Pragmatic Modeling of a Full Waveform Inversion using Finite Difference Method to Solve Wave Equation

10003 and 10010

Department of Geoscience and Petroleum, Norwegian University of Science and Technology

November 11, 2018

This report was prepared as a requirement for TPG4155 Course of Fall 2018 at Norwegian University of Science and Technology.

Abstract

This study presents a construction of Full Wave Inversion in its pragmatic form by applying with a finite difference method to solve the acoustic wave equation. Full Wave Inversion is a technique in seismic modeling with the minimization of misfit between modeled and true recorded data. The finite difference is used as a numerical approach to model the wavefield.

The numbers of grids within the model are 101 by 51 spatial grid cells, and 800 timesteps. Every space region in the model is discretized in every 10 m, while the time region is discretized in every 1 ms. MATLAB software is used to compute the problem.

The main objective of FWI is to minimize error ϕ that states how accurate the synthetic model is to the true model. A step length value α that can best decrease ϕ is chosen for every iteration. There are two methods based on numerical optimization that is used to determine α in this research: a form of "blind" Steepest Descent method, and Golden Search method.

The resulting model using the two different optimizations is compared. Even though the blind Steepest Descent seems too simplistic compared to Golden Search, its error value is not significantly larger than using Golden Search, with much less program runtime. Therefore, simply using a blind Steepest Descent method would provide a satisfactory model that is not time-consuming.

Keywords: FWI, acoustic wave equation, finite difference, numerical optimization

1 Theory

1.1 Full Waveform Inversion

Full Wave Inversion (FWI) was firstly introduced by Lailly (1983) and Tarantola (1984). This technique is an optimization of a geological structure model construction by minimizing the difference between the recorded and synthetic data. The difference formed a gradient, by correlating the wave field emitted from the source and the promulgated residual wave field in reversed time using an adjoint calculation. At the end of every iteration, a gradient is combined with the initial model and, thus, updates the synthetic model, which is utilized for the next iteration step (Virieux and Operto, 2009). The optimal synthetic model is achieved when the misfit reached its minimum value. FWI is considered an effective tool to obtain a higher resolution seismic imaging compared to the conventional method, thus, this technique is able to solve the low-frequency part of the model. Figures 1 and 2 pictures how in FWI synthetic model evolves over time to approximate the true recorded model.

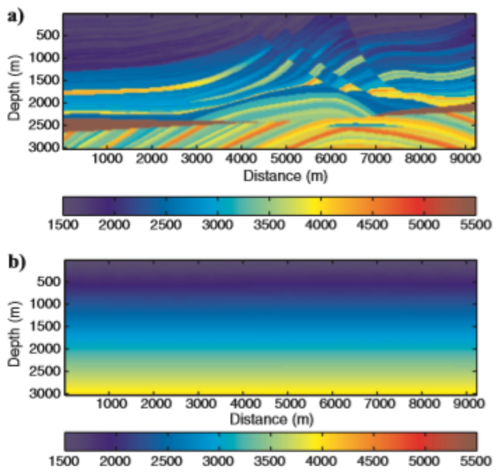


Figure 1: **a)** True recorded model from the field. **b)** Initial background velocity model for FWI. (Source: Abubakar et al. (2009))

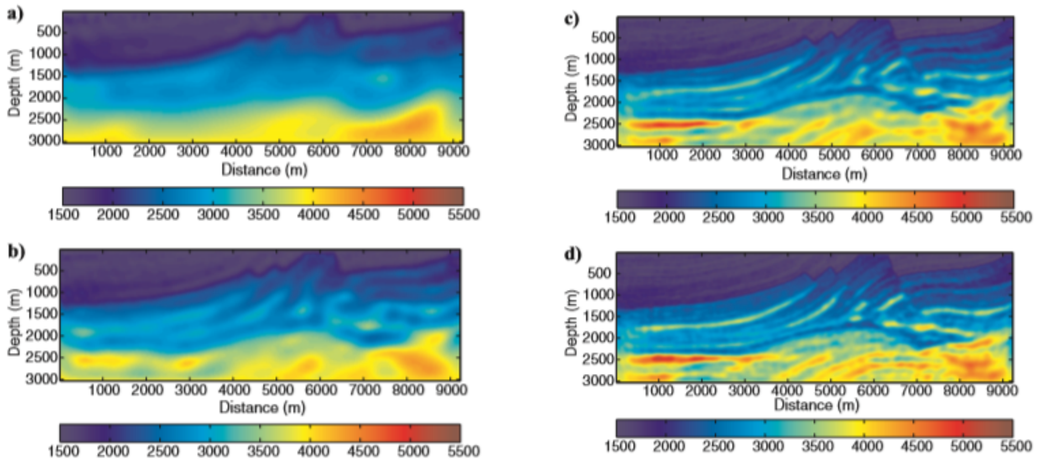


Figure 2: Updating the synthetic model continuously with different frequencies for inversion: **a)** f = 3 Hz, **ab** f = 7.5 Hz, **c)** f = 12 Hz, **d)** f = 16.5 Hz. As higher frequencies included in inversion, the error of the test model decreases and it slowly resembles the true model. (Source: Abubakar et al. (2009))

The adjoint calculation uses a forward modeling from a presumed characteristic to form a predicted model from source to the spot where observed field, which the predicted model is compared to, is located.

The actual model properties are used as an approximated solution for the predicted model, which is to be determined by an inversion method, and accordingly, the following forward modeling is updated. After the inversion, a certain misfit data between the models is yielded and analyzed thereafter. With an adequately small misfit, the forward model is identical to the actual condition. Otherwise, the process of forward-modeling is reiterated and re-adjusts the predicted model until the misfit is decreased and sufficiently low (Hursky et al., 2004).

FWI can be solved in time or frequency domain. Referring to Marfurt (1984), the time domain system applies

$$M(x)\frac{d^2u(x,t)}{dt^2} = A(x)u(x,t) + s(x,t) \tag{1}$$

where M is the mass, A is the stiffness matrix, s is source term and u is seismic wave field. Those parameters are functions of time, t, and position, x. On the other hand, the frequency domain system can be defined as

$$B(x,\omega)y(x,\omega) = s(x,\omega) \tag{2}$$

where B is the impedance matrix and ω is angular frequency.

There are two approaches to applying this inversion modeling: 2D and 3D seismic data. The differences between them lay on the numerical computations, amplitudes of the model decay, and the phase difference on wave propagation (Raknes and Arntsen, 2017). In the 2D system, it offers less complicated calculation and computer resources in solving the wave field equation than the 3D system. However, the 3D setting generates more realistic model since the observed data from the real field is constructed in 3-dimensional direction, while in 2D setting, the modification to real data is required to simplify the geological structures into an acoustic medium.

Many studies have expanded the usage of this technique to characterize the subsurface. For example is the application of Elastic Full Waveform Inversion (E-FWI) on shallow Loranca Basin (Kormann et al., 2016) with the reduction on offset along the iterations as the frequency get higher. Various improvements are also implemented to acoustic wave medium. One of the developments is made by Zhang and Joardar (2018) which is FWI for slowness using the cross hole data generated by an acoustic wave, resulting in an increased robustness and better inversion.

1.2 Wave Equation

The acoustic wave is a longitudinal wave of pressure system or particle movements spreading in a fluid medium, such as liquid or gas (Weik, 1989) which neglects the shear forces in the system (Fitchner, 2011). Introducing the elastic wave equation as the earth model is

$$\rho(x)\ \ddot{u}(x,t) - \nabla.\sigma(x,t) = f(x,t) \tag{3}$$

with u denotes displacement field, ρ mass density, σ stress tensor and f denotes external forces. If the stress tensor is stated in term of displacement gradient, it is written as

$$\sigma(x,t) = C(x) : \nabla u(x,t) \tag{4}$$

Commonly, earth can be described as an isotropic bulk with elastic tensor

$$C_{ijk} = k\ \delta_{ij}\delta_{kl} + \mu\ \delta_{ik}\delta_{jl} + \mu\delta_{il}\delta_{jk} \tag{5}$$

where μ is shear modulus, and k is bulk modulus. Since the $\mu = 0$, then Equation 4 rearranged to

$$\sigma_{ij} = k\delta_{ij}\nabla.u = \ -p\ \delta_{ij}. \tag{6}$$

Inserting Equation 6 to Equation 3 and adding the relation of the displacement field with scalar pressure, $p = -k\ \nabla\ .\ u$, it yields

$$\rho\ddot{u}\ +\ \nabla p\ =\ f. \tag{7}$$

Equation 7 divided by density, taking the divergence and eliminating the term u by including the scalar pressure definition will form the following:

$$k^{-1}\ \ddot{p}\ -\ \nabla\ .\ (\rho^{-1}\ \nabla\ p) = -\nabla\ .\ (\rho^{-1}\ f). \tag{8}$$

Density differential is much lower than pressure p and external forces f. so that *acoustic wave equation* can be simplified to

$$\ddot{p} \, - c^2 \, \Delta p = \, -c^2 \, \nabla \, . \, f. \tag{9}$$

with speed $c = \sqrt{\frac{k}{\rho}}$, f as the source term, and p single scalar pressure p. This wave equation usually used for 2D FWI with less complexity compared to elastic wave equation in Equation 3.

If no external forces (sources) are involved, the right hand term in Equation 9 will be zero. Returning the scalar pressure p into its initial form $-k \, \nabla \, . \, u$ results in

$$-\, k\ddot{u} \;=\; c^2 \Delta(-k\nabla \, . \, u) \tag{10}$$

Taking out bulk modulus k, the equation is simplified into a basic acoustic wave equation generally used and solved in partial differential equation problems:

$$\ddot{u} \;=\; c^2 \, \nabla^2 u \tag{11}$$

With $\ddot{u}$ indicates second derivative of displacement in time and $\nabla^2 u$ second derivative of displacement in every space dimension concerned.

1.3 Finite Difference Method for Wave Equation

1.3.1 Definition

Finite difference is based on the approximation of an exact derivative $\partial_x f(x_i)$ of a function f at a position x_i (in 1D case) evaluated for a number of neighbouring grid points (Matthews and Fink, 2004). Construction of finite-difference approximations generally uses truncated Taylor expansions (Fitchner, 2011) that results in Equation 12 for the second derivative of f evaluated at point x_i and two neighboring points (known as three-points stencil).

$$\partial_{xx} f(x) = \frac{1}{(\Delta x)^2}(f(x_i + \Delta x) - 2f(x_i) + f(x_i - \Delta x)) + O((\Delta x)^2) \tag{12}$$

The truncation error is stated in the last term of having order of $(\Delta x)^2$.

1.3.2 Discretization of wave equation

The central-difference formulas for approximating $u_{tt}(x,t)$ and $u_{xx}(x,t)$ for three-points stencil are:

$$u_{tt}(x,t) = \frac{1}{g^2}(f(x,t+g) - 2f(x,t) + f(x,t-g)) + O(g^2) \tag{13}$$

$$u_{xx}(x,t) = \frac{1}{h^2}(f(x,t+h) - 2f(x,t) + f(x,t-h)) + O(h^2) \tag{14}$$

With $g = \Delta t$ and $h = \Delta x$. Next, state for space $x_{i+1} = x_i + h$ and $x_{i-1} = x_i - h$, while for time $t_{j+1} = t_j + g$ and $t_{j-1} = t_j - g$. Discarding the truncation error term in each equation and referring to Equation 11, the acoustic wave equation in its discretized form is:

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{g^2} = c^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \tag{15}$$

1.3.3 Stability Analysis

In order to ensure any error made at one step is eventually dampened out and does not "explode", i.e. the solution is stable, some criteria have to be achieved. R. Courant, K. Friedrichs, and H. Lewy in their research have established a necessary condition for convergence while retaining the partial differential equation solution. The stability criterion is named CFL condition (Courant et al., 1928). It requires that

$$r = \frac{c\Delta t}{\Delta x} = \frac{cg}{h} \leq 1 \tag{16}$$

for 1D case. The r parameter is called *Courant's Number.* In 2D problems, the Courant's number is

$$r = \frac{c\Delta t}{\Delta x} + \frac{c\Delta t}{\Delta y} \leq 1 \tag{17}$$

Figure 3 below is a depiction of numerical stability and lack thereof. In the image, while keeping c and Δx constant, changing Δt above 0.4 will cause $r > 1$ and numerical stability criterion is no longer fulfilled.

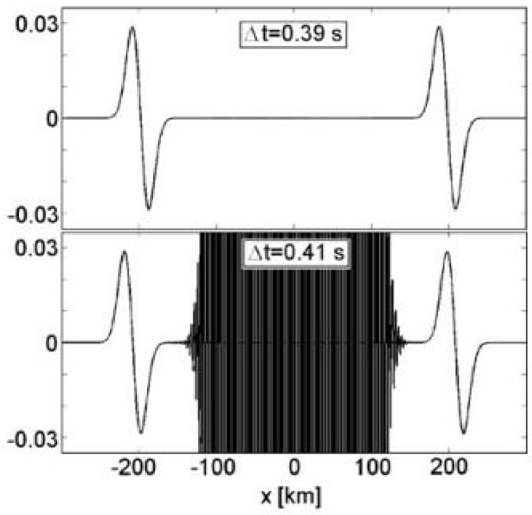


Figure 3: Numerical stability and instability. The CFL condition requires $\Delta t \leq 0.4$. Top image shows a stable solution, while bottom image no longer satisfies CFL condition, rendering the solution unstable. (Source: Fitchner (2011))

2 Implementation and Methods

2.1 Objective

This study will present the best method of modeling result of FWI using finite difference method to numerically solve the wave equation, based on procedure discussed in Section 1.1. To consider if a synthetic model is indeed the "best", one should define a criterion (or criteria) of the objective. True recordings of FWI are available, and comparations will be done with the model from this project. Taking the difference between the synthetic test model $u(m_k; x_r, t)$ and the true model $d_o(t)$, a "residual" value can be obtained at each timestep

$$\chi(t) = u(m_k; x_r, t) - d_o(t) \tag{18}$$

The second norm of the residual can be taken as the objective-function or the "error" that measures fit of the test model to the true model

$$\phi = \parallel \chi \parallel \tag{19}$$

This "error" value is to be minimized as best as possible in order to fulfill the objective of this project.

2.2 The Synthetic Model

2.2.1 Discretization and stability criterion

The model is two-dimensional in space and one-dimensional in time: 1010 m in length (y-dimension), 510 m in depth (x-dimension) and 800 ms in time. Discretization is every 10 m for each space dimension, and 1 ms (10^{-3} s) for time. In total, there are 101 by 51 spatial grid cells and 800 timesteps.

Recalling CFL condition for stability analysis, the Courant's Number for this simulation is

$$r = \frac{c\Delta t}{\Delta x} + \frac{c\Delta t}{\Delta y} = \frac{10^{-3}c}{10} + \frac{10^{-3}c}{10} = 2x10^{-4}\ c \tag{20}$$

For this discretization, background model must not have $c > 5000$ m/s in order to get a stable solution.

2.2.2 Initial conditions

The initial model consists of two distinct layers of rock, each having homogeneous properties so they have the same velocity value at any point.

- $v = 2000$ m/s from the 1st to 15th grid in x-dimension
- $v = 2300$ m/s from the 16th to 50th grid in x-dimension

Inserting all initial velocity to Equation 20, no values would make $r > 1$. Thus it can be concluded that **this model would give a stable solution.**

2.2.3 Boundary conditions

If no boundary conditions are defined, running the simulation model will result in waves reflecting off the model boundary. This is not representative because, in the subsurface, the wave should continue to propagate and not fully reflected somewhere. To avoid this, the model is extended beyond the spatial dimension it needs, creating a "buffer" area big enough such that no unwanted reflections appear in the model.

The extended model has a spatial dimension of 3010 m (y-dimension) by 2010 m (x-dimension), so total grid cells are 301 x 201.

2.3 Numerical Solution to the Wave Equation

The central difference is used for both time and space since it gives the least error compared to forward or backward difference (Matthews and Fink, 2004). For space, a seven-points stencil for each x-dimension and y-dimension is used for better accuracy and less numerical diffraction, while for time only three points are used because there is no extension of time backward and forwards in the model like there is for space. The stencil is pictured in Figure 4.

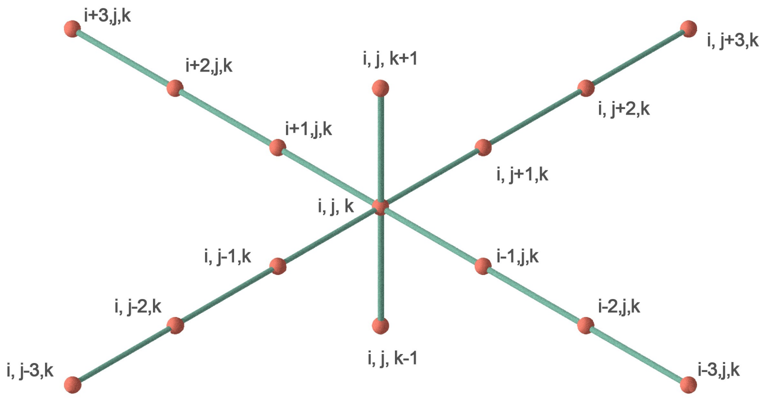


Figure 4: Stencil for solving the wave equation.

The wave equation now has a form in which it can be solved numerically:

$$\begin{aligned}u_{i,j}^{k+1} = &\frac{r_x}{180}(2(u_{i+3,j}^k + u_{i-3,j}^k) - 27(u_{i+2,j}^k + u_{i-2,j}^k) + 270(u_{i+1,j}^k + u_{i-1,j}^k) - 490u_{i,j}^k) \\ &+ \frac{r_y}{180}(2(u_{i,j+3}^k + u_{i,j+3}^k) - 27(u_{i,j+2}^k + u_{i,j+2}^k) + 270(u_{i,j+1}^k + u_{i,j+1}^k) - 490u_{i,j}^k) \\ &+ 2u_{i,j}^k - u_{i,j}^{k-1}\end{aligned} \quad (21)$$

Where i,j, and k respectively refer to x-dimension, y-dimension, and time dimension. $r_x = \frac{c^2(\Delta x)^2}{(\Delta t)^2}$ and $r_y = \frac{c^2(\Delta y)^2}{(\Delta t)^2}$. Full derivation to obtain Equation 21 is available in Appendix A.

2.4 Updating the Model

2.4.1 Sources and receivers

Sources and receivers are located at the top grid cell. Sources are set off (given signature) using the time-derivative of a Ricker wavelet at every timestep, and recorded at every receiver, making a total of 101 x 101 traces for each timestep.

True model available in this study is divided into several recordings each having their own dominant frequency used in the Ricker wavelet. Frequencies used are 4,6,8,10, and 12 Hz. Therefore, in order to recreate a true model the same frequency range is applied to the synthetic test model.

2.4.2 Solving the wave equation

The underlying principle of FWI is recreating the true model using the synthetic test model. To be able to make an accurate recreation, seismic wave propagation needs to be simulated in the synthetic model. Equation 21 will be used to solve the wave equation in this project to approach the problem numerically.

Tracing waves for the next timestep $k+1$ according to Equation 21 would need wavefield at current timestep k and at previous timestep $k-1$. Apart from that, the source signature of each timestep should be injected into each source as the driving force of the wave propagation.

Wave propagation is ideally traced from each source and detected in each receiver. To get less computing time, stacking results only from a few select sources might be done, and fortunately summing up from every 10th source is representative enough for the real result and that will be implemented here.

2.4.3 Calculating gradient

An "adjoint" field can be described as the difference in the wavefield of synthetic and true traces recording. The gradient of the synthetic model m is proportional to the time derivative of an adjoint field and the forward field (recording of wavefield at a given time) for every timestep

$$\delta m(x) \propto \int_0^\tau \partial_t u^\dagger(x,t)\partial_t u(x,t)dt \quad (22)$$

To create the adjoint field, the residual from Equation 18 is calculated for each time step and then time-reversed. The wave equation is solved again, this time using each receiver as a source and the time-reversed residual as the source signature.

Aligning the adjoint field and forward field in time, and stacking recordings of different sources, the gradient of the model can be calculated

$$\delta m(x) = \sum_{s=0}^{N_s}\sum_{t=0}^{\tau} \partial_t u^\dagger(x,t)\partial_t u(x,t) \quad (23)$$

One problem that could be encountered by calculating gradient this way is unwanted numerical noise close to the receivers, where the synthetic model should be more accurate. To mitigate this, tapering the calculated gradient using sine taper is shown to be effective.

The resulting gradient after tapering is the gradient needed for updating the synthetic model.

2.4.4 Calculating step length

Step length in this implementation is defined as the "scale factor" α that is applied to gradient in order to update the model m

$$m_{k+1} = m_k + \alpha \delta m_k \tag{24}$$

The value of α must be chosen such that the updated model m_{k+1} has lowest error ϕ possible compared to the original model m_k. This can be solved using various numerical optimization methods, and two of them will be examined and compared to see which one has better application.

1. Since FWI computation is quite demanding, to minimize runtime of the program a straightforward method is preferred. The first method to be tested is a **"Blind" Steepest Descent** method, where essentially it just tries a number, see if the error decreases compared to previous error, and if not the number is decreased.

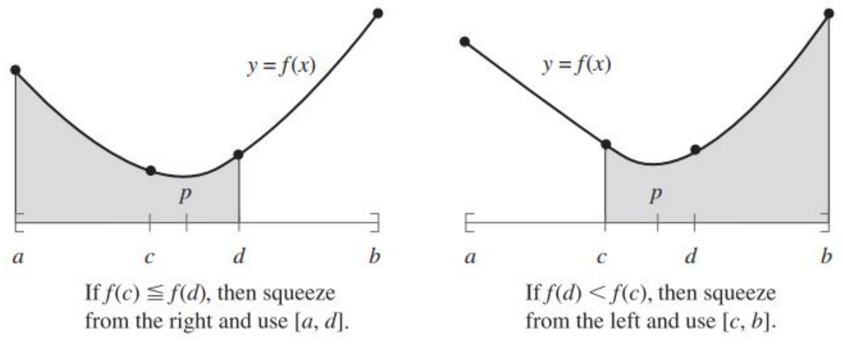
   Starting with a large number, e.g. $\alpha = 256$, if the error is not decreased then halve the number, continuing this process until a α that reduces error is obtained.

2. One of the most simple yet effective methods is the **Golden Search** method, where golden ratio is used in searching a value that minimizes a function (Matthews and Fink, 2004).

   In seeking the function minima inside interval $[a, b]$, Golden Search method creates two interior points c and d such that $a < c < d < b$. If $f(c) \leq f(d)$, the minima must be in $[a, d]$, so b is replaced with d in the next interval. If $f(d) < f(c)$, the minima must be in $[c, b]$, so a is replaced with c. The interior points is symmetrical; that is $b - d = c - a$, where

$$c = a + (1 - r)(b - a) \tag{25}$$
$$d = b - (1 - r)(b - a) \tag{26}$$

   The value of r is desired to be constant for every interval. This can be achieved when $r = \frac{-1+\sqrt{5}}{2}$, otherwise known as the golden ratio. Chosen initial interval $[0, 256]$ is sufficiently large for determining the optimum value of α. After a step length value is found, the model can be updated properly using Equation 24. The decision process of the Golden Search method is illustrated in Figure 5.



y = f(x)
p
a
c
d
b
If f(c) ≦ f(d), then squeeze from the right and use [a, d].
y = f(x)
p
a
c
d
b
If f(d) < f(c), then squeeze from the left and use [c, b].

Figure 5: Decision process of Golden Search method in numerical optimization. (Source: Matthews and Fink (2004))

   Alas, Golden Search is also known to be time-consuming. For computers with less than decent performance, calculating a complex problem such as FWI (even at its pragmatic form) could be highly time-inefficient.

Starting from the lowest frequency for use in source signature, ten iterations will be implemented to find gradient and step length that minimizes the objective function ϕ. After the maximum number of iterations is reached, the whole process is repeated using the next higher frequency, i.e. 50 total iterations will be done considering there are five frequency values.

3 Results and Discussions

3.1 Final State of the Synthetic Model

3.1.1 Comparison between methods used

As discussed in Section 2.4.4, two methods of calculating step length α will be discussed and compared. After implementing both methods to continuously update the synthetic test model until the number of maximum iterations is attained, the model turns into Figure 6 displayed below.

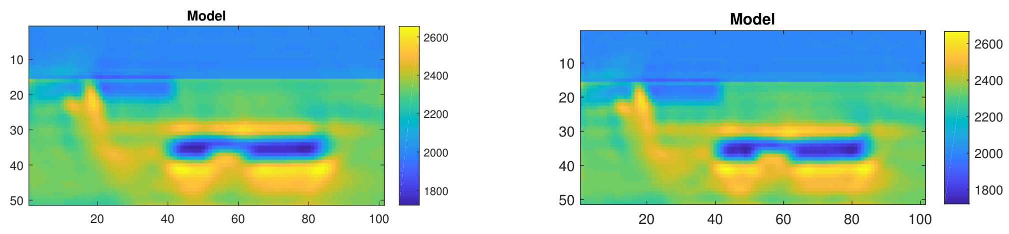


Figure 6: Final updated synthetic test model. **Left:** Blind Steepest Descent is used to determine α. **Right:** Golden Search is used to determine α. Colorbar represents background velocity in every grid cell in m/s.

Both resulting models show that the top 15 grids make up a formation with different properties than grids below it, as already modeled in the initial state. A clear velocity distinction near the middle region can be observed, which corresponds to rocks or structures with varying properties.

3.1.2 Error analysis

To determine if this model answers the objective, evolution of the error value needs to be observed. Figure 7 shows how the error of this model changes for every iteration, with the step length α needed to update the synthetic model.

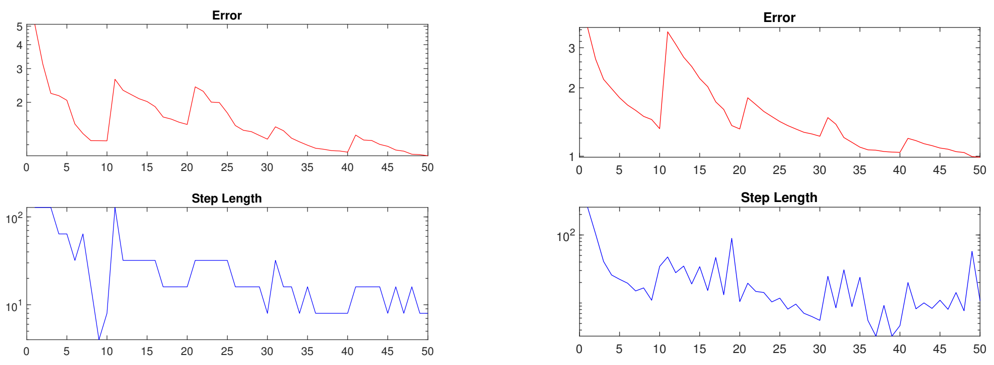


Figure 7: Error and step length α for each iteration. **Left:** Blind Steepest Descent is used to determine α. **Right:** Golden Search is used to determine α. After every 10th iteration, a new frequency is used for source signature.

The objective function or error ϕ is seen to decrease in each iteration, with the exceptions for every ten iterations due to changing inversion frequency. The final error of the test model for each method used is:

- **Blind Steepest Descent:** $\phi = 1.0506$
- **Golden Search:** $\phi = 0.9900$

Step length necessary for model update is noticed to fluctuate from iteration to iteration, ranging from around 4 to 128 using Blind Steepest Descent, and 3 to 256 for Golden Search; It can be professed that $[0, 256]$ is indeed an adequate initial interval that contains step length values to properly minimize error for both methods used.

3.2 Recommendations

Pragmatic modeling of FWI demonstrated in this study is proven to successfully minimize the difference between the test model and true model using both numerical optimization methods. However, when comparing the program runtime a huge distinction between them become noticeable:

- **Blind Steepest Descent:** 5145 seconds runtime ($\sim$ 1 hours and 26 minutes)
- **Golden Search:** 23256 seconds runtime ($\sim$ 6 hours and 28 minutes)

For a minuscule difference in error ϕ compared to Blind Steepest Descent, the Golden Search method consume much more runtime. Unless one has a powerful computing device(s) at their disposal, it is not recommended to use Golden Search method. Preferably, a much more accurate and time-efficient numerical optimization could be utilized to determine α and also at the same time reducing ϕ further. There are a lot of such methods that can be found in Matthews and Fink (2004). Nevertheless, take into consideration that they require more advanced comprehension before they could be actualized in this program.

Additionally, to accomplish an even accurate synthetic model, all traces from every source could be taken into account when calculating gradient instead of only tracing a select few. Again though, be warned that this will involve a massive amount of computation and in most cases is not recommended.

4 Conclusions

1. Modeling FWI using finite difference method to solve the wave equation gives satisfactory results that minimize error ϕ between synthetic model and the true model.
2. Two different methods are used for calculating step length α: Blind Steepest Descent and Golden Search. The error for each respective method is **1.0506** and **0.9900**.
3. In spite of a lower ϕ value, Golden Search method consumes much more computational power and time. Moreover, the difference in error between the two methods is not significant. Therefore, Golden Search is not recommended for use in common practice.

References

Abubakar, A., Hu, W., M, T., Habashy, and van den Berg, P. M. (2009). Application of finite-difference contrast-source inversion algorithm to seismic full-waveform data. *Geophysics*, 74(6):WCC47–WCC58.

Courant, R., Friedrichs, K., and Lewy, H. (1928). Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74.

Fitchner, A. (2011). *Full Seismic Waveform Modelling and Inversion*. Springer-Verlag Berlin Heidelberg, Berlin.

Hursky, P., Porter, M. B., Comuelle, B. D., Hodgkiss, W. S., and Kuperman, W. A. (2004). adjoint modeling for acoustic inversion. *The Journal of the Acoustical Society od America*, 115:607–619.

Kormann, J., Rodriguez, J. E., Ferrer, M., Gutierrez, N., de la Puente, J., Hanzich, M., and Cela, J. M. (2016). Elastic fwi of reflection data with a phase misfit function. *High Performance Computer Applications*, pages 277–284.

Lailly, P. (1983). The seiscmic inverse problem as sequence of before stack imigiration. *Conference on Inverse Scattering, Theory and Application*, pages 206–220.

Marfurt, K. (1984). Accurancy of finite-difference and finite-elements modeling of the scalar and elastic wave equation. *Geophysics*, 49:533–549.

Matthews, J. H. and Fink, K. D. (2004). *Numerical Methods using MATLAB.* Pearson Prentice Hall, New Jersey, 4th edition.

Raknes, E. B. and Arntsen, B. (2017). Challenges and solution for performing 3d time-domain elastic ful-waveform inversion. *The Leading Edge*, 36:88–93.

Tarantola, A. (1984). Inversion of seismic reflection data in acoustic approximation. *Geophysics*, 49:1259–1266.

Taylor, C. (2016). Finite difference coefficients calculator. http://web.media.mit.edu/∼*crtaylor/calculator.html*.

Virieux, J. and Operto, S. (2009). An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC127–WCC152.

Weik, M. (1989). Acoustic wave equation. https://www.its.bldrdoc.gov/fs-1037/dir-001/0133.*htm*.

Zhang, W. and Joardar, A. K. (2018). Acoustic based crosshole full waveform slowness inversion in the time domain. *Journal of Applied Mathematics and Physics*, 6:1086–110.

A Derivation of Numerical Solution to the Wave Equation (Equation 21)

A generalized equation exists (Taylor, 2016) to obtain the finite difference equation from any finite difference stencil given the desired derivative order. Given a stencil s of length N and derivative order $d < N$, the coefficients c are given by the following equation:

$$s_1^n c_1 + s_2^n c_2 + \cdots + s_N^n c_N = \frac{d!}{h^d}\delta(n-d) \quad ; \quad 0 \le n \le N-1 \tag{27}$$

Since here $N = 7$ (stencil is from $x-3$ to $x+3$) and derivative order $d = 2$:

$$s_1^n c_1 + s_2^n c_2 + \cdots + s_7^n c_7 = \frac{2!}{h^2}\delta(n-2) \quad ; \quad 0 \le n \le 6 \tag{28}$$

Equation 28 can be solved in matrix form:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \frac{1}{h^2} \begin{bmatrix} s_1^0 & \cdots & s_7^0 \\ \vdots & \ddots & \vdots \\ s_1^6 & \cdots & s_7^6 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2! \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{29}$$

The resulting coefficient matrix will be:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1/90 \\ -3/20 \\ 3/2 \\ -49/18 \\ 3/2 \\ -3/20 \\ 1/90 \end{bmatrix} \tag{30}$$

Therefore, the second derivative for $f(x)$ for a seven-point stencil using central difference approach is:

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{2f(x-3) - 27f(x-2) + 270f(x-1) - 490f(x) + 270f(x+1) - 27f(x+2) + 2f(x+3)}{180h^2} \tag{31}$$

The discretized wave equation in Equation 15 is still in three-point stencil form for both time and 1D space. Converting the equation to 2D space and lengthen the differentiators to seven-point stencil (for only space dimension):

$$\begin{aligned} \frac{u_{i,j}^{k+1} - 2u_{i,j}^k + u_{i,j}^{k-1}}{(\Delta t)^2} = c^2 \Big(& \frac{2u_{i+3,j}^k - 27u_{i+2,j}^k + 270u_{i+1,j}^k - 490u_{i,j}^k + 270u_{i-1,j}^k - 27u_{i-2,j}^k + 2u_{i-3,j}^k}{180(\Delta x)^2} \\ & + \frac{2u_{i,j+3}^k - 27u_{i,j+2}^k + 270u_{i,j+1}^k - 490u_{i,j}^k + 270u_{i,j-1}^k - 27u_{i,j-2}^k + 2u_{i,j-3}^k}{180(\Delta y)^2} \Big) \end{aligned} \tag{32}$$

With i denotes space in the x-dimension, j denotes space in the y-dimension, and k denotes time dimension. Bringing $(\Delta t)^2$ to right term and defining $r_x = \frac{c^2(\Delta x)^2}{(\Delta t)^2}$ and $r_y = \frac{c^2(\Delta y)^2}{(\Delta t)^2}$:

$$\begin{aligned} (u_{i,j}^{k+1} - 2u_{i,j}^k + u_{i,j}^{k-1}) = {} & \frac{r_x^2}{180}\left(2u_{i+3,j}^k - 27u_{i+2,j}^k + 270u_{i+1,j}^k - 490u_{i,j}^k + 270u_{i-1,j}^k - 27u_{i-2,j}^k + 2u_{i-3,j}^k\right) \\ & + \frac{r_y^2}{180}\left(2u_{i,j+3}^k - 27u_{i,j+2}^k + 270u_{i,j+1}^k - 490u_{i,j}^k + 270u_{i,j-1}^k - 27u_{i,j-2}^k + 2u_{i,j-3}^k\right) \end{aligned} \tag{33}$$

Grouping terms with the same coefficient:

$$\begin{aligned}(u_{i,j}^{k+1} - 2u_{i,j}^{k} + u_{i,j}^{k-1}) \;=\; & \frac{r_x}{180}\left(2(u_{i+3,j}^{k} + u_{i-3,j}^{k}) - 27(u_{i+2,j}^{k} + u_{i-2,j}^{k}) + 270(u_{i+1,j}^{k} + u_{i-1,j}^{k}) - 490u_{i,j}^{k}\right) \\ & + \frac{r_y}{180}\left(2(u_{i,j+3}^{k} + u_{i,j+3}^{k}) - 27(u_{i,j+2}^{k} + u_{i,j+2}^{k}) + 270(u_{i,j+1}^{k} + u_{i,j+1}^{k}) - 490u_{i,j}^{k}\right) \quad (34)\end{aligned}$$

To calculate wavefield at the next timestep, terms $-2u_{i,j}^{k} + u_{i,j}^{k-1}$ is moved to right side. This would result Equation 34 to be the same as Equation 21.

B MATLAB Code

Code scripts also available at https://github.com/fdlberylian/TPG4155_Project2018

B.1 Pragmatic Full Wave Inversion (Main Code)

```
clear;
%% Setting up dimensions
dims.dy =     10; % [m]
dims.dx =     10; % [m]
dims.dt = 1.0e-3; % [s]

dims.ny = 201; % Cells in y-direction
dims.nx = 301; % Cells in x-direction
dims.nt = 801; % Amount of time steps

%% Model dimensions
dims.modely = 100:150;
dims.modelx = 100:200;
dims.my = length(dims.modely);
dims.mx = length(dims.modelx);

%% Source locations
sx = min(dims.modelx):max(dims.modelx);
sy = min(dims.modely)*ones(1,length(sx));
dims.srcPos = sy + dims.ny*sx;

%% Receiver locations
rx = min(dims.modelx):max(dims.modelx);
ry = min(dims.modely)*ones(1,length(rx));
dims.recPos = ry+dims.ny*rx;

%% Creating background model
bg = zeros(dims.ny,dims.nx,'single');
bg(:) = 2.0e3;          % [m/s] - Background
bg(115:end,:) = 2.3e3; % [m/s] - Layer

%% Begin iteration
model = bg;     % Starting model
dims.ds = 10;   % Grid point distance between sources
maxIter = 10;   % Maximum number of iterations per frequency
freqs = [4,6,8,10,12];  % Frequencies to use in inversion

it = 1; tic;
for f = freqs
    fprintf('<<Frequency: %2.0f Hz>> \n',f);
    %% Generating ricker source signature wavelet
    source = rickerWave(f,dims);
    %% Load true recording
    load (['trueRec_',num2str(f),'Hz.mat']);
    for i = 1:maxIter
        %% Calculate gradient
        fprintf('Iteration Number: %2.0f \n',i);
        [gradient,err] = calculateGradient(dims,source,model,trueRec);

        %% Taper and plot gradient
        gradient = taperGradient(gradient);
        figure(1);
        imagesc(gradient(dims.modely,dims.modelx));
        title('Gradient');
        axis('image');
        colorbar();
        drawnow();

        %% Calculate step length
        [stepLength,err] = calculateStepLength(dims,err,gradient,source,
           model,trueRec);

        %% Update model
        model = model + stepLength*gradient;
        figure(2);
        imagesc(model(dims.modely,dims.modelx));
        title('Model');
        axis('image');
        colorbar();
        drawnow();

        errVec(it) = err;
        alpha(it) = stepLength;
        figure(3);
        subplot(2,1,1);
        semilogy(errVec,'r-');
        title('Error');
        xlim([0,length(freqs)*maxIter]);
        subplot(2,1,2);
        semilogy(alpha,'b-');
        title('Step Length');
        xlim([0,length(freqs)*maxIter]);
        drawnow();

        it = it + 1;
        if stepLength < 1
            toc
            fprintf('\n');
            break
        end
        toc
        fprintf('\n');
    end
end
fprintf('Run completed.');
```

B.2 Calculate Gradient

```
function [gradient, err] = calculateGradient(dims,source,model,trueRec)
%% Setup
    recording = zeros(dims.nt,length(dims.recPos),'single');
    gradient  = zeros(dims.ny,dims.nx,'single');
    forwardField = zeros(dims.my,dims.mx,dims.nt,'single');
    adjointField = zeros(dims.my,dims.mx,dims.nt,'single');
    err = 0;
    for s = 1:dims.ds:length(dims.srcPos)
        %% Run forward simulation on background model
        uold = zeros(dims.ny,dims.nx,'single');
        u = zeros(dims.ny,dims.nx,'single');
        unew = zeros(dims.ny,dims.nx,'single');
        for t = 1:dims.nt
            % Solve wave equation
            srcPos = dims.srcPos(s);
            unew = solveWaveEqn(dims,source,model,srcPos,t,uold,u,unew);
            % Record traces
            recording(t,:) = unew(dims.recPos);
            % Save forward field for use in correlation
            forwardField(:,:,t) = unew(dims.modely,dims.modelx);
            uold = u;
            u = unew;
        end
        %% Calculate difference and error
        chi = recording(:,:)-trueRec(:,:,s);
        err = err + norm(chi);
        %% Run adjoint simulation
        chi = flipud(chi);
        uold = zeros(dims.ny,dims.nx,'single');
        u = zeros(dims.ny,dims.nx,'single');
        unew = zeros(dims.ny,dims.nx,'single');
        for t = 1:dims.nt
            % Solve wave equation using the difference (chi) as sources
            recPos = dims.recPos;
            unew = solveWaveEqn(dims,chi,model,recPos,t,uold,u,unew);
            % Save adjoint field for use in correlation
            adjointField(:,:,dims.nt-t+1) = unew(dims.modely,dims.modelx
                );
            uold = u;
            u = unew;
        end
        %% Correlate
        for t = 2:dims.nt-1
            % Calculate the time derivative of the displacement to
            % gradient.
            dadj = zeros(dims.my,dims.mx,'single');
            dfor = zeros(dims.my,dims.mx,'single');
            for i=1:length(dims.modely)
                for j=1:length(dims.modelx)
                    dadj(i,j) = (adjointField(i,j,t+1)-adjointField(i,j,
                        t))/dims.dt;
                    dfor(i,j) = (forwardField(i,j,t)-forwardField(i,j,t
                        -1))/dims.dt;
                    ii = dims.modely(i);
```

```matlab
                    jj = dims.modelx(j);
                    gradient(ii,jj) = gradient(ii,jj) + dadj(i,j)*dfor(i,j);
                end
            end
        end
    end
end
```

B.3 Finite Difference Method to Solve Wave Equation

```matlab
function unew = solveWaveEqn(dims,source,c,s,t,uold,u,unew)
    %% Inject source and solve wave equation for one timestep
    u(s) = u(s) + source(t,:);

    rx = (dims.dt^2)/(dims.dx^2);
    ry = (dims.dt^2)/(dims.dy^2);

    for i=4:dims.ny-3
        for j=4:dims.nx-3
            unew(i,j) = (1/180)*ry*c(i,j)*c(i,j)*...
                    (2*(u(i-3,j)+u(i+3,j)) - ...
                    27*(u(i-2,j)+u(i+2,j)) + ...
                    270*(u(i-1,j)+u(i+1,j)) - ...
                    490*(u(i,j)))...
                    +...
                    (1/180)*rx*c(i,j)*c(i,j)*...
                    (2*(u(i,j-3)+u(i,j+3)) - ...
                    27*(u(i,j-2)+u(i,j+2)) + ...
                    270*(u(i,j-1)+u(i,j+1)) - ...
                    490*(u(i,j))) + ...
                    2*u(i,j)-uold(i,j);
        end
    end
end
```

B.4 Ricker Wavelet for Source Signature

```matlab
function [ricker] = rickerWave(freq,dims)
%RICKERWAVE Generates the time derivative of a Ricker wavelet
% to be used as a source signature
    t = 0:dims.dt:dims.dt*(dims.nt);
    tau  = (t-1/freq)*freq*pi;

    ricker = (1-tau.*tau).*exp(-tau.*tau);
    ricker = (ricker(2:end) - ricker(1:end-1))';
end
```

B.5 Taper Gradient using Sine Tapering

```matlab
function G = taperGradient(G)
    %% Taper gradient near sources to avoid interference
    [~,n] = size(G);
    taper = sin(linspace(0,pi/2,16));
    taper = repmat(taper,n,1)';
    G(100:115,:) = taper.*G(100:115,:);

    %% Normalise gradient
    scale = 1.0/max(abs(G(:)));
    G = scale*G;
end
```

B.6 Calculate Step Length

B.6.1 Method 1: "Blind" Steepest Descent

```matlab
function [stepLength,newErr] = calculateStepLength(dims,oldErr,gradient,
    source,model,trueRec)
%% Setup
     recording = zeros(dims.nt,length(dims.recPos),'single');
     stepLength = 256;
     newErr = inf;
     while (newErr > oldErr)
         newErr = 0;
         stepLength = stepLength/2;
         fprintf('  Current Step Length: %3.2f \n',stepLength);
         fprintf('    Original Error: %5.4f \n',oldErr);
         % Test model update
         modelt = model + stepLength*gradient;
         for s = 1:dims.ds:length(dims.srcPos)
             uold = zeros(dims.ny,dims.nx,'single');
             u = zeros(dims.ny,dims.nx,'single');
             unew = zeros(dims.ny,dims.nx,'single');
             for t = 1:dims.nt
                 %  Solve wave equation using test model update
                 srcPos = dims.srcPos(s);
                 unew = solveWaveEqn(dims,source,modelt,srcPos,t,uold,u,
                     unew);
                 %  Record traces
                 recording(t,:) = unew(dims.recPos);
                 uold = u;
                 u = unew;
             end
             %% Calculate new error and check against old
             chi = recording(:,:)-trueRec(:,:,s);
             newErr = newErr + norm(chi);
         end
         fprintf('    New Error: %5.4f \n',newErr);
     end
end
```

B.6.2 Method 2: Golden Search

```matlab
function [stepLength,newErr] = calculateStepLength_v2(dims,oldErr,
    gradient,source,model,trueRec)
%% Setup
     recordinga = zeros(dims.nt,length(dims.recPos),'single');
     recordingb = zeros(dims.nt,length(dims.recPos),'single');
     recordingc = zeros(dims.nt,length(dims.recPos),'single');
     recordingd = zeros(dims.nt,length(dims.recPos),'single');
     a = 0; b = 256; eps = 10^-2; del = 10^-2;
     r1 = (sqrt(5)-1)/2; r2 = r1^2; h = b-a;
     c = a+r2*h; d = a+r1*h;
     erra = 0; errb = 0; errc = 0; errd = 0;
     modelta = model + a*gradient;
     modeltb = model + b*gradient;
     modeltc = model + c*gradient;
    modeltd = model + d*gradient;
        for s = 1:dims.ds:length(dims.srcPos)
            uolda = zeros(dims.ny,dims.nx,'single');
            ua = zeros(dims.ny,dims.nx,'single');
            unewa = zeros(dims.ny,dims.nx,'single');
            uoldb = zeros(dims.ny,dims.nx,'single');
            ub = zeros(dims.ny,dims.nx,'single');
            unewb = zeros(dims.ny,dims.nx,'single');
            uoldc = zeros(dims.ny,dims.nx,'single');
            uc = zeros(dims.ny,dims.nx,'single');
            unewc = zeros(dims.ny,dims.nx,'single');
            uoldd = zeros(dims.ny,dims.nx,'single');
            ud = zeros(dims.ny,dims.nx,'single');
            unewd = zeros(dims.ny,dims.nx,'single');
            for t = 1:dims.nt
                %  Solve wave equation using test model update
                srcPos = dims.srcPos(s);
                unewa = solveWaveEqn(dims,source,modelta,srcPos,t,uolda,
                   ua,unewa);
                unewb = solveWaveEqn(dims,source,modeltb,srcPos,t,uoldb,
                   ub,unewb);
                unewc = solveWaveEqn(dims,source,modeltc,srcPos,t,uoldc,
                   uc,unewc);
                unewd = solveWaveEqn(dims,source,modeltd,srcPos,t,uoldd,
                   ud,unewd);
                %  Record traces
                recordinga(t,:) = unewa(dims.recPos);
                recordingb(t,:) = unewb(dims.recPos);
                recordingc(t,:) = unewc(dims.recPos);
                recordingd(t,:) = unewd(dims.recPos);
                uolda = ua; ua = unewa;
                uoldb = ub; ub = unewb;
                uoldc = uc; uc = unewc;
                uoldd = ud; ud = unewd;
            end
            %% Calculate new error and check against old
            chia = recordinga(:,:)-trueRec(:,:,s);
            chib = recordingb(:,:)-trueRec(:,:,s);
            chic = recordingc(:,:)-trueRec(:,:,s);
            chid = recordingd(:,:)-trueRec(:,:,s);
            erra = erra + norm(chia); errb = errb + norm(chib);
            errc = errc + norm(chic); errd = errd + norm(chid);
        end
       stepLength = b; newErr = errb;
    while (abs(erra-errb)>eps)||(h>del)
        stepLength = b; newErr = errb;
        fprintf('  Current Step Length: %3.2f \n',stepLength);
        fprintf('     Original Error: %5.4f \n',oldErr);

        if(errc < errd)
            b = d; errb = errd;
            d = c; errd = errc;
            errc = 0;
            h = b-a; c = a+r2*h;
            modeltc = model + c*gradient;
            for s = 1:dims.ds:length(dims.srcPos)
```

```
                uoldc = zeros(dims.ny,dims.nx,'single');
                uc = zeros(dims.ny,dims.nx,'single');
                unewc = zeros(dims.ny,dims.nx,'single');
                for t = 1:dims.nt
                    %  Solve wave equation using test model update
                    srcPos = dims.srcPos(s);
                    unewc = solveWaveEqn(dims,source,modeltc,srcPos,t,uoldc,uc,unewc);
                    %  Record traces
                    recordingc(t,:) = unewc(dims.recPos);
                    uoldc = uc; uc = unewc;
                end
                %% Calculate new error and check against old
                chic = recordingc(:,:)-trueRec(:,:,s);
                errc = errc + norm(chic);
            end
        else
            a = c; erra = errc;
            c = d; errc = errd;
            errd = 0;
            h = b-a; d = a+r1*h;
            modeltd = model + d*gradient;
            for s = 1:dims.ds:length(dims.srcPos)
                uoldd = zeros(dims.ny,dims.nx,'single');
                ud = zeros(dims.ny,dims.nx,'single');
                unewd = zeros(dims.ny,dims.nx,'single');
                for t = 1:dims.nt
                    %  Solve wave equation using test model update
                    srcPos = dims.srcPos(s);
                    unewd = solveWaveEqn(dims,source,modeltd,srcPos,t,uoldd,ud,unewd);
                    %  Record traces
                    recordingd(t,:) = unewd(dims.recPos);
                    uoldd = ud; ud = unewd;
                end
                %% Calculate new error and check against old
                chid = recordingd(:,:)-trueRec(:,:,s);
                errd = errd + norm(chid);
            end
        end
        fprintf('    New Error: %5.4f \n',newErr);
    end
end
```