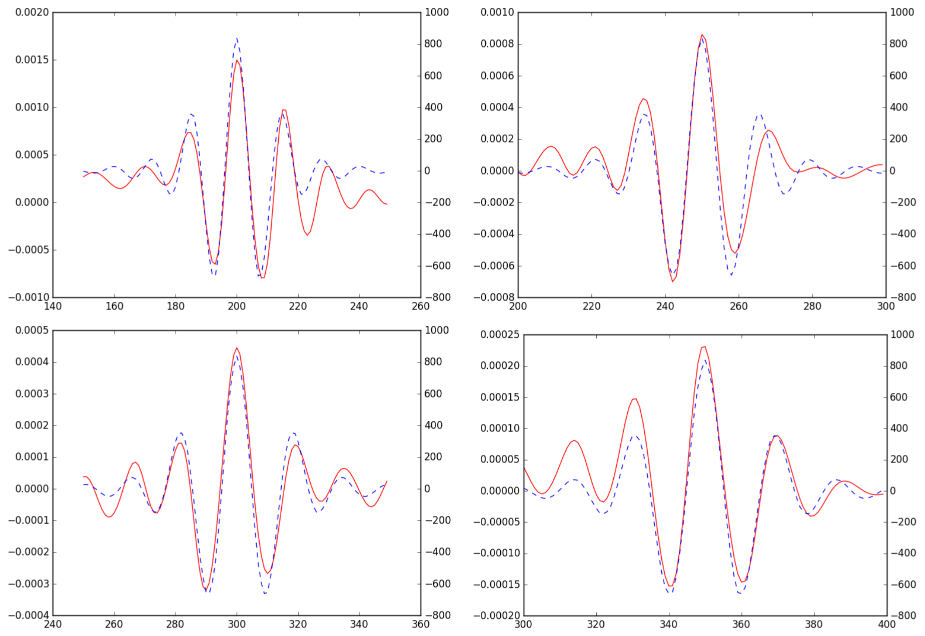


Figure 5.26: Elements of the preconditioned Hessian rows near the main diagonal (red, left axis), superimposed with a laterally-stretched autocorrelation of the source wavelet (dotted blue, right axis), for various depths. This shows that the spectrum of the wavelet, stretched into the wavenumber domain by the local velocity, defines the oscillatory behaviour of the one-dimensional Hessian around the main diagonal.

The second is the banded diagonal structure of the deep part. I would suggest that this originates from near-normal-incidence reflections. If the wavelet oscillates approximately with a period T, then two reflections arriving $0.5T$ apart will have opposite effects on the data, two reflections arriving with a spacing T will have similar effects on the data. Thus, it is not surprising to observe an oscillatory effect around the main diagonal in the region dominated by near-normal reflections. As shown in Figure 5.26, the width of the bands varies in proportion to the local velocity, and the overall shape is determined by the amplitude spectrum (i.e. autocorrelation) of the source wavelet.

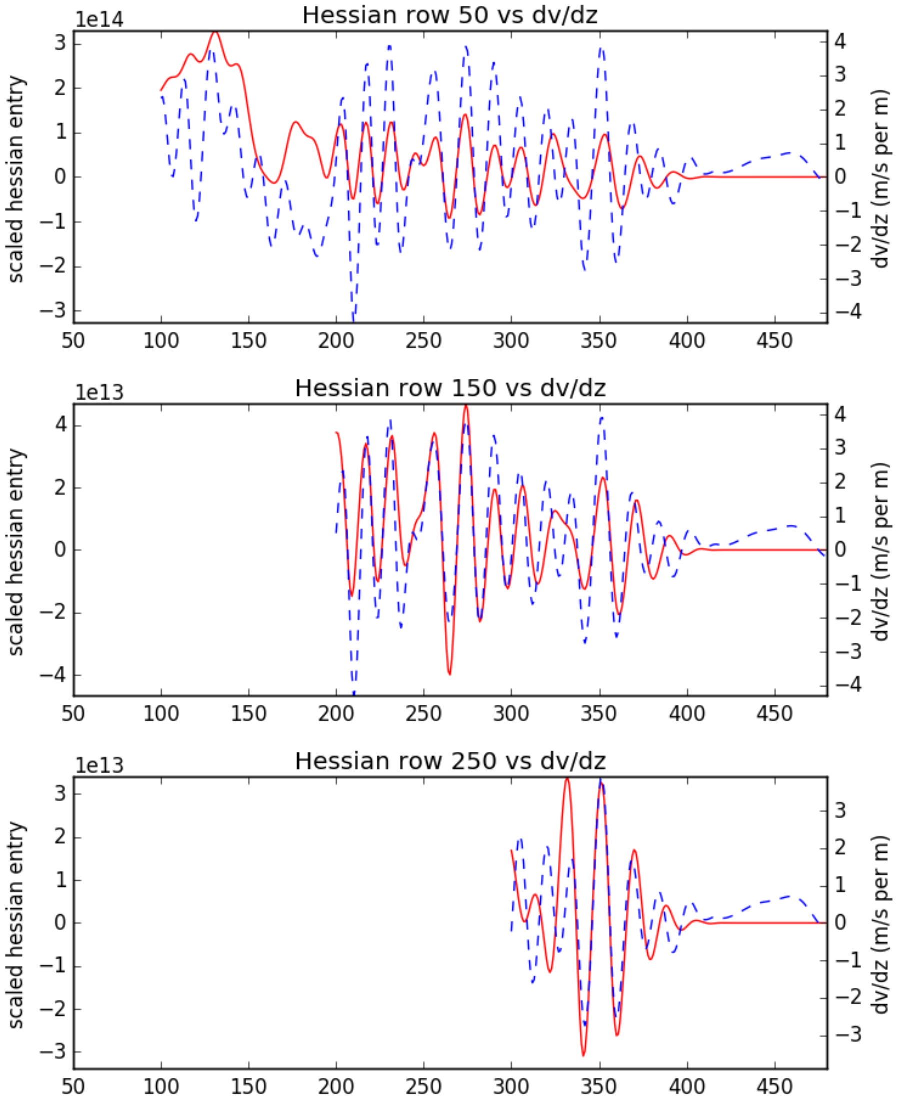


Figure 5.27: Elements of the preconditioned Hessian away from the main diagonal (red, left axis), superimposed with the vertical derivative of velocity (dotted blue, right axis). The strong correlation in the deeper region illustrates the connection between the high- and low- wavenumber components of the model in reflection data: increasing the velocity above a reflector has a similar effect to shifting the reflector upward slightly.

The third thing to note is the structure away from the diagonal in the deeper part. I suggest that this is to do with the change in apparent depth of reflection events due to velocity variation above. If a shallow velocity is increased, then reflected signals from velocity variations below it will arrive earlier. This is a similar effect to if the reflector itself had been shifted up slightly. As such, one would expect these variations to correlate with the derivative of velocity with depth $\partial v / \partial z$, and Figure 5.27 shows that this is indeed the case, though there is some consistent disagreement at around depth sample 300-330 which I cannot explain.

Finally, there is the notable absence of any sensitivity at all in the very deepest parts of the model. This is because of the conservative mute applied to the shot records, as in the earlier experiment, to prevent the numerical reflection from the imperfect boundary from being used to recover deep velocity information.

5.6 Eigendecomposition of the 1D Hessian

Having established the qualitative structure of the Hessian matrix for this simple problem, I now examine the implications this has on the convergence of the FWI problem.

Since $\mathbf{H}$ describes, to a first order approximation, how the gradient varies in the neighbourhood of the current model, it can be used to predict the behaviour of a descent-based solver such as that used in many implementations of FWI. A model of how the gradient changes in response to small model changes around a partially-converged result is often sufficient to describe the behaviour of descent for a large number of iterations. Typically, a descent iteration takes the form

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{P}_k^{-1} (\nabla f)(\mathbf{m}_k) \tag{5.7}$$

with $\mathbf{P}_k$ some kind of easily-invertible preconditioner that in general may vary from iteration to iteration, and α_k a step length, typically chosen using a line search at each iteration.

Making the first order approximation to the gradient around some model $\mathbf{m}_*$, this expression becomes:

$$\begin{aligned} \mathbf{m}_{k+1} &\approx \mathbf{m}_k - \alpha_k \mathbf{P}_k^{-1}((\nabla f)(\mathbf{m}_*) + \mathbf{H}_*(\mathbf{m}_k - \mathbf{m}_*)) \\ \mathbf{m}_{k+1} - \mathbf{m}_* &\approx -\alpha_k \mathbf{P}_k^{-1}(\nabla f)(\mathbf{m}_*) + (\mathbf{I} - \alpha_k \mathbf{P}_k^{-1}\mathbf{H}_*)(\mathbf{m}_k - \mathbf{m}_*) \end{aligned}$$

This expression is a recurrence relation describing how descent iterations behave near the point $\mathbf{m}_*$. If $\mathbf{m}_*$ is a locally optimal point, then the gradient at $\mathbf{m}_*$ is zero, leaving:

$$\mathbf{m}_{k+1} - \mathbf{m}_* \approx (\mathbf{I} - \alpha_k \mathbf{P}_k^{-1}\mathbf{H}_*)(\mathbf{m}_k - \mathbf{m}_*)$$

Assuming that the preconditioner $\mathbf{P}$ varies sufficiently slowly, the behaviour of a descent solver iterating toward this optimal point $\mathbf{m}_*$ can therefore be described by the preconditioned Hessian matrix $\mathbf{P}^{-1}\mathbf{H}_*$. In particular, considering the eigendecomposition

$\mathbf{P}^{-1}\mathbf{H}_* = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$ of the preconditioned Hessian, the convergence formula above as:

$$\mathbf{m}_{k+1} - \mathbf{m}_* \approx \mathbf{Q}(\mathbf{I} - \alpha_k \boldsymbol{\Lambda})\mathbf{Q}^{-1}(\mathbf{m}_k - \mathbf{m}_*)$$

This means that the descent iteration can be thought of as a three step process:

1. projecting the current model error into a different basis
2. multiplying each of the components by $1 - \alpha_k \lambda$, where λ is the eigenvalue for that component
3. projecting back into the original basis

It also offers a convenient form in which to express the results of a large number of iterations:

$$\mathbf{m}_{k+n} - \mathbf{m}_* \approx \mathbf{Q}\left(\prod_{i=k}^{k+n-1} (\mathbf{I} - \alpha_i \boldsymbol{\Lambda})\right)\mathbf{Q}^{-1}(\mathbf{m}_k - \mathbf{m}_*)$$

It is clear from the above that the values of α_k must be limited to ensure stability of the convergence - for a constant step size, this requirement is $\alpha\lambda_{\max} < 2$, and line searching algorithms also tend to generate sequences of α_k in this range. The consequence of this limitation of step size is that components of the model error corresponding to smaller eigenvalues converge much more slowly, with a rate roughly proportional to $\lambda/\lambda_{\max}$.

The eigendecomposition of the preconditioned Hessian is therefore useful when determining the recoverability and rate of convergence of various components of the model. It is also useful for determining the effectiveness of the preconditioning matrix $\mathbf{P}$: the number of eigenvalues with $\lambda/\lambda_{\max} > 0.01$ is a good indicator of the number of components that can be recovered within a reasonable number of iterations, i.e. several hundred.

Since the preconditioned Hessian matrix $\mathbf{P}^{-1}\mathbf{H}$ is not symmetric, the eigendecomposition is slightly more involved, and in general is complex-valued even for a matrix with only real entries. The problem is best solved, therefore, by exploiting the fact that $\mathbf{P}$, and therefore $\mathbf{P}^{-1}$, is symmetric positive definite. For general $\mathbf{P}$, obtain the Cholesky decomposition

$\mathbf{P} = \mathbf{U}^T\mathbf{U}$ and use a transformation to obtain a symmetric eigenproblem:

$$\begin{aligned}
\mathbf{P}^{-1}\mathbf{H} &= \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1} \\
\mathbf{U}^{-1}\mathbf{U}^{-T}\mathbf{H} &= \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1} \\
\mathbf{U}^{-T}\mathbf{H}\mathbf{U}^{-1} &= \mathbf{U}\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}\mathbf{U}^{-1} \\
&= (\mathbf{U}\mathbf{Q})\boldsymbol{\Lambda}(\mathbf{U}\mathbf{Q})^{-1} \\
&= (\mathbf{U}\mathbf{Q})\boldsymbol{\Lambda}(\mathbf{U}\mathbf{Q})^{T}
\end{aligned}$$

This shows that the eigenvalues of $\mathbf{P}^{-1}\mathbf{H}$ are the same as those of $\mathbf{U}^{-T}\mathbf{H}\mathbf{U}^{-1}$, which is symmetric by definition. Furthermore, the matrix $\mathbf{U}$ gives a mapping between between the two sets of eigenvectors: if $\mathbf{U}^{-T}\mathbf{H}\mathbf{U}^{-1} = \mathbf{Q}'\boldsymbol{\Lambda}\mathbf{Q}'^{T}$, then $\mathbf{Q} = \mathbf{U}^{-1}\mathbf{Q}'$ and the inverse $\mathbf{Q}^{-1} = \mathbf{Q}'^{T}\mathbf{U}$. Of course, in the common case that $\mathbf{P}$ is diagonal then the Cholesky factor $\mathbf{U}$ is simply the element-wise square root.

Figure 5.28 shows a representation of the eigendecomposition of the diagonally-preconditioned Hessian for our 1D problem (the original matrix is shown in Figure 5.25). The first thing to note is the *huge* range of eigenvalues. As shown above, the rate of convergence is proportional to the ratio $\lambda/\lambda_{\max}$, so this suggests that the vast majority of the components will converge so slowly as to be effectively unrecoverable. This range is so large that the condition number cannot be estimated: the numerical errors present in obtaining the Hessian matrix itself make determination of the smallest eigenvalues impossible.

The second thing to note is the presence of negative eigenvalues - denoted with the dotted line since negatives cannot be represented on a conventional logarithmic scale. These represent negative, i.e. downward, curvature in the functional surface, and are not theoretically possible since we are at the true model, and $\mathbf{H} = \mathbf{J}^T\mathbf{J}$, which is positive semidefinite. I attribute this partially to numerical imprecision - the values are 10^{-8} and the wavefield propagations (not the eigendecomposition) were computed in single-precision arithmetic, which only permits accuracy to around 10^{-7}. It is also possible that these originate from approximation error in computing the transpose/adjoint operator of the wave

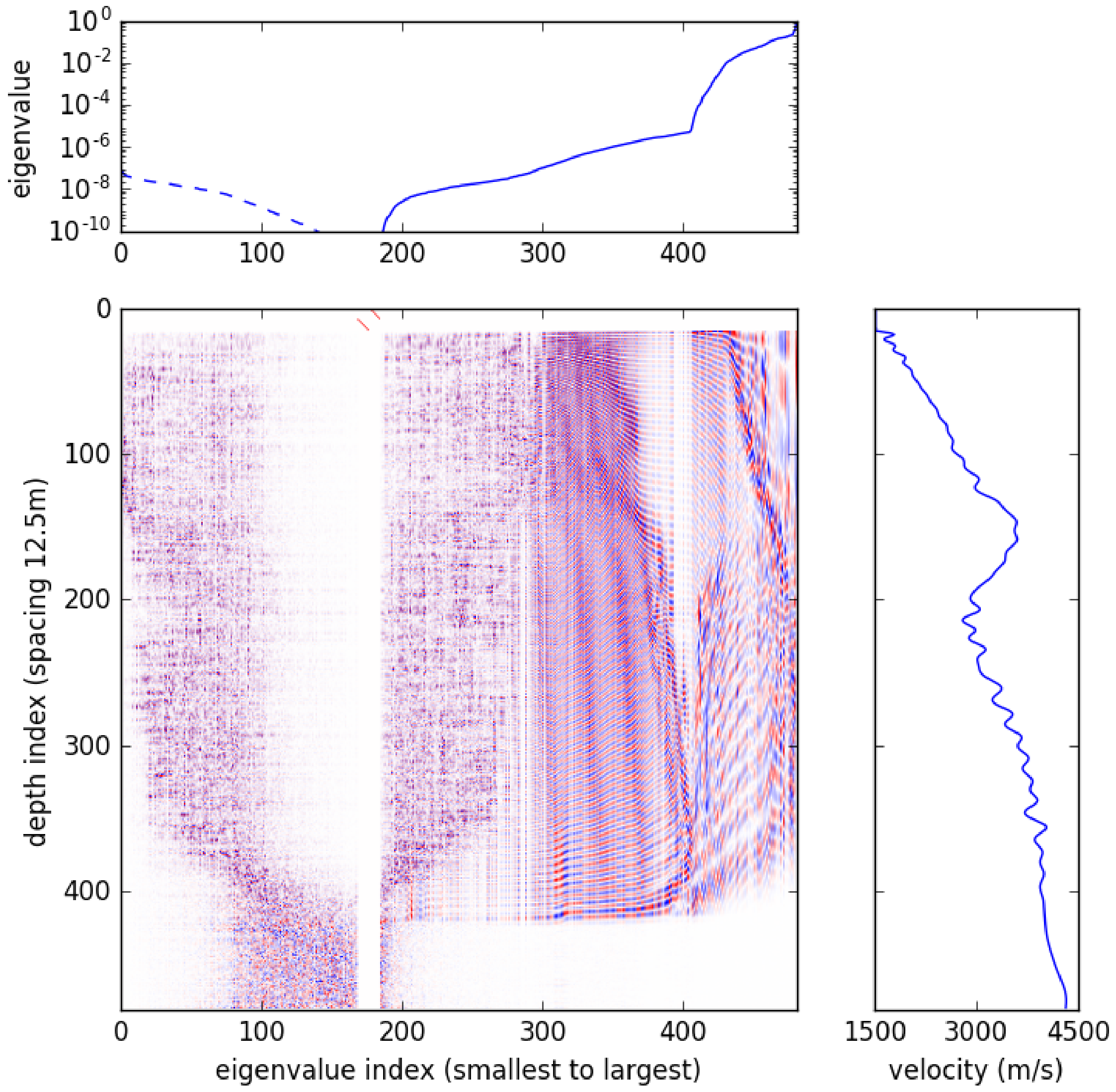


Figure 5.28: The eigendecomposition of the diagonally-preconditioned Hessian matrix shown in Figure 5.25, aligned with the velocity profile (right). Each eigenvector corresponds to a possible update direction for the model, with the corresponding eigenvalue indicating the sensitivity of the functional to model changes in that direction. The eigenvalues themselves are shown on a log scale, with the dotted line indicating negative values. This matrix should not theoretically have any negative eigenvalues, so these are to be considered noise - artefacts of the imprecise algorithm used to obtain the matrix.

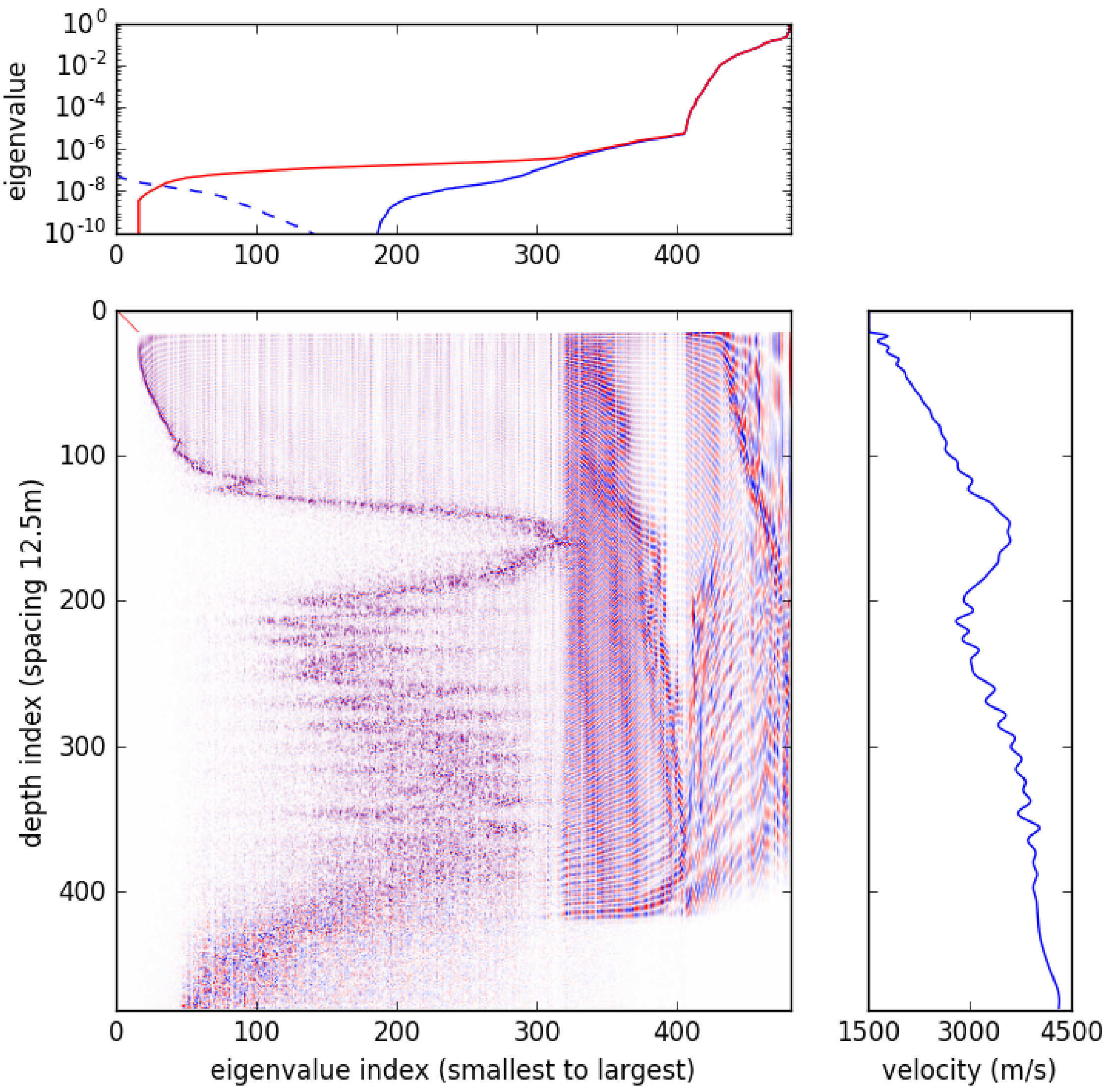


Figure 5.29: The eigendecomposition of the diagonally-preconditioned Hessian matrix obtained by constructing an approximation of $\mathbf{J}$ and then computing $\mathbf{J}^T\mathbf{J}$, thus eliminating the possibility of negative eigenvalues. Eigenvalues from the more precise technique are shown on the red line, with the eigenvalues from Figure 5.28 shown in blue to illustrate the match in the larger values. This suggests that while the approximation used to obtain the original matrix can lead to errors, they do not affect the dominant part of the eigenvalue spectrum. I believe that the velocity-profile-like behaviour in the eigenvectors on the left part of the figure is due to the scaling of random numerical noise by the diagonal preconditioner before the decomposition is performed.

propagation. To illustrate that the dominant part of the spectrum is not affected by these precision issues, Figure 5.29 shows the eigendecomposition of the matrix $\mathbf{J}^T\mathbf{J}$ computed by explicit computation of the Jacobian matrix $\mathbf{J}$. The eigenvalue spectrum and corresponding eigenvectors match quite precisely down to about $\lambda/\lambda_{\max} = 10^{-6}$.

To see the implications for recoverability of deeper macromodel components, Figure 5.30 shows a zoomed onto the top 130 or so eigenvalues, where the numerical errors are negligible. The ellipsed regions show eigenvectors containing low-wavenumber components in the turning ray region, and below. The deeper macromodel updates are associated with normalised eigenvalues of around $\lambda/\lambda_{\max} = 10^{-4}$, suggesting that several thousand iterations would be required to recover them - which fits well with observations from my earlier run, showing marginal improvements after 3600 iterations. To show this more clearly, Figure 5.31 shows how the checkerboard from that earlier inversion is recovered as the contribution from each eigenvector is accounted for.

This picture is more encouraging than the initial decomposition - it shows that even though the vast majority of directions in the model space are within the numerical noise and cannot be recovered, the components we are currently interested in are more accurately described as uneconomical rather than impossible. The remaining, likely unrecoverable, components tend to describe very high wavenumber updates which are effectively invisible due to the comparatively low acoustic frequencies (only up to 11Hz) present in the source wavelet and filtered data. This can be seen in the left hand sections of the eigendecompositions in figures 5.28 and 5.29.

5.7 Using the full Hessian for preconditioning

Having obtained the Hessian matrix for this 1D problem, I will now illustrate the potential for second order methods that make use of such information to greatly improve the recovery of deeper macromodel information.

Newton's method is the canonical example of such a method, and computes updates at

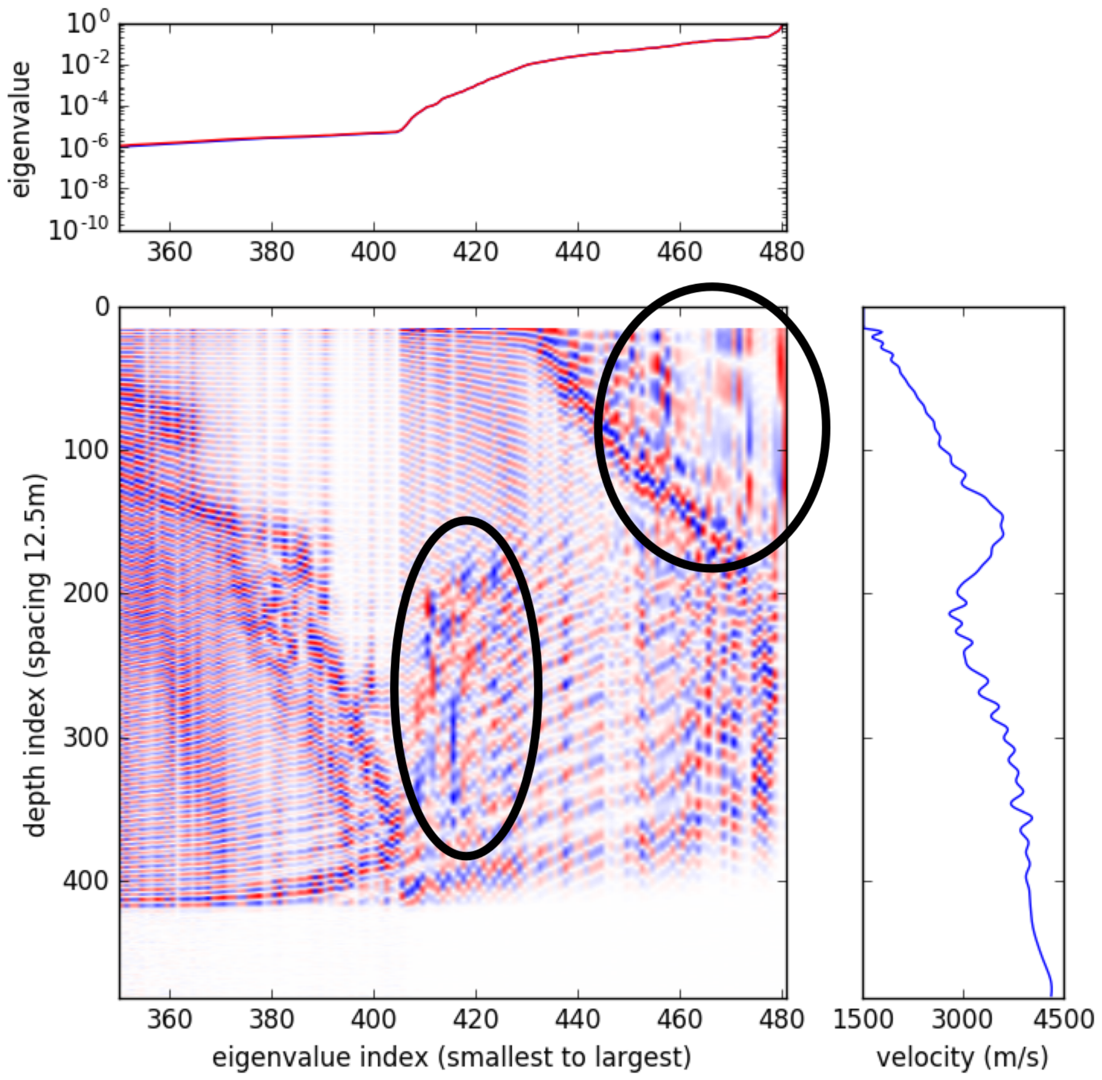


Figure 5.30: A zoomed-in section of Figure 5.28, showing the ranges of eigenvectors corresponding to macromodel updates in the shallow and deeper sections.

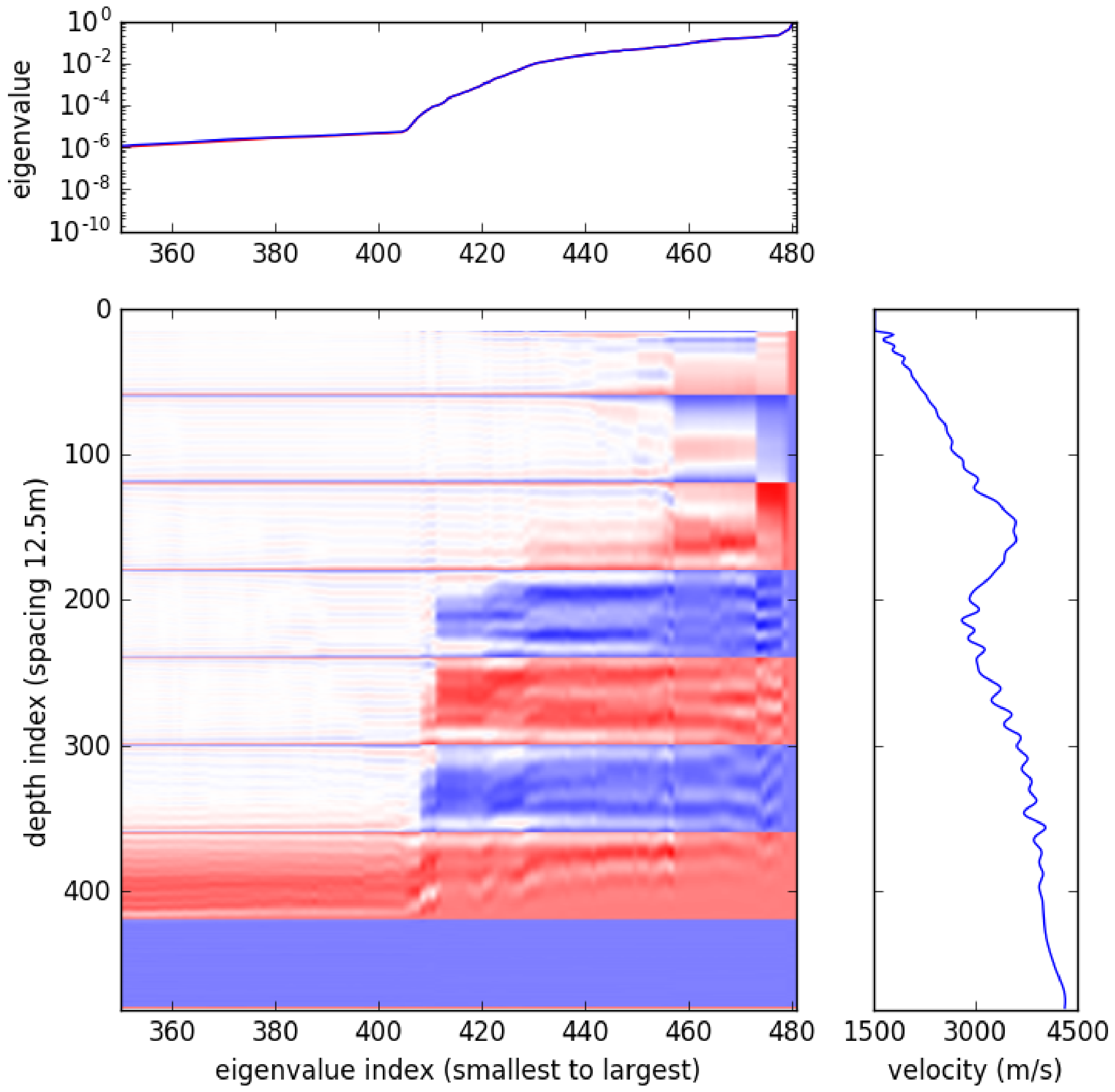


Figure 5.31: The projection of an oscillating 1D perturbation onto increasingly large sets of eigenvectors, showing which eigenvector components must be recovered in order to make significant macromodel updates to various sections of the model. Moving from the right, it shows how much of the model error is removed if increasing numbers of eigenvectors were able to be recovered. The components we are interested in - low wavenumber updates below depth index around 200 - have normalised eigenvalues of around $\lambda/\lambda_{\text{max}} = 10^{-4}$, suggesting that several thousand iterations would be required to recover them to any degree. This is in agreement with the earlier observations in Figure 5.21.

each iteration by solving a linear system involving the Hessian matrix:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \mathbf{H}^{-1}(\nabla f)(\mathbf{m}_k)$$

Comparing this with equation (5.7), it can be seen that this is equivalent to using the Hessian matrix itself as the preconditioner $\mathbf{P}$, and using a step length of $\alpha = 1$. This presents three serious problems in the context of FWI.

1. To go downhill, we require that the preconditioning matrix $\mathbf{P}$ be positive-definite. Since the functional surface in FWI is not convex, this is not always true. Newton's method will therefore sometimes take us 'uphill'.

2. Even when matrix $\mathbf{H}$ does not have negative eigenvalues, as in the examples above when it is computed at the true model, the large number of eigenvalues that are approximately zero mean that $\mathbf{H}$ is, for all practical purposes, singular. The inverse $\mathbf{H}^{-1}$ does not exist, or contains extremely large values that will render Newton's method unstable for FWI.

3. While this is not the case for my one-dimensional example, real FWI problems have so many unknowns that even obtaining $\mathbf{H}$, let alone inverting it, at every iteration is beyond reasonable economic limits.

The first of these is often addressed by instead using the so-called Gauss-Newton method. For problems where the objective function is expressed as the sum of squares of other functions, using the matrix $\mathbf{J}^T\mathbf{J}$ in place of $\mathbf{H}$ ensures that the step direction always points in the direction of decreasing functional.

Having removed the possibility of negative eigenvalues, the second effect - zero and near-zero eigenvalues - can be mitigated by adding a positive-definite damping term to the matrix. This prevents gradient components for the least-sensitive directions, which while usually very small are still influenced by approximations and the finite accuracy of floating point computation, from being amplified too much.

The third effect - the inability to obtain or store $\mathbf{H}$ - is usually dealt with by instead using the so-called *truncated* Newton (or Gauss-Newton) method (e.g. Métivier et al. (2014)).

In this method, the damped-inverse-Hessian operation $(\mathbf{H}+\alpha\mathbf{I})^{-1}\nabla f$ is solved iteratively, using for example a conjugate gradient method along with a second-order adjoint state method to compute products $\mathbf{Hv}$.

To illustrate the usefulness of second order methods such as these, I will perform a demonstration using a very similar technique: using a damped variation of the Hessian computed above as an additional, constant preconditioner.

Looking at the eigendecomposition in Figures 5.30 and 5.31, we see that after applying the approximate diagonal preconditioner from the previous chapter, the components of the model we wish to recover are above eigenvalue index 410. Letting the diagonal preconditioner used to construct the matrix above be called $\mathbf{D}$, our first attempt at an additional preconditioner, therefore, is:

$$\mathbf{P}_{410} = \mathbf{D}^{-0.5}\mathbf{H}\mathbf{D}^{-0.5} + \lambda_{410}\mathbf{I} \tag{5.8}$$

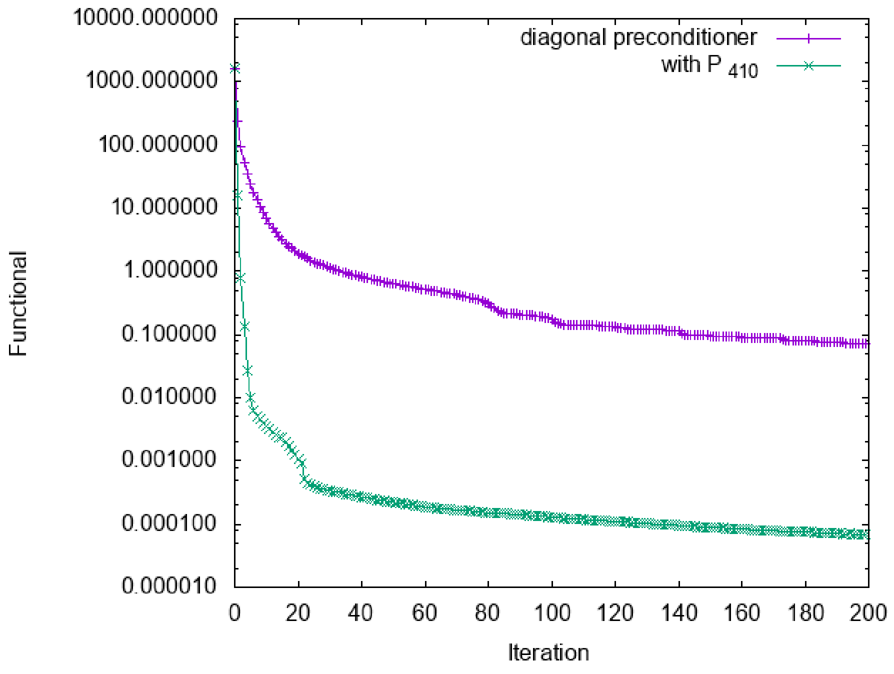
where λ_{410} is the 410th eigenvalue (ranked from smallest to largest) of $\mathbf{D}^{-1}\mathbf{H}$.

This gives an update relation:

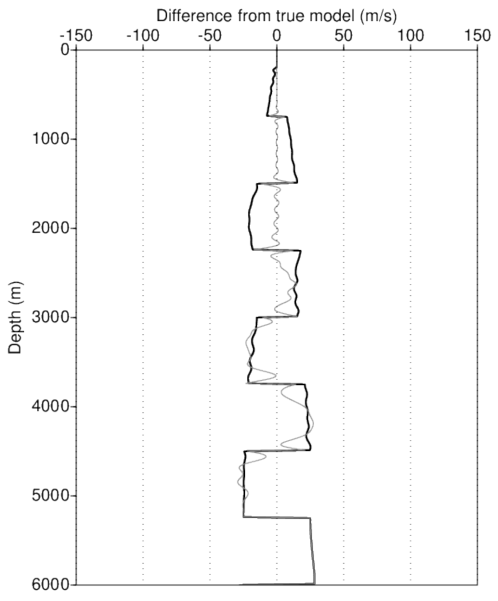
$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k\mathbf{D}_k^{-0.5}(\mathbf{D}^{-0.5}\mathbf{H}\mathbf{D}^{-0.5} + \lambda_{410}\mathbf{I})^{-1}\mathbf{D}_k^{-0.5}(\nabla f)(\mathbf{m}_k)$$

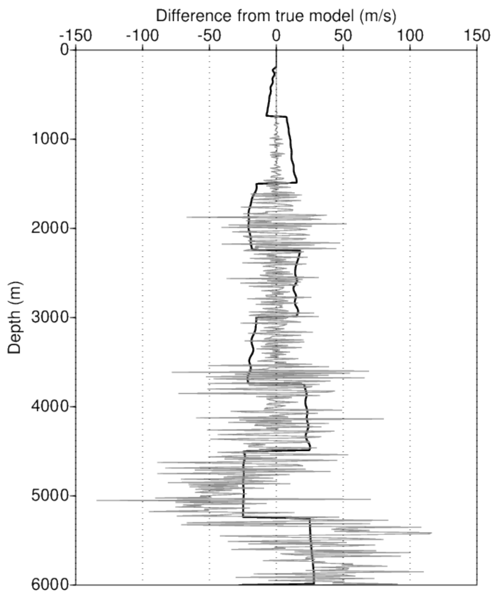
Using this additional preconditioning step to compensate for non-amplitude-based effects, I then performed FWI starting with the same checkerboard as previous tests, this time with an amplitude of 1% - i.e. $\Delta m = \pm 0.01(m - 1515)$. Figure 5.32a shows the evolution of the FWI functional with and without the preconditioner.

Clearly, the extra preconditioning yields much faster convergence of the residuals - after less than 50 iterations the functional has been reduced by a factor of more than 1 million. Looking at the model, however, tells a less happy side of the story, as can be observed in Figure 5.32c: even though the inversion has correctly recovered the macromodel down to about 4500m depth, high-wavenumber artefacts have been introduced throughout the result. I suspect that this is due to the extra preconditioner amplifying components of the gradient which are very close to the null space of the FWI problem - model directions associated with tiny eigenvalues, to which the functional is insensitive but which nevertheless are



(a) Evolution of the functional for FWI preconditioned using (5.8), and with the diagonal preconditioner from Chapter 4.





(b) The initial perturbation from the true model, and the remaining perturbation after 200 iterations of FWI with the diagonal preconditioner from Chapter 4.

(c) The initial perturbation from the true model, and the remaining perturbation after 200 iterations of FWI preconditioned with $\mathbf{P}_{410}$.

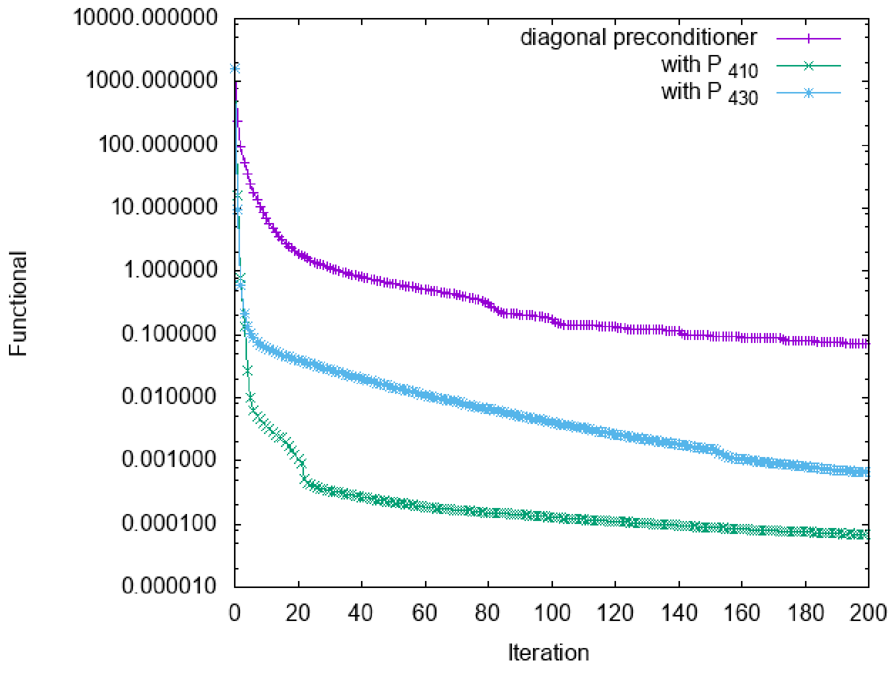
Figure 5.32: Results of the inversion using the preconditioner $\mathbf{P}_{410}$ from (5.8). The convergence in the functional is much faster, but the resulting model is badly contaminated with artefacts.

undesirable in the result. If the gradient were exactly equal to the first-order approximation $\mathbf{H}\Delta\mathbf{m}$, these components would have been smaller than the desired corrections, even when amplified by the inverse of the damped Hessian. However, the combination of numerical imprecision and the fact that the gradient varies nonlinearly mean that, in this case at least, the artefacts dominate the result. This model is a good example of the effect I mentioned in the introductory chapter: an excellent match between the modelled and observed data does not necessarily imply that the model itself is correct!

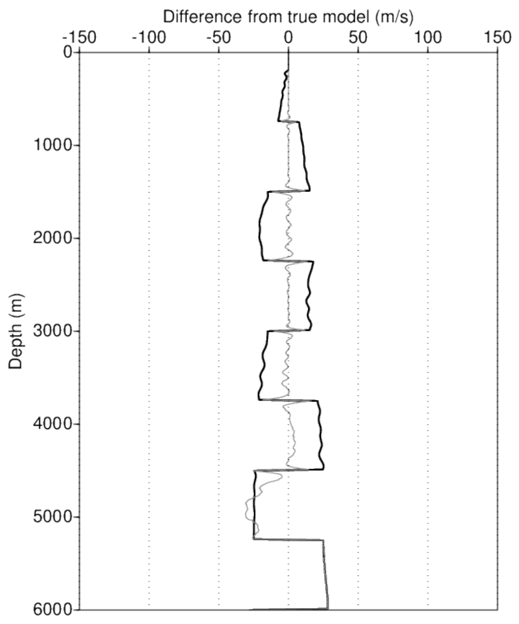
This can be resolved in a number of ways, but the most obvious is to damp the Hessian a little more when constructing the extra preconditioner. If, instead of choosing eigenvalue λ_{410}, we damp the Hessian with λ_{430}, these artefacts will not be amplified quite as much. Figure 5.33 shows the evolution of the functional with this extra damping, and the model after 200 iterations - the improvement in the functional is not as significant, but the resulting model is much better.

There are two crucial results here. The first, demonstrated earlier, is that even under ideal conditions - starting vanishingly close to the true model, with perfect physics, and no noise - diagonally-preconditioned FWI is unable to make significant corrections to the deep low-wavenumber velocity model within a reasonable number of iterations. The second is that, at least under these ideal conditions, off-diagonal preconditioning can remedy that situation. I would therefore assert that any effort to address this phenomenon must, at a minimum, deal with this underlying problem.

Relaxing the ideal conditions a little, it is reasonable at this point to ask how far from the true model one can start. Figure 5.34 shows, in a different form, the evolution of model perturbations which start with checkerboard amplitudes of 5%, 10% and 20% of the true model, rather than the 1% used until now. The colour scale indicates the difference between the model and the true model, relative to the original checkerboard amplitude at that depth. The horizontal axis is the number of descent iterations. In all of these runs it can be observed that in the early iterations the errors in some parts of the model, around the 2500m mark for the 5% perturbation but becoming shallower as the perturbation grows, an artefact appears where the model moves further away from the true model



(a) Evolution of the function for FWI preconditioned using $\mathbf{P}_{410}$, $\mathbf{P}_{430}$, and the diagonal preconditioner from Chapter 4.



(b) The initial perturbation from the true model, and the remaining perturbation after 200 iterations of FWI preconditioned with $\mathbf{P}_{430}$. The artefacts present in Figure 5.32c are not introduced.

Figure 5.33: Results of the inversion using the preconditioner $\mathbf{P}_{430}$. The convergence in the functional is slower than the earlier result using $\mathbf{P}_{410}$, but the resulting model is no longer polluted by high-wavenumber artefacts. This is another example of the effect, noted in Chapter 4, where faster convergence of the objective function is a poor proxy for improved convergence in the model space.

(despite the functional still improving). As the iterations continue, this artefact tends to be pushed deeper and eventually removed. In the 5% and 10% cases, the Hessian-derived preconditioner $\mathbf{P}_{430}$ seems to create larger artefacts, but also to accelerate the process of removing them, finally permitting recovery of the deeper macromodel components. In the 20% case, the preconditioner introduces such large artefacts that the inversion does not recover within the number of iterations tested. This probably indicates that by stepping in a different direction - but still in a direction where the functional decreases - the inversion has inadvertently crossed into either a local minimum, or at least a part of the model space where the curvature is different from that compensated for by the preconditioner, causing inversion progress to become very slow.

These experiments clearly show that preconditioning helps in this very simple, inverse-crime problem. However, it is difficult to say exactly which elements of the data are being used to improve the macromodel result. Ideally, one would hope that the preconditioner has made the inversion more sensitive to the moveout effects that tomography traditionally makes use of, since these are robust to real-world effects and would indicate that this technique could work outside of the inversion-crime scenario. Noting that the inversion already worked well near the interfaces of the checkerboard, another possibility is that the preconditioning has increased the size of the recovered region near interfaces by somehow amplifying the effect of the very weak, but not completely absent, low acoustic frequencies. This is a less desirable outcome, since the low signal-to-noise ratio at those frequencies in real data might thwart the application of this method. To test this second theory, I conducted another test where instead of a checkerboard, the velocity was uniformly decreased by $0.01(c - 1515)$ through the whole model. While there are obviously still velocity variations present in the "true" model, this removes the sharp checkerboard variations around which even the diagonally-preconditioned inversion did well. Figure 5.35 shows that the preconditioner still helps dramatically with deeper macromodel recovery in this case.

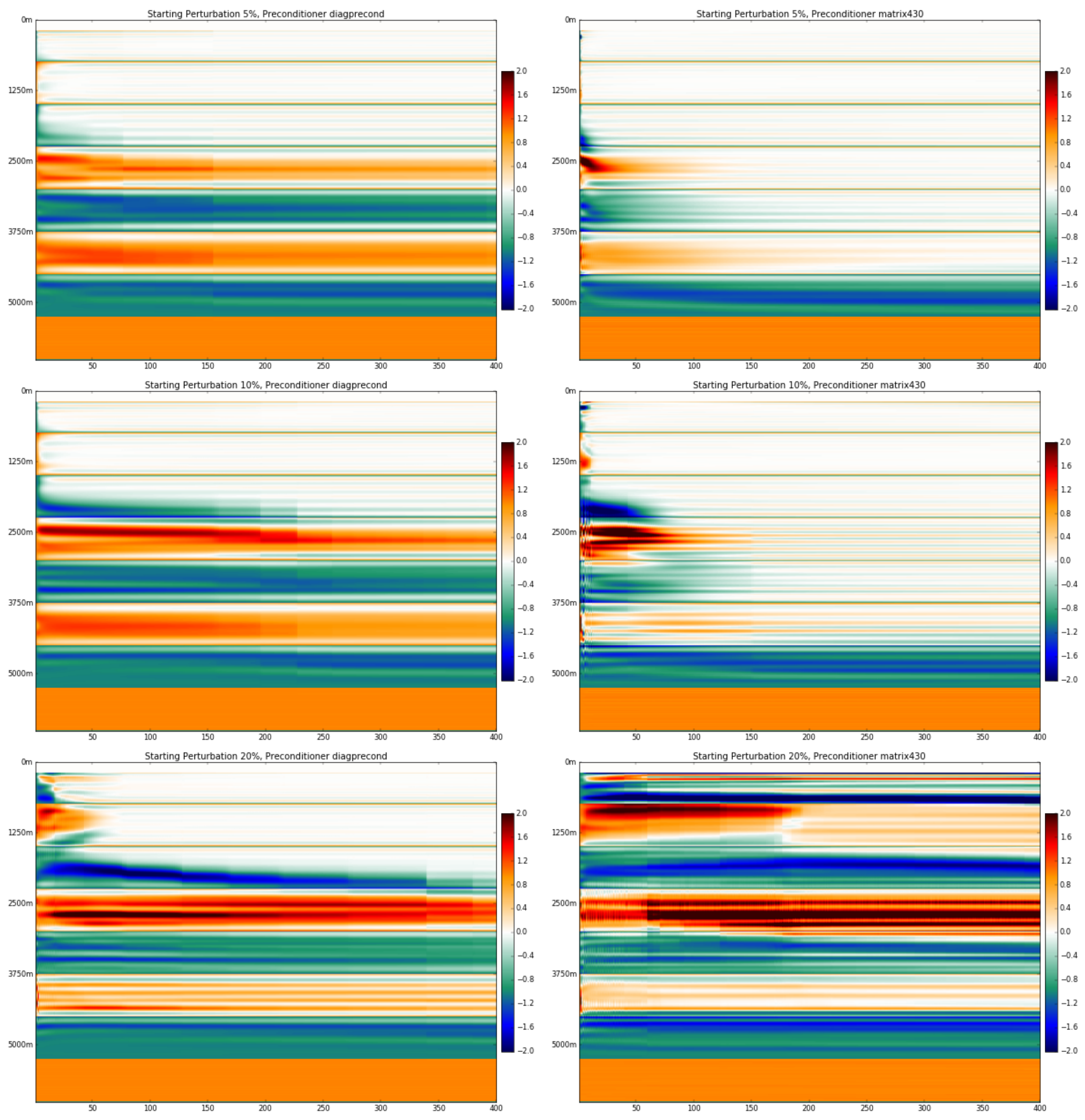


Figure 5.34: Evolution of the perturbation from the true model for inversions with different preconditioners and different amplitudes of the starting perturbations. The images on the left show iterations preconditioned with the diagonal preconditioner from chapter 4, images on the right show iterations preconditioned with $\mathbf{P}_{430}$ in a similar way to equation (5.8). Initial perturbations are based on equation (5.1), with coefficients of 5% (top), 10% (middle), 20% (bottom). Value plotted is the ratio of difference between the current model and the true model to the amplitude of the initial perturbation - white indicates recovery of the true model. Using the preconditioner derived from the Hessian matrix allows for recovery of the deep part of the macromodel, but introduces additional artefacts when starting too far from the true model.

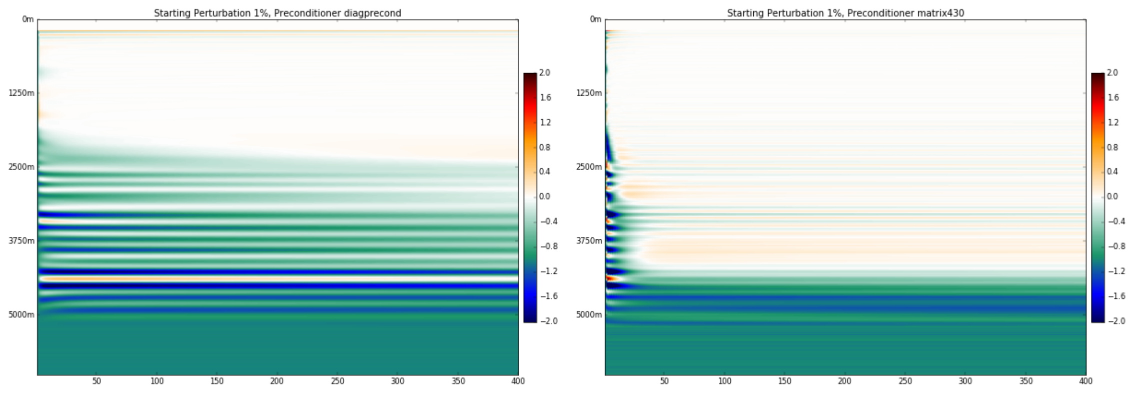


Figure 5.35: Analagous to Figure 5.34 but with a uniform percentage perturbation based on equation (5.1) rather than a checkerboard, and a 1% initial perturbation size. This demonstrates that the improved macromodel recovery is not only near sharp interfaces.

5.8 Conclusion

In this chapter, I have proposed a simple QC for full waveform inversion problems, and used it to illustrate FWI's well-known inability to update low-wavenumber components of velocity in the deeper section of models poorly sampled by wide-angle scattering. I have demonstrated, using a proprietary reflection tomography code from Downunder Geosolutions, that this deeper macromodel information can be extracted from the same data used by FWI. I have obtained and decomposed the matrix of second functional derivatives for a simple 1D FWI problem, and shown that poor conditioning in this deeper region is at least partially responsible for the lack of update. Finally, I have constructed a preconditioner by applying damping to the true Hessian, and shown that it allows recovery of deeper macromodel components with FWI in this one-dimensional inverse crime problem.

This is an important result. It is, I believe, a lens through which to look at all attempts to solve the "Reflection FWI" problem. I have shown that part of the action of the Hessian in the deeper region is a convolution with the spectrum of the wavelet - deghosting, as in Ramos-Martinez et al. (2013), broadens the spectrum of this convolution, improving the conditioning of the functional surface. Splitting the unknowns into high- and low-wavenumber components in an alternating inversion, as in Xu et al. (2012); Yao et al. (2014) allows the now-separate steps to be scaled differently, which can be thought of as a kind of off-diagonal preconditioner, or perhaps a variant of the subspace method

(Kennett et al., 1988). Using the pseudo-depth domain, as in Brossier et al. (2014), reduces the crosstalk between these two parameter classes, which is especially important in an alternating inversion. I believe that these techniques all tackle the same underlying problem - one which I hope I have shed some more light on with this work. The results in this chapter suggest to me that applying higher-order optimisation schemes, such as truncated Newton or Gauss-Newton, could potentially achieve many of the same gains when it comes to reflection FWI.

I would also suggest that the observations I have made regarding the relationship between the one-dimensional Hessian and the model itself, detailed in Section 5.5, could form the basis of a cheap, approximate Hessian-vector product. In combination with a linear solver, this could in turn be used as an affordable preconditioner either for FWI itself, or for the linear solver within a Newton or Gauss-Newton method.

"Sometimes I lie awake at night, and I ask 'Where have I gone wrong?'. Then a voice says to me 'This is going to take more than one night.' "

Charles M. Schulz

6 Analysis of the Hessian for a realistically-sized FWI problem

6.1 Introduction

In the previous chapter, I showed that in the context of a 1.5D inverse-crime problem, knowledge of the curvature information of the objective function, as represented by the Hessian matrix, was sufficient to permit recovery of low-wavenumber model information from deeper reflection events. The curvature of the functional surface in that case was causing the background velocity components in the deeper part of the model to be severely under-represented in the gradients, and this needs to be compensated for in order to allow updates to these components within an economically viable number of iterations.

For larger problems, however, obtaining, storing and inverting the Hessian to assist with inversion is prohibitively expensive: even a small 3D problem might have one million model parameters, implying a Hessian matrix of up 4TB, costing the equivalent of a nearly a million iterations to obtain. Therefore, some kind of approximation is needed.

This approximation can come in many forms. Preconditioning based on an approximation to the diagonal of the Hessian is simple and nearly universal in practice, as discussed in Chapter 4. Truncated Newton and Gauss-Newton methods, as used in e.g. Pratt et al. (1998) and Métivier et al. (2013), form an approximation to the Hessian implicitly, by computing a step $\mathbf{\Delta m} = \mathbf{H}^{-1}\nabla f$ using only a small number of conjugate gradient iterations involving Hessian-vector products. Quasi-Newton methods, such as L-BFGS (Nocedal, 1980) build a diagonal-plus-low-rank approximation to the Hessian based on gradient changes as the inversion proceeds.

All of these have their downsides. Diagonal preconditioning, as I have shown earlier in this thesis, still leaves a very large spectrum of eigenvalues in the preconditioned Hessian matrix, due to the presence of significant off-diagonal elements in the original matrix. This kind of compensation might be able to boost updates in the deeper region compared to those in the shallow, but it cannot, for example, weight different wavenumbers unequally. Truncated Newton methods carry the cost of several iterations of conjugate gradient per outer "FWI iteration", with each Hessian-vector product costing approximately the same as a traditional descent iteration. Quasi-Newton methods, by building a diagonal-plus-low-rank approximation, are able to compensate for the effect of the very largest eigenvalues only. This is nonetheless an important effect - as I will show in this chapter, the shape of the eigenvalue spectrum means that compensating for the twenty or so largest eigenvalues could potentially give a speedup of around 5x. However, even a 5x speedup does not quite bring the recovery of deep macromodel anomalies within reach - these components are far further down the spectrum, down at around $\lambda/\lambda_{\max} = 10^{-4}$.

My aim with the experiment of this chapter, therefore, is to lay the foundations for a better non-diagonal approximation to the Hessian which, once developed, could be used as a preconditioner either for the descent method on its own, as a complement to L-BFGS, or even to precondition the truncated Newton steps. To this end, I will obtain the full Hessian and Gauss-Newton Hessians for the full, two-dimensional model from which the previous work was extracted, and attempt to gain some generic understanding - along the lines of the qualitative observations made about the 1.5D case in the previous chapter - that could form the basis of such an approximation.

Additionally, I will obtain these matrices at several stages of the inversion, to try and understand how the curvature changes as we slowly reduce the functional.

Note: this work was done in collaboration with Dr Lluis Guasch, whose assistance I gratefully acknowledge. Many of the observations herein are the product of fruitful discussions between us.

6.2 Rows of the Hessian for a 2D problem

To obtain the Hessian and Gauss-Newton Hessian matrices for a larger, 2D problem, I followed the same procedure described in the previous chapter. In this case, however, the problem is vastly larger. Not only are there many more model variables, but due to the lateral variation of the model, the computation of the gradient can no longer be performed by simulating only a single shot.

To make the problem tractable, I used a downsampled version of the Chevron/SEG blind test problem that I have been using throughout this thesis. By filtering the data down to 6Hz, I was able to increase the grid spacing to 50m while still remaining almost within the dispersion criteria laid out in the first chapter (which recommends a maximum spacing of 40m). Furthermore, I chose a random 15% of the shots in the original dataset to use for the gradient computations.

Nonetheless, the problem size remains large. Even with the grid reduced from 4000x480 points down to 1000x120 points, and the data set reduced from 1600 down to 240 shots, computation of the matrices for a single model takes approximately 24 node-weeks, using 20-core nodes. Each of the resulting matrices is 120000x120000, or 57.6GB in size. With the aim of seeing how the curvature of the functional surface changes as we move towards the solution, I obtained these matrices for models at three different stages of the inversion:

1. the starting model used in previous inversions. Hereafter this is referred to as the

"Starting Model".

2. a relatively good inversion result obtained from running FWI on the downsampled grid, obtained after 20 iterations. Hereafter this is referred to as the "Inverted Model".
3. an inversion result obtained by running FWI on the downsampled grid for 800 iterations - after which many undesirable artefacts have appeared in the resulting model. I will refer to this as the "Overcooked Model".

These models are shown in Figure 6.1

Unfortunately, visualisation of the matrices in two dimensions is unintuitive, since each side of the matrix represents a two-dimensional domain that needs to be flattened. I believe it is informative, however, to look at individual rows of the matrices, much as the individual rows of the one-dimensional Hessian in the previous chapter were looked at in graph form. Individual rows of this matrix are model-sized and can be visualised similarly to a model.

Figure 6.2 shows some rows of the Hessians, computed at the model after 20 iterations of FWI. The crosshairs on each figure indicate the model parameter for the particular row being presented - so the value of the point under the crosshairs is a diagonal entry in the Hessian matrix. As in the one-dimensional case, the matrix is shown after scaling both rows and columns by the square root of the approximate diagonal Hessian constructed in chapter 4.

The qualitative structure that was visible in the one-dimensional Hessian is also visible here. In particular, the stretched autocorrelation of the wavelet appears in each row near the crosshairs, along with a component linking the position of reflectors to the sum of velocities above them. This is most visible in panels 5 and 6 of Figure 6.2. Also still visible is the noticeable change in character between rows where the model is well-sampled by turning rays, and those where it is not - i.e. between the uppermost and lowermost panels in Figure 6.2.

What is newly visible in the two-dimensional case is the appearance of the wavelet convolution even for the shallow rows, in the turning-ray region (e.g. in the top panel of Figure 6.2). This was previously being obscured by the summation along the horizontal axis. As we look at rows for elements further down the model, the range of angles sampled by the wavelet contribution narrows - from 360 degrees in the shallow region down to the narrow band of sampled reflection angles in the deep. The positive Hessian elements in the turning ray region are also better defined - they appear along all source-receiver paths where the row in question forms part of the path.

This effect is also visible in the results of Tang and Lee (2015) - who use products of Hessians with point diffractors to compute "point-spread functions". However, I don't believe the reflector-position/macrovelocity interaction has been previously noted in this context.

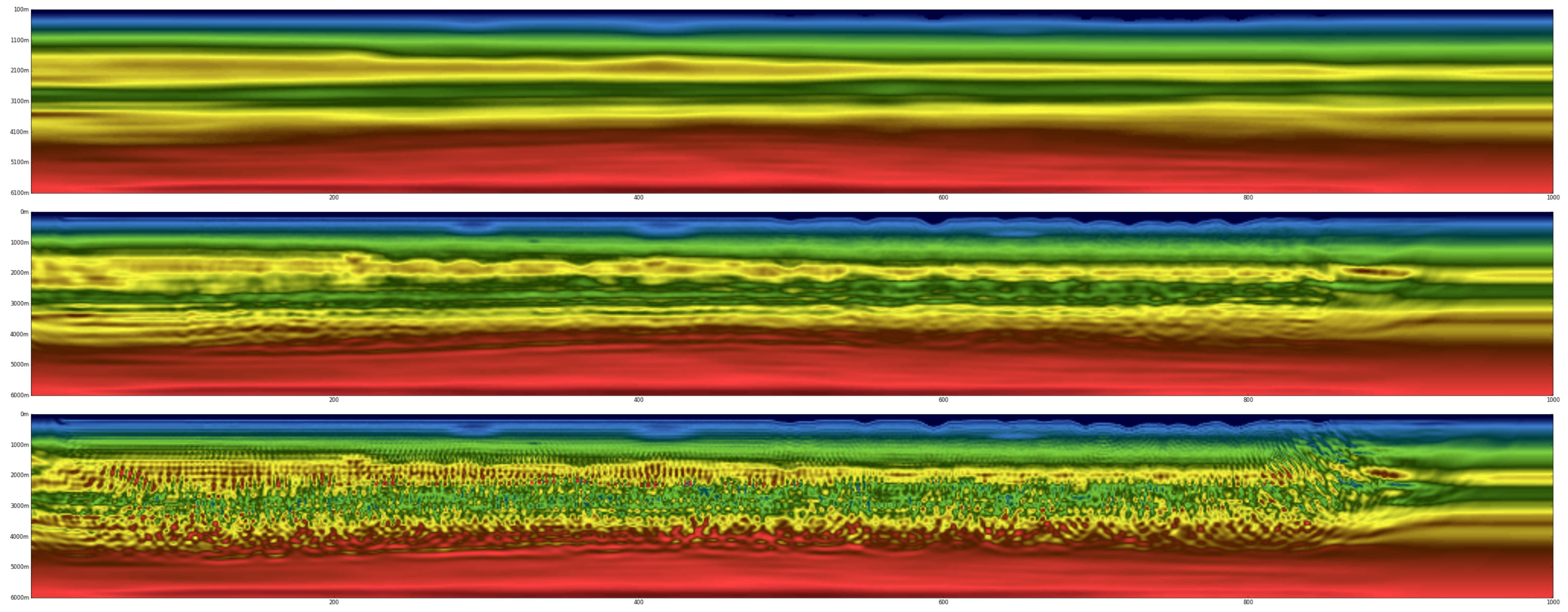


Figure 6.1: The three models used for the Hessian analysis of this chapter. Top: a smooth starting model; Centre: inverted result after 20 iterations; Bottom: overcooked inversion result after 800 iterations.

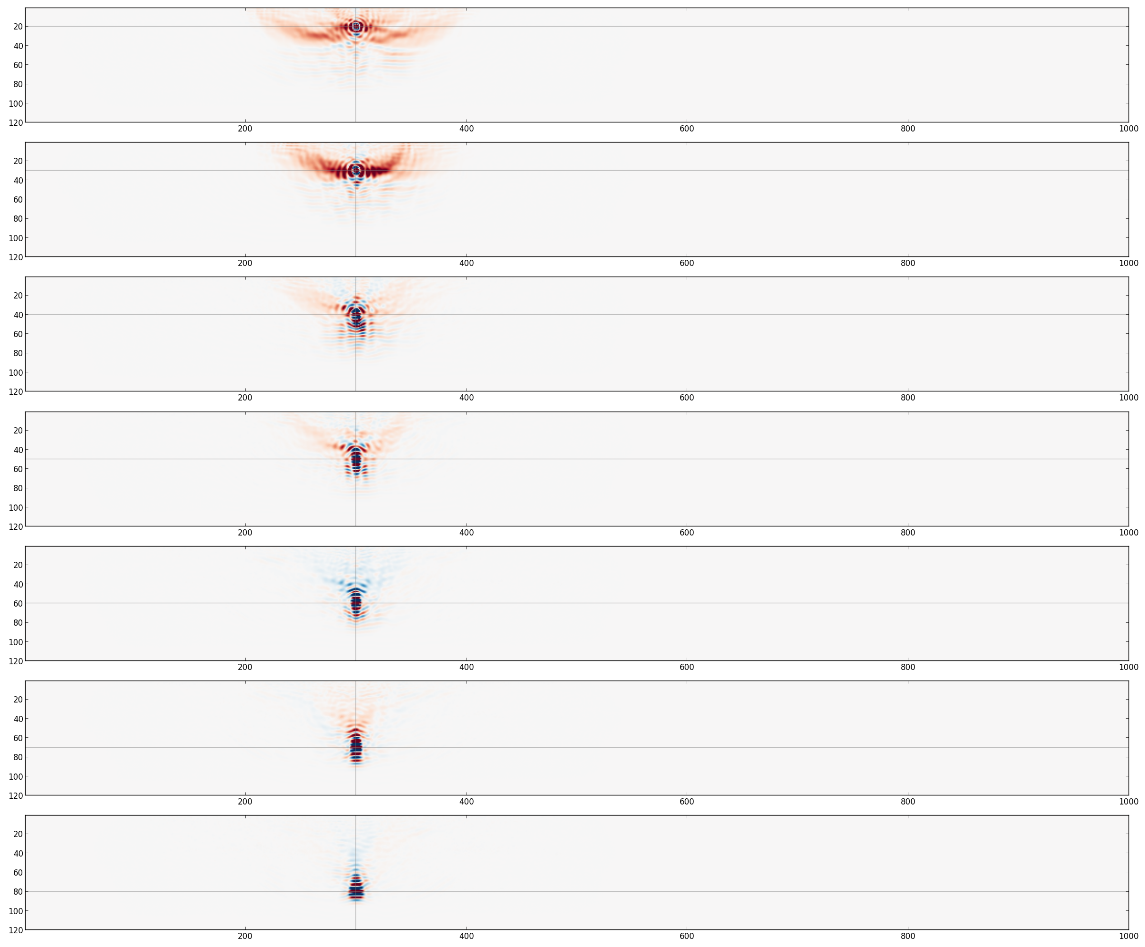


Figure 6.2: Some rows of the Hessian obtained at the inverted model, after scaling by the approximate diagonal preconditioner. The row displayed is indicated by the location of the crosshairs. Note the transition in character from the transmitted-arrival region (top two panels) to the reflection only region (bottom three panels).

6.3 Eigendecomposition of these matrices

As in the one dimensional case, I will now examine the eigendecomposition of these matrices. Unlike in the previous case, however, this is now a significant computational problem, requiring specialised algorithms.

In this section, I will briefly review the meaning of the eigendecomposition, describe the algorithm used to recover it for a matrix as large as this, and then show some results which again correlate with empirical observations about FWI.

The decompositions that I will be computing are of the diagonally-preconditioned Hessians and Gauss-Newton Hessians, $\mathbf{P}^{-1}\mathbf{H}$ and $\mathbf{P}^{-1}\mathbf{J}^T\mathbf{J}$. As touched on in the previous chapter, these are nonsymmetric matrices, which complicates the meaning of the decomposition somewhat.

The matrix $\mathbf{P}^{-1}\mathbf{H}$ is an important description of the local behaviour of the inversion process: it maps a change in the model parameters to the (negative of the) change in the step direction. For an easy problem, we would like these two directions to be perfectly aligned - perturbing the model parameters in any direction should result in a step back along the same direction. However, as we saw in the simple quadratic figures of Chapter 4, this is usually not the case.

Rather, as we saw in the preceding section, a perturbation of a single model parameter results in a step direction which includes both region of the surrounding points, depending on the spectrum of the wavelet used for inversion, and a small contribution along all of the paths taken by acoustic energy to reach that model point, be scattered, and return to the receiver array.

The eigendecomposition $\mathbf{P}^{-1}\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$ gives us another way of thinking about the action of this model perturbation to step direction mapping:

1. first, we decompose the model perturbation onto a basis of eigenvectors
2. then, weight each component according to the corresponding eigenvalue ($\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues)
3. then, sum the resulting weighted components back together to obtain the step direction

While the 480x480 matrix from Chapter 5 could be trivially eigendecomposed in Python using Scipy, computing the eigenvalues and vectors of a large matrix such as this (120000x120000) requires the use of specialised libraries. This is necessary because the memory required exceeds that contained in any one node of the compute cluster, but also because the computational complexity of eigendecomposition is $O(n^3)$, and the expected runtime would be very high without a lot of parallelism (n^3 in this case is 1.7×10^{15}).

I wrote a relatively simple code to call the library Elemental (Poulson et al., 2013), a recently-developed C++ library for distributed-memory (i.e. MPI) dense matrix algebra. This, in turn, uses the highly efficient algorithm PMRRR (Bientinesi et al., 2005) to compute the decomposition. Using these advanced algorithms, performing the eigendecompositions was actually much faster than obtaining the Hessian itself - taking approximately 4 hours on 12 nodes of the Imperial College compute cluster cx1. Note that while the Hessian matrix itself was obtained using single-precision arithmetic (used in solving the wave equation), the eigendecompositions were computed in double-precision.

6.4 Eigenvalue spectrum of the various matrices

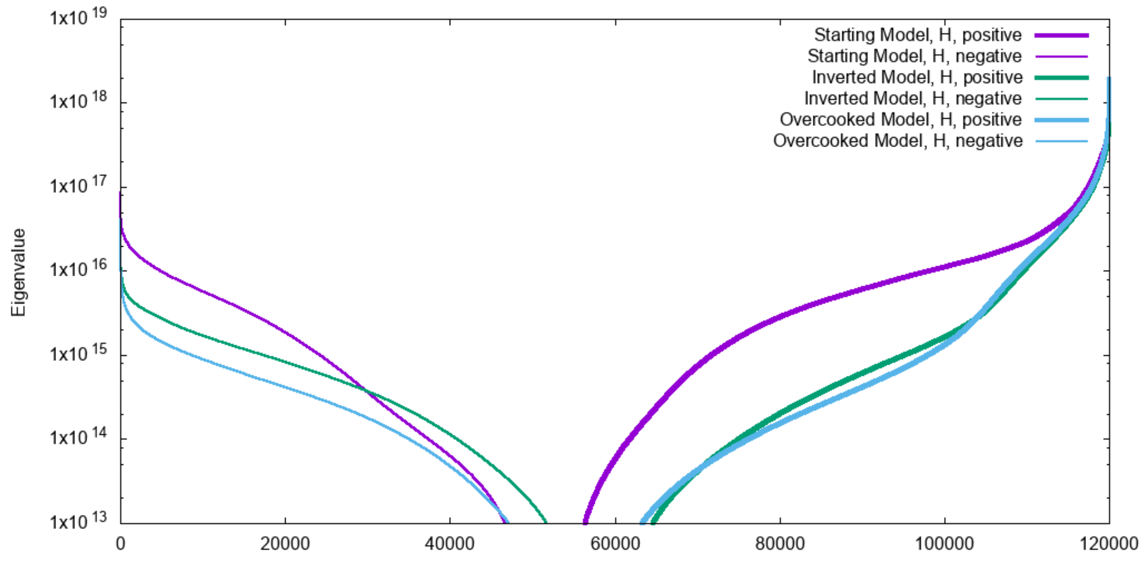


Figure 6.3: Eigenvalue spectrum of the diagonally preconditioned Hessian matrices for each of the three models shown in Figure 6.1. Negative values are indicated by the thinner lines on the left hand side. Values below the bottom of the graph are likely to be below the meaningful precision of the computation.

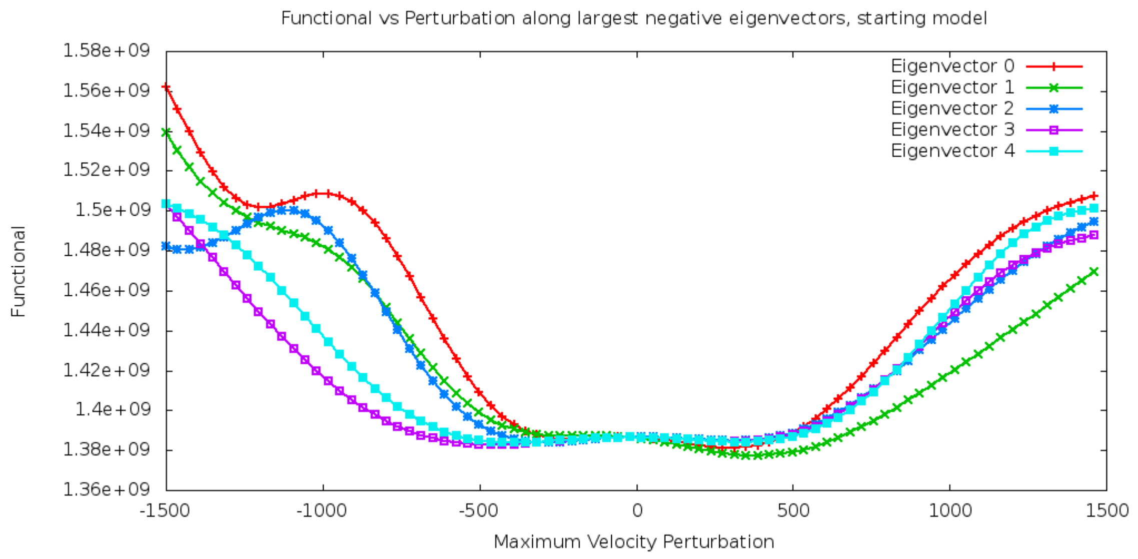


Figure 6.4: Objective function vs perturbation of the model along the 5 most negative eigenvectors corresponding to the leftmost values in Figure 6.3. Eigenvectors and perturbations are with respect to the starting model.

Figure 6.3 shows the eigenvalue spectra of the three Hessian matrices, on a log scale, with negative values shown on the left hand side of the graph with the smaller line width. I have clipped the graph at about $10^{-5}\lambda_{\max}$, as I believe that the values smaller than that are beyond the accuracy limit of the algorithm used to obtain the matrices themselves. Note that, so that they may be directly compared, the same preconditioner $\mathbf{P}$, computed at the smooth starting model, was used when constructing all of these decompositions.

There are a few interesting observations to be made about this figure. The first is that there are a lot of directions with negative curvature, and sometimes this negative curvature is relatively strong - at the starting model, it is around one-tenth of the maximum upward curvature. This lack of convexity is significant - it means that along these directions, the starting model does not lie towards the bottom of a "valley" but more towards the top of a "hill". Indeed, Figure 6.4 shows this to be the case. To me, this suggests that the inversion is starting sufficiently far from the true model that local inversion techniques such as gradient descent are potentially risky - in the neighbourhood of the global minimum there is no negative curvature.

Another interesting thing to notice about Figure 6.3 is that the overall strength of the negative curvatures seems to be reduced as we iterate, even as we iterate so much that we

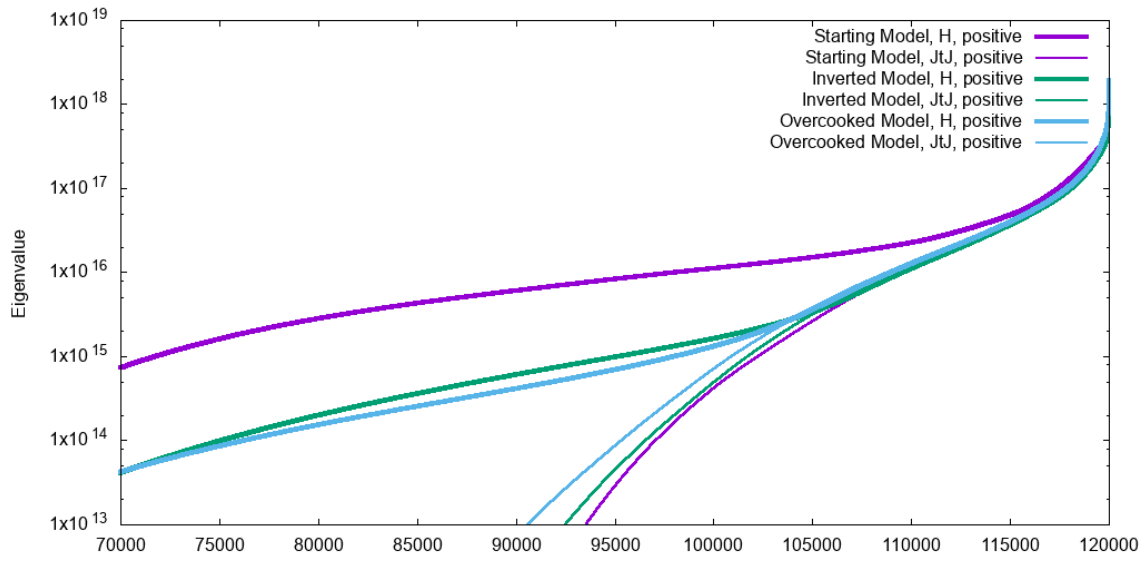


Figure 6.5: A zoomed-in view of the right hand section of Figure 6.3, also showing the eigenvalues of the corresponding Gauss-Newton Hessians $\mathbf{J}^T\mathbf{J}$.

introduce artefacts into the model. This is somewhat intuitive, as we are moving downhill along the functional surface, even if it is into a local minimum or poorly-conditioned valley.

A more surprising aspect of Figure 6.3 is that the overall strength of the *positive* curvatures also seems to be reducing as we iterate, especially in the early iterations - by as much as a factor of 20 for much of the spectrum. This is true for all but the strongest positive eigenvalues. I believe this has some important consequences for the inversion, as it suggests that early iterations may update the model in directions to which it later becomes relatively insensitive. This could introduce artefacts early in the inversion that will not be removed later, as due to other changes updates in these directions might come to have a comparatively small effect on the functional. Another (not mutually exclusive) possibility is that along certain directions, the bottom of the valley is extremely flat, with steep sides, resembling a function like $y = x^4$. This is somewhat supported by Figure 6.4 and - if true - could prove to be a fundamental problem for FWI, at least in the traditional squared-data-residuals formulation.

Figure 6.5 shows the righthand half of the spectrum from Figure 6.3, but with the largest eigenvalues of the corresponding Gauss-Newton Hessians $\mathbf{J}^T\mathbf{J}$. The first interesting thing to note is how little these spectra change during inversion, compared to those of the Hessian.

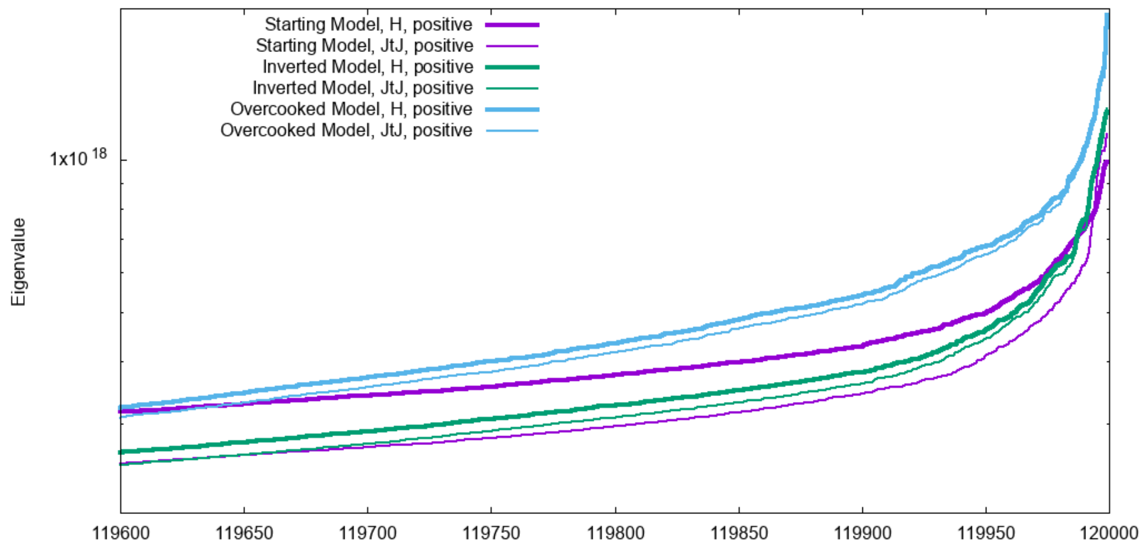


Figure 6.6: A zoomed-in view of the very largest eigenvalues from Figure 6.5.

Also interesting to me is how the spectra of $\mathbf{H}$ and $\mathbf{J}^T\mathbf{J}$ seem to converge on each other - as if the Hessian spectrum is being squeezed down onto that of the Gauss-Newton Hessian. I have no explanation for this, but if repeatable in other circumstances then it suggests that the spectrum of $\mathbf{J}^T\mathbf{J}$ could be a good predictor of what the spectrum of $\mathbf{H}$ might look like closer to the global minimum. That might be expected in an inverse crime problem, as equation (5.3) shows that the difference is proportional to the magnitude of the residual vector, but is more of a surprise in this context, where the data is unlikely to be able to be matched exactly, and the functional has only been reduced by around a factor of 3 - implying that significant residuals remain.

Finally, it is worth examining the very top end of the spectrum. Figure 6.6 shows a zoomed-in view of the largest eigenvalues for all 6 of the matrices being looked at so far. It shows that the convergence between $\mathbf{H}$ and $\mathbf{J}^T\mathbf{J}$ occurs as we iterate, though the trend of the overall positive curvature reducing is reversed for these larger curvatures. One final takeaway from this zoomed-in view is the change in the slope of the curve at the very top of the spectrum: the first 20 or so eigenvectors dominate the spectrum. I speculate that this could be useful information, for instance, when choosing a memory size for the L-BFGS algorithm, which aims to form a diagonal-plus-low-rank approximation to the Hessian.

6.5 Eigenvectors

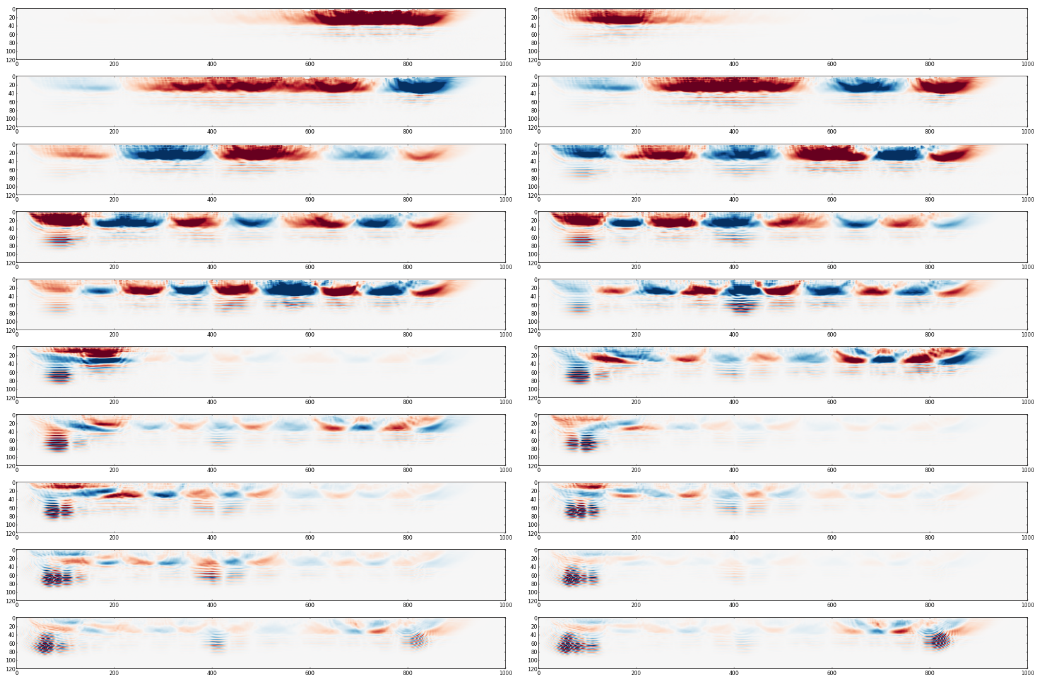


Figure 6.7: The eigenvectors for the twenty largest positive eigenvalues of the diagonally preconditioned Hessian, evaluated at the starting model.

The eigenvectors themselves are enlightening, as well. Figure 6.7 shows the twenty largest positive eigenvectors of the preconditioner-scaled Hessian, evaluated at the starting model. The first ten eigenvectors, taken together, seem to form a Fourier-like basis for the low-wavenumber components of the upper region of the model. Successively weaker eigenvectors seem to correspond to the first few harmonics of the model in horizontal wavenumber. The next set of ten, while showing some extra sensitivity near the edges of the model, perhaps due to over-compensation by the preconditioner, exhibit similar properties but along something approximating a second harmonic in vertical wavenumber. This explains why, cycle-skipping notwithstanding, FWI is able to recover low-wavenumber model perturbations in the shallow region so quickly.

Figure 6.8 shows the five largest negative eigenvectors for the same matrix. These are the strongest downward-curving directions, and are the directions along which the model was perturbed when generating the functional curves from Figure 6.4. These, too, are interesting, as they coincide spatially with the appearance of some of the artefacts that

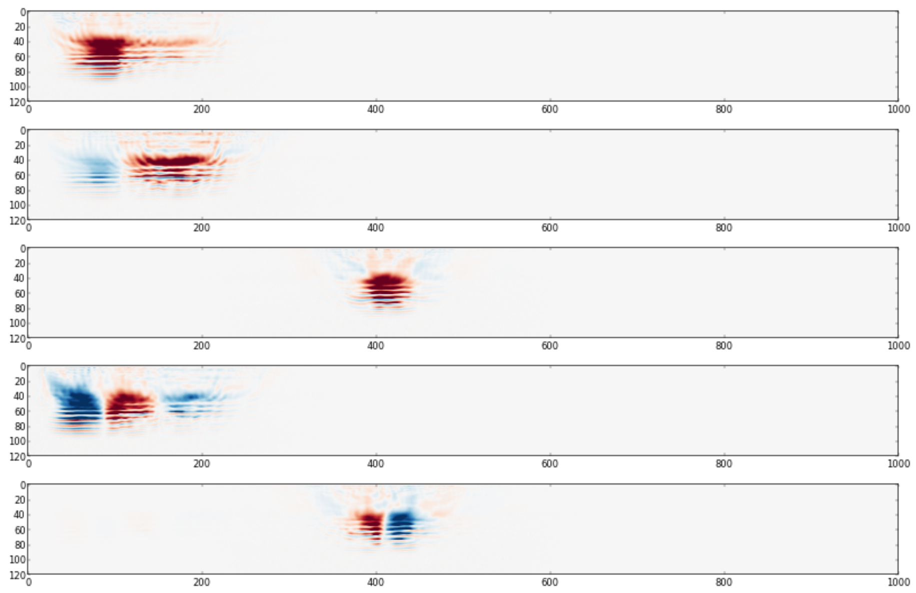


Figure 6.8: The eigenvectors for the five largest negative eigenvalues of the diagonally preconditioned Hessian, evaluated at the starting model.

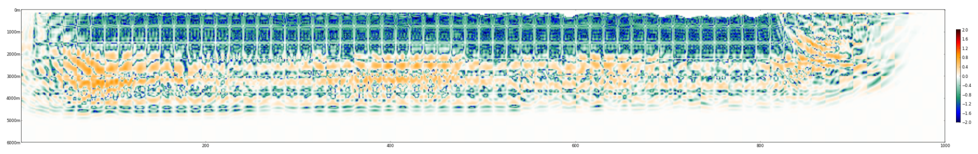


Figure 6.9: The QC procedure of Section 5.2 applied to iteration 20 of the low-frequency inversion performed in this Chapter. Note the correspondence in position between the largest growth in difference between the models and the largest negative eigenvalues shown in Figure 6.8.

appear when performing an inversion. Indeed, using the QC procedure of Section 5.2 on the lower-frequency inversion performed here shows - see Figure 6.9 - that these are the regions where small differences in the starting model are amplified by the inversion.

To better illustrate which parts of the update are in which parts of the eigendecomposition, I projected a checkerboard made up of 5km x 1km blocks onto bands of eigenvectors, each corresponding to an order of magnitude of eigenvalues. This is shown in Figure 6.10 for the Hessian evaluated at the starting model, Figure 6.11 for the Hessian evaluated at the inverted model, and Figure 6.12 for the Hessian matrix obtained at the overcooked model.

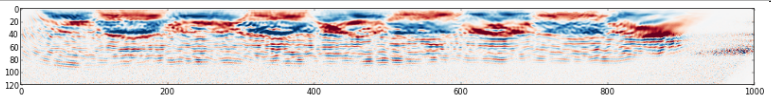
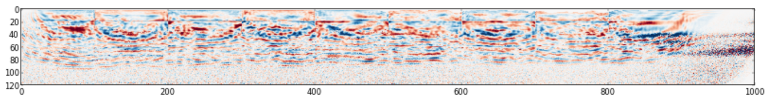
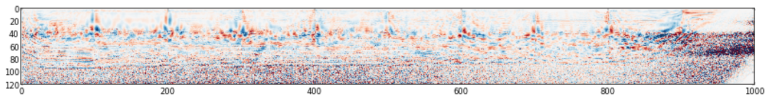
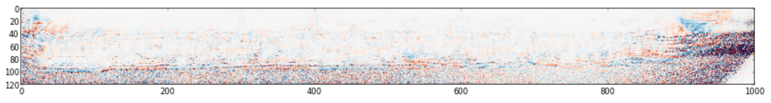
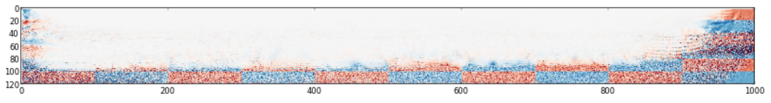
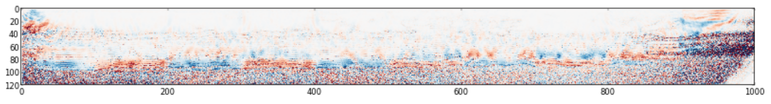
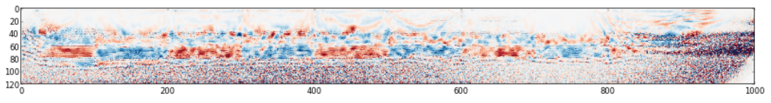
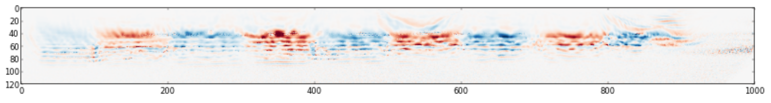
$0.1 < \frac{\lambda}{\lambda_{\max}} \leq 1$	
$0.01 < \frac{\lambda}{\lambda_{\max}} \leq 0.1$	
$0.001 < \frac{\lambda}{\lambda_{\max}} \leq 0.01$	
$10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 0.001$	
$-10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 10^{-4}$	
$-0.001 < \frac{\lambda}{\lambda_{\max}} \leq -10^{-4}$	
$-0.01 < \frac{\lambda}{\lambda_{\max}} \leq -0.001$	
$-0.1 < \frac{\lambda}{\lambda_{\max}} \leq -0.01$	

Figure 6.10: Projection of a checkerboard with 5km x 1km checks onto ranges of eigenvectors of the preconditioner-scaled Hessian, evaluated at the starting model. Note how much of the deeper macromodel velocity structure, in the 3rd and 4th bands of checkers, projects onto eigenvectors with negative eigenvalues, meaning that in these directions the functional surface is concave, not convex - as also shown in Figure 6.4. The deepest part of the model, below around 4.5km, is unrecoverable because of the conservative mute chosen on the data. This was added to avoid any contribution from the numerical reflection from the lower boundary. Note, these projections are all orthogonal to one another - the original checkerboard is recovered by adding together the panels in this figure. Note that because these Hessians are not evaluated at the true model, but rather at the starting model, negative eigenvalues are somewhat expected, and are not merely present due to round-off errors.

6.6 Conclusion

There are several interesting takeaways from the experiment performed in this chapter. Most importantly, I have shown that - as in the 1.5D case - the structure of rows of the Hessian is predictable, and appears to be formed of the sum of three types of contribution. Firstly, the autocorrelation of the source wavelet, stretched by the local background velocity and with a distribution of orientation dependent on the angles sampled by the acquisition

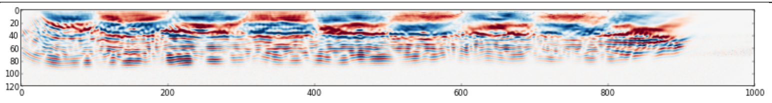
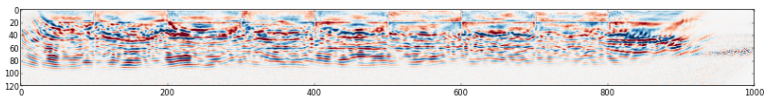
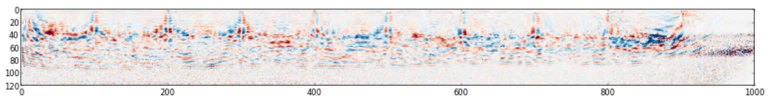
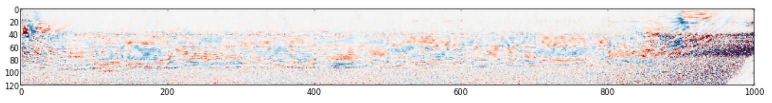
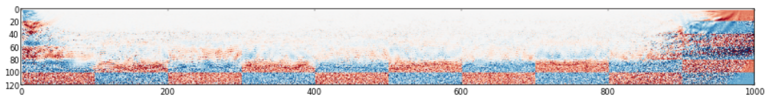
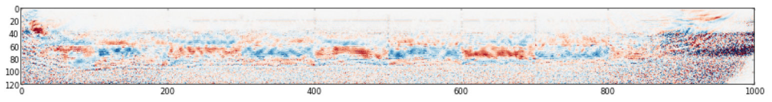
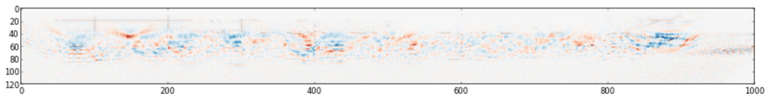
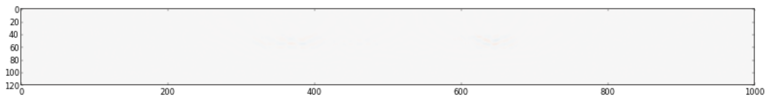
$0.1 < \frac{\lambda}{\lambda_{\max}} \leq 1$	
$0.01 < \frac{\lambda}{\lambda_{\max}} \leq 0.1$	
$0.001 < \frac{\lambda}{\lambda_{\max}} \leq 0.01$	
$10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 0.001$	
$-10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 10^{-4}$	
$-0.001 < \frac{\lambda}{\lambda_{\max}} \leq -10^{-4}$	
$-0.01 < \frac{\lambda}{\lambda_{\max}} \leq -0.001$	
$-0.1 < \frac{\lambda}{\lambda_{\max}} \leq -0.01$	

Figure 6.11: Projection of a checkerboard with 5km x 1km checks onto ranges of eigenvectors of the preconditioner-scaled Hessian, evaluated at the inverted model. Compare with Figure 6.10. The strongest negative curvatures are no longer present, and some of the deeper macrovelocity model information is now mapping onto directions which are in the positive (i.e. convex) part of the spectrum.

geometry at that depth. This imprint of the wavelet appears as concentric circles all around the point in the shallowest regions, where angle coverage is complete, and changes to a more linear pattern in the deeper region where the range of opening angles is small. This first contribution is closely connected to the "Point-spread functions" noted in Tang and Lee (2015). The second contribution, for points that are on transmission paths from a source point to a receiver point, there is a contribution all along the path, related to how the functional changes if the modelled arrival time of that event changes. Thirdly, there is a reflector-position contribution, which links the two almost-ambiguous effects of changing the velocity above a reflector, and changing its apparent depth.

Another interesting point to note is that as the inversion proceeds and some model

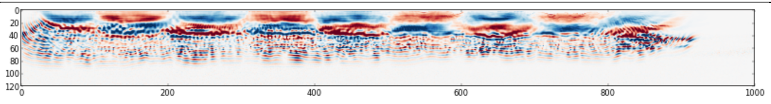
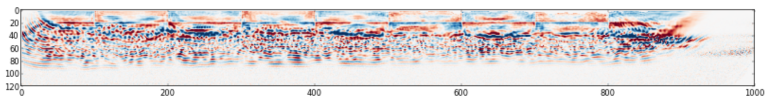
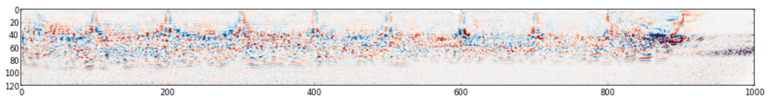
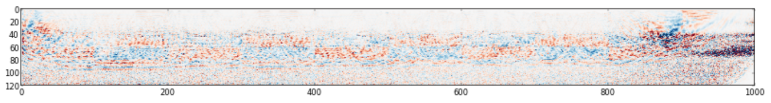
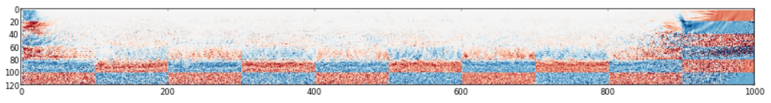
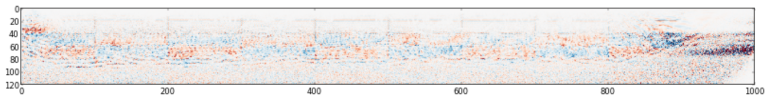
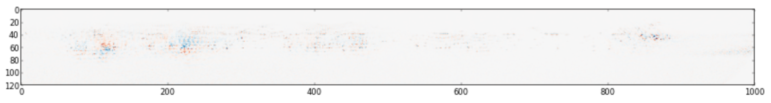
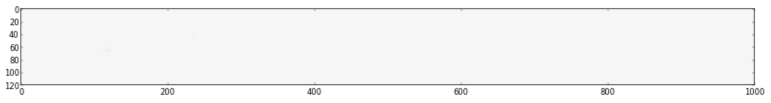
$0.1 < \frac{\lambda}{\lambda_{\max}} \leq 1$	
$0.01 < \frac{\lambda}{\lambda_{\max}} \leq 0.1$	
$0.001 < \frac{\lambda}{\lambda_{\max}} \leq 0.01$	
$10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 0.001$	
$-10^{-4} < \frac{\lambda}{\lambda_{\max}} \leq 10^{-4}$	
$-0.001 < \frac{\lambda}{\lambda_{\max}} \leq -10^{-4}$	
$-0.01 < \frac{\lambda}{\lambda_{\max}} \leq -0.001$	
$-0.1 < \frac{\lambda}{\lambda_{\max}} \leq -0.01$	

Figure 6.12: Projection of a checkerboard with 5km x 1km checks onto ranges of eigenvectors of the preconditioner-scaled Hessian, evaluated at the overcooked model. Compare with Figure 6.10 and 6.11. The deeper macrovelocity model information is now no longer mapping significantly into the concave part of the eigenspectrum. This suggests that several thousand more iterations could potentially have generated a significant update in this region, had the inversion been continued.

directions become better-resolved, the gap between the corresponding eigenvalues of the Newton and Gauss-Newton Hessians appears to shrink. This is unexpected, and may warrant further research. Yet another is the correspondence between parts of the model which are poorly resolved and negative curvature. These can be identified either through negative-eigenvalue components (in this case where we have the matrix), or in the more general situation by applying the QC procedure of Section 5.2. Once concave regions of the model have been identified, perhaps a global scheme such as that in Debens et al. (2015) could be applied to find an improved starting model in those regions.

Finally, the existence of a significant null-space is again confirmed, and the eigenvalue spectra obtained suggest that this null space changes in size, potentially growing, as the inversion proceeds. This argues strongly for the need for some kind of targeted regularisation, which would permit recovery of some low-eigenvalue model components, while suppressing artefacts to which the data is truly insensitive.

In science one tries to tell people, in such a way as to be understood by everyone, something that no one ever knew before. But in poetry, it's the exact opposite.

— Paul Dirac

7 Conclusions and Future Work

Going into this project, my aims were twofold. Firstly, to make use of my software background to make some contributions to the field, and the FULLWAVE group in particular, and secondly to chip away at the problem of "Reflection FWI".

This thesis, I believe, covers some aspects of both of these goals. The new high-order kernel in FULLWAVE3D now sees extensive use among the group's industrial sponsors, and allows both them and the group itself to invert at higher frequencies than would otherwise have been possible within computational resource constraints. I believe that efforts are underway to extend that integration to include a specialised propagator for isotropic models, which I regarded as a lower priority at the time. PyFWI, while far from elegant in some ways, can I hope serve as an introduction to FWI for computationally-savvy future students coming through the group.

On the second point, I hope that I have presented a convincing argument in these pages that the relative weakness of FWI when it comes to inverting reflection data is not fundamental, but rather numerical, in nature. I have reproduced the same problem in an inverse crime context, and used sophisticated preconditioning to show that it can be resolved in that case. By conducting a thorough sensitivity analysis on a realistic (albeit two-dimensional) problem, I have obtained further evidence that while FWI does in fact suffer from a very large null space, the deeper macrovelocity components are not part of it.

I think this has some important implications for future FWI research. Firstly, the low relative sensitivity of these deep, low wavenumber components of velocity to FWI means that their recovery may well require more, possibly vastly more, computational expense than is traditionally invested in an inversion. This might take the form of more inversion iterations, or more expensive iterations, such as are required for the Newton or Gauss-Newton methods, or the use of more sophisticated, computationally expensive preconditioners. However, I believe that the twin trends of improved algorithms and more affordable computation will make this viable in the near future.

Secondly, the existence of such a large null space, combined with the changing curvature of the FWI problem as the inversion proceeds, implies that some form of regularisation will be required to prevent artefacts from remaining in the final model. This is an area of active research (Guitton, 2011; Li et al., 2013; Esser et al., 2016), and will become essential as multiparameter inversion becomes mainstream and null spaces grow even further.

It is my sincere wish that others can build on the findings of this project, and develop FWI such that it can eventually fulfil the goal that I expressed for it in the introduction: simultaneous inversion of both reflection and refraction data, with limited preprocessing, for the full spectrum of model wavenumbers to which the data is sensitive.

7.1 Future Work

There are several research directions that could be pursued to build upon this work.

Firstly, I think it is important to examine the nature of the null space revealed by this type of analysis. These are the model components to which our data is essentially insensitive - as shown in Chapter 5, the model can contain substantial noise along these directions and still fit the data very well. This ought to guide the development of regularisation methods: an ideal regularisation should constrain these directions to something geologically plausible, or perhaps even just visually pleasing, without contaminating or damping model components which do actually have an effect on the data. The most obvious parts of the null space in the data considered here are the extremely high wavenumber components of the model, which are effectively invisible to the source wavelet used. The direction of propagation must also be considered - high lateral wavenumbers in regions where energy is predominantly only propagating vertically also will have little to no effect on the data. Adapting the regularisation to take these kinds of effects into account will, I believe, be necessary.

Secondly, I believe that the empirical observations I have made about the nature of the Hessian could be used to construct an improved preconditioner for FWI. A row of the Hessian can be approximated by a the combination of the three effects I have noted: transmission travel time matching along refraction ray paths passing through the point, a convolution with the spectrum of the source wavelet, and a reflector-position/migration-velocity relationship based on the local derivative of velocity. If raytracing information can be used to construct a computational approximation to the action of the Hessian on a vector, then this can be used to construct a preconditioner by applying a linear solver to the approximate system at each FWI iteration.

Another possibility for preconditioning is based on observing a rank structure within the Hessian matrix. Hierarchical Semi-Separable (HSS), and Hierarchical (H-) matrices allow for vastly compressed representations of some data, based on certain off-diagonal blocks having limited rank. In the one-dimensional case, such as Figure 5.25, this kind of structure is immediately apparent - many off diagonal blocks would be well-approximated by a rank-1 matrix $\mathbf{u}\mathbf{v}^T$. With an appropriate reordering of the rows and columns into some kind of geometric layout, I believe that an off-diagonal low rank approximation could be constructed for two- and three-dimensional problems, which would allow for the sampling (via matrix-vector products) and storage of the second-order derivative information in FWI. This could then pave the way for much more efficient second order methods.

Bibliography

Asnaashari, A., R. Brossier, S. Garambois, F. Audebert, P. Thore, and J. Virieux, 2013, Regularized seismic full waveform inversion with prior model information: GEOPHYSICS, **78**, R25–R36.

Ben-Hadj-Ali, H., S. Operto, and J. Virieux, 2011, An efficient frequency-domain full waveform inversion method using simultaneous encoded sources: GEOPHYSICS, **76**, R109–R124.

Bentley, J. L., 1975, Multidimensional binary search trees used for associative searching: Commun. ACM, **18**, 509–517.

Bientinesi, P., I. S. Dhillon, and R. A. van de Geijn, 2005, A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations: SIAM J. Sci. Comput., **27**, 43–66.

Bleistein, N., J. K. Cohen, and F. G. Hagin, 1985, Computational and asymptotic aspects of velocity inversion: GEOPHYSICS, **50**, 1253–1265.

Bondhugula, U., A. Hartono, J. Ramanujam, and P. Sadayappan, 2008, A practical automatic polyhedral parallelizer and locality optimizer: SIGPLAN Not., **43**, 101–113.

Bourgeois, A., M. Bourget, P. Lailly, M. Poulet, P. Ricarte, and R. Versteeg, 1990, Marmousi, model and data: EAEG Workshop – Practical Aspects of Seismic Data Inversion, European Assocaition of Geoscientists & Engineers, 5–16.

Brossier, R., S. Operto, and J. Virieux, 2014, Velocity model building from seismic reflection data by full-waveform inversion: Geophysical Prospecting, **63**, 354–367.

Burgess, T., and M. Warner, 2015, Preconditioning FWI with approximate receiver Green's functions: SEG Technical Program Expanded Abstracts ’15, Society of Exploration Geophysicists, 1116–1121.

Cerveny, V., 2005, Seismic ray theory: Cambridge University Press.

Chavent, G., 1974, Identification of functional parameters in partial differential equations: Joint Automatic Control Conference, Institute of Electrical and Electronics Engineers, 155–156.

Chevron/SEG, 2014, SEG 2014 Chevron Full Waveform Inversion Synthetic: https://s3.amazonaws.com/open.source.geoscience/open_data/seg_workshop_fwi_2014/seg_workshop_fwi_2014.html.

Cohen, J. K., and J. J. W. Stockwell, 2015, Cwp/su: Seismix un*x release no. 44: an open source software package for seismic research and processing.

Courant, R., K. Friedrichs, and H. Lewy, 1928, Über die partiellen Differenzengleichungen der mathematischen Physik: Mathematische Annalen, **100**, no. 1, 32–74.

da Silva, N. V., A. Ratcliffe, G. Conroy, V. Vinje, and G. Body, 2014, A new parameterization for anisotropy update in full waveform inversion: SEG Technical Program Expanded Abstracts ’14, Society of Exploration Geophysicists, 1050–1055.

Debens, H. A., M. Warner, A. Umpleby, and N. V. da Silva, 2015, Global anisotropic 3d fwi: SEG Technical Program Expanded Abstracts 2015, 1193–1197.

Díaz, E., and A. Guitton, 2011, Fast full waveform inversion with random shot decimation: SEG Technical Program Expanded Abstracts ’11, Society of Exploration Geophysicists, 2804–2808.

Esser, E., L. Guasch, T. van Leeuwen, A. Y. Aravkin, and F. J. Herrmann, 2016, Total-variation regularization strategies in full-waveform inversion: Computing Research Repository. ((Computing Research Repository)).

Frigo, M., and V. Strumpen, 2005, Cache oblivious stencil computations: Proceedings of the 19th Annual International Conference on Supercomputing, ACM, 361–366.

Gauthier, O., J. Virieux, and A. Tarantola, 1986, Two-dimensional nonlinear inversion of seismic waveforms: Numerical results: Geophysics, **51**, no. 7, 1387–1403.

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**, no. 4, 1091–1106.

Guasch, L., and M. Warner, 2014, Adaptive waveform inversion – FWI without cycle skipping – Applications: 76th EAGE Conference & Exhibition, European Assocaition of Geoscientists & Engineers, We–E106–14.

Guitton, A., 2011, A blocky regularization scheme for full waveform inversion: SEG Technical Program Expanded Abstracts 2011, 2418–2422.

Hastings, W. K., 1970, Monte carlo sampling methods using markov chains and their applications: Biometrika, **57**, 97–109.

Hicks, G. J., 2002, Arbitrary source and receiver positioning in finite-difference schemes using Kaiser windowed sinc functions: Geophysics, **67**, no. 1, 156–166.

Houbiers, M., E. Wiarda, J. Mispel, D. Nikolenko, D. Vigh, B.-E. Knudsen, M. Thompson, and D. Hill, 2012, 3d full-waveform inversion at mariner — a shallow north sea reservoir: SEG Technical Program Expanded Abstracts 2012, 1–5.

Irabor, K., and M. Warner, 2016, Reflection fwi: SEG Technical Program Expanded Abstracts 2016, 1136–1140.

Kennett, B., M. Sambridge, and P. Williamson, 1988, Subspace methods for large inverse problems with multiple parameter classes: Geophysical Journal International, **94**, no. 2, 237–247.

Keppner, G., 1991, Ludger Mintrop: The Leading Edge, **10**, 21–28.

Kim, S., and H. Lim, 2007, High-order schemes for acoustic waveform simulation: Applied Numerical Mathematics, **57**, 402–414.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi, 1983, Optimization by simulated annealing: Science, **220**, 671–680.

Lehmann, I., 1936, P: Publications du Bureau Central Seismologique International, Série A, Travaux Scientifique, **14**, 87–115.

Li, M., L. Liang, A. Abubakar, and P. M. van den Berg, 2013, Structural similarity regularization scheme for multiparameter seismic full waveform inversion: SEG Technical Program Expanded Abstracts 2013, 1089–1094.

Li, Y., C. Liu, and S. Franke, 1990, Three-dimensional green's function for wave propagation in a linearly inhomogeneous medium—the exact analytic solution: Journal of the Acoustical Society of America, **87**, 2285–2291.

Li, Y. E., and L. Demanet, 2016, Full-waveform inversion with extrapolated low-frequency data: GEOPHYSICS, **81**, R339–R348.

Lighthill, M. J., 1965, Group velocity: IMA J Appl Math, **1**, no. 1, 1–28.

Lions, J. L., and E. Magenes, 1972, Non-homogeneous boundary value problems and applications I: Springer-Verlag Berlin Heidelberg New York.

Mallet, R., 1848, On the dynamics of earthquakes: Presented at the Transactions of the Royal Irish Academy, M. H. Gill.

Mallet, R., 1862, Great Neapolitan Earthquake of 1857: The First Principles of Observational Seismology: Chapman and Hall.

Métivier, L., F. Bretaudeau, R. Brossier, S. Operto, and J. Virieux, 2014, Full waveform inversion and the truncated Newton method: Quantitative imaging of complex subsurface structures: Geophysical Prospecting, **62**, no. 6, 1353–1375.

Métivier, L., R. Brossier, Q. Mérigot, E. Oudet, and J. Virieux, 2016, An optimal transport approach for seismic tomography: application to 3d full waveform inversion: Inverse Problems, **32**, 115008.

Métivier, L., R. Brossier, J. Virieux, and S. Operto, 2013, Full waveform inversion and the truncated newton method: SIAM Journal on Scientific Computing, **35**, B401–B437.

Mohorovičić, A., 1910, Potres od 8. X 1909: Naklada Kr. Hrv.-Slav.-Dalm. Zem. Vlade, Odjela za Bogoštovje i Nastavu, volume **9** *of* Godišnje Izvješće Zagrebačkog Meteorološkog Opservatorija za Godinu 1909.

Morgan, J., M. Warner, R. Bell, J. Ashley, D. Barnes, R. Little, K. Roele, and C. Jones, 2013, Next-generation seismic experiments: Wide-angle, multi-azimuth, three-dimensional, full-waveform inversion: Geophysical Journal International, **195**, no. 3, 1657–1678.

Nocedal, J., 1980, Updating Quasi-Newton Matrices with Limited Storage: Mathematics of Computation, **35**, 773–782.

Oldham, R. D., 1906, The constitution of the interior of the earth, as revealed by earthquakes: Quarterly Journal of the Geological Society, **62**, no. 1–4, 456–475.

Plessix, R.-E., and W. A. Mulder, 2004, Frequency-domain finite-difference amplitude-preserving migration: Geophysical Journal International, **157**, 975–987.

Poulson, J., B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, 2013, Elemental: A new framework for distributed memory dense matrix computations: ACM Trans. Math. Softw., **39**, 13:1–13:24.

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**, no. 2, 341–362.

Pratt, R. G., and R. M. Shipp, 1999, Seismic waveform inversion in the frequency domain, part 2: Fault delineation in sediments using crosshole data: GEOPHYSICS, **64**, 902–914.

Ramos-Martinez, J., N. Chemingui, S. Crawley, Z. Zou, A. Valenciano, and E. Klochikhina, 2016, A robust fwi gradient for high-resolution velocity model building: SEG Technical Program Expanded Abstracts 2016, 1258–1262.

Ramos-Martinez, J., K. Zou, S. Kelly, and B. Tsimelzon, 2013, Reflection fwi from fully deghosted towed-streamer data: A field data example: SEG Technical Program Expanded Abstracts 2013, 887–891.

Rosenbrock, H. H., 1960, An automatic method for finding the greatest or least value of a function: The Computer Journal, **3**, no. 3, 175–184.

SEG Technical Standards Committee, 2002, SEG-Y rev 1 Data Exchange Format, release 1.0: https://www.seg.org/Portals/0/SEG/News%20and%20Resources/Technical%20Standards/seg_y_rev1.pdf.

Shah, N., M. Warner, T. Nangoo, A. Umpleby, I. Štekl, J. Morgan, and L. Guasch, 2012, Quality assured full-waveform inversion: Ensuring starting model adequacy: SEG Technical Program Expanded Abstracts '12, Society of Exploration Geophysicists, 1–5.

Shannon, C. E., 1949, Communication in the presence of noise: Proc. Institute of Radio Engineers, **37**, no. 1, 10–21.

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**, no. 5, 592–606.

Sirgue, L., and R. G. Pratt, 2004, Efficient waveform inversion and imaging: A strategy for selecting temporal frequencies: Geophysics, **69**, no. 1, 231–248.

Tang, Y., and S. Lee, 2015, Multi-parameter full wavefield inversion using non-stationary point-spread functions: SEG Technical Program Expanded Abstracts 2015, 1111–1115.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, no. 8, 1259–1266.

Umpleby, A., M. Warner, and I. Štekl, 2010, Time vs frequency for 3D wavefield tomography: 72nd EAGE Conference & Exhibition – Workshops and Fieldtrips, European Assocaition of Geoscientists & Engineers, WS06.

van Leeuwen, T., and F. J. Herrmann, 2013, Fast waveform inversion without source-encoding: Geophysical Prospecting, **61**, 10–19.

van Leeuwen, T., and F. J. Herrmann, 2014, 3d frequency-domain seismic inversion with controlled sloppiness: SIAM J. Scientific Computing, **36**.

van Leeuwen, T., F. J. Herrmann, and B. Peters, 2014, A new take on FWI: wavefield reconstruction inversion: Presented at the EAGE Annual Conference Proceedings.

Verschuur, D. J., A. J. Berkhout, and C. P. A. Wapenaar, 1992, Adaptive surface-related multiple elimination: GEOPHYSICS, **57**, 1166–1177.

Versteeg, R. J., 1993, Sensitivity of prestack depth migration to the velocity model: Geophysics, **58**, no. 6, 873–882.

Virieux, J., and S. Operto, 2009, An overview of full-waveform inversion in exploration geophysics: Geophysics, **74**, no. 6, WCC1–WCC26.

Wang, S., X. S. Li, F.-H. Rouet, J. Xia, and M. V. De Hoop, 2016, A parallel geometric multifrontal solver using hierarchically semiseparable structure: ACM Trans. Math. Softw., **42**, 21:1–21:21.

Warner, M., and L. Guasch, 2014, Adaptive waveform inversion – FWI without cycle skipping – Theory: 76th EAGE Conference & Exhibition, European Assocaition of Geoscientists & Engineers, We–E106–13.

Warner, M., A. Ratcliffe, T. Nangoo, J. Morgan, A. Umpleby, N. Shah, V. Vinje, I. Štekl, L. Guasch, C. Win, G. Conroy, and A. Bertrand, 2013, Anisotropic 3D full-waveform inversion: Geophysics, **78**, no. 2, R59–R80.

Whitmore, N. D., and S. Crawley, 2012, Applications of rtm inverse scattering imaging conditions: SEG Technical Program Expanded Abstracts 2012, 1–6.

Wolfe, M., 1989, More iteration space tiling: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, ACM, 655–664.

Xu, S., D. Wang, F. Chen, Y. Zhang, and G. Lambare, 2012, Full waveform inversion for reflected seismic data: Presented at the 74th EAGE Conference and Exhibition incorporating EUROPEC 2012.

Yao, G., N. da Silva, M. Warner, A. Umpleby, and D. Wu, 2017, Improved fwi convergence using efficient receiver-side spatial preconditioning employing ray theory: Presented at the 79th EAGE Conference and Exhibition 2017.

Yao, G., M. Warner, and A. Silverton, 2014, Reflection fwi for both reflectivity and background velocity: Presented at the 76th EAGE Conference and Exhibition 2014.

Yount, C., 2015, Vector folding: Improving stencil performance via multi-dimensional simd-vector representation: Presented at the 2015 IEEE 17th International Conference of High Performance Computing and Communications (HPCC).

Zhang, J., and Z. Yao, 2013, Optimized explicit finite-difference schemes for spatial derivatives using maximum norm: Journal of Computational Physics, **250**, 511–526.

Zhou, W., R. Brossier, S. Operto, and J. Virieux, 2015, Full waveform inversion of diving & reflected waves for velocity model building with impedance inversion based on scale separation: Geophysical Journal International, **202**, 1535–1554.