Current implementation of Conjugate Gradient (with Regularisation)

ALTEN Netherlands
Morelli, L.

June 5, 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 as the PhD thesis from Peter Haffinger simply as 'Thesis'.
The main body of the text is taken from `.../doc/ReadMe/1_ProjectDescription.pdf`, and only subsection 3.3 will be modified.
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 parts that have been added are highlighted in red and appear in subsection 3.3.2.

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.

[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 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

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:

$$\chi(\mathbf{x}) = 1 - \left(\frac{c0(\vec{x})}{c(\vec{x})} \right)^2. \tag{8}$$

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 (3) 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 [**?**]. Both the first order ABC and PML are implemented. Second order ABC are not implemented.[2] 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 [**?**], i.e.

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

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 [**?**].

3.2.2 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

[2] For the theory, see appendix B.

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

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

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

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

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

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 [**?**]. 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.

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

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

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

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

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

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

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 [**?**]. 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 [**?**].

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.

3.2.6 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 β for the sine source implementation as given in the section above and [**?**]. 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_{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_{data} can be implemented by implementing the receivers via the method given in [**?**]. 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_{tot} and p_{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_{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:

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

`residualMeasureModel`

with

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

`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

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

`errorFunc`

withp

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

`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

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

`updateDirectionsCG`

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

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

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:

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

`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 :

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

`gradientRunc`

and we get the first update parameter as :

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

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

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

`errorFuncRegul`

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). During this operation what is computed is actually eq. (31) assuming $\chi \equiv 1$. When invoking $\_forwardModel \rightarrow calculateKappa()$ there are no mentions of χ, but uniquely to $\mathcal{G}$ and p^{tot}.

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). This way of updating the direction (which is actually called g in the documents) 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()**):

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

`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

$$\_gradient_j = \eta(\_regularisation_{j-1}.errorFunctional)\mathrm{Re}\left\{[\mathcal{K}^\star \mathbf{r}_{n-1}]\right\}(\vec{x}) + \tag{48}$$

$$+ (||r_{j-1}||^2)(\_regularisation_j.gradient), \tag{49}$$

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

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())

where $cellVolume$ represents the volume of a discretisation cell coming from $\_grid.getCellVolume()$. Observe how the formula for $errorFunctional_{(j)}$ resembles eq. (2.21) from Thesis for the computation of F_n^{reg}, although the *.summation()* operation is done on the whole fraction at once rather than for numerator and denominator separately.

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 is claimed to be representing eq. (2.25) from Thesis. The first term on the right hand side of the above equation matches, 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 $\_regularisation_j.gradient$, we can see that in eq. (47) this variable more or less contains a b_{j-1}^2 term, although inside of a compositve derivative.
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 A_0, ..., A_2, B_2 in appendix A.2.
Finally, we find the minimum of eq. (53) by computing $\partial_\alpha \mathcal{F}_j^{tot} = 0$, 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.
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)

[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*.

and *_regularisationCurrent.gradientChi* (Line 98).
There is another Bug (Line 99, just noticed), where a function that should update the *_regularisationCurrent* and *_regularisationPrevious* is actually invoked with two instances of *_regularisationPrevious*, thus destroying any hope of validity of the algorithm.

A Appendices

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

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

A.2 Prefactors of the Total Cost Equation

$$\mathcal{A}_2 = \eta \int \int \int \mid \mathcal{K}\zeta_n \mid^2 d\vec{x_s} d\vec{x_r} d\omega \tag{54}$$

`totalCostA2`

$$\mathcal{A}_1 = -2\eta \mathrm{Re} \left\{ \int \int \int r^\star_{n-1} \mid \mathcal{K}\zeta_n \mid d\vec{x_s} d\vec{x_r} d\omega \right\}, \hat{E} \tag{55}$$

`totalCostA1`

$$\mathcal{A}_0 = \eta \int \int \int \mid r_{n-1} \mid^2 d\vec{x_s} d\vec{x_r} d\omega = \mathcal{F}^{data}_{n-1}, \hat{E} \tag{56}$$

`totalCostA0`

$$\mathcal{B}_2 = \mid\mid b_n \nabla \zeta_n \mid\mid^2_D, \hat{E} \tag{57}$$

`totalCostB2`

$$\mathcal{B}_1 = 2 < b_n \nabla \chi^{n-1}, b_n \nabla \zeta_n >_D, \hat{E} \tag{58}$$

`totalCostB1`

$$\mathcal{B}_0 = || \; b_n \nabla \chi^{n-1} \; ||_D^2 + \delta_{n-1}^2 \; || \; b_n \; ||_D^2 \tag{59}$$

`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 [**?**]. They can be formulated as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - in\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial \mathbf{s}^2} = 0, \tag{60}$$

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

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = 0, \tag{61}$$

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. (60) can be discretized as follows.

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} - in\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial \mathbf{s}^2} = -\frac{\partial u}{\partial z} - in\omega u - \frac{i}{2\omega n} \frac{\partial^2 u}{\partial x^2} = 0 \tag{62}$$

$$\rightsquigarrow \quad -\frac{u_{i,j+1} - u_{i,j-1}}{2h_z} - i\omega n_{ij} u_{ij} - \frac{i}{2\omega n_{ij}} \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_x^2} = 0 \tag{63}$$

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

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 $(0, 0)$.

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = -\frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2} i\omega n u = 0 \tag{65}$$

$$\rightsquigarrow \quad -\frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j}}{h_z} - \frac{3}{2} i\omega n_{ij} u_{ij} = 0 \tag{66}$$

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

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

$$\frac{\partial u}{\partial \hat{\mathbf{s}}} - \frac{3}{2} i\omega n u = \frac{\partial u}{\partial x} - \frac{\partial u}{\partial z} - \frac{3}{2} i\omega n u = 0 \tag{68}$$

$$\rightsquigarrow \quad \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} - \frac{u_{i,j+1} - u_{i,j}}{h_z} - \frac{3}{2} i\omega n_{ij} u_{ij} = 0 \tag{69}$$

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