Reverse Time Migration with Optimal Checkpointing

William W. Symes [*]

ABSTRACT

The optimal checkpointing algorithm (Griewank and Walther, 2000) minimizes the computational complexity of the adjoint state method. Applied to reverse time migration, optimal checkpointing eliminates (or at least drastically reduces) the need for disk i/o, which is quite extensive in more straightforward implementations. This paper describes optimal checkpointing in a form which applies both to reverse time migration and to other applications of the adjoint state method, such as construction of velocity updates from prestack wave equation migration.

INTRODUCTION

Reverse time migration ("RTM") was introduced in the 1980's (Whitmore, 1983; Baysal et al., 1983), and has recently gained renewed attention from the seismic imaging community (Yoon et al., 2003; Biondi and Shan, 2002; Yoon et al., 2004; Mulder and Plessix, 2004; Bednar and Bednar, 2006). Similar "reversed" computations have been used to extract velocity updates from wave equation depth migration (Shen et al., 2003; Biondi and Sava, 2004; Soubaras and Gratacos, 2006; Albertin et al., 2006; Khoury et al., 2006), and in many other subjects to link parameter perturbations to simulations (Munk et al., 1995; Wang et al., 1998; Akcelik et al., 2003; Talagrand, 2007).

All of these algorithms are instances of the *adjoint state method*, a way of organizing the computation of a *gradient* of a cost or *objective* function depending on a recursive simulation, with roots in optimal control (Bryson and Ho, 1979; Pontryagin, 1987). Chavent and his co-workers introduced this concept into the study of geophysical inverse problems (Chavent and Lemmonier, 1974), and later applied it in their study of seismic inversion posed as a nonlinear least squares problem (Bamberger et al., 1977; Bamberger et al., 1979; Bamberger et al., 1982). Very soon after the introduction of RTM, several authors pointed out that it amounted to a single step of a gradient descent algorithm for output least squares inversion, computed via the adjoint state method (Lailly, 1983; Lailly, 1984; Tarantola, 1984). Plessix (2006) provides an excellent overview of the method and some of its geophysical applications.

[*]The Rice Inversion Project, Department of Computatio nal and Applied Mathematics, Rice University, Houston, TX 77005-1892 USA, email `symes@caam.rice.edu`

The adjoint state method, in any of its guises, poses an interesting computational complexity problem. The core of the algorithm is the crosscorrelation of two fields at the same (time or depth) level, one computed via forward (in time or depth) recursion, the other via backward recursion. It is natural to carry out the forward recursion first, but then its entire history must be made accessible during the backward recursion. For small problems, one simply stores all states reached during the forward recursion, and re-uses them as needed in the rest of the algorithm. For very large instances of the adjoint state method, such as 3D RTM, the required storage is so extensive as to require resort to the slowest levels of the memory hierarchy (i.e. disk i/o on modern platforms - this assertion amounts to a definition of “large”). This need for very large amounts of slow memory complicates the logistics of algorithm implementation and degrades performance.

The purpose of this short paper is to (re)introduce to the seismic imaging community an algorithm devised by Griewank which largely ameliorates the complexity problem just described (Griewank, 1992; Griewank and Walther, 2000; Griewank, 2000; Blanch et al., 1998; Akcelik et al., 2003). This *optimal checkpointing* algorithm trades floating point arithmetic for memory. It increases the computational complexity of the adjoint state method by a factor logarithmic in the total number of steps, while reducing the memory complexity to a number of state buffers also logarithmic in the total number of steps. Griewank’s prescription for this tradeoff is provably optimal. For 2D RTM, optimal checkpointing eliminates any need for disk i/o, for a factor of two (or so) increase in flops. For 3D RTM, optimal checkpointing may eliminate the need for disk i/o, but in any case drastically reduces it, with similar floating point penalty. Informal comparisons suggest that with current technology, the reduction in time required for slow memory access more than compensates for the extra flops. If floating point throughput advances more quickly than memory bandwidth, as seems likely, optimal checkpointing will become even more attractive.

In the next section, I will detail the adjoint state method in a form which applies to all of the problems mentioned above. The following sections describe the checkpointing concept, optimal checkpoint schedules, a few observations about implementation, and a RTM application. An appendix gives a complete derivation of the adjoint state method, in its general form.

ADJOINT STATE METHOD

I will use the notation $\mathbf{u}$ to denote the *state* vector of a discrete time- (or depth-) dependent system. For leapfrog (three-level) timestepping of a second-order wave equation for the displacement field in elasticity or the pressure in acoustics, $\mathbf{u}$ represents two successive time levels of the discrete displacement or pressure field. For a stress-velocity scheme, $\mathbf{u}$ holds one time level each of pressure and the velocity components. For depth extrapolation, $\mathbf{u}$ is simply a depth-slice of the reflected wavefield.

The evolution operator of the system depends on a vector $\mathbf{c}$ of model or *control* parameters. For most seismic problems, this vector represents the velocity model, and perhaps other parameters which influence the predicted seismic response.

A discrete linear evolution takes the form

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

in which the *evolution operator* $\mathbf{H}^n$ may depend on the step n as does the inhomogeneous or "source" term $\mathbf{f}^n$, and certainly depends on the control $\mathbf{c}$. For example, in a timestepping scheme for seismic modeling, $\mathbf{H}^n$ encodes the finite difference stencil. For depth extrapolation, it is the depth step operator.

The adjoint state computation associated to the system (1) is a *backwards* (i.e. backwards in step index) evolution for an *adjoint state* $\mathbf{w}$, of the same type as the system state $\mathbf{u}$. Input to the adjoint state computation is a datum $\mathbf{r}^n$ of the same type (data structure) as the output data of the simulation. The latter is usually related to the state $\mathbf{u}^n$ by a sampling operator $\mathbf{S}^n$ (for example, extracting time samples at receiver points). The output of the adjoint state computation is an object $\mathbf{g}$ of the same type as the control (or model) vector $\mathbf{c}$:

$$\left.\begin{array}{lcl}\mathbf{w}^N & = & 0, \\ \mathbf{g} & = & 0, \\ \mathbf{w}^n & = & (\mathbf{H}^{n+1}[\mathbf{c}])^T\mathbf{w}^{n+1} + (\mathbf{S}^n)^T\mathbf{r},\ n = N-1, ..., 0, \\ \mathbf{g} & = & \mathbf{g} + \mathbf{A}^n[\mathbf{c}, \mathbf{u}^n]^T\mathbf{w}^{n+1},\ n = N-1, ...0.\end{array}\right\} \tag{2}$$

As indicated by the indexing, the adjoint state $\mathbf{w}$ evolves *via* the *adjoint state equation* in the direction of decreasing index (i.e. decreasing time or depth, in the examples described above). The *imaging operator* $\mathbf{A}^n[\mathbf{c}, \mathbf{u}^n]^T$ is typically applied during this backwards loop, and increments the output vector $\mathbf{g}$ as indicated.

The Appendix gives a detailed derivation of the adjoint state method, as an algorithm to compute the gradient of a function of the output data.

The imaging (last) equation in the system (2) implies that the fields $\mathbf{u}^n$ and $\mathbf{w}^{n+1}$ must be available at the same time in an implementation of this algorithm. This is not a natural result of the pair of evolutions (1) and (2), as they run the step index in opposite directions. To make $\mathbf{u}^n$ available during the nth step of the backward evolution of $\mathbf{w}^n$, several possibilities suggest themselves, differing in balance between memory and computation:

1. Compute $\mathbf{u}^n$ from step $n = 0$, for each $n = N-1, ...0$. This involves $O(N^2)$ evolution steps in total, an unacceptable computational burden for problems of any significant size.

2. Store the entire time history of the state, $\{\mathbf{u}^n, n = 0, ..., N-1\}$, and access as needed. This option needs only the $O(N)$ steps of the basic evolution, but much larger amounts of storage.

3. Store only every kth step, $k > 1$ of the time history, and interpolate the rest in some way. This appears to be the approach taken by a number of commercial RTM implementations. Has the computation cost advantage of approach 2 with a k-fold reduction in memory use, but produces only an uncalibrated approximation to $\mathbf{g}$.

In the discussion to come, I will refer to these three approaches to implementation of the adjoint state system (2) as strategies 1, 2, and 3 respectively.

CHECKPOINTING

The title of this section describes an alternative to the three implementation strategies for the adjoint state method (2), described in the last section. For a detailed description of the concept, see (Griewank and Walther, 2000). I shall give only a brief overview.

Select *checkpoints* $\{n_i : i = 0, ...N_C\}$ between 0 and N, and provide *buffers* $\{\mathbf{b}_j : j = 1, ..., N_B\}$ to which states can be stored and from which states can be initialized. Typically $N_B << N_C << N$.

During the forward loop (i.e. solution of the system (1), store an initial selection of states $\mathbf{u}^n$ at checkpoint steps $\{n = n_{i_j} : j = 1, ..., N_B\}$ in the buffers. An essential constraint is that the last checkpoint selected must be the last checkpoint $n_{i_{N_B}} = \max\{n_i : i = 0, ..., N_C\}$.

During the backwards loop (i.e. solution of the third equation in display (2)), adopt strategy 1 from the last section, but compute $\mathbf{u}^n$ repeatedly *from the last checkpoint*, using the state value stored in the corresponding buffer $\mathbf{b}_{i_{N_B}}$ to initialize the evolution (1).

When the backwards step reaches the last checkpoint: $n = n_{i_{N_B}}$, use the buffer $\mathbf{b}_n$ storing the corresponding state value to store a new state value, at one of the previously unrecorded checkpoints (remember there are more checkpoints than buffers!). Compute this new value by initializing the evolution (1) at the last previously recorded checkpoint before the chosen one.

The backwards loop proceeds by repeated application of strategy 1, always from the last recorded checkpoint before the current index (thus over index intervals possibly much shorter than $[0, N]$). As each checkpoint n_i is reached, the state stored in its corresponding buffer $\mathbf{b}_{n_i}$ is used in the application of the imaging operator (last equation in display (2)), and then another previously unrecorded checkpoint is stored in the buffer.

The index values $n = 0, ..., N_B - 1$ are always checkpoints. At the end of the algorithm, these are the only remaining unused checkpoints and the corresponding buffers store the corresponding state vectors. Then an application of strategy 2 from the last section finishes the computation.

Note that a number of steps in the forward loop (1) are computed more than once, due to the employment of strategy 1 over limited intervals. A convenient way to measure the resulting computational cost is the *recomputation ratio*, i.e the ratio of the total number of steps of the forward loop taken in the algorithm divided by N.

The question remains: how should the checkpoints be chosen and in what order should they be used in the backward loop? The strategies 1 and 2 introduced above are obviously end-members: strategy 2 uses $N_C = N$ checkpoints, strategy 1 uses none ($N_C = 0$). The reader will easily see that $\sqrt{N}$ evenly spaced checkpoints and $\sqrt{N}$ buffers can be used to produce $\mathbf{g}$ in $O(N^{\frac{3}{2}})$ evolution steps, for a recomputation ratio of $\sqrt{N}$, a big improvement over the $N/2$ of strategy 1. As it turns out, it is possible to do much better than this.

OPTIMAL CHOICE OF CHECKPOINTS

In a beautiful piece of combinatorial mathematics, (Griewank, 1992), Andreas Griewank completely determined the optimal choice of checkpoints. Griewank's prescription is optimal in the sense that

- for given numbers of timesteps and buffers, the recomputation ratio is minimum amongst all possible checkpointing schedules, and
- for a given number of timesteps and a prescribed maximum recomputation ratio, the number of buffers required is minimum amongst all possible checkpointing schedules.

For a forward loop of length N, the number of buffers required is $O(\log N)$, and the recomputation ratio is also $O(\log N)$. If either of these two numbers is given, the other is determined. Otherwise put, for given recomputation ratio, the maximum length N of the forward loop grows exponentially with the number of buffers used. Contrast this behaviour with strategy 3 described above, in which N is linear in the number of buffers, the ratio being determined by the quality of interpolation (hence by bandwidth or some similar signal attribute).

Other accounts of this algorithm, with some improvements, appear in (Griewank and Walther, 2000; Griewank, 2000). The author and his collaborators used optimal checkpointing in a viscoacoustic least-squares inversion scheme (Blanch et al., 1998). Ghattas and his coworkers have also used the algorithm in the context of least-squares inversion of basin structure from earthquake data (Akcelik et al., 2003).

Table 1 illustrates the tradeoff between computation and memory achieved by optimal checkpointing, for a forward loop with $N = 10000$ steps. This number of steps is more or less a median value for finite difference time-domain simulation of a typical reflection seismic survey. For velocity analysis via wave equation depth migration (for example (Shen et al., 2003)), $N = 1000$ would be more appropriate.

Even for only 3 buffers, the recomputation ratio is under 28. This seems large, but should be compared to the ratio of 5000 to be expected for a straightforward implementation of strategy 1. The ratio drops rapidly as buffers are added, up to around 15 buffers, after which the incremental gain by adding each buffer becomes small. At about 36 buffers, the recomputation ratio is roughly 3. Since the backwards loop involves the same steps as the forwards loop plus application of the imaging operator, the cost of the adjoint state computation using optimal checkpointing and 36 buffers is somewhat above four times the cost of a simulation.

A straightforward implementation of strategy 2, i.e. maximal memory and minimal computation, would by the same reasoning cost somewhat more than twice the cost of a simulation. Thus the optimal checkpointing approach with 36 buffers completes the $N = 10000$ adjoint state loop in less than twice the computational cost of strategy 2. This comparison seriously understates the total cost of strategy 2, however, for modern NUMA architectures: it typically requires large amounts of slow (disk) memory access for

buffers	3	5	10	15	20	25	30	35	40	60
ratio	27.9	11.3	5.8	4.5	3.8	3.6	3.4	3.1	2.9	2.8

Table 1. Number of buffers and corresponding recomputation ratio for Griewank's optimal checkpointing scheme. All figures for N=10000 steps.

problems of even modest size. The time devoted to slow memory access can easily shift the comparison in favor of optimal checkpointing.

IMPLEMENTATION

The algorithm determining optimal checkpoint schedules described in (Griewank, 1992) is fairly complex. Fortunately, its author has provided a full, public domain implementation in both C and Fortran 77 (Griewank and Walther, 2000), along with an excellent article covering the topic. The algorithm presented in (Griewank and Walther, 2000) offers several improvements over that presented in the earlier reference (Griewank, 1992). The code package contains several useful utilities, one of which computes the recomputation ratio from the numbers of steps and buffers. I used this utility to create Table 1.

The attentive reader will note that up to this point our discussion of simulation and the adjoint state method has not specified any particular physical system or numerical method. In fact, the framework described up to this point is entirely abstract, posed only in terms of vector calculus concepts. I have implemented the adjoint state method with optimal checkpointing within an object-oriented framework for simulation-driven optimization (Symes et al., 2004; Symes et al., 2005), which permits the expression of abstract algorithms such as these. Creation of a full-blown simulation package for a specific system, including adjoint state, requires only the implementation of concrete classes defining the specific choices of the step operator $\mathbf{H}$ and its partial derivatives and their adjoints (one of which is the imaging operator $\mathbf{A}$), and the sampling operator $\mathbf{S}$.

The object-oriented framework described in Symes et al. (2004; 2005) uses the code provided by Griewank and Walther (2000) to compute the checkpointing schedule. I believe that the object-oriented approach to expression of high-level algorithms such as those described here has much to recommend it. However, it is relatively straightforward to use the codes from Griewank and Walther (2000) to add optimal checkpointing to a procedural implementation of the adjoint state method for a particular system, such as acoustic reverse time migration, see for example Blanch et al. (1998).

EXAMPLE: REVERSE TIME MIGRATION

This section describes the realization of a particular approach to RTM, namely leapfrog time marching for a scalar field (pressure), which will serve as an "existence proof" and an illustration of the computational efficiency achievable by optimal checkpointing. Both second and higher order (in time) schemes can be represented this way. RTM based on the pressure-velocity formulation of acoustics (for example so-called staggered grid schemes) also fit into the framework discussed in this paper, as do elastic RTM schemes.

Leapfrog time marching for the constant-density acoustic wave equation can be written as

$$p^{n+1} = 2p^n - p^{n-1} + \Delta t^2 v^2 L p^n + \Delta t^2 f^n \tag{3}$$

Here p^n approximates the pressure at time n, and L is a spatial operator approximating the Lapacian. For regular grid finite differences, $p^n_{ijk}, i = 0, ..., n_x, j = 0, ...n_y, i = 0, ...n_z$ approximates the pressure at $t = n\Delta t, x = i\Delta x, y = j\Delta y, z = k\Delta x$, and L is a finite difference Laplacian approximation. The inhomogeneous term f^n represents the system excitation via a body force density. The control parameters in this model are the velocity grid values, represented by v.

The scheme (3) fits into the form (1) if we define

$$\mathbf{u}^n = \left(\begin{array}{c} p^n \\ p^{n-1} \end{array} \right), \, \mathbf{c} = (v), \, \mathbf{H}^n[\mathbf{c}]\mathbf{u}^n = \left(\begin{array}{c} 2p^n = p^{n-1} + \Delta t^2 v^2 L p^n + \Delta t^2 f^n \\ p^n \end{array} \right) \tag{4}$$

In the example presented below, we have used the fourth order Laplace approximation by centered differences, which results in the so-called (2,4) displacement scheme (even though it computes pressure!) (Levander, 1989). The sampling operator S extracts estimates of hydrophone output, typically point samples, at the time sample rate of the output data traces. Many codes apparently accomplish this with nearest-neighbor interpolation, or assume that the receiver (and source) locations lie on gridpoints. The implementation reported here does not assume any relation between the computation grid and source/receiver locations, or between the simulation time step and the sample rate of the output traces. S is implemented using polynomial (space) and spline (time) interpolation to allow for arbitrary source and receiver locations and sample rates. Sources $\mathbf{f}$ are represented as linear combinations of points sources with (possibly) different wavelets. Source fields are adjoint-interpolated onto the computational grid.

The adjoint state field $\mathbf{w}$ has the same structure as does the acoustic field $\mathbf{u}$: write $\mathbf{w}^n = (q^n, q^{n+1})^T$. The imaging operator ($(\mathbf{A}^n)^T$ in the last equation of display (2)) is the partial derivative in $\mathbf{c}$ of the RHS of the evolution (1), according to the derivation given in the appendix. For the choices given in equations (3) and (4), imaging operator acts as

$$\mathbf{A}^n[\mathbf{c}, \mathbf{u}^n]^T \mathbf{w}^{n+1} = 2\Delta t^2 v L p^n q^{n+1} \tag{5}$$

That is, the last equation in (2) represents in this case accumulation of the expected crosscorrelation between p (source field) and q (receiver field). The various other factors, in particular the Laplacian, are necessary to ensure that this crosscorrelation actually computes the output of the adjoint operator to the Born seismic modeling operator. Simpler ad-hoc crosscorrelations may be adequate for imaging purposes but do not (necessarily) accurately compute the adjoint action.

The code uses Perfectly Matched Layer absorbing boundary conditions to simulate unbounded domain wave propagation (Cohen, 2001), which actually augments the systems (3) and (4) with additional fields and evolution laws which absorb energy near the boundary. On those parts of the boundary not subject to absorbing boundary conditions, the code employs a method-of-images implementation of the Dirichlet (pressure-release) condition.

I have implemented a 2D version of this algorithm in the object-oriented framework described in the preceding section. All of the finite difference code is written in Fortran 77, partly in order to use the automatic differentiation package TAMC (Giering and Kaminski, 1998) to generate code for the partial derivatives of **H** and their adjoints automatically. Especially with complicated absorbing boundary condition and numerous subloops, these computations are difficult (though always in principle possible) to carry out by hand. The code has elementary tuning applied (for example, none of the absorbing boundary computations are done away from the boundary) but no explicit source-level loop unrolling, blocking, or tiling. Tests show that at least 99% of the cycles in a simulation of even modest size occur in the Fortran portion of the code, even though in terms of line count it is a small part of the code packate.

For 2D applications of any reasonable size, each simulation handily fits (with plenty of room to spare) on an individual workstation or cluster node. I built parallelization over subsimulations (shots, in this case) into the structure of the object-oriented framework mentioned earlier (Symes et al., 2004). Three dimensional simulation and associated adjoint computations are likely to require loop-level parallelism (domain decomposition) at the level of the finite difference code, for use with typical contemporary cluster platforms.

The Marmousi 2D synthetic data (Versteeg and G.Grau, 1991) provides a reasonable test case. It involves a complex velocity model, lots of reflectivity, geological plausibility, 240 shots, and a 2.5 km cable. I simulated "true Born" data by smoothing the original Marmousi velocity model, subtracting a somewhat less stringent smoothing from the original to create a short wavelength perturbation (reflectivity), and computing a synthetic data set with the linearized ("Born") simulation code built according to the principles described here. I used the Rice University Cray XD1 cluster, consisting of 336 AMD Opteron 275 dual-core nodes, and the gcc4 compiler suite. On 120 CPUs (cores), the linearized simulation required approximately 39 min ("wallclock" - total CPU time was 78 hours). This is slightly more than twice the cost of a nonlinear simulation which ran in a bit over 18 minutes on the same platform. For completeness, Figures 1 and 2 show the velocity and reflectivity models, and Figure 3 shows a typical shot gather probing the middle (complex) part of the model.

Figure 4 displays the result of RTM applied to the output of this simulation, using the same (reference) velocity. I used 32 buffers in this computation, which for the approximately 8000 time steps in this simulation gave a recomputation ratio of approximately 3. This result required 90 minutes wallclock, also on 120 CPUs, consistent with the analysis given above, which suggests a cost in this case of about five times that of a simulation. No disk i/o was performed during the migration, except for reading in data traces and writing out the final result.

CONCLUSIONS

I have described an checkpointing method due to Griewank (Griewank, 1992; Griewank and Walther, 2000) for practical implementation of the adjoint state method, and its application to reverse time migration. A straightforward implementation of the adjoint state method stores the entire history of the reference state (source wavefield for RTM). The checkpointing method trades floating point operations for some - or even most - of this storage. With optimal choice of checkpoints, the increase in computation time is logarithmic in the number of steps, and the memory complexity is also logarithmic in the number of steps. For problems of modest size, the checkpointing approach eliminates altogether any need for disk i/o during RTM, assuming typical modern workstation or cluster node resources.

Direct comparison of optimal checkpointing performance with alternatives is difficult, due to the wide variety of techniques employed to optimize these other strategies. Notably, commercial implementations of strategy 3 described earlier in this paper appear to employ aggressive decimation of the state history, various types of interpolation, reduced wordlength, and other devices (citable descriptions seem difficult to come by). Of course the seek speed and bandwidth of the disk subsystem also has a dominating influence on the performance of such algorithms. As control of all such factors is impossible, I have restricted myself to pointing out the performance of checkpointing in units of simulations, with examples, rather than attempt a comparison with a strawman implementation of an alternative strategy. It does seem reasonable to suppose however that avoidance of the slowest level of the memory hierarchy might more than compensate for the increase in floating point complexity implicit in checkpointing. Even when disk i/o is necessary (as may the case for 3D RTM, for example), optimal checkpointing requires much less than any alternative approach.

Various technologies appear poised to dramatically increase floating point throughput available to applications like those discussed here, in the next few years: for example, both the IBM Cell Broadband Engine (IBM, 2004) and field programmable gate arrays appear to promise significant speedup over contemporary CPUs. If these gains are indeed realized, they will very likely far outstrip the improvements in NUMA memory access bandwidth, especially at the slow end, i.e disk i/o. Optimal checkpointing's judicious tradeoff of modest increase in floating point complexity for dramatically reduced memory complexity would therefore appear to offer an attractive alternative to contemporary practice for RTM and similar adjoint computations in applied seismology.

ACKNOWLEDGEMENT

This work was supported in part by a Major Research Infrastructure grant from the National Science Foundation (CNS-0421109) and partnerships with AMD and Cray, and by the sponsors of The Rice Inversion Project. My understanding of optimal checkpointing owes a great deal to discussions with Prof. Andreas Griewank and Dr. Joakim Blanch.

REFERENCES

Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E., Lopez, J., O'Hallaron, D., Tu, T., and Urbanic, J., 2003, High resolution forward and inverse earthquake modeling on terascale computers: Supercomputing 03, Association for Computing Machinery, Proceedings.

Albertin, U., Sava, P., Etgen, J., and Maharramov, M., 2006, Adjoint wave equation velocity analysis: 76th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, TOM2.1.

Bamberger, A., Chavent, G., and Lailly, P., Etude mathematique et numerique d'un problem inverse pour l'equation des ondes a une dimension:, Rapport Interne 14, Centre de Mathematiques Appliques, Ecole Polytechnique, , 1977.

Bamberger, A., Chavent, G., and Lailly, P., 1979, About the stability of the inverse problem in 1-d wave equation — application to the interpretation of seismic profiles: Appl. Math. Opt., **5**, 1–47.

Bamberger, A., Chavent, G., Hemon, and Lailly, P., 1982, Inversion of normal incidence seismograms: Geophysics, pages 757–770.

Baysal, E., Kosloff, D. D., and Sherwood, J. W. C., 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Bednar, J. B., and Bednar, C. J., 2006, Two-way vs. one-way: a case study style comparison: 76th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, SPMI1.4.

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

Biondi, B., and Shan, G., 2002, Prestack imaging of overturned reflections by reverse time migration: 72nd Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, 1284–1287.

Blanch, J., Symes, W., and Versteeg, R., 1998, A numerical study of linear inversion in layered viscoacoustic media *in* Keys, R., and Foster, D., Eds., Comparison of Seismic Inversion Methods on a Single Real Dataset:: Society of Exploration Geophysicists.

Bryson, A., and Ho, Y.-C., 1979, Applied Optimal Control: Wiley, New York.

Chavent, G., and Lemmonier, P., 1974, Identification de la non-linéarité d'une équation parabolique quasilinéaire: Applied Mathematics and Optimization, **1**, 121–162.

Cohen, G. C., 2001, Higher order numerical methods for transient wave equations: Springer, New York.

Giering, R., and Kaminski, T., 1998, Recipes for adjoint code construction: ACM Transactions on Mathematical Software, **24**, 437–474.

Griewank, A., and Walther, A., 2000, Algorithm 799: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation: ACM Transactions on Mathematical Software, **26**, 19–45.

Griewank, A., 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.

IBM, 2004, The Cell project at IBM Research, `http://www.research.ibm.com/cell/`.

Khoury, A., Symes, W. W., Williamson, P., and Shen, P., 2006, DSR migration velocity analysis by differential semblance optimization: 76th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, SPMI3.4.

Lailly, P., 1983, The seismic inverse problem as a sequence of before-stack migrations *in* Bednar, J., et al., Eds., Conference on Inverse Scattering: Theory and Applications:: SIAM, 206–220.

Lailly, P., 1984, Migration methods: partial but efficient solutions to the seismic inverse problem *in* Santosa, et al., Eds., Inverse Problems of Acoustic and Elastic Waves:: SIAM.

Levander, A., 1989, Finite difference forward modeling in seismology *in* James, D. E., Ed., The Encyclopedia of Solid Earth Geophysics:: Van Nostrand Reinhold, 410–431.

Mulder, W., and Plessix, R.-E., 2004, A comparison between one-way and two-way wave equation migration: Geophysics, **69**, 1491–1504.

Munk, W., Worcester, P., and Wunsch, C., 1995, Ocean acoustic tomography: Cambridge University Press, Cambridge.

Plessix, R.-E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophys. J. Int., **167**, 495–503.

Pontryagin, L., 1987, Mathematical Theory of Optimal Processes: CRC, New York.

Shen, P., Symes, W., and Stolk, C., 2003, Differential semblance velocity analysis by wave-equation migration: Society of Exploration Geophysicists, 73rd Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, 2135–2139.

Soubaras, R., and Gratacos, B., 2006, Velocity model building by semblance maximization of modulated-shot gathers: 76th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, SVIP1.3.

Symes, W. W., Dussaud, E., Dajani, H., and Padula, A. D., A time-stepping library for simulation-driven optimization:, Technical Report 04-XX, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA, 2004.

Symes, W. W., Padula, A. D., and Scott, S. D., A software framework for the abstract expression of coordinate-free linear algebra and optimization algorithms:, Technical Report 05-12, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA, 2005.

Talagrand, O., 2007, Data assimilation in meteorology and oceanography: Academic Press, New York.

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

Versteeg, R., and G.Grau, Eds., 1991, The Marmousi experience: Proceedings of the eaeg workshop on practical aspects of inversion IFP/Technip, The Hague.

Wang, D. Z., Droegemeier, K. K., and White, L., 1998, The adjoint newton algorithm for large-scale unconstrained optimization in meteorology applications: Computational Optimization and Applications, **10**, 281–318.

Whitmore, N. D., 1983, Iterative depth migration by backwards time propagation: 53rd Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, S10.1.

Yoon, K., Shin, C., Suh, S., Lines, L., and Hong, S., 2003, 3d reverse-time migration using the acoustic wave equation: An experience with the SEG/EAGE data set: The Leading Edge, **22**, 38.

Yoon, K., Marfurt, K., and Starr, E. W., 2004, Challenges in reverse time migration: 74th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, 1057–1060.

APPENDIX: DERIVATION OF GENERAL ADJOINT STATE METHOD

In this Appendix, I derive the discrete-in-time form of the adjoint state method. For an analogous account of the method for continuous time, see (Plessix, 2006).

In addition to evolution (equation (1)), modeling involves a *sampling* step, which extracts a data prediction from the system state. I assume that the *sampling operator* $\mathbf{S}^n$ is linear and independent of control parameters, while possibly depending on the step index. For example, seismic time series sampling records trace samples at specific physical locations, depth imaging involves extracting the zero-offset section at each depth, and so on. If you write the entire time history of the state as a column vector of length $N =$ number of time samples:

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

and define $\mathbf{S}$ to be the row vector of sampling operators:

$$\mathbf{S} = \left(\mathbf{S}^0 \mathbf{S}^1 .... \mathbf{S}^N \right)$$

then the predicted data for control $\mathbf{c}$, denoted $\mathbf{F}[\mathbf{c}]$, is give by $\mathbf{F}[\mathbf{c}] = \mathbf{S}\mathbf{u}$. $\mathbf{F}$ is the *prediction operator* and is generally nonlinear in $\mathbf{c}$, even for linear evolutions.

The *adjoint state method*, described in display (2), computes the adjoint action of the linearized prediction operator, *via* a recursion similar to the evolution defining the state. The need for this computation arises in computing the *gradient* of an *objective function* of the control $J[\mathbf{c}]$, which is actually a function E of the predicted data:

$$J[\mathbf{c}] = E[\mathbf{F}[\mathbf{c}]]$$

For output least squares ("waveform") inversion, E computes the mean square difference between predicted and observed data. For migration velocity analysis, E computes the semblance or differential semblance or some other function of the depth extrapolated reflected field.

The chain rule and the definition of gradient give

$$\nabla J[\mathbf{c}]^T \delta\mathbf{c} = DJ[\mathbf{c}]\delta\mathbf{c} = (\nabla E[\mathbf{F}[\mathbf{c}]])^T D\mathbf{F}[\mathbf{c}]\delta\mathbf{c} = (D\mathbf{F}[\mathbf{c}]^T \nabla E[F[\mathbf{c}]])^T \delta\mathbf{c},$$

in which D denotes the derivative, i.e. $DF[\mathbf{c}]$ is the Jacobian matrix of F at $\mathbf{c}$. Since this relation must hold for every perturbation $\delta\mathbf{c}$, it follows that

$$\nabla J[\mathbf{c}] = D\mathbf{F}[\mathbf{c}]^T \nabla E[\mathbf{F}[\mathbf{c}]]$$

The gradient of the error function E tends to be very easy to compute, so the issue posed by the preceding formula is the application of the transpose Jacobian $DF[\mathbf{c}]^T$.

From its definition,

$$D\mathbf{F}[\mathbf{c}]\delta\mathbf{c} = \mathbf{S}\delta\mathbf{u} \tag{6}$$

where the perturbation ("Born") field $\delta\mathbf{u}$ has its own evolution law, derived from that for $\mathbf{u}$:

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

$$\left(\begin{array}{l} 0 \\ D\mathbf{H}[\mathbf{c}]\mathbf{u}^0 \\ \ddots \\ D\mathbf{H}[\mathbf{c}]\mathbf{u}^{N-1} \end{array}\right)\delta\mathbf{c}. \tag{7}$$

Denote by $\mathcal{H}$ the first matrix on the right hand side of the preceding equation, and by $\mathcal{H}'$ the second. Note that $\mathcal{H}$ is *lower triangular*, and so defines a forward evolution in step index (representing time or depth or...).

The preceding equation may be abbreviated as

$$\delta\mathbf{u} = \mathcal{H}\delta\mathbf{u} + \mathcal{H}'\delta\mathbf{c}$$

which when solved for $\delta\mathbf{u}$ and inserted in equation (6) yields an expression for the Jacobian:

$$D\mathbf{F}[\mathbf{c}] = \mathbf{S}(I-\mathcal{H})^{-1}\mathcal{H}'\delta\mathbf{c}$$

whence

$$D\mathbf{F}[\mathbf{c}]^T = (\mathcal{H}')^T(I-\mathcal{H}^T)^{-1}\mathbf{S}^T \tag{8}$$

The adjoint state method consists in unwinding the formula (8) to reveal a recursion for the action of the transposed Jacobian on a (data-like) vector $\mathbf{r}$ (which, for the gradient calculation, will be $\nabla E[\mathbf{F}[\mathbf{c}]]$). The algorithm proceeds as follows:

Step 1: Apply the adjoint sampling operator:

$$\mathbf{r} \mapsto \mathbf{S}^T\mathbf{r} = \left(\begin{array}{l} (\mathbf{S}^0)^T \\ (\mathbf{S}^1)^T \\ \vdots \\ (\mathbf{S}^N)^T \end{array}\right)\mathbf{r}.$$

Since $\mathbf{S}^n$ 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*

$$(I-\mathcal{H}^T)\mathbf{w} = \mathbf{S}^T\mathbf{r}$$

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

$$\mathbf{w}^N = 0;\ \mathbf{w}^n = (\mathbf{H}^{n+1}[\mathbf{c}])^T\mathbf{w}^{n+1} + (\mathbf{S}^n)^T\mathbf{r},\ n = N-1, ..., 0 \tag{9}$$

Step 3 ("imaging"): apply the operator $\mathcal{H}'$ to $\mathbf{w}$, which amounts to computing

$$\sum_{n=0}^{N-1}(D\mathbf{H}^n[\mathbf{c}]\mathbf{u}^n)^T\mathbf{w}^{n+1} \tag{10}$$

It is natural to accumulate the sum in equation (10) term-by-term as the factors $\mathbf{w}^n$ are produced in the backpropagation loop (9). With the notation

$$A^n[\mathbf{c}, \mathbf{u}] \equiv D\mathbf{H}^n[\mathbf{c}]\mathbf{u}$$

the representation (2) of the adjoint computation is established.

The definition of an operator adjoint depends on the inner products used in domain and range spaces. The computation outlined here assumes the unscaled Euclidean norm in both domain and range. If scaled norms are used in domain or range of the modeling operator these scale factors must also be taken into account in the definition of the adjoint. For example, the Euclidean inner product might be scaled to create a quadrature rule for the integral or L^2 inner product of functions. This sort of scaling makes the inner products of sampled grid functions essentially independent of grid size. However the cell volumes used to scale the Euclidean inner product must then be included in the definition of the adjoint.

To verify the proper inclusion of scale factors and realization of other facets of the calculation, the author strongly recommends that any adjoint state implementation be subjected to the obvious null test: for random choices of control and data perturbations $\delta\mathbf{c}$ and $\delta\mathbf{d}$, ensure that

$$\langle D\mathbf{F}[\mathbf{c}]\delta\mathbf{c}, \delta\mathbf{d}\rangle_R \simeq \langle \delta\mathbf{c}, D\mathbf{F}[\mathbf{c}]^T\delta\mathbf{d}\rangle_D$$

in which $\langle\cdot,\cdot\rangle_D$ and $\langle\cdot,\cdot\rangle_R$ are domain and range (of $\mathbf{F}$) inner products respectively. The difference between the two sides should be a modest multiple of machine precision.

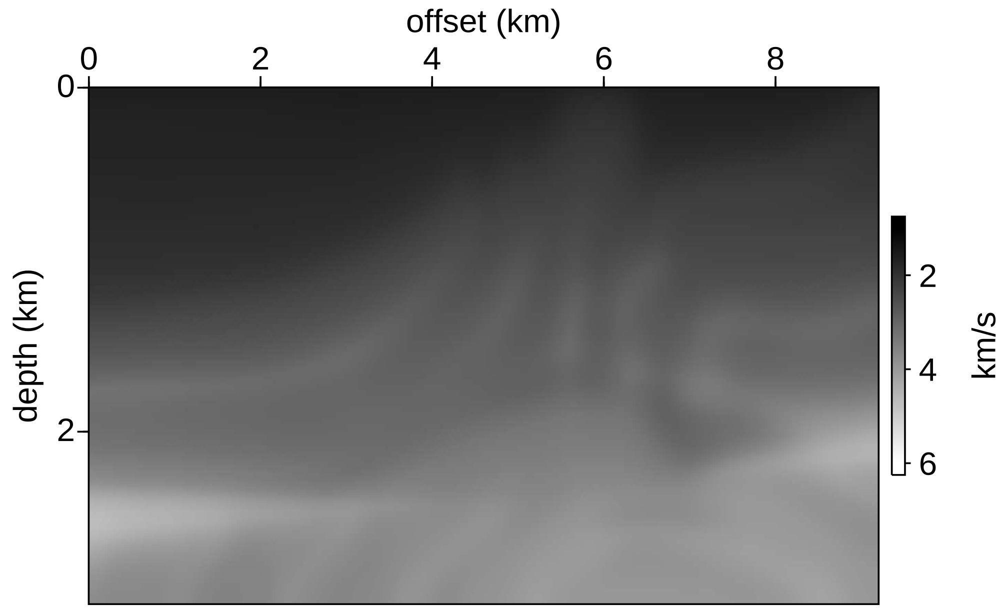
CAPTIONS

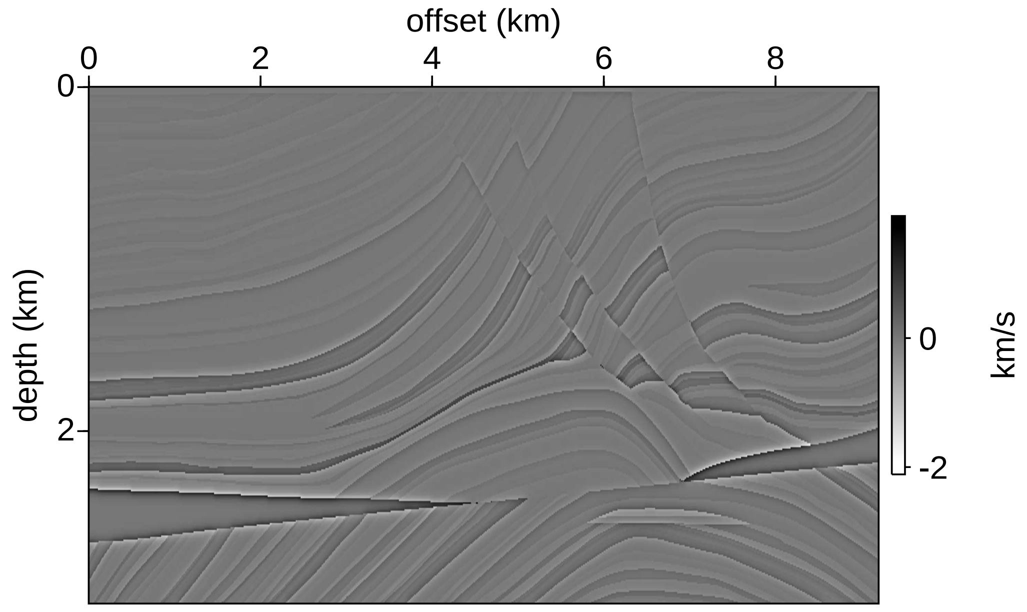
Figure 1. Marmousi velocity model, smoothed by a 160 m tapered moving average.

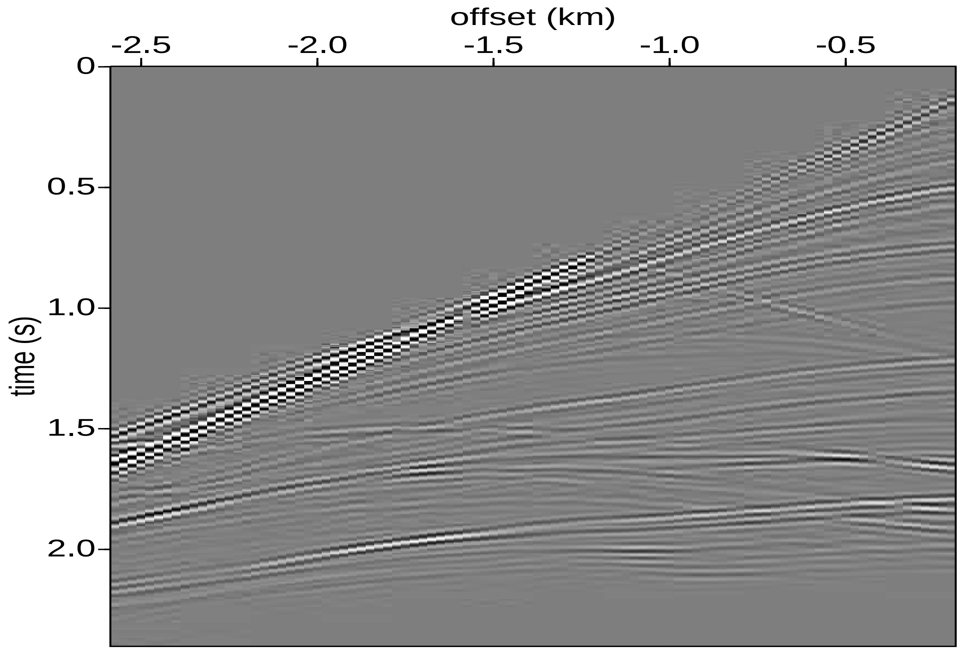
Figure 2. Reflectivity model derived from Marmousi velocity model, by subtracting a 40 m tapered moving average.

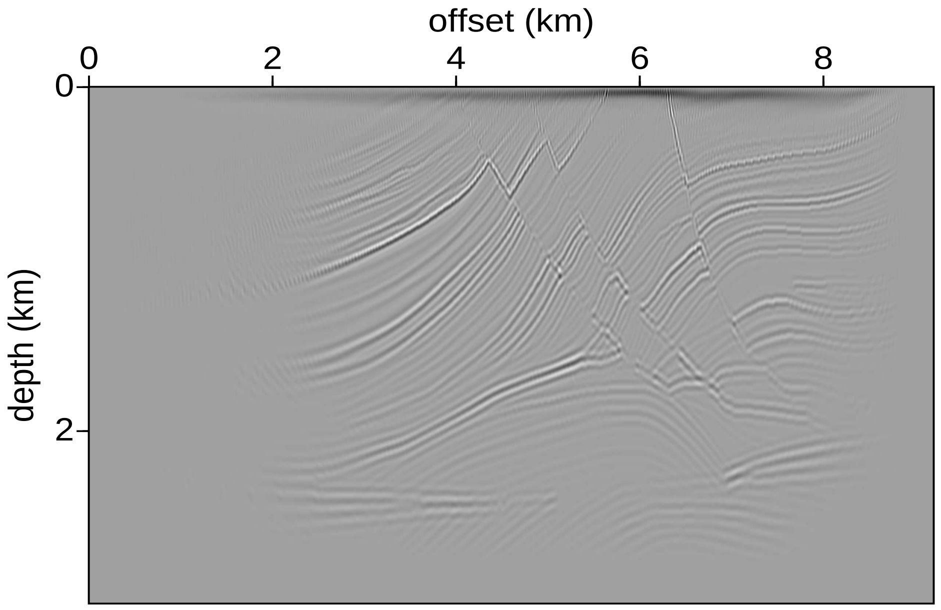
Figure 3. Linearized ("Born") simulation of shot gather at 7.5 km from left edge of model. Source depth is 8 m, receiver depth 12 m, group interval 25 m, near offset -200 m, 96 traces per shot. 240 shots were simulated at 25 m intervals, the leftmost shot sited 3 km from the left edge of the model. Source is point radiator, zero phase bandpass filter 5-13-40-55 Hz.

Figure 4. Reverse time migration of data described in preceding caption.


offset (km)
0
2
4
6
8
0
2
depth (km)
2
4
6
km/s


offset (km)
0
2
4
6
8
0
2
depth (km)
0
-2
km/s


offset (km)
-2.5
-2.0
-1.5
-1.0
-0.5
0
0.5
1.0
1.5
2.0
time (s)


offset (km)
0
2
4
6
8
0
2
depth (km)