Geophysical Prospecting (200?) **XX,** 000–000

From Modeling to Inversion: Designing a Well-Adapted Simulator

William W. Symes [⋆], Dong Sun [†], and Marco Enriquez [‡]

Accepted 200? ?? ??. Received 2007 Januari ??; in original form 2007 Januari ??

SUMMARY

This paper describes a few mild design constraints which permit rapid adaptation of modeling code for linear wave propagation to imaging/inversion or design optimization applications, retaining parallelism and other performance enhancements of the underlying simulator. It also describes an abstract software framework preserving the modularity of both optimization and modeling software in building inversion applications, and illustrates this possibility via a an example framework implemented in C++. Wave inverse problems tend to be afflicted by a variety of features, including extreme ill-conditioning and nonlinearity, which degrade the performance of optimization formulations. Extended modeling variants of least-squares inversion, motivated by migration velocity analysis, may relieve some of these difficulties. The framework described also accommodates these extensions to standard inversion.

1 INTRODUCTION

Seismic inversion extracts information about the subsurface from surface or downhole data by adjusting model parameters to predict this data via numerical simulation. The least-squares approach to inversion, now widely known as Full Waveform Inversion ("FWI"), seeks to minimize the RMS difference between predicted and observed data. The simplest mathematical statement of this least-squares approach (ignoring regularization, data and model representation details, and so on) is:

$$\min_{m \in \mathcal{M}} J[m] := \frac{1}{2}\|F[m] - d\|^2, \tag{1}$$

where the modeling or prediction operator $F : \mathcal{M} \rightarrow \mathcal{D}$ maps the model space $\mathcal{M}$ - a set of possible models of Earth structure - to the data space $\mathcal{D}$, a Hilbert space of possible data sets with norm (error measure) $\|.\|$. First proposed by (Tarantola(1984)) and others in the 80's, this approach to seismic data processing has recently become feasible in 3D at field scale, and under favorable circumstances produces economically important information not obtainable in other ways (Barkved *et al.*(2010)).

Computational efficiency suggests a gradient-descent approach to the optimization problem (1): that is, update the model m by computing the objective gradient

$$\nabla J[m] = DF[m]^T \left(F[m] - d\right), \tag{2}$$

searching for a new m in a descent direction related to this gradient, at which $J[m]$ decreases substantially. These updates are repeated until the objective is decreased sufficiently, or no further improvment occurs, or some other stopping criterion is satisfied. In equation (2), $DF[m]^T$ denotes the adjoint or transpose of the Jacobian $DF[m]$, representing the derivative of F at point m.

Both modeling (implementation of F above) and optimization software may be quite complex, and developed by different groups of experts. Consequently, production of inversion applications may require extensive modification of both types of input codes. Our goal in the following pages is to suggest approaches to design of modeling and optimization software that can considerably ease the task of combining them to produce inversion applications. We will offer a proof of our approach, in the form of an inversion package that links virtually unmodified modeling and optimization packages, via a generic middleware package and a minimal amount of additional code. The necessary constraints on modeling code are minimal, and flow naturally from the typical form of methods for solution of partial differential equation systems. The design principles to which optimization and middleware layers must adhere are equally natural.

In the remainder of this introduction, we will explain further the structure of the software design problem for inversion, and our approach to its solution, and overview the organization of the paper.

Evaluation of the prediction operator F involves solution of one or more partial differential equations or systems, discretized via a finite difference or finite element or pseudospectral (or some other) method. So do the applications of $DF[m]$ and $DF[m]^T$. These calculations are very large in scale for 3D (and even 2D) models, so a variety of high performance computing techniques are commonly employed in their implementation, such as parallelization via domain decomposition and MPI, multithreaded execution using OpenMP (on multicore CPUs) or CUDA or OpenCL (on GPUPUs) and other specialized techniques which must be regarded as enabling technology. The algorithms embodied in simulation code also include a variety of sophisticated enhancements such as high-order schemes, regridding, pseudoanalytic timestepping, and complex boundary conditions ((Moczo *et al.*(2006)) for example). Simulators thus involve a considerable investment of time and programming effort. Finally, modeling choices considered reasonable today (typically acoustics or pseudoacoustic TI systems) will likely give way in the future to other systems incorporating more complete wave physics. This evolution will have consequences for all aspects of simulator implementation. For example, methods currently in vogue for reconstructing source wavefields in the adjoint state implementation of $DF[m]^T$ rely on the time-reversible nature of the wavefield, which no longer obtains for models with appropriate physical attenuation.

Optimization algorithms on the other hand involve generic linear algebra operations such as linear combination or inner product, in addition to interaction with the modeling packages just described. These algorithms also gain complexity as their effectiveness increases, especially with regard to the incorporation of model constraints (see (Nocedal & Wright(1999)) for a good overview). Thus some method for protecting investment in effective optimization algorithms is also appealing.

This paper describes three interlinked developments, which together open an avenue to straightforward incorporation of sophisticated modeling techniques in inversion software.

First, we explain an approach to generation of modeling algorithms (for F) which facilitates the implementation of algorithms for the auxiliary Born and adjoint Born maps (DF, DF^T) essential for inversion. This approach involves minor constraints on the expression of common finite difference and finite element algorithms, in return for which the auxiliary algorithms are nearly automatic, and inherit the implementation features (parallelism, absorbing boundary conditions,...) of the base algorithm.

Second, optimization, linear algebra, and other numerical code can be written to avoid dependence on details of data representation or simulator construction, hence apply without alteration to a wide range of data fitting and similar scientific computing problems. The utility of this approach for scientific programming has been explored by many authors over the last two decades (Balay *et al.*(2001); Heroux *et al.*(2003); Kolda & Pawlowski(2003); Benson *et al.*(2007); Padula *et al.*(2009)). Either implicitly or explicitly, this approach involves so-called object-oriented programming, which centers around the definition and use of domain-adapted data types. This approach to programming is ubiquitous in the commercial software world. While the principles of object-oriented design can be implemented using any computer language, we prefer to work in a language environment explicitly supporting this programming style. Our group has developed an object-oriented abstract numerics library, the Rice Vector Library ("RVL"), written in C++, which serves as the foundation for our inversion software. RVL has be described extensively elsewhere (Padula *et al.*(2009)); we give a brief overview below. Our intent is not to focus on this particular library - alternatives exist, and more will appear - but to elucidate the generic role of this type of library in effective software engineering for inversion.

Finally, linking the specific data structures of a simulator to an abstract numerics library like RVL requires a "middleware" translation layer. Since we focus on the time domain in our work, the middleware layer expresses the formation of abstract numerical objects (operators, functions, model/data spaces...) from timestepping algorithms. Since the auxiliary (linearized, adjoint state) computations figure in aspects of the abstract objects (operator derivatives and their adjoints), this layer is a natural home for algorithms such as various methods for source wavefield reconstruction, which then apply *ipso facto* to all timestepping algorithms. We have implemented an middleware layer, the Timestepping Library for Optimization ("TSOpt", (Enriquez & Symes(2009))), and give a brief description. Again, our goal is not particularly an exposition of TSOpt, but rather an explanation of the generic role it plays in converting simulation code into inversion software via linkage with abstract numerical algorithms.

In the following sections, we will describe the simulation of wavefields, in a time-discrete form that facilitates description of the Born and adjoint simulations required for inversion algorithms. While we discuss only first-order perturbation computations, we note that very similar techniques may be used to compute second derivatives as are required in full-blown Newton method implementations (Symes & Santosa(1988)). Certain simple and commonplace properties of the basic evolution scheme suggest implementation using functions that can be reused in Born and adjoint schemes. The next section describes abstraction of these several simulators at the loop level, and various solutions to the well-known source field storage problem for adjoint simulation, including the *optimal checkpointing* algorithm, that are are naturally implemented in terms of these loop abstractions. Implicitly, these schemes define the forward map F and its associated constructs. We explain how these relations might be made explicit by providing abstract definitions for the vector calculus attributes of a nonlinear function or operator, and building these computational objects out of simulation loops. The vector calculus layer allows us to formulate optimization and linear algebra algorithms, thus completing the path from difference stencils to inversion.

To demonstrate that an implementation is actually possible, we present an inversion framework following the principles outlined in the preceding sections. This software package, IWAVE++ (Sun & Symes(2010a); Sun & Symes(2010b)), builds on the IWAVE simulator package which we developed as a benchmarking tool for the SEAM project (Fehler(2009)). We were able to use IWAVE, with essentially no modification, as the core of our inversion software, which also uses the aforementioned TSOpt and RVL packages. The underlying finite difference code required virtually no change, and the algorithmic and software features built into the simulator (high order methods, i/o, absorbing boundary conditions, parallelization via domain decomposition) are inherited by the inversion application. We provide a few examples of typical computational results, including the so-called dot product test (which verifies the quality of the $DF[m]^T$ computation) and our recreation of some of the examples from the landmark paper of (Gauthier *et al.*(1986)).

Least-squares inversion, as formulated in (1) for example, has some well-known computational drawbacks, which (Gauthier *et al.*(1986)) were amongst the first to explore. In the final section, we give a brief overview of *extended modeling* algorithms designed to overcome these drawbacks, and show how the computational framework described here can accommodate them.

While we focus on time-domain computations, much recent work on waveform inversion has employed the frequency domain (Pratt(1999); Sirgue & Pratt(2004); Brenders & Pratt(2007a); Brenders & Pratt(2007b); Plessix(2009); Brossier *et al.*(2009); Plessix *et al.*(2010); Barkved *et al.*(2010)). Some aspects of our discussion also apply in that domain, for example the sufficient conditions for reuse of forward modeling kernels in adjoint modeling.

An appendix provides a detailed derivation of the adjoint state method, in the form presented in next section.

2 FROM MODELING STEPS TO LINEARIZED AND ADJOINT MODELING STEPS

The formula (2) reveals that the adjoint operator $DF[m]^T$ is a key ingredient in the gradient computation (and the gradient is in turn a key ingredient in any Newton-related optimization method). The adjoint operator also has some value in itself: its output is *reverse-time migration.* The linearized or *Born* operator $DF[m]$ is also useful for some optimization approaches (notably the so-called Gauss-Newton algorithm or its more sophisticated Newton-Krylov variants (Akcelik *et al.*(2003))), and is also invaluable in quality control of the adjoint operator. A modeling package, on the other hand, computes $F[m]$. The purpose of this section is to lay out mild restrictions under which the components of a (time-domain) computation of $F[m]$ can be reused in computing the actions of $DF[m]$ and $DF[m]^T$.

We will use $\mathbf{u}$ to denote the state vector of a generic discretized evolution system, deriving from a system of partial differential equations encapsulating a model of seismic wave propagation. The precise method of discretization is immaterial - finite difference, conforming or discontinuous Galerkin, pseudospectral,... - the reasoning which follows applies in any case. The evolution is also discrete in time: $\mathbf{u}^n$ holds the dynamical wavefields at one time or several related times around $n\Delta t$, e.g., the displacement field in elasticity or the pressure in acoustics at two adjacent time levels, or stress and particle velocity at two time levels. For convenience, we will imagine that the discrete times are uniformly spaced, with step Δt, as is commonplace in seismic simulation. This constraint is really not necessary, and in fact we have been careful not to build it into our software in any essential way, but it does make the exposition simpler, so we adopt it for the purposes of this paper.

The evolution (time step) operator of the system depends on a vector $\mathbf{m}$ of model or *control* parameters. which represents material parameter fields describing of Earth structure, e.g., the p-wave velocity or density or shear modulus or ... Seismic simulation generally takes place on time scales over which the material parameter fields are constant in time (autonomous). Thus the control vector is independent of time. We will sometimes refer to the state vector as containing the *dynamical fields*, and the control vector as containing the *static fields*, of the evolution problem.

The evolution operator H relating the state $\mathbf{u}$ at $n\Delta t$ times to the state $\mathbf{u}^{n+1}$ at $(n+1)\Delta t$ is linear, and depends on the parameter vector $\mathbf{m}$:

$$\mathbf{u}^{n+1} = H[\mathbf{m}, \mathbf{u}^n] + \mathbf{f}^n, n = 0, 1, \ldots, N-1 \tag{3}$$

in which the “source” term $\mathbf{f}^n$ represents time-varying external energy input to the system.

It’s quite common for H to consist of the composition of one or more *sub-steps* $H_0, ..., H_k$ and a parameter-independent “cleanup” step W:

$$H[\mathbf{m}, \mathbf{u}] = WH_k[\mathbf{m}, H_{k-1}[\mathbf{m}, \cdots H_0[\mathbf{m}, \mathbf{u}] \ldots]]. \tag{4}$$

This structure is familiar for staggered-grid finite difference schemes (Virieux(1984); Virieux(1986); Levander(1988a)) for instance, in which H_0 updates pressure (for acoustics) or stress (for elasticity), and H_1 updates particle velocity. Many other schemes conform to this structure, however. For example, the standard second-order pressure scheme in constant-density acoustics,

$$p^{n+1} = 2p^n - p^{n-1} + v^2\Delta t^2 Lp^n \tag{5}$$

with L denoting an approximate Laplace operator, and v the sound velocity field, may be rewritten in terms of the vector

$$\mathbf{u}^n = (p^n, a^n)^T,\ a^n = (p^n - p^{n-1})/\Delta t.$$

The second component is proportional to the acceleration at the half time step $(n-\frac{1}{2})\Delta t$. The three-level evolution (5) is equivalent to the two-level scheme

$$a^{n+1} = a^n + v^2\Delta t L p^n \tag{6}$$

$$p^{n+1} = p^n + \Delta t a^{n+1} \tag{7}$$

, which is in turn equivalent to the composite step $H = WH_0$ for the state vector $\mathbf{u} = (p,a)^T$ and $\mathbf{m}$ related to v:

$$H_0[\mathbf{m},\mathbf{u}] = \begin{pmatrix} I & 0 \\ v^2\Delta t L & I \end{pmatrix}\begin{pmatrix} p \\ a \end{pmatrix},\ W = \begin{pmatrix} I & \Delta t I \\ 0 & I \end{pmatrix}. \tag{8}$$

In each row of the matrices above, I denotes the identity, 0 the zero operator - that is, these are block matrices, each entry of which is itself an operator acting on one of the discrete fields comprising the state vector.

The system representation (8) of constant density acoustics exhibits another common feature: the sub-step operator(s) depending on the material parameters are *affine* in a suitably chosen parameters, and the constant term is the identiy operator. In (8) the appropriate parameter is the squared velocity v^2, that is, $\mathbf{m} = (v^2)$:

$$H_0[\mathbf{m},\mathbf{u}] = \left(\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} + v^2\begin{pmatrix} 0 & 0 \\ \Delta t L & 0 \end{pmatrix}\right)\begin{pmatrix} p \\ a \end{pmatrix}$$

In the case of staggered grid schemes (either standard or rotated, see (Moczo *et al.*(2006))), the appropriate parameters are bouyancy (reciprocal density) and the Hooke tensor, or a set of parameters derived from it in cases of additional symmetry.

We will assume that the sub-step operators $H_0, ..., H_k$ are each affine in some set of material parameter fields. Note that this may require a different choice of material parameter fields than is natural or normal for description of the model - certainly squared velocity is not a particularly natural parameter (although, after multiplication by density, it does become the bulk modulus). Nonetheless, it is in squared velocity, rather than velocity, that the component operator H_0 in (8) is affine. Similarly, the inertial property of rock is usually denominated in mass per unit volume, rather than volume per unit mass, yet the latter is the "affine" choice of parameter for staggered grid velocity-stress schemes.

Assuming from now on that $\mathbf{m}$ is a vector of parameters in which the sub-step operators are affine, we write

$$H_j[\mathbf{m}, \mathbf{u}] = \mathbf{u} + L_j[\mathbf{m}, \mathbf{u}] \tag{9}$$

in which $L_j, j = 0, ...k$ are $k + 1$ *bilinear state-vector valued* operators. That is, $L_j[\mathbf{m}, \mathbf{u}]$ is linear in $\mathbf{m}$ and $\mathbf{u}$ separately.

Because of the form of (9), each sub-step is an *increment*, which we indicate by the increment symbol $+=$ (following common practice in C). We presume that the scheme is implemented in *overwrite* form, that is each field is updated in place so that at time step n, $\mathbf{u}$ holds the field that we have denoted as $\mathbf{u}^n$, and each step in the algorithm updates it - thus the symbols $=$ and $+=$ in the following algorithm descriptions indicate *assignment* of the right-hand side to the left. If the algorithm is to be organized in this overwrite mode, then the state vector $\mathbf{u}$ must have the current simulation time as one of its attributes. Since the state vector represents the system state at a particular time only in the corresponding step of (3), a *sampling operator* S needs to be supplied. This operator extracts partial information from the state and records it in a time-dependent data structure, consisting of data traces, movie frames, and perhaps other quantities (such as vorticity extracted from the velocity field, for example). The time information stored in the state is used to determine the output of S and the way in which it is stored in the output data objects. For convenience only, we assume that the sampling operator is linear. Its output increments the *data* vector $\mathbf{d}$. Finally, a similar *source insertion* operator R is required to increment the state vector by the source field ($\mathbf{f}$ in (3)). While the usual notation does not suggest it, we can assume that this operator acts only on the current time level of the state vector.

With these conventions, the discrete evolution (3) takes the form

```
SIM (1) $\mathbf{u} = 0$ ;
SIM (2) For $n = 0, \dots, N-1$ do:
    SIM (2.1) For $j = 0, \dots, k$ do: $\mathbf{u} \mathrel{+}= L_j[\mathbf{m}, \mathbf{u}]$;
    SIM (2.2) $\mathbf{u} = W\mathbf{u}$
    SIM (2.3) $\mathbf{u} \mathrel{+}= R\mathbf{f}$
    SIM (2.4) $\mathbf{d} \mathrel{+}= S\mathbf{u}$
```

Born modeling or linearized simulation refers to the system for the first-order perturbation field $\delta\mathbf{u}$ corresponding to a perturbation $\delta\mathbf{m}$ in the material parameter fields, which follows from the system (3) by implicit differentiation: for $n = 0, 1, \dots, N-1$,

$$\delta\mathbf{u}^{n+1} = D_{\mathbf{u}}H[\mathbf{m}, \mathbf{u}^n]\delta\mathbf{u}^n + D_{\mathbf{m}}H[\mathbf{m}, \mathbf{u}^n]\delta\mathbf{m} \qquad (10)$$

which must proceed synchronously with the simulation algorithm (3). This system can also be written in incremental form, in the same way that the SIM system above implements (3) under the assumption of affine sub-steps. The sampled output produces a perturbational output field $\delta\mathbf{d}$.

```
BORN SIM (1) $\mathbf{u} = 0$, $\delta\mathbf{u} = 0$;
BORN SIM (2) For $n = 0, \dots, N-1$ do:
    BORN SIM (2.1) For $j = 0, \dots, k$ do:
        $\delta\mathbf{u} \mathrel{+}= L_j[\mathbf{m}, \delta\mathbf{u}]$;
        $\delta\mathbf{u} \mathrel{+}= L_j[\delta\mathbf{m}, \mathbf{u}]$;
        $\mathbf{u} \mathrel{+}= L_j[\mathbf{m}, \mathbf{u}]$
```

BORN SIM (2.2) $\mathbf{u} = W\mathbf{u}; \delta\mathbf{u} = W\delta\mathbf{u}$

BORN SIM (2.3) $\mathbf{u} \mathrel{+}= R\mathbf{f}$

BORN SIM (2.3) $\delta\mathbf{d} \mathrel{+}= S\delta\mathbf{u}$

The adjoint state computation associated to the system BORN SIM is a *backwards* evolution for an *adjoint state* $\lambda\mathbf{u}$ and *accumulation* for an output object (or *image volume*) $\lambda\mathbf{m}$. The field $\lambda\mathbf{u}$ is a *Lagrange multiplier* for the system (3), viewed as a constraint on $\mathbf{u}$ - see (Plessix(2006)) for details.

An vector of operators M_j related to $L_j, j = 0, \ldots, k$ is a critical ingredient in the adjoint state computation. The relation depends on inner (dot) products $\langle , \rangle_S$ in the state vector space, and $\langle , \rangle_M$ in the model space. For any state vectors $\mathbf{u}_1, \mathbf{u}_2$ and any material parameter field $\mathbf{m}$,

$$\langle L_j[\mathbf{m}, \mathbf{u}_1], \mathbf{u}_2 \rangle_S = \langle \mathbf{m}, M_j[\mathbf{u}_1, \mathbf{u}_2] \rangle_M. \tag{11}$$

This sounds abstruse, but in concrete cases construction of M_j is typically straightforward. For example, a sub-step in the staggered grid acoustic Born modeling scheme (Virieux(1984)) is

$$\delta p \mathrel{+}= \kappa\nabla \cdot \delta\mathbf{v} + \delta\kappa\nabla \cdot \mathbf{v}.$$

with $\nabla\cdot$ denoting a discretized divergence operator mapping half-cell displaced velocity fields onto the pressure grid. In this system $\mathbf{u} = (p, \mathbf{v})^T$ and $\mathbf{m} = (\kappa, b)^T$ with $b = \frac{1}{\rho}$ being the bouyancy. Thus the preceding equation may be re-written

$$\mathbf{u} \mathrel{+}= L[\mathbf{m}, \delta\mathbf{u}] + L[\delta\mathbf{m}, \mathbf{u}]$$

in which κ acts by pointwise multiplication at gridpoints in

$$L[\mathbf{m}, \mathbf{u}] = (\kappa\nabla \cdot \mathbf{v}, 0)^T.$$

Thus with the obvious choices of Euclidean dot products in model space (pressure grid, half-cell displaced velocity grids), for both the pressure and bulk modulus perturbations, one obtains

$$M[\mathbf{u}_1, \mathbf{u}_2] = ((\nabla \cdot \delta\mathbf{v}_1)p_2, 0)^T.$$

Note that this is actually not new: in this case, $M[\mathbf{u}_1, \mathbf{u}_2] = L[\mathbf{u}_2, \mathbf{u}_1]$ - so implementing L automatically implements M!

The adjoint computation also requires the state-space adjoint L_j^T of L_j,

$$\langle L_j[\mathbf{m}, \mathbf{u}_1], \mathbf{u}_2 \rangle_S = \langle \mathbf{u}_1, L_j^T[\mathbf{m}, \mathbf{u}_2] \rangle_S.$$

This adjoint is equally easy to compute in concrete cases.

Besides the material parameter fields $\mathbf{m}$, input for the adjoint computation is a sample vector $\lambda\mathbf{d}$ of the same type as the output $\mathbf{d}$ of the SIM system.

In update form, the adjoint evolution may be written:

```
ADJ SIM (1)  $\lambda\mathbf{u} = 0$
ADJ SIM (2)  For $n = N, \ldots, 1$ do:
  set time in $\mathbf{u}, \mathbf{d}$ to $n - 1$;
    ADJ SIM (2.1)  $\lambda\mathbf{u} \mathrel{+}= S^T \lambda\mathbf{d}$;
    ADJ SIM (2.2)  For $j = k \ldots, 0$ do:
                         $\lambda\mathbf{m} \mathrel{+}= M_j[\lambda\mathbf{u}, \mathbf{u}]$
                         $\lambda\mathbf{u} \mathrel{+}= L_j^T[\mathbf{m}, \lambda\mathbf{u}]$
```

As in the other two simulation algorithms, the adjoint sampling operator S^T is presumed to act only on the current time level of the adjoint data field $\lambda\mathbf{d}$ Appendix A gives a detailed derivation of the adjoint scheme ADJ SIM.

We note that $\lambda\mathbf{m} = 0$ must be part of the initialization of the algorithm, before any adjoint modeling loops are performed.

Two major observations arise from examination of the schemes SIM, BORN SIM, and ADJ SIM. First , everything hinges on the sub-step operators L_j. Both SIM and BORN SIM boil down to recursions involving the function

$$(\mathbf{u}, \mathbf{w}, \mathbf{m}) \mapsto \mathbf{u} + L_j[\mathbf{m}, \mathbf{w}], \tag{12}$$

for various choices of state vectors $\mathbf{u}$ and $\mathbf{w}$ and material parameter fields $\mathbf{m}$, and $j = 0, \ldots, k$.. With the addition of the adjoints L_j^T and M_j, the adjoint simulation ADJ SIM becomes available. We have already pointed out that functions implementing M_j and L_j may be related, even identical, with appropriate identification of arguments. For self-adjoint systems like linear elasticity and acoustics, it is even the case that $L_j = L^T_{k-j}$, $j = 0, \ldots, k$. In these cases, only the implementation of the *generalized time (sub-)step function* (12) is required to implement all of the components of the modeling, Born modeling, and adjoint modeling algorithms.

Second, the appearance of the reference state vector $\mathbf{u}$ in ADJ SIM implies a well-known access conflict: $\mathbf{u}$ is computed forward in time, whereas $\lambda\mathbf{u}$ is computed backwards. A common solution to this problem (Griewank(2000)) is to store all time steps of the reference field $\mathbf{u}$. In fact, other more efficient solutions are available; we discuss some possibilities in the next section.

3 FROM MODELING STEPS TO LOOPS AND MODELING PACKAGES

The algorithms descriptions in the last section are natural from the mathematical point of view. In this section we begin the process of converting them into descriptions of computational algorithms, by identifying natural computational elements. We will refer to a this collection of software elements as a *modeling package*.

To begin with, since the argument of the evolution operator H (in (3) and related equations) is neither the control vector $\mathbf{m}$, nor the state vector $\mathbf{u}$, but rather the *pair* $(\mathbf{m}, \mathbf{u})$, it seems reasonable to regard the pair as the fundamental data object.

Taking into account this observation, an implementation of SIM require these components:

- `State`: a data structure holding the (state, control) pair vector $(\mathbf{m}, \mathbf{u})$. This structure must include (i) all static (material parameter) fields; (ii) all dynamical fields; (ii) the time level n, and (iv) the sub-step index j, assuming that the evolution is divided into sub-steps.
- `Step`: the collection `Step` of time-step operators $L_0, \ldots, L_k$, or rather functions which apply the generalized time-step operators (12). We will confound `Step` with its application

to the state vector, i.e. regard `Step` as a function that can be called. That is, at time step n and sub-step j, `Step` applies the the operator (12) for index j.

- `Time`: a structure containing start and end times for the simulation, also a function that updates both the time step n and the sub-step j in the `State` data structure: if $j < k$ then $j \mathrel{+}= 1$, else $j = 0$ and $n \mathrel{+}= 1$. We will use `Time` synonymously for this function, and assume that it returns the boolean value `true` for time step indices n within the simulation range $0, \dots, N$, else false.

- `Sample`: To get the (trace or movie frame) data out, we require a computational realization of the sampling operator S, which we will call `Sample`. This function must transfer data between the `State` and an external data structure, which we discuss below. We also give `Sample` the task of adding the source data ($\mathbf{f}^n$ in equation (3)). Both (input) source data and (output) trace data are attributes of the `Sample` object, as we construe it here.

Restated in terms of these pseudo-code components, an implementation of the SIM algorithm looks like

```
Sim<State,Step,Sample,Time>:
while (Time(State))
    Step(State)
    Sample(State)
```

To complete the description of the timestepping algorithm, we must identify methods to initialize the `State`, and to extract output data from the `Sample` operator: that is, the means by which the algorithm communicates with its environment. We will denote by `Model` the data structure containing information required to initialize the static fields in the simulator `State`. `Model` thus encapsulates one or several fields, represented in any convenient fashion (sample arrays on regular grids, vectors on unstructured finite element meshes, splines,...). Whether grouped together in a formal data structure, or residing in unconnected data structures, the sample information forms a logical unit with the geometric (grid, spline node,...)

and physical (units, field identity,...) information, and we abstract this explicit or implicit structure as `Model`.

Looking forward to the linkage to inversion software, we note that `Model` represents only that part of the input simulator data which is part of the solution of the inverse problem, that is, m in (1). Other parameters not participating in m are not included in `Model`. For example, source parameters may be fixed in an inversion for velocities etc., in which case source parameters do not appear in `Model`, but are implicitly regarded as part of the fixed structure of `State`. On the other hand, if the energy source is part of the information sought in the inversion, then parameters describing it must also be included in `Model`.

Similarly, the `Data` type (representing d in (1)) encapsulates all survey geometry and sampling information, along with data samples - this information must exist somewhere in the implementation, so we regard it as part of this object, either explicitly or implicitly. Note that an array of SEGY traces, for example, is a data structure explicitly containing this auxiliary information, and would form a natural basis for a `Data` type.

In terms of these types, we can formally identify the necessary initialization functions:

- `initialize_static(State, Model)` initializes the static fields in the simulator `State` from the information in `Model`;
- `initialize_dynamic(State)` initializes the dynamic fields `State` (typically, to a quiescent initial state, specified *a priori*);
- `initialize_sampler(Sampler, State, Time, Data)` connects the sampler with the dynamic field representation in `State` and with the external data structure `Data`. Since the `Data` object contains time sampling information, this function also initializes the `Time` object.

As stated in the last section, we have assigned the sampler the role of source insertion, and presume that the `Data` object also contains any necessary geometric and other information about the source not included in `State`.

This initialization brings up a fundamental point about the nature of seismic survey simulation: a survey is a collection of experiments, rather than a single trial, so the abstract simulator framework needs to accommodate *multisimulations*. For each simulation, the earth remains the same, so the material parameter (static) fields need only be initialized once. The dynamic fields however should be reinitialized for every simulation. Thus `initialize_static` will be called once per multisimulation, `initialize_dynamic` once per simulation. The `Sample` object must store information about the locations of sources and receivers, and possibly other information about the acquisition geometry. This information likely changes from simulation to simulation, so `initialize_sample` function will be called once per simulation, and (re)calculates the relation between the sampling parameters and state vector. This function has access to the extent of the survey, so it is natural to have it return a boolean, `true` if more simulations are to be performed, `false` otherwise.

Note: as will be explained in the Discussion section, inversion based on *extended modeling* may require that material parameter fields be updated with each simulation.

We can now give a complete description of the modeling algorithm, which runs the `Sim` timestepping loop:

```
initialize_static(State,Model)
while (initialize_sample(Sample, State, Time, Data))
   initialize_dynamic(State)
   run Sim<State,Step,Sample,Time>
```

The Born simulation algorithm (10) defines an evolution of the the state vector perturbation $\delta\mathbf{u}$, but also involves the state vector $\mathbf{u}$ and the perturbed and unperturbed control vectors $\delta\mathbf{m}$ and $\mathbf{m}$. As for the state vector, it's natural to pack this information $(\mathbf{m}, \mathbf{u}, \delta\mathbf{m}, \delta\mathbf{u})$ into a `LinState` object. We elected to construct the `LinState` type so that it includes a `State` - that is, `LinState` contains two pairs $(\mathbf{m}, \mathbf{u})$ and $(\delta\mathbf{m}, \delta\mathbf{u})$. The joint step (for perturbation fields first, then for background fields) defined in SIM and BORN SIM is captured in a `LinStep` function. The sampling operator S applies to $\delta\mathbf{u}$ (BORN SIM (2.3)) but is otherwise the same, assuming that it's linear; the `Data` output type remains the same (assuming linear sampling) and the `Time` "clock" object as well. Supplying other necessary functions with obvious naming conventions, the linearized simulation becomes

```
initialize_lin_static(LinState,LinModel)
while (initialize_sample(Sample, LinState, Data, Time))
   initialize_dynamic(LinState)
   run Sim<LinState,LinStep,Sample,Time>
```

The adjoint state evolution ADJ SIM uses a state object of the same structure as that of the linearized simulation, that is a `LinState` - the adjoint state vector $\lambda\mathbf{u}$ contains precisely the same fields as the Born state $\delta\mathbf{u}$. The adjoint step `AdjStep` implements the two steps in ADJ SIM (2.2). The *input* data for the adjoint simulation has exactly the same structure as the `output` data for the linearized simulation - that is, the `Data` output type for the basic modeling task represents both. As one sees from the ADJ SIM loop structure, the adjoint evolution is backwards in time, so the implementation requires a "clock" object `AdjTime` which runs backwards, but is initialized in the same way as `Time`.

The simulation however must have an additional attribute: it must be able to deliver *random access* to simulation times. Such a random-access simulation object `RASim` cannot simply follow the pattern laid down in SIM, but can be built in one of several ways, which we will review below. Amongst many methods one might choose to speficy the target time step at each time of the adjoint loop ADJ SIM, we prefer a *synchronization* object `AdjSynch` which couples the (backwards-in-time) adjoint time step to the simulator `RASim` - the role of this object (that is, of a suitable function associated to it) is to make sure that the time step at which $\mathbf{u}$ is evaluated in ADJ SIM (2.2) is $n-1$, in the notation of the last section. This object will couple an adjoint time step `AdjStep` with an `RASim` object, so that the two can be used together. An adjoint sample function `AdjSample` (implementing S^T) must also be supplied.

With these conventions, a pseudo-code implementation of ADJ SIM is

```
initialize_adj_statlic(LinState,Model)
construct AdjSynch(AdjStep, RASim<State,Step,Sample,Time>)
while (initialize_sample(AdjSample, LinState, Data, AdjTime))
   initialize_adj_dynamic(LinState)
```

```
while (AdjTime())
    AdjSample(LinState....)
    AdjStep(LinState)
    AdjSynch(LinState)
```

The question remains, how to construct a random-access simulation object `RASim`. Functionally, all methods are the same: they move the state from the current time to a target time. Two obvious methods to update the time index to n are:

- restart the simulation at the initial time $n = 0$ and simulate until time step n;
- run the full simulation (3), storing all time levels $0, \ldots, N$; to access the state at time n, retrieve it from storage.

The first option has a computational cost of $O(N)$ times that of a single simulation, and is excessively expensive relative to other options for all but the smallest simulations. The second option has been used surprisingly often, but requires excessive storage, and indeed impossibly large amounts for 3D problems of typical exploration geophysics dimensions.

For time-reversible problems, the dynamics can simply be run backwards if $n < m$, leading to the most efficient possible algorithms in terms of both floating point operations and storage. If the dynamics are time-reversible in most of the domain, as is the case for acoustics or linear elasticity with absorbing boundary conditions, then a small region around the time-reversible volume can be stored and used to supply boundary conditions for backwards-in-time evolution (Gauthier *et al.*(1986); Dussaud *et al.*(2008)). (Clapp(2009)) proposed an elegant variation on this idea, using a random region in the boundary to cause the field to act diffusively there, thus permitting time-reversal of the dynamics with small error and without any extra storage at all.

For dissipative dynamics, such as viscoelasticity with typical Q_s and Q_p values for sedimentary rocks, time reversal is impossible and methods that use only forward time stepping are required. Various intermediate options to the two mentioned above have been explored, such as storing only some of the time levels and interpolating. Of these, the best performance is obtained from the so-called optimal checkpointing methods (Griewank(1992); Blanch *et al.*(1998); Griewank(2000); Symes(2007)), which save state vectors at a selection of intermediate times ("checkpoints"). To compute the state at an arbitrary time n, the algorithm retrieves from storage the stored checkpoint immediately before n and runs the evolution (3) forward from that time to n. In the context of ADJ SIM, further storage savings are feasible: since times are accessed in reverse order, only the checkpoints needed for times near N need be stored at early stages. When a checkpoint has been used to compute all subsequent times, it can be overwritten with an earlier checkpoint. This further tradeoff of computation and storage permits ADJ SIM to be accomplished in $O(N \log N)$ time steps and $O(\log N)$ storage, and these numbers are provably optimal in a precise sense (Griewank(2000)).

4 COMPONENTS OF OPTIMIZATION ALGORITHMS

Having organized the modeling and related algorithms in the preceding paragraphs, it is now necessary to describe how these algorithms fit together to define the components of an optimization approach to full waveform inversion. Before describing a natural development of this transition layer of software, we explore briefly the nature of these components, and what their nature implies for implementation. We will refer to a software implementation of these components as a *vector calculus* package, since the concepts of vector calculus form the essential mathematical underpinning.

The function value and gradient are the primary ingredients in unconstrained optimization algorithms for smooth objective functions, and are also key components for constrained optimization algorithms such as Sequential Quadratic Programming (Nocedal & Wright(1999)). For nonlinear least squares problems such as least squares full waveform inversion, the fundamental relations (1) and (2) imply that expression of algorithms requires types for vectors, operators (vector-valued functions), and linear operators, such as the Jacobian DF and its transpose DF^T. We refer to the Jacobian as a linear operator, rather than a matrix, because only access to the action on a vector (the residual vector $F[m] - d$

in (2)) is actually required - access to the matrix elements of the Jacobian is not. In fact, DF is a dense matrix of prohibitive size, for problems of industrially relevant dimensions (even in 2D). Time-stepping methods in effect factor DF into a product of extremely sparse factors, which makes its multiplication by a vector vastly cheaper than general matrix multiplication of the same dimensions. Matrices for which only matrix-vector multiplication is provided (rather than access to matrix entries) have come to be called *linear operators*.

These mathematical abstractions can be realized computationally in a vast variety of ways. In all cases, the vectors representing model and data are of central importance, and properly contain more data than merely the samples. In one way or another, either explicitly or implicitly, the additional information (beyond samples) needed to interpret the vector data must be available at those points where it is needed - numerical PDE solvers require grid information, filtering and muting operations require time-grid information, interpolation and sampling operations require position information for sources and receivers, and so on. This is a very familiar concept - seismic disk and tape formats such as SEGY group data samples together with other identifying information, for example.

Note however that none of that additional physical information is proper to the expression of optimization algorithms of gradient-descent type, widely and correctly regarded as the most appropriate for these problems. Consider for instance the *steepest descent method with line search globalization.* Using a simple backtracking line search, this algorithm reads

```
0. choose a maximum step factor $\alpha_{\max} > 0$, a relative reduction tolerance $0 < \beta < 1$ and
a gradient length tolerance $\epsilon > 0$
1. compute $J[m], p = \nabla J[m]$ via (2), $r = p^T p$;
2. if $\sqrt{r} \leq \epsilon$ stop;
3. else set $\alpha = \alpha_{\max}$;
    3.0 set $m_+ = m - \alpha p$.
    3.1 compute $J[m_+]$
    3.2 if $J[m_+] - J[m] > -\alpha\beta r$, $\alpha \leftarrow \alpha/2$, go to 3.0.
    3.3 else set $m = m_+$, go to 1.
```

This extremely simple algorithm is actually usable. Examination reveals a need for four types of computation:

- evaluation of $m \mapsto F[m]$; $m, \delta m \mapsto DF[m]\delta m; m, d \mapsto DF[m]^T d$;
- inner products like $p^T p$;
- linear combinations like $m_+ = m = \alpha p$;
- provision of vector workspace like p, m_+.

At no point in the expression of this algorithm are the *components* (samples) of the various vectors accessed directly, nor are any of the additional geometric or physical attributes necessary for interpretation of the vectors as simulator data. Instead, if we have black-box computations of the F and its relatives, inner product, linear combination, and vector creation (memory management), we can proceed. Obviously the components of m play a role in evaluation of $J[m]$, as does the other non-sample information associated with m. However at the level of the algorithm just described, all of that detail is *hidden*.

Note that constraints such as positivity or physical bounds on material parameters would impose constraints on the model update beyond those described in our simple algorithm. Many common algorithms for constrained optimization (notably those of the *active set* class, such as LBFGS-B (Zhu *et al.*(1997))) have implementations involving explicit manipulation of coordinates, but in fact all of these can be hidden behind projection operators and similar abstract devices.

In order to formulate algorithms that will apply to a variety of models and data types without alteration, the data hiding just mentioned is *essential*. This *data abstraction* can be accomplished in a variety of ways, but all define in some way or another an abstract vector type, providing all of the attributes necessary to express algorithms like the one above, while deferring other attributes (samples, grids, units, coordinates,...) to subtypes, which use them to implement the basic vector attributes.

5 FROM MODELING PACKAGES TO OPTIMIZATION COMPONENTS

.

The penultimate section explained how a modeling package for inversion might be structured internally, and the last described the natural components of gradient-descent optimization algorithms. It is now possible to describe in general terms the necessary structure of the interface between vector calculus and modeling packages.

A conventional approach bases the vector calculus package on a concrete universal data structure with minimal attributes, usually an intrinsic array type. Optimization code implemented in Fortran, C, or Matlab typically entails this approach: the data structure manipulated by such packages is simply the array. This approach presumes that all additional information required for interpretation of such data structures as defining material parameter fields or other modeling components must reside entirely in the modeling package. That is, the modeling data structures `Model`, `Data` effectively define subtypes of a single rigidly defined concrete vector data type.

Inversion applications built on this principle are unnecessarily error-prone, difficult to maintain in the face of component evolution, and very hard to extend to new models and optimization approaches. The exchange of data between the two software layers is entirely a feature of the modeling package, which must be equipped with packing/unpacking functions which are peculiar to each optimization implementation. Thus the two sides of the inversion application interpenetrate and lose modularity. Furthermore, data on the optimization side is completely anonymous - it has lost all identifying features of the fields which it represents. This anonymity is an invitation to egregious errors, and also works against any natural modifications of vector operators, such as scaling of inner products by volume elements, which actually do partake of the modeling-side data attributes. Time-honored hacks around this problem include data exchange via common blocks and `void *` parameters, reverse communication, and straightforward modification of all function interfaces to pass necessary information. None of these hacks alleviate the lack of modularity and extensibility inherent in the design.

[h!]

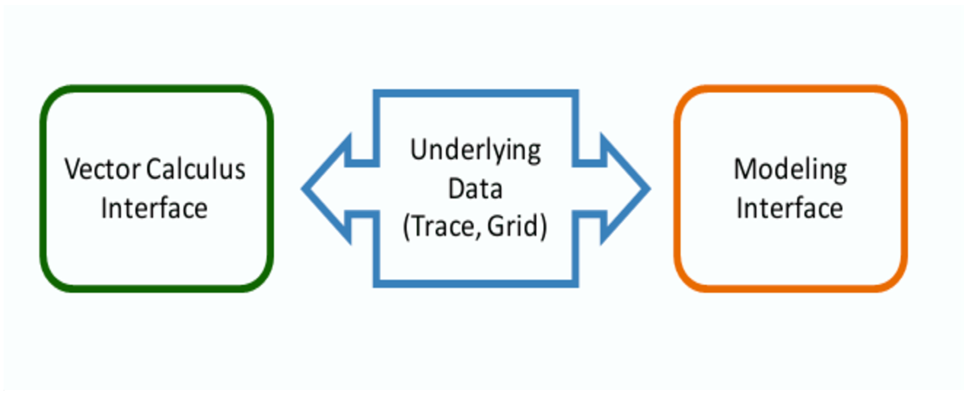

Figure 1. Mechanism of information-sharing between vector calculus, modeling software: each defines an interface to common data structures for material parameter grids, data traces, and other principal data types treated as vectors in the mathematical description of inversion. These dual interfaces allow read and write access (as appropriate) for either package, to the same underlying data.

Vector calculus packages designed around the data abstraction principles explained in the last section allow a much looser coupling between simulator and optimization data types. This *independence of internal representation* is at the heart of our proposal for construction the inversion applications. Given data abstraction, there is no reason whatsoever for the modeling package to employ the same software infrastructure as does the vector calculus package. The sole necessary point of contact between the two is the underlying data of the various vectors. This underlying data must be exposed to both packages and shared in some way; from exposure onwards, the two types of algorithm (modeling, vector operations) can be implemented with complete independence.

This relationship is depicted in Figure 1. The vector calculus object m and the modeling data type `Model` represent the same material parameter fields. This commonality is realized by having both *reference* the same underlying *external* data structure, which need be identical with neither. The same situation obtains with the data vector d and its modeling representation `Data`: both reference a common data structure, which need not be part of either.

The loose coupling between vector calculus, modeling, and external data structures is particularly natural when the external data resides on disk or is distributed over the web. In fact, rewriting procedural algorithms to accommodate such non-incore data forces the introduction of data abstraction, at least to some extent.

Once a connection between data objects is established along the lines depicted in Figure 1, the two sides of the inversion application proceed independently but completely in tandem. For example, provision of vectors m, m_+ and p in the steepest-descent algorithm described above entails (implicitly) the formation of three `Model` objects (which only need to be realized on access to the modeling package). If we indicate the `Model` object associated with m by `Model[`m`]`, and the others similarly, then steps 3.0 and 3.1 of the algorithm are realized as

```
3.0  set $m_+ = m - \alpha p$.
3.1  compute $J[m_+]$:
  3.1.0  compute $F[m_+] = r$:
    initialize_static(State,Model[$m_+$]);
    while (initialize_sample(Sample, State, Time, Data[$r$]))
        initialize_dynamic(State)
        run Sim<State,Step,Sample,Time>
  3.1.1  return $J[m_+] = \frac{1}{2}(r-d)^T(r-d)$.
```

In each transition between the vector calculus and modeling packages (indicated by change of font in the foregoing display), the collaboration passes through the external data, which both reference.

6 PUTTING IT ALL TOGETHER: IWAVE++

The preceding sections have given a complete, if abstract, description of a system of functions and types which can envelope a modeling package in supporting software to create an inversion application. This system is so organized as to imply minimal modifications of the simulator itself, provided that its design adheres to a few basic rules as explained in section 2.

This section reviews the implementation of IWAVE++, a framework that takes advantage of the concepts discussed in previous sections to build a variety of inversion applications.

6.1 Modeling: IWAVE

IWAVE is the name of the modeling package created by our group to assist the SEG Advanced Modeling (SEAM) project in verification of synthetic seismic data. Several main principles guided the design of this package:

- IWAVE is coded entirely in ISO C, following the 1999 standard.
- Nonetheless, the design of IWAVE is highly modular and object-oriented.
- Its target task is time-stepping simulation for regularly gridded fields.
- Its performance is adequate to carry out modeling on the scale envisaged by SEAM, with the maximum accuracy attainable with contemporary finite difference methods. See (Fehler(2009)) for an account of the SEAM verification effort and the requirements which it placed on modeling performance.

The principal designer of IWAVE was Igor Terentyev, then a graduate student in our research group (Terentyev(2009)). He was assisted by Tetyana Vdovina, Xin Wang, and William Symes. The first version (1.0) was released to the public in late 2009 (Terentyev *et al.*(2010)). Several subsequent releases have added functionality, and more releases are planned.

IWAVE separates into two parts, generic and model-specific. The generic part consists of a base package supplying parameter parsing and other utility services, i/o packages for various standard data structures, and a model-independent simulation package, defining structures and functions playing the roles assigned to `State`, `Time`, `Step`, and `Sample` in preceding sections. The generic simulation package declares number of atomic function signatures for which implementations must be provided to define a working application, and uses these to create a time step including finite difference stencil execution and data exchange between processes linked by MPI. Parallelization is achieved by domain decomposition; all information about the parallel framework, including neighbor lists, ghost cell buffers, and local grid definitions, is *computed* from the output of the atomic functions - in other words, parallelization is fully automated, with all calls to MPI implemented in the generic package. The design of the generic package is described in Terentyev's MA thesis (Terentyev(2009)).

The model-specific part of IWAVE contains implementations of the atomic functions mentioned in the last paragraph, and driver (main program) source. The package currently contains a fully functional implementation of acoustics created for the SEAM verification effort; various other elastic and viscoelastic modleling applications are under development. This acoustic modeling package will accommodate 1D, 2D, or 3D models, with either free surface or PML absorbing boundary conditions on any face of the rectangular domain (Hu *et al.*(2007)). The package implements staggered grid (leapfrog) finite difference schemes (Virieux(1984); Virieux(1986); Levander(1988b); Moczo *et al.*(2006)) of order 2 in time and $2k$, $k = 1, \ldots, 7$ in space. It is an extremely conservative implementation: many simple and not-so-simple go-fast tricks, such as expanding computational domains and stretched grids, are not employed, consistent with the original benchmark role intended for the package. It has successfully simulated shots over the SEAM model (in which a single single-precision field occupies 80 GB, and the simulation outputs up to 500,000 traces) using thousands of cores of a large cluster, and small 2D models on laptops, and many models sizes in between.

The acoustics implementation follows the pattern laid out in section 2. The timestep functions, amongst the atomic functions which must be supplied to implement any IWAVE application, are arranged in the form of successive updates, with bulk modulus and bouyancy fields as the linear parameters. Other atomic functions read various combinations of physical quantities from files and convert these to bulk modulus and bouyancy, for user convenience. The timestep function accepts (essentially) the `State` struct as its main argument, which has some consequences for the implementation of the generalized timestep function (12), as we shall explain.

We emphasize two additional properties of IWAVE, embedded in the generic part of the package, that will be important in the sequel:

- IWAVE is by default out-of-core oriented: it is primarily designed to read input data from disk files, and write output data to disk files.

• Job control in IWAVE runs through a parameter table or associative array, typically instantiated from a file containing "key=value" pairs. Amongst the parameters are filenames for the files containing key data structures such as material parameter grids and data traces. The `Model` and `Data` data structures are implicit in these parameters (represented by the contents of the named files) and the `State`, `Sampler`, and `Time` objects are initialized from these. The model-dependent atomic functions mentioned earlier convert these parameters into structural information peculiar to the particular model, and this structure information is used by the generic part of the package to complete the construction of `State` and `Step`.

6.2 Vector Calculus: RVL

The Rice Vector Library, or RVL, is described at length in (Padula *et al.*(2009)). It provides definitions of data types which allow the expression of coordinate-invariant algorithms of based on vector calculus, including many effective algorithms in iterative linear algebra and constrained and unconstrained numerical optimization. Of the several languages available that support user-level definition of types (aka object-oriented programming), we chose C++ for development of RVL, in part because it is a superset of C hence offers immediate access to most high-performance computing libraries such as MPI and CUDA.

C++ realizes types (beyond the built-in types) as *classes*, which group data and functions together and are supported by services which ease initialization and cleanup. The main classes defined in RVL are designed around vector calculus concepts. Their mnemonic names, and principal functions, are:

• Vector: scaling, addition, inner product, set to zero, reference to Space in which it is a member

• Space: creates Vectors (workspace factory), comparison with other spaces,

• Functional - (possibly nonlinear) scalar-valued function: value, gradient

• Operator - (possibly nonlinear) vector valued function: value, derivative (as linear operator)

• LinearOp - linear vector-valued function: value, adjoint

RVL provides many other auxiliary classes and functions which are useful in constructing inversion and other simulation-driven applications. Some of these will be mentioned below. However note that the fundamental classes, listed above, reflect precisely the mathematical objects that figure in the steepest descent algorithm described in section 4 and similar optimization and linear algebra algorithms: these algorithms can be formulated entirely in terms of RVL classes and their attributes.

The classes listed above are *abstract base classes*: they define what objects of various types must do (interface declaration), but not how they do it - that is, implementations are lacking (for the most part). Use of RVL implies construction of a set of *derived classes* implementing subtypes, for which all functions are completely defined.

6.3 Middleware: TSOpt

TSOpt (Enriquez & Symes(2009)) defines base classes for `Time`, `Step` and `Sample` types introduced in section 3, and uses these to define a collection of canonical simulator loops. The `Step` type and the `Sim` loops defined in terms of it depend on a `State` type (via the C++ template mechanism), which is intended to represent the `State` type of section 3. As mentioned there, `State` objects must provide access to time. Any `Sim` is supplied with a mechanism to set a target (final) simulation time, and references a `State`.

Some `Sim`s are unidirectional in time, and will enter an error condition (throw an exception, in C++-ese) if the target time is set earlier than the current time, as recorded in the referenced `State`. Others are bidirectional.

TSOpt provides several implemented random access `Sim` subtypes (`RASim`s, in the diction of section 3). All of these depend on a `Stack` type, for which TSOpt provide a base class - this is an abstraction of the commonplace stack data structure, used to record whatever intermediate `State` data (checkpoints). We also provide several implemented `Stack` subtypes, for use in computationally small settings. For large scale problems, `Stack` implementation may involve a mixture of in-core and out-of-core storage, so part of the effort involved in using TSOpt for such problems lies in creating an appropriate `Stack` subtype.

Finally, TSOpt includes implementations for synchronization objects, such as `AdjSynch` in algorithm REF, which tie together a `Step` object and a `Sim` - this task is best accomplished with an object, rather than a function, as the relation is persistent. TSOpt thus provides ready-to-use implementations of several key ingredients in the adjoint state algorithm.

6.4 Synthesis: IWAVE++

It should be evident at this point that the raw ingredients of an inversion software solution are mostly present in the packages described up to this point (RVL, IWAVE, TSOpt). Several additional steps are required, however, to synthesize these packages into a solution.

One low-level technical detail we have neglected in our conceptual description of IWAVE and TSOpt is that IWAVE is written in C: therefore combinations of structs and functions that play the roles of `State`, `Step`, `Sample`, and so on are not actually C++ classes as expected by TSOpt. Thus we must develop "wrapper" code to generate classes which implement their important member functions to the via calls to IWAVE functions, and use IWAVE structs as data members. For the most part, this is a straightforward process.

The only interesting aspect of this development originates in a conceptual nonconformity: recall that (a) as explained in section 3, the linearized state type should in essence include two (basic) states, each a pair $(\mathbf{m}, \mathbf{u})$ of static and dynamic components. On the other hand, the generalized time step function (12) defined in section 2 takes three arguments - a dynamic variable to be updated, an input dynamic variable, and a static variable. This interface is not in the form indicated for `LinStep` in section 3.

Of the many possible resolutions to this mismatch, we chose one which involves no copying of floating point data and takes direct advantage of the already implemented `State` "virtual type" in IWAVE. The solution is explained in Figure 2 and its caption: it involves copying only of pointers, and allows the maximum re-use of IWAVE code. The modified generalized time step interface takes three IWAVE `State` arguments, rather than two dynamic and one static state variable. In this way, the `State` remains encapsulated until it must be exposed for the actual loops of the time step function. The `IWaveState` and `IWaveLinState` classes so implemented are part of IWAVE++, and have all of the necessary attributes of the TSOpt `State` type.

Other aspects of the IWAVE-TSOpt linkage are straightforward.

Second, we must define the common linkage to external data depicted in Figure 1, through which the modeling package (IWAVE/TSOpt) and the vector calculus package (RVL) will communicate. The data structures to be linked on the modeling side are already defined: these are the model and data components of `IWaveState` and `IWaveLinState`. The choice of external data representation is natural, given the out-of-core nature of IWAVE: we presume that the external data object is a structured disk file. Many of the many possible file structures, we elected to use the RSF file structure for material parameter fields (Fomel(2009)), and SEGY for seismic traces (Barry *et al.*(1980); Cohen & Stockwell(2008)). The third element for each type of external data is an appropriate RVL Vector class, representing the left-hand-side of the arrow in Figure 1. Since RVL Vectors implement low-intensity operations like scaling, vector addition, and inner product, communication costs will probably overwhelm any benefit of distribution, on the scale of modeling cost. Therefore we implement these operations as disk-to-disk filters running on the root process. The corresponding `Space` classes generate Vector workspace of these types, which involves creation of files. We distinguish between archival workspace, for which filenames are externally supplied, and temporary workspace, for which temporary filenames are generated via one of the Unix temp filename utilities. Temporary workspace files are removed from the file system on destruction of their corresponding objects.

One other aspect of this linkage must be addressed: the roles assigned to data files in IWAVE is regulated by a parameter table, as explained in subsection 6.1. For files, the key in the "key=value" pair indicates the role of the data contained in the file, and the value is the filename. Thus a sound velocity field stored in an RSF file structure `foo.rsf` would be indicated by

```
velocity=foo.rsf.
```

For an RVL Vector of the RSF type to provide an input velocity file to IWAVE, both the

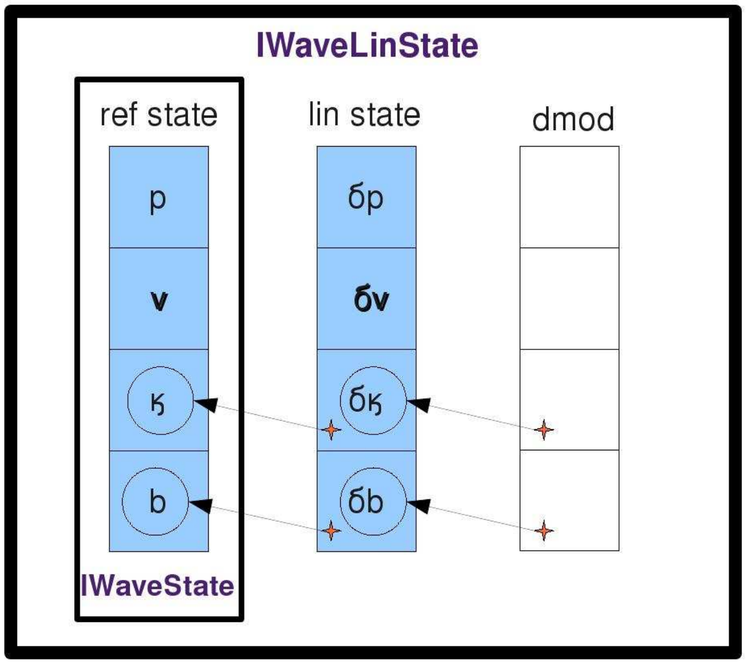

Figure 2. Construction of `IWaveLinState`. In this diagram, the lower case descriptions represent IWAVE structs which constitute the principal data of IWAVE++ classes (with upper case). An IWaveLinState object consists of three IWAVE state structs. Construction begins by initializing the first two IWAVE state objects from file data. Each consists of static and dynamic discrete fields - for acoustics, the former are bulk modulus κ and bouyancy b, the latter are pressure p and velocity $\mathbf{v}$. The first IWAVE state object consists of $(\mathbf{m},\mathbf{u}) = (\kappa, b, p, \mathbf{v})$ (as in the left box), the second of $(\delta\mathbf{m}, \delta\mathbf{u}) = (\delta\kappa, \delta b, \delta p, \delta\mathbf{v})$. Both are read from files as directed by the IWAVE parameter table driven i/o functions. Next, pointers to the static perturbation parameters $\delta\kappa, \delta b$ are copied to the third IWAVE state struct, which has had no memory allocated to it, and pointers to the static reference parameters κ, b are copied over the pointers to static perturbation parameters. A generalized IWAVE time step function takes the form `int gts(r,p,c);` in which `r, p,` and `c` are IWAVE state structs. This function takes the dynamic fields to be udpated from its first argument `r`, the dynamic fields to be used on the right hand side of the update from `p`, and the static (material parameter) fields from `c`. An `IWaveLinState` containing three IWAVE structs `ref_state`, `lin_state`, and `dmod`, as indicated in the diagram, is updated by the `IWaveLinStep` object via *three* calls to the `gts` function: first with `r = p = c = lin_state`, which executes the first update in BORN SIM (2.1); second, with `r=lin_state, p=ref_state, c=dmod`, which executes the second update in BORN SIM (2.1); and third, with `r = p = c = ref_state`, which executes the third update in BORN SIM (2.1). ADJ SIM is implemented similarly.

fact that the associated file stores a velocity, and the name of the file, must be extracted from the RVL Vector's data and passed through to IWAVE. Since the file data completely determines both realizations (as RVL Vector and as a component of IWAVE internal state), this information must be present in the file as well. Accordingly, we have extended the RSF header file structure to include data type information, as for example

```
data_type = velocity
```

and provided a function that parses this information and adds the appropriate line to the IWAVE parameter table. We have also added a scale parameter to the header to permit on-the-fly conversion between metric units.

Ultimately we intend to replace this device with encoding for units, which are intrinsic to the data and imply both data type and conversion to any internal units.

Finally, we have not yet addressed the last step: the way in which the structure so far built up defines an RVL operator. TSOpt defines a generic RVL Operator interface, but there are enough special features in this structure that a one-off operator interface seemed advisable. This `IWaveOp` class defines three member functions which transfer filenames to parameter tables as needed, initialize `State` and `Sample` objects, create `Sims` as necessary, and run the timestepping loops needed to define F, DF, and DF^T, according to the pattern set out in section 3. Because of the linkage with the specialized RVL vector classes, cartooned in Figure 1, calling these member functions is equivalent to evaluation of various RVL Operator methods.

6.5 Numerical examples

Our exposition would be incomplete without a couple of examples, illustrating that a package can be built according to the principles we have outlined, and will perform as required: modeling and its auxiliary operations combine with optimization algorithms based on vector calculus to carry out successful inversions. We give two 2D examples, one of quality control type, the other a revival of examples from a seminal paper.

6.5.1 A dot-product test

The dot-product test assesses the quality of the adjoint operator implementation $DF[m]^T$, by comparing the data space inner product $\langle DF[m]\delta m\,,\,\delta d\rangle_D$ with the model space inner product $\langle \delta m\,,\,DF[m]^T\delta d\rangle_M$, for randomly generated 2D model perturbation vectors $\delta m = (\delta\kappa, \delta b)$ and δd. Except for the random assignment of vectors, this test lies entirely within the realm of RVL base class attributes. RVL also provides an abstract interface for evaluation of componentwise functions such as random sample assignment, and uses this interface to compose a `AdjTest` function, a call to which performs the dot product tests. We have used `AdjTest` many times to assess the accuracy of the IWAVE++ adjoint (RTM) operator for acoustic modeling. The example presented here uses a simple homogeneous model m ($\kappa = 11109$ MPa, $\rho = 2100$ kg/m^3 for which sound velocity is $c = 2.3$ km/s). A point source with a 15 Hz Ricker pulse is located at the position $z = 40, x = 3300$ m, and receivers are placed at positions $z = 80$ m, $x = 2650 + i * 20$ m for $i = 0, \ldots, 49$. We used a mean-zero pseudorandom number generator to create input vectors δm and δd, sampled on the 10m square computational grid. The computational domain is 1.8×6.6 km. The scattered wavefield $DF[m]\delta m$ is shown in Figure 3. The bulk modulus component of the migrated model perturbation $DF^T[m]\delta d$ is shown in 4; the bouyancy component is very similar.. Table 1 displays the two inner products, and their difference relative to an estimate for the size of the bilinear form - note that some normalization reflecting the sizes of all three factors (δm, δd, and $DF[m]$), is necessary for a meaningful assessment. Somewhat arbitrarily, we select 100 times the machine precision (maximum relative roundoff, approximately 10^{-7} for the single-precision arithmetic used in this example) as the cutoff level for success in this comparison; it is achieved.

6.5.2 FWI: the "Camembert" model

We used IWAVE++ to recreate the milestone "Camembert" full waveform inversion experiments of (Gauthier *et al.*(1986)). We present one of our recreations here, as well as the result of an interesting test not included in the original paper.

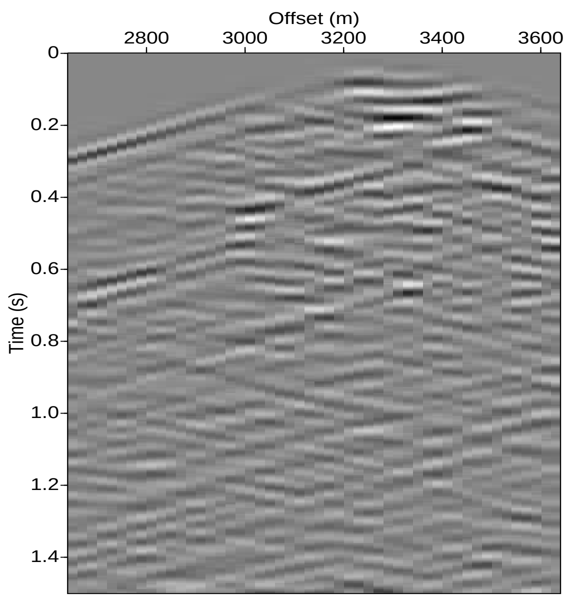

Figure 3. Born modeling acoustic synthetic shot gather ($DF[m]\delta m$) resulting from homogeneous background m and random δm.

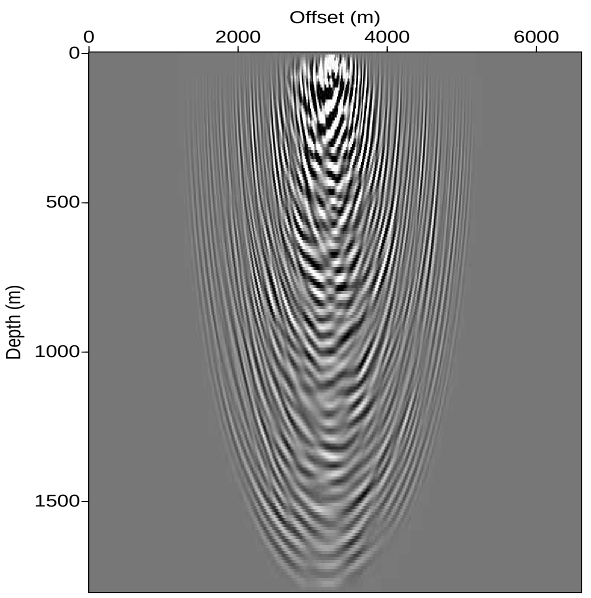

Figure 4. Migrated bulk modulus (component of $DF[m]^T\delta d$) with random δd.

$\langle DF[m]\delta m\,,\,\delta d\rangle_D$	`-2.91666225e+06`
$\langle \delta m\,,\,DF[m]^T\delta d\rangle_D$	`-2.94833250e+06`
$\Vert DF[m]\delta m\Vert_M \Vert\delta d\Vert_D$	`3.05501747e+09`
$\frac{\langle\left\vert\langle DF[m]\delta m\,,\,\delta d\rangle_D - \langle\delta m\,,\,DF[m]^T\delta d\rangle_M\right\vert}{\Vert DF[m]\delta m\Vert_M\Vert\delta d\Vert_D}$	`1.03666343e-05`
$100 *$ `macheps`	`1.19209290e-05`

Table 1. Standard RVL test for accuracy of adjoint operator pair: adequate quality if model space and data space inner products differ by less than a modest multiple of machine precision, relative to data-space norms of input and output data perturbations.

The Camembert model consists of a circular perturbation, the diameter of which is about ten wavelengths (500 m), superimposed on a homogeneous medium of bulk modulus $\kappa_0 = 2.5 \times 10^4$ MPa and density $\rho_0 = 4 \times 10^3$ kg/m^3. Inside the circular perturbation zone, the bulk modulus is 20% higher than in the surrounding region, that is, 3.0×10^4 MPa.

This model is set up on a grid of 200×200 points with $\Delta x = \Delta z = 5$ m. Eight sources and 100 receivers are located respectively at positions $(110 \times s, 40)$ m for $s = 1, 2, \dots, 8$ and $(10 \times r, 80)$ m for $r = 0, 1, \dots, 99$ (the reflection configuration from (Gauthier *et al.*(1986))

To recreate one of the tests, we synthesized data (using IWAVE). In this experiment, the source pulse is a Ricker wavelet of central frequency about 50 Hz. Figure 5 shows the inverted bulk modulus after 5 iterations. Only the boundary of the perturbation is found, that is, its higher spatial frequency components. The objective function decreased by more than an order of magnitude; inspection of the residual data shows that its energy is concentrated in the lower end of the data passband.

This example was central to a major message of (Gauthier *et al.*(1986)), namely that inversion of bandlimited *reflection* data is very difficult: the estimated model is unlikely to update the slowly varying model components, hence will exhibit mispositioned and/or imperfectly focused reflectors. One interesting variation on this experiment is therefore to remove the bandlimitation. We performed an inversion using data created with a low-pass filter with high-cut at 60 Hz. Figure 6 shows the inverted bulk modulus after BFGS 5 it-

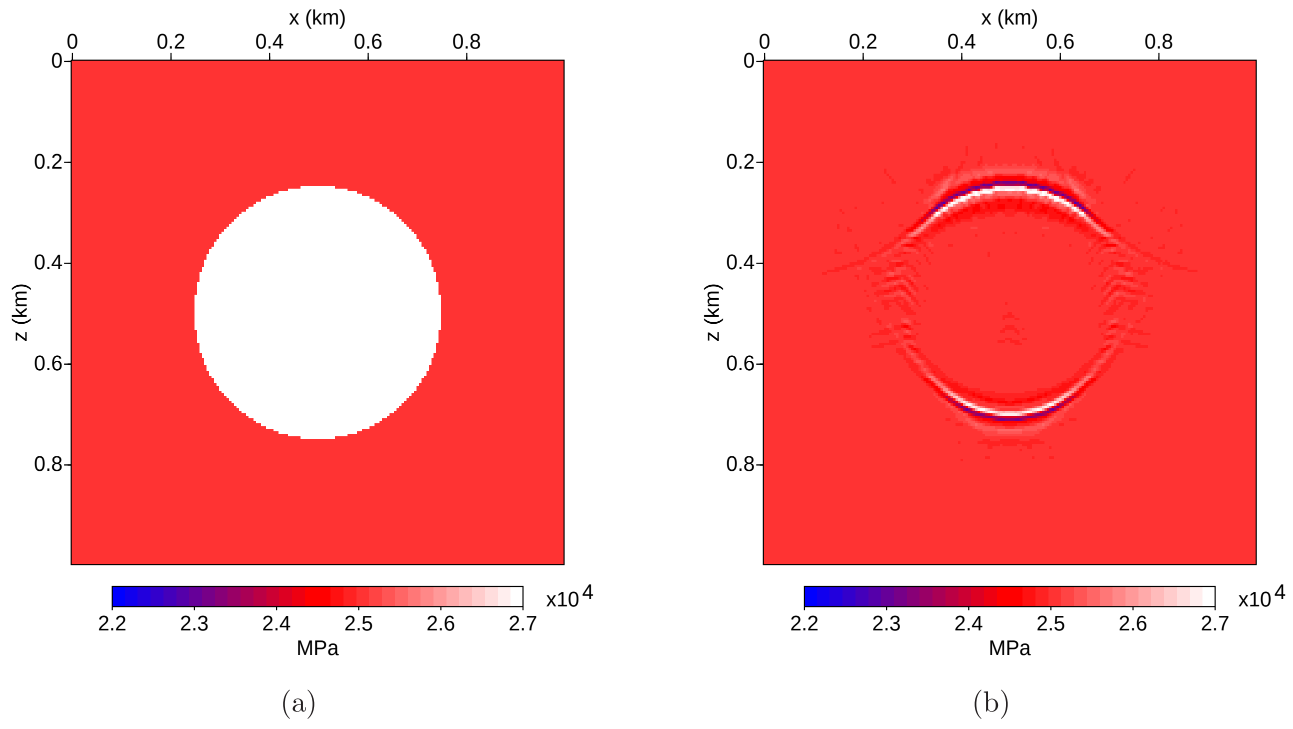

Figure 5. Camembert model, reflection configuration, Inverted bulk modulus, 5 BFGS iterations, 50 Hz Ricker source: (a) true bulk modus model; (b) inverted bulk modus model.

erations. Except for aperture-induced artifacts, the full spatial bandwidth of the velocity anomaly is recovered, and the decrease in the objective function is almost two orders of magnitude, decisively greater than in the bandlimited example.

7 BEYOND INVERSION: EXTENDED MODELING

The paper from which we have taken the "Camembert" examples, (Gauthier *et al.*(1986)), was the first of many to explore the intrinsic difficulty of inversion of long wavelength model components. It appears that, as a function of velocity, the least squares objective (1), in all of its variants, has multiple local minima at many models differing significantly from the optimal model. This difficulty is less severe for transmission than for reflection data, as was clear from the examples in (Gauthier *et al.*(1986)) and verified in much recent work (Pratt(1999); Brenders & Pratt(2007a); Krebs *et al.*(2009); Plessix(2009); Brossier *et al.*(2009); Plessix *et al.*(2010); Vigh *et al.*(2010)). High S/N at low frequency is also very helpful (Bunks *et al.*(1995); Shin *et al.*(2001); Sirgue & Pratt(2004); Cheong *et al.*(2006))

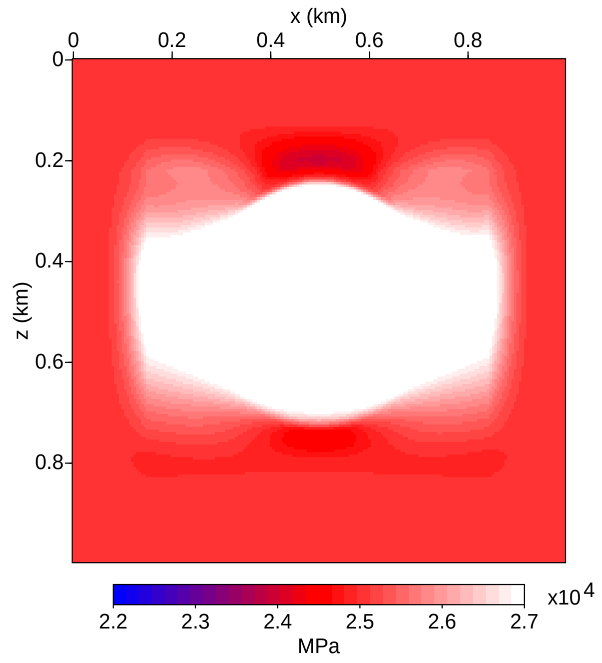

Figure 6. Camembert model, reflection configuration, Inverted bulk modulus, 5 BFGS iterations, 60Hz low-pass filter source (true bulk modulus model same as in Figure 5).

for both reflection and transmission data (the second "Camambert" example above is an illustration of this fact). In general, it appears that for reflection configurations, sufficiently high S/N at sufficiently low frequency will be difficult to obtain.

Two other methods are in widespread use for determining velocity models from reflection data of conventional bandwidth. Reflection tomograophy (Bishop *et al.*(1987); Delprat-Jannaud & Lailly(1993); Billette & Lambaré(1998)) replaces waveform data with reflection traveltime or slope picks, therefore is not directly comparable to Full Waveform Inversion. Migration velocity analysis ("MVA") (Kleyn(1983); Yilmaz & Chambers(1984); Deregowski(1990); Lafond & Levander(1993); Liu & Bleistein(1995); Yilmaz(2001)) uses waveform data directly, by optimizing attributes of prestack migrated image volumes which express consistency between data and velocity structure. MVA can have a large interpretational or manual component, but several researchers have proposed to make MVA quantitative via an optimization principle. Some of these proposals rely on a semblance measure related to stack power or image matching (Toldi(1989); Biondi & Sava(2004); Soubaras &

Gratacos(2007)). Others measure data-model compatibility via similarity of *nearby* image sections. This *differential semblance* approach, introduced in (Symes(1986)), has a number of variants; the survey paper (Symes(2008)) gives extensive references.

As explained in (Symes(2008)), differential semblance (and in fact migration velocity analysis in general) depends on *extended modeling*: that is, on an *extended model space* $\bar{\mathcal{M}}$, a *model extension operator* $\chi : \mathcal{M} \rightarrow \bar{\mathcal{M}}$, and an *extended modeling operator* $\bar{F} : \bar{\mathcal{M}} \rightarrow D$ so that $F[m] = \bar{F}[\chi[m]]$ for every $m \in \mathcal{M}$. Prestack migration, from this point of view, is simply the application of the adjoint of the extended Born modeling operator, that is, the image volume for a velocity model m and data d is simply $I = D\bar{F}[\chi[m]]^T d$. Quantitative MVA is then the minimization over m of

$$J_{rmMVA}[m] = \frac{1}{2}\|AD\bar{F}[\chi[m]]^T d\|^2$$

in which A is a so-called annihilator. Stack-power, image-matching, and differential semblance MVA can all be expressed this way, with various choices of A.

Some mathematical, and much numerical, evidence points to a tentative verdict: so long as A is chosen in the fashion suggested by the differential semblance principle, J_{MVA} is *effectively unimodal* - that is, lacking in local minima other than models which are kinematically consistent with the data. The differential semblance method appears to share this property with reflection tomography, but uses waveform data instead of traveltime picks.

As shown in (Symes(2008)), minimization of J_{MVA} is in fact a waveform inversion method for the "joint" Born modeling operator $(m, \delta m) \rightarrow DF[m]\delta m$: that is, simultaneous inversion for "velocity" m and "reflectivity" δm. It is natural to think that full waveform inversion itself could also be formulated in terms of extended models, and that this nonlinear modification of differential semblance MVA could remove some of the limitations associated with the failure of Born modeling to account for multiply reflected energy, while retaining the unimodularity property noted above.. The formal structure of this approach is described in (Symes(2008)), and early numerical results appear there and in Sun's MA thesis (Sun(2009); Sun & Symes(2009)).

We now turn to the inclusion of extended modeling in the software framework we have proposed in the preceding sections. Two variants of extended modeling underly commonly used MVA algorithms, as is explained in (Symes(2008)): *surface oriented* and *depth-oriented.*

7.1 Surface-oriented extension

In the first variant of extended modeling, the extended model space $\mathcal{M}$ is embedded in a space of functions of subsurface location and *source position* (or some other acquisition-related parameter - we use source position here for convenience. That is, an (ordinary) model is $m(x, y, z)$, whereas an extended model takes the form $m(x, y, z, x_s, y_s)$. The modeling operator is identical to the usual one, except that *a different model is used for each source.* Almost nothing changes in the computational framework, *except that the static field initialization moves inside the simulation loop*: in contrast to the basic algorithm presented in section 3,

```
while (initialize_sample(Sample, State, Time, Data))
   initialize_static(State,Model,Data)
   initialize_dynamic(State)
   run Sim<State,Step,Sample,Time>
```

The *initialize_static* function must now use information from the `Data` object to select (possibly by interpolation) an appropriate `Model`.

Note that in the adjoint loop ADJ SIM, the initialization $\lambda\mathbf{m} = 0$ moves inside the simulation loop; it is part of `initialize_adj_static`, which also moves inside the loop:

```
construct AdjSynch(AdjStep, RASim<State,Step,Sample,Time>)
while (initialize_sample(Sample, LinState, Data, AdjTime))
   initialize_adj_statlic(LinState,Model}
   initialize_adj_dynamic(LinState)
   while (AdjTime())
      AdjSample(LinState....)
```

```
AdjStep(LinState)

AdjSynch(LinState)
```

and `AdjSample` accumulates into the *current* `Model` object (part of `LinState`).

Altogether, this extension mode is quite straightforward. See (Sun(2009)) for an example computation.

7.2 Depth-oriented extension

In this extension, related conceptually to *shot-geophone migration* (Claerbout(1971)), the material parameter vector m becomes a vector of *operators*. A natural physical interpretation is that the extension permits action-at-a-distance: some of the material parameters (for example) are Hooke tensor components mediating between stress and strain, and permitting these parameters to become operators means that the dependence of stress on strain is non-local.

In this interpretation, material parameters depend on at least some additional spatial vectors (that is, act as the *kernels* of the operators): for example, $\bar{m}(x, y, z, h_x, h_y)$. The action of such a parameter on a dynamical field parameter via a partial time step operator L is represented as an integral, at least formally:

$$L[m, u] = \int_{H_{\min}(x,y)}^{H_{\max}(x,y)} dh_x \int_{H_{\min}(x,y)}^{H_{\max}(x,y)} dh_y m(x, y, z, h_x, h_y) u(x + h_x, y + h_y, z). \qquad (13)$$

Note that this action is still bilinear. For physical (ordinary) material parameters, $\bar{m}(x, y, z, h_x, h_y) = m(x, y, z)\delta(h_x)\delta(h_y) = \chi[m](x, y, z, h_x, h_y)$.

In a sense, this extension is even simpler, as absolutely nothing changes about the computational framework, except the definition of (some of) the partial time step operators. These changes are opaque to the rest of the framework.

For this extension to be practical, the computational cost of matrix multiplication in every time step, inherent in any discretized version of (13), must be mitigated somehow. Standard MVA practice suggests one approach: assume that the Hooke fields are physical (i.e. $H_{\min}(x, y), H_{\max}(x, y) \simeq 0$ except at a discretely sampled selection of midpoints (x, y). Thus the cost per time step is reduced to matrix multiplies at only the "analysis" midpoints.

As explained in (Symes(2008)), for this type of extension, the adjoint map $D\bar{F}[\chi[m]]^T$ is a prestack RTM shot-geophone imaging operator as discussed by (Biondi & Shan(2002)), for instance.

8 CONCLUSION

The design discussed in this paper accomplishes three major goals:

- reuse of modeling time step functions in Born and adjoint modeling;
- transfer of high-performance computing features, such as parallelization and sophisticated algorithmic options, from modeling to inversion software without requiring extensive rewrites; and
- modular implementation of optimization and high-level modeling algorithms (such as checkpointing for reverse time loops), without sacrifice of performance.

Some of the design features developed in this paper, such as the role of bilinear stencil operators in organizing the several modeling functions that underly gradient-based inversion, are unavoidable consequences of the mathematics. Others, such as use of an external and independent data representation to connect the internal data structures of vector calculus and modeling components of an inversion algorithm, are the results of more subjective design decisions which are, we believe, well-supported by the end result.

While we have for the most part devoted our attention to standard FWI, we have also noted that the design extends in a straightforward fashion to MVA-motivated extended modeling, and so can serve as the foundation for an approach to FWI which may avoid the well-known "local minimum" problem.

In this paper, we have avoided any detailed description of low-level implementation issues such as choice of language, and have in particular presented no actual code. We made apparently natural decisions on language and similar choices in implementing our demonstration framework, IWAVE++. Some of the references describing the constituents of IWAVE++ (IWAVE, TSOpt, RVL) delve into these issues, and complement the description in this paper of the natural abstract roles that these packages play, and how they interact to form a complete inversion application. This system of abstract roles could serve as the foundation for many implementations beyond the one we have created, with many of the same advantages.

Those readers wishing to inspect the code may download the currently available packages from The Rice Inversion Project's software distribution web site (Terentyev *et al.*(2010)).

9 ACKNOWLEDGMENTS

The work reported here was partially supported by the National Science Foundation under grant DMS 0620821, and by the sponsors of The Rice Inversion Project. The SEAM project provided partial support for the development of IWAVE. Development of RVL and TSOpt was supported in part by the National Science Foundation under grants DMS-9973423, DMS-9973308, and EAR-9977697, by the Department of Energy under Contracts 74837-001-03-49 and 86192-001-04-49 through the Los Alamos National Laboratory Computer Science Institute (LACSI), by the Department of Energy EMSP grant DE-FG07-97-ER14827, by grants from ExxonMobil Upstream Research Co.,and by the sponsors of The Rice Inversion Project.

REFERENCES

Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E., Lopez, J., O'Hallaron, D., Tu, T., & Urbanic, J., 2003. High resolution forward and inverse earthquake modeling on terascale computers, in *Proceedings*, Association for Computing Machinery.

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Curfman McInnes, L., & Smith, B. F., 2001. PETSc home page.

Barkved, O., Heavey, P., Kommedal, J. H., van Gestel, J.-P., Synnove, R., Pettersen, H., Kent, C., & Albertin, U., 2010. Business impact of Full Waveform Inversion at Valhall, in *Expanded Abstracts*, pp. 925–929, Society of Exploration Geophysicists.

Barry, K., Cavers, D., & Kneale, C., 1980. SEG-Y - recommended standards for digital tape formats, in *Digital Tape Standards*, Society of Exploration Geophysicists, Tulsa.

Benson, S., Curfman McInnes, L., Moré, J., Munson, T., & Sarich, J., 2007. TAO user manual (revision 1.9, Tech. Rep. ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, http://www.mcs/anl.gov/tao.

Billette, F. & Lambaré, G., 1998. Velocity macro-model estimation from seismic reflection data by stereotomography, *Geophysical Journal International*, **135**, 671–680.

Biondi, B. & Sava, P., 2004. Wave-equation migration velocity analysis - I: Theory, and II: Subsalt imaging examples, *Geophysics*, **52**, 593–623.

Biondi, B. & Shan, G., 2002. Prestack imaging of overturned reflections by reverse time migration, in *Expanded Abstracts*, pp. 1284–1287, Society of Exploration Geophysicists.

Bishop, T.N., Bube, K.P., Cutler, R.T., Laingan, R.T., Love, P.L., Resnick, J.R., Shuey, R.T., Spindler, D.A., & Wyld, H.W., 1987. Tomographic determination of velocity and depth in laterally varying media, *Geophysics*, **50**, 903–923.

Blanch, J.O., Symes, W. W., & Versteeg, R., 1998. A numerical study of linear inversion in layered viscoacoustic media, in *Comparison of Seismic Inversion Methods on a Single Real Dataset*, edited by R. Keys & D. Foster, Society of Exploration Geophysicists, Tulsa, OK.

Brenders, A. & Pratt, G., 2007. Full waveform tomography for lithospheric imaging: results from a blind test in a realistic crustal model, *Geophysical Journal International*, **168**, 133–151.

Brenders, A. & Pratt, G., 2007. Efficient waveform tomography for lithospheric imaging: implications for realistic, 2-d acquisition geometries and low frequency data, *Geophysical Journal International*, **168**, 152–170.

Brossier, R., Operto, S., & Virieux, J., 2009. Seismic imaging of complex onshor structures by two-dimensional elastic frequency-domain full-waveform inversion, *Geophysics*, **74**, WCC63–WCC76.

Bunks, C., Saleck, F., Zaleski, S., & Chavent, G., 1995. Multiscale seismic waveform inversion, *Geophysics*, **60**, 1457–1473.

Cheong, S., Pyun, S., & Shin, C.-S., 2006. Two efficient steepest-descent algorithms for source signature-free waveform inversion: synthetic examples, *Journal of Seismic Exporation*, **14**, 335–345.

Claerbout, J. F, 1971. Toward a unified theory of reflector mapping, *Geophysics*, **36**, 467–481.

Clapp, R. E., 2009. Reverse time migration with random boundaries, in *Expanded Abstracts*, pp. 2809–2813, Society of Exploration Geophysicists.

Cohen, J. K. & Stockwell, Jr. J. W., 2008. CWP/SU: Seismic Unix release no. 39: a free package for seismic research and processing, Center for Wave Phenomena, Colorado School of Mines.

Delprat-Jannaud, F. & Lailly, P., 1993. Ill posed and well posed formulation of the reflection traveltime tomography problem, *Journal of Geophysical Research*, **98**, 6589–6605.

Deregowski, S.M., 1990. Common offset migrations and velocity analysis, *First Break*, **8**, 225–234.

Dussaud, E., Symes, W. W., Williamson, P., Lemaistre, L., Singer, P., Denel, B., & Cherrett, A., 2008. Computational strategies for reverse-time migration, in *Expanded Abstracts*, pp. 2267–2271, Society of Exploration Geophysicists.

Enriquez, M. & Symes, W. W., 2009. An overview of timesteping classes for optimization, Tech. Rep. 09-33, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Fehler, M., 2009. SEAM Phase I Progress Report: numerical simulation verification, *The Leading Edge*, **28 (3)**, 270–1.

Fomel, S., 2009. Madagascar web portal, http://www.reproducibility.org, accessed 5 April 2009.

Gauthier, O., Tarantola, A., & Virieux, J., 1986. Two-dimensional nonlinear inversion of seismic waveforms, *Geophysics*, **51**, 1387–1403.

Griewank, Andreas, 1992. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, *Optimization Methods and Software*, **1**, 35–54.

Griewank, A., 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics (Frontiers in Applied Mathematics 19), Philadelphia.

Heroux, M. A.., Barth, T., Day, D., Hoekstra, R., Lehoucq, R., Long, K., Pawlowski, R., Tuminaro, R., & Williams, A., 2003. Trilinos: object-oriented, high-performance parallel solver ligraries for the solution of large-scale complex multi-physics engineering and scientific applications, Tech. rep., Sandia National Laboratories, Albuquerque, NM.

Hu, W., Abubakar, A., & Habashy, T., 2007. Application of the nearly perfectly matched later in acoustic wave modeling, *Geophysics*, **72**, SM169–SM176.

Kleyn, A.H., 1983. *Seismic Reflection Interpretation*, Applied Science Publishers, New York.

Kolda, T. & Pawlowski, R., 2003. NOX: An object-oriented, nonlinear solver package, Tech. rep., Sandia National Laboratories, Livermore, CA.

Krebs, J. R., Anderson, J. E., Hinkley, D., Neelamani, R., Lee, S., Baumstein, A., & Lacasse, M.-D., 2009. Fast full-waveform seismic inversion using encoded sources, *Geophysics*, **74**, WCC177–WCC188.

Lafond, C. F. & Levander, A. R., 1993. Migration moveout analysis and depth focusing, *Geophysics*, **58**, 91–100.

Levander, A., 1988. Fourth-order finite-difference P-SV seismograms, *Geophysics*, **53**, 1425–1436.

Levander, A.R., 1988. Fourth order finite difference P-SV seismograms, *Geophysics*, **53**, 1425–1434.

Liu, Z. & Bleistein, N., 1995. Migration velocity analysis: theory and an interative algorithm, *Geophysics*, **60**, 142–153.

Moczo, P., Robertsson, J. O. A., & Eisner, L., 2006. The finite-difference time-domain method for modeling of seismic wave propagation, *Advances in Geophysics*, **48**, 421–516.

Nocedal, J. & Wright, S., 1999. *Numerical Optimization*, Springer Verlag, New York.

Padula, A. D., Symes, W. W., & 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.

Plessix, R.-E., 2006. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications, *Geophysical Journal International*, **167**, 495–503.

Plessix, R.-E., 2009. Three dimensional frequency-domain full waveform inversion with an iterative solver, *Geophysics*, **74**, WCC149–WCC163.

Plessix, R.-E., Baeten, G., de Maag, J. W., Klaassen, M., Zhang, R., & Tao, Z., 2010. Application of acoustic full waveform inversion to a low-frequency large-offset land data set, in *Expanded Abstracts*, pp. 930–934, Society of Exploration Geophysicists.

Pratt, R.G, 1999. Seismic waveform inversion in the frequency domain, part 1: Theory, and verification in a physical scale model, *Geophysics*, **64**, 888–901.

Shin, C., Jang, S., & Min, D.-J., 2001. Improved amplitude preservation for prestack depth migration by inverse scattering theory, *Geophysical Prospecting*, **49**, 592–606.

Sirgue, L. & Pratt, G., 2004. Efficient waveform inversion and imaging: a strategy for selecting temporal frequencies, *Geophysics*, **69**, 231–248.

Soubaras, R. & Gratacos, B., 2007. Velocity model building by semblance maximization of modulated-shot gathers, *Geophysics*, **72**, U67.

Sun, D, 2009. The nonlinear differential semblance algorithm for plane waves in layered media, Tech. Rep. 09-04, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Sun, D. & Symes, W. W., 2009. A nonlinear differential semblance strategy for waveform inversion: Experiments in layered media, in *Expanded Abstracts*, pp. 2526–2530, Society of Exploration Geophysicists.

Sun, D. & Symes, W. W., 2010. IWAVE implementation of Born simulation, Tech. Rep. 10-05, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Sun, D. & Symes, W. W., 2010. IWAVE implementation of adjoint state method, Tech. Rep. 10-06, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Symes, W. W., 1986. Stability and instability results for inverse problems in several-dimensional wave propagation, in *Proc. 7th International Conference on Computing Methods in Applied Science and Engineering*, edited by R. Glowinski & J. Lions, North-Holland, New York.

Symes, W. W., 2007. Reverse time migration with optimal checkpointing, *Geophysics*, **72**, SM213–222.

Symes, W. W., 2008. Migration velocity analysis and waveform inversion, *Geophysical Prospecting*, **56**, 765–790.

Symes, W. W. & Santosa, F., 1988. Computation of the Newton Hessian for least-squares solution of inverse problems in reflection seismology, *Inverse Problems*, **4**, 211–233.

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

Terentyev, I., 2009. A software framework for finite difference simulation, Tech. Rep. 09-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA.

Terentyev, I., Vdovina, T., Wang, X., & Symes, W. W., 2010. IWAVE: a framework for wave simulation.

Toldi, J., 1989. Velocity analysis without picking, *Geophysics*, **54**, 191–199.

Vigh, D., Starr, W., Kapoor, J., & Li, H., 2010. 3d full waveform inversion on a Gulf of Mexico WAZ data set, in *Expanded Abstracts*, pp. 957–961, Society of Exploration Geophysicists.

Virieux, J., 1984. SH-wave propagation in heterogeneous media: Velocity stress finite-difference method, *Geophysics*, **49**, 1933–1957.

Virieux, J., 1986. P-SV wave propagation in heterogeneous media: Velocity stress finite-difference method, *Geophysics*, **51**, 889–901.

Yilmaz, O., 2001. Seismic data processing, in *Investigations in Geophysics No. 10*, Society of Exploration Geophysicists, Tulsa.

Yilmaz, O. & Chambers, R., 1984. Migration velocity analysis by wavefield extrapolation, *Geophysics*, **49**, 1664–1674.

Zhu, C., Byrd, R. H., & Nocedal, J., 1997. L-BFGS-B, FORTRAN routines for large scale bound constrained optimization, *ACM Transactions on Mathematical Software*, **23**, 550–560.

APPENDIX A: ADJOINT STATE COMPUTATION

In this Appendix, we show a detailed derivation of adjoint state algorithm based on the basic evolution scheme discussed in section 2. The basic evolution system (3) together with sampling operators S^i $(i = 0, \dots, N)$ defines the prediction operator F, i.e., let

$$\vec{\mathbf{u}} = \begin{pmatrix} \mathbf{u}^0 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{u}^N \end{pmatrix}$$

and $S = \left(S^0 S^1 \dots S^N\right)$, the predicted data for control vector $\mathbf{m}$ is given by $F[\mathbf{m}] = S\vec{\mathbf{u}}$, where $\vec{\mathbf{u}}$ is derived from the system (3). From the definition, the derivative of F at control $\mathbf{m}$ is written as

$$DF[\mathbf{m}]\,\delta\mathbf{m} = S\,\delta\vec{\mathbf{u}} \tag{A1}$$

, where $\delta\vec{\mathbf{u}}$ is derived from the system (10), whose matrix form is:

$$\begin{pmatrix} \delta\mathbf{u}^0 \\ \delta\mathbf{u}^1 \\ \vdots \\ \delta\mathbf{u}^N \end{pmatrix} = \begin{pmatrix} 0 & 0 & & 0 \\ D_{\mathbf{u}}H[\mathbf{m},\mathbf{u}^0] & 0 & & 0 \\ & \ddots & & \\ 0 & & D_{\mathbf{u}}H[\mathbf{m},\mathbf{u}^{N-1}] & 0 \end{pmatrix} \begin{pmatrix} \delta\mathbf{u}^0 \\ \delta\mathbf{u}^1 \\ \vdots \\ \delta\mathbf{u}^N \end{pmatrix} +$$

$$\begin{pmatrix} 0 \\ D_{\mathbf{m}}H[\mathbf{m},\mathbf{u}^0] \\ \ddots \\ D_{\mathbf{m}}H[\mathbf{m},\mathbf{u}^{N-1}] \end{pmatrix} \delta\mathbf{m}. \tag{A2}$$

Due to the chain rule and the equation (4), we have

$$\begin{aligned} D_{\mathbf{u}}H[\mathbf{m},\mathbf{v}] &= W\,D_{\mathbf{u}}H_k[\mathbf{m},\mathbf{v}_k]\,D_{\mathbf{u}}H_{k-1}[\mathbf{m},\mathbf{v}_{k-1}]\,\cdots\,D_{\mathbf{u}}H_0[\mathbf{m},\mathbf{v}_0] \\ &= W\prod_{j=k}^{0} D_{\mathbf{u}}H_j[\mathbf{m},\mathbf{v}_j] \end{aligned} \tag{A3}$$

and

$$\begin{aligned} D_{\mathbf{m}}H[\mathbf{m},\mathbf{v}] = W\,\{&D_{\mathbf{m}}H_k[\mathbf{m},\mathbf{v}_k] + D_{\mathbf{u}}H_k[\mathbf{m},\mathbf{v}_k]\,D_{\mathbf{m}}H_{k-1}[\mathbf{m},\mathbf{v}_{k-1}] + \cdots \\ &+ D_{\mathbf{u}}H_k[\mathbf{m},\mathbf{v}_k]\,D_{\mathbf{u}}H_{k-1}[\mathbf{m},\mathbf{v}_{k-1}]\cdots D_{\mathbf{u}}H_1[\mathbf{m},\mathbf{v}_1]\,D_{\mathbf{m}}H_0[\mathbf{m},\mathbf{v}_0]\} \\ = W&\sum_{j=0}^{k}\left(\prod_{i=k}^{j+1} D_{\mathbf{u}}H_i[\mathbf{m},\mathbf{v}_i]\;\;D_{\mathbf{m}}H_j[\mathbf{m},\mathbf{v}_j]\right) \end{aligned} \tag{A4}$$

, where $\mathbf{v}_l = H_{l-1}[\mathbf{m}, \cdots H_0[\mathbf{m}, \mathbf{v}] \ldots]$ for $l = k, k-1, \ldots, 1$ and $\mathbf{v}_0 = \mathbf{v}$.

Given the assumption that

$$H_j[\mathbf{m}, \mathbf{v}] = \mathbf{v} + L_j[\mathbf{m}, \mathbf{v}]$$

and L_j is a bilinear state-vector valued operator for $j = 0, \ldots, k$, we have

$$D_{\mathbf{u}} H_j[\mathbf{m}, \mathbf{v}]\delta\mathbf{v} = \delta\mathbf{v} + L_j[\mathbf{m}, \delta\mathbf{v}] \tag{A5}$$

and

$$D_{\mathbf{m}} H_j[\mathbf{m}, \mathbf{v}]\delta\mathbf{m} = L_j[\delta\mathbf{m}, \mathbf{v}] \tag{A6}$$

for $j = 0, \ldots, k$.

For the bilinear state-vector values operator L_j ($j = 0, \ldots, k$), we define two kinds of adjoint operators as follows: for any model vector $\mathbf{m}$ and state vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, the state-space adjoint L_j^T of L_j satisfies

$$\langle L_j[\mathbf{m}, \mathbf{v}_1], \mathbf{v}_2 \rangle_S = \langle \mathbf{v}_1, L_j^T[\mathbf{m}, \mathbf{v}_2] \rangle_S \tag{A7}$$

; and the state-model-space adjoint M_j of L_j satisfies

$$\langle L_j[\mathbf{m}, \mathbf{v}_1], \mathbf{v}_2 \rangle_S = \langle \mathbf{m}, M_j[\mathbf{v}_1, \mathbf{v}_2] \rangle_M. \tag{A8}$$

Denote by $\mathcal{H}_u$ the first matrix on the right hand side of the equation (A2), and by $\mathcal{H}_m$ the second. The equaiton (A2) may be written as

$$\delta\vec{\mathbf{u}} = \mathcal{H}_u \delta\vec{\mathbf{u}} + \mathcal{H}_m \delta\mathbf{m}$$

which together with equation (A1) yields:

$$DF[\mathbf{m}] = S(I - \mathcal{H}_u)^{-1}\mathcal{H}_m$$

whence

$$DF[\mathbf{m}]^T = \mathcal{H}_m^T (I - \mathcal{H}_u^T)^{-1} S^T. \tag{A9}$$

The adjoint state algorithm aims to compute the action of the transposed Jacobian $DF[\mathbf{m}]^T$ on a (data-like) vector $\lambda\mathbf{d}$. It proceeds as follows:

Step 1: Apply the adjoint sampling operator:

$$\lambda\mathbf{d} \mapsto S^T\lambda\mathbf{d} = \begin{pmatrix} (S^0)^T \\ (S^1)^T \\ \vdots \\ (S^N)^T \end{pmatrix} \lambda\mathbf{d}.$$

Since S^n extracts the data from a state at time step n, its adjoint inserts the data into the state at time step n.

Step 2 ("backpropagation"):Solve the *adjoint state system*

$$(I - \mathcal{H}_u^T)\lambda\vec{\mathbf{u}} = S^T\lambda\mathbf{d}$$

for the *adjoint state vector* $\lambda\vec{\mathbf{u}} = (\lambda\mathbf{u}^0, ..., \lambda\mathbf{u}^N)^T$. Since $\mathcal{H}_u^T$ is *upper* triangular, solution of this system unfolds into a recursion *backwards* in the step index:

$$\lambda\mathbf{u}^N = (S^N)^T\lambda\mathbf{d};\ \lambda\mathbf{u}^n = (D_{\mathbf{u}}H[\mathbf{m},\mathbf{u}^n])^T\lambda\mathbf{u}^{n+1} + (S^n)^T\lambda\mathbf{d},\ n = N-1,\ldots,0 \qquad \text{(A10)}$$

In the update form, the above recursion may be written:

```
Backward SIM (1) $\lambda\mathbf{u} = 0$;
Backward SIM (2) For $n = N-1,\ldots,0$ do:
    Backward SIM (2.1) $\lambda\mathbf{u} \mathrel{+}= (S^{n+1})^T\lambda\mathbf{d}$;
    Backward SIM (2.2) $\lambda\mathbf{u} = W^T\lambda\mathbf{u}$;
    Backward SIM (2.3) For $j = k\ldots,0$ do:
        $\lambda\mathbf{u} \mathrel{+}= L_j^T[\mathbf{m},\lambda\mathbf{u}]$
```

Step 3 ("imaging"): apply the operator $\mathcal{H}_m^T$ to $\lambda\mathbf{u}$, which amounts to computing

$$\sum_{n=0}^{N-1}(D_{\mathbf{m}}H[\mathbf{m},\mathbf{u}^n])^T\lambda\mathbf{u}^{n+1} \qquad \text{(A11)}$$

In the update form, the above accumulation may be written:

```
IMG ACC (1) $\lambda\mathbf{m} = 0$
IMG ACC (2) For $n = N-1,\ldots,0$ do:
    IMG ACC (2.1)  $\lambda\mathbf{u} = W^T\lambda\mathbf{u}^{n+1}$;
    IMG ACC (2.2)  For $j = k, \ldots, 0$ do:
            $\lambda\mathbf{m} \mathrel{+}= M_j[\mathbf{u}^n_j, \lambda\mathbf{u}]$;
            $\lambda\mathbf{u} \mathrel{+}= L^T_j[\mathbf{m}, \lambda\mathbf{u}]$
```

It is natural to accumulate the sum in equation A11 term-by-term as the factors $\lambda\mathbf{u}^n$ are produced in the backpropagation loop (equation A10).

Hence, combining Step 2 and Step 3, we may write the adjoint evolution as:

```
ADJ SIM (1)  $\lambda\mathbf{u} = 0$, $\lambda\mathbf{m} = 0$;
ADJ SIM (2)  For $n = N-1, \ldots, 0$ do:
    ADJ SIM (2.1)  $\lambda\mathbf{u} \mathrel{+}= (S^{n+1})^T\lambda\mathbf{d}$;
    ADJ SIM (2.2)  $\lambda\mathbf{u} = W^T\lambda\mathbf{u}$;
    ADJ SIM (2.2)  For $j = k, \ldots, 0$ do:
            $\lambda\mathbf{m} \mathrel{+}= M_j[\mathbf{u}^n_j, \lambda\mathbf{u}]$;
            $\lambda\mathbf{u} \mathrel{+}= L^T_j[\mathbf{m}, \lambda\mathbf{u}]$
```