IWAVE Implementation of Born Simulation

Dong Sun and William W Symes

ABSTRACT

The single-scattering (or Born) approximation is the most fundamental assumption shared by all seismic imaging methods, and plays a crucial role in the nonlinear waveform inversion, an iterative process of linearized inversions. The Born simulator (linearized forward map) shares a computational core with the corresponding simulator (forward map), which has been well implemented in the modeling package IWAVE. This report focuses on implementing the Born simulator based on IWAVE, and reviews the main adaptations we made in IWAVE to accommodate such an implementation in C++.
Our goal is to construct a C++ wrapper of IWAVE, which fits into a general framework for inversion. This report is the first of several describing an implementation of such a wrapper.

INTRODUCTION

A common objective of reflection seismology is to make inferences about physical features (*model*) of subsurface (e.g., velocity) from data (*seismogram*) recorded on or near the surface. This inverse problem is often formed as a simulation-driven optimization problem in Hilbert space. Its implementation involves different levels of abstraction: simulation of wave propagation requires a variety of computational types and data structures specific to physical modeling and numerical implementation; Instead, optimization algorithms involve a more abstract layer of mathematical constructs (vectors, functions, gradients, ...) which are independent of physics and its numerical realization. Object-oriented programming offers a way to combine different levels of abstraction (e.g., finite difference grids v.s. vectors in Hilbert space) while keeping them in separate code. A concrete example is the Rice Vector Library (RVL) (Padula et al., 2009). 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. We will use the interfaces provided by RVL to implement this inversion. First of all, we need 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 and its adjoint modeling). This report is the first of several describing an operator implementation for acoustics based on the modeling package IWAVE (Terentyev, 2009).

IWAVE offers many advantages: it provides a parallel framework for solving time-dependent partial differential equations, has lots of modeling options already implemented for acoustics, and is well-tested and built in C. However, IWAVE doesn't refer to any of the ingredients of optimization, such as vectors, operators, functions, gradients, etc.. To embed it in inversion, we need to "view" IWAVE as defining an operator in a vector space, and "wrap" it in an RVL operator class, with methods to get its value, derivative and adjoint derivative.

IWAVE is a time-stepping algorithm; and derivative and adjoint derivative operators for time-stepping operators 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. IWAVE was written to make this adaptation as straightforward as possible. This includes implementing the derivative and adjoint derivative using IWAVE with minimal modifications. We call the resulting system of classes IWAVE++. This report covers the derivative.

THE ACOUSTIC MODEL

The acoustic model connects the pressure field $p(x,z,t)$, the particle velocity $\mathbf{v}(x,z,t)$, the buoyancy $b(x,z) = \frac{1}{\rho(x,z)}$ (the density $\rho(x,z)$), and the bulk modulus $\kappa(x,z)$ through the wave equations

$$\begin{aligned} \frac{1}{\kappa}\frac{\partial p}{\partial t} + \nabla\cdot\mathbf{v} &= \omega(t)\delta\left(x-x_s, z-z_s\right), && (1)\\ \frac{1}{b}\frac{\partial \mathbf{v}}{\partial t} + \nabla p &= 0, && (2)\\ p\equiv 0,\ \mathbf{v}\equiv\mathbf{0}, &\qquad t<0. \end{aligned}$$

The right-hand side of (1) represents an isotropic point source radiating with time-varying (transient) intensity $\omega(t)$ ("the source wavelet"). Let $m := (\kappa, b)$ denote the model (parameter vector).

The above equation system defines the forward map

$$F[m] := Sp, \tag{3}$$

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

The linearized forward map at model m is defined as

$$DF[m]\delta m := S\delta p = \left\{\delta p(x_r, z_r, t)\right\}, \tag{4}$$

where δm is a model perturbation, δp and $\delta\mathbf{v}$ are the corresponding first-order wave-field perturbation and solve the following equation system

$$\begin{aligned}
\frac{1}{\kappa}\frac{\partial \delta p}{\partial t} + \nabla\cdot\delta\mathbf{v} &= -\frac{\delta\kappa}{\kappa}\nabla\cdot\mathbf{v}, && (5)\\
\frac{1}{b}\frac{\partial \delta\mathbf{v}}{\partial t} + \nabla\delta p &= -\frac{\delta b}{b}\nabla p, && (6)\\
\delta p \equiv 0,\ \delta\mathbf{v} \equiv \mathbf{0}, &\qquad t<0.
\end{aligned}$$

As the above two equation systems have almost the same form other than the right hand side terms, we are using this similarity to adapt the existing simulator (IWAVE) to a Born simulator.

IMPLEMENTATION OF BORN SIMULATION

Before discussing the implementation of the Born simulator, I would like to briefly introduce IWAVE. To demonstrate the idea, a staggered grid scheme with second order in time is used to discretize the two equation systems.

Forward Simulation

Using a staggered grid scheme with second order in time, we do the simulation via the following time stepping procedure:

$$\begin{aligned}
p^{k+1/2} &= p^{k-1/2} - \kappa\Delta t\,\nabla\cdot\mathbf{v}^k + \kappa\omega(t)\delta\left(x-x_s, z-z_s\right), && (7)\\
\mathbf{v}^{k+1} &= \mathbf{v}^k - b\Delta t\,\nabla p^{k+1/2}, && (8)
\end{aligned}$$

in which p, $\mathbf{v}$, κ and b are grid functions, and $\nabla\cdot$ and ∇ are finite difference spatial discretizations of differential operators, and Δt is the time-step.

The implementation of IWAVE is based on two major concepts: data storage and time-stepping functions. With the current physical states (say $p^{k-1/2}$ and $\mathbf{v}^k$) and input data (say κ, b, etc.), a time-stepping function (i.e., presented by `TIMESTEP_FUN` type) is called to update the physical states p and $\mathbf{v}$ (to $p^{k+1/2}$ and $\mathbf{v}^{k+1}$). 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:

```
typedef int (*TIMESTEP_FUN)(RDOM *rdom, int iarr, void * tspars);
```

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

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

```
 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;
```

An `IMODEL` object together with parallel and other additional information constitute an `IWAVE` object:

```
typedef struct {
    PARALLELINFO pinfo;    /* parallel information */
    IMODEL model;
    ...
  } IWAVE;
```

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

```
class IWaveState {
    ...
  protected:
    ...
    mutable PARARRAY pars;       /* parameter array    */
    mutable IWAVE iwstate;
    TSIndex tsi;
         /* time object indicating the current state status */
    ...
  public:
    ...
}
```

We use `Sim` classes from the TSOpt abstract time-stepping package to implement the time loops. Briefly speaking, the actions taken in each time loop are:

- step: the calls to time-stepping function to update the pressure and particle velocities, and exchange information among processors in an appropriate order;

- post-step:
  - the calls to insert source appropriately;
  - the calls to sample and write out the results to traces;
  - the calls to update state time.

IWAVE structures its time loops in exactly this way, so the `TSOpt::Sim` classes are simple wrappers around the corresponding IWAVE functions, and all of the complicated data exchange code written into IWAVE is encapsulated and re-used. In addition, the `Sim` classes may implement other functions, such as check-pointing, needed in forward or adjoint simulation.

Born Simulation

As in the previous section, we do Born simulation via the following time stepping procedure:

$$\delta p^{k+1/2} = \delta p^{k-1/2} - \kappa \Delta t\, \nabla \cdot \delta \mathbf{v}^k - \delta\kappa\, \Delta t\, \nabla \cdot \mathbf{v}^k, \tag{9}$$

$$\delta \mathbf{v}^{k+1} = \delta \mathbf{v}^k - b\Delta t\, \nabla \delta p^{k+1/2} - \delta b\, \Delta t\, \nabla p^{k+1/2}, \tag{10}$$

where δp, $\delta \mathbf{v}$, κ, $\delta\kappa$, b and δb are grid functions, and $\nabla\cdot$ and ∇ are finite difference spatial discretizations of differential operators. As the reference fields are used in every time step of Born simulation, we need to create a new `RDOM` object to load the perturbation fields.

The update to δp is the sum of $-\kappa\Delta t\, \nabla \cdot \delta\mathbf{v}^k$ and $-\delta\kappa\, \Delta t\, \nabla \cdot \mathbf{v}^k$. Both of the two terms have the same form as $-\kappa\Delta t\, \nabla \cdot \mathbf{v}^k$, which is the update to p computed via a call to the time-stepping function in the forward simulation. So we want to reuse the time-stepping functions in IWAVE to update the perturbed wave-field.

To compute $-\kappa\Delta t\, \nabla \cdot \mathbf{v}^k$, a time-stepping function needs the reference arrays κ and $\mathbf{v}$, which are contained in the same `RDOM` object. As a contrast, to compute $-\kappa\Delta t\, \nabla\cdot\delta\mathbf{v}^k$ and $-\delta\kappa\, \Delta t\, \nabla\cdot\mathbf{v}^k$, a time-stepping function needs both the perturbation arrays ($\delta\mathbf{v}$ and $\delta\kappa$) and the reference arrays (κ and $\mathbf{v}$), which are contained in different `RDOM` objects. Thus, we change the signature of time-stepping functions to

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

so that we are able to use the same time-stepping functions in both the forward and Born simulation. Here, `dom`, `rdom` and `cdom` are pointers to the `RDOM` objects that respectively hold the dynamic fields to be updated, the reference and perturbation fields. When these three pointers point to the same `RDOM` object that holds the reference fields, these time-stepping functions work in the same way as the previous ones do.

To maintain the information for both the reference and perturbation states, the `IWaveLinState` class is derived from the class `IWaveState`:

```
class IWaveLinState: public IWaveState {
  private:
    mutable IWAVE linstate;  /* perturbation state */
    mutable IMODEL dmod;
        /* store pointers to the non-dynamic perturbation fields*/
    TSIndex ltsi;
       /* time object indicating the current pert-state status */
    ...
  public:
    ...
  };
```

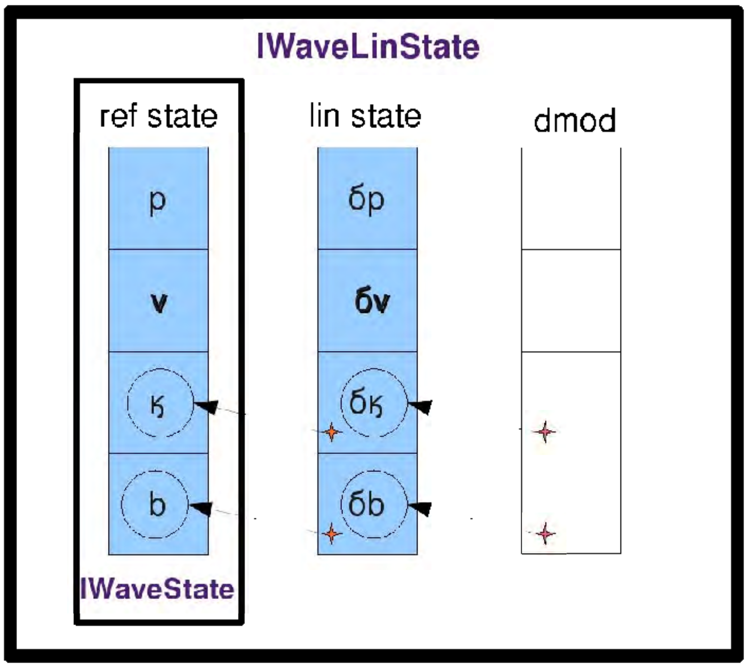

Figure 1: Diagram of `IWaveLinState`: Arrows represent pointer copies. Memory is originated and managed by the ref state `IWaveState::iwstate` and lin state `linstate`. The `dmod` consists of virtual arrays and holds pointers to the data of non-dynamic perturbation fields. No memory leaks created.

The reference state, contained in the base class `IWaveState`, is used to supply non-dynamic fields (e.g., κ) for updating the perturbation state (e.g., $-\kappa\Delta t\,\nabla\cdot\delta\mathbf{v}^k$) and dynamic fields (e.g., $\mathbf{v}^k$) for computing the born source (e.g., $-\delta\kappa\,\Delta t\,\nabla\cdot\mathbf{v}^k$), whose non-dynamic fields (e.g., $\delta\kappa$) is supplied by an `IMODEL` object `dmod`. This is accomplished by copying the pointers to the data for non-dynamic fields from the reference state *domain* to the perturbation state *domain*. The pointers to non-dynamic fields originally supplied in the perturbation state *domain* are assigned to an `IMODEL` object `dmod`. Figure 1 shows this idea.

We use `Sim` classes from TSOpt to implement the time loops. The working flow of each time loop looks like:

- step: the calls to run time-stepping functions to update the perturbation fields and exchange information among processors in an appropriate order;
- post-step:
  - the calls to insert born source appropriately;
  - the calls to record and sample the perturbation fields in traces;
  - update the lin-state time;
  - sync-step: march the forward simulation one step further to catch up with the linearized simulation.

NUMERICAL EXAMPLES

In this section, I will demonstrate the correct behavior of the Born simulator via three numerical examples.

Example I

This example is done for a simple model shown in Figure 2, which consists of a homogeneous background ($\kappa = 11109$ MPa, $\rho = 2100 \quad kg/m^3 \Rightarrow$ acoustic velocity $c = 2.3 \quad km/s$) and a line perturbation ($\delta\kappa = 2641$ Mpa, $\delta\rho = 100 \quad kg/m^3 \Rightarrow \delta c = 0.2 \quad km/s$). A point source is located at the position $(3000, 40)\, m$, and receivers are placed at positions $(3100 + i * 10, 80)\ m$ for $i = 0, \ldots, 99$.

Figure 3 shows the full seismogram computed via IWAVE, which contains primary and multiple reflections; Figure 4 shows the first order pressure perturbation computed via the Born simulator; Figure 5 shows the difference between $DF[m]\delta m$ and $\left(F(m + 0.1\delta m) - F(m - 0.1\delta m)\right)/0.2$. Table 1 shows the results from a derivative test, which compares $\left(F(m + h\delta m) - F(m - h\delta m)\right)/(2h)$ with $DF[m]\delta m$.

As we can see, the first-order perturbation is correctly achieved by the Born simulator. And the derivative test demonstrates that the action of the Born map on vector δm does yield the directional derivative of the forward map $F[m]$ along δm.

h	norm of difference	relative error	convergence rate
1	150.12201	0.063421488	——
0.9	120.2859	0.050816741	2.1030352
0.8	94.127052	0.039765507	2.0820141
0.7	71.458405	0.030188767	2.0634089
0.6	52.117603	0.022017932	2.0474308
0.5	35.970844	0.01519647	2.0337362
0.4	22.905241	0.0096766921	2.0226574
0.3	12.833994	0.00542193	2.0135708
0.2	5.6885972	0.0024032407	2.0066679
0.1	1.4168549	0.00059857342	2.0053811

Table 1: norm of difference = $\left\|\frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m\right\|$, relative error = $\left\|\frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m\right\| / \|DF[m]\delta m\|$

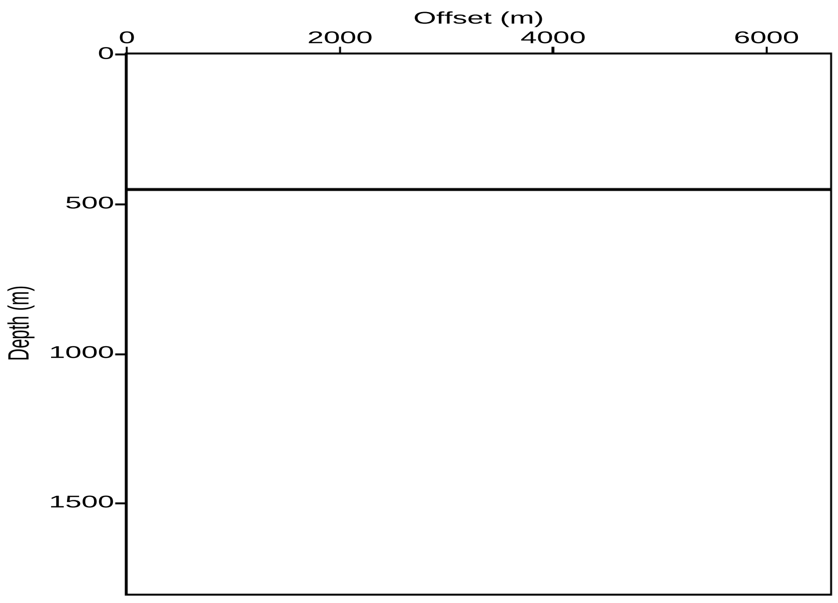

Figure 2: Homogeneous Model with Line Perturbation: homogeneous background $\kappa = 11109$ MPa, $\rho = 2100 \;\; kg/m^3 \Rightarrow$ acoustic velocity $c = 2.3 \;\; km/s$; line perturbation $\delta\kappa = 2641$ Mpa, $\delta\rho = 100 \;\; kg/m^3 \Rightarrow \delta c = 0.2 \;\; km/s$

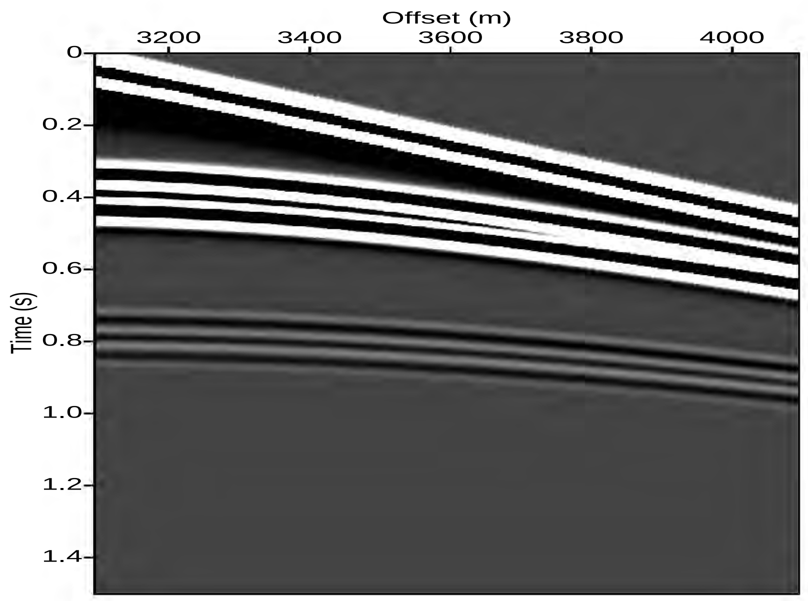

Figure 3: Full Seismogram: primary reflections around 0.4s; multiple reflections around 0.8 s, 1.2 s, etc..

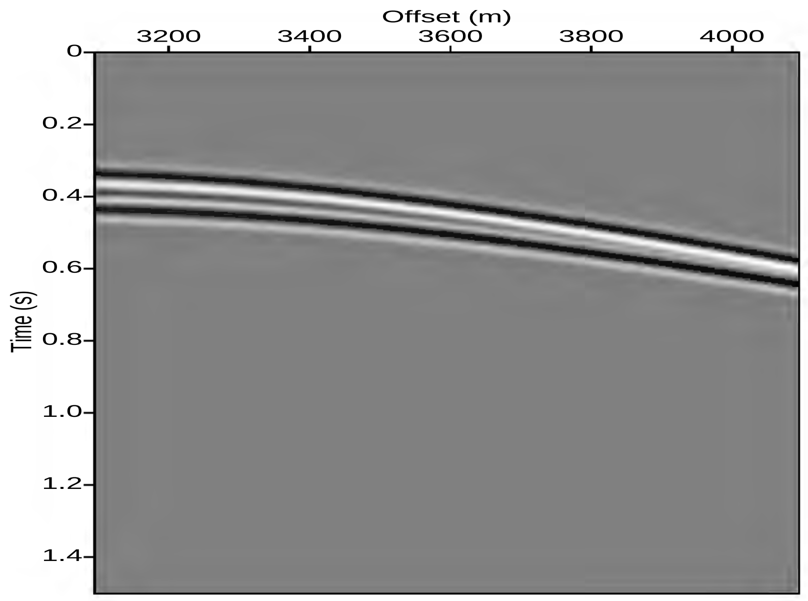

Figure 4: First Order Pressure Perturbation: only primary reflections around 0.4 s.

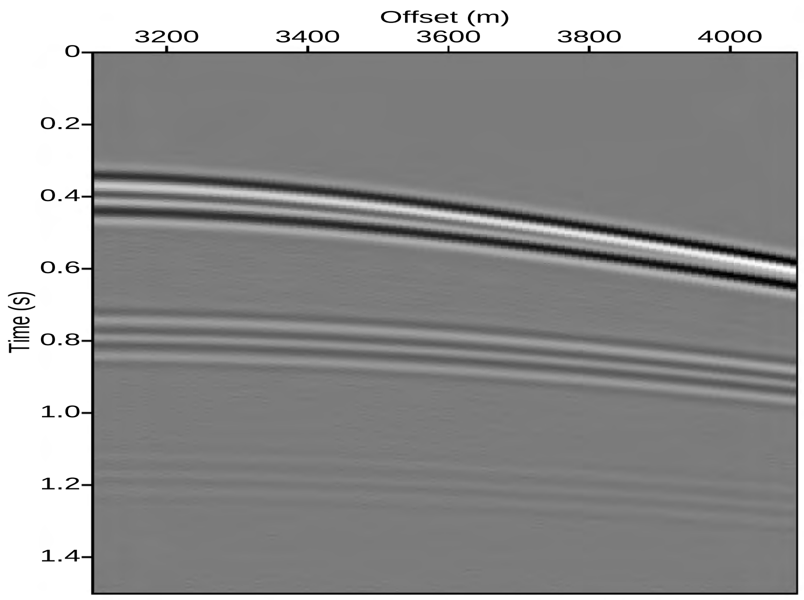

Figure 5: Difference $\frac{F(m+0.1\delta m)-F(m-0.1\delta m)}{0.2} - DF[m]\delta m$

Example II

This example is done for a layered model shown in Figure 6, which consists of a homogeneous background ($\kappa = 11109$ MPa, $\rho = 2100 \quad kg/m^3 \Rightarrow$ acoustic velocity $c = 2.3 \quad km/s$) and a block perturbation ($\delta\kappa = 2641$ Mpa, $\delta\rho = 100 \quad kg/m^3 \Rightarrow \delta c = 0.2 \quad km/s$). A point source and receivers are located at the same locations as those in the first example.

Figure 7 shows the full seismogram computed via IWAVE; Figure 8 shows the corresponding first order pressure perturbation computed via the Born simulator; Figure 9 presents the difference between $\left(F(m+0.01\delta m) - F(m-0.01\delta m)\right)/0.02$ and $DF[m]\delta m$. Table 2 shows the results from a derivative test.

Obviously, the first-order perturbation is correctly achieved by the Born simulator. And the derivative test demonstrates that the action of the Born map on vector δm does yield the directional derivative of the forward map $F[m]$ along δm.

h	norm of difference	relative error	convergence rate
0.1	110.67387	0.02450726	——
0.09	89.866638	0.019899776	1.9766518
0.08	71.166061	0.015758781	1.9808515
0.07	54.597832	0.012089966	1.9847171
0.06	40.183228	0.0088980431	1.9886029
0.05	27.947681	0.0061886436	1.9916205
0.04	17.913837	0.0039667818	1.9931601
0.03	10.074574	0.0022308808	2.0006759
0.02	4.5133157	0.0009994139	1.9803987
0.01	1.6904721	0.00037433265	1.4167614

Table 2: norm of difference $= \left\| \frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m \right\|$, relative error $= \left\| \frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m \right\| / \|DF[m]\delta m\|$

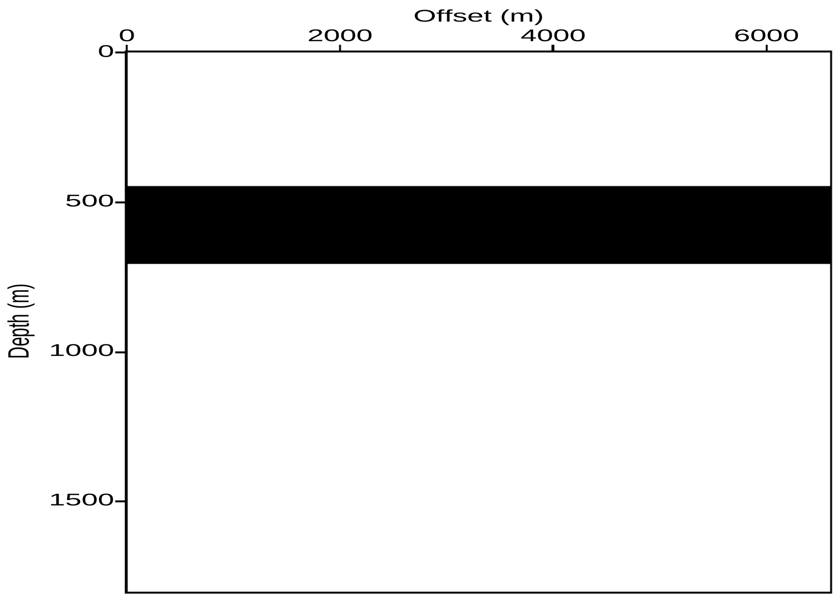

Figure 6: Homogeneous Model with Block Perturbation: homogeneous background $\kappa = 11109$ MPa, $\rho = 2100 \;\; kg/m^3 \Rightarrow$ acoustic velocity $c = 2.3 \;\; km/s$; line perturbation $\delta\kappa = 2641$ Mpa, $\delta\rho = 100 \;\; kg/m^3 \Rightarrow \delta c = 0.2 \;\; km/s$

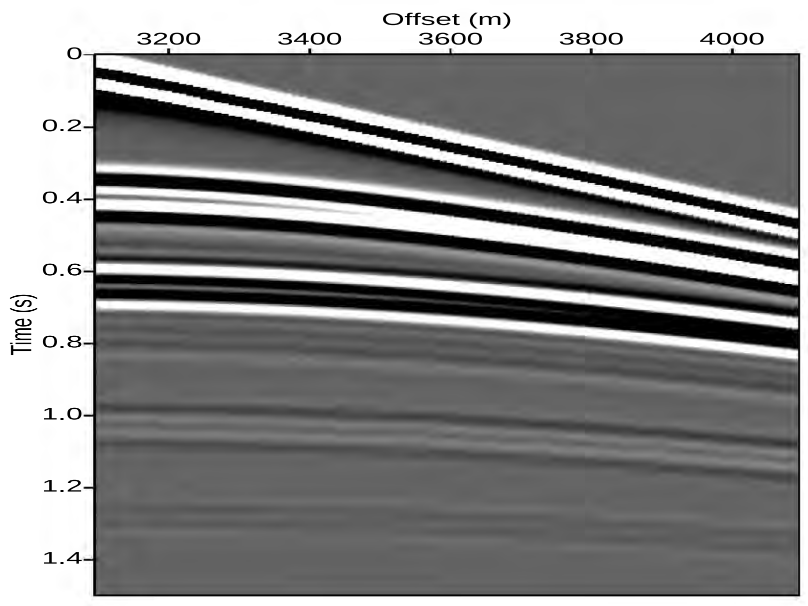

Figure 7: Full Seismogram: primary reflections around 0.4s, 0.6s; multiple reflections around 0.8 s, 1.0s, 1.2 s, 1.4s, etc..

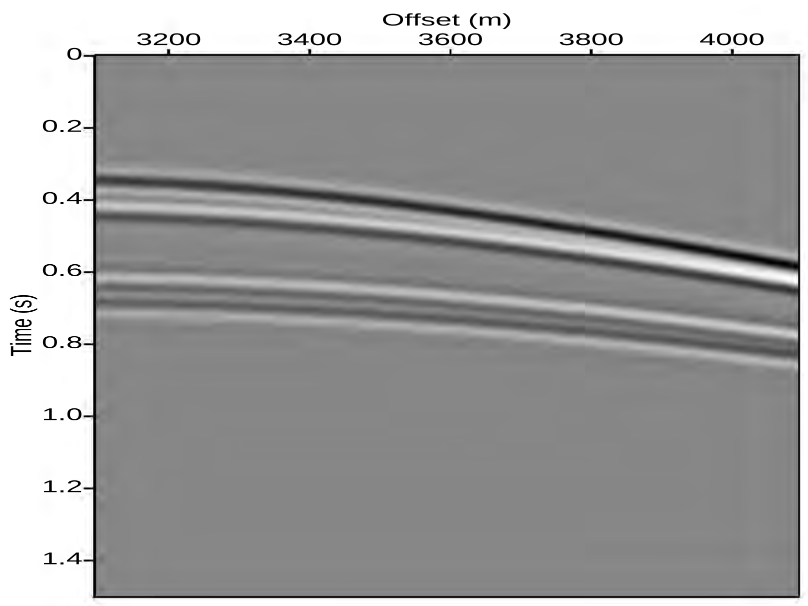

Figure 8: First Order Pressure Perturbation: only primary reflections around 0.4 s, 0.61s.

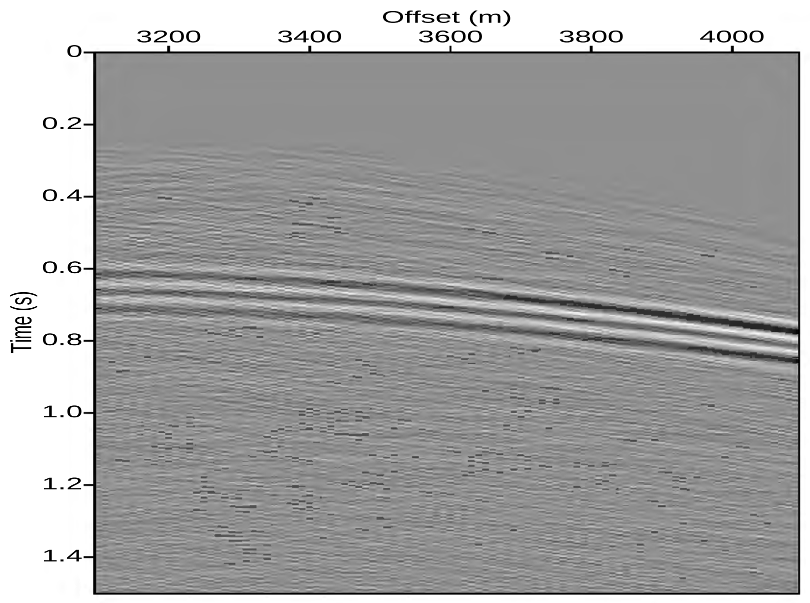

Figure 9: Difference $\frac{F(m+0.01\delta m)-F(m-0.01\delta m)}{0.02} - DF[m]\delta m$

Example III

This example uses a similar model to the one in the previous example, which consists of the same homogeneous background and a block perturbation at the same location but with much bigger perturbation value ($\delta\kappa = 12849$ Mpa, $\delta\rho = 100 \quad kg/m^3 \Rightarrow \delta c = 1.0 \quad km/s$). All settings are the same as those in the previous examples.

Figure 7 shows the full seismogram computed via IWAVE; Figure 11 shows the corresponding first order pressure perturbation computed via the Born simulator; Figure 12 shows the difference between $(F(m+0.01\delta m) - F(m-0.01\delta m))/0.02$ and $DF[m]\delta m$. Table 3 shows the results from a derivative test.

Note that the Born simulator works as expected. The first-order perturbation is correctly achieved; and the derivative test demonstrates that the action of the Born map on vector δm does yield the directional derivative of the forward map $F[m]$ along δm. But for this model, the perturbation is not small with respect to the homogeneous background, and does affect the travel-time so that we do see obvious phase-shitting besides multiple reflections, comparing Figure 10 and Figure 11.

h	norm of difference	relative error	convergence rate
0.1	9701.1377	0.4383547	——-
0.09	8310.4307	0.37551433	1.468593
0.08	6906.1387	0.31206012	1.5715401
0.07	5529.8013	0.24986908	1.6644682
0.06	4224.4155	0.19088404	1.7468039
0.05	3032.552	0.1370286	1.8180863
0.04	1994.4301	0.090120129	1.8779219
0.03	1145.9557	0.051781043	1.9261518
0.02	517.10992	0.023366081	1.9625334
0.01	130.47507	0.0058956342	1.9866968

Table 3: norm of difference = $\left\|\frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m\right\|$, relative error = $\left\|\frac{F(m+h\delta m)-F(m-h\delta m)}{2h} - DF[m]\delta m\right\| / \|DF[m]\delta m\|$

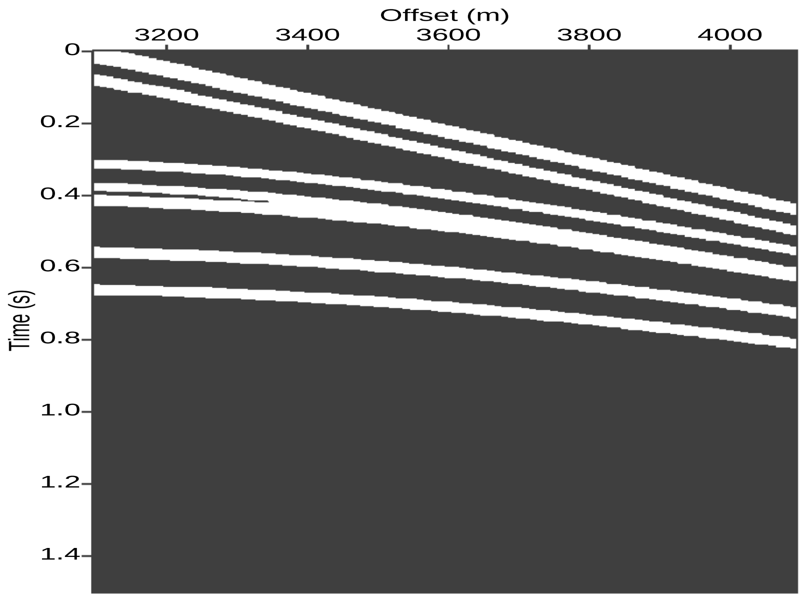

Figure 10: Full Seismogram (from 3100 m to 4100 m): point-source located at (3000,40), receivers placed at depth 80 m; primary reflections around 0.4s, 0.55s; multiple reflections around 0.8 s, 0.95s, 1.2 s, etc..

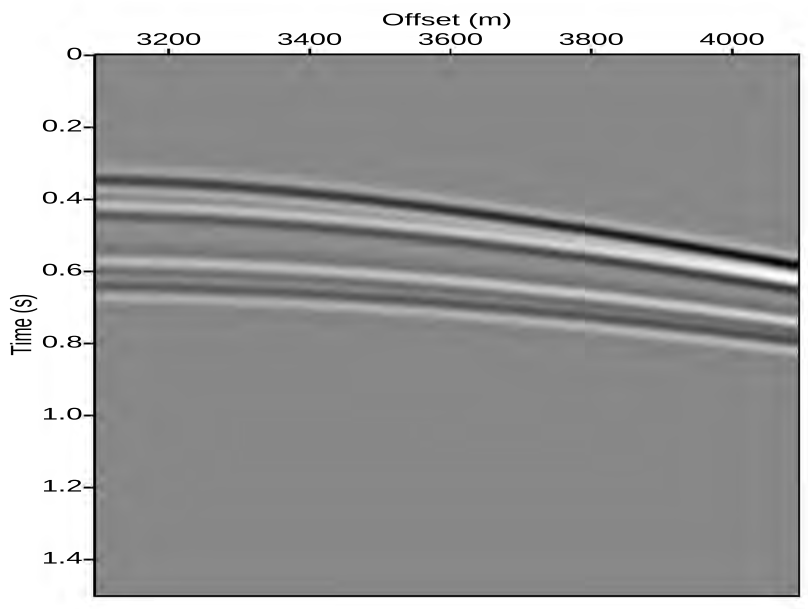

Figure 11: First Order Pressure Perturbation at receivers (from 3100 m to 4100 m): only primary reflections around 0.4 s, 0.61s.

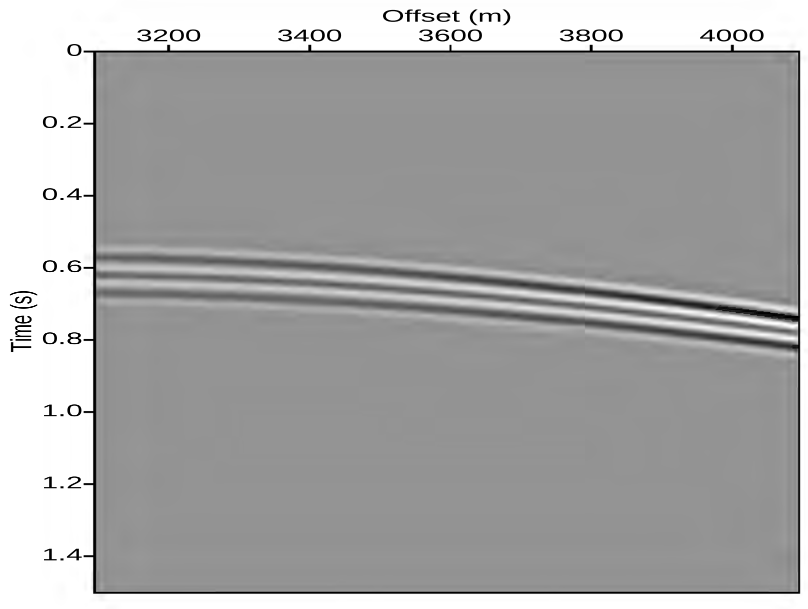

Figure 12: Difference $\frac{F(m+0.01\delta m)-F(m-0.01\delta m)}{0.02} - DF[m]\delta m$

SUMMARY

In this report, I present a way to implement the Born simulation based on IWAVE. The implementation achieves the desired results. More importantly, we have wrapped the modeling package IWAVE as an RVL operator class, which regards the Born simulator as one of its three basic methods. The next report will focus on implementing another basic method, i.e., the adjoint action of Born map. After that, it becomes much more straightforward to embed such a RVL operator into a general optimization framework for inversion.

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

Enriquez, M. and Symes, W. W., 2009, An overview of time-stepping classes for optimization (tsopt): Technical Report TR09-33, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Padula, A. D., Symes, W. W., and Scott, S. D., 2009, A software framework for the abstract expression of coordinate-free linear algebra and optimization algorithms: ACM Transactions on Mathematical Software, **36**, 8:1–8:36.

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.