Full Waveform Inversion

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

November 30, 2018

Contents

1 Preface

This document describes a method to predict the size and shape of unexplored oil reserervoirs 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).

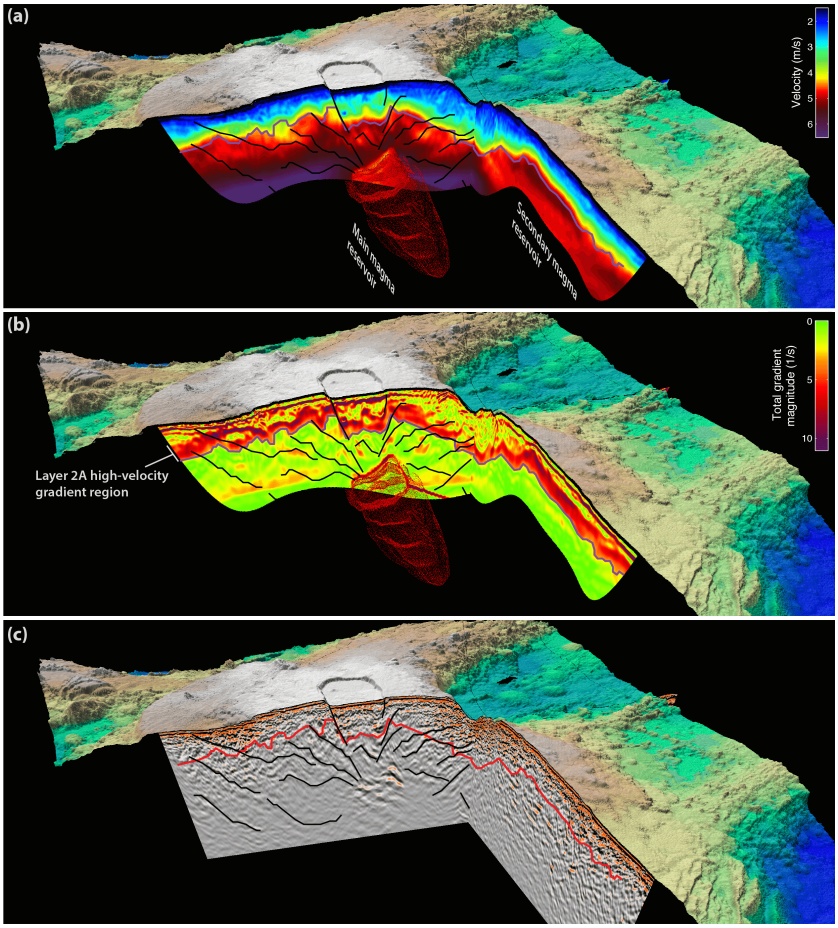

Figure 1: Seismic Full Waveform Inversion

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.

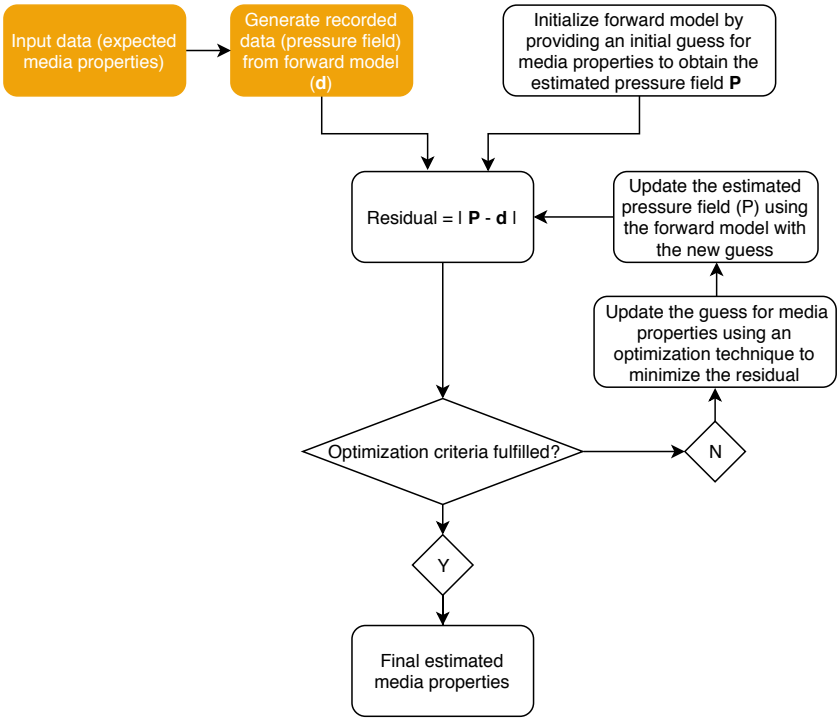

Figure 2: Full Waveform Inversion Methodology

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_{tot} at each grid point in the subsurface, the contrast function χ and the Green's function $\mathcal{G}$ in a background medium:

$$p_{\text{data}}(\mathbf{x}_{\text{r}}, \mathbf{x}_{\text{s}}, \omega) = \int \mathcal{G}(\mathbf{x}_{\text{r}}, \mathbf{x}, \omega) p_{\text{tot}}(\mathbf{x}, \mathbf{x}_{\text{s}}, \omega) \chi(\mathbf{x}) \, \mathrm{d}\mathbf{x} \tag{1}$$

`eq:calculateData, see include/inversion.h`[1]

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_{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 χ 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_{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

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

`step:GeneralGreensFunc`

The solution to (2) is then given by

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

`step:GeneralGreensSol`

[1] "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

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

`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`

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:

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

`step:generalHelmholtzFunc`

The wave number k is given by $k = \frac{\omega}{c}$ where ω 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,

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

`step:splitHelmholtzU0`

Substituting in (6) results in

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

`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:

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

`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 (**??**) each step.

3.2 Solver and Noise

3.2.1 The Conjugate Gradient (CG) Scheme

In the inversion scheme, we try to minimise the difference between the measured data p_{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:

$$r(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega) = p_{\text{data}}(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega) - [\mathcal{K}_\chi]\,(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega), \tag{9}$$

`residualMeasureModel`

with

$$[\mathcal{K}_\chi]\,(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega) = \int \mathcal{G}(\mathbf{x}_\text{r}, \mathbf{x}, \omega) p_{\text{tot}}(\mathbf{x}, \mathbf{x}_\text{s}, \omega) \chi(\mathbf{x})\, \mathrm{d}\mathbf{x} \tag{10}$$

`modelledDataGreensConv`

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

$$F(\chi) = \eta \int |r(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega)|^2 \, \mathrm{d}\mathbf{x}_\text{s}\, \mathrm{d}\mathbf{x}_\text{r}\, \mathrm{d}\omega, \tag{11}$$

`errorFunc`

with

$$\eta^{-1} = \int |p_{\text{data}}(\mathbf{x}_\text{r}, \mathbf{x}_\text{s}, \omega)|^2 \, \mathrm{d}\mathbf{x}_\text{s}\, \mathrm{d}\mathbf{x}_\text{r}\, \mathrm{d}\omega, \tag{12}$$

`errorFuncSubEtaInv`

so that for $\chi = 0$ we have $F = 1$. Notice the implicit dependency on χ.
We want to find a sequence of contrast functions, $\chi^{(n)}(\vec{x}), n = 1, 2, ...$, in which error functional decreases with increasing iterations. Therefore, after each iteration the contrast function is updated as:

$$\chi^{(n)}(\vec{x}) = \chi^{(n-1)}(\vec{x}) + \alpha_n \zeta_n(\vec{x}) \tag{13}$$

`contrastUpdate`

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

$$\zeta_1(\vec{x}) = g_1(\vec{x}), \zeta_n(\vec{x}) = g_n(\vec{x}) + \gamma_n \zeta_{n-1}(\vec{x}) \tag{14}$$

`updateDirectionsCG`

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

$$\begin{aligned}\frac{\partial F(\chi,\mathbf{d})}{\partial \chi} &= \lim_{\epsilon\to 0}\frac{F(\chi+\epsilon\mathbf{d})-F(\chi)}{\epsilon} \\ &= -2\eta\int \mathrm{Re}\left\{[\mathcal{K}_{\mathbf{d}}]\left(\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega\right)^{\dagger} r(\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)\right\}\,\mathrm{d}\mathbf{x}_{\mathrm{s}}\,\mathrm{d}\mathbf{x}_{\mathrm{r}}\,\mathrm{d}\omega \qquad \texttt{functionalDerWRTD}\end{aligned} \tag{15}$$

The derivation of this equation is provided in section 7.1.
For the discrete $\mathcal{K}$ operator the integrand will have the form

$$g_n(\vec{x}) = \eta\mathrm{Re}\left\{[\mathcal{K}^{\star}\mathbf{r}_{n-1}](\vec{x})\right\} \tag{16}$$

`integrandForDiscreteK`

where the $\star$ denote element wise multiplication per row and Re denotes that only the real part will be used. The adjoint operator $[\mathcal{K}^{\star}\mathbf{r}_{n-1}]$ 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,

$$\gamma_n = \frac{\int g_n(\vec{x})[g_n(\vec{x})-g_{n-1}(\vec{x})]d(\vec{x})}{\int g_{n-1}(\vec{x})g_{n-1}(\vec{x})d(\vec{x})} \tag{17}$$

`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:

$$\alpha_n = \frac{\mathrm{Re}\left\{\int\int\int r^{\star}_{n-1}(\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)[\mathcal{K}\zeta_n](\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)d\vec{x_s}d\vec{x_r}d\omega\right\}}{\int\int\int \mid [\mathcal{K}\zeta_n](\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)\mid^2 d\vec{x_s}d\vec{x_r}d\omega} \tag{18}$$

`optimalStepSizeCG`

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

$$g_1(\vec{x}) = \eta\mathrm{Re}\left\{[\mathcal{K}^{\star}p_{\mathrm{data}}](\vec{x})\right\} \tag{19}$$

`gradientRunc`

and we get the first update parameter as :

$$\alpha_1 = \frac{\mathrm{Re}\left\{\int\int\int p^{\star}_{\mathrm{data}}(\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)[\mathcal{K}g_1](\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)d\vec{x_s}d\vec{x_r}d\omega\right\}}{\int\int\int \mid [\mathcal{K}g_1](\mathbf{x}_{\mathrm{r}},\mathbf{x}_{\mathrm{s}},\omega)\mid^2 d\vec{x_s}d\vec{x_r}d\omega} \tag{20}$$

`firstStepSize`

3.2.2 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 $\mathcal{F}^{reg}$:

$$\mathcal{F}_n^{tot} = \mathcal{F}_n^{data}\mathcal{F}_n^{reg} \tag{21}$$

`errorFuncRegul`

The following weighting function is used:

$$b_n^2 = (\int_{\vec{x}} d\vec{x})^{-1}(\mid \nabla\chi^{n-1} \mid^2 + \delta_{n-1}^2)^{-1} \tag{22}$$

`errorFuncRegulWeighting`

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

$$\delta_n^2 = (\int_{\vec{x}} d\vec{x})^{-1} \int_{\vec{x}} \mid \nabla\chi^{n-1} \mid^2 d\vec{x} \tag{23}$$

`errorFuncRegulSteering`

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

$$\mathcal{F}_n^{tot} = (\mathcal{A}_2\alpha_n^2 + \mathcal{A}_1\alpha_n + \mathcal{A}_0)(\mathcal{B}_2\alpha_n^2 + \mathcal{B}_1\alpha_n + \mathcal{B}_0) \tag{24}$$

`errorFuncFourthOrder`

with the constants A_0, ..., A_2, B_2 in appendix 7.2.

3.2.3 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 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:

$$\phi_n(\mathbf{x}_\mathrm{r}, \mathbf{x}_\mathrm{s}, \omega) = \int_{\vec{x}\in D} \mathcal{G}(\mathbf{x}_\mathrm{r}, \mathbf{x}_\mathrm{s}, \omega)\partial\mathcal{W}_n d\vec{x}' \tag{25}$$

`eq:buildField, see calcField.h and rel green_rect_2D_cpu.h`

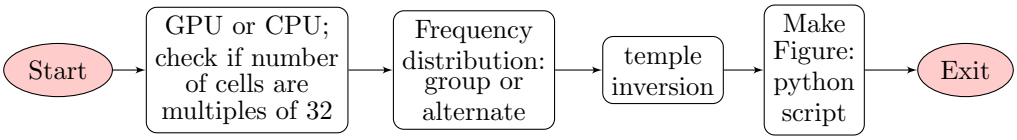

Figure 3: High Level activity diagram for main.cpp

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

$$\partial\mathcal{W}_1 = \chi^{(1)} p_{tot}^0, \mathcal{W}_n = \chi^{(n)} p_{tot}^{n-1} - \chi^{(n-1)} p_{tot}^{n-2}, n > 1. \tag{26}$$

`eq:incrementalContrastSrcs, see calcField.h`

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

$$p_{tot}^{(N)}(\mathbf{x}_\mathrm{r}, \mathbf{x}_\mathrm{s}, \omega) = p_0(\mathbf{x}_\mathrm{r}, \mathbf{x}_\mathrm{s}, \omega) + \sum_{N=1}^{N} \alpha_n^{(N)}(\mathbf{x}_\mathrm{s}, \omega)\phi_n(\mathbf{x}_\mathrm{r}, \mathbf{x}_\mathrm{s}, \omega) \tag{27}$$

`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`, 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 and 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 3.2.3.

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 3.2.3 are calculated. At first the incremental contrast sources are defined according to equation 26 and then the ϕ_n is constructed. Based on this 27 is constructed. This step concludes the forward modelling step.

Based on the input parameters, α is used if α 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 3.2.1 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 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 $1 - (\frac{2000}{2218})^2$. The Results obtained from running the FWI code can be seen in the following figures:

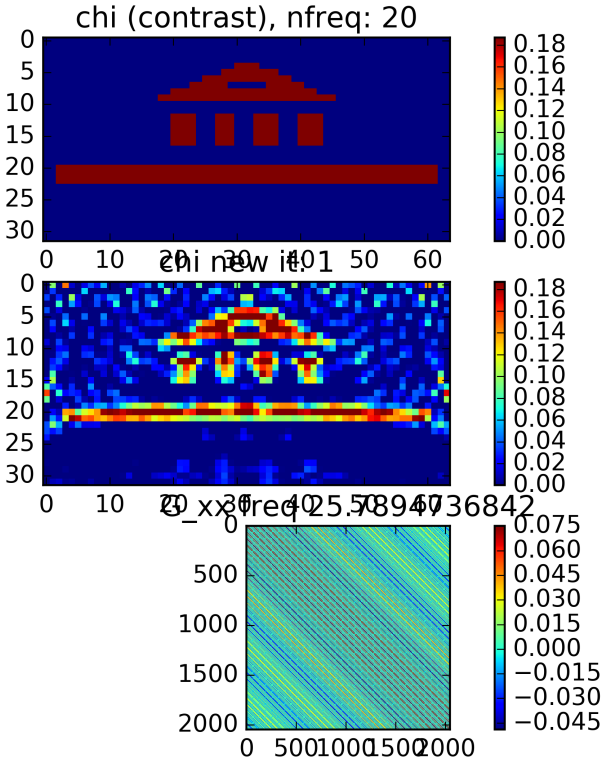

Figure 4: Chi estimation Result 1

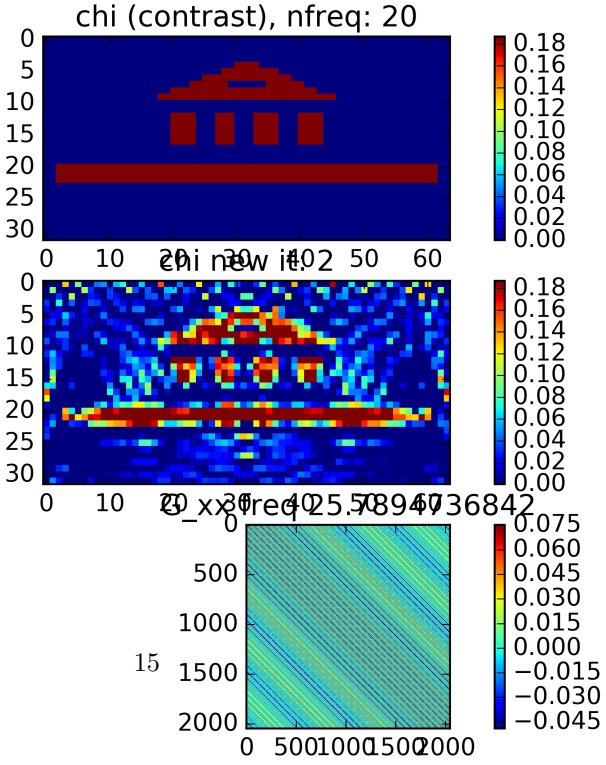

Figure 5: Chi estimation Result 2

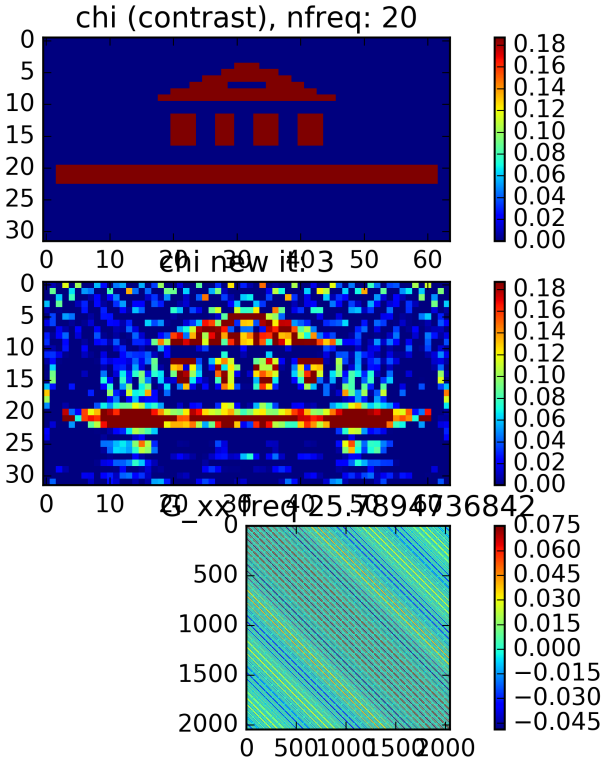

Figure 6: Chi estimation Result 3

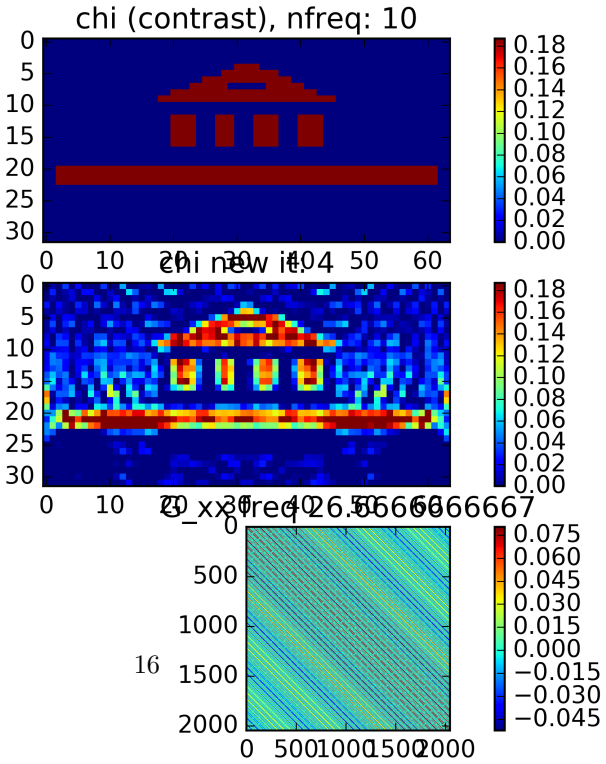

Figure 7: Chi estimation Result 4

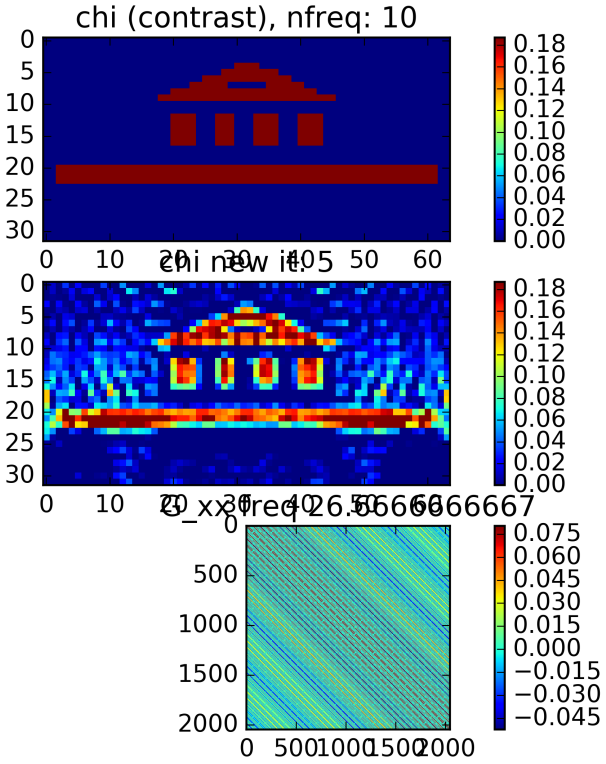

Figure 8: Chi estimation Result 5

7 Appendices

7.1 Derivation of the functional derivative of the error functional

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

7.2 Prefactors of the Total Cost Equation

$$\mathcal{A}_2 = \eta\int\int\int \mid \mathcal{K}\zeta_n \mid^2 d\vec{x_s}d\vec{x_r}d\omega \tag{28}$$

`totalCostA2`

$$\mathcal{A}_1 = -2\eta\mathrm{Re}\left\{\int\int\int r^\star_{n-1} \mid \mathcal{K}\zeta_n \mid d\vec{x_s}d\vec{x_r}d\omega\right\}, \hat{E} \tag{29}$$

`totalCostA1`

$$\mathcal{A}_0 = \eta\int\int\int \mid r_{n-1} \mid^2 d\vec{x_s}d\vec{x_r}d\omega = \mathcal{F}^{data}_{n-1}, \hat{E} \tag{30}$$

`totalCostA0`

$$\mathcal{B}_2 = \mid\mid b_n\nabla\zeta_n \mid\mid^2_D, \hat{E} \tag{31}$$

`totalCostB2`

$$\mathcal{B}_1 = 2 < b_n\nabla\chi^{n-1}, b_n\nabla\zeta_n >_D, \hat{E} \tag{32}$$

`totalCostB1`

$$\mathcal{B}_0 = \mid\mid b_n\nabla\chi^{n-1} \mid\mid^2_D + \delta^2_{n-1} \mid\mid b_n \mid\mid^2_D \tag{33}$$

`totalCostB0`