IWAVE Implementation of Adjoint State Method

Dong Sun and William W Symes

ABSTRACT

Adjoint state method is a well-known method to efficiently compute the gradient of a cost or objective function for a simulation-driven optimization problem. Essentially, it computes the adjoint action of Born operator (the linearized forward map) on any given vector. This report presents a derivation of adjoint state algorithm for an acoustic system discretized by staggered grid finite difference schemes, and discusses its implementation based on the modeling package IWAVE.

INTRODUCTION

The inverse problem in reflection seismology is often formed as a simulation-driven optimization problem in Hilbert space. RVL is a collection of C++ classes to express core concepts (vectors, functions,...) of calculus in Hilbert space, and provides standardized interfaces for optimization and linear algebra algorithms. To use the interfaces provided by RVL to implement this inversion, the very first thing is to construct the modeling code as an RVL operator class (for vector valued functions) with at least methods to compute its value (modeling), first derivative and adjoint derivative (Born modeling and its adjoint). This report focuses on the adjoint Born modeling, and is the second of several describing an operator implementation for acoustics based on the modeling package IWAVE (Terentyev, 2009).

IWAVE is a time-stepping algorithm; the derivative and its adjoint for operators defined by time-stepping share the same abstract structure for all time-stepping algorithms. This structure is encapsulated in TSOpt, which gives an abstract RVL operator interface for all time-stepping algorithms (Enriquez and Symes, 2009). I will show how to embed IWAVE in a RVL operator class using TSOpt classes as helpers. This task includes implementing the derivative (Sun and Symes, 2010) and its adjoint using IWAVE with minimal modifications. We call the resulting system of classes IWAVE++. This report covers the implementation of the adjoint action of derivative operator.

In the following two sections, I show a general derivation of adjoint state algorithm for an acoustic system discretized by staggered grid finite difference schemes with second order in time. Then, I elaborate this algorithm with a one dimensional model problem in section 4, and discuss its implementation in section 5. Finally in section 6, I verify the implementation via some numerical experiments including dot-product tests.
THE ACOUSTIC MODEL

The acoustic model connects the state variable (wave-field) consisting of pressure \( p(x,t) \) and particle velocity \( v(x,t) \) with the control variable (model) consisting of buoyancy \( b(x) = \frac{1}{\rho(x)} \) (\( \rho(x) \) denotes density) and bulk modulus \( \kappa(x) \) through the wave equations

\[
\begin{align*}
\left( \frac{1}{\kappa(x)} \frac{\partial p}{\partial t} + \nabla \cdot v \right) (x,t) &= f(x,t), \\
\left( \frac{1}{b(x)} \frac{\partial v}{\partial t} + \nabla p \right) (x,t) &= 0, \\
p(x,t) &\equiv 0, \quad v(x,t) \equiv 0, \quad t < 0,
\end{align*}
\]

where \( x \in \mathcal{R}^l \) denotes a \( l \)-dimensional space location, \( f(x,t) \) is the source. Let \( m(x) := (\kappa(x), b(x)) \) denote the model (control variable). The above equation system defines the forward map

\[ F[m] := Sp, \]

where \( S \) is a sampling operator, such as \( Sp := \{ p(x_r,t) \} \) in which \( x_r \) denotes the coordinates of selected receivers.

The linearized forward map at model \( m \) is defined as

\[ DF[m] \delta m := S \delta p, \quad (1) \]

where \( \delta m \) denotes model perturbation, \( \delta p \) and \( \delta v \) are the corresponding first-order wave-field perturbation and solve the following equation system

\[
\begin{align*}
\frac{1}{\kappa(x)} \frac{\partial \delta p}{\partial t} + \nabla \cdot \delta v &= -\frac{\delta \kappa(x)}{\kappa} \nabla \cdot v, \\
\frac{1}{b(x)} \frac{\partial \delta v}{\partial t} + \nabla \delta p &= -\frac{\delta b(x)}{b(x)} \nabla p, \\
\delta p(x,t) &\equiv 0, \quad \delta v(x,t) \equiv 0, \quad t < 0.
\end{align*}
\]

Adjoint state method is to compute the adjoint action of Born operator \( DF[m] \) on any given vector \( r \), i.e., \( DF[m]^T r \).

DERIVATION OF ADJOINT STATE METHOD

Using a staggered grid scheme with second order in time, we do the simulation via the following time stepping procedure for \( k = 0, 1, \ldots, N - 1 \):

\[
\begin{align*}
p^0 &\equiv 0, \quad v^1 \equiv 0, \\
p^{k+1} &\equiv p^k - \kappa \Delta t \nabla \cdot v^{k+\frac{1}{2}} + \kappa \Delta t f^{k+\frac{1}{2}}, \\
v^{k+\frac{1}{2}} &\equiv v^{k+\frac{1}{2}} - b \Delta t \nabla p^{k+1},
\end{align*}
\]
in which $p$, $v$, $\kappa$ and $b$ are grid functions, and $\nabla v \cdot$ and $\nabla p$ are finite difference spatial discretizations of differential operators, and $\Delta t$ is the time step.

Similarly, Born simulation is computed as follows for $k = 0, 1, \ldots, N - 1$:

\[
\begin{align*}
\delta p^0 &= 0, \quad \delta v^\frac{1}{2} = 0, \\
\delta p^{k+1} &= \delta p^k - \kappa \Delta t \nabla v \cdot \delta v^{k+\frac{1}{2}} - \delta \kappa \Delta t \nabla v \cdot v^{k+\frac{1}{2}}, \\
\delta v^{k+\frac{1}{2}} &= \delta v^{k+\frac{1}{2}} - b \Delta t \nabla p \delta p^{k+1} - \delta b \Delta t \nabla p p^{k+1},
\end{align*}
\]

(3)

where $\delta p$, $\delta v$, $\delta \kappa$, and $\delta b$ are grid functions, and $\nabla \cdot$ and $\nabla$ are finite difference spatial discretizations of differential operators.

Regarding the above grid functions as vectors, we could write the system 3 in the following matrix form:

\[
Aw^{k+1} = -Bw^k - C^{k,k+1}q,
\]

(4)

where

\[
\begin{align*}
w^k &= \left( \begin{array}{c} \delta p^k \\ \delta v^{k+\frac{1}{2}} \end{array} \right), \\
q &= \left( \begin{array}{c} \delta k \\ \delta b \end{array} \right), \\
A &= A[\Delta t, \kappa, b, \nabla p] = \begin{pmatrix} \text{diag} \left\{ \frac{1}{\kappa} \right\} & 0 \\ \Delta t \nabla p & \text{diag} \left\{ \frac{1}{b} \right\} \end{pmatrix}, \\
B &= B[\kappa, b, \Delta t \nabla v \cdot] = \begin{pmatrix} -\text{diag} \left\{ \frac{1}{\kappa} \right\} & \Delta t \nabla v \cdot \\ 0 & -\text{diag} \left\{ \frac{1}{b} \right\} \end{pmatrix},
\end{align*}
\]

and

\[
C^{k,k+1} = \Delta t \begin{pmatrix} \text{diag} \left\{ \frac{1}{\kappa} \right\} \nabla v \cdot v^{k+\frac{1}{2}} & 0 \\ 0 & \text{diag} \left\{ \frac{1}{b} \right\} \nabla p p^{k+1} \end{pmatrix}.
\]

Remark: $C^{k,k+1}$ depends on both $v^{k+\frac{1}{2}}$ and $p^{k+1}$, which come from two time levels $k$ and $k + 1$ respectively.

Then, the above linear evolution can be written as:

\[
G(w, q) = G_w w + G_q q = 0,
\]

(5)

where

\[
\begin{align*}
G_w &= \begin{pmatrix} A & B \\ B & A \end{pmatrix}, \\
G_q &= \begin{pmatrix} C^{0,1} \\ C^{1,2} \\ \cdots \\ C^{N-1,N} \end{pmatrix}.
\end{align*}
\]

Thus, $w = -G_w^{-1}G_q q$, and then the discretized Born operator $DF[m]$ has the form

\[
DF[m] = -SG_w^{-1}G_q,
\]

(6)

\footnote{In this report, the same notations are used for operators in both continuous and discrete versions.}
where the sampling operator $S$ is defined to be the row vector of sampling operators: $S = (S^1, S^2, \ldots, S^N)$. Adjoint state method aims to compute the action of $DF[m]^T$ on any vector $r$, i.e.,

$$DF[m]^T r = -G_q^T G_w^{-T} S^T r.$$  \(7\)

The algorithm proceeds as follows:

**Step 1:** Apply the adjoint sampling operator:

$$r \mapsto w_r = S^T r = \begin{pmatrix} (S_1)^T \\ (S_2)^T \\ \vdots \\ (S_N)^T \end{pmatrix} r = \begin{pmatrix} w_1^r \\ w_2^r \\ \vdots \\ w_N^r \end{pmatrix}.$$

Since the sampling operator extracts the data from a state at specific locations for each time step, its adjoint inserts the data back into the state at those locations for each time step. In our case, it is convenient to write

$$w_k^r = \begin{pmatrix} w_{rp}^k \\ w_{rv}^k \end{pmatrix}$$

for $k = 1, 2, \ldots, N$. If only pressure is recorded, then $w_{rv}^k$ will always be zero for $k = 1, 2, \ldots, N$.

**Step 2 (backpropagation):** Solve the adjoint state system for the adjoint state vector (“receiver wavefield” in RTM):

$$G_w^T \xi = w_r,$$

where

$$G_w^T = \begin{pmatrix} A^T & B^T \\ B & A \end{pmatrix}, \quad \xi = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{pmatrix}.$$

Recalling that $w^k = \begin{pmatrix} \delta p^k \\ \delta v^{k+\frac{1}{2}} \end{pmatrix}$, let $\xi^k = \begin{pmatrix} \xi_p^k \\ \xi_v^{k+\frac{1}{2}} \end{pmatrix}$ for $k = 1, 2, \ldots, N$.

Since $A^T = \begin{pmatrix} \text{diag} \left\{ \frac{1}{\kappa} \right\} & -\Delta t \nabla_p^T \\ 0 & \text{diag} \left\{ \frac{1}{b} \right\} \end{pmatrix}$ and $B = \begin{pmatrix} \text{diag} \left\{ \frac{1}{\kappa} \right\} & 0 \\ -\Delta t \left( \nabla_v^T \right)^T & \text{diag} \left\{ \frac{1}{b} \right\} \end{pmatrix}$, then $G_w^T$ is upper triangular. Solution of this system becomes a backward substitution for $k = N, N-1, \ldots, 1$:

$$\xi_{p+1}^N = 0, \quad \xi_{v+\frac{1}{2}}^N = 0$$

$$\xi_v^{k+\frac{1}{2}} = \xi_v^{k+\frac{1}{2}} - b \Delta t \left( \nabla_v \right)^T \xi_p^{k+1} + bw_{rv}^{k+\frac{1}{2}},$$

$$\xi_p^k = \xi_p^{k+1} - \kappa \Delta t \nabla_p \xi_v^{k+\frac{1}{2}} + \kappa w_{rp}^k.$$
Given appropriate boundary conditions* (e.g., homogeneous Dirichlet’s boundary conditions), we have \( \nabla \cdot v = -\nabla^T_p \). Then, the above backward-recursion becomes for \( k = N, N - 1, \ldots, 1 \):

\[
\begin{align*}
\xi_p^{N+1} &= 0, \quad \xi_v^{N+\frac{1}{2}} = 0, \\
\xi_v^{k+\frac{1}{2}} &= \xi_v^{k+\frac{3}{2}} + b\Delta t \nabla_p \xi_p^{k+1} + bw_{rv}^{k+\frac{1}{2}}, \quad \text{(8)} \\
\xi_p^k &= \xi_p^{k+1} + \kappa \Delta t \nabla_v \cdot \xi_v^{k+\frac{1}{2}} + \kappa w_{rp}^k. \\
\xi_v^{k-\frac{1}{2}} &= \xi_v^{k+\frac{1}{2}} - b\Delta t \nabla_p \xi_p^k + bw_{rv}^{k-\frac{1}{2}}, \quad \text{(9)}
\end{align*}
\]

To reuse the forward simulator for this backward recursion, we need to appropriately swap the inner updating procedures (8) and (9) as follows for \( k = N, N - 1, \ldots, 1 \):

\[
\begin{align*}
\xi_p^{N+1} &= 0, \quad \xi_v^{N+\frac{1}{2}} = bw_{rv}^{N+\frac{1}{2}}, \\
\xi_p^k &= \xi_p^{k+1} - \kappa \hat{\Delta} t \nabla_v \cdot \xi_v^{k+\frac{1}{2}} + \kappa w_{rp}^k, \\
\xi_v^{k-\frac{1}{2}} &= \xi_v^{k+\frac{1}{2}} - b\hat{\Delta} t \nabla_p \xi_p^k + bw_{rv}^{k-\frac{1}{2}}, \quad \text{(10)}
\end{align*}
\]

where \( \hat{\Delta} t = -\Delta t \).

Remark: After rearranging equations (8) and (9), the backward recursion (10) shares the same structure as the forward time-stepping procedure (3) except for the time-marching directions (i.e., \( \hat{\Delta} t = -\Delta t \)). This fact becomes more clear with the comparison between (4) and the matrix form of (10) :

\[
\hat{A} \hat{\xi}^k = -\hat{B} \hat{\xi}^{k+1} - \hat{w}_r^k, \quad \text{(11)}
\]

in which

\[
\begin{align*}
\hat{\xi}^k &= \begin{pmatrix} \xi_p^k \\ \xi_v^{k-\frac{1}{2}} \end{pmatrix}, \\
\hat{w}_r^k &= \begin{pmatrix} w_{rp}^k \\ w_{rv}^{k-\frac{1}{2}} \end{pmatrix}, \\
\hat{A} &= A \left[ \Delta t, \kappa, b, \nabla_p \right] = A \left[ -\Delta t, \kappa, b, \nabla_p \right], \\
\hat{B} &= B \left[ \Delta t, \kappa, b, \nabla_p \right] = B \left[ -\Delta t, \kappa, b, \nabla_p \right].
\end{align*}
\]

*It is commonplace to assume zero pressure on the surface and employ so-called absorbing boundary conditions on the other sides of the rectangle to simulate wave propagation in an infinite domain.
Step 3 (imaging): compute $\mathcal{I} = DF[m]^T r = -G_q^T \xi$.

$$\mathcal{I} = -G_q^T \xi = -\sum_{k=1}^{N} (C^{k-1,k})^T \xi^k$$

$$= -\sum_{k=1}^{N} \begin{pmatrix} (\Delta t \text{ diag} \left\{ \frac{1}{\kappa} \right\} \nabla \cdot v^{k-\frac{1}{2}}) & 0 \\ 0 & (\Delta t \text{ diag} \left\{ \frac{1}{\beta} \right\} \nabla p^{k}) \end{pmatrix} \begin{pmatrix} \xi^k_p \\ \xi^k_v \end{pmatrix}$$

$$= -\sum_{k=2}^{N} \begin{pmatrix} (\Delta t \text{ diag} \left\{ \frac{1}{\kappa} \right\} \nabla \cdot v^{k-\frac{1}{2}}) & 0 \\ 0 & (\Delta t \text{ diag} \left\{ \frac{1}{\beta} \right\} \nabla p^{k-1}) \end{pmatrix} \begin{pmatrix} \xi^k_p \\ \xi^k_v \end{pmatrix}$$

Hence,

$$\mathcal{I} = -G_q^T \xi = -\sum_{k=2}^{N} (C^{k-1,k-1})^T \hat{\xi}^k;$$

where

$$C^{k,k} = \Delta t \begin{pmatrix} \text{ diag } \left\{ \frac{1}{\kappa} \right\} \nabla \cdot v^{k+\frac{1}{2}} & 0 \\ 0 & \text{ diag } \left\{ \frac{1}{\beta} \right\} \nabla p^{k} \end{pmatrix},$$

and

$$\hat{\xi}^k = \begin{pmatrix} \xi^k_p \\ \xi^k_v \end{pmatrix}.$$

Remark: After regrouping the summation, $C^{k,k}$ only depends on state variables at time level $k$. And it becomes natural to accumulate the sum in equation (12) term-by-term as the adjoint state variables $\xi^k$ are produced in the back-propagation loop (10). Actually, the accumulation proceeds in the same way as inserting Born source during the simulation (3): for $k = N, N-1, \ldots, 2$,

$$\mathcal{I}^N = \begin{pmatrix} I^N_{\kappa} \\ I^N_{\beta} \end{pmatrix} = 0,$$

$$I_{\kappa}^{k-1} = I_{\kappa}^k - \Delta t \xi^k_p \nabla \cdot v^{k-\frac{1}{2}},$$

$$I_{\beta}^{k-1} = I_{\beta}^k - \Delta t \xi^{k-\frac{1}{2}} \nabla p^{k-1},$$

$$\mathcal{I} = \begin{pmatrix} I_{\kappa} \\ I_{\beta} \end{pmatrix} = \begin{pmatrix} I_{\kappa}^1 \\ I_{\beta}^1 \end{pmatrix},$$

or

$$\mathcal{I}^N = \begin{pmatrix} I^N_{\kappa} \\ I^N_{\beta} \end{pmatrix} = 0,$$

$$I_{\kappa}^{k-1} = I_{\kappa}^k - \frac{1}{\kappa} \xi^k_p \Delta t \nabla \cdot v^{k-\frac{1}{2}},$$

$$I_{\beta}^{k-1} = I_{\beta}^k - \frac{1}{\beta} \xi^{k-\frac{1}{2}} \Delta t \nabla p^{k-1},$$

$$\mathcal{I} = \begin{pmatrix} I_{\kappa} \\ I_{\beta} \end{pmatrix} = \begin{pmatrix} I_{\kappa}^1 \\ I_{\beta}^1 \end{pmatrix},$$

(13)

(14)
where $p, v, \kappa,$ and $b$ are grid functions, and $\nabla \cdot$ and $\nabla$ are finite difference spatial discretizations of differential operators.

1D EXAMPLE

In the following, I will elaborate the above derivation for a 1D acoustic model with homogeneous Dirichlet’s boundary conditions:

\[
\left( \frac{1}{\kappa(z)} \frac{\partial p}{\partial t} + \nabla \cdot v \right) (z, t) = f(z, t), \quad (z, t) \in (0, z_{\text{max}}) \times (0, T),
\]
\[
\left( \frac{1}{b(z)} \frac{\partial v}{\partial t} + \nabla p \right) (z, t) = 0, \quad (z, t) \in (0, z_{\text{max}}) \times (0, T),
\]

$p(0, t) = p(z_{\text{max}}, t) = 0, \quad 0 < t < T,$

$v(0, t) = v(z_{\text{max}}, t) = 0, \quad 0 < t < T,$

$p(z, t) \equiv 0, \quad v(z, t) \equiv 0, \quad t \leq 0.$

Let $m(z) := (\kappa(z), b(z))$ denote the model. The forward map is

\[ F[m] := Sp = p(z_g, t), \]

i.e., $Sp := p(z_g, t)$ in which $z_g$ denotes the receiver depth.

Let’s discretize the above system with a 2-2 staggered grid scheme ($2^{nd}$ order in both time and space) on the following partition of $(z, t)$: with the space and time step sizes $\Delta z$ and $\Delta t$, $(z_i, t_j) = (i \Delta z, j \Delta t)$ for $i = 0, 1, \ldots, L; \quad j = 0, 1, \ldots, N$, and $z_L = z_{\text{max}}$ and $t_N = T$.

Denote

\[
p_i^j = p(i \Delta z, j \Delta t); \quad v_{i+1/2}^j = v \left( \left( i + \frac{1}{2} \right) \Delta z, \left( j + \frac{1}{2} \right) \Delta t \right)
\]

for $i = 1, \ldots, L - 1; \quad j = 1, \ldots, N$. The initial and boundary conditions yield

\[
p_0 = p_L = 0.
\]

Then, (2) becomes

\[
p_i^j = 0, \quad i = 1, \ldots, L - 1,
\]

\[
v_{i+1/2}^j = 0, \quad i = 1, \ldots, L,
\]

\[
p_0^k = p_L^k = 0, \quad k = 0, 1, 2, \ldots, N
\]
and for $k = 1, 2, \ldots, N - 1$

\[
\begin{align*}
    p_{i}^{k+1} &= p_{i}^{k} - \kappa_i \frac{\Delta t}{\Delta z} \left( v_{i+\frac{1}{2}}^{k+\frac{1}{2}} - v_{i-\frac{1}{2}}^{k+\frac{1}{2}} \right) + \kappa_i \Delta t f_{i}^{k+\frac{1}{2}}, \quad i = 1, 2, \ldots, L - 1, \\
v_{i+\frac{1}{2}}^{k+\frac{3}{2}} &= v_{i+\frac{1}{2}}^{k+\frac{1}{2}} - b_{i+\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( p_{i+1}^{k+1} - p_{i}^{k+1} \right), \quad i = 1, 2, \ldots, L - 2, \\
v_{\frac{1}{2}}^{k+\frac{3}{2}} &= v_{\frac{1}{2}}^{k+\frac{1}{2}} - b_{\frac{1}{2}} \frac{\Delta t}{\Delta x} p_{i}^{k+1}, \\
v_{L-\frac{1}{2}}^{k+\frac{3}{2}} &= v_{L-\frac{1}{2}}^{k+\frac{1}{2}} - b_{L-\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( -p_{L-1}^{k+1} \right). 
\end{align*}
\]

Similarly, (3) becomes

\[
\begin{align*}
    \delta p_{i}^{0} &= 0, \quad i = 1, \ldots, L - 1, \\
v_{i}^{\frac{3}{2}} &= 0, \quad i = 1, \ldots, L, \\
    \delta p_{k}^{0} &= \delta p_{L}^{k} = 0, \quad k = 0, 1, 2, \ldots, N
\end{align*}
\]

and for $k = 1, 2, \ldots, N - 1$

\[
\begin{align*}
    \delta p_{i}^{k+1} &= \delta p_{i}^{k} - \kappa_i \frac{\Delta t}{\Delta z} \left( \delta v_{i+\frac{1}{2}}^{k+\frac{1}{2}} - \delta v_{i-\frac{1}{2}}^{k+\frac{1}{2}} \right) - \delta \kappa_i \frac{\Delta t}{\Delta z} \left( v_{i+\frac{1}{2}}^{k+\frac{1}{2}} - v_{i-\frac{1}{2}}^{k+\frac{1}{2}} \right), \\
    \delta v_{i+\frac{1}{2}}^{k+\frac{3}{2}} &= \delta v_{i+\frac{1}{2}}^{k+\frac{1}{2}} - b_{i+\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( \delta p_{i+1}^{k+1} - \delta p_{i}^{k+1} \right) - \delta b_{i+\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( p_{i+1}^{k+1} - p_{i}^{k+1} \right), \\
    \delta p_{L}^{k+1} &= \delta p_{L}^{k} - \kappa_{L} \frac{\Delta t}{\Delta z} \left( \delta v_{L-\frac{1}{2}}^{k+\frac{1}{2}} - \delta v_{L+\frac{1}{2}}^{k+\frac{1}{2}} \right) - \delta \kappa_{L} \frac{\Delta t}{\Delta z} \left( v_{L-\frac{1}{2}}^{k+\frac{1}{2}} - v_{L+\frac{1}{2}}^{k+\frac{1}{2}} \right), \\
    \delta v_{L-\frac{1}{2}}^{k+\frac{3}{2}} &= \delta v_{L-\frac{1}{2}}^{k+\frac{1}{2}} - b_{L-\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( -\delta p_{L-1}^{k+1} \right) - \delta b_{L-\frac{1}{2}} \frac{\Delta t}{\Delta x} \left( -p_{L-1}^{k+1} \right). 
\end{align*}
\]

The above system yields the same matrix form as (4), i.e.,

\[
Aw^{k+1} = -Bw^{k} - C^{k,k+1}q,
\]

in which

\[
\begin{align*}
    w^{k} &= \left( \delta p_{1}^{k}, \ldots, \delta p_{L-1}^{k}, \delta v_{\frac{1}{2}}^{k+\frac{1}{2}}, \ldots, \delta v_{L-\frac{1}{2}}^{k+\frac{1}{2}} \right)^{T} \in \mathbb{R}^{2L-1}, \\
    q &= \left( \delta \kappa_{1}, \ldots, \delta \kappa_{L-1}, \delta b_{\frac{1}{2}}, \ldots, \delta b_{L-\frac{1}{2}} \right)^{T} \in \mathbb{R}^{2L-1}, \\
    A &= \begin{pmatrix} \mathsf{diag} \left\{ \frac{1}{\Delta z} \right\} & 0 \\ \Delta t \nabla_{p} & \mathsf{diag} \left\{ \frac{1}{\Delta t} \right\} \end{pmatrix}, \\
    B &= -\begin{pmatrix} \mathsf{diag} \left\{ \frac{1}{\Delta t} \right\} & -\Delta t \nabla_{v} \cdot \nabla \end{pmatrix}, \\
    C^{k,k+1} &= \Delta t \begin{pmatrix} 0 & \mathsf{diag} \left\{ \frac{1}{\Delta t} \right\} \nabla_{p} \nabla \end{pmatrix},
\end{align*}
\]
where

\[
\text{diag}\left\{ \frac{1}{\kappa} \right\} = \begin{pmatrix} \frac{1}{\kappa_1} & \cdots & \frac{1}{\kappa_{L-1}} \\ \vdots & \ddots & \vdots \\ \frac{1}{\kappa_{L-1}} & \cdots & \frac{1}{\kappa_L} \end{pmatrix} \in \mathbb{R}^{(L-1)\times(L-1)}, \\
\text{diag}\left\{ \frac{1}{b} \right\} = \begin{pmatrix} \frac{1}{b_1} & \cdots & \frac{1}{b_{L-1}} \end{pmatrix} \in \mathbb{R}^{L\times L},
\]

\[
\nabla_p = \frac{1}{\Delta x} \begin{pmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{pmatrix} \in \mathbb{R}^{L\times(L-1)}, \\
\nabla_v = \frac{1}{\Delta x} \begin{pmatrix} -1 & 1 \\ \vdots & \vdots \\ -1 & 1 \end{pmatrix} \in \mathbb{R}^{(L-1)\times L},
\]

\[
\nabla_v \cdot v^{k+\frac{1}{2}} = \frac{1}{\Delta x} \begin{pmatrix} -1 & 1 \\ \vdots & \vdots \\ -1 & 1 \end{pmatrix} \begin{pmatrix} v_{k+\frac{1}{2}}^1 \\ \vdots \\ v_{L-\frac{1}{2}}^{k+\frac{1}{2}} \end{pmatrix},
\]

\[
\nabla_p p^{k+1} = \frac{1}{\Delta x} \begin{pmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{pmatrix} \begin{pmatrix} p_{k+1}^1 \\ \vdots \\ p_{L-1}^{k+1} \end{pmatrix}.
\]

Notice that \((\nabla_v)^T = -\nabla_p\) and equations (5)-(7) stay the same.

Given any vector \(r \in \mathbb{R}^N\), adjoint state method aims to compute

\[
DF[m]^T r = -G_q^T G_w^T S^T r
\]

in the standard three steps:

**Step 1:** Apply the adjoint sampling operator:

\[
r \mapsto w_r = S^T r = \begin{pmatrix} (S^1)^T \\ (S^2)^T \\ \vdots \end{pmatrix} \begin{pmatrix} w_1^r \\ w_2^r \\ \vdots \end{pmatrix} \in \mathbb{R}^{N(2L-1)}
\]

In this example, the sampling operator \(S\) extracts pressure at depth \(z_q\) for each time level, and its adjoint inserts data \(r\) back into the state \(w_r\) at position \(\gamma = \frac{z_q}{\Delta z}\) for each time level. For simplicity, assume \(\frac{z_q}{\Delta z}\) is an integer (if not,
we can always interpolate this value to nearby points, e.g., \( \lfloor \frac{z}{\Delta z} \rfloor \) and \( \lceil \frac{z}{\Delta z} \rceil + 1 \). Thus, for \( k = 1, 2, \ldots, N \),

\[
w^k_r = (S^k)^T r = \begin{pmatrix} w^k_{rp} \\ w^k_{rv} + \frac{1}{2} \end{pmatrix} \in \mathcal{R}^{2L-1},
\]

where \( w^k_{rp} = (0, \ldots, r^k_{i/h}, 0, \ldots, 0)^T \in \mathcal{R}^L \), and \( w^k_{rv} + \frac{1}{2} = 0 \in \mathcal{R}^L \).

**Step 2 (back propagation):** Solve the adjoint state system \( G^T_u \zeta = w_r \) for the adjoint state vector \( \zeta \in \mathcal{R}^{N(2L-1)} \) through a backward substitution for \( k = N, N - 1, \ldots, 1 \):

\[
\zeta^N_i = (\xi^N_p)_i = 0, \quad i = 1, \ldots, L - 1,
\]

\[
\zeta^N_{L-1+i} = \left(b \xi^N_{i+\frac{1}{2}}\right)^T_{i-\frac{1}{2}} = 0, \quad i = 1, \ldots, L,
\]

and for \( k = N, N - 1, \ldots, 1 \)

\[
\zeta^k_i = \zeta^{k+1}_i - \kappa_i \frac{\Delta t}{\Delta z} (\xi^k_{L+i} - \xi^k_{L-1+i}) + \kappa_i r^k_{i} \delta_{p}, \quad i = 1, 2, \ldots, L - 1,
\]

\[
\zeta^{k-1}_{L+i} = \zeta^k_{L+i} - b_{i+1} \frac{\Delta t}{\Delta x} (\xi^k_{i+1} - \xi^k_{i-1}), \quad i = 1, 2, \ldots, L - 2,
\]

\[
\zeta^{k-1}_L = \zeta^k_L - \frac{1}{2} \frac{\Delta t}{\Delta x} \zeta^k_1,
\]

\[
\zeta^{k-1}_{2L-1} = \zeta^k_{2L-1} - b_{L-1} \frac{\Delta t}{\Delta x} (\xi^k_{L-1}).
\]

where \( \Delta t = -\Delta t \) and \( \delta_{p} = \left\{ \begin{array}{ll} 1, & i = \gamma \text{ for } i = 1, \ldots, L - 1. \\ 0, & i \neq \gamma \end{array} \right. \)

Notice that the backward recursion (17) shares the same structure as the forward time-stepping procedure (15) except for the source term and time-marching directions (i.e., \( \Delta t = -\Delta t \)). The matrix form of this backward time-stepping is:

\[
\hat{A} \hat{\zeta} = -\hat{B} \hat{\zeta}^{k+1} + \hat{w},
\]

in which

\[
\hat{\zeta}^k = \begin{pmatrix} \xi_p^k \\ \xi^k_{i-\frac{1}{2}} \end{pmatrix}, \quad \hat{w}^k_r = \begin{pmatrix} w^k_{rp} \\ w^k_{rv} + \frac{1}{2} \end{pmatrix},
\]

\[
\hat{A} = A \left[ \Delta t, \kappa, b, \nabla_p \right] = A \left[ -\Delta t, \kappa, b, \nabla_p \right],
\]

\[
\hat{B} = B \left[ \Delta t, \kappa, b, \nabla_p \right] = B \left[ -\Delta t, \kappa, b, \nabla_p \right].
\]

Hence, the same implementation (e.g., IWave) accommodates both the forward and backward time-stepping.
Step 3 (imaging): compute $DF[m]^T r = -C_q^T \xi$ in the same way as (12), i.e.,

$$
I = -G_q^T \xi = - \sum_{k=2}^{N} (C^{k-1,k-1})^T \hat{\xi}^k,
$$

where

$$
C^{k,k} = \Delta t \begin{pmatrix}
\begin{pmatrix} 1 \over \kappa \end{pmatrix} & \nabla v \cdot v^{k+\frac{1}{2}} & 0 \\
0 & \begin{pmatrix} 1 \over b \end{pmatrix} \nabla p^k
\end{pmatrix}, \quad \hat{\xi}^k = \begin{pmatrix} \hat{\xi}_p^k \\ \hat{\xi}_v^k \end{pmatrix}.
$$

So

$$
(G_q^T \xi)_i = \frac{1}{\kappa_i} \sum_{k=2}^{N} \hat{\xi}_i^k \Delta t \left( v^{k-\frac{1}{2}} - v^{k-\frac{1}{2}} \right), \quad i = 1, \ldots, L - 1,
$$

$$
(G_q^T \xi)_L = \frac{1}{b_L^\frac{1}{2}} \sum_{k=2}^{N} \hat{\xi}_L^k \Delta t \left( p^{k-1}_1 \right),
$$

$$
(G_q^T \xi)_{L+1} = \frac{1}{b_{L+\frac{1}{2}}} \sum_{k=2}^{N} \hat{\xi}_{L+1}^k \Delta t \left( p^{k-1}_{L+1} - p^{k-1}_L \right), \quad i = 1, \ldots, L - 2,
$$

$$
(G_q^T \xi)_{2L-1} = \frac{1}{b_{L-1}^\frac{1}{2}} \sum_{k=2}^{N} \hat{\xi}_{2L-1}^k \Delta t \left( -p^{k-1}_{L-1} \right).
$$

(19)

Remark: After regrouping the summation, $C^{k,k}$ only depends on state variables at time level $k$. And it becomes natural to accumulate the sum in equation (12) term-by-term as the adjoint state variables $\xi^k$ are produced in the back propagation loop (10).

**IMPLEMENTATION**

**IWAVE Basics**

Before discussing an implementation of adjoint state computation with IWAVE, I would like to briefly introduce IWAVE. IWAVE is based on two major concepts: data storage and time-stepping functions. With the current physical states (say $p^{k-1/2}$ and $v^k$) and input data (say $\kappa$, $b$), a time-stepping function (presented by `TIMESTEP_FUN` type) is called to update the physical states $p$ and $v$ (to $p^{k+1/2}$ and $v^{k+1}$). All the physical variables and input data are stored in multidimensional arrays described by the `RARR` data type. The arrays defining a particular model constitute a domain described by the `RDOM` data type. IWAVE provides the following type to describe a pointer to the time-stepping function:

```c
typedef int (*TIMESTEP_FUN)(RDOM *rdom, int iarr, void * tspars);
```
Here, \texttt{rdom} is a pointer to the \texttt{RDOM} struct whose \texttt{RARR}s represent dynamic and static fields in the acoustic simulation. The index \texttt{iarr} indicates which \texttt{RARR}, representing a dynamic field, is to be updated, and \texttt{tspars} points to a struct containing appropriate time-stepping parameters (difference coefficients, scaled quotients of steps, etc.).

\texttt{IWAVE} stores all the allocated domain, the virtual computational domains, the pointers to time-stepping functions and other additional parameters in an \texttt{IMODEL} object:

\begin{verbatim}
typedef struct IMODEL {
    TIMESTEP_FUN ts;   /* pointers to time-stepping functions */
    void *tspars;      /* pointers to time-stepping parameters */
    RDOM ld_a, ld_c, ld_p;  /* allocated domain and computational virtual domains */
    RDOM *ld_s, *ld_r;  /* receive and send virtual domains */
    .......
} IMODEL;
\end{verbatim}

An \texttt{IMODEL} object together with parallel and other additional information constitute an \texttt{IWAVE} object:

\begin{verbatim}
typedef struct {
    PARALLELINFO pinfo; /* parallel information */
    IMODEL model;
    ...
} IWAVE;
\end{verbatim}

A C++ wrapper to the \texttt{IWAVE} struct in IWAVE++ is the class \texttt{IWaveState}, which contains an \texttt{IWAVE} type object \texttt{iwstate} and provides methods to access the information in \texttt{iwstate}:

\begin{verbatim}
class IWaveState {
    ...
    protected:
    ...
    mutable PARARRAY pars;   /* parameter array */
    mutable IWAVE iwstate;
    TSIIndex tsi;
    /* time object indicating the current state status */
    ...
    public:
    ...
}
\end{verbatim}
In adjoint state computation, one needs to appropriately get access to both the forward and backward fields. In our previous implementation of the Born simulator (Sun and Symes, 2010), we developed the class \texttt{IWaveLinState} to maintain information for the tangent state consists of the reference and perturbation states. Regarding the backward and forward fields as the tangent and reference fields respectively, we are able to adopt the same class \texttt{IWaveLinState} for adjoint state computation (see Figure 1). Recall that the class \texttt{IWaveLinState} is derived from the class \texttt{IWaveState} as:

```cpp
class IWaveLinState: public IWaveState {
    private:
        mutable IWave linstate; /* backward state */
        mutable MODEL dmod;
            /* store pointers to the non-dynamic migrated model*/
        TSIndex ltsi;
            /* time object indicating the current bwd-state status */
    ...
    public:
        ...
};
```

![Figure 1: Diagram of IWaveLinState](image)

Figure 1: Diagram of \texttt{IWaveLinState}: Arrows represent pointer copies. Memory is originated and managed by the ref state \texttt{IWaveState::iwstate} and lin state \texttt{linstate}. The \texttt{dmod} consists of virtual arrays and holds pointers to the non-dynamic fields storing accumulated model. No memory leaks created.

The reference state, contained in the base class \texttt{IWaveState}, holds the static fields (e.g., $\kappa$ and $b$) used in both forward and backward simulations, and the forward
dynamic fields (e.g., \( p, \mathbf{v} \)); the \( \text{linstate} \) holds the backward dynamic fields (e.g., \( \xi_p, \xi_v \)), and the pointers to the static fields of the reference state; the pointers to non-dynamic fields originally supplied in \( \text{linstate} \) are assigned to an \( \text{IMODEL} \) object \( \text{dmod} \), which stores the accumulated image.

Now, I would like to introduce our implementation of adjoint state computation and discuss the most basic components of this computation.

As a time-stepping procedure, our computation follows the following simulation flow:

1. initialization step: initialize forward field \( \overrightarrow{\mathbf{q}} \), backward field \( \overleftarrow{\mathbf{q}} \), and their time objects \( \overrightarrow{\mathbf{t}} \) and \( \overleftarrow{\mathbf{t}} \), e.g., \( \overrightarrow{\mathbf{q}} = \overleftarrow{\mathbf{q}} = 0 \), forward time level \( \overrightarrow{\mathbf{t}}.it = 0 \), backward time level \( \overleftarrow{\mathbf{t}}.it = N \)

2. outer loop (backward simulation loop):
   (a) pre-step: synchronize forward and backward fields, i.e., run forward-Sim until \( \overrightarrow{\mathbf{t}}.it == \overleftarrow{\mathbf{t}}.it \)
   (b) step: update backward fields \( \overrightarrow{\mathbf{q}} \)
   (c) post-step:
      - insert source into backward fields
      - accumulate image, i.e., apply imaging condition on the current forward and backward fields
      - update backward time \( \overleftarrow{\mathbf{t}} \)

Notice that when the forward simulation terminates at time level \( k \), the forward state holds the forward fields at time level \( k - 1 \); when the backward simulation terminates at time level \( k \), the backward state holds the backward fields at time level \( k + 1 \).

In the following, I discuss the most basic components of this computation: the backward simulation, forward simulation, and image accumulation.

**Forward Simulation**

As discussed in (Sun and Symes, 2010), we use the \( \text{Sim} \) and its derived classes from the TSOpt abstract time-stepping package to implement the time loops. For a standard simulation, the actions taken in each time loop are:

1. initialization: initialize wave-field;
2. step: call time-stepping functions to update pressure and particle velocities;
3. post-step:
- insert source;
- sample and write out the results to traces (if needed);
- update time object;

The Sim and its derived classes also implement other strategies, such as checkpointing, which is adopted in our adjoint state computation. To use the checkpointing enabled simulation class CPSim, we just need to provide it a specific stack class derived from the virtual base class stackBase, which implements the most basic methods to store and manage wave-fields at different time levels, such as \texttt{push\_back()}, \texttt{pop\_back()}, \texttt{at(int)}, etc..

**Backward Simulation**

As discussed in Section 2, the adjoint state vector $\xi$ (backward fields or “receiver wavefield” in RTM) is computed via the backward recursion (10):

For $k = N, N - 1, \ldots, 1$:

$$
\begin{align*}
\xi_{p}^{N+1} &= 0, \quad \xi_{v}^{N+\frac{1}{2}} = bw_{r_{v}}^{N+\frac{1}{2}}, \\
\xi_{p}^{k} &= \xi_{p}^{k+1} - \kappa \Delta t \nabla_{v} \cdot \xi_{v}^{k+\frac{1}{2}} + \kappa \Delta t w_{r_{p}}^{k}, \\
\xi_{v}^{k-\frac{1}{2}} &= \xi_{v}^{k+\frac{1}{2}} - b \Delta t \nabla_{p} \xi_{p}^{k} + b \Delta t w_{r_{v}}^{k-\frac{1}{2}},
\end{align*}
$$

where $\Delta t = -\Delta t$.

This backward recursion shares the same structure as the forward time-stepping procedure (3) except for the time-marching directions ($\Delta t = -\Delta t$). So we are able to reuse the time-stepping functions in the Born simulator to do the backward propagation. And the only change needed is to flip the signs of the time-stepping parameters for the backward field during initialization.

**Image Accumulation**

Recall (13) in Section 2, the accumulation proceeds in a similar way as inserting Born source during the simulation (3): for $k = N, N - 1, \ldots, 2$,

$$
\begin{align*}
\mathcal{I}^{N} &= \left( \frac{\mathcal{I}_{b}^{N}}{\mathcal{I}_{b}^{r}} \right) = 0, \\
\mathcal{I}_{r}^{k-1} &= \mathcal{I}_{r}^{k} - \Delta t \xi_{p}^{k} \nabla_{v} \cdot \textbf{v}^{k-\frac{1}{2}}, \\
\mathcal{I}_{b}^{k-1} &= \mathcal{I}_{b}^{k} - \Delta t \xi_{v}^{k-\frac{1}{2}} \nabla_{p} \xi_{p}^{k-1}, \\
\mathcal{I} &= \left( \frac{1}{\lambda} \frac{\mathcal{I}_{r}}{\mathcal{I}_{b}} \right),
\end{align*}
$$

Sun and Symes (2010) presents a way to insert Born source in simulation (3) via generalized time-step functions:
int gts(RDOM *dom, RDOM *rdom, RDOM *cdom, int iarr, void *pars);

where dom, rdom and cdom are pointers to the RDOM objects that respectively hold 
dynamic perturbation fields, reference fields and non-dynamic perturbation model. 
Basically, these time-step functions compute

\[ \delta p = \delta p - \Delta t \delta \kappa \nabla \cdot \mathbf{v} \]

and

\[ \delta \mathbf{v} = \delta \mathbf{v} - \Delta t \delta b \nabla p \]

, which share a similar form as

\[ I_\kappa = I_\kappa - \Delta t \xi_p \nabla \cdot \mathbf{v} \]

and

\[ I_b = I_b - \Delta t \xi_v \nabla p \]

, except that the accumulation procedure is updating non-dynamic fields instead of 
dynamic fields.

So we change the signature of generalized time-stepping functions to

int gts(RDOM *dom, RDOM *rdom, RDOM *cdom, int iarr, void *pars, int _fwd);

Based on the flag _fwd, during image accumulation, the generalized time-stepping 
functions swap the pointers between dom and cdom before updating fields and then 
swap them back when updating is finished.

NUMERICAL EXAMPLES

In this section, I will verify this adjoint state computation via three numerical exam-
pies. Recall that adjoint state computation aims to compute the action of \( DF[m]^T \) 
on any vector \( r \), i.e., \( DF[m]^T r \).

Example I: single-shot experiment

This example uses a simple homogeneous model \( m ( \kappa = 11109 \text{ MPa}, \rho = 2100 \text{ kg/m}^3 \Rightarrow \text{acoustic velocity } c = 2.3 \text{ km/s} ) \), and chooses the adjoint source \( r \) to be the first 
order pressure perturbation computed by our Born simulator for a model perturbation 
\( \delta m \) shown in Figure 2 ( \( \delta \kappa = 987 \text{ Mpa} \Rightarrow \delta c = 0.1 \text{ km/s} \) ). A point source is located 
at the position \((3000, 40) \text{ m}\), and receivers are placed at positions \((3100 + i \times 10, 80) \text{ m}\) 
for \( i = 0, \ldots, 99 \).

Figure 3 shows the adjoint source \( r : = DF[m] \delta m \); Figure 4 and 5 respectively 
present the migrated bulk modulus and buoyancy \(( (\kappa_m, b_m) : = DF[m]^T r ) \). Table 1 
shows that adjoint relation holds for this computation. As shown, this adjoint state 
computation correctly yields the action of \( DF[m]^T \) on \( r \).
\begin{align*}
(DF[m]x, y) & \quad 1.94933750e+05 \\
(x, DF[m]^Ty) & \quad 1.94933859e+05 \\
\|DF[m]x\|y\| & \quad 1.94933750e+05 \\
\frac{|(DF[m]x, y)-(x, DF[m]^Ty)|}{\|DF[m]x\|y\|} & \quad 5.61088086e-07 \\
100 \ast \text{macheps} & \quad 1.19209290e-05
\end{align*}

Table 1: adjoint relation holds when $\frac{|(DF[m]x, y)-(x, DF[m]^Ty)|}{\|DF[m]x\|y\|} < 100\ast\text{macheps}$; Here, $A = DF[m]$, $x = \delta m$, $y = r = DF[m]\delta m$.

Figure 2: Block Perturbation: $\delta\kappa = 987$ Mpa $\Rightarrow \delta c = 0.1 \ km/s$
Figure 3: Adjoint Source: first order pressure perturbation $r := DF[m]δm$

Figure 4: Migrated Bulk modulus: part of $DF[m]^Tr$
Example II: multiple-shots experiment

This example is based on the same setting as the previous example, except that there are 20 shots in this computation. The shots are located at positions 
$(2000 + i \times 100, 40) \text{ m}$ for $i = 0, \ldots, 19$, and receivers corresponding to the $i'th$ shot are placed at positions $(2100 + i \times 100 + j \times 10, 80) \text{ m}$ for $j = 0, \ldots, 99$.

Figure 6 shows the adjoint source $r := DF[m] \delta m$; Figure 7 and 8 respectively present the migrated bulk modulus and buoyancy $((\kappa_m, b_m) := DF[m]^T r)$. Table 2 shows that adjoint relation holds for this computation. As shown, this adjoint state computation correctly yields the action of $DF[m]^T$ on $r$. 
Table 2: adjoint relation holds when \[
\frac{|(DF[m]x,y) - (x, DF[m]^T y)|}{||DF[m]x||y||} < 100 \times \text{macheps}; \]
Here, \(A = DF[m], x = \delta m, y = r = DF[m]\delta m\)

Figure 6: Adjoint Source: first order pressure perturbation \(r := DF[m]\delta m\)
Figure 7: Migrated Bulk modulus: part of $DF[m]^T r$

Figure 8: Migrated Buoyancy: part of $DF[m]^T r$
Example III: dot-product test

In a dot-product test, we are comparing two inner products $(DF[m]x, y)$ and $(x, DF[m]^Ty)$ for randomly generated vectors $x = (\delta\kappa, \delta\theta)$ and $y$. This example chooses the same reference model $m$ as the one used in the previous two examples. A point source is located at the position $(3300, 40)$ m, and receivers are placed at positions $(2650 + i \times 20, 80)$ m for $i = 0, \ldots, 49$. Figure 9, 10, and 11 respectively present randomly generated model perturbation $x(\delta\kappa, \delta\theta)$ and adjoint source $y$. The scattered wave-field $DF[m]x$ is shown in Figure 12. The migrated model perturbation $DF^T[m]y$ is shown in 13 and 14. Table 3 shows that adjoint relation holds for this computation. As shown, this adjoint state computation correctly yields the action of $DF[m]^T$ on a random vector $y$.

Figure 9: Random Perturbation: $\delta\kappa$
Figure 10: Random Perturbation: $\delta b$

Figure 11: Random Adjoint Source: $y$
Figure 12: Random Adjoint Source: $DF[m]x$

Figure 13: Migrated Bulk modulus: part of $DF[m]^Ty$
Figure 14: Migrated Buoyancy: part of $DF[m]T_y$

Table 3: adjoint relation holds when $\frac{|(DF[m]x, y)-(x, DF[m]^T y)|}{\|DF[m]x\|\|y\|} < 100 \ast \text{macheps}$; Here, $A = DF[m]$, $x = \delta m$, $y = r = DF[m] \delta m$
SUMMARY

In this report, I presents a derivation of adjoint state algorithm for an acoustic system discretized by staggered grid finite difference schemes, and briefly describe a way to implement the adjoint action of Born operator, which reuse IWave to do the core computation. The implementation achieves the desired results. More importantly, we have wrapped the modeling package IWave as an RVL operator class, which now includes the three basic methods. Now, it becomes much more straightforward to embed such a RVL operator into a general optimization framework for inversion, which will be the next step.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under grant DMS 0620821, and by the sponsors of The Rice Inversion Project.

REFERENCES

Sun, D. and Symes, W. W., 2010, IWave implementation of born simulation: Technical Report TR10-05, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.
Terentyev, I., 2009, A software framework for finite difference simulation: Technical Report TR09-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.