

# Full Waveform Inversion

ALTEN Netherlands  
MELISSEN, S.T.A.G., version of November 21, 2018

April 15, 2019

# Contents

<b>1 Preface</b>	<b>3</b>
<b>2 Introduction</b>	<b>4</b>
<b>3 The Model</b>	<b>6</b>
3.1 Physical model . . . . .	6
3.1.1 The Green's function . . . . .	6
3.1.2 Helmholtz equation . . . . .	7
3.1.3 The Contrast function . . . . .	7
3.2 Finite Difference Approach . . . . .	8
3.2.1 First Order ABC . . . . .	8
3.2.2 Discretization . . . . .	9
3.2.3 Perfectly Matched Layers . . . . .	9
3.2.4 Discretization . . . . .	10
3.2.5 Source Implementation . . . . .	10
3.2.6 Input Card Parameters . . . . .	11
3.2.7 Python Scripts . . . . .	11
3.2.8 Notes and Future Work . . . . .	11
3.3 Solver and Noise . . . . .	12
3.3.1 The Conjugate Gradient (CG) Scheme . . . . .	12
3.3.2 Multiplicative Regularisation . . . . .	14
3.3.3 Nonlinear field update based on the domain equation . . . . .	14
<b>4 Implementation</b>	<b>15</b>
4.1 Input parameters . . . . .	15
4.2 Subroutines . . . . .	16
4.2.1 main.cpp . . . . .	16
4.2.2 temple_inversion . . . . .	16
4.3 Parallelization: from a single CPU to Multiple and GPU . . . . .	17
<b>5 Downloading, Installing and Running the Code</b>	<b>17</b>
5.1 Pre-requisites . . . . .	17
5.2 Cloning the Repository . . . . .	17
5.3 Build/Run . . . . .	18
<b>6 Results and Analysis - example: Temple Reservoir</b>	<b>18</b>
<b>A Appendices</b>	<b>23</b>
A.1 Derivation of the functional derivative of the error functional . . . . .	23
A.2 Prefactors of the Total Cost Equation . . . . .	23
<b>B Absorbing Boundary Conditions</b>	<b>24</b>
B.1 Second Order ABC . . . . .	24
B.1.1 Discretization . . . . .	24

## 1 Preface

This document describes a method to predict the size and shape of unexplored oil reservoirs based on acoustic testing that ALTEN developed based on Peter Haffinger's Ph.D. thesis (TU Delft 2013). The current version is in progress and written by S. Melissen, under the supervision of André Prins and Bernhard Righolt. It follows a pre-version by Nikita Mahto. The development currently takes place on the "serial"-branch of the git repository, where the parallelization and non-C++ code is being completely split out to have a comprehensible, single CPU version that works on a variety of machines in a more or less "plug-and-play" manner. The Master branch functions relatively well (see "Results"). This branch will be split out into a development branch, where the parallelization introduced in the Master branch is cleaned up completely and clearly split out as one option upon several computational methods. The Docs branch - where this document will be posted - is supposed to clear up documentation issues, the code being in a functioning, albeit very confusing state.

## 2 Introduction

The Full Wave inversion is an advanced imaging technique which can be achieved by irradiating the interior of an object with e.g. acoustic or electromagnetic waves, while receivers placed around the object measure the response. From these measurements an image of the properties of the object's interior can be derived.

It has various applications including,

1. seismology
2. medical: ultrasonic applications

Figure 1 displays the perspective views of Axial Volcanos internal structure, constructed using elastic full waveform inversion (FWI) results along seismic lines across caldera and along the secondary magma reservoir. (a): P-wave velocity structure. (b): Total gradient magnitude of the P-wave velocity structure. (c) Reflectivity structure. The red mesh marks the extent of the main magma reservoir on (a) and (b).

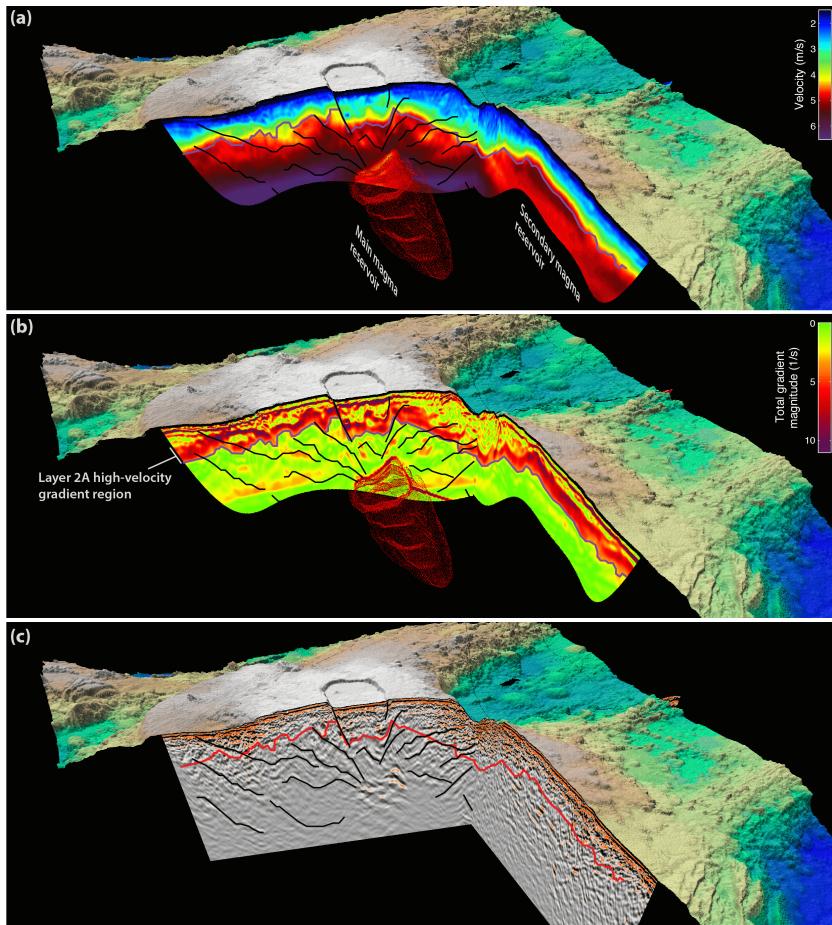


Figure 1: Seismic Full Waveform Inversion

In FWI, the aim is to find the media properties on a dense subsurface grid. It is important to mention that unlike the real applications in which the recorded data by the receivers are provided, in this project only the expected image (indicating the media properties) is provided while the recorded data by the receivers (pressure field) are not available. Therefore in the first step, the recorded pressure field should be generated, knowing the expected media properties (for example, the input image of the temple) and a forward model which is explained in this report. This part has been developed in a separate *preProcessing* application. After generating the recorded pressure field, the same forward model has been used to reconstruct the expected media properties (input image). First the initial estimated pressure field is initialized by an arbitrary value of media properties in the existing forward model. Afterwards, the difference between the estimated pressure field and the recorded data (objective function) are calculated. In an iterative process, the idea is to minimize this objective function (residual) by updating the unknown media property values using any optimization technique. Finally, we expect to ideally estimate the expected values of the media properties. This part is implemented in the *processing* application.

Figure 2, shows a flowchart of the FWI project. In future, by having the access to real recorded data by the receivers, the developed *preProcessing* application (orange blocks in the figure) should be replaced by just reading the measured data.

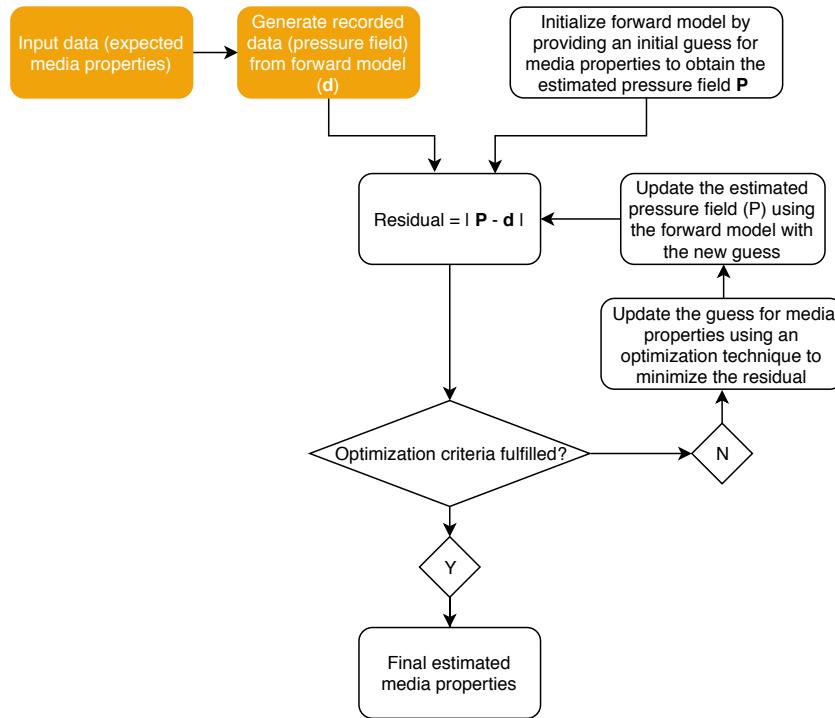


Figure 2: Full Waveform Inversion Methodology

This document provides an overview of the equations employed, the computational model and the code algorithm. The approach is based on the Ph.D. thesis by Peter R. HAFFINGER (TUDelft 2013) "Seismic Broadband Full Waveform Inversion by shot/receiver refocusing"

## 3 The Model

### 3.1 Physical model

The mathematical model outlines the equations used in the code for the FWI. We begin with the Green's function and apply it to the simplest acoustic equation - the Helmholtz equation. For the acoustic case with constant density, the wave equation can be described by two equations, being the data equation and the object equation. The data equation describes the seismic dataset, in terms of a total field  $p_{\text{tot}}$  at each grid point in the subsurface, the contrast function  $\chi$  and the Green's function  $\mathcal{G}$  in a background medium:

$$p_{\text{data}}(\mathbf{x}_r, \mathbf{x}_s, \omega) = \int \mathcal{G}(\mathbf{x}_r, \mathbf{x}, \omega) p_{\text{tot}}(\mathbf{x}, \mathbf{x}_s, \omega) \chi(\mathbf{x}) d\mathbf{x} \quad (1)$$

eq:calculateData, see include/inversion.h<sup>1</sup>

Reading from right to left, equation 1 can be understood as follows: A source transmits a wavefield that propagates to every point in the subsurface. Note that this wavefield  $p_{\text{tot}}$  is generally quite complex because it takes the interaction of all scatterers in the subsurface already into account. This wavefield is creating secondary sources in all points where the contrast function  $\chi$  is non-zero. The secondary sources transmit energy through a smooth background medium to the receivers, represented by the Green's function  $\mathcal{G}$  in equation 1.

The measured seismic data at every receiver are then a summation of all secondary sources. It should be mentioned that direct waves, including ground roll and surface waves are supposed to be removed from the measured data to obtain  $p_{\text{data}}$ .

We use the 2-D case for the Helmholtz equation. Further, the Green's function is calculated for this equation. The contrast function has to be determined. Further the conjugate scheme is established to determine the error functional.

Measured seismic data always contains some form of noise, so the inversion process is required to be regularised. Therefore, we extend the conjugate gradient scheme so as to include a multiplicative regularisation factor.

#### 3.1.1 The Green's function

A Green's function is the impulse response of an inhomogeneous linear differential equation defined on a domain, with specified initial conditions or boundary conditions. The Green's function of this equation is defined as the solution  $\mathcal{G}(\mathbf{x}, \mathbf{y})$ , of the equation

$$[\nabla^2 \mathcal{G}(\mathbf{x}, \mathbf{y}) + k_0^2 \mathcal{G}(\mathbf{x}, \mathbf{y})] = -\delta(\mathbf{x} - \mathbf{y}).] \quad (2)$$

step:GeneralGreensFunc

The solution to (2) is then given by

$$[u_{\text{ind}}(\mathbf{x}) = \int_{\mathbf{y} \in \mathbb{R}^n} \mathcal{G}(\mathbf{x}, \mathbf{y}) f_{\text{ind}}(\mathbf{y}) d\mathbf{y}.] \quad (3)$$

step:GeneralGreensSol

---

<sup>1</sup> “eq:” denotes equations that are implemented in the code, “step:” denotes equations we present to understand the flow of this manual. Inside the code, the implementation of these equations is marked with such symbols. There, “rel:” means that an equation is related to the “eq”, similar to the use of “step” here.

The interesting cases are the 2D and 3D case. We will focus on the 2D case. The Green's function is then given by

$$[\mathcal{G}(\mathbf{x}, \mathbf{y}) = \frac{i}{4} H_0^{(1)}(k_0 \|\mathbf{x} - \mathbf{y}\|) = -\frac{1}{4} Y_0(k_0 \|\mathbf{x} - \mathbf{y}\|) + \frac{i}{4} J_0(k_0 \|\mathbf{x} - \mathbf{y}\|).] \quad (4)$$

eq:GreensFunc2d, see GreensFunctions.h

In the above equation, the  $H_0$ ,  $J_0$  and  $Y_0$  are defined as the Henkel's functions and can be further researched at : [http://amcm.pcz.pl/get.php?article=2012\\_1/art\\_06.pdf](http://amcm.pcz.pl/get.php?article=2012_1/art_06.pdf)

### 3.1.2 Helmholtz equation

The simplest model we can use is the acoustic Helmholtz equation. We use a scalar pressure field  $u(\mathbf{x})$  and the domain is modeled using  $\chi(\mathbf{x})$ . and is called the contrast and can be directly related to the wave speed at that point in the domain. Mathematically the equation has the form:

$$\nabla^2 u(\mathbf{x}) + k(\mathbf{x})^2 u(\mathbf{x}) = -f_{\text{ext}}(\mathbf{x}). \quad (5)$$

step:generalHelmholtzFunc

The wave number  $k$  is given by  $k = \frac{\omega}{c}$  where  $\omega$  is the angular frequency and  $c$  the acoustic wave velocity in m/s of the true medium. We split the pressure field into  $u(\mathbf{x}) = u_0(\mathbf{x}) + u_{\text{ind}}(\mathbf{x})$ , where  $u_0(\mathbf{x})$  is defined as the field given by the background velocity and external sources,

$$\nabla^2 u_0(\mathbf{x}) + k_0^2 u_0(\mathbf{x}) = -f_{\text{ext}}(\mathbf{x}). \quad (6)$$

step:splitHelmholtzU0

Substituting in (6) results in

$$\nabla^2 u_{\text{ind}}(\mathbf{x}) + k_0^2 u_{\text{ind}}(\mathbf{x}) = -f_{\text{ind}}(\mathbf{x}), \quad (7)$$

step:splitHelmholtzUind

with  $f_{\text{ind}}(\mathbf{x}) = k_0^2 \chi(\mathbf{x}) u(\mathbf{x})$  the induced source term.

### 3.1.3 The Contrast function

The contrast function is defined as:

$$\chi(\mathbf{x}) = 1 - \left( \frac{c_0(\vec{x})}{c(\vec{x})} \right)^2. \quad (8)$$

step:contrastFunc

It depends on the difference between a known background medium  $c_0(\vec{x})$  and the true, but unknown, subsurface model  $c(\vec{x})$ . The total field on the right-hand side in equation 1 is dependent on the contrast, because it contains the interaction between all subsurface scatterers. Then it follows that there is a nonlinear relationship between the subsurface properties and the measured seismic data.

Notice that the contrast is generally unknown, and will be approximated using an iterative scheme. During each step the induced source is considered constant and known so we basically solve the same equation as (3) each step.

## 3.2 Finite Difference Approach

Instead of solving the Helmholtz equation using Green's functions, it is also possible to solve the equation using a finite difference approach. In 2D the domain  $\Omega = [x_{\min}, x_{\max}] \times [z_{\min}, z_{\max}]$  is discretized as follows:

$$G_{h_x, h_z} = \{(x_i, z_j) : x_i = x_{\min} + ih_x, z_j = z_{\min} + jh_z; 1 \leq i, j \leq n\}, \quad (9)$$

where  $h_x$  and  $h_z$  denote the cell size in their respective directions. Note that the positive direction for  $x$  is from left to right ( $x \rightarrow$ ), and for  $z$  downwards ( $z \downarrow$ ). For internal grid points, i.e. points not on the boundary, the discretized Helmholtz equation is given as

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_z^2} + k_{ij}^2 u_{ij} = f_{ij}, \quad (10)$$

where  $u_{i,j} = u(x_i, z_j)$ . Boundary conditions are required to minimize reflections. There are three basic approaches.

1. Add a damping term  $i\omega\alpha u$  to the Helmholtz equation, add an extra layer to the domain and set  $\alpha$  increasing towards the boundary.
2. Use Absorbing Boundary Conditions (ABC), which are boundary conditions that filter out waves.
3. Use Perfectly Matching Layers (PML), which is basically a more sophisticated version of the first option.

The problem with the first option is that  $\alpha$  needs tweaking based on  $\omega$ . ABC can only prevent some reflections since they are bad at preventing reflections from waves traveling almost tangential to the boundary. Therefore, PML seems to be the best option [5]. Both the first order ABC and PML are implemented. Second order ABC are not implemented.<sup>2</sup> The user can choose which approach to use by adjusting input card parameters (see section 3.2.6 for more information). In practice, ABC perform better than PML. Implementing second order ABC may perform even better.

### 3.2.1 First Order ABC

ABC can be used to (partially) prevent reflections. First we assume that the index of refraction  $n(\mathbf{x}) \equiv C_1$  close to the boundary for some  $C_1 \in \mathbb{R}$ . The main idea of ABC is that close to the boundary the solution to the full problem can be approximated by a plane wave that propagates in a direction normal to the boundary [7], i.e.

$$u(\mathbf{x}) \approx Ce^{i\omega C_1 \hat{\mathbf{n}} \cdot \mathbf{x}}, \quad (11)$$

where  $\hat{\mathbf{n}}$  denotes the outward pointing normal of the boundary. Now consider, for example, the upper boundary  $\Gamma_0$ . We want the ABC to block all outgoing waves, so that they cannot cause reflections. Since we assumed that waves close to the boundary can be approximated by a wave traveling in a direction normal to the boundary, we want the waves with direction  $[0, -1]^\top$  to be blocked. Now

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega C_1 u = i\omega C C_1 e^{-i\omega C_1 z} - i\omega C C_1 e^{-i\omega C_1 z} = 0 \quad (12)$$

---

<sup>2</sup>For the theory, see appendix B.

for  $\hat{\mathbf{n}} = [0, -1]^\top$  and for all  $C \in \mathbb{R}^3$ . Hence the ABC can be formulated as

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega n u = 0, \quad (13)$$

The ABC formulated above are first order accurate in the angle at which the wave strikes the boundary [2].

### 3.2.2 Discretization

For points on the boundary not all neighbours exist. Consider, for example, the top left boundary point. There  $u_{i-1,j}$  and  $u_{i,j+1}$  do not exist. From the boundary conditions it follows that

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega n u = -\frac{\partial u}{\partial x} - i\omega n u = 0 \quad (14)$$

$$\leadsto -\frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - i\omega n_{ij} u_{ij} = 0 \quad (15)$$

$$\implies u_{i-1,j} = 2h_x i\omega n_{ij} u_{ij} + u_{i+1,j} \quad (16)$$

where  $\hat{\mathbf{n}} = [-1, 0]^\top$  is the outward pointing normal and

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega n u = -\frac{\partial u}{\partial z} - i\omega n u = 0 \quad (17)$$

$$\leadsto -\frac{u_{i,j+1} - u_{i,j-1}}{2h_z} - i\omega n_{ij} u_{ij} = 0 \quad (18)$$

$$\implies u_{i,j-1} = 2h_z i\omega n_{ij} u_{ij} + u_{i,j+1} \quad (19)$$

where  $\hat{\mathbf{n}} = [0, -1]^\top$  is the outward pointing normal (since the positive  $z$  direction is downwards). Here we used a central difference discretization. By substituting these relations into eq. (10) the final discretization for the boundary point is obtained.

### 3.2.3 Perfectly Matched Layers

The basic idea of Perfectly Matched Layers is that an artificial boundary layer is introduced in which the waves decay exponentially. This is done via a coordinate transformation of the Helmholtz equation such that in the original domain the solutions correspond to the solutions of the original equation and in the newly introduced artificial boundary layer the to exponentially decaying ones [3]. The following coordinate transformation is used for both  $x$  and  $z$ .

$$\frac{\partial}{\partial y} \mapsto \frac{1}{S(y)} \frac{\partial}{\partial y}, \quad (20)$$

where

$$S(y) = 1 + \frac{\sigma(y)}{i\omega n} \quad (21)$$

---

<sup>3</sup>Since, for  $\hat{\mathbf{n}} = [\hat{n}_1, \hat{n}_2]^\top$ , we have

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega n u = \nabla u \cdot \hat{\mathbf{n}} - i\omega n u = i\omega n C (\hat{n}_1^2 e^{i\omega n(\hat{n}_1 x + \hat{n}_2 z)} + \hat{n}_2^2 e^{i\omega n(\hat{n}_1 x + \hat{n}_2 z)}) - i\omega n C e^{i\omega n(\hat{n}_1 x + \hat{n}_2 z)}$$

By choosing  $\sigma_y(y)$  equal to the square of the distance into the PML, the desired properties are obtained. As a consequence of the coordinate transformation we have to solve the adjusted Helmholtz equation

$$\frac{\partial}{\partial x} \frac{S(z)}{S(x)} \frac{\partial}{\partial x} u + \frac{\partial}{\partial z} \frac{S(x)}{S(z)} \frac{\partial}{\partial z} u + n^2 \omega^2 S(x) S(z) u = f(x, z), \quad (22)$$

where we assume  $u \equiv 0$  (Dirichlet) on the outer boundary. In the original domain the equation above is simply reduced to the original Helmholtz discretization.

### 3.2.4 Discretization

Within the original domain, the discretization in eq. (10) can be used. In order to obtain the discretization of eq. (22) within the PML, we first have to expand it.

$$\frac{\partial}{\partial x} \frac{S(z)}{S(x)} \frac{\partial}{\partial x} u + \frac{\partial}{\partial z} \frac{S(x)}{S(z)} \frac{\partial}{\partial z} u + n^2 \omega^2 S(x) S(z) u \quad (23)$$

$$= \frac{S(z)}{S(x)} \frac{\partial^2 u}{\partial x^2} - \frac{S(z)}{S^2(x)} S'(x) \frac{\partial u}{\partial x} + \frac{S(x)}{S(z)} \frac{\partial^2 u}{\partial z^2} - \frac{S(x)}{S^2(z)} S'(z) \frac{\partial u}{\partial z} + n^2 \omega^2 S(x) S(z) u \quad (24)$$

By using central difference discretizations for the first and second derivatives of  $u$  we obtain the following discretization.

$$\frac{S(z_j)}{S(x_i)} \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_x^2} - \frac{S(z_j)}{S^2(x_i)} S'(x_i) \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} \quad (25)$$

$$+ \frac{S(x_i)}{S(z_j)} \frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h_z^2} - \frac{S(x_i)}{S^2(z_j)} S'(z_j) \frac{u_{i,j+1} - u_{i,j-1}}{2h_z} \quad (26)$$

$$+ n_{ij}^2 \omega^2 S(x_i) S(z_j) u_{ij} = f_{ij} \quad (27)$$

### 3.2.5 Source Implementation

The easiest method to implement the source is by just using a point source, that is, a source located at one precise grid point. In order to achieve this the source coordinates are rounded to the nearest grid point. For a point source, the integral over the cell size should equal 1, hence the point source value equals  $1/h_x h_z$ . This is necessary to calculate the pressure field using the finite difference model. The problem with this source implementation is that errors are introduced due to rounding the source positions to the nearest grid point. Especially when combining different forward and inversion models, one would want to avoid this.

For this reason another source implementation is added which should more accurately reflect the actual position of the source. This implementation is described in [6]. Basically, the source function used is given by

$$\frac{\sin(\pi d(x))}{\pi d(x)} \frac{\sin(\pi d(z))}{\pi d(z)} \quad (28)$$

where  $d(y)$  denotes the distance in grid points from the source position. This source function is then multiplied by a so-called window function which basically determines over how many grid points the source is spread out. The final implementation then becomes

$$\frac{I_0(\beta\sqrt{1-(d(x)/r)^2})}{I_0(\beta)} \frac{\sin(\pi d(x))}{\pi d(x)} \frac{I_0(\beta\sqrt{1-(d(z)/r)^2})}{I_0(\beta)} \frac{\sin(\pi d(z))}{\pi d(z)}, \quad (29)$$

where  $I_0(y)$  denotes the Bessel function of the first kind. The parameters  $\beta$  and  $r$  denote the shape and half-width, respectively. The half-width gives the range in grid points from the actual source position in which the source term will be non-zero. The parameters  $r = 4$  and  $\beta = 6.31$  are a good choice under most circumstances [6].

For both source implementations the original domain is enlarged so that it includes all the sources. Note that for  $r > 0$  additional grid points are required to ensure the whole source is included in the domain.

### 3.2.6 Input Card Parameters

Finite difference forward model has its own input card containing four parameters. First the PMLWidthFactor, which determines the width of the PML by multiplying this factor in the  $x$  and  $z$  direction by the wavelength of the background subsurface. Setting this parameter equal to 0.0 in both directions entails that ABC are used instead of PML, which is the default setting because in practice ABC performs better than PML.

The SourceParameter determines the half-width  $r$  (which is a natural number) and shape  $\beta$  for the sine source implementation as given in the section above and [6]. By default the parameters are set to  $r = 4$  and  $\beta = 6.31$  for the reason given in previous section. When setting  $r = 0$ , the point source implementation will be used.

### 3.2.7 Python Scripts

Python scripts were written for prototyping purposes. The `Helmholtz.py` script is a script in which one can compare the exact solution to the (first and second order) ABC and PML on a homogeneous background. The `HelmholtzPMLWholeGrid.py` contains a PML implementation that also enlarges the grid to include sources outside the grid. Note that both scripts use coordinate systems different from the C++ implementation. These scripts can be extended to test out, for example, a method for extracting  $p_{\text{data}}$  (see next section).

### 3.2.8 Notes and Future Work

Some notes:

- For the best accuracy, one would need at least 5 and ideally around 10 grid points per wavelength. For this reason, higher frequencies tend to lead to worse results (given a fixed grid size).

There is still work that can be done to improve or expand the finite difference forward model.

1. Second order ABC can be implemented, which are expected to improve the results (see Python script and appendix B).
2. Currently the domain is always enlarged so it includes all the source. In a finite difference context this is necessary for computing the pressure field. Perhaps it is possible to find a method to project the effect of the source on the boundary of the subsurface domain, and use that as a boundary condition to solve the pressure field.

3. Extraction of  $p_{\text{data}}$  can be implemented by implementing the receivers via the method given in [6]. This would also require the receivers to always be included in the domain, just as the sources are now always included.
4. Currently, methods which are required by the Conjugate Gradient inversion model are implemented in the forward model. If an inversion model which does not directly require these methods is implemented, the forward model can be simplified even further (although it then would no longer work with the CG inversion model). After all, the forward model is only there to provide  $p_{\text{tot}}$  and  $p_{\text{data}}$ .
5. A different method to solve the system of linear equations can be implemented decrease the execution time. Currently, the *LU*-decomposition is used to solve the system. Faster methods could be explored. One possibility would be to use the same *LU*-decomposition, but permute the matrix before factorization to reduce its bandwidth.

### 3.3 Solver and Noise

#### 3.3.1 The Conjugate Gradient (CG) Scheme

In the inversion scheme, we try to minimise the difference between the measured data  $p_{\text{data}}$  and the modelled data that is obtained using the currently best estimate of the contrast function and the fixed total field.

The residual between the measured and modelled data is obtained as:

$$r(\mathbf{x}_r, \mathbf{x}_s, \omega) = p_{\text{data}}(\mathbf{x}_r, \mathbf{x}_s, \omega) - [\mathcal{K}_\chi](\mathbf{x}_r, \mathbf{x}_s, \omega), \quad (30)$$

residualMeasureModel

with

$$[\mathcal{K}_\chi](\mathbf{x}_r, \mathbf{x}_s, \omega) = \int \mathcal{G}(\mathbf{x}_r, \mathbf{x}, \omega) p_{\text{tot}}(\mathbf{x}, \mathbf{x}_s, \omega) \chi(\mathbf{x}) d\mathbf{x} \quad (31)$$

modelledDataGreensConv

a linear operator in  $\chi$ , and where we adopt the specific notation by Haffinger (Haffinger, Ph.D. thesis 2013, TU Delft) for consistency. The error functional is equal to

$$F(\chi) = \eta \int |r(\mathbf{x}_r, \mathbf{x}_s, \omega)|^2 d\mathbf{x}_s d\mathbf{x}_r d\omega, \quad (32)$$

errorFunc

with  $\eta$

$$\eta^{-1} = \int |p_{\text{data}}(\mathbf{x}_r, \mathbf{x}_s, \omega)|^2 d\mathbf{x}_s d\mathbf{x}_r d\omega, \quad (33)$$

errorFuncSubEtaInv

so that for  $\chi = 0$  we have  $F = 1$ . Notice the implicit dependency on  $\chi$ .

We want to find a sequence of contrast functions,  $\chi^{(n)}(\vec{x})$ ,  $n = 1, 2, \dots$ , in which error functional decreases with increasing iterations. Therefore, after each iteration the contrast function is updated as:

$$\chi^{(n)}(\vec{x}) = \chi^{(n-1)}(\vec{x}) + \alpha_n \zeta_n(\vec{x}) \quad (34)$$

`contrastUpdate`

Here, the step size of the update is determined by the parameter  $\alpha_n$  while the update directions are conjugate gradient directions given by

$$\zeta_1(\vec{x}) = g_1(\vec{x}), \zeta_n(\vec{x}) = g_n(\vec{x}) + \gamma_n \zeta_{n-1}(\vec{x}) \quad (35)$$

`updateDirectionsCG`

The functional derivative w.r.t.  $\chi$  in direction  $\mathbf{d}$  of the error functional is equal to

$$\begin{aligned} \frac{\partial F(\chi, \mathbf{d})}{\partial \chi} &= \lim_{\epsilon \rightarrow 0} \frac{F(\chi + \epsilon \mathbf{d}) - F(\chi)}{\epsilon} \\ &= -2\eta \int \operatorname{Re} \{ [\mathcal{K}_\mathbf{d}] (\mathbf{x}_r, \mathbf{x}_s, \omega)^\dagger r(\mathbf{x}_r, \mathbf{x}_s, \omega) \} d\mathbf{x}_s d\mathbf{x}_r d\omega \end{aligned} \quad (36)$$

`functionalDerWRTD`

The derivation of this equation is provided in section A.1.

For the discrete  $\mathcal{K}$  operator the integrand will have the form

$$g_n(\vec{x}) = \eta \operatorname{Re} \{ [\mathcal{K}^* \mathbf{r}_{n-1}] (\vec{x}) \} \quad (37)$$

`integrandForDiscreteK`

where the  $\star$  denote element wise multiplication per row and  $\operatorname{Re}$  denotes that only the real part will be used. The adjoint operator  $[\mathcal{K}^* \mathbf{r}_{n-1}]$  can be seen as a backprojection operator that maps the residual between measured data and modelled data from the surface domain to its associated location in the scattering domain.

In our conjugate gradient scheme we make use of the Polak-Ribière direction,

$$\gamma_n = \frac{\int g_n(\vec{x}) [g_n(\vec{x}) - g_{n-1}(\vec{x})] d(\vec{x})}{\int g_{n-1}(\vec{x}) g_{n-1}(\vec{x}) d(\vec{x})} \quad (38)$$

`PolakRibiereDirection`

The optimal step size is found from the minimisation of cost functional equation, by setting the derivative equal to zero. Consequently the optimal step size becomes:

$$\alpha_n = \frac{\operatorname{Re} \left\{ \int \int \int r_{n-1}^*(\mathbf{x}_r, \mathbf{x}_s, \omega) [\mathcal{K} \zeta_n](\mathbf{x}_r, \mathbf{x}_s, \omega) d\vec{x}_s d\vec{x}_r d\omega \right\}}{\int \int \int |[\mathcal{K} \zeta_n](\mathbf{x}_r, \mathbf{x}_s, \omega)|^2 d\vec{x}_s d\vec{x}_r d\omega} \quad (39)$$

`optimalStepSizeCG`

To initialise the conjugate gradient scheme, we assume,  $\chi^0(\vec{x}) = 0$ , leading to  $r_0(\mathbf{x}_r, \mathbf{x}_s, \omega) = p_{data}(\mathbf{x}_r, \mathbf{x}_s, \omega)$ . Therefore, we find the gradient as :

$$g_1(\vec{x}) = \eta \operatorname{Re} \{ [\mathcal{K}^* p_{\text{data}}](\vec{x}) \} \quad (40)$$

gradientRunc

and we get the first update parameter as :

$$\alpha_1 = \frac{\operatorname{Re} \{ \int \int \int p_{\text{data}}^*(\mathbf{x}_r, \mathbf{x}_s, \omega) [\mathcal{K} g_1](\mathbf{x}_r, \mathbf{x}_s, \omega) d\vec{x}_s d\vec{x}_r d\omega \}}{\int \int \int | [\mathcal{K} g_1](\mathbf{x}_r, \mathbf{x}_s, \omega) |^2 d\vec{x}_s d\vec{x}_r d\omega} \quad (41)$$

firstStepSize

### 3.3.2 Multiplicative Regularisation

Since all seismic data contains some form of noise, the inversion process needs to be stabilised. Therefore, the CG scheme is extended in a way that it contains a multiplicative regularisation factor.

The error functional  $\mathcal{F}^{tot}$  becomes a product of the original error functional and a newly introduced regularisation factor  $\mathcal{F}^{reg}$ :

$$\mathcal{F}_n^{tot} = \mathcal{F}_n^{data} \mathcal{F}_n^{reg} \quad (42)$$

errorFuncRegul

The following weighting function is used:

$$b_n^2 = (\int_{\vec{x}} d\vec{x})^{-1} (| \nabla \chi^{n-1} |^2 + \delta_{n-1}^2)^{-1} \quad (43)$$

errorFuncRegulWeighting

In the above equation, the  $\delta_n^2$  is the steering factor and can be defined as:

$$\delta_n^2 = (\int_{\vec{x}} d\vec{x})^{-1} \int_{\vec{x}} | \nabla \chi^{n-1} |^2 d\vec{x} \quad (44)$$

errorFuncRegulSteering

The new total cost functional then becomes a product of two second order polynomials:

$$\mathcal{F}_n^{tot} = (\mathcal{A}_2 \alpha_n^2 + \mathcal{A}_1 \alpha_n + \mathcal{A}_0)(\mathcal{B}_2 \alpha_n^2 + \mathcal{B}_1 \alpha_n + \mathcal{B}_0) \quad (45)$$

errorFuncFourthOrder

with the constants  $A_0, \dots, A_2, B_2$  in appendix A.2.

### 3.3.3 Nonlinear field update based on the domain equation

Initially, the subsurface properties are unknown and the very first inversion is based on the assumption that wavefield propagation occurs in smooth non-scattering background medium only. Using an approximate smooth property model of the subsurface immediately tells us that

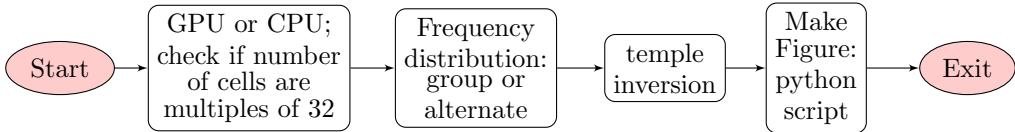


Figure 3: High Level activity diagram for main.cpp

the wavefield propagation in such an approximate medium cannot be accurate and that for accurate results we should make the wavefields consistent with the currently best estimate of the true medium. For this reason we assume  $p_{tot} \approx p_0$ .

Since the output of linear full waveform inversion is a property model, we do not only obtain structural information but also a “first order” approximation of the subsurface properties at every grid point in the inversion domain. We can now use this property model to update the total field by solving the domain equation with in principle any suitable numerical method. Therefore, we iteratively build up the total fields as a sum of the background field and a number of basis functions:

$$\phi_n(\mathbf{x}_r, \mathbf{x}_s, \omega) = \int_{\vec{x} \in D} \mathcal{G}(\mathbf{x}_r, \mathbf{x}_s, \omega) \partial \mathcal{W}_n d\vec{x}' \quad (46)$$

eq:buildField, see calcField.h and rel green\_rect\_2D\_cpu.h

And the incremental contrast sources  $\partial \mathcal{W}_n$  are given by:

$$\partial \mathcal{W}_1 = \chi^{(1)} p_{tot}^0, \mathcal{W}_n = \chi^{(n)} p_{tot}^{n-1} - \chi^{(n-1)} p_{tot}^{n-2}, n > 1. \quad (47)$$

eq:incrementalContrastSrcs, see calcField.h

The weighting factors are then determined and  $p_{tot}$  is updated according to the equation:

$$p_{tot}^{(N)}(\mathbf{x}_r, \mathbf{x}_s, \omega) = p_0(\mathbf{x}_r, \mathbf{x}_s, \omega) + \sum_{N=1}^N \alpha_n^{(N)}(\mathbf{x}_s, \omega) \phi_n(\mathbf{x}_r, \mathbf{x}_s, \omega) \quad (48)$$

weightingFactorsField, see calcField.h

## 4 Implementation

The current code implementation for ALTEN can be found on [http://amcm.pcz.pl/get.php?article=2012\\_1/art\\_06.pdf](http://amcm.pcz.pl/get.php?article=2012_1/art_06.pdf), with the host at <https://git.alten.nl/parallelized-fwi.git>. The Code Algorithm is complicated and can be divided into the mathematical functions and the computational functions.

For the mathematical functions, please refer to the FWI.vpp. It explains the `main.cpp` and the classes defined for this project. The main function basically creates the objects required for the inversion and then called the inversion `cpu.h` function which performs the main tasks. A high level representation of the `main.cpp` can be seen in Figure 3.

### 4.1 Input parameters

The inputs for the program are as follows.

## 4.2 Subroutines

### 4.2.1 main.cpp

A high level representation of the `main.cpp` can be viewed in the `FWI.vpp` as a UML diagram.

The `main.cpp` initialises the MPI functions responsible for parallel computations. Frequency distribution group and alternate are determined here and depending on the user input, the objects are created. Further, the main function creates the objects for the temple inversion and calls the appropriate functions as seen in `FWI.temple`.

### 4.2.2 temple\_inversion

The main part of the code can be seen in the `temple_inversion`. Over here, the Green's functions are created and the mathematical model detailed in section 2 is executed. A brief explanation of the functions can be seen as follows:

#### 4.2.2.1 createGreens()

This function creates an array of Green's functions of the Helmholtz equations. It is called from the Inversion class and refers to the equation 4.

#### 4.2.2.2 CreateP0()

$p_0$  is the field calculated with the known contrast and follows the explanation given in section 3.3.3.

#### 4.2.2.3 createTotalField()

After the initial approximation, first iterative  $p_{tot}$  is calculated here. This is done by calling the function `calcfield()`. In this function, the equations defined in section 3.3.3 are calculated. At first the incremental contrast sources are defined according to equation 47 and then the  $\phi_n$  is constructed. Based on this 48 is constructed. This step concludes the forward modelling step.

Based on the input parameters,  $\alpha$  is used if  $\alpha$  is 1 and is not used if it is set to 0.

#### 4.2.2.4 eq:calculateData()

Once the total field  $p_{tot}$  is obtained, the  $p_{data}$  is calculated according to 1.

#### 4.2.2.5 Reconstruct()

The reconstruct function might be the most important function of the code. The code runs in an iterative loop to minimise the residuals. According to tolerance values determined by the user and the max number of iterations, the residuals are minimised in order to obtain  $p_{data}$ . Once the residuals are minimised below the tolerance level, the final model is obtained.

The Conjugate Gradient scheme detailed in section 3.3.1 is coded here.

#### 4.2.2.6 MakeFigure()

Finally, the `MakeFigure()` function is called which calls a python script to generate the figures.

### 4.3 Parallelization: from a single CPU to Multiple and GPU

The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.

In parallel computing, multiple computers - or even multiple processor cores within the same computer - are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks.

For the FWI, the user sets in the input parameters, if GPU or CPU should be used. This is already checked in the beginning of the code and the appropriate functions corresponding to each are used. Further, the user also sets whether the frequency distribution should be group or alternate. This means that instead of splitting calculations over frequency chunks, split them alternatively over frequency for a better load balancing.

## 5 Downloading, Installing and Running the Code

### 5.1 Pre-requisites

The prerequisite development tools needed can be installed using the following commands.

1. `sudo apt-get install git`
2. `sudo apt-get install qt5-default`
3. `sudo apt-get install libeigen3-dev`
4. `sudo apt-get install python2.7-dev`
5. `sudo apt-get install python2.7`
6. `sudo apt-get install python-tk`
7. `sudo apt-get install python-numpy`
8. `sudo apt-get install python-matplotlib`

### 5.2 Cloning the Repository

To clone the FWI repository using git,

```
git clone -o redmine https://git.alten.nl/parallelized-fwi.git
```

This will create a copy of the repository, in a folder named **parallelized-fwi**

Any branch as needed can then be checked out from inside the **parallelized-fwi** folder, e.g.  
the develop branch

```
git checkout develop
```

### 5.3 Build/Run

To build the project, first create a folder titled **build** outside the **parallelized-fwi** folder.  
NOTE: This folder should be exactly 1 level outside the **parallelized-fwi** folder.

```
mkdir Build
cd Build
cmake -DCMAKE_BUILD_TYPE=Release ../parallelized-fwi/
make -j4 (the flag -j is used to build in parallel)
```

Now, the individual scripts for the preProcessing and the processing part can be run as shown below:

```
cd applications
cd preProcessing
./FWIPreProcess
cd ../processing
./FWIProcess
```

The input parameters for the code are provided in the input card i.e. default.in. User can create his/her own input card with a new name e.g. newCard.in. To use this input card use the card name as an argument when running the executables, ./FWIPreProcess newCard and ./FWIProcess newCard.

For post-processing (i.e. generation of image using the estimated chi values), the python script `imageCreator_CMake.py` can be used. This script is located inside the **parallelized-fwi** folder and can be used as,

```
python imageCreator_CMake.py
```

The pre-processing, processing and the image creation step can all be grouped together using the python wrapper `wrap_FWI_CMake.py` located inside the **parallelized-fwi** folder.

```
python wrap_FWI_CMake.py
```

## 6 Results and Analysis - example: Temple Reservoir

The example chosen for the FWI code is the logo of the Delphi research consortium. The logo includes a Greek temple consisting of four major pillars under a triangularly shaped roof, as shown in Figure 4. Additionally we added a layer below the temple to represent the basement. The whole structure is embedded in a homogeneous background which carries waves at a uniform acoustic velocity of  $c_0 = 2000m/s$ . The temple material's carriage speed is  $c = 2218m/s$ . In the original data in `temple.txt`, you can see a list of zeros and `1.869...e-01`. The zeros are the background's contrast "with itself" and the `1.869...e-01` the result of  $1 - \left(\frac{2000}{2218}\right)^2$ . The results obtained from running the FWI code can be seen in the following figures:

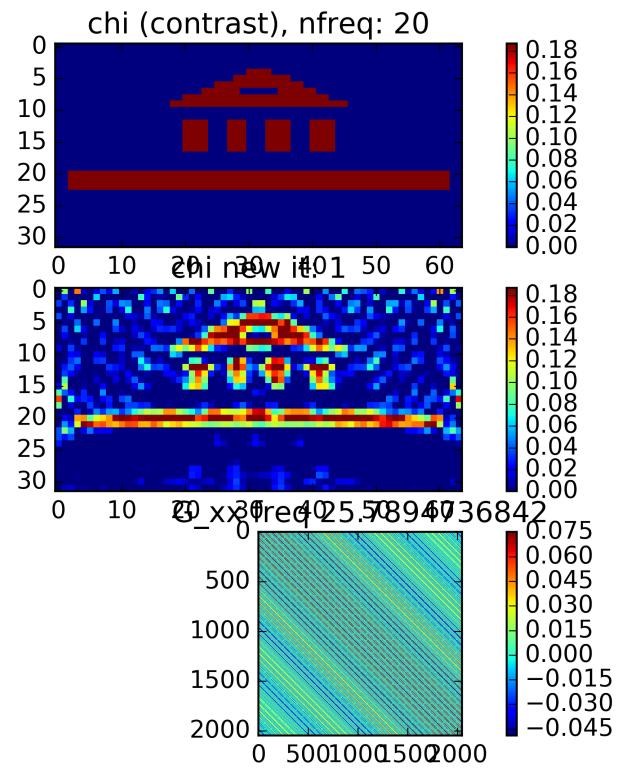


Figure 4: Chi estimation Result 1

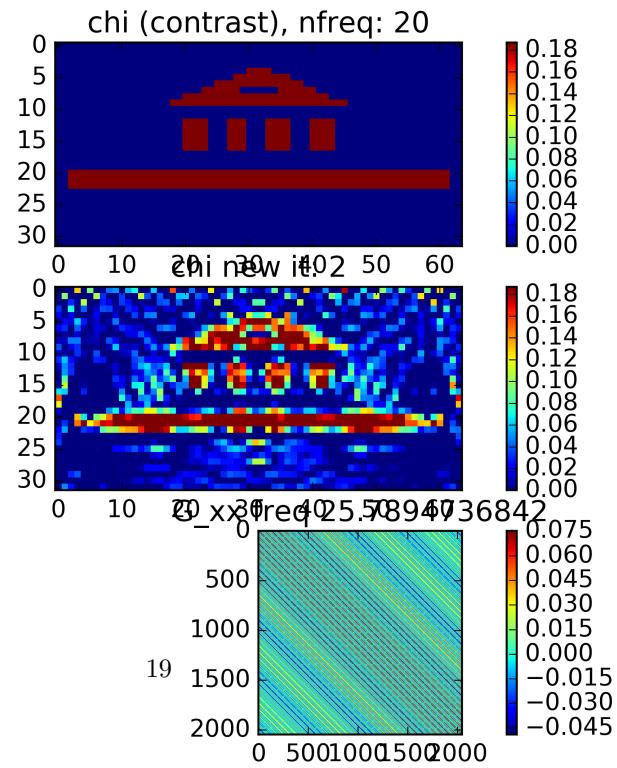


Figure 5: Chi estimation Result 2

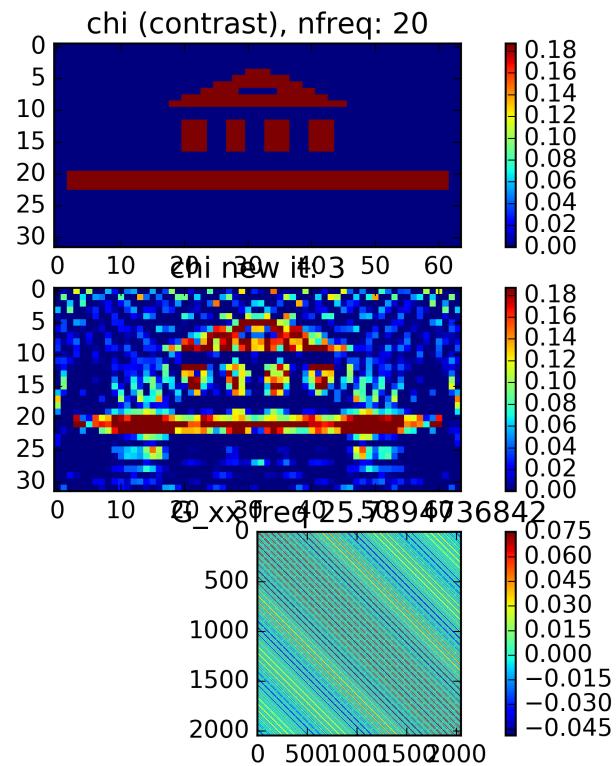


Figure 6: Chi estimation Result 3

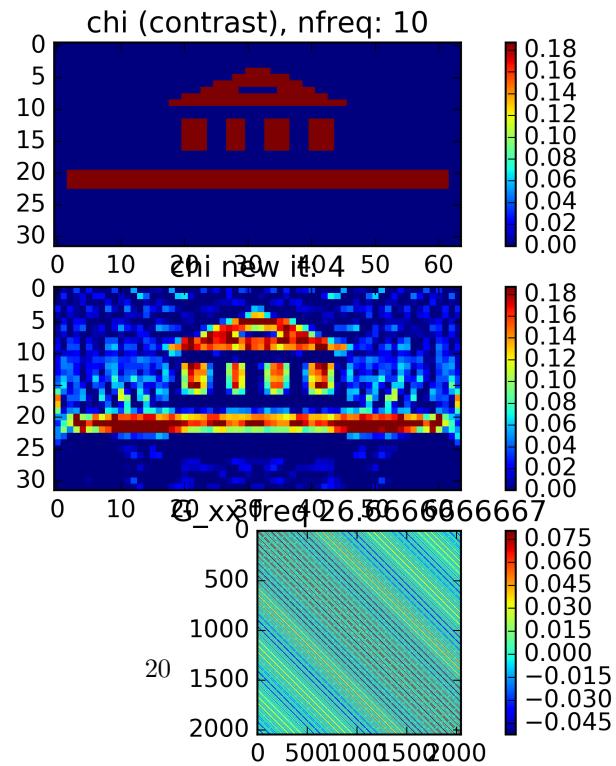


Figure 7: Chi estimation Result 4

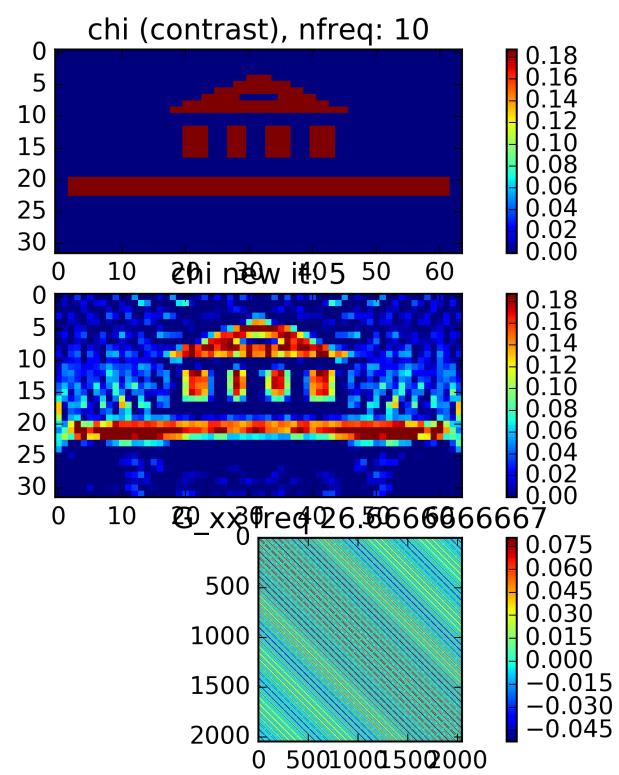


Figure 8: Chi estimation Result 5

## References

- [1] A. Bamberger, P. Joly, and J. E. Roberts. Second-order absorbing boundary conditions for the wave equation: A solution for the corner problem. *SIAM Journal on Numerical Analysis*, 27:323–352, 04 1990.
- [2] B. Engquist and A. Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computation*, 31(139):629–651, 1977.
- [3] Y. A. Erlangga. Advances in iterative methods and preconditioners for the helmholtz equation. *Archives of Computational Methods in Engineering*, 15(1):37–66, Mar 2008.
- [4] A. Fichtner. *Full Seismic Waveform Modelling and Inversion*. 01 2011.
- [5] Y. Gao, H. Song, J. Zhang, and Z. Yao. Comparison of artificial absorbing boundaries for acoustic wave equation modelling. *Exploration Geophysics*, 48(1):76–93, 2017.
- [6] G. J. Hicks. Arbitrary source and receiver positioning in finite-difference schemes using kaiser windowed sinc function. *Geophysics*, 67, 01 2002.
- [7] O. Runborg. Numerical solutions of differential equation – lecture notes. 2012.

## A Appendices

### A.1 Derivation of the functional derivative of the error functional

$$\begin{aligned}
\frac{\partial F(\chi, \mathbf{d})}{\partial \chi} &= \lim_{\epsilon \rightarrow 0} \frac{F(\chi + \epsilon \mathbf{d}) - F(\chi)}{\epsilon} \\
&= \lim_{\epsilon \rightarrow 0} \frac{\eta}{\epsilon} \int (p_{\text{data}} - [\mathcal{K}_\chi] - \epsilon [\mathcal{K}_\mathbf{d}]) (p_{\text{data}} - [\mathcal{K}_\chi] - \epsilon [\mathcal{K}_\mathbf{d}])^\dagger \\
&\quad - (p_{\text{data}} - [\mathcal{K}_\chi]) (p_{\text{data}} - [\mathcal{K}_\chi])^\dagger d\mathbf{x}_s d\mathbf{x}_r d\omega \\
&= -\lim_{\epsilon \rightarrow 0} \frac{\eta}{\epsilon} \int \epsilon \left[ [\mathcal{K}_\mathbf{d}] (p_{\text{data}} - [\mathcal{K}_\chi])^\dagger + [\mathcal{K}_\mathbf{d}]^\dagger (p_{\text{data}} - [\mathcal{K}_\chi]) \right] \\
&\quad - \epsilon^2 [\mathcal{K}_\mathbf{d}] [\mathcal{K}_\mathbf{d}]^\dagger d\mathbf{x}_s d\mathbf{x}_r d\omega \\
&= -2\eta \int \text{Re} \left\{ [\mathcal{K}_\mathbf{d}] (p_{\text{data}} - [\mathcal{K}_\chi])^\dagger \right\} d\mathbf{x}_s d\mathbf{x}_r d\omega \\
&= -2\eta \int \text{Re} \left\{ [\mathcal{K}_\mathbf{d}] (\mathbf{x}_r, \mathbf{x}_s, \omega)^\dagger r(\mathbf{x}_r, \mathbf{x}_s, \omega) \right\} d\mathbf{x}_s d\mathbf{x}_r d\omega
\end{aligned}$$

### A.2 Prefactors of the Total Cost Equation

$$\mathcal{A}_2 = \eta \int \int \int |\mathcal{K}\zeta_n|^2 d\vec{x}_s d\vec{x}_r d\omega \tag{49}$$

totalCostA2

$$\mathcal{A}_1 = -2\eta \text{Re} \left\{ \int \int \int r_{n-1}^* |\mathcal{K}\zeta_n| d\vec{x}_s d\vec{x}_r d\omega \right\}, \hat{E} \tag{50}$$

totalCostA1

$$\mathcal{A}_0 = \eta \int \int \int |r_{n-1}|^2 d\vec{x}_s d\vec{x}_r d\omega = \mathcal{F}_{n-1}^{\text{data}}, \hat{E} \tag{51}$$

totalCostA0

$$\mathcal{B}_2 = \| b_n \nabla \zeta_n \|_D^2, \hat{E} \tag{52}$$

totalCostB2

$$\mathcal{B}_1 = 2 \langle b_n \nabla \chi^{n-1}, b_n \nabla \zeta_n \rangle_D, \hat{E} \tag{53}$$

totalCostB1

$$\mathcal{B}_0 = \| b_n \nabla \chi^{n-1} \|_D^2 + \delta_{n-1}^2 \| b_n \|_D^2 \tag{54}$$

totalCostB0

## B Absorbing Boundary Conditions

### B.1 Second Order ABC

To improve the absorption rate of the ABC, we will consider ABC which are second order accurate in the angle at which the wave strikes the boundary [2]. They can be formulated as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial \mathbf{s}^2} = 0, \quad (55)$$

where  $\mathbf{s}$  denotes the vector tangential to the boundary (positive or negative direction does not matter). In the corner specials ABC apply [1]

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = 0, \quad (56)$$

where  $\hat{\mathbf{s}}$  denotes the vector which is the sum of the tangential outward pointing vectors in the  $x$  and  $z$  directions.

#### B.1.1 Discretization

Similar to the first order discretization, at the boundary we need to find an expression for the ghost points. Consider, for example, the top boundary ( $z = 0$ ). Here  $\hat{\mathbf{n}} = [0, -1]^\top$  and  $\mathbf{s} = [\pm 1, 0]$ . Now eq. (55) can be discretized as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - i\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial \mathbf{s}^2} = -\frac{\partial u}{\partial z} - i\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial x^2} = 0 \quad (57)$$

$$\rightsquigarrow -\frac{u_{i,j+1} - u_{i,j-1}}{2h_z} - i\omega n_{ij} u_{ij} - \frac{i}{2\omega n_{ij}} \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_x^2} = 0 \quad (58)$$

$$\implies u_{i,j-1} = 2h_z i\omega n_{ij} u_{ij} + u_{i,j+1} + \frac{h_z i}{\omega n_{ij} h_x^2} (u_{i+1,j} - 2u_{ij} + u_{i-1,j}) = 0 \quad (59)$$

This can be implemented in a straightforward manner by substituting the relation above into eq. (10).

For corner points, e.g.,  $(x, z) = (0, 0)$  with  $\hat{\mathbf{s}} = [-1, -1]^\top$  it is not possible to fully discretize the  $\partial u / \partial \hat{\mathbf{s}}$  using the central difference formula. Then there would be two unknowns, in this case  $u_{i-1,j}$  and  $u_{i,j-1}$ , and hence no straightforward expression for either of these unknowns to be substituted in eq. (10). For this reason, a central difference scheme is only used in the  $x$  direction and a backward/forward scheme is used in the  $z$  direction. This leads to the following discretization at  $(0, 0)$ .

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = -\frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2} i\omega n u = 0 \quad (60)$$

$$\rightsquigarrow -\frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j-1}}{2h_z} - \frac{3}{2} i\omega n_{ij} u_{ij} = 0 \quad (61)$$

$$\implies u_{i-1,j} = (3h_x i\omega n_{ij} - 2\frac{h_x}{h_z})u_{i,j} + u_{i+1,j} + 2\frac{h_x}{h_z}u_{i,j+1} = 0 \quad (62)$$

At the top right corner, with  $\hat{\mathbf{s}} = [1, -1]^\top$ , the following discretization is used.

$$\frac{\partial u}{\partial \hat{s}} - \frac{3}{2}i\omega n u = \frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2}i\omega n u = 0 \quad (63)$$

$$\rightsquigarrow \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j}}{h_z} - \frac{3}{2}i\omega n_{ij}u_{ij} = 0 \quad (64)$$

$$\implies u_{i,j+1} = (3h_x i\omega n_{i,j} - 2\frac{h_x}{h_z})u_{i,j} + u_{i-1,j} + 2\frac{h_x}{h_z}u_{i,j+1} = 0 \quad (65)$$