Current implementation of Conjugate Gradient (with Regularisation)

ALTEN Netherlands
Morelli, L.

June 23, 2020

Contents

1 Preface

Here we will describe the current implementation of ConjugateGradientInversion and its refactored version ConjugateGradientWithRegularisationCalculator, highlighting the differences between the documentation (`.../doc/ReadMe/1_ProjectDescription.pdf` and `.../doc/BackGroundInfo/phd-Peter-Haffinger-seismic-broadband-full-waveform-inversion.pdf`) and the code.
Throughout this document, we will refer to the PhD thesis from Peter Haffinger simply as 'Thesis'.
The main body of the text is taken from `.../doc/ReadMe/1_ProjectDescription.pdf`, and mostly subsection 3.3 will be modified (some pieces have been added to the other sections for clarification).
There might be a discrepancy in a few variables' names, as during the refactor some have been made member variables of the class, thus obtaining a $'\_'$ character at the beginning of their names.
Moreover, whenever Line numbers are mentioned, we refer to the original **conjugateGradientInversion.cpp** file.

2 Introduction

Please consult `.../doc/ReadMe/1_ProjectDescription.pdf` for the full document. The only section that has been modified **subsection 3.3.2**, while we added **Section 4: Conclusions** to summarize the findings of this document.

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:

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

`eq:calculateData, see ConjugateGradientInversion.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

[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.

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 Helmholtz equation

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

$$\nabla^2 u(\vec{x}) + k(\vec{x})^2 u(\vec{x}) = -f(\vec{x}) \tag{2}$$

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 $u(\vec{x}) = u_0(\vec{x}) + u_{\text{ind}}(\vec{x})$, where $u_0(\vec{x})$ is defined as the field given by the background velocity and external sources,

$$\nabla^2 u_0(\vec{x}) + k_0^2 u_0(\vec{x}) = -f(\vec{x}) \tag{3}$$

step:splitHelmholtzU0

$$\nabla^2 u_{\text{ind}}(\vec{x}) + k_0^2 u_{\text{ind}}(\vec{x}) = -f_{\text{ind}}(\vec{x}), \tag{4}$$

step:splitHelmholtzUind

with $f_{\text{ind}}(\vec{x}) = k_0^2 \chi(\vec{x}) u_0(\vec{x})$ the induced source term.

3.1.2 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 equation (2) is defined as the solution $\mathcal{G}(\vec{x}, \vec{y})$ of the equation

$$\nabla^2 \mathcal{G}(\vec{x}, \vec{y}) + k_0^2 \mathcal{G}(\vec{x}, \vec{y}) = -\delta(\vec{x} - \vec{y}), \tag{5}$$

step:GeneralGreensFunc

with $\delta(\vec{x} - \vec{y})$ being the Dirac's delta. The solution to the Helmholtz equation (2) is then given by

$$u(\vec{x}) = \int_{\vec{y} \in \mathbb{R}^2} \mathcal{G}(\vec{x}, \vec{y}) f(\vec{y}) \, \mathrm{d}\vec{y} \tag{6}$$

step:GeneralGreensSol

The Green's function is in general very complicated and rarely explicit. In this case however it can be expressed as

$$\mathcal{G}(\vec{x},\vec{y}) = \frac{i}{4}H_0^{(1)}(k_0\|\vec{x}-\vec{y}\|) = -\frac{1}{4}Y_0(k_0\|\vec{x}-\vec{y}\|) + \frac{i}{4}J_0(k_0\|\vec{x}-\vec{y}\|) \tag{7}$$

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

3.1.3 The Contrast function

The contrast function is defined as:

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

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 solve the same equation as (6) each step.

3.2 Finite Difference Approach

Instead of solving the Helhmoltz 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:

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

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

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

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 α 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 α needs tweaking based on ω. 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 have been recently implemented.[2] The user can choose which approach to use by adjusting input card parameters. In practice, ABC perform better than PML. Second order ABC should prove to 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 $n(\vec{x}) \equiv C_1$ 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 [1], i.e.

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

where $\hat{\mathbf{n}}$ denotes the outward pointing normal of the boundary. Now consider, for example, the upper boundary Γ_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

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

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

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

The ABC formulated above are first order accurate in the angle at which the wave strikes the boundary [3].

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

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

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

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

[2] For the theory, see appendix B.

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

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

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

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

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

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

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 [4]. The following coordinate transformation is used for both x and z.

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

where

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

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

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

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.

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

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

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

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

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

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

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 do due rounding the source positions to the nearest grid point. Especially when combing 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 [7]. Basically, the source function used is given by

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

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

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

where $I_0(y)$ denotes the Bessel function of the first kind. The parameters β 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 [7].

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

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_{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:

$$r(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega) = p_\mathrm{data}(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega) - [\mathcal{K}_\chi]\,(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega), \tag{30}$$

`residualMeasureModel`

with

$$[\mathcal{K}_\chi]\,(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega) = \int \mathcal{G}(\vec{x}_\mathrm{r}, \vec{x}, \omega) p_\mathrm{tot}(\vec{x}, \vec{x}_\mathrm{s}, \omega) \chi(\vec{x})\,\mathrm{d}\vec{x} \tag{31}$$

`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

$$F(\chi) = \eta \int |r(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega)|^2\,\mathrm{d}\vec{x}_\mathrm{s}\,\mathrm{d}\vec{x}_\mathrm{r}\,\mathrm{d}\omega, \tag{32}$$

`errorFunc`

with

$$\eta^{-1} = \int |p_\mathrm{data}(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega)|^2\,\mathrm{d}\vec{x}_\mathrm{s}\,\mathrm{d}\vec{x}_\mathrm{r}\,\mathrm{d}\omega, \tag{33}$$

`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:

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

`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 (with ζ being the greek letter *zeta*)

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

`updateDirectionsCG`

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

$$\begin{aligned} \frac{\partial F(\chi)}{\partial \mathbf{d}} &= \lim_{\varepsilon \to 0} \frac{F(\chi + \varepsilon \mathbf{d}) - F(\chi)}{\varepsilon} \\ &= -2\eta \int \mathrm{Re}\left\{ [\mathcal{K}_\mathbf{d}]\,(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega)^\dagger r(\vec{x}_\mathrm{r}, \vec{x}_\mathrm{s}, \omega) \right\} \mathrm{d}\vec{x}_\mathrm{s}\,\mathrm{d}\vec{x}_\mathrm{r}\,\mathrm{d}\omega \end{aligned} \tag{36}$$

`functionalDerWRTD`

The derivation of this equation is provided in section A.1.
For the discrete $\mathcal{K}$ operator the integrand will have the form

$$g_n(\vec{x}) = \eta \mathrm{Re}\left\{[\mathcal{K}^\star \mathbf{r}_{n-1}](\vec{x})\right\} \tag{37}$$

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,

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

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:

$$\alpha_n = \frac{\mathrm{Re}\left\{\int\int\int r^\star_{n-1}(\vec{x_r}, \vec{x_s}, \omega)[\mathcal{K}\zeta_n](\vec{x_r}, \vec{x_s}, \omega) d\vec{x_s} d\vec{x_r} d\omega\right\}}{\int\int\int \mid [\mathcal{K}\zeta_n](\vec{x_r}, \vec{x_s}, \omega) \mid^2 d\vec{x_s} d\vec{x_r} d\omega} \tag{39}$$

optimalStepSizeCG

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

$$g_1(\vec{x}) = \eta \mathrm{Re}\left\{[\mathcal{K}^\star p_\mathrm{data}](\vec{x})\right\} \tag{40}$$

gradientRunc

and we get the first update parameter as :

$$\alpha_1 = \frac{\mathrm{Re}\left\{\int\int\int p^\star_\mathrm{data}(\vec{x_\mathrm{r}}, \vec{x_\mathrm{s}}, \omega)[\mathcal{K}g_1](\vec{x_\mathrm{r}}, \vec{x_\mathrm{s}}, \omega) d\vec{x_s} d\vec{x_r} d\omega\right\}}{\int\int\int \mid [\mathcal{K}g_1](\vec{x_\mathrm{r}}, \vec{x_\mathrm{s}}, \omega) \mid^2 d\vec{x_s} d\vec{x_r} d\omega} \tag{41}$$

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

$$\mathcal{F}^{tot}_n = \mathcal{F}^{data}_n \mathcal{F}^{reg}_n. \tag{42}$$

errorFuncRegul

A first remark is that in the implemented code it is absolutely not clear which variables refer to $\mathcal{F}^{data}$ and which to $\mathcal{F}^{reg}$.
The structure of the algorithm is going to be heaviliy modified. During every iteration in the *main loop*, indexed by n, we will perform the following operations:

1. (Line 39) using the most recent approximation of χ_{est} (for $n = 0$ we have $\chi_{est} \equiv 0$), we update $[\mathcal{K}_\chi]$ following eq. (31).
2. (Line 40) Update the residual r_n following eq. (30).
3. (Line 43) Update the undocumented parameter $\delta_n^{ampl} = \frac{\delta_{start}^{ampl}}{(n\ \delta_{slope}^{ampl})+1}$.
4. (Line 52) update ζ_n according to eq. (35). At the moment the default values are $\delta_{start}^{ampl} = 100$, $\delta_{slope}^{ampl} = 10$. This way of updating the direction (which actually corresponds to g_n in this document) is valid only for the first iteration ever of the algorithm, although here it is clearly applied n times.
5. (Line 53) α_n is updated according to eq. (39) (no regularisation so far).
6. (Line 56) We update χ_n according to eq. (34).
7. (Line 59) We update the residual array and its squared norm using the new χ_n.
8. (Lines 70-106) We enter in an undocumented *inner loop* indexed by $it1$, but that we will index with $1 \leq j < ConjugateGradientWithRegularisationParametersInput.\_nRegularisationIterations$ for conciseness.

Since most all of the 'regularisation' part is computed within the *inner loop*, it seems legit asking what is the purpose of the *main loop*, which also contains operations that should be performed only once (see point 4. of above list). The only part that is actually unique to the *main loop* is the updating of the δ_n^{ampl} parameter, used in the computation of the Steering Factor in **calculateSteeringFactor()**.
Moreover, as in the inner loop the direction *zeta* is updated along with the stepSize *alpha*, it required modifications to the whole *StepAndDirection* refactoring, such as creating a class that is both *DirectionCalculator* and *StepSizeCalculator*, thus creating an inheritance diamond issue and forcing to modify the factory for this 'unsplittable' algorithm. If we were however to follow the algorithm in the Thesis, it could be implemented simply as a *StepSizeCalculator*, which would fit extremely well with the new design. Please note that currently the regularisation algorithm described in Thesis is not present anywhere in the project.

3.3.2.1 Inner Loop

Once we are inside the inner loop indexed by j, we actually take over the enumeration of most quantities already introduced in the *main loop* such as $\chi_{est}, \alpha, \zeta$ and more. Also this mixture of indexing, which is mathematically speaking bad practice, seems to point to the fact that the *main loop* should not in fact exist.
First, we compute the Regularisation Parameters $b_j^2(\vec{x})$[4], δ_j^2 and $\_regularisationCurrent.gradient$ by invoking the method **calculateRegularisationParameters() (Line 72)**:
The following weighting function is used (via **calculateWeightingFactor()**):

[4]please note that b_j is actually a function and not a scalar.

$$b_j^2(\vec{x}) = (\int_{\vec{x}} d\vec{x})^{-1}(\mid \nabla\chi_{j-1}(\vec{x}) \mid^2 + \delta_{j-1}^2)^{-1}. \tag{43}$$

`errorFuncRegulWeighting`

In the above equation, the δ_j^2 is the steering factor and should be defined as:

$$\delta_j^2 = (\int_{\vec{x}} d\vec{x})^{-1} \int_{\vec{x}} \mid \nabla\chi_{j-1}(\vec{x}) \mid^2 d\vec{x}, \tag{44}$$

`errorFuncRegulSteering`

while in fact is (questionably) computed in **calculateSteeringFactor()** as:

$$\delta_j^2 = \frac{1}{2}\delta_n^{ampl} \frac{\int_{(x,z)=\vec{x}} \left(((b_j(\vec{x})\partial_x\chi_{j-1}(\vec{x}))^2 + (b_j(\vec{x})\partial_z\chi_{j-1}(\vec{x}))^2\right) d\vec{x}}{\int_{\vec{x}} b_j^2(\vec{x})d\vec{x}}. \tag{45}$$

Bringing together the $b_j^2(\vec{x})$ in the upper integral, we obtain

$$\delta_j^2 = \frac{1}{2}\delta_n^{ampl} \left(\int_{\vec{x}} b_j^2(\vec{x})d\vec{x}\right)^{-1} \int_{\vec{x}} b_j^2(\vec{x}) \mid \nabla\chi_{j-1}(\vec{x}) \mid^2 d\vec{x}, \tag{46}$$

which vaguely resembles eq. (44).

Observe that here the parameter δ_n^{ampl} keeps the n index. The last variable that is going to be updated by **calculateRegularisationParameters()** is *_regularisationCurrent.gradient*, by invoking **calculateRegularisationGradient()**:

$$\_regularisationCurrent.gradient = \partial_x(b_{j-1}^2(\vec{x})\partial_x\chi_{j-1}(\vec{x})) + \partial_z(b_{j-1}^2(\vec{x})\partial_z\chi_{j-1}(\vec{x})). \tag{47}$$

The presence of these second order derivatives is not very clear, but the method calls a *.gradient* (Lines 220, 224) on quantities to which was already invoked another *.gradient* (Line 98, end of *inner loop*). Moreover, calling a variable *gradient* without mentioning it is multiplied by b_{j-1}^2 is quite misleading.

We then proceed to update ζ_j by computing g_j according to eq. (37) in the method **calculateUpdateDirectionRegularisation()**, in which also $gradientCurrent = gradient_j$ gets updated as

$$\begin{aligned} \_gradient_j =& \eta(\_regularisation_{j-1}.errorFunctional)\mathrm{Re}\left\{[\mathcal{K}^\star \mathbf{r}_{n-1}]\right\}(\vec{x}) + & (48)\\ &+ (||r_{j-1}||^2)(\_regularisation_j.gradient), & (49)\end{aligned}$$

with $\_regularisation_{j-1} = \_regularisationPrevious$, $\_regularisation_j = \_regularisationCurrent$ and[5]

$$\_regularisation_{j-1}.errorFunctional = (\int_{\vec{x}} d\vec{x})^{-1} \int_{\vec{x}} \left(\frac{(|\nabla\chi_j\vec{x}|^2 + \delta_{j-1}^2)}{(|\nabla\chi_{j-1}\vec{x}|^2 + \delta_{j-1}^2)} d\vec{x}\right)(cellVolume), \tag{50}$$

(calculateRegularisationErrorFunctional())

[5]the variables *_regularisationPrevious* and *_regularisationCurrent* are declared in the first part of the *main loop* and the method **calculateRegularisationErrorFunctional()** is called only at the end of the *inner loop*, which means that every time we start the *inner loop* the quantity *_regularisationPrevious.errorFunctional* is reinitialized as a *double 0.0*.

where $cellVolume$ represents the volume of a discretisation cell coming from $\_grid.getCellVolume()$. Besides the multiplication for this $cellVolume$ term, it perfectly matches eq, (2.21) from Thesis.

We now compute new direction ζ_j by invoking the method **calculateUpdateDirectionRegularisation()**, which computes $g_j(\vec{x})$ by means of

$$gradient_j = \eta(\_regularisation_{j-1}.errorFunctional)\ \mathrm{Re}\{[\mathcal{K}^\star \mathbf{r}_{n-1}]\} \tag{51}$$
$$+ (residual_{j-1})(\_regularisation_j.gradient), \tag{52}$$

which represents eq. (2.25) from Thesis. The first term on the right hand side of the above equation is easily recognizable, while in the second term we see that $residual_{j-1} = residualPrevious = ||r_{j-1}||^2$ is linearly dependent to F^{data} in eq. (2.8) from Thesis. Regarding the $\_regularisation_j.gradient$, it corresponds to the term described in equation (2.24) from Thesis, although its name is fairly misleading (it is actually a gradient, but it is not specified of what).
Once we have obtained $gradient_j = gradientCurrent$, we compute γ_j using the Polak-Ribiere formula (38) and we obtain the new ζ_j following eq. (35).
We then invoke **calculateStepSizeRegularisation()** where the parameters $\mathcal{A}_0$, $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{B}_0$, $\mathcal{B}_1$ and $\mathcal{B}_2$ that represent the coefficients of the Taylor expansion for $\mathcal{F}_j^{data}$ (the $\mathcal{A}$'s) and $\mathcal{F}_j^{reg}$ (the $\mathcal{B}$'s) are computed.
The new total cost functional then becomes a product of two second order polynomials:

$$\mathcal{F}_j^{tot}(\alpha_j) = (\mathcal{A}_2\alpha_j^2 + \mathcal{A}_1\alpha_j + \mathcal{A}_0)(\mathcal{B}_2\alpha_j^2 + \mathcal{B}_1\alpha_j + \mathcal{B}_0) \tag{53}$$

`errorFuncFourthOrder`

with the constants $\mathcal{A}_i$, $\mathcal{B}_i$ as in appendix A.2.
Finally, we find the minimum of eq. (53) by computing

$$\partial_\alpha \mathcal{F}_j^{tot}(\alpha) = 4\alpha^3(\mathcal{A}_2\mathcal{B}_2) + 3\alpha^2(\mathcal{A}_2\mathcal{B}_1 + \mathcal{A}_1\mathcal{B}_2) + 2\alpha(\mathcal{A}_2\mathcal{B}_0 + \mathcal{A}_1\mathcal{B}_1 + \mathcal{A}_0\mathcal{B}_2) + \mathcal{A}_1\mathcal{B}_0 + \mathcal{A}_1\mathcal{B}_1 = 0, \tag{54}$$

which is computed by finding a real root using the procedure described in subsection (3.8.2) of *Abramowitz, M., and Stegun, I. A., 1970* from Thesis' bibliography). It is not clear how we are sure that we find the 'correct' real root, as this polynomial could very well possess three distinct real roots. Moreover, the operations have been unnecessarily modified (although obtaining the expected final value), resulting in arbitrary variables' names (very unclear) and sign and constant multiplications that have no reasons to be. A simple fix to this issue could be checking whether the other two roots are also real (easily doable with the implemented computation), and in that case return all three of them, so that afterwards the one actually minimizing eq. (53) can be picked.
The new $chiEstimate$ is finally updated by means of $chiEstimate+ = alpha * zeta$ (Line 79), followed by an update of $residualCurrent$ (containing the square norm of the $residual$, Line 82) and $\_regularisationCurrent.gradientChi$ (Line 98).

3.3.2.2 Unvalidated parts and regression tests

A first tentative way to justify the presence of the unexpected computations in the code is to study how their presence/absence/variation affects the performances of the regression tests.

The paremeters for the **conjugateGradientInversion** that we will focus on are (together with their *default* values):

1. the range of the outer loop (indicated by $Iter1.n$) $n_{max} = 10$ and of the inner loop (indicated by n_max) $j_{max} = 5$,
2. the δ^{ampl} parameters ($\delta^{ampl}_{start} = 100.0$, $\delta^{ampl}_{slope} = 10.0$),
3. The presence of $b_j^2(\vec{x})$ in eq. (46),
4. the presence of the multiplicative $cellVolume$ in eq. (50).

There is an outdated *bool do_reg* that was used to indicate whether or not we wanted the regularisation process. Now a $j_{max} = 0$ is used instead.

Presence of inner loop Many hints inside of the code show that the outer loop should be run only once, thus making it redundant in the first place. A set of regression tests ran with $n_{max} = 1$, $j_{max} = 33$ brought to a computational time in line with the default, but on average worse precision.

However, when running a test with $n_{max} = 50$, $j_{max} = 0$ (and in particular no regularisation), we find on average much better results for the regression tests. The computational time improves, and the precision too (except for *smallgrid* and *layers*, where it performs much worse). Increasing n_{max} to 55 brings to comparable running time and slightly improved precision (except for *smallgrid* and *layers* where it actually worsens even more).

δ^{ampl} parameters Comparing equations (44) and (46), a regression test was tried with parameters 2.0 and 0.1 (they cannot be chosen ≤ 0). Unfortunately, most of the regression tests showed worsened performances in precision and/or time.

Presence of $b_j^2(\vec{x})$ in eq. (46) such a modification of the metric could in some cases improve stability, but only when replacing some metric that could have values close to 0 or ∞ with something more regular. In this case, the constant function 1 was substituted. A comparison of regression tests while removing the b_j^2 from eq. (46) against the default version (both with default parameters) brought to an overall worsening of performance when removing the b_j^2 parameters.

***cellVolume* in (50)** By comparing the default regression tests with the ones obtained after removing the term $cellVolume$ from eq. (50) we observe an overall increase in precision (slight decrease in *manyfreq* and *highquality*), especially in the *smallgrid* test, where we finally manage to obtain a positive value for the VAF and FIT coefficients (3% and 1.7% respectively, compared to the benchmark -7% and -3%)[6].

A combination of removing the $cellVolume$ multiplication in (50) and eliminating the influence of δ^{ampl} (by setting $\delta^{ampl}_{start} = 2$, $\delta^{ampl}_{slope} = 0.01$) worsened the overall performance, except for the *smallgrid* test, where we actually reached a VAF value of 14% and a FIT value of 7%.
If we instead run the regression tests without $cellVolume$ together with $n_{max} = 25$, $j_{max} = 2$ (for a total of 50 steps) we find that every single test has improved precision and slight faster

[6] these parameters are actually defined as the maximum between 0 and the outcome of some operations, but we have kept negative values to monitor these poorly performing cases.

processing. This combination is also more precise than the benchmark simulations in 6 tests out of 8 (excluding *small grid* and *High quality*, where the values are however comparable).

4 Conclusions

While researching the origin of the undocumented parts of the code I realized that these parts have been implemented since a long time, since the beginning of the git tree.
The presence of these not validated parts might be a trial and error procedure to improve the performances of the inversion process, but this is not clear.
This makes the validation of the implemented code extremely hard, since there is no mathematical proof that those undocumented parts improve the performances, either in speed or precision.
During the upcoming Validation Epics, User Stories should be added to understand the presence of these terms, and either justify them or remove them.
Moreover, due to the presence of the double loop in **conjugateGradientInversion.cpp**, its only possible refactoring in the new 'StepAndDirection' style created an inheritance diamond problem, also forcing to create a customized method in the factory.
An implementation of the Conjugate Gradient with Regularisation algorithm that follows the existing documentation could be implemented simply as a 'StepSizeCalculator', which would satisfy the very reason behind the refactoring, that is splitting 'Directions' from 'StepSizes'. The extimated time to create such new class would approximately be a couple of days, as most of the code can be recycled. Note that at the moment there is no test for the 'StepSize' part.
A validated version of this algorithm could also bring more meaning to regression tests and in future to performance comparison of different algorithms.

A Appendices

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

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

A.2 Prefactors of the Total Cost Equation

$$\mathcal{A}_2 = \eta \int\int\int \mid \mathcal{K}\zeta_n \mid^2 d\vec{x_s} d\vec{x_r} d\omega \tag{55}$$

`totalCostA2`

$$\mathcal{A}_1 = -2\eta \mathrm{Re}\left\{ \int\int\int r^{\star}_{n-1} \mid \mathcal{K}\zeta_n \mid d\vec{x_s} d\vec{x_r} d\omega \right\}, \tag{56}$$

`totalCostA1`

$$\mathcal{A}_0 = \eta \int\int\int \mid r_{n-1} \mid^2 d\vec{x_s} d\vec{x_r} d\omega = \mathcal{F}^{data}_{n-1}, \tag{57}$$

`totalCostA0`

$$\mathcal{B}_2 = \mid\mid b_n \nabla \zeta_n \mid\mid^2_D, \tag{58}$$

`totalCostB2`

$$\mathcal{B}_1 = 2 < b_n \nabla \chi^{n-1}, b_n \nabla \zeta_n >_D, \tag{59}$$

`totalCostB1`

$$\mathcal{B}_0 = \mid\mid b_n \nabla \chi^{n-1} \mid\mid^2_D + \delta^2_{n-1} \mid\mid b_n \mid\mid^2_D \tag{60}$$

`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 [3]. They can be formulated as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - in\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial \mathbf{s}^2} = 0, \tag{61}$$

where $\mathbf{s}$ denotes the vector tangential to the boundary (positive or negative direction does not matter). In the corner specials ABC apply [2]

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = 0, \tag{62}$$

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. (61) can be discretized as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - in\omega u - \frac{i}{2\omega n}\frac{\partial^2 u}{\partial \mathbf{s}^2} = -\frac{\partial u}{\partial z} - in\omega u - \frac{i}{2\omega n}\frac{\partial^2 u}{\partial x^2} = 0 \tag{63}$$

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

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

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 $(0, 0)$.

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = -\frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2} i\omega n u = 0 \tag{66}$$

$$\rightsquigarrow \quad -\frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j}}{h_z} - \frac{3}{2} i\omega n_{ij} u_{ij} = 0 \tag{67}$$

$$\implies \quad u_{i-1,j} = (3h_x i\omega n_{i,j} - 2\frac{h_x}{h_z})u_{i,j} + u_{i+1,j} + 2\frac{h_x}{h_z} u_{i,j+1} = 0 \tag{68}$$

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

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = \frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2} i\omega n u = 0 \tag{69}$$

$$\rightsquigarrow \quad \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j}}{h_z} - \frac{3}{2} i\omega n_{ij} u_{ij} = 0 \tag{70}$$

$$\implies \quad u_{i,j+1} = (3h_x i\omega n_{i,j} - 2\frac{h_x}{h_z})u_{i,j} + u_{i-1,j} + 2\frac{h_x}{h_z} u_{i,j+1} = 0 \tag{71}$$

References

[1] Olof Runborg, Numerical Solutions of Differential Equation – Lecture Notes, 2012.

[2] Alain Bamberger, Patrick Joly and Jean E. Roberts, Second-Order Absorbing Boundary Conditions for the Wave Equation: A Solution for the Corner Problem,

[3] Bjorn Engquist and Andrew Majda, Absorbing boundary conditions for the numerical simulation of waves, 1977.

[4] Yogi A. Erlangga, Advances in Iterative Methods and Preconditioners for the Helmholtz Equation, 2008.

[5] , Yingjie Gao, Hanjie Song, Jinhai Zhang and Zhenxing Yao, Comparison of artificial absorbing boundaries for acoustic wave equation modelling, 2017.

[6] Andreas Fichtner, Full Seismic Waveform Modelling and Inversion, 2011.

[7] J. Hicks, Graham, Arbitrary source and receiver positioning in finite-difference schemes using Kaiser windowed sinc function, 2002.