IFOS3D

User Manual

SOFI
IFOS

Authors

The IFOS3D code was first developed by Simone Butzer at the Karlsruhe Institute of Technology (KIT) from 2009-2014. Contributions were given by Andre Kurzmann and Lisa Groos.

The forward code is based on the viscoelastic FD code SOFI3D by Bohlen (2002).

Different external libraries for time domain filtering are used. The copyright of the source codes are held by different persons:

cseife.c, cseife.h:
Copyright © 2005 by Thomas Forbriger (BFO Schiltach)

cseife_deriv.c, cseife_gauss.c, cseife_rekfl.c, cseife_rfk.c and cseife_tides.c:
Copyright © 1984 by Erhard Wielandt
This algorithm was part of seife.f. A current version of seife.f can be obtained from http://www.software-for-seismometry.de/

The Matlab implementation of a few SU routines, mainly used to read and write SU files are:
Copyright © 2008, Signal Analysis and Imaging Group
For more information: http://www-geo.phys.ualberta.ca/saig/SeismicLab
Author: M.D.Sacchi

License

IFOS3D is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 2.0 of the License only.

IFOS3D is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with IFOS3D. See file COPYING and/or `http://www.gnu.org/licenses/gpl-2.0.html`.

The Authors of IFOS3D are listed in file `AUTHORS`.

Acknowledgments

We thank the seismic working group of the Geophysical Institute at KIT for constructive discussions.
The development of the code was supported by the Karlsruhe Institute of Technology (KIT), the Deutsche Forschungsgemeinschaft (DFG), the Bundesministerium für Bildung und Forschung (BMBF) and the Wave Inversion Technology (WIT) Consortium.
The code was tested and optimized at the computing centres of the Karlsruhe Institute of Technology (KIT), the supercomputing center in Juelich and the HLRS in Stuttgart.

References

Butzer, S., Kurzmann, A., & Bohlen, T., 2013. 3D elastic full-waveform inversion of small-scale heterogeneities in transmission geometry, *Geophysical Prospecting*, **61**(6), 1238–1251

Butzer, S., 2015. 3d elastic time-frequency full-waveform inversion, dissertation, Karlsruhe Institute of Technology

Contents

Chapter 1

Introduction

In the 1980's Lailly (1983), Tarantola (1984) and Mora (1987) suggested a new inversion strategy known as full waveform inversion (FWI). This method aims to reconstruct multi-parameter images of the subsurface by iteratively minimising the misfit between modeled and observed data. Thus it takes the full information content of the seismograms into account and can offer a significantly improved resolution compared to conventional methods. The optimisation problem is generally solved with gradient-based methods, which can be implemented very efficiently for FWI using the adjoint approach (e.g. Tarantola, 1984; Mora, 1987).
Different approximations are generally used to limit the large number of unknown subsurface parameters and to mitigate the computational costs of the inversion. Many applications are performed in the 2D approximation. This leads to an enormous decrease of the number of subsurface parameters. Still, the 2D approximation is unable to explain 3D scattering and can lead to artefacts in 3D heterogeneous medium. Furthermore the use of a 3D FWI offers the possibility to invert for 3D structures and to gain a 3D image of the subsurface. Often, wave propagation in FWI applications is described with the (visco-) acoustic wave equation. Herby only compressional waves are considered which can be sufficient in marine seismics. However, in land seismics the abundance of shear waves and surface waves favours the (visco-) elastic wave equation. Still, studies on the implementation and application of 3D elastic FWI as performed by Epanomeritakis et al. (2008), Fichtner et al. (2009), Castellanos et al. (2011), Guasch et al. (2012) and Butzer et al. (2013) are rare and computationally expensive.

IFOS3D is a 3D elastic full waveform inversion tool which aims to resolve the elastic parameters (compressional and shear wave velocity and density) of the 3D subsurface. It is based on the conjugate gradient method. For a good computational performance the gradient is calculated with a time-frequency approach. Hereby the wavefields are simulated with the fast and efficient finite-difference forward solver SOFI3D in time domain (Bohlen, 2002). The gradient calculation is then performed in frequency domain for discrete frequencies. A discrete Fourier transform on the fly enables the transformation from time to frequency domain. For an optimisation of the gradient method IFOS3D offers the calculation of a diagonal Hessian approximation and application of the L-BFGS method.
The IFOS3D program is an extension of the SOFI3D forward modeling code for inversion and is thus closely linked to this program. SOFI3D is based on the FD approach described by Virieux (1986) and Levander (1988). The present program SOFI3D (elastic version) has the following extensions

- employs higher order FD operators,
- applies Perfectly Matched Layer boundary conditions at the edges of the numerical mesh (Komatitsch & Martin, 2007),
- works in MPI parallel environment ONLY, i.e. SOFI3D is a implementation based on a domain decomposition (Bohlen, 2002).

The manual of SOFI3D gives a detailed description of the forward modelling solver, its features and the corresponding input parameters.

IFOS3D was successfully tested in different synthetic applications including transmission and surface acquisition geometries (Butzer et al., 2013; Butzer, 2015). A detailed description of the theory and implementation of the program is given by Butzer (2015). This manual will give an overview about the structure and implementation of IFOS3D which is necessary to run the program. The different input parameters are decribed in detail. Additionally IFOS3D comes with a toy example which is described in this guide. It can be performed in a few steps and therefore offers a good access to IFOS3D.

Chapter 2

Theory and implementation

2.1 The inverse problem

Full waveform inversion (FWI) aims to minimise the misfit between modeled and observed data to find a subsurface model **m** which explains the observed data optimally. IFOS3D uses the L_2-norm based misfit function E given by

$$E(\mathbf{m}) = \frac{1}{2}\sum_s \int_0^T dt \sum_r \delta u_i(\mathbf{x}_s, \mathbf{x}_r, t)^2, \tag{2.1}$$

with the i-th component of the displacement residual $\delta u_i = u_i - u_{i,obs}$ at source position $\mathbf{x}_s$ and receiver position $\mathbf{x}_r$. $\mathbf{u}_{obs}$ denotes the observed data, whereas $\mathbf{u}$ denotes the modelled data. The squared residuals are summed up over all sources s and receivers r and integrated over time t for the full record length T. In this and the following equations we use the Einstein notation to sum up over double indices.

The gradient methods

A conjugate gradient approach is applied to solve this optimisation problem. Gradient methods are local inversion methods, which iteratively minimise the misfit by updating the model in the direction of the steepest descent of E. The model in iteration k is updated by

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{p}_k \quad \text{with} \quad \mathbf{p}_k = \mathbf{P}\nabla_m E_k. \tag{2.2}$$

Hereby $\nabla_m E_k$ is the gradient of the misfit function, $\mathbf{P}$ is a preconditioning operator, which mitigates amplitude effects in the gradient, and α_k is the step length. Gradient directions in FWI can be implemented very efficiently with the adjoint approach (e.g. Tarantola, 1984; Mora, 1987) and these methods are thus applicable even for larger problem sizes. Note however, that the misfit function in FWI is generally very rough and contains local minima. In order to reach the global minimum with this local inversion method, a good starting model and the application of inversion strategies, like frequency filtering are essential.
The use of the conjugate gradient $\mathbf{c}_k$ compared to the general gradient $\nabla_m E_k$ can improve the convergence. The conjugate gradient direction $\mathbf{c}_k$ in iteration k is given by

$$\mathbf{c}_k = \mathbf{P}\nabla_m E_k + \beta_k \mathbf{c}_{k-1}. \tag{2.3}$$

This stabilises the inversion by adding some part of the previous gradient. The scalar β is estimated with the Polak-Ribiére method as

$$\beta_k = \frac{\mathbf{p}_k^T(\mathbf{p}_k - \mathbf{p}_{k-1})}{||p_{k-1}||^2} \text{ for } \beta_k > 0; \quad \beta_k = 0 \text{ for } \beta_k < 0. \tag{2.4}$$

The distinction of cases ensures, that the model is updated in the decent direction of the misfit function. The new model is then estimated as

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \alpha_k \mathbf{c}_k. \tag{2.5}$$

2.2 Forward modeling with SOFI3D

Before explaining the inversion process, we will shortly describe the solution of the forward problem. 3D elastic FWI requires a forward solver, which accurately simulates seismic wave propagation in the elastic media. Otherwise artefacts and deviations in the inversion result can be caused by fitting modelling errors in the wavefield. Additionally, a computationally efficient forward solver is required, because the inversion process is characterised by a high number of forward modellings and the FWI runtime is thus mainly influenced by the runtime spend for wavefield modelling.
"IFOS3D" uses the elastic version of the time-domain finite-difference solver "SOFI3D" (Bohlen, 2002) for wavefield modelling, which very efficiently and fast solves the 3D elastic wave equation. The code discretizes the velocity stress formulation of the elastic wave equation on a staggered grid (Virieux, 1986; Levander, 1988). It is able to accurately model wave propagation in 3D complex media. A viscoelastic version of SOFI3D exits, but was not tested for FWI yet. In the following, I will give a short overview of SOFI3D and its characteristics, but refer to the SOFI3D manual for a more detailed description of its theory, implementation and performance.
Wave propagation can be described in the velocity-stress formulation. This formulation uses the particle velocity $\mathbf{v} = \partial \mathbf{u}/\partial t$ as wavefield parameter. For the elastic isotropic medium wave propagation due to a force $\mathbf{f}$ can then be described as first order differential equations:

$$\begin{aligned} \rho \frac{\partial v_i}{\partial t} &= f_i + \frac{\partial \tau_{ij}}{\partial x_j}, \\ \frac{\partial \tau_{ij}}{\partial t} - \lambda \frac{\partial \Theta}{\partial t}\delta_{ij} + 2\mu \frac{\partial \varepsilon_{ij}}{\partial t} &= 0, \\ \frac{\partial \varepsilon_{ij}}{\partial t} &= \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right). \end{aligned} \tag{2.6}$$

The elastic medium is described by the density ρ and the Lamé parameters λ and μ. τ_{ij} denotes the elements of the stress tensor and ε_{ij} denotes the elements of the strain tensor. The trace of the strain tensor is given by Θ.

2.2.1 Finite difference modeling

For the simulations of 3D elastic wave propagation these equations are approximated by finite differences (FD). For the calculation of partial derivatives, the wavefield parameters and model

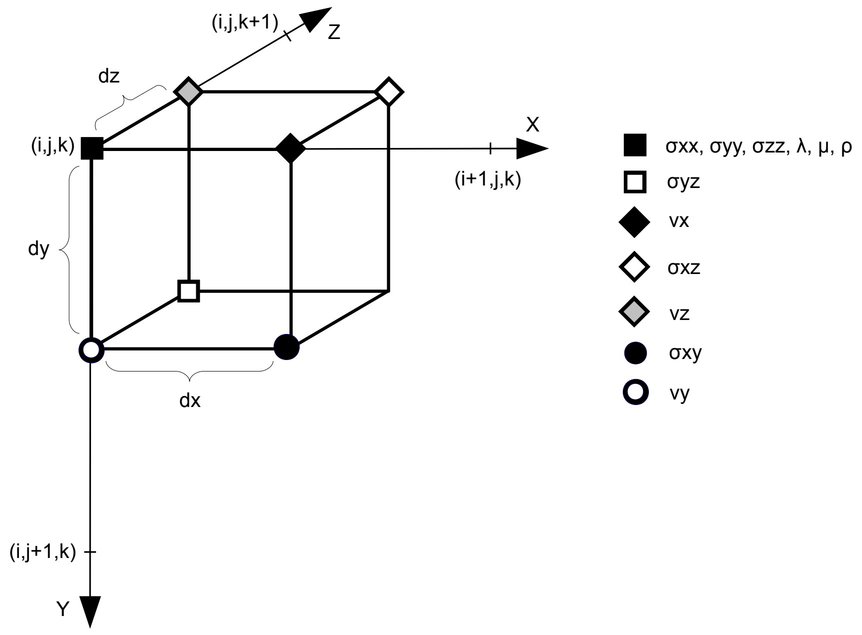

Figure 2.1: Staggered-grid coordinate system used for 3D FD modelling in SOFI3D

parameters are discretised on a staggered-grid system (Virieux, 1986; Levander, 1988). The 3D staggered-grid system is sketched in Figure 2.1. We use a Cartesian grid (x,y,z)=(i,j,k). The model parameters λ, μ and ρ and the diagonal stress components σ_{ii} are localised on the full grid points, whereas velocities and off-diagonal stress components are calculated on a grid which is half a grid point shifted to the original system. Compared to a conventional Cartesian grid, the staggered grid enables a larger grid spacing for the same level of accuracy.

The derivatives are approximated by centered finite differences. The easiest finite-differences are of second order, where the two adjacent grid points with a grid space dh are used to calculate the derivative between these grid points as follows

$$\frac{\partial f(x)}{\partial x}|_{i+1/2} \approx \frac{f[i+1]-f[i]}{dh} \tag{2.7}$$

The consideration of additional grid points in higher order schemes can increase the accuracy and thus enable a larger grid spacing. However, it needs to be considered, that the number of operations increases and the number of parameters which are interchanged during MPI communication rises. Additionally, high order finite-difference operators can cause problems in case of large discontinuities, like tunnels. The SOFI code uses second order temporal and second to 12th order spatial finite difference operators, which can be chosen according to the problem.
To solve the system of equations 2.6 two steps are performed for each time step:

- velocity updates are calculated using spatial stress derivatives and are added to the previous velocity values
- stress value updates are calculated using spatial velocity derivatives and Hook's law and are summed to the previous stress values

Hereby the summation of the update values over all time steps accounts for the time derivatives of velocity and stress values.

2.2.2 Additional charactersistics of SOFI3D

Boundary conditions

There are two ways to avoid arteficial reflections from model boundaries in SOFI3D. The first exponentially damps waves within a boundary zone surrounding the model by multiplying the amplitudes with an exponentially decaying factor (Cerjan et al., 1985). At least 30 gridpoints thickness at boundary zones is required. The second method is the implementation of convolutional perfectly matched layers (C-PMLs) (Berenger, 1994; Komatitsch & Martin, 2007). This method can offer a much better performance compared to the conventional exponential damping and a boundary zone of 10 gridpoints thickness is generally sufficient. However, in case of strong contrasts and heterogeneities at the model boundary instabilities can be caused.

The free surface

At the free surface, the vertical stress components σ_{xy}, σ_{yy} and σ_{zy} are zero. A planar free surface is implemented implicitely in SOFI3D with the mirroring technique described by Levander (1988).

Grid dispersion and stability

To mitigate numerical dispersion and grid anisotropy of the wavefield the grid spacing must be chosen sufficiently small. For a fourth-order spatial and second-order temporal scheme the criterion

$$dh < \frac{\lambda_{min}}{6} = \frac{v_{min}}{6 f_{max}} \tag{2.8}$$

ensures a wavefield error of less than 5% (Bohlen, 2002). It depends on the minimum wavelength λ_{min}, defined by the minimum velocity v_{min} and the maximum frequency f_{max}. Higher order FD-operators enable a larger grid spacing.
In order to ensure stability of the simulation, the temporal spacing must satisfy the Courant-Friedrichs-Levy (CFL) stability criterion (Courant et al., 1967). It is related to the maximum velocity v_{max} and is given by

$$dt <= \frac{dh}{h\sqrt{3}v_{max}}. \tag{2.9}$$

The constant factor h amounts to $\frac{7}{6}$ for a Taylor operator of fourth order. For different FD orders and Holberg coefficients we refer to the SOFI manual.

Viscoelasticity

A viscoelastic version of SOFI3D exists. For its implementation we refer to Bohlen (2002). So far we only employed the elastic version of SOFI3D for IFOS3D. However, it would be possible to include the viscoelastic update functions. This enables the use of viscoelasticity as passive parameter in the inversion process. This could already be successfully tested for the 2D elastic FWI with IFOS2D.

2.3 The inversion process - overview

Figure 2.2 explains the different steps, which are performed within each iteration of the conjugate gradient algorithm in IFOS3D. Each iteration can be divided into three parts. First, the misfit gradient $\Delta E_k(\mathbf{m})$ for each shot and model parameter ($\mathbf{m}$) is calculated. Second, to calculate the search direction ($\mathbf{c}_k$), the misfit is summed up over all shots, preconditioned and the conjugate gradient $\mathbf{c}_k$ is calculated. Third, a step length estimation is performed, which determines the total size of the update. IFOS3D additionally enables the calculation of a diagonal Hessian approximation and the use of the L-BFGS sheme, which will be described in section 2.10 and 2.11, respectively.
The conjugate gradient method is realised with a time-frequency approach (Sirgue et al., 2008). Wavefield modeling is performed in time domain using the finite difference solver SOFI, whereas the gradient is calculated in frequency domain. A discrete Fourier transformation on the fly is applied to calculate the frequency domain wavefields. The main advantage of the frequency domain inversion is the possibility to calculate the gradient only for few discrete frequencies (e.g. Pratt, 1999; Sirgue & Pratt, 2004; Brossier et al., 2009). This results in an enormous reduction of storage costs compared to a time domain inversion, which generally requires the storage of the full forward field. The time-frequency approach additionally allows the use of the efficient and fast 3D time-domain FD forward solver. Thus, for 3D elastic FWI it is recommendable regarding computational performance.

2.4 Gradient calculation

The misfit gradient is calculated with the adjoint method, which can be derived from perturbation theory (e.g. Tarantola, 1984; Mora, 1987). The blue box of the workflow in Figure 2.2 gives a sketch of its implementation and shows the gradient as multiplication of the forward wavefield and the conjugate adjoint wavefield. A detailed derivation of the gradient equations in IFOS3D is given by Butzer (2015). Here, we will only give a short overview about the employed equations and concentrate on the implementation.

2.4.1 Extraction of monochromatic wavefields

For the gradient calculation in frequency domain wavefield parameters of the forward and backpropagated wavefields are required for discrete frequencies. We follow the approach by Sirgue et al. (2008) and use the discrete Fourier transformation

$$\tilde{A}_i(\mathbf{x},\omega) = \sum_{l=0}^{nt} \exp(i\omega l\Delta t) A_i(\mathbf{x}, l\Delta t)\Delta t, \tag{2.10}$$

to extract the monochromatic wavefields, where nt is the number of timesteps, Δt the time sampling and A_i a wavefield parameter. The discrete Fourier transformation is performed on the fly, summing up Fourier summands of A_i in each time step. Therefore, no storage of the time domain wavefields at different time steps is required. Furthermore, the number of additional operations is low, as long as the transformation is only performed for few frequencies. Thus, the change from time to frequency domain can be achieved at low extra costs.

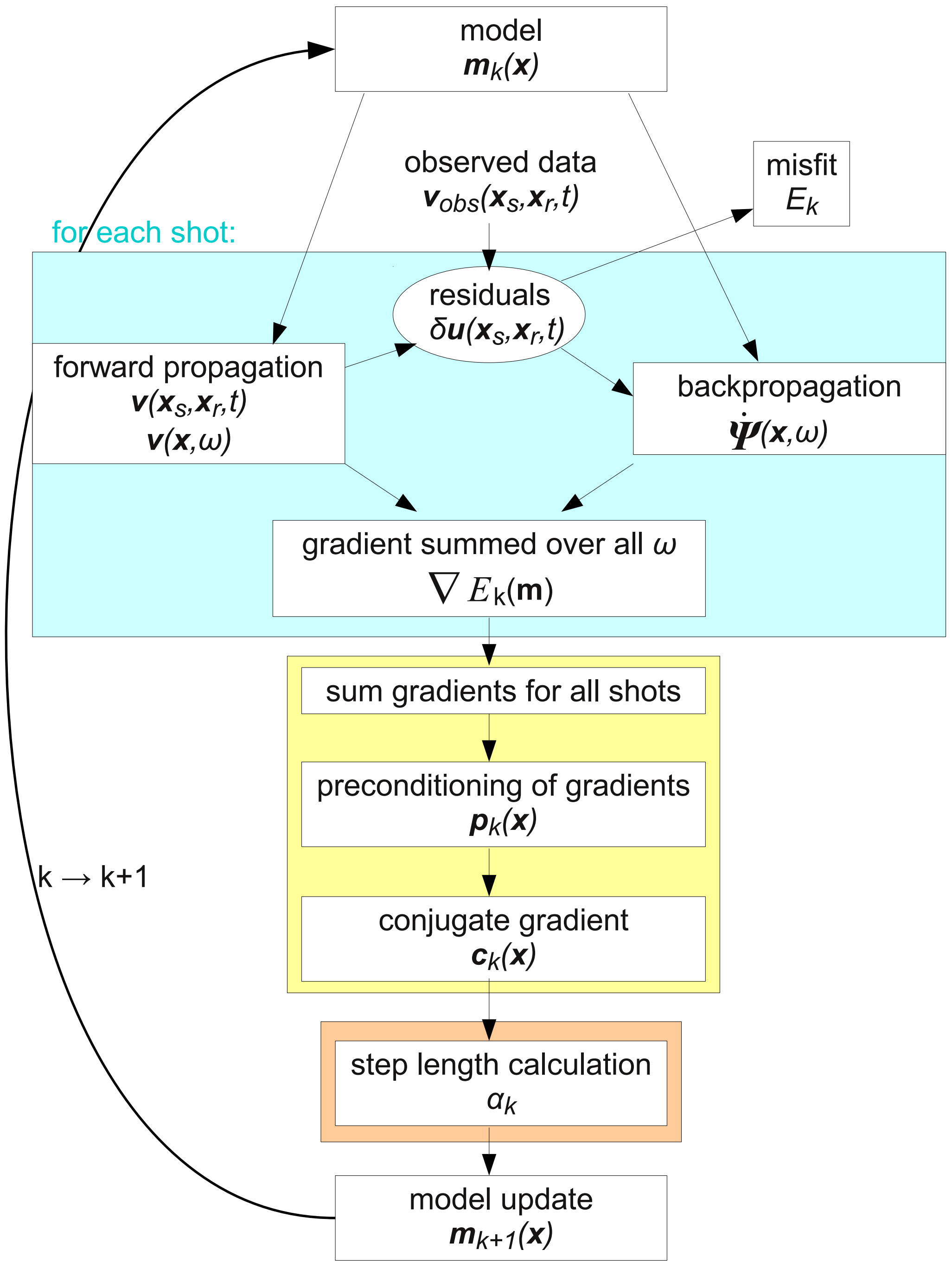

Figure 2.2: The workflow of IFOS3D shows the different steps performed during each iteration.

2.4.2 Wavefield calculation

In a first step, the forward-propagated wavefield for the model $\mathbf{m}_k$ is simulated, which travels from source into medium. At each receiver position the seismograms $v(\mathbf{x}_r,t)$ are stored. Additionally, by applying a discrete Fourier transformation (equation 2.10), we calculate the particle velocities $\tilde{v}(\mathbf{x},\omega)$, which are stored in memory for each discrete inversion frequency and at each grid point.
At each receiver position the time-reversed, residual displacement seismograms $\delta u_i(\mathbf{x}_r, T-t')$ are estimated. This residual is back-propagated from all receivers into the medium simultaneously. The corresponding backpropagated wavefield is defined as

$$\Psi_j(\mathbf{x},T-\tau) = \sum_r \int_0^T dt' G_{ji}(\mathbf{x},\tau;\mathbf{x}_r,t')\delta u_i(\mathbf{x}_r,T-t'). \tag{2.11}$$

Hereby we use the same forward solver, which is based on the first-order wave-equation and therefore calculates the time derivative $\partial\Psi/\partial t = \dot{\Psi}$. For the backpropagation, the time line is reversed and we calculate wavefields starting from time T backwards. Same as for the forward propagation, we extract the frequency wavefields $\tilde{\dot{\Psi}}(\mathbf{x},\omega)$ on the fly. These frequency wavefields are stored in memory for each frequency.

2.4.3 Gradient equations in time and frequency domain

Finally the gradients are calculated as multiplication of the forward and conjugate adjoint wavefields for each frequency and summed up over all frequencies. In the following the corresponding equations are described in more detail.
In time domain the misfit gradients for the elastic parameters λ, μ and ρ can be estimated as zero-lag cross-correlation between forward and backpropagated wavefields as (e.g. Mora, 1987; Butzer, 2015)

$$\begin{aligned}
\frac{\partial E(\mathbf{m})}{\partial\rho(\mathbf{x})} &= -\sum_s \int_0^T d\tau \frac{\partial\Psi_j(\mathbf{x},\tau)}{\partial\tau}\frac{\partial u_j(\mathbf{x},\tau)}{\partial\tau}\\
\frac{\partial E(\mathbf{m})}{\partial\lambda(\mathbf{x})} &= -\sum_s \int_0^T d\tau \frac{\partial\Psi_j(\mathbf{x},\tau)}{\partial x_j}\frac{\partial u_p(\mathbf{x},\tau)}{\partial x_p}\\
\frac{\partial E(\mathbf{m})}{\partial\mu(\mathbf{x})} &= -\frac{1}{2}\sum_s \int_0^T d\tau \left(\frac{\partial\Psi_j(\mathbf{x},\tau)}{\partial x_k}+\frac{\partial\Psi_k(\mathbf{x},\tau)}{\partial x_j}\right)\left(\frac{\partial u_j(\mathbf{x},\tau)}{\partial x_k}+\frac{\partial u_k(\mathbf{x},\tau)}{\partial x_j}\right).
\end{aligned} \tag{2.12}$$

These equations can be transformed to frequency domain. Using Fourier transforms the wavefield in time domain $\mathbf{u}(\mathbf{x},t)$ are transformed into frequency domain $\tilde{\mathbf{u}}(\mathbf{x},\omega)$ and backwards with

$$\tilde{\mathbf{u}}(\mathbf{x},\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\mathbf{u}(\mathbf{x},t)e^{-i\omega t}dt \qquad \mathbf{u}(\mathbf{x},t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\tilde{\mathbf{u}}(\mathbf{x},\omega)e^{i\omega t}d\omega \tag{2.13}$$

Additionally we use that the zero-lag cross-correlation of two real signals A and B in time domain is replaced by a multiplication of one signal with the conjugate of the other signal integrated over the full frequency spectra in frequency domain.

$$\int_{-\infty}^{+\infty} dtA(t)B(t) = \int_{-\infty}^{+\infty} d\omega\tilde{A}(\omega)\tilde{B}^*(\omega) \tag{2.14}$$

This is now applied to the time-domain gradient expressions in 2.12. The time integration $\int_0^T d\tau$ can be extended to $\int_{-\infty}^{+\infty} d\tau$ if the condition of $\mathbf{u}(\mathbf{x},t) = 0$ and $\partial\mathbf{u}(\mathbf{x},t)/\partial t = 0$ for $t < 0$ and $t > T$ is fulfilled. In the density gradient, the time derivatives each transform to frequency domain by multiplication with a factor of $i\omega$. Spatial derivatives are not affected by the transformation. The gradients in frequency domain can then be expressed as

$$\begin{aligned}
\frac{\partial E(\mathbf{m})}{\partial\rho(\mathbf{x})} &= \sum_s \sum_{\nu=1}^{n_f} Re[\omega_\nu^2 \tilde{u}_i(\mathbf{x},\omega_\nu)\tilde{\Psi}_i^*(\mathbf{x},\mathbf{x}_s,\omega_\nu)] \\
\frac{\partial E(\mathbf{m})}{\partial\lambda(\mathbf{x})} &= -\sum_s \sum_{\nu=1}^{n_f} Re\left[\frac{\partial u_p(\mathbf{x},\omega_\nu)}{\partial x_p}\frac{\partial\Psi_j^*(\mathbf{x},\omega_\nu)}{\partial x_j}\right] \\
\frac{\partial E(\mathbf{m})}{\partial\mu(\mathbf{x})} &= -\frac{1}{2}\sum_s \sum_{\nu=1}^{n_f} Re\left[\left(\frac{\partial u_j(\mathbf{x},\omega_\nu)}{\partial x_k}+\frac{\partial u_k(\mathbf{x},\omega_\nu)}{\partial x_j}\right)\left(\frac{\partial\Psi_j^*(\mathbf{x},\omega_\nu)}{\partial x_k}+\frac{\partial\Psi_k^*(\mathbf{x},\omega_\nu)}{\partial x_j}\right)\right].
\end{aligned} \tag{2.15}$$

Hereby we replace the integral over the full frequency spectra by a summation over few discrete frequencies. The use of only few frequencies in contrast to the full spectra is possible in FWI as shown for example by (e.g. Sirgue & Pratt, 2004; Plessix, 2009; Brossier, 2011) as long as the wavenumber spectra is covered continously during inversion. However note that the better redundancy in time-domain FWI can lead to a better performance in case of complex wavefields (Virieux & Operto, 2009).

2.4.4 Implementation of gradient calculation

For each shot the gradients are calculated using the stored forward wavefield $\tilde{v}(\mathbf{x},\omega)$ and back-propagated wavefield $\tilde{\Psi}(\mathbf{x},\omega)$ according to the equations 2.15. Hereby the gradient of ρ can be calculated directly as multiplication of velocity components $\tilde{v}_i$ and $\tilde{\Psi}_i$. To derive displacements $\tilde{u}_i$ and $\tilde{\Psi}_i$ as required for the computation of the gradients of λ and μ, the frequency domain enables a simple integration with

$$\tilde{u}_i(\mathbf{x},\omega) = \frac{1}{i\omega}\tilde{v}_i(\mathbf{x},\omega). \tag{2.16}$$

The spatial derivatives of $\tilde{u}_i(\mathbf{x},\omega)$ and $\tilde{\Psi}_i(\mathbf{x},\omega)$ are calculated with the use of finite differences.

2.4.5 Gradient expressions for the seismic velocity parametrisation

IFOS3D uses the seismic velocity parametrisation for FWI. The relations between the different parameter classes are given by

$$\begin{gathered}
v_p = \sqrt{\frac{\lambda+2\mu}{\rho}} \quad v_s = \sqrt{\frac{\mu}{\rho}} \quad \rho' = \rho \\
\lambda = \rho'(v_p^2 - 2v_s^2) \quad \mu = \rho' v_s^2 \quad \rho = \rho'
\end{gathered} \tag{2.17}$$

For the gradients of v_p, v_s and ρ' we find by applying the chain rule

$$\begin{aligned}
\frac{\partial E}{\partial v_p} &= \frac{\partial E}{\partial \lambda}\frac{\partial \lambda}{\partial v_p} + \frac{\partial E}{\partial \mu}\frac{\partial \mu}{\partial v_p} + \frac{\partial E}{\partial \rho}\frac{\partial \rho}{\partial v_p} \\
&= 2\rho v_p \frac{\partial E}{\partial \lambda} \\
\frac{\partial E}{\partial v_s} &= \frac{\partial E}{\partial \lambda}\frac{\partial \lambda}{\partial v_s} + \frac{\partial E}{\partial \mu}\frac{\partial \mu}{\partial v_s} + \frac{\partial E}{\partial \rho}\frac{\partial \rho}{\partial v_s} \\
&= -4\rho v_s \frac{\partial E}{\partial \lambda} + 2\rho v_s \frac{\partial E}{\partial \mu} \\
\frac{\partial E}{\partial \rho'} &= \frac{\partial E}{\partial \lambda}\frac{\partial \lambda}{\partial \rho'} + \frac{\partial E}{\partial \mu}\frac{\partial \mu}{\partial \rho'} + \frac{\partial E}{\partial \rho}\frac{\partial \rho}{\partial \rho'} \\
&= (v_p^2 - 2v_s^2)\frac{\partial E}{\partial \lambda} + v_s^2 \frac{\partial E}{\partial \mu} + \frac{\partial E}{\partial \rho}
\end{aligned} \tag{2.18}$$

IFOS3D first estimates the gradients of λ, μ and ρ as described in the previous section. The gradients for the velocity parametrisation are then calculated as linear combinations of the Lamé-gradients. Note, that the density gradient depends on the choice of parametrisation.

2.5 Gradient preconditioning

The amplitudes of the gradients are strongly influenced by the geometric amplitude decay of the wavefields. This especially results in high values around sources and receivers. Without preconditioning, the model update would thus focus in the area next to sources and receivers, and the inversion fails. A thorough preconditioning which mitigates geometric effects in the gradients is therefore significant for a successful inversion.
In a local approach we use exponential functions of the form

$$D_1(\mathbf{x}) = (1 + ae^{-br})^{-1} \quad \text{or} \quad D_2(\mathbf{x}) = (1 + ae^{-br^2})^{-1}. \tag{2.19}$$

The parameter r is the distance to the source position ($|\mathbf{x} - \mathbf{x}_s|$) or receiver position ($|\mathbf{x} - \mathbf{x}_r|$). The functions D are characterised by small values adjacent to $\mathbf{x}_s$ or $\mathbf{x}_r$ but approach the value one for large r. Hereby the positive value a defines the minimum value of D whereas b influences the taper radius. In Figure 2.3 the functions D_1 and D_2 are plotted for a constant $a = 1000$ and different values of b. The function D_1 allows a smooth taper over a larger distance, whereas the function D_2 is a sharper taper, which is especially suited for very local tapering. In practise, the latter taper is often suitable to damp receiver artefacts, which generally extend only few grid points, whereas the D_1 taper is useful for the elimination of the more extended source artefacts.
Three types of local preconditioning are implemented in IFOS3D:
- a spherical tapering around source and receiver positions,
- a taper around source and receiver planes,
- a taper of the C-PML boundaries
The tapering of the C-PML boundaries $D_{PML}(\mathbf{x})$ is implemented using the D_2-function, where the distance to the model boundary is used for r. It can be necessary because FWI tends to

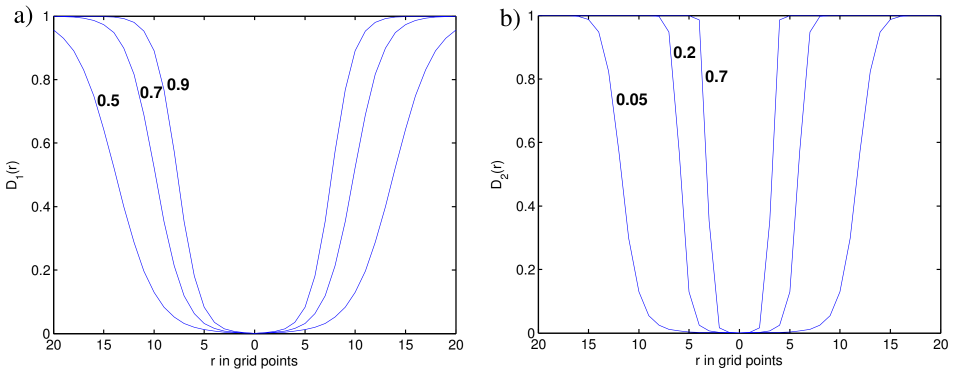

Figure 2.3: Taper functions D_1 (a) D_2 (b) with $a = 1000$ for a source or receiver located at zero distance plotted for different values of b.

produce artefacts in the boundary. Using a source taper D_s and a receiver taper D_r the total preconditioned gradient can be estimated as

$$\mathbf{p}_k(\mathbf{x}) = D_s D_r D_{PML} \nabla E_k. \tag{2.20}$$

2.6 Calculation of the conjugate gradient as search direction

The preconditioned gradient can be used as search direction for updating the model as seen in equation 2.2. However, the use of the conjugate gradient can stabilise the inversion and improve convergence. The preconditioned gradient of the current iteration ($\mathbf{p}_k$) and the preconditioned gradient ($\mathbf{p}_{k-1}$) and conjugate gradient ($\mathbf{c}_{k-1}$) of the previous iteration are used to calculate the conjugate gradient direction according to the equations 2.3 and 2.4. Thus, the application of the conjugate gradient method requires the storage of preconditioned gradient and conjugate gradient of the previous iteration.
The conjugate gradient gives the direction for the model update, but does not contain information about its size (also with respect to other parameter classes). We therefore normalise $\mathbf{c}_k$ to its maximum and multiply it with a reference value of the respective parameter class (v_p^0, v_s^0, ρ^0):

$$c_k^{v_p}(\mathbf{x}) \rightarrow \frac{c_k^{v_p}(\mathbf{x})}{max(c_k^{v_p}(\mathbf{x}))} v_p^0 \qquad c_k^{v_s}(\mathbf{x}) \rightarrow \frac{c_k^{v_s}(\mathbf{x})}{max(c_k^{v_s}(\mathbf{x}))} v_s^0 \qquad c_k^{\rho}(\mathbf{x}) \rightarrow \frac{c_k^{\rho}(\mathbf{x})}{max(c_k^{\rho}(\mathbf{x}))} \rho^0 \tag{2.21}$$

Additionally, a step length estimation is performed as described in the next section.

2.7 Step length calculation

A good step length calculation is crucial for a good performance in FWI. However, run time for its estimation should be low. IFOS3D uses a parabola method (e.g. Kurzmann et al., 2009) to find the optimal step length, which can be achieved at reasonable computational costs. Additionally to the current model misfit the misfit values for two test step lengths are calculated. To mitigate runtime, this is only done for a subset of shots (N_s(step)). The three misfit values are

used to find a parabola. The location of its minimum is adopted as optimal steplength α_k for the model update. This results in $(2 \times N_s(\text{step}))$ additional forward modelings.
Of course, the parabola is only a rough approximation of the real misfit curve. Therefore additional condtitions are employed for a successfull steplength estimation:

1. in case, the parabola extremum is a maximum, I use the test steplength with the smallest misfit
2. a maximum steplength is defined, which cannot be exceeded
3. if a steplength of zero is estimated or the steplength is negative, the model is updated using a small ratio of the first test-steplength

The choice of the test step length is important for the performance of this method. Generally, the test step length can be larger at the beginning of the inversion, when larger model updates determine the rough structure of the subsurface. Later smaller changes of the model are required for the finer model structures. In IFOS3D an initial test step length (TESTSTEP) is chosen and employed in the first iteration within each frequency band. For later iterations the optimal steplength α_k of the previous iteration is used and the new test step length is determined as $\alpha_k/2$. If the new test step length is larger than TESTSTEP, TESTSTEP is used for the next iteration. If the new test step length is smaller than 25% of TESTSTEP, it is set to 25% TESTSTEP.
Note, that one steplength is estimated for all parameter classes. It can be useful to limit the update of one parameter class with respect to another. This can be achieved by applying weighting factors in the model update.

2.8 The model update

Starting from some starting model the model in each iteration k is updated with $(-\alpha_k \mathbf{c}_k)$.
In general, the elastic FWI updates three parameter classes (v_p, v_s and ρ). The conjugate gradients are calculated for each parameter class and are scaled with an average of this parameter to take into account the size of these parameters with respect to ech other. However, note that only one step length is estimated for all parameters. It can be useful to limit the update of one parameter class with respect to another. This can be achieved by applying additional weighting factors in the model update.

2.9 The full inversion process

Full waveform inversion is an iterative process. We search for the optimal model by updating the model parameters in the direction of the steepest descent of the misfit function to reach its minimum. However, the inversion of seismic data is generally highly nonlinear and the misfit function is not a smooth function, but contains a lot of local minima. Using a local inversion method like the conjugate gradient approach, it is possible to end up in a local minimum. In this case, the observed data might be explained relatively well by the inverted model, but the model structures are not correct. To find the real subsurface model, the choice of the starting model and different inversion strategies are important.

2.9.1 The starting model

The choice of the starting model is crucial for FWI. The starting model should contain preliminary information about the subsurface, especially for complex data. It can be useful to use a smooth starting model because hard contrasts of interfaces and structures are difficult to change with FWI. The starting model should already approximately explain the low frequency data so that no cycle skipping problems arise. This means, that the required smoothness of the starting model is dependent on the availability of low frequency data.

2.9.2 Multi-scale inversion

It is very helpful not to invert the whole data set at once, but to start with a portion of data and add more data in different stages of the inversion. There are generally three different types for the selection of data:

1. frequency filtering: The inversion starts with the inversion of low frequency data and subsequently adds higher frequencies,

2. time windowing: first arrivals are inverted as a first step, and more data is added in later stages,

3. offset windowing: seperate long and short offset data

The choice of the inversion strategy depends on the dataset and on the complexity of the problem and a combination of different strategies can be useful. In IFOS3D only the first strategy, the frequency filtering is available at the moment. In the following, we will explain this method in more detail.

2.9.3 Frequency stages

IFOS3D calculates the gradient from few monochromatic frequency domain wavefields. Hereby the inversion is subdivided into different frequency stages, which use a different set of frequencies. It is a common technique in FWI to start from low frequencies, to invert for the larger-scaled subsurface structures and then include higher frequencies to find finer model structures. This approach is well demonstrated by Figure 2.4 from (Bunks et al., 1995). The misfit function is plotted for different scale lengths, which increase from a) to e). With increasing scale, and thus with decreasing frequency, the misfit function becomes smoother and the number of local minima decreases. Therefore, to reach the global minimum, a model can be much farther away from the real model for low frequencies, whereas a much closer model is required for high frequencies. This can also be understood when looking at the data. Phase differences between the modelled and the observed data need to be smaller than half a period, so that the phases can be assigned correctly to each other. Otherwise cycle skipping can occure and the inversion might end up in a local minimum. This condition is easier to fullfill for lower frequencies. Hence, a multi-scale approach in FWI can mitigate the problem of cycle-skipping and help to reach the global minimum.

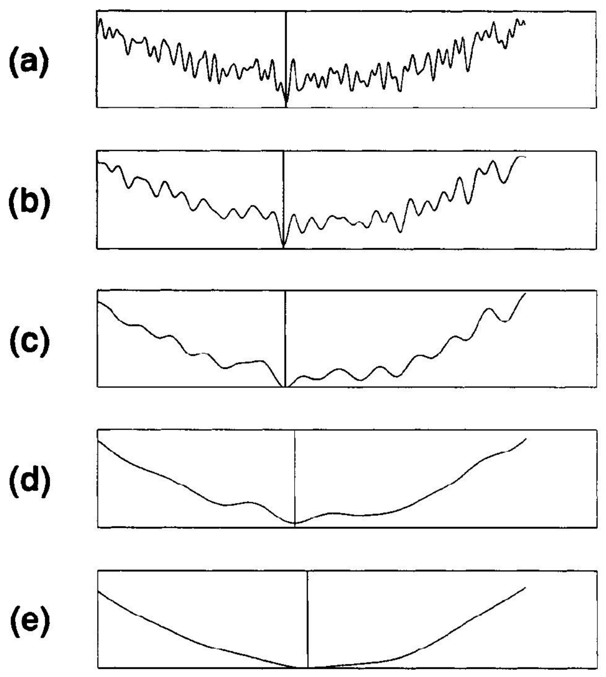

Figure 2.4: Illustration of the misfit function for scale lengths increasing from a) to e) [from: Bunks et al. (1995)].

The choice of frequencies

There are different ways to include higher frequencies in a multi-scale inversion. Brossier et al. (2009) tested three different approaches:

- sequential approach: single frequency inversion using one frequency at a time
- Bunks approach: starting from one low frequency, one higher frequency per stage is added, so that the number of frequencies per stage increases
- simultaneous approach: a group of typically 3-5 frequencies is inverted in each stage, overlapping frequency bands

Overall, they found the best performance for the simultaneous inversion of several frequencies, with overlapping frequency bands for their reflection geometry application. Thus, the entrainment of the lower, already inverted frequencies in the Bunks approach does not seem to improve the inversion. Pratt (1999) also showed an improved performance, when using five instead of three frequencies during one frequency stage. Thus a higher number of frequencies can improve the inversion and a single frequency approach does not seem to be favourable, even though successful applications show that single frequency inversions are possible.
Fortunately, the use of frequency groups with few frequencies per stage instead of a single frequency in a time-frequency FWI does not increase the computational costs by much. By contrast when employing a frequency domain forward solver, these frequencies need to be simulated additionally.

Frequency intervals

The choice of the frequency interval needs to ensure a continuous coverage of the wavenumber spectra. Sirgue & Pratt (2004) showed, that this can generally be fulfilled for frequency intervals larger the sampling interval of $1/T$ (T: length of time series), when considering not only one trace but a range of offsets. Hereby larger offsets enable the coverage of the wavenumber spectra with a lower frequency sampling. For a simple 1D model, the relevant frequencies can be calculated with

$$f_{n+1} = \frac{f_n}{\alpha_{min}}, \tag{2.22}$$

where α_{min} depends on the offset-to-depth ratio and is smaller for larger maximum offsets (Sirgue & Pratt, 2004). Still, the use of additional frequencies increases the redundancy of the data which generally results in improved images, especially in case of higher nonlinearity of the inversion.

2.10 Preconditioning with the diagonal Hessian approximation

In section 2.5 we discussed a very simple method of preconditioning, which locally damps the gradient around sources and receivers. This method works nicely for simple problems as for example often found in transmission geometries. However, in case of complex wavefields this method is not sufficient. If we look at surface geometries, we find that the gradient is very concentrated near the surface, where most energy of the wavefield travels. This means that deeper model parts are not updated without an enhancement by preconditioning. IFOS3D offers the possibility to employ an approximation of the diagonal Hessian for preconditioning. This approach takes information about the second derivative of the misfit function, the Hessian into account and is a physically funded method. The diagonal elements of the Hessian predict the geometrical amplitude spreading contained in the gradient and can thus be used to find an improved spatial scaling of the gradient. In surface seismic, the scaling of the gradient with the diagonal elements of $\mathbf{H}_a$ enhances the influence of deeper parts of the model. In the following we will give an overview about the implementation. For additional information about the role of the Hessian in FWI we refer to Pratt et al. (1998); Virieux & Operto (2009); Brossier (2011) and Butzer (2015).

2.10.1 The Hessian operator

The full Newton method uses the Hessian marix $\mathbf{H}$ for its model update given by

$$\delta\mathbf{m} = -\mathbf{H}^{-1}\nabla_m E. \tag{2.23}$$

In case of a linear inversion problem, an update with $\delta\mathbf{m}$ would lead into the minimum of the misfit function within one step. Non-linear inversion problems, like FWI are linearised stepwise and iteratively approach the minimum. Still the convergence of the Newton method is much improved compared to the gradient methods. The Hessian is the second derivative of the misfit function with respect to the model parameters:

$$H_{jl} = \frac{\partial^2 E(\mathbf{m})}{\partial m_j \partial m_l} \quad (j = 1, ...n) \ \ (l = 1, ...n). \tag{2.24}$$

The variable n denotes the number of model parameters. The size of $\mathbf{H}$ with $n \times n$ is huge and its explicit calculation is therefore very expensive. For large FWI problems like the 3D elastic FWI applications it is not possible. Still the gradient method can be improved by using some information about the Hessian. Comparing the model update of the gradient method (equation 2.2) with the Newton update, the gradient method approximates $\mathbf{H}$ by $\alpha\mathbf{P}$. By preconditioning the gradient with an approximation of the diagonal Hessian it is thus possible to improve the gradient method.

2.10.2 Theory

Calculation of the diagonal Hessian approximation

The second derivative of the L_2-norm based misfit function (equation 2.1) can be expressed as

$$H_{jl} = \sum_s \int_0^T dt \sum_r \frac{\partial u_i}{\partial m_j} \frac{\partial u_i}{\partial m_l} + R_{jl} = (H_a)_{jl} + R_{jl}. \tag{2.25}$$

In the following we neglect the second term R_{jl} which contains second order partial derivative wavefields and constrain to the first term, the so-called “approximate Hessian” $(H_a)_{jl}$. It is calculated as a zero-lag cross-correlation of the first order partial derivative wavefields summed up over all sources s and receivers r.
For preconditoning in IFOS3D we only use the diagonal elements and calculate the elements $(H_a)_{jj}$ for discrete frequencies. Equation 2.25 then transforms to (Butzer, 2015)

$$(H_a)_{jj} = \sum_s \sum_r \sum_{v=1}^{nf} \frac{\partial \tilde{u}_i}{\partial m_j} \left(\frac{\partial \tilde{u}_i}{\partial m_j} \right)^* . \tag{2.26}$$

The adjoint gradient method does not directly calculate the partial derivative wavefields also known as Fréchet derivative kernels which are now required for the estimation of the elements $(H_a)_{jj}$. The derivatives with respect to the elastic parameters λ, μ and ρ in frequency domain can be expressed as (see Butzer (2015))

$$\begin{aligned}
\frac{\partial \tilde{u}_i(\mathbf{x}_r, \omega_v)}{\partial \rho(\mathbf{x})} &= \omega_v^2 \tilde{G}_{ji}(\mathbf{x}, \omega_v; \mathbf{x}_r, 0) \tilde{u}_j(\mathbf{x}, \omega_v), \\
\frac{\partial \tilde{u}_i(\mathbf{x}_r, \omega_v)}{\partial \lambda(\mathbf{x})} &= -\frac{\partial}{\partial x_j} \tilde{G}_{ji}(\mathbf{x}, \omega_v; \mathbf{x}_r, 0) \frac{\partial \tilde{u}_p(\mathbf{x}, \omega_v)}{\partial x_p}, \\
\frac{\partial \tilde{u}_i(\mathbf{x}_r, \omega_v)}{\partial \mu(\mathbf{x})} &= -\frac{1}{2} \left(\frac{\partial}{\partial x_k} \tilde{G}_{ji}(\mathbf{x}, \omega_v; \mathbf{x}_r, 0) + \frac{\partial}{\partial x_j} \tilde{G}_{ki}(\mathbf{x}, \omega_v; \mathbf{x}_r, 0) \right) \left(\frac{\partial \tilde{u}_j(\mathbf{x}, \omega_v)}{\partial x_k} + \frac{\partial \tilde{u}_k(\mathbf{x}, \omega_v)}{\partial x_j} \right).
\end{aligned} \tag{2.27}$$

The terms $\tilde{G}_{ji}(\mathbf{x}, \omega_v; \mathbf{x}_r, 0)$ denote the Green’s receiver functions for discrete frequencies, which are multiplied with the forward wavefields. Same as for the gradient calculation we transform these equations to the seismic velocity parametrisation by using the chain rule

$$\begin{aligned}
\frac{\partial \tilde{u}_i}{\partial v_p} &= 2\rho v_p \frac{\partial \tilde{u}_i}{\partial \lambda}, \\
\frac{\partial \tilde{u}_i}{\partial v_s} &= -4\rho v_s \frac{\partial \tilde{u}_i}{\partial \lambda} + 2\rho v_s \frac{\partial \tilde{u}_i}{\partial \mu}, \\
\frac{\partial \tilde{u}_i}{\partial \rho'} &= (v_p^2 - 2v_s^2)\frac{\partial \tilde{u}_i}{\partial \lambda} + v_s^2 \frac{\partial \tilde{u}_i}{\partial \mu} + \frac{\partial \tilde{u}_i}{\partial \rho}.
\end{aligned} \tag{2.28}$$

Finally these partial derivative wavefields can be used for the estimation of the diagonal Hessian approximation $H_a^{v_p}(\mathbf{x})$, $H_a^{v_s}(\mathbf{x})$ and $H_a^{\rho'}(\mathbf{x})$ (equation 2.26).

Hessian preconditioning

As preconditioning operator ($P^\rho(\mathbf{x})$, $P^\lambda(\mathbf{x})$ and $P^\mu(\mathbf{x})$) for the different parameters we use the inverse of the diagonal of $\mathbf{H}_a$, which is straightforward for a diagonal matrix:

$$\begin{aligned}
P^\rho(\mathbf{x}) &= (H_a^\rho(\mathbf{x}) + \varepsilon_\rho)^{-1}, \\
P^{v_p}(\mathbf{x}) &= (H_a^{v_p}(\mathbf{x}) + \varepsilon_{v_p})^{-1}, \\
P^{v_s}(\mathbf{x}) &= (H_a^{v_s}(\mathbf{x}) + \varepsilon_{v_s})^{-1}.
\end{aligned} \tag{2.29}$$

The coefficients ε_ρ, ε_λ and ε_μ are water levels, which are summed to the Hessian approximations for reasons of stability. Otherwise, very small elements of $\mathbf{H}_a$ in areas of very low wave coverage cause a strong increase of the gradient in these areas which can result in artefacts.

2.10.3 Implementation of Hessian preconditioning

The implementation of Hessian preconditioning in IFOS3D is illustrated in figure 2.5. Note, that the Hessian computation depends on the acqusition geometry and the current model $\mathbf{m}_k$ but does not contain information about the true subsurface model.

Wavefield calculation

The forward propagation of wavefields and the extraction of monochromatic frequency wavefields ($\mathbf{v}(\mathbf{x},\omega)$) is already performed in the framework of the gradient calculation (section 2.4). The main additional computational effort is spend for the computation of the Green's receiver functions $\tilde{G}_{ji}(\mathbf{x},\omega_\nu;\mathbf{x}_r,0)$. Differently to the gradient calculation, where the backpropagated wavefield is propagated from all receivers simultanously, the complete set of Green's receiver functions requires one propagation for each receiver and each component, resulting in $3\times N_{rec}$ additional wavefield simulations (N_{rec}: total number of receivers). We calculate the Green's receiver functions for displacement by using a Heaviside step function (Θ) as a force-time function, defined as

$$\Theta(t - t_{step}) = \begin{cases} 0 \text{ for } t < t_{step} \\ 1 \text{ for } t \geq t_{step}, \end{cases}$$

with the $\delta(t - t_{step})$-function as its derivative. The same time shift t_{step} is used for the forward source. To estimate the Green's receiver functions in frequency domain, we apply a discrete

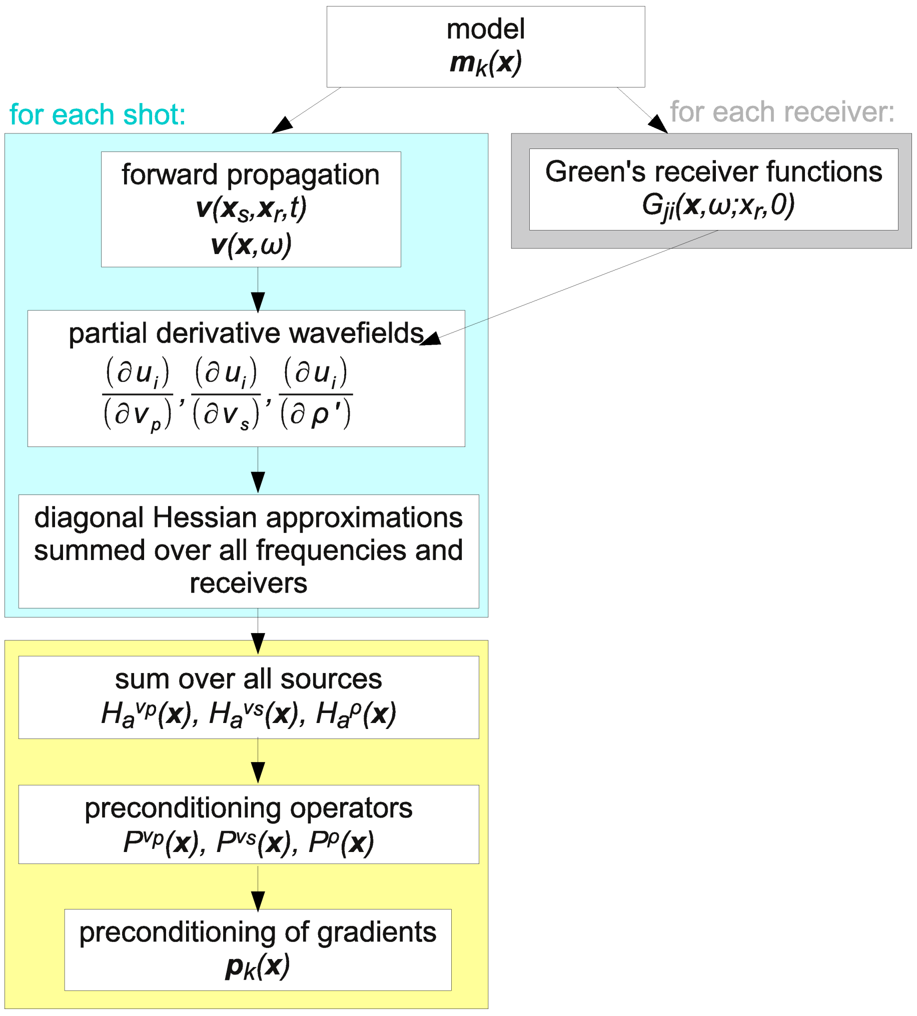
model
$m_k(\mathbf{x})$
for each shot:
for each receiver:
forward propagation
$v(\mathbf{x}_s, x_r, t)$
$v(\mathbf{x}, \omega)$
Green's receiver functions
$G_{ji}(\mathbf{x}, \omega; x_r, 0)$
partial derivative wavefields
$\frac{(\partial u_i)}{(\partial v_p)}, \frac{(\partial u_i)}{(\partial v_s)}, \frac{(\partial u_i)}{(\partial \rho')}$
diagonal Hessian approximations summed over all frequencies and receivers
sum over all sources
$H_a^{vp}(\mathbf{x}), H_a^{vs}(\mathbf{x}), H_a^{\rho}(\mathbf{x})$
preconditioning operators
$P^{vp}(\mathbf{x}), P^{vs}(\mathbf{x}), P^{\rho}(\mathbf{x})$
preconditioning of gradients
$p_k(\mathbf{x})$

Figure 2.5: Workflow for the calculation of the Hessian preconditioning operator at iteration k as implemented in IFOS3D.

Fourier transform (equation 2.10) to the function $G_{ij}(\mathbf{x}_r, t; \mathbf{x}, t_{step})$ on the fly. The functions $G_{ij}(\mathbf{x}_r, \omega_v; \mathbf{x}, 0)$ are stored in memory for each discrete frequency and each receiver. Due to the high computational runtime and storage demand, we employ two approximations here. Firstly, we only use a subset of receivers. Secondly, we only calculate the component of the Green's receiver functions which corresponds to the source of the forward wavefield and thus calculate only one component of the partial derivative wavefield. These limitations have only small effects on the gradient corrections near the source, where preconditioning is most important. However, an additional local preconditioning of the smaller receiver artefacts becomes necessary.

Hessian calculation

For each shot the partial derivative wavefields are computed by multiplying the forward field with each Green's receiver function for each frequency according to equations 2.27 and 2.28. Hereby, the velocities of the forward wavefield are integrated with $(i\omega)^{-1}$ to find the displacement fields and spatial derivatives are calculated with finite differences of fourth order. The partial derivative wavefields are multiplied with their complex conjugate and summed up over all frequencies and receivers and finally over all shots to find the diagonal Hessian approximation for v_p, v_s and ρ (equation 2.26).

Preconditioning

Finally, the preconditioning operator can be calculated according to equation 2.29 and applied to the gradient ($\mathbf{P}(\mathbf{x})\nabla E_k$). At the moment no automatic estimation of the water levels ε is implemented and the water levels are taken from the input file. Thus an empirical try of ε and its effects on the gradient preconditioning is required using the gradient and Hessian of the current iteration.
An additional local damping of the gradient around the receiver positions and in the C-PML boundaries is applied.

The full inversion process

The Hessian preconditioning operator is calculated once at the beginning of each frequency stage. It is applied throughout this stage. Unfortunately, the estimation of the water level ε is not automised at the moment. That means, that in practice before the inversion starts a new frequency stage, the gradient and Hessian are estimated and the water level is determined empirically. This results in as additional calculation of the first gradient of each frequency stage.
Apart from the calculation and application of the Hessian preconditioning operator the inversion process, like gradient normalisation, model update and steplength calculation remains unchanged.

Costs of Hessian preconditioning

The Hessian preconditioning operator is calculated once per frequency stage. The main extra run time is spend for the modelings of the Green's receiver functions which number corresponds to the subset of receivers used for these calculations. Additionally, one extra gradient computation is performed as long as the empirical determination of ε before each frequency stage is required.
In summary, the following approximations are made compared to the use of the full Hessian matrix:

- use of diagonal elements of $\mathbf{H}_a$ only,
- employ only a subset of receivers for its calculation,
- only one component of partial derivative wavefields by backpropagating only one component δ-pulse and
- estimate Hessian only once for each frequency stage.

2.11 The L-BFGS method

This section introduces the L-BFGS (low memory BFGS) method, named after the BFGS-method developers, i.e. Broyden, Fletcher, Goldfarb and Shannon. The method belongs to the class of the quasi-Newton methods and is implemented in IFOS3D as an optimisation of the inversion process. The Hessian operator and Newton methods were shortly explained in the previous chapter. Instead of calculating the inverse Hessian $\mathbf{H}_k^{-1}$ directly, quasi-Newton methods approximate the (inverse) Hessian by using the change of the gradients over the iterations. Starting from an initial Hessian approximation $\mathbf{H}_0$ the approximation of the (inverse) Hessian is updated in each iteration to find a more accurate estimate for the Hessian.
The application of the inverse Hessian to the gradient acts as deconvolution of the gradient from limited-bandwidth effects and geometric amplitude effects (Pratt et al., 1998; Brossier et al., 2009). Previous applications for 2D FWI by Brossier et al. (2009) and Brossier (2011) showed higher convergence rates and sharper images when using the L-BFGS approach compared to the conventional conjugate gradient method. Additionally, a correct inversion of multiple parameter classes can be achieved which requires no empirical weighting of the different parameter classes.

2.11.1 Theory of the BFGS and L-BFGS method

Using the changes of the gradients $\boldsymbol{\gamma}_k = \nabla E_{k+1} - \nabla E_k$ and the model changes $\mathbf{s}_k = \mathbf{m}_{k+1} - \mathbf{m}_k$ the BFGS-condition is formulated as (Nocedal & Wright, 1999)

$$\mathbf{H}_{k+1}^{-1} = \mathbf{V}_k^T \mathbf{H}_k^{-1} \mathbf{V}_k + r_k \mathbf{s}_k \mathbf{s}_k^T \quad \text{with } r_k = \frac{1}{\boldsymbol{\gamma}_k^T \mathbf{s}_k} \quad \text{and } \mathbf{V}_k = \mathbf{I} - r_k \boldsymbol{\gamma}_k \mathbf{s}_k^T. \tag{2.30}$$

The inverse Hessian approximation is updated in each iteration starting from an initial guess $\mathbf{H}_0^{-1}$ without explicitely calculating the second derivatives. However, it is not suitable for a large number n of model parameters due to the huge size of the Hessian matrix ($n \times n$).
By contrast, the L-BFGS method does not explicitely store the Hessian matrix, but uses changes in gradient and model of only the recent m iterations and calculates the inverse Hessian newly in each iteration. Applying equation 2.30 repeatedly for the previous m iterations results in the following formula (Nocedal & Wright, 1999):

$$\begin{aligned}
\mathbf{H}_k^{-1} =& (\mathbf{V}_{k-1}^T ... \mathbf{V}_{k-m}^T) \mathbf{H}_{k0}^{-1} (\mathbf{V}_{k-m} ... \mathbf{V}_{k-1}) \\
&+ r_{k-m} (\mathbf{V}_{k-1}^T ... \mathbf{V}_{k-m+1}^T) \mathbf{s}_{k-m} \mathbf{s}_{k-m}^T (\mathbf{V}_{k-m+1} ... \mathbf{V}_{k-1}) \\
&+ r_{k-m+1} (\mathbf{V}_{k-1}^T ... \mathbf{V}_{k-m+2}^T) \mathbf{s}_{k-m+1} \mathbf{s}_{k-m+1}^T (\mathbf{V}_{k-m+2} ... \mathbf{V}_{k-1}) \\
&+ ... \\
&+ r_{k-1} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^T.
\end{aligned} \tag{2.31}$$

$\mathbf{H}_{k0}^{-1}$ is an initial inverse Hessian approximation, which is allowed to vary from iteration to iteration.

2.11.2 The L-BFGS update

The L-BFGS method can be implemented very efficiently in FWI. In IFOS3D we apply the recursive algorithm described by Nocedal & Wright (1999) which estimates the product $\mathbf{H}_k^{-1} \nabla E_k$

without forming the inverse Hessian approximation directly. It can be written as

```
$\mathbf{q} \leftarrow \nabla E_k$
for $(i = k-1, ..., k-m)$
    $\alpha_i \leftarrow r_i \mathbf{s}_i^T \mathbf{q}$
    $\mathbf{q} \leftarrow \mathbf{q} - \alpha_i \boldsymbol{\gamma}_i$
end for
$\mathbf{z} \leftarrow \mathbf{H}_{k0}^{-1} \mathbf{q}$
for $(i = k-m, ...k-1)$
    $\beta \leftarrow r_i \boldsymbol{\gamma}_i^T \mathbf{z}$
    $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{s}_i(\alpha_i - \beta)$
end for
$\mathbf{H}_k^{-1} \nabla E_k = \mathbf{z}$
```

This algorithm is performed once in each iteration and consists of $(4mn+n)$ multiplications if a diagonal $\mathbf{H}_{k0}^{-1}$ is chosen. The L-BFGS algorithm requires additional storage of $(2mn+m)$ floating numbers for $\boldsymbol{\gamma}, \mathbf{s}$ and r. For reasonable numbers of m the L-BFGS method can therefore be performed at relatively low extra costs of runtime an storage. In general, values between 5 and 20 are used for m. In the first $(m-1)$ iterations the result corresponds to the BFGS method.

2.11.3 The optimised FWI workflow

Hessian preconditioning and L-BFGS

Following the approach by Brossier (2011) we apply the L-BFGS method by using changes in gradients preconditioned by the diagonal Hessian approximation. The Hessian preconditioning (section 2.10) already offers an improved scaling of the gradient. By additionally using the L-BFGS algorithm, the off-diagonal elements of the gradient are taken into account. Furthermore, the L-BFGS method is calculated in each iteration, whereas the diagonal Hessian matrix is estimated only once per frequency stage. When using the preconditioned gradients a scaled identity matrix (Nocedal & Wright, 1999) is sufficient as initial guess for the Hessian:

$$\mathbf{H}_{k0}^{-1} = \frac{\mathbf{s}_{k-1}^T \boldsymbol{\gamma}_{k-1}}{\boldsymbol{\gamma}_{k-1} \boldsymbol{\gamma}_{k-1}^T} \mathbf{I}, \tag{2.32}$$

Parameter normalisation

The conjugate gradient does not offer information about the update of the different model parameters with respect to each other. This is changed when taking the off-diagonal elements of the Hessian into account. The L-BFGS method can thus improve the multi-parameter inversion by adding information about off-diagonal elements of the Hessian. To achieve a dimensionless L-BFGS sheme for the different parameter classes we normalise the inversion sheme with some representative model parameters v_{p0}, v_{s0} and ρ_0 as suggested by Brossier (2011) with

$$\hat{v}_p(\mathbf{x}) = \frac{v_p(\mathbf{x})}{v_{p0}}, \quad \hat{v}_s(\mathbf{x}) = \frac{v_s(\mathbf{x})}{v_{s0}}, \quad \hat{\rho}(\mathbf{x}) = \frac{\rho(\mathbf{x})}{\rho_0}. \tag{2.33}$$

This leads to a normalisation of the gradients calculated as

$$\frac{\partial E}{\partial \hat{v}_p(\mathbf{x})} = \frac{\partial E}{\partial v_p(\mathbf{x})} v_{p0}, \qquad \frac{\partial E}{\partial \hat{v}_s(\mathbf{x})} = \frac{\partial E}{\partial v_s(\mathbf{x})} v_{s0}, \qquad \frac{\partial E}{\partial \hat{\rho}(\mathbf{x})} = \frac{\partial E}{\partial \rho(\mathbf{x})} \rho_0 \tag{2.34}$$

and a normalisation of the Hessian approximation given by

$$H_a^{\hat{v}_p} = H_a^{v_p} v_{p0}^2, \qquad H_a^{\hat{v}_p} = H_a^{v_p} v_{p0}^2, \qquad H_a^{\hat{\rho}} = H_a^{\rho} \rho_0^2. \tag{2.35}$$

The L-BFGS algorithm calculates a normalised model update, which needs to be denormalised by multiplication with the representative model parameters:

$$\delta v_p = \delta \hat{v}_p v_{p0}, \qquad \delta v_s = \delta \hat{v}_s v_{s0}, \qquad \delta \rho = \delta \hat{\rho} \rho_0. \tag{2.36}$$

This dimensionless inversion sheme enables a simultanous L-BFGS algorithm for all parameter classes.

Workflow overview

The workflow of the optimised FWI inversion can be described as follows:

---

for each frequency stage

- calculate diagonal Hessian approximation $(H_a^{\hat{v}_p}, H_a^{\hat{v}_s}, H_a^{\hat{\rho}})$ for normalised parameters

in each iteration

- calculate gradients of normalised parameters
- apply diagonal Hessian approximation to gradient
- save change in preconditioned gradients and models, discard iterations smaller k-m
- L-BFGS algorithm to find normalised model update
- denormalise model update
- step length calculation: try steplength $\alpha = 1$ first
- model update

Newton methods in general calculate an absolute model update and for a linearised Newton method steplenghts close to one are expected. This is also the case for the quasi-Newton L-BFGS sheme. If a good approximation of the Hessian can be achieved a steplength close to one will be favourable. It is possible to apply a steplength estimation as described in section 2.7 with a test step length of 0.5 and 1. If the L-BFGS sheme works well, a steplength close to 1 will be estimated after a few iterations. Note, that in the first iteration the inversion corresponds to a gradient method with Hessian preconditioning.

Chapter 3

Getting started

3.1 Requirements

IFOS3D was tested on different Linux platforms. To run IFOS3D an MPI implementation (one of the three listed below) and a C-Compiler is required. The code was tested on our local workstation cluster using Suse Linux and OpenMPI. Additionally applications were performed on the JUROPA and JURECA supercomputers in Juelich (Germany) and the Hermit supercomputer at HLRS in Stuttgart (Germany). The following programs should be installed on you machine for a proper run and processing of data. These requirements are similar to the forward solver SOFI3D.

Program	Description	Weblink
OpenMPI	MPI Implementation (for the parallelization)	http://www.open-mpi.org
MPICH2	MPI Implementation (for the parallelization)	http://www.mcs.anl.gov/research/projects/mpich2
LAM/MPI	MPI Implementation (for the parallelization)	http://www.lam-mpi.org
C-Compiler	the whole code is written in C, any C-Compiler should be able to compile it	
Seismic Un*x (SU)	Seismic processing package, SOFI3D outputs seismic data in the SU format	http://www.cwp.mines.edu/cwpcodes
Matlab (commercial)	Preferred program package for the visualization of snapshots, also useful for the display and processing of seismograms	http://www.mathworks.de
xmovie	Console based movie program, can be used for the quick display of snapshot data	usually included in Linux distribution otherwise install from repository

3.2 Installation - The folder structure

After unpacking the software package (e.g. by **tar -zxvf ifos3D.tgz**) and changing to the directory ifos3D (**cd ifos3D**) you will find different subdirectories:

bin
This directory contains all executable programs, like ifos3D and snapmerge. These executables are generated using the command **make <program>** (see below).

doc
This directory contains documentation on the software (this users guide).

mfiles
Here some Matlab routines (m-files) are stored. They can be used vor processing and visualisation of data.

par
This directory contains the following folders:

- **in_and_out**: contains in- and outputfiles of IFOS3D parameters
- **sources**: source parameters
- **receiver**: receiver locations
- **su**: seismogram outputfiles
- **su_obs**: observed seismograms
- **model**: model in- and output
- **grad**: gradient output
- **hess**: diagonal hessian in- and output

scripts
Here, you will find examples of script-files used to submit modeling jobs on cluster-computers.

src
This directory contains the complete source codes. The different subprograms are listed in appendix A.

libcseife
The libcseife library serves for filtering seismograms

3.3 Compilation

To compile ifos3D change to the *par*-directory and perform **make**. The Makefile will first compile the external libseife library and then compile the main program. In some cases it's necessary to remove the *.d files in the *libseife*-directory. If problems with the compilation arise go to the *libseife*-directory open *Makefile* and adjust the compiler options for your system.

To change the compiler options open the *Makefile* in *src*-directory. Here you can also find examples for compiler options on different systems where IFOS3D or SOFI3D were used in the past.

3.4 Running IFOS3D

To start the program with OpenMPI you can use the command
mpirun -np 8 nice -19 ../bin/ifos3D ./in_and_out/ifOS3D_toy.json | tee ./in_and_out/ifos3D.out
which runs IFOS3D on 8 processors with lowest priority. The standard output is written to */in_and_out/ifos3D.out*. You can also use the shell-script *par/startIFOS3D.sh*.
For 3D FWI applications it is often useful to use supercomputers in high performance computing centers. Some examples for job scripts can be found in the directory *scripts*.

Chapter 4

In- and output

4.1 The input file - *.json

In the following we will describe the input parameters of IFOS3D. The verbatim text shows, how they appear in *in_and_out/ifos3d_inv_all_parameters.json*. There is also a commented json file: *in_and_out/ifos3d_inv_all_parameters_commented.json*. The parameters used for the wavefield simulation mostly resemble the parameters of SOFI3D and are also described in the SOFI manual.

4.1.1 The grid and its decomposition

The grid

```
"Note that y denotes the vertical direction !" : "comment",
"3-D Grid" : "comment",
"NX" : "160",
"NY" : "184",
"NZ" : "160",
"DX" : "0.8",
"DY" : "0.8",
"DZ" : "0.8",
```

The cartesian grid system is specified by the number of grid points in each direction (**NX, NY, NZ**) and by the distance between grid points **DX, DY, DZ** chosen in the input file. Hereby **NY** and **DY** denote the vertical direction! The grid spacing needs to satisfy the dispersion criterion at least up to the maximum frequency and minimum wavelength used in the inversion. In case the dispersion criterion is violated for the frequency range of the source wavelet, a warning is given at the beginning of the simulation. The dispersion criterions are given in the SOFI3D manual.

Domain decomposition

```
"Domain Decomposition" : "comment",
"NPROCX" : "2",
"NPROCY" : "2",
"NPROCZ" : "2",
```

The parallelisation is based on domain decomposition. Each processor performs the FWI algorithm for a small subvolume of the total grid system. **NPROCX**, **NPROCY** and **NPROCZ** define the number of processors in each direction, which gives a total number of $NPROCX \times NPROCY \times NPROCZ$ processors. Note, that the number of processors in each direction needs to be a common factor of the number of grid points in this direction.

4.1.2 FD-modeling parameters

Order of FD operator

```
"FD order" : "comment",
"FDORDER" : "4",
"FDCOEFF" : "2",
```

The order of the spatial FD-sheme, which is used for the stress and velocity updates can be chosen as **FDORDER**=2, 4, 6, 8, 10, and 12. These orders result in different dispersion and stability criterions (section 2.2.2). A larger FD-order enables a larger grid spacing, however in case of strong heterogeneities a smaller FD-operatot is more accurate. With the option **FDCOEFF** the user can switch between Taylor (**FDCOEFF**=1) and Holberg (**FDCOEFF**=2) FD coefficients. Within the CPML boundary a fourth order operator is applied automatically. We also use fourth order operators to calculate the spatial derivatives of the wavefield velocity, which are needed for the gradient calculation. IFOS3D uses a second order time discretisation.

Time Stepping

```
"Time Stepping" : "comment",
"TIME" : "0.06",
"DT" : "5.0e-05",
```

TIME defines the total simulation length. The time stepping **DT** must be chosen to satisfy the stability criterion (SOFI3D maunual) and thus depends on the grid spacing and the maximum velocity. In case of violation IFOS3D will give a warning and terminate. The number of timesteps of the inversion is given by $TIME/DT$.

4.1.3 Sources

```
"Source" : "comment",
"SOURCE_SHAPE" : "4",
"SOURCE_TYPE" : "4",
"ALPHA" : "45.0",
"BETA" : "45.0",
"SIGNAL_FILE" : "./STF/stf.su",
"SRCREC" : "1",
"SOURCE_FILE" : "./sources/sources_toy.dat",
"RUN_MULTIPLE_SHOTS" : "1",
```

The wavelet

Different wavelets for the modeling can be chosen:

- **SOURCE_SHAPE**=1: Ricker wavelet:
  $s = (1.0 - 2.0\tau^2)\exp(-\tau^2); \quad \tau = \pi f_c(t - 1.5/f_c - t_d);$
- **SOURCE_SHAPE**=2: Fuchs-Müller wavelet:
  $s(t) = \sin(2.0\pi(t-t_d)f_c - 0.5\sin(4.0\pi(t-t_d)f_c);$ for $(t_d < t < t_d + 1/f_c)$ else $s(t) = 0$
- **SOURCE_SHAPE**=4: $\sin^3$-wavelet:
  $s(t) = 0.75\pi f_c \sin(\pi f_c(t-t_d)^3;$ for $(t_d < t < t_d + 1/f_c)$ else $s(t) = 0$
- **SOURCE_SHAPE**=5: step function:
  $s(t) = 0.0;$ if $(t < t_d)$ else $s(t) = 1.0;$

The employed center frequency f_c and the time delay of the source wavelet t_d are defined in the source file. An exemplary plot of the different wavelets and the corresponding spectra can be seen in the SOFI3D manual. Note, that the symmetric Ricker wavelet is delayed and its maximum period is excited at the source location after one period. The step function is used for the calculation of the diagonal Hessian approximation.
If you want to define your own wavelet, you can include this either in the source code in *src/wavelet.c* or use **SOURCE_SHAPE**=3 and read in an external wavelet from **SIGNAL_FILE**. This file should contain the source signal in ASCII with the correct time sampling and with one sample per line, like

```
---------------
0.0
0.01
0.03
...
---------------
```

The source type

You can either choose an explosive source (**SOURCE_TYPE**=1) exciting compressional waves or a point force directed in x, y (vertical) or z (**SOURCE_TYPE**=2,3,4) direction. With **SOURCE_TYPE**=5 it is also possible to define a force in arbitrary direction by using the angles **ALPHA** and **BETA**. This is illustrated in the SOFI manual. The plane wave excitation (**SOURCE_REC**=2) was not yet tested for inversion in IFOS3D but can also be chosen here.

The source location

The source locations are defined in the **SOURCE_FILE** located in the folder *sources*. For each source the parameters XSRC, YSRC and ZSRC are defined, which are the x-, y- and z-coordinates of the source position in meter. Additionally a time delay TD in seconds, a center frequency of the source signal (FC) in Hz and an amplitude of the source signal are chosen. An example file for a source located at (XSRC=100m, YSRC=50m and ZSRC=2m) with a center frequency of 20Hz and no time delay looks like:

——————-

100.0 50.0 2.0 0.0 20.0 1.0

——————-

4.1.4 The model

Model input

```
"Model" : "comment",
"READMOD" : "0",
"MFILE" : "model/toy",
```

The elastic subsurface model is parametrised by the compressional wave velocity v_p, the shear wave velocity v_s and the density ρ at each grid point. Model parameters are required for the forward modeling and as starting model for the inversion. There are two options to read in the model parameters:
READMOD=1: The model parameters of an external model are read in from the model files defined as **MFILE**, here for example *toy.*v_p, *toy.*v_s and *toy.rho* placed in the folder *par/model*. In these files, each material parameter value must be saved as 32 bit (4 byte) native float. Velocities must be in m/s, density values in kg/m^3 defined at each grid point. The values are read in with *src/readmod.c* and you may look at this source code for the correct order of parameters.
READMOD=0: It is also possible to generate the model parameters in the program on the fly. The corresponding C-function, e.g. *src/hh.c* is included in the *src/Makefile*.

Viscoelasticity

```
"Q-approximation" : "comment",
"L" : "0",
"FL1" : "1000.0",
"TAU" : "0.000001",
```

The inversion code was only tested in the elastic version (**L**=0). However, the viscoelasic version of SOFI3D is included in the package and can be used for forward modeling. In this case, the elastic update functions for the velocities need to be replaced by the viscoelastic update functions available in *src*. No inversion code is included for viscoelastic parameters and they can be only used as passive parameters. For further information on viscoelasticity please look in the SOFI manual.

4.1.5 Boundary conditions

The free surface

```
"Free Surface" : "comment",
"FREE_SURF" : "0",
```

A plane stress free surface is applied at the top of the global grid if FREE SURF = 1 using the imaging method proposed by (Levander, 1988). If FREE SURF = 0 a full space is simulated.

Model boundaries

```
"Absorbing Boundary" : "comment",
"ABS_TYPE" : "1",
"FW" : "10",
"DAMPING" : "8.0",
"VPPML" : "6200.0",
"FPML" : "200.00",
"BOUNDARY" : "0",
```

The model includes a boundary layer of **FW** gridpoints at each side, where the wavefield is damped. There are two options included in IFOS3D:
ABS_TYPE=1: IFOS3D offers the use of convolutional perfectly matching (C-PML) layers to damp wavefields in the model boundaries. This technique shows a very good ability to remove reflections from the model boundaries. The parameters **FPML** as the dominant frequency and **VPPML** as the compressional wave velocity near the boundary are employed to calculate the C-PML coefficients. For the C-PML techique a width of **FW**=10 gridpoints is generally enough. Sometimes, instabilities can be created in the PML boundaries, especially in case of strong model heterogeneities in this area. For more information we refer to the SOFI manual.
ABS_TYPE=2: The “traditional“ way to prevent reflections from the model boundary is the exponential damping. This technique is very robust, but less efficient. A thickness of at least **FW**=30 gridpoints is required. For the parameter **DAMPING** you can use about 8%. Note that much higher values cause reflections at the boundary interface.
The periodic boundary condition (**BOUNDARY**=1) is taken from SOFI3D. It was not yet tested in the IFOS3D code and needs to be fully implemented and tested before use.

4.1.6 Seismogram output

```
"Receiver" : "comment",
"SEISMO" : "1",
"READREC" : "0",
"REC_FILE" : "./receiver/receiver.dat",
"REFRECX, REFRECY, REFRECZ" : "0.0 , 0.0, 0.0",
"XREC1, YREC1, ZREC1" : "90.0 , 90.0, 90.0",
"XREC2, YREC2, ZREC2" : "90.0 , 90.0, 90.0",
"NGEOPH" : "1",
```

In every iteration and for each shot IFOS3D stores the seimograms. The parameter **SEISMO** defines the output:

- 0: no seismograms
- 1: particle velocity (*x*-, *y*- and *z*-component)
- 2: pressure
- 3: curl and div

- 4: everything

For the inversion with IFOS3D we generally output particle velocities (one file for each component), which can be used to look at the fit of the waveforms. The inversion of pressure wavefields was not tested with IFOS3D. For more information about the curl and div output we refer to the SOFI manual.

Receiver locations

The receivers define the location of the seismogram output. They are always located on the grid points (no interpolation) and if necessary they are shifted to the nearest grid point. Note, that y defines the vertical direction. There are three options to define receiver locations in IFOS3D:

If the option **READREC**=1 is used, receivers are read from **REC_FILE**. This ASCII file defines one receiver in a row using the $x-$, y- and z coordinates in meter. It is possible to shift the receiver locations by employing a reference point defined by **REFREC**.
The second option (**READREC**=0) defines a straight line of receivers, like a horizontal or vertical profile. This profile is defined by its starting point (**XREC1,YREC1,ZREC1**), its end point (**XREC2,YREC2,ZREC2**) and the distance between the receivers (**NGEOPH**).

```
 "Receiver array" : "comment",
"REC_ARRAY" : "1",
"REC_ARRAY_DEPTH" : "24.0",
"REC_ARRAY_DIST" : "30.0",
"DRX" : "10",
"DRZ" : "10",
```

The last option (**READREC**=2) is very attractive for 3D FWI because it defines seismic arrays which are very useful for 3D synthetic FWI applications. It is possible to define several horizontal arrays (**REC_ARRAY**) in different depths. The upper plane is at **REC_ARRAY_DEPTH** with a vertical distance of **REC_ARRAY_DIST** in meter to the next receiver plane. The horizontal distance between receivers is defined by **DRX** and **DRY**. Note, that the receiver array starts in a distance of 10 gridpoints from the absorbing boundary.

Seismograms

```
"Seismograms" : "comment",
"NDT" : "1",
"NDTSHIFT" : "0",
"SEIS_FORMAT" : "1",
"SEIS_FILE" : "su/IFOS",
```

For the seismogram output every **NDT**th sample is written to file starting at timestep **NDTSHIFT** of the FD modeling. The file format can be choosen with **SEIS_FORMAT**:
1: SU (native 4-byte-floats (IEEE on PC)/little endian on PC)
2: TEXTUAL (native ASCII)

3: BINARY (IEEE-4-byte-floats on PC/little endian on PC)
We recommend the use of the SU-format. This format stores a header in front of each trace with information like time sampling, source and receiver position and tracenumber. Seismic Unix offers different functions for data processing of SU-files.
The seismograms are stored for each component, each shot and each iteration in the folder *su*. **SEIS_FILE** defines the name of the files, here for example *cal_toy_vx_it3.su.shot2*

Snapshots

```
"Snapshots" : "comment",
"SNAP" : "0",
```

It is also possible to store snapshots of the wavefield (**SNAP**>0), that means the wavefield in the whole volume for a series of times. This output is generally not useful in a full inversion due to the great amount of stored data. It can however be used for the forward modeling. It is also possible to save snapshots of the backpropagated wavefield, however in this case the function *snap()* needs to be included in the timeloop of the backpropagation. For the different options to save snapshots and for the processing (merging and visualisation) of the snapshots we refer to the SOFI3D manual.

4.1.7 Forward modeling and inversion

```
"Method" : "comment",
"METHOD" : "0",
```

Using **METHOD**=0 IFOS3D performs a forward simulation similar to SOFI3D. For synthetic tests this option can be used to create observed data of the ”true“ model. For this option the stored seismograms are stored for each processor containing a receiver seperately and as a merged file for all processors. This way the data can be directly used as observed data in the FWI.
With the choice of **METHOD**=1 IFOS3D performs a conjugate gradient FWI as specified by the following parameters in the input file.

4.1.8 General inversion parameters

```
"General" : "comment",
"ITMIN, ITMAX" : "1 , 80",
"FILT" : "1",
"NFMAX" : "5",
"TAST" : "100",
"VP0, VS0, RHO0" : "6200.0, 3600.0, 2800.0",
"WEIGHT_VP,WEIGHT_VS,WEIGHT_RHO" : "1.0, 1.0, 0.0",
```

The parameters **ITMIN** and **ITMAX** define the total number of iterations performed by IFOS3D. Each inversion starts with **ITMIN**=1. However it is possible to split the inversion in parts and use **ITMIN, ITMAX**=(1,15) in a first step and then proceed by choosing **ITMIN, ITMAX**=(16,30).

The gradients are only calculated for discrete frequencies, which works as a natural filter for the inversion with IFOS3D. However, the misfit is calculated in time domain to include the full waveforms. Thus it is reasonable to use filtered seismograms. By choosing **FILT**=1 IFOS3D filters the source wavelet and the observed seismograms below the maximum frequency of the current iteration stage with a Butterworth filter of 4th order. This way the misfit is calculated for data up to the maximum inversion frequency.

The parameter **NFMAX** defines the maximum number of frequencies employed in one inversion stage.

The parameter (**TAST**) defines the time sampling used to extract the monochromatic frequency wavefields via discrete Fourier transformation. The period used for the calculation of the sample rate corresponds to $1/f_{max}$, where f_{max} is the maximum frequency of the current frequency stage. In general it is not necessary to use the small FD time sampling for the discrete Fourier transformation and runtime can be saved. However, when the number of timesteps used for the discrete FD-transformation is too low, the gradient looks pixelated. It is therefore recommendable to check the gradients when using this option.

The parameters **VP0,VS0** and **RHO0** are reference values used for the different parameter classes, like for example the average of the starting model. They are used for the model update of the conjugate gradient method and for the parameter normalisation in the L-BFGS method.

It is possible to employ weighting factors (**WEIGHT**) for each parameter class. A value of 1.0 means, that the parameter is fully updated. A value of 0.0 corresponds to no update of this parameter. This enables an empirical weighting of the different parameter classes.

4.1.9 Observed data input

```
"SEIS_OBS_FILE" : "./su_obs/obs_toy",
"EXTOBS" : "0",
```

The inversion requires observed or measured data as input, which is read in from the folder *su_obs*. These data are required to

- be in SU format
- provide the same time sampling and number of timesteps used in IFOS3D
- be labeled like (**SEIS_OBS_FILE**_vx_it1.su.shot2)

It is possible to choose between internal and external observed data input:
For **EXTOBS**=0 the observed data is data produced by a forward modeling of IFOS3D. In this case, the data is already stored in seperate files for each processor, which can be directly read in by the corresponding processor in the inversion process. The filenames for each processor are

extended by the processor number (e.g.*obs_toy_vy_it1.su.shot4.5*). Note, that the same number of processors must be used in forward modeling and inversion.
For **EXTOBS**=1 the observed data is read in from one file for all processors. In this case IFOS3D splits the data before starting the inversion and saves it to seperate files for each processor. When using this option the order of the seismograms is required to resemble the order of receivers in the receiver file.

4.1.10 Gradients

Gradient output

```
"GRAD_FILE" : "./grad/toy_grad",
"DAMPTYPE" : "2",
```

Gradients of the L2-norm based data misfit are calculated for each parameter class (v_p, v_s and ρ) for each shot and summed over all shots and used frequencies. In each iteration raw gradients are stored for each grid point in the folder *grad* in binary format. The order of the values resembles the order in the model files. Their file name include the label given in the input file (GRAD_FILE), the maximum inversion frequency of this iteration, the parameter class and the iteration number (e.g. *grad/toy_grad.vp_200.00Hz_it6*). Additionally, the conjugate gradients are stored. They are labelled with numbers iteration+2000 (e.g.*grad/toy_grad.vp_200.00Hz_it2006*). It is possible to save additional gradients by including the function *outgrad()* at the required position in the *src/ifos3d.c* source code. This can be useful to look at gradients at single shots or to test a preconditioning function.

Gradient preconditioning

The option **DAMPTYPE** defines a local preconditioning of the gradients at source and receiver positions as described in section 2.5. The following options are avalable:
0: no damping
1: Gaussian taper around receivers
2: Gaussian taper around sources and receivers
3: Gaussian tapering of source and receiver planes
Note, that additionally a taper of the PML boundaries is applied. If the preconditioning is not sufficient for your gradient, it is possible to change the taper parameters in *src/precongrad.c*.

4.1.11 Step length calculation

```
"Steplength estimation" : "comment",
"NSHOTS_STEP" : "4",
"TESTSTEP" : "0.02",
```

The step length calculation is performed using the parabola method as decribed in section 2.7. To avoid long runtimes for the calcultion only a subset of **NSHOTS_STEP** is used for the estimation of the optimal steplength. The parameter **TESTSTEP** defines an initial test steplength used in the first iteration of each frequency stage. In later iterations this test steplength is adjusted to the inversion proceedings. For a conjugate gradient inversion a value of about 0.02 is generally a good choice. This means that for steplength estimation an update of 2% and 4% of the average model parameter is tested and that a maximum model update of five percent is possible.

4.1.12 Model output

```
"MOD_OUT_FILE" : "./model/toy",
```

After model update, the model parameters at each grid point are written to disc into the folder *model*. One file for each parameter (v_p, v_s and ρ) is written in each iteration. The structure of the binary files is similar to the gradient output. The filename is labelled by **MOD_OUT_FILE** (e.g. *model/toy.vs_it6*).

4.1.13 The workflow

```
"INV_FILE" : "./in_and_out/workflow_toy.dat",
```

The workflow file (*in_and_out/workflow.dat*) organises the different inversion stages. An example is given in figure 4.1. One frequency stage is defined in each row. This includes the number of iterations per stage, the number of frequencies and a list of these frequencies. We did not yet include an abortion criteria, so that the number of iterations in one stage is a fixed value and corresponds to the maximum numver of iterations given in the workflow. Note that the maximum number of frequencies used in one stage is defined as **NFMAX** in **.json*. IFOS3D always starts to read the upper row. When an inversion is split up, the upper row must resemble the current frequency stage.
There are some parameters included in the workflow which are useful in FWI, but not yet implemented in IFOS3D. These include parameters for time and offset windowing and an abortion criteria. They will be included in future work.

4.1.14 Hessian

```
"Hessian" : "comment",
"HESS" : "0",
"READ_HESS" : "0",
"REC_HESS" : "1",
"WATER_HESS_VP, WATER_HESS_VS, WATER_HESS_RHO" : "0.0192, 0.0192, 1.0e-14",
"HESS_FILE" : "./hess/toy_hess",
```

If the option *HESS*=1 is chosen in the input file, a diagonal Hessian approximation is applied for preconditioning of the gradients. If the Hessian approximation is already calculated it can be read from file (**READ_HESS**=1). Hereby the hessian approximation is read from the folder *hess*. The files written at the beginning of the current frequency stage are used, labeled for example *hess/toy_hess.vp_200.00Hz_it11* when the new frequency stage started with iteration 11. For **READ_HESS**=0 the diagonal Hessian approximation is calculated in IFOS3D as described in section 2.10.
Due to runtime and storage it is recommendable to use only a subset of receivers for its calculation. The option **REC_HESS** is not fully implemented at the moment, and it is therefore

necessary to change the receiver file or the distance in the receiver array to its double or its triple value.
The water level (**WATER_HESS**) used for the calculation of the preconditioning matrix (equation 2.29) can be estimated from the gradient of the first iteration and the diagonal Hessian matrix. It is not calculated directly in IFOS3D. This results in the fact, that it is not possible to perform a full inversion at a stretch, but the inversion needs to be split up into the different inversion stages.
The diagonal Hessian matrix is stored in the folder *hess* and is labeled
e.g. *hess/toy_hess.vp_200.00Hz_it11*. It is calculated at the beginning of each frequency stage and is applied within this frequency stage.

4.1.15 L-BFGS

```
"L-BFGS" : "comment",
"LBFGS" : "0",
"NUMPAR" : "2",
"BFGSNUM" : "5"
```

If **LBFGS**=1 the L-BFGS method is applied as described in section 2.11. The L-BFGS approach can only be used combined with the Hessian preconditioning (**HESS**=1). So far we tested the L-BFGS approach only for the seismic velocities, not for density. The number of inverted parameter classes (**NUMPAR**) can thus be chosen as 1 (v_p only) or 2 (v_p and v_s).
The L-BFGS algorithm estimates the Hessian from changes in gradients and models of the previous **BFGSNUM** iterations.

4.2 IFOS3D output

4.2.1 ifos3D.out

The proceedings of the inversion can be followed in the file *in_and_out/ifos3D.out* written by the first processor (MYID=0). This file gives information about parameters used in IFOS3D, grid dispersion and stability of the code. The proceedings of the forward modelings and output of the different sub-programs can be viewed. In case of an early termination of the program this file can be checked for an error message.

4.2.2 ifos3D_invers.out

The output of misfit values and steplength information is written to *in_and_out/ifos3D_invers.out* during the run. The following values given for each iteration show the proceedings of the inversion:

- parameters as defined in the current iteration: the parameter **cdf** (set to one for the first iteration in each inversion stage), the lowest frequency used in this iteration and the number of frequencies for this stage.
- misfit value (time domain) for each shot (**L2**), summed over shots (**L2all**) and misfit used for steplength calculation (**misfit[0]**)

- test steplengths used for steplength estimation (**steplength**) and the resulting L2 misfits
- the steplength parabel and possible warnings given by *src/steplength.c*
- the ”optimal“ steplength used for the model update and the estimated misfit for this steplength
- the new test steplength

If problems in the inversion occure this becomes generally quickly visible in its converge and in problems to find an optimum steplength.

```
15  15  0.050   1        0.000   0.110   0.0      500.0   5        160.0 170.0 180.0 190.0 200.0
15  15  0.050   1        0.000   0.110   0.0      500.0   5        200.0 210.0 220.0 230.0 240.0
15  15  0.050   1        0.000   0.110   0.0      500.0   5        240.0 250.0 260.0 270.0 280.0
15  15  0.050   1        0.000   0.110   0.0      500.0   5        280.0 290.0 300.0 310.0 320.0
```

column	description
1	Minimum number of iterations for this row of parameter.
2	Maximum number of iterations for this row of parameter.
3	!!not in use!! Switch to the next input line automatically by using this threshold value
4	!!not in use!! MUTE_SEIS: 0 = disabled, 1 = enabled.
5	!!not in use!! Time-windowing: Begin of the time-window which will be taken into account.
6	!!not in use!! Time-windowing: Length of the time-window.
7	!!not in use!! Offset-windowing: Begin of the offset-window which will be taken into account.
8	!!not in use!! Offset-windowing: Length of the offset-window.
9	Number of frequencies
10+	List of frequencies

Figure 4.1: The file workflow.dat organises the different inverion stages.

Chapter 5

Toy example - the box model

This chapter describes the inversion of a simple toy example with IFOS3D, which is a subdivided box in a homogeneous full space. This example can be performed as a first synthetic application to test IFOS3D on your system. The necessary input files and scripts are included in the IFOS3D folders and you can follow the following guidelines to perform a successfull inversion. Your results can be compared to those given in this chapter.
The 3D inversion is costly even for a simple model. To enable a successfull inversion we therefore recommend the use of a parallel system with a larger number of CPU's as for example found in supercomputing centers. We performed this inversion using 512 CPU's for 7.5 hours. Before starting this toy example, IFOS3D needs to be installed on your system (see chapter 3).

5.1 Inversion setup

5.1.1 Model and grid system

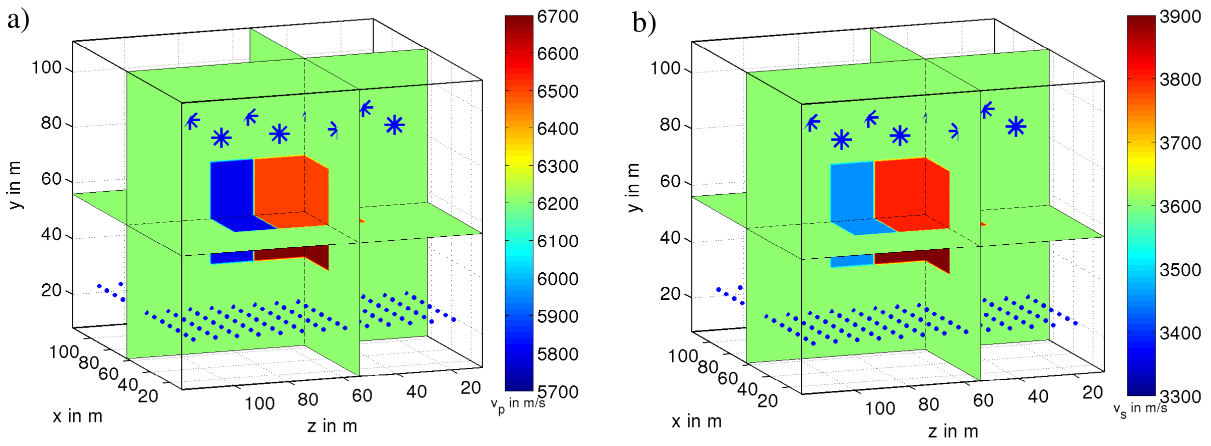

Figure 5.1: The toy example - real model v_p (a) and v_s (b), sources and receivers indicated as stars and crosses, respectively

The true model for the seismic velocities is plotted in figure 5.1. A box divided into four differently-sized parts with different positive and negative velocity variations is placed into a

homogeneous full space of $v_p = 6200\,\mathrm{m/s}$ and $v_s = 3600\,\mathrm{m/s}$. The v_p/v_s ratio is not constant. The density model is homogeneous with $2800\,\mathrm{kg/m^3}$ and kept constant during inversion. The elastic model is generated in IFOS3D on the fly (**READMOD**=0) using the function *src/hh_toy.c* as defined in *src/Makefile*.
The model size is 128 m$\times$128 m $\times$147.2 m in x-, z- and y-direction. We use a grid point distance of **DX=DZ=DY**=0.8 m and thus a grid size of **NX**=160, **NZ**=160 and **NY**=184 points.
We use spatial FD-operators of **FDORDER**=4 and Holberg coefficients (**FDCOEFF=2**). This fullfills the dispersion criterion (see SOFI manual) up to a frequency of 500 Hz ($v_{s,min} = 3450\,\mathrm{m/s}$):

$$dh <= \frac{v_{s,min}}{n f_{max}} = \frac{v_{s,min}}{8.32 f_{max}} \tag{5.1}$$

For our maximum frequency of 320 Hz used for inversion this is well sufficient.
The total calculation time is **TIME**=0.06 s with a time stepping of $5e^{-5}$s resulting in 1200 tiesteps per forward modeling. The stability criterion is fullfilled ($v_{p,max} = 6700\,\mathrm{m/s}$):

$$dt <= \frac{dh}{0.487 v_{p,max}} = 2.4e^{-4} \tag{5.2}$$

The model does not contain a free surface (**FREE_SURF**=0) and as boundary condition we use a PML-layer (**FW**) of 10 grid points width at each model side.

5.1.2 Sources and receivers

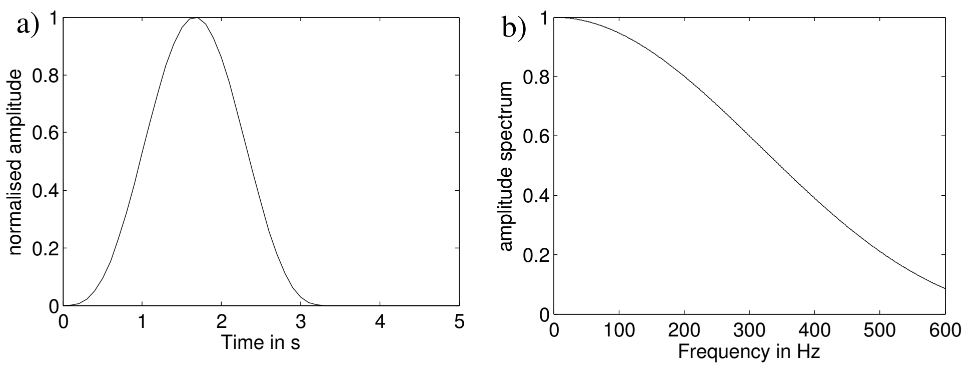

Figure 5.2: The toy example - normalised amplitude and spectrum of $\sin^3$-source wavelet

Sources and receivers are arranged within x-z-planes, as indicated in Figure 5.1. The sources are listed in *sources/sources_toy.dat* (**SOURCE_FILE**). We use 12 (3$\times$4) sources in 92 m depth with a central frequency of **FC**=300 Hz. The sources are vertical directed point forces (**QUELLTYP**=4) with $\sin^3$-wavelets (**QUELLART=4**) as source time functions. The corresponding source wavelet can be seen in figure 5.2a. This source wavelet comprises the low frequencies with 40% of the maximum amplitude left at about 400 Hz (figure 5.2b).
The receivers are arranged using a horizontal receiver array (**READREC**=2, **REC_ARRAY**=1) in 24 m depth. The distance between the receivers is **DRX=DRZ**=10 gridpoints, which gives a total number of 169 receivers.

5.1.3 Inversion parameters

For the inversion we use homogeneous starting models with $v_s = 3600$ m/s and $v_p = 6200$ m/s. In total 60 iterations are performed (**ITMIN, ITMAX** = 1, 60) and five frequencies are employed simultanously (**NFMAX**=5). For preconditioning of gradients a local taper is applied around source and receiver positions (**DAMPTYPE**=2). We do not use the diagonal Hessian approximation (**HESS**=0) or the L-BFGS method (**LBFGS**=0). Out of the total number of 12 shots we use 4 shots for the steplength estimation (**NSHOTS_STEP**=4) and start with an initial test steplength of 0.02 (**TESTSTEP**=0.02). The filenames for gradient and model output are given in the input file.

The workflow

The workflow file *in_and_out* defines the different frequency stages. They are listed in table 5.1.
We used four frequency stages with five frequencies, which increase from stage to stage.

iteration	frequencies in Hz	$\lambda_{min}(v_p)$ in m	$\lambda_{min}(v_s)$ in m
1-15	160,170,180,190,200	31	18
16-30	200,210,220,230,240	26	15
31-45	240,250,260,270,280	22	13
46-60	280,290,300,310,320	19	11

Table 5.1: Frequency stages used for the transmission geometry box example with minimum wavelengths $\lambda_{min}(v_p)$ and $\lambda_{min}(v_s)$.

The frequency bands are slightly overlapping. The table also lists the minimum and maximum wavelengths, which define the resolution of the result. In the first stage lower frequencies enable the reconstruction of rough model structures, whereas higher frequencies and smaller wavelengths result in a higher resolution of smaller model structures.

5.2 Step by step guideline

5.2.1 Step 1 - forward modeling

In a first step, a forward modeling is performed to calculate data of the box model, which can be used as observed data in the inversion. This corresponds to a SOFI3D simulation. The model is created on the fly by the function *src/hh_toy.c*. The parameters are already set in *in_and_out/ifos3D_toy.inp*. For the forward modeling we use **METHOD**=0. The seismograms are written in the folder *su_obs*. For the option **METHOD**=0 the seismograms are written in one file for each processor containing receiver locations. From these files the data can be directly used as input in the inversion. Note, that **FILT**=0 and the program calculates the unfiltered seismograms. The box model is saved in the folder *model*.
To perform the modeling the following steps are applied:

- compile IFOS3D (execute *./compileIFOS3D.sh*)
- adapt processor number in *in_and_out/ifos3D_toy.inp* and your job script (e.g. *startIFOS3D.sh*)

- start IFOS3D (e.g. execute *./startIFOS3D.sh*)

The progress of IFOS3D can be viewed in *in_and_out/ifos3D_toy.out*.

5.2.2 Step 2 - the FWI

In a second step, the inversion is performed. Before starting the inversion perform the following steps:

- set the parameter (kasten=0) in *src/hh_toy.c* to gain homogeneous starting models
- compile IFOS3D (execute *./compileIFOS3D.sh*)
- change the following parameters in *in_and_out/ifos3D_toy.inp*: **METHOD**=1, **FILT**=1, **SEIS_FILE** = ./su/cal_toy
- start IFOS3D (e.g. execute *./startIFOS3D.sh*)

Now the inversion is performed. The proceedings of IFOS3D can be seen in the output-file *in_and_out/ifos3D.out*. An overview how the inversion succeeds, e.g. misfit values and steplengths can be found in the output file *in_and_out/ifos3D_invers.out*. Both files are written during simulation.

5.3 Results

In this section we look at the output of IFOS3D and show the inversion results. All output is located in the folder *par*. For processing and plotting of data we use Matlab programs, located in the folder *mfiles* and Seismic Unix.

5.3.1 Seismograms

Output

The observed seismograms are calculated in step 1- forward modeling. They are located in the folder *su_obs* and provide unfiltered particle velocities for each shot and component in the su-format (e.g. *obs_toy_vx_it1.su.shot3*).
The inverted seismograms are stored in each iteration for each shot and each component in the folder *su*. They are also in the su-format and named for instance *cal_toy_vy_it30.su.shot2*. Note that these data are filtered below the highest frequency used within the corresponding frequency stage.
The su-header in front of each trace contains i.a. information about source and receiver location, time stepping and time samples. It can be viewed using the Seismic Unix function *surange* < FILENAME or the Matlab function *su2matlab*.

Data plot

For a first look, the seismograms can be plotted with the SU-routine *suxwigb* < FILENAME. Note, that it can be necessary to use the function *segyclean* < FILENAME > FILENAME_OUT before plotting.
For a comparison of observed and inverted data, it is necessary to apply a lowpass filter to the observed data, using the maximum frequency of the corresponding stage. Figure 5.3 shows a comparison between initial, observed and inverted seismograms for one source and receiver and x-, z- and y-component of the particle velocity. Note, that the observed data was filtered with the maximum frequency of the first iteration for a comparison in a) and with the maximum inversion frequency for a comparison with the final inverted data in b). The plot was produced with the Matlab program *seismo_trace_toy.m*. This program uses the binary format as input, which is why the files need to be transformed before use, applying the SU-routine *sustrip* < FILENAME > FILENAME_OUT.

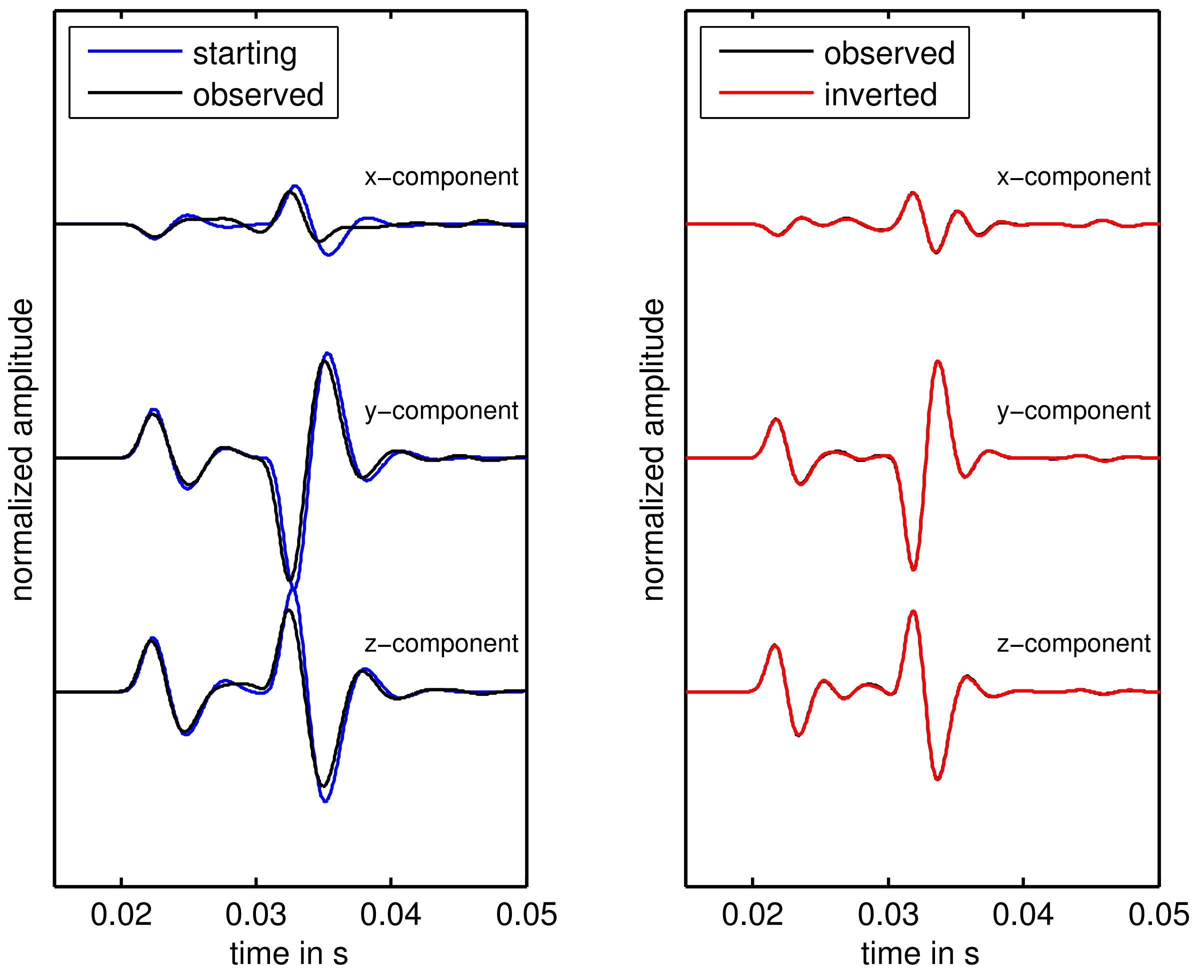

Figure 5.3: Multi-component seismograms exemplarily for source at (x_s, z_s, y_s) = (32 m, 96 m, 92 m) and receiver at (x_r, z_r, y_r) = (48 m, 40 m, 24 m), normalised to one trace in each subfigure. left) initial vs. observed data filtered below 200 Hz and right) observed vs. inverted data filtered below 320 Hz

Discussion

Figure 5.3 shows the comparison of initial and observed data for the low frequencies of the first inversion stage. The waveforms show only small differences and clearly no cycle skipping. Thus the homogeneous starting model is already sufficient for this simple example.
The success of the inversion can be seen when comparing the observed and inverted data. The waveforms, including the small oscillations are fitted very well for all components.

5.3.2 The gradient

Gradients for v_p, v_s and ρ are stored in binary format in each iteration in the folder *grad* (e.g. *toy_grad.vp_200.00Hz_it5*). Additionally to the raw gradients named by iteration number, the conjugate gradients are stored with labels of (iteration number +2000), like (e.g. *toy_grad.vp_200.00Hz_it2005*).
The gradients can be plotted with the Matlab program *slice_3D_toy_grad.m*. This program plots the 3D grid as two perpendicular slices. The acquisition geometry of the toy example is also included.
Here, we show the "raw" gradients and the preconditioned gradients of v_p and v_s normalised to their maximum value for the first iteration (figure 5.4). In the raw gradients (a,b) the high amplitudes around sources and receivers are clearly visible. Especially the source artefacts are very distinct compared to the small scaled receiver artefacts. By preconditioning these artefacts can be removed for the greater part (c,d) and the main update concentrates on the box area. Due to the smaller wavelengths of the shear wave, the v_s-gradient already shows more structure than the v_p-gradient.

5.3.3 Misfit and steplength proceedings

The misfit and steplength values are stored in *in_and_out/ifos3D_invers.out*. They give insights into the proceedings of the inversion. After storing the values one in a row in textfiles, they can be plotted with the Matlab functions *misfit_toy.c* and *steplength_toy.c*.
The misfit proceedings are plotted in figure 5.5. The values are normalised to the initial misfit. The curve shows a high convergence at the beginning of the inversion, where the rough model structures are included. The curve flattens as the inversion proceeds and only finer changes in the model are performed. Note, that with each new frequency stage data is added to the misfit and the misfit steps to higher values, which is then reduced within this stage.
The steplength curve (figure 5.6) shows a similar behaviour. At the beginning steplengths of few percent are found. That means the size of the model update is few percent of the average model parameter. These steplengths quickly decrease to less than 0.5% after few iterations. Most model changes are thus made in the first few iterations whereas only the fine work is performed in the later iterations and at higher frequencies.

5.3.4 The final models

Model parameters are written into the folder *model* in each iteration for each parameter (v_p, v_s, ρ) in binary format (e.g. *toy.vp_it22*). They can be plotted similarly to the gradient with the program *slice_3D_toy_model.m*. Note, that gradient and model files share the same structure.

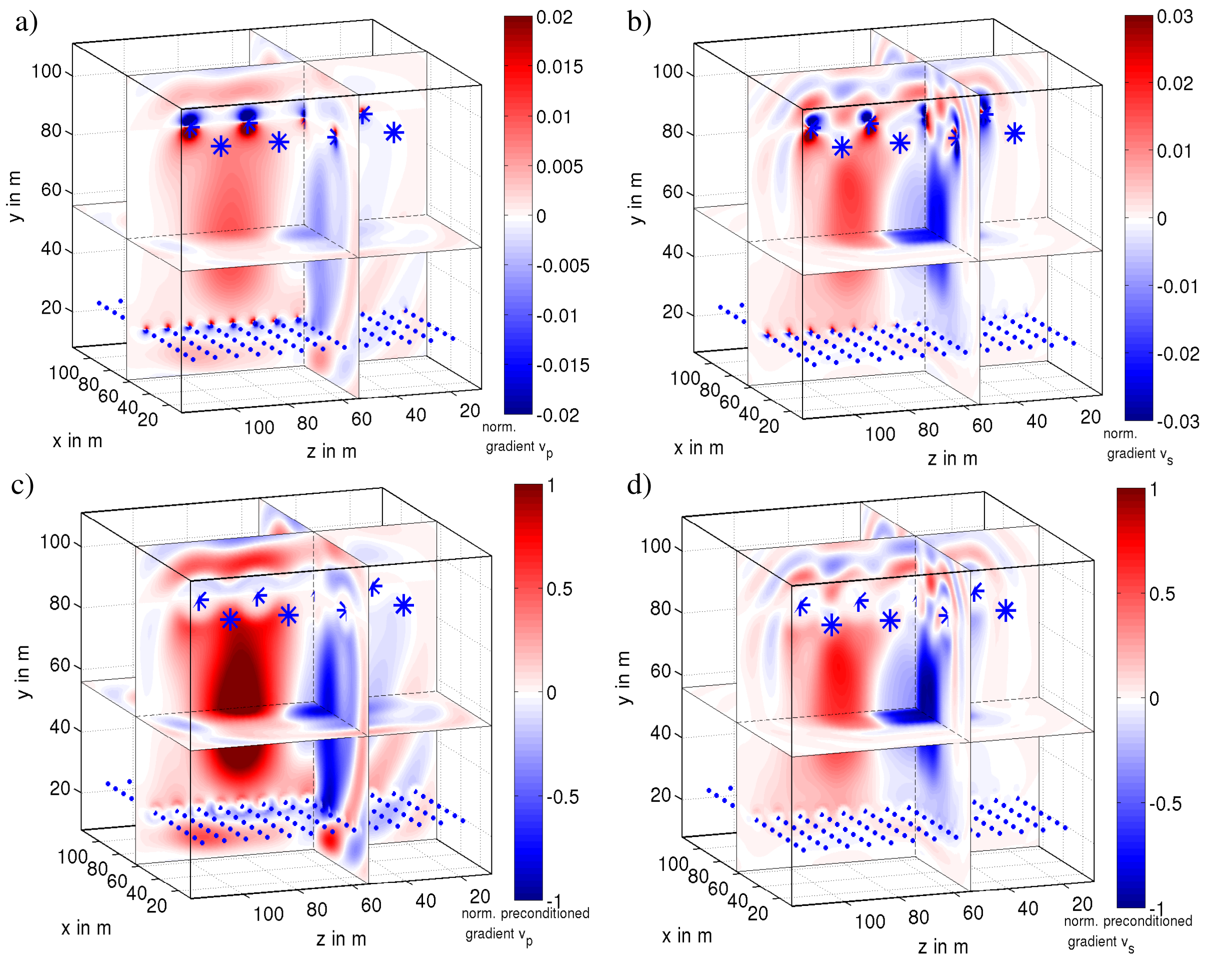

Figure 5.4: Normalised gradients of first iteration : a) “raw” gradient v_p, b) “raw” gradient v_s, c) preconditioned gradient v_p and d) preconditioned gradient v_s; sources (stars) and receivers (crosses) are indicated.

The results for the toy example is plotted for two 2D slices. Figure 5.7 shows a horizontal slice through the box area. The three different box areas seen in the real model in a) and c) are well resolved for v_p (b) and v_s (d) after 60 iterations. However the v_s-model offers a clearly higher resolution due to the smaller wavelengths compared to the v_p-model. In transmission geometry a resolution down to one wavelength is possible and higher frequencies would be necessary to gain a better resolved box in v_p. A vertical slice of the models is plotted in figure 5.8. Overall, the vertical direction is more difficult to reconstruct, as can be ssen in the inverted models of v_p (b) and v_s (d) compared to the real models (a,b). The boundaries of the box are relatively well recovered for both parameters. The inversion of v_p reconstructs the two main areas of the box, however the small high velocity area (dark red) cannot be resolved. This area is indicated in the inverted v_s model, again showing the higher resolution of this parameter.

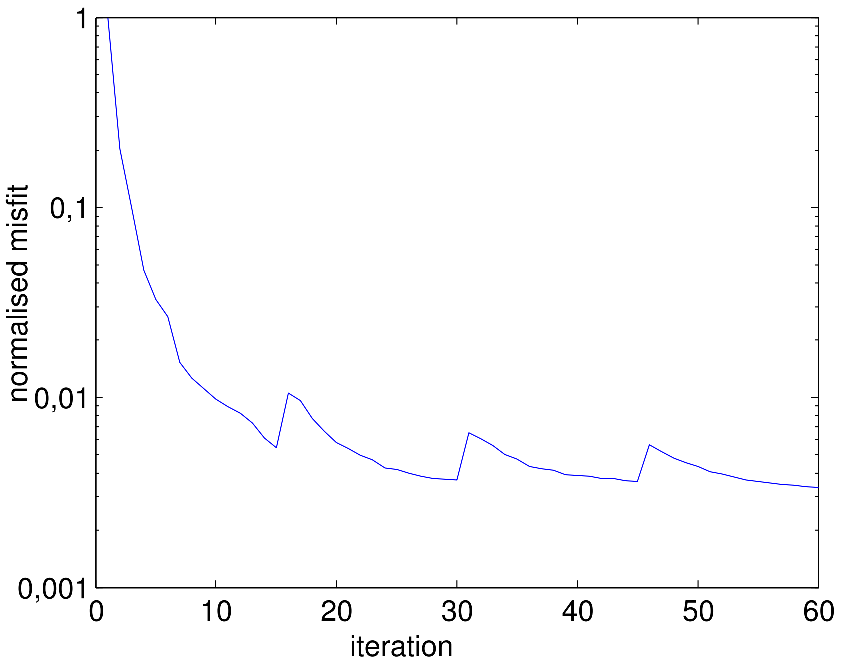

Figure 5.5: Misfit normalised to initial value shows the convergence

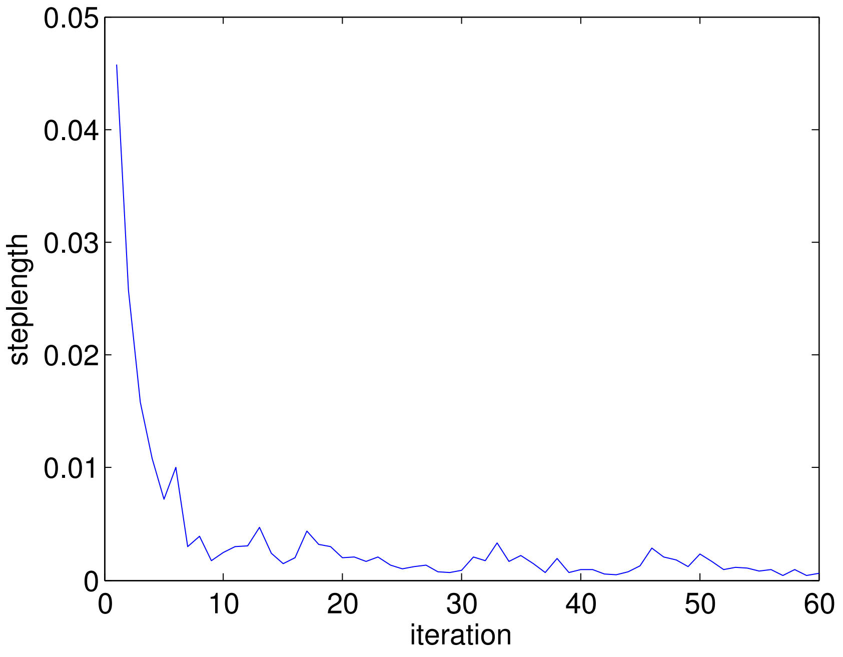

Figure 5.6: Steplength proceedings

5.4 Outlook

This example is also included by Butzer (2015). Here the applications using the diagonal Hessian approximation and the L-BFGS method are shown. You can also look at the other appli-

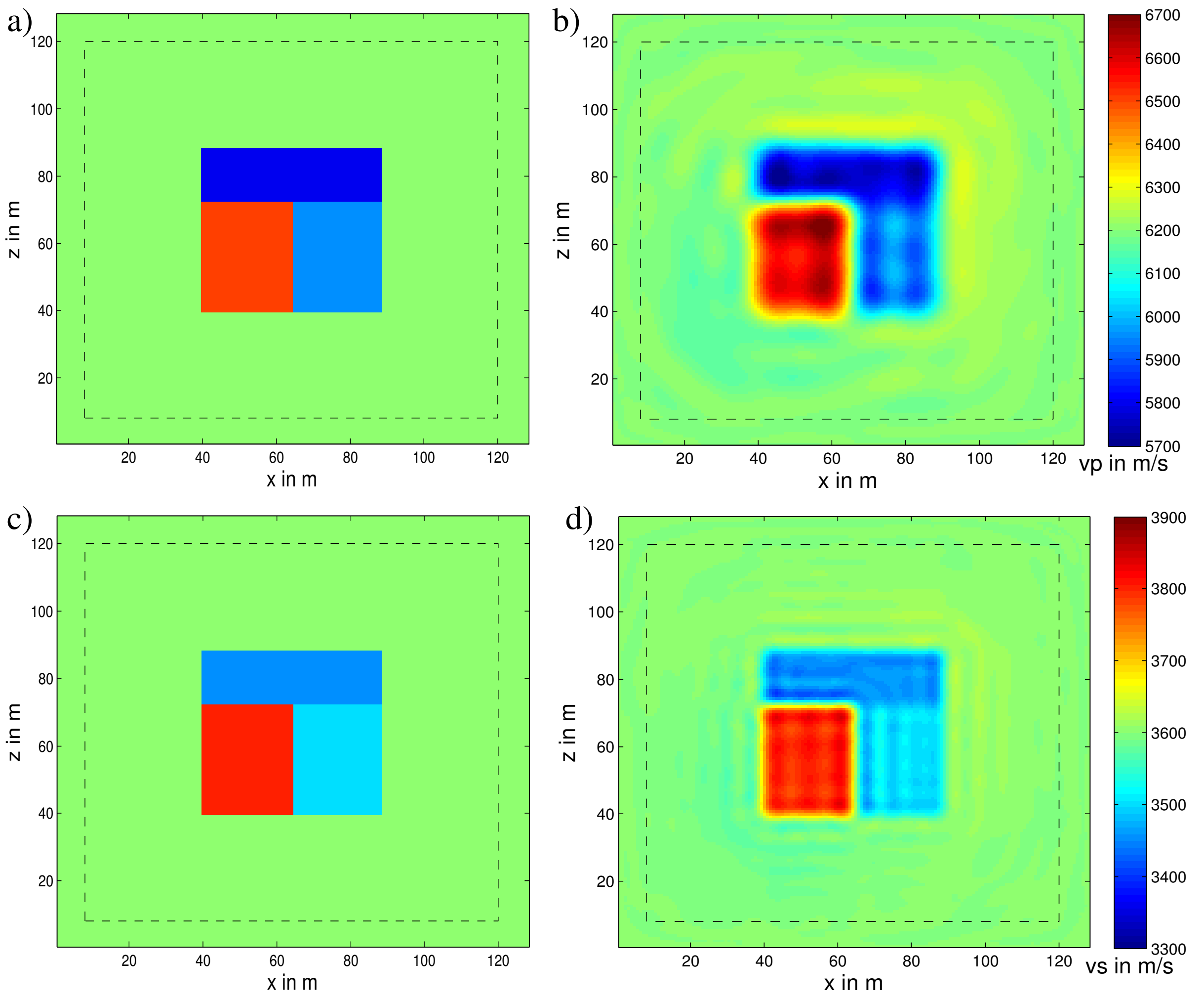

Figure 5.7: Final inverted models (60 iterations) compared to real models for horizontal slice at y=60 m: a) real model v_p, b) inverted model v_p, c) real model v_s and d) inverted model v_s. The dashed line indicates the absorbing frame;

cations (Butzer et al., 2013; Butzer, 2015) which include the inversion of a random medium model in different transmission geonetries and a surface geometry model.

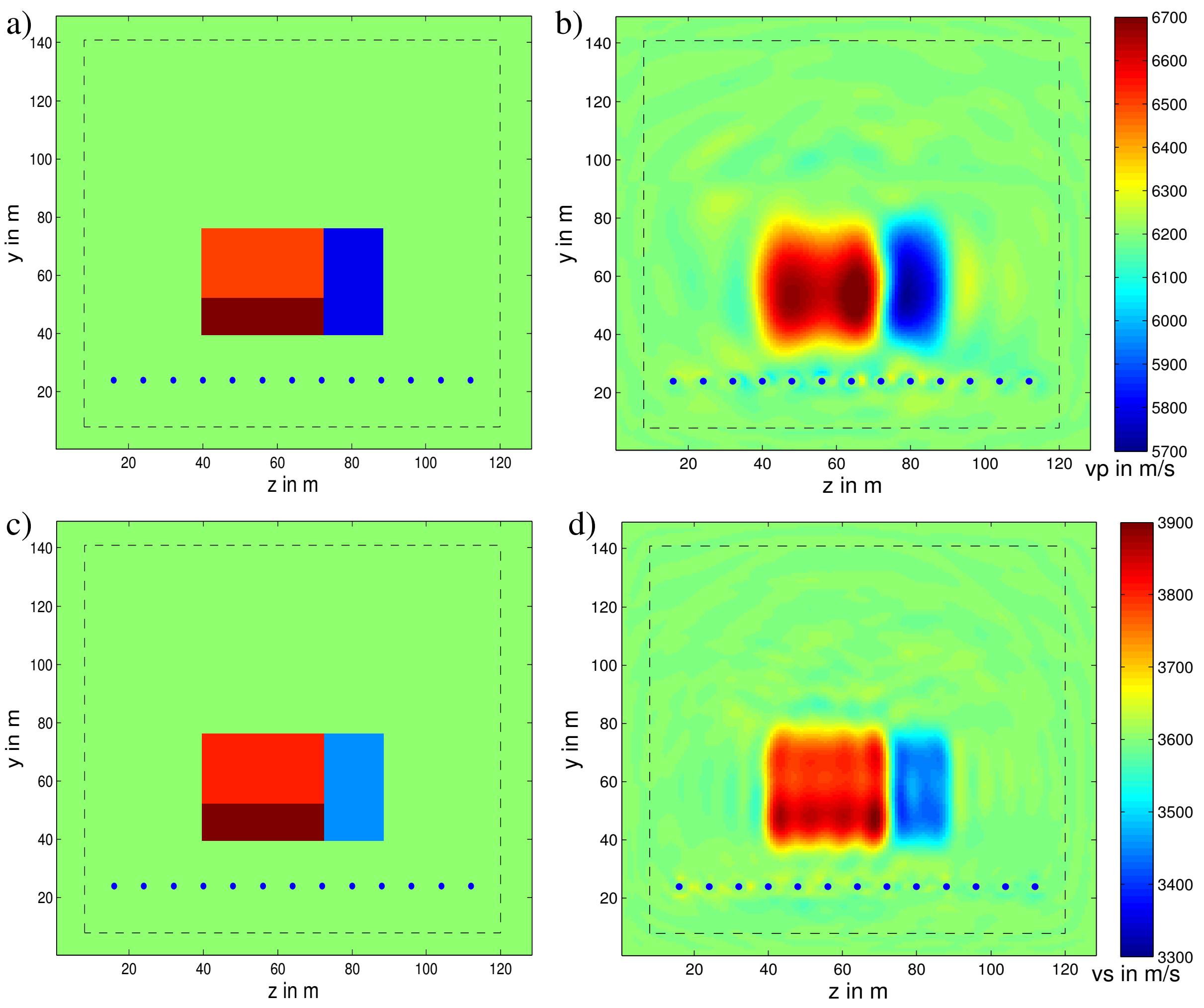

Figure 5.8: Final inverted models (60 iterations) compared to real models for vertical slice at x=56 m: a) real model v_p, b) inverted model v_p, c) real model v_s and d) inverted model v_s;

Bibliography

Berenger, J., 1994. A perfectly matched layer for the absorption of electromagnetic waves, *Journal of Computational Physics*, **114**, 185–200.

Bohlen, T., 2002. Parallel 3-D viscoelastic finite difference seismic modeling, *Computers and Geoscience*, **28**, 887–889.

Brossier, R., 2011. Two-dimensional frequency-domain visco-elastic full waveform inversion: parallel algorithms, optimization and performance, *Computers and Geosciences*, **37**, 444–455.

Brossier, R., Operto, S., & Virieux, J., 2009. Seismic imaging of complex onshore structures by 2d elastic frequency-domain full-waveform inversion, *Geophysics*, **74**(6), WCC105–WCC118.

Bunks, C., Saleck, F., Zaleski, S., & Chavent, G., 1995. Multiscale seismic waveform inversion, *Geophysics*, **60**(5), 1457–1473.

Butzer, S., 2015. 3d elastic time-frequency full-waveform inversion, dissertation, Karlsruhe Institute of Technology.

Butzer, S., Kurzmann, A., & Bohlen, T., 2013. 3D elastic full-waveform inversion of small-scale heterogeneities in transmission geometry, *Geophysical Prospecting*, **61**(6), 1238–1251.

Castellanos, C., Etienne, V., Hu, G., Operto, S., Brossier, R., & Virieux, J., 2011. Algorithmic and methodological developments towards full waveform inversion in 3d elastic media, in *Extended Abstract, SEG Annual Meeting*, San Antonio, 2793-2797.

Cerjan, C., Kosloff, D., Kosloff, R., & Reshef, M., 1985. A nonreflecting boundary condition for discrete acoustic and elastic wave equations, *Geophysics*, **50**(4), 705–708.

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

Epanomeritakis, I., Akcelik, V., Ghattas, O., & Bielak, J., 2008. A newton-cg method for large-scale three-dimensional elastic full-waveform seimic inversion, *Inverse Problems*, **24**, 26pp.

Fichtner, A., Kennett, B., Igel, H., & Bunge, H.-P., 2009. Full seismic waveform tomography for upper-mantle structure in the australasian region using adjoint methods, *Geophysical Journal International*, **179**, 1703–1725.

Guasch, L., Warner, M., Nangoo, T., Morgan, J., Umpleby, A., Stekl, I., & Shah, N., 2012. Elastic 3d full-waveform inversion, in *Extended Abstract, SEG Annual Meeting*, Las Vegas, Nevada.

Komatitsch, D. & Martin, R., 2007. An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation, *Geophysics*, **72**(5), SM155–SM167.

Kurzmann, A., Köhn, D., Przebindowska, A., Nguyen, N., & Bohlen, T., 2009. 2D acoustic full waveform tomography: performance and optimization, in *Extended Abstracts,71th EAGE meeting*, Amsterdam, Netherlands.

Lailly, P., 1983. The seismic inverse problem as a sequence of before stack migrations, in *Conference on Inverse scattering, Theory and application, Society for Industrial and Applied Mathematics*, Philadelphia, 206-220.

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

Mora, M., 1987. Nonlinear two-dimensional elastic inversion of multioffset data, *Geophysics*, **52**(9), 1211–1228.

Nocedal, J. & Wright, S. J., 1999. Numerical optimisation, in *Springer series in operations research*, Springer-Verlag New York.

Plessix, R.-E., 2009. Three-dimensional frequency-domain full-waveform inversion with an iterative solver, *Geophysics*, **74**(6), WCC149–WCC157.

Pratt, R., Chin, C., & Hicks, G., 1998. Gauss-newton and full newton methods in frequency-space seismic wavefrom inversion, *Geophysical Journal International*, **133**, 341–362.

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

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B., 1990. *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Sirgue, L. & Pratt, R. G., 2004. Efficient waveform inversion and imaging: A strategy for selecting temporal frequencies, *Geophysics*, **69**(1), 231–248.

Sirgue, L., Etgen, J., & Albertin, U., 2008. 3d frequency domain waveform inversion using time domain finite difference methods, in *Extended Abstracts, 70th EAGE meeting*, Rome, Italy, F022.

Tarantola, A., 1984. Linearized inversion of seismic reflection data, *Geophysical Prospecting*, **32**, 998–1015.

Virieux, J., 1986. P-sv wave propagation in heterogeneous media: velocity-stress finite difference method, *Geophysics*, **51**(4), 889–901.

Virieux, J. & Operto, S., 2009. An overview of full-waveform inversion in exploration geophysics, *Geophysics*, **74**(6), WCC1–WCC26.

Appendix A

Source code short description

In the following we will give a short overview about the different programs of the IFOS3D source code.

ifos3d - main program.

Makefile - makefile for IFOS3D.

fd.h - include file for IFOS3D.

globvar.h - defines global variables for IFOS3D.

Subprograms

absorb.c - calculation of absorbing boundary coefficients using exponential damping (Cerjan et al., 1985).

av_mat.c - averaging material parameters for the staggered-grid

checkfd_ssg.c - check for stability, grid dispersion and the accessibility of data output directories and files.

comm_ini.c - initialisation of repeated comunications. This may reduce the network overhead.

conjugategrad.c - calculation of the conjugate gradient direction (MORA, 1987).

CPML_coeff.c - defining damping profiles for CPML boundary condition. This C-PML implementation is adapted from the 2nd order isotropic CPML code by Dimitri Komatitsch and based in part on formulas given in Roden and Gedney (2000).

CPML_ini_elastic.c - definition of CPML boundary domains for each submodel.

cpmodel.c - copying of model parameters into a testmodel.

disc_fourier.c - calculation of a discrete Fourier transfomation on the fly: Fouriercomponents of forward or backpropagated wavefields are summed up for each frequency.

exchange_Fv.c - exchange of wavefield velocities (frequency donain) at grid boundaries between processors.

exchange_par.c - exchange of input parameters between processors.

exchange_s.c - exchange of stress values at grid boundaries between processors.

filt_seis.c - filtering of seismograms in time domain with a Butterworth filter of the libseife library. Lowpass or highpass filtering can be applied.

gradient_F.c - gradient calculation in frequency domain: gradient as multiplication of forward and conjugate backpropagated wavefield spatial derivatives are calculated by 4th order finite differences.

hess_apply.c - preconditioning of gradient with diagonal Hessian approximation.

hess_F.c - calculation of diagonal Hessian approximation in frequency domain.

hh.c - generation of an elastic or viscoelastic model specified by v_p, v_s and ρ.

info.c - printing information about IFOS3D.

initproc.c - dividing the 3-D FD grid into domains and assigning the processors to these domains,

lbfgs.c - calculation of L-BFGS update.

lbfgs_save.c - saving gradient for L-BFGS calculation; only first iteration, later saved in lbfgs.c.

matcopy.c - exchange of model parameters to neighbouring processors for the averaging of material properties.

merge.c - merge snapshots files written by the different processes to a single file.

mergemod.c - merge model files written by the different processes to a single file. Used for gradients and model output in IFOS3D.

model2_5D.c - creation of a 2.5D model from a 3D model, (attention: not in parallel!!); 2.5D model parameters constant in z-direction.

modelupdate.c - update of model for next iteration.

note.c - writing note to stdout.

outgrad.c - output of gradients to GRAD_FILE.

outmod.c - output of model parameters v_p, v_s and ρ to MOD_OUT_FILE.

output_source_signal.c - output source signal e.g. for cross-correlation, deconvolution or comparison with analytical solutions.

outseis.c - writing seismograms to disk.

precongrad.c - gradient preconditioning around sources, receivers and at model boundaries.

psource.c - generation of explosive source at source nodes.

rd_sour.c - reading external source wavelet.

read_checkpoint.c - reading wavefield from checkpoint file.

readdsk.c - reading one single amplitude from file.

readhess.c - reading Hessian from files.

readinv.c - reading inversion parameters from workflow.

readmod.c - reading elastic model properties (v_p, v_s, density) from file.

read_par.c - reading FD-Parameters from input-file.

readseis.c - reading seismograms from files.

receiver.c - finding global grid positions for the receivers.

residual.c - calculation of data residuals (displacement) and L2 norm.

rwsegy.c - reading and writing SEG-Y, SU, BIN, TXT and UKOOA P190.

save_checkpoint.c - saving wavefield to checkpoint file.

saveseis.c - writing seismograms to files.

segy.h - include file for SEGY traces.

seismerge.c - merging SEG-Y files.

seismo_ssg.c - storing amplitudes (particle velocities or pressure) at receiver positions in arrays.

smooth.c - smoothing model parameters (complete model or boundaries only), not in parallel.

snapmerge.c - loop over snapshotfiles which have to be merged.

snap_ssg.c - writing 3D snapshot for current timestep to disk.

sources.c - reading source parameters from source file.

splitrec.c - computation of local receiver coordinates (within each subgrid).

splitsrc.c - computation of local source coordinates (within each subgrid).

steplength.c - steplength calculation using a parabola method.

surface_ssg.c - stress free surface condition using the mirroring technique.

surface_ssg_elastic.c - stress free surface condition, elastic case.

timing.c - output timing information (real time for updates etc.).

update_s_ssg.c - updating stress values by a staggered grid finite difference scheme of nth order accuracy in space and second order accuracy in time (viscoelastic version).

update_s_ssg_CPML.c - updating stress values in the CPML-boundaries (4th order spatial FD sheme) (viscoelastic version).

update_s_ssg_CPML_elastic.c - updating stress values in the CPML-boundaries (4th order spatial FD sheme) (elastic version).

update_s_ssg_elastic.c - updating stress values by a staggered grid finite difference scheme of nth order accuracy in space and second order accuracy in time (elastic version).

update_v_ssg.c - updating velocity values by a staggered grid finite difference scheme of nth order accuracy in space and second order accuracy in time.

update_v_ssg_CPML.c - updating velocity values in the CPML-boundaries (4th order spatial FD sheme).

util.c - some utility-routines from numerical recipes (Press et al., 1990).

wavelet.c - Calculating source signal at different source positions for input source parameters.

writedsk.c - writing one single amplitude on disk.

writemod.c - writing local model to file (not in use, see outmod).

writepar.c - writing FD-Parameter to output file stdout or log-file.

zero_grad.c - initialise gradient with zero.

zero_invers.c - initialise wavefield (frequency domain) with zero.

zero_wavefield.c - initialise wavefield with zero.