Journal of Computational Physics **157,** 234–255 (2000)
doi:10.1006/jcph.1999.6373, available online at http://www.idealibrary.com on IDEAL®

Efficient Calculation of Jacobian and Adjoint Vector Products in the Wave Propagational Inverse Problem Using Automatic Differentiation[1]

Thomas F. Coleman,* Fadil Santosa,† and Arun Verma‡

*Department of Computer Science and Center for Applied Mathematics, Cornell University, Ithaca, New York 14850; †Minnesota Center for Industrial Mathematics, School of Mathematics, University of Minnesota, Minneapolis, Minnesota 55455; ‡Cornell Theory Center, Cornell University, Ithaca, New York 14850
E-mail: coleman@tc.cornell.edu, santosa@math.umn.edu, verma@cs.cornell.edu

Received February 18, 1999; revised July 27, 1999

Wave propagational inverse problems arise in several applications including medical imaging and geophysical exploration. In these problems, one is interested in obtaining the parameters describing the medium from its response to excitations. The problems are characterized by their large size, and by the hyperbolic equation which models the physical phenomena. The inverse problems are often posed as a nonlinear data-fitting where the unknown parameters are found by minimizing the misfit between the predicted data and the actual data. In order to solve the problem numerically using a gradient-type approach, one must calculate the action of the Jacobian and its adjoint on a given vector. In this paper, we explore the use of automatic differentiation (AD) to develop codes that perform these calculations. We show that by exploiting structure at 2 scales, we can arrive at a very efficient code whose main components are produced by AD. In the first scale we exploite the time-stepping nature of the hyperbolic solver by using the "Extended Jacobian" framework. In the second (finer) scale, we exploit the finite difference stencil in order to make explicit use of the sparsity in the dependence of the output variables to the input variables. The main ideas in this work are illustrated with a simpler, one-dimensional version of the problem. Numerical results are given for both one- and two- dimensional problems. We present computational templates that can be used in conjunction with optimization packages to solve the inverse problem.

[1] This research is sponsored in part by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under Grants DE-FG02-97ER25013 and DF-FG02-94ER25225, and also by the National Science Foundation Grant DMS 9503114, and by the Air Force Office of Scientific Research Grant F49620-95-I-0305. We acknowledge helpful discussions with William Symes, who has a similar on-going effort on automatic differentiation as ours [15]. Some of the ideas in this work were inspired by his presentation at the Institute for Mathematics and its Applications, Minnesota, in July 1997.

0021-9991/00 $35.00

Key Words: automatic differentiation; wave propagation; inverse problems seismic inversion; hyperbolic PDEs; adjoints; finite difference computation; stencils; Jacobian and adjoint products.

1. INTRODUCTION

In the type of wave propagational inverse problems under consideration, the goal is to determine parameters, such as sound speed distribution and density distribution, from measured data, which are collected at a set of receivers. Figure 1 explains the situation. An incident disturbance is generated, as it travels in the unknown medium and produces reflections and refractions. This information is collected at receivers placed at a set of locations. Several such experiments are carried out for a set of incident disturbances. The inverse problem is to determine properties of the unknown medium from the set of measured response.

Problems of this type arise in several applications including geophysical exploration and medical imaging. A common feature in these applications is that the problem is very large. Typically, the number of unknowns and equations could be in the range of 10^3 to 10^6. Often, the most convenient way to solve this type of inverse problem is to pose it as an optimization problem, either using nonlinear least-squares [16, 12] or another approach specialized to take advantage of the properties afforded by the particular application [14, 1]. In any event, what one will need for computation is derivative information concerning the relation between medium parameters and data. Because of the size of the problem, we cannot compute and store the entire Jacobian of the function, but rather, we must find ways of computing the action of the Jacobian and its transpose on a given vector, or the so-called direct and adjoint products.

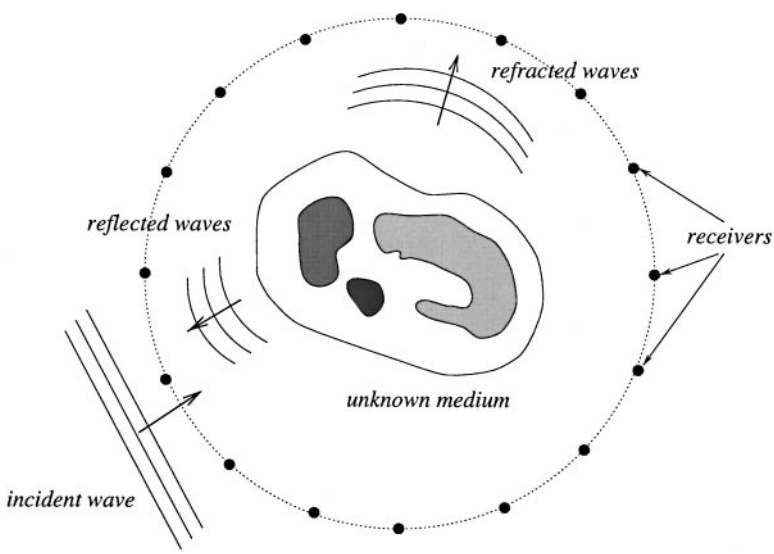

FIG. 1. In this figure, the problem is to identify the unknown medium. An incident wave is generated, and as it travels into the medium being probed, reflected and refracted signals are generated. These are captured at the receivers. Several such experiments are carried out for a set of incident disturbances. The inverse problem is to find the properties of the unknown medium from the collected data.

The goal of this work is to show that efficient calculation of the direct and adjoint product is possible. The approach we take is to use automatic differentiation (AD) while exploiting structure to the extent possible. We emphasize that without taking advantage of structure, a direct application of current AD technology to the codes simulating the wave phenomena will lead to memory problems.

As we will show in the next section, the wave propagation can be modeled effectively using time-stepping finite difference schemes. The time-stepping nature of the scheme can be exploited using the general extended Jacobian framework [3, 4]. The spatial discretization by finite differences reveals further structure. Each finite difference stencil encodes the dependence of a computed intermediate variable on other variables. In particular, it shows that there is an inherent sparsity in the Jacobian. A combination of these structure exploitations allows us to overcome the problem posed by size, and its consequence on memory requirements.

In our implementation, we apply AD on the finite difference stencils and use the resulting codes to assemble a procedure for computing the Jacobian and adjoint vector products. The resulting code is as efficient as those that are obtained by directly performing summations-by-parts calculation on the simulation program. The advantage here is that we have avoided the error-prone and tedious procedure [13]. Instead, we can view the code writing process at a higher level, leaving the most difficult parts to AD.

The plan of this article is as follows. We proceed with a short introduction to the inverse problem for acoustic waves. The model for the physics and its numerical discretization are described in Section 2. In Section 3, we review the extended Jacobian framework and show how it can be used for our problem. We will provide templates for calculating the adjoint-vector product. The stencil approach and its implementation is presented in Section 4. We will also show how the stencil can be described at a higher level as projections. Templates for calculating Jacobian and adjoint vector products that use stencils are given. Section 5 summarizes our experience with this method of computation. A final section contains concluding remarks.

2. AUTOMATIC DIFFERENTIATION BACKGROUND

Automatic differentiation is based on the fact that all computer programs, no matter how complicated, use a finite set of *elementary functions* as defined by the programming language. The function computed by the program is simply a composition of these elementary functions. The partial derivatives of the elementary functions are known, and the overall derivatives are computed using the chain rule; this process is known as automatic differentiation [9].

Abstractly, the program to evaluate the solution u (an m-vector) as a function of x (generally a n-vector) has the form

$$
\begin{aligned}
x &\equiv (x_1, x_2, \ldots, x_n) \\
&\quad\downarrow \\
z &\equiv (z_1, z_2, \ldots, z_p), \qquad p \gg m + n \\
&\quad\downarrow \\
y &\equiv (y_1, y_2, \ldots, y_m),
\end{aligned}
$$

where the intermediate variables z are related through a series of these elementary functions which may be unary,

$$z_k = f^k_{\text{elem}}(z_i), \qquad i < k,$$

consisting of operations such as $(-, \text{pow}(\cdot), \sin(\cdot), \ldots)$ or binary,

$$z_k = f^k_{\text{elem}}(z_i, z_j), \qquad i < k,\ j < k.$$

such as $(+, /, \ldots)$.

There are a number of cases when the elementary function is not differentiable (e.g., $f^k_{\text{elem}}(z_i) = \text{abs}(z_i)$ or $f^k_{\text{elem}}(z_i, z_j) = \max(z_i, z_j)$). Sophisticated heuristic techniques are developed to treat these cases. For more details consult [9].

Automatic differentiation has two basic modes of operations, the forward mode and the reverse mode. In the forward mode the derivatives are propagated throughout the computation using the chain rule, e.g., for the elementary step $z_k = f^k_{\text{elem}}(z_i, z_j)$ the intermediate derivative, dz_k/dx, can be propagated in the forward mode as

$$\frac{dz_k}{dx} = \frac{\partial f^k_{\text{elem}}}{\partial z_i}\frac{dz_i}{dx} + \frac{\partial f^k_{\text{elem}}}{\partial z_j}\frac{dz_j}{dx}.$$

This chain rule based computation is done for all the intermediate variables z and for the output variables u, finally yielding the derivative $\frac{du}{dx}$.

The reverse mode computes the derivatives du/dz_k for all intermediate variables backwards (i.e., in the reverse order) through the computation. For example, for the elementary step $z_k = f^k_{\text{elem}}(z_i, z_j)$, the derivatives are propagated as

$$\frac{du}{dz_i} = \frac{\partial f^k_{\text{elem}}}{\partial z_i}\frac{du}{dz_k} \qquad \text{and} \qquad \frac{du}{dz_j} = \frac{\partial f^k_{\text{elem}}}{\partial z_j}\frac{du}{dz_k}.$$

At the end of computation of the reverse mode the derivative $\frac{du}{dx}$ will be obtained.

The forward and reverse modes can be used to compute the direct and the adjoint products, Jv and $J^T v$ given a vector v, where J is the Jacobian of a nonlinear mapping [9]. Both these computations require time proportional to one function evaluation, with the adjoint product being approximately twice as costly as the direct mode. The Hessian-vector product Hv can also be computed via AD in time proportional to one function evaluation.

3. INVERSE PROBLEMS AND NUMERICAL MODELING

3.1. *One-Dimensional Problem*

Consider a bar or string of length L whose sound speed is location dependent. Let $u(x, t)$ represent a measure of the disturbance at time t and location x. Then u satisfies the wave equation

$$u_{tt} = c^2(x)u_{xx} \qquad \text{for } 0 < x < L, \tag{1a}$$

where $c(x)$ is the sound speed of the medium. We assume that the medium is quiescent at $t = 0$,

$$u(x, 0) = 0, \qquad \text{and} \qquad u_t(x, 0) = 0, \quad 0 \le x \le L. \tag{1b}$$

Disturbance is introduced at the boundary $x = 0$ as a Neumann boundary condition

$$u_x(0, t) = f(t), \qquad \text{for } t > 0. \tag{1c}$$

We will assume that $f(t)$ is compactly supported away from $t = 0$. On the right end, we assume a radiation boundary condition

$$[u_t - c(x)u_x]|_{x=L} = 0. \tag{1d}$$

We are given $u(0, t) = g(t)$ for $0 < t < T$. The problem is to find the unknown $c(x)$.

A convenient way to view the problem is to define the forward map as one that associates a given $c(x)$ with a boundary data $u(0, t)$. Let

$$A[c](t) := u(0, t), \qquad 0 < t \le T,$$

where it is understood that the evaluation of $A[c](\cdot)$ is through the initial-boundary value problem (IBVP) in (1). A least-squares formulation of this problem is to solve the minimization

$$\min_{c(x)} \int_0^T |A[c](t) - g(t)|^2 \, dt. \tag{2}$$

A typical solution to the above nonlinear least squares problem requires the knowledge of the gradient of the above functional. This translates to computing the adjoints of the function $A[c](\cdot)$ [12, 13]. The first step in the numerical computation of the adjoints is to discretize the problem.

A common discretization for this problem is to use finite difference methods. Let

$$u_i^k \approx u(x_i, t_k), \qquad \text{where } x_i = i\Delta x, i = 0 : n, t_k = k\Delta t, k = 0 : m,$$

and $\Delta x = L/n$, and $\Delta t = \lambda \Delta x$ for some $\lambda > 0$. A second order finite difference is chosen. The partial differential equation in (1a) is replaced by

$$u_i^{k+1} = 2\big(1 - \lambda^2 c_i^2\big)u_i^{k-1} - u_i^k + \lambda^2 c_i^2\big(u_{i+1}^k + u_{i-1}^k\big), \qquad \text{for } i = 1 : n-1, k \ge 0. \tag{3a}$$

We use the initial conditions

$$u_i^{-1} = u_i^0 = 0. \tag{3b}$$

We discretize the boundary conditions as

$$u_0^{k+1} = 2\big(1 - \lambda^2 c_0^2\big)u_0^k - u_0^{k-1} + 2\lambda^2 c_0^2 u_1^k - 2\, f^k \lambda^2 c_0^2 \Delta x, \tag{3c}$$

for the inhomogeneous Neumann condition on the left end, and

$$u_n^{k+1} = u_n^k - c_n \frac{\Delta t}{\Delta x}\big(u_n^k - u_{n-1}^k\big) \tag{3d}$$

for the radiation boundary condition on the right. The discrete version of the forward map is obtained by running the finite difference forward in time and recording the left end value for u_i^k, that is,

$$A[c]^k := u_0^k.$$

A way to describe the function evaluation is through a vector notation. Let us write the vectors $\mathbf{u}^k = [u_0^k, u_1^k, \ldots, u_n^k]^T$ and $\mathbf{c} = [c_0, c_1, \ldots, c_n]^T$. Then the finite difference scheme amounts to

$$\mathbf{u}^{k+1} = F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1}), \qquad \text{with } \mathbf{u}^{-1} = \mathbf{u}^0 = \mathbf{0}. \tag{4}$$

The forward map from $\mathbf{c}$ to $A[\mathbf{c}]$ is given by, letting $\mathbf{e}_1 = [1, 0, \ldots, 0]^T$,

$$A[\mathbf{c}]^k = \mathbf{e}_1^T \mathbf{u}^k, \qquad \text{for } k = 1 : m. \tag{5}$$

The inverse problem is to solve for $\mathbf{c}$ in

$$\min_{\mathbf{c}} \|A[\mathbf{c}] - \mathbf{g}\|^2, \tag{6}$$

where $\mathbf{g}$ is a data vector corresponding to a measurement.

3.2. *Two-Dimensional Problem*

The two-dimensional problem is motivated by a problem is acoustic imaging of human tissues. The geometry of the problem has been described in the previous section, and elsewhere [11]. Here we give a mathematical model of the physics.

Because any computational domain is necessarily finite, we will consider a box $\Omega := [-a, a] \times [-a, a]$. Letting $u(x, y, t)$ represent the excess pressure, a model for acoustics is given by the partial differential equation

$$u_{tt} = c(x, y)^2 \Delta u + f(x, y, t) \qquad \text{in } \Omega, t > 0. \tag{7a}$$

Here $c(x, y)$ represents the unknown soundspeed distribution, while $f(x, y, t)$ is a known acoustic source. Initially, the system is at rest, hence

$$u(x, y, 0) = u_t(x, y, 0) = 0. \tag{7b}$$

We need to simulate an unbounded medium with a bounded domain. In the unbounded medium, we would have a boundary condition for $|x^2 + y^2|$ large that amounts to saying that waves which are sufficiently far away from the origin and traveling outward will be radiated to infinity. To simulate the unbounded medium, we assume that c is constant near the boundary of Ω and apply the Engquist–Majda boundary conditions [6] along the flat parts of $\partial\Omega$ (and a modification of Engquist–Majda at the corners of $\partial\Omega$). For points away from the corners, the boundary condition is given by

$$c u_{xt} - u_{tt} + \frac{c^2}{2} u_{yy} = 0 \qquad \text{for } \{x = \pm a; |y| < a\}, \tag{7c}$$

$$c u_{yt} - u_{tt} + \frac{c^2}{2} u_{xx} = 0 \qquad \text{for } \{y = \pm a; |x| < a\}. \tag{7d}$$

Let R represent the collection of coordinate points where receivers have been placed to record u. Thus,

$$R = \{(x_r, y_r) = (\rho \cos\theta_r, \rho \sin\theta_r), r = 1 : p\}$$

for some $\rho > 0$. The forward map is given by

$$A[c;\ f]_r := u(x_r, y_r, t).$$

The source term $f(x, y, t)$ is assumed to be null for $t = 0$. We will view the forward map $A[\ ]$ as dependent on c and parameterized by f.

The nonlinear least-squares formulation is given by

$$\min_c \sum_l \sum_{r=1}^{p} \int_0^T |A[c;\ f_l]_r(t) - g_{rl}(t)|^2,$$

where $g_{rl}(t)$ is the measured response at location (x_r, y_r) for the source $f_l(x, y, t)$.

Discretization of (7) is quite straightforward. The only tricky part comes in discretizing the Enquist–Majda boundary condition. Letting $(x_i, y_j) = (i\Delta, j\Delta)$, $-n \le i \le n$, and $-n \le j \le n$, we discretized the domain Ω by a regular mesh of size $\Delta = a/n$. Time is discretized as in the 1-D case: $t_k = k\Delta t$ for $k = 0 : m$.

Let the $(2n+1)^2$ vector $\mathbf{u}^k$ represent the value for $u(x, y, t)$ at the node points at time t_k. The finite difference scheme can be written in shorthand as

$$\mathbf{u}^{k+1} = F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1}), \qquad \text{for } k = 1 : m,$$

with $\mathbf{u}^{-1} = \mathbf{u}^0 = \mathbf{0}$. The discrete forward map evaluates $\mathbf{u}$ at each receiver, thus

$$A[\mathbf{c};\ f]^k = T\mathbf{u}^k,$$

where T is a matrix of size p-by-$(2n+1)^2$ and its function is to "grab" values of $\mathbf{u}$ at time step k at the recievers. In place of the integration in the nonlinear least-squares, we have

$$\min_{\mathbf{c}} \sum_l \sum_{r=1}^{p} \sum_{k=1}^{m} \left| A[c;\ f_l]_r^k - g_{rl}^k \right|^2. \tag{8}$$

Here g_{rl}^k is the measured response at receiver r at time step k when the excitation is f_l.

4. THE EXTENDED JACOBIAN FRAMEWORK

We restrict our discussion to the 1-D problem for clarity of presentation. The prescription for computing Jacobian vector and adjoint vector products for the more complex 2-D problem follows the same lines as for the 1-D problem. An algorithm for the forward map for the 1-D case is

$$\begin{aligned}
&\mathbf{u}^{-1} = \mathbf{u}^0 = 0 \\
&\texttt{for } k = 0 : m - 1 \\
&\qquad \mathbf{u}^{k+1} = F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1}) \\
&\qquad h_{k+1} = \mathbf{e}_1^T \mathbf{u}^{k+1} \\
&\texttt{end}
\end{aligned} \tag{9}$$

We use the notation $h_k = A[\mathbf{c}]_k$. Thus the function in question is the mapping from $\mathbf{c}$ to $\mathbf{h} = (h_1, h_2, \ldots, h_m)^T$.

We can give an alternate description of this mapping by enumerating through the loop

$$\begin{aligned}
&\mathbf{u}^1 = F(\mathbf{c}, \mathbf{u}^0, \mathbf{u}^{-1})\\
&\mathbf{u}^2 = F(\mathbf{c}, \mathbf{u}^1, \mathbf{u}^0)\\
&\vdots\\
&\mathbf{u}^m = F(c, \mathbf{u}^{m-1}, \mathbf{u}^{m-2})\\
&\mathbf{h} = \mathbf{e}_1\mathbf{e}_1^T\mathbf{u}^1 + \mathbf{e}_2\mathbf{e}_1^T\mathbf{u}^2 + \cdots + \mathbf{e}_m\mathbf{e}_1^T\mathbf{u}^m
\end{aligned}$$

We call this the extended function. The extended function allows for an easy way to compute the Jacobian and its transpose. Formally, the directional derivative of $\mathbf{h}$ in the direction $d\mathbf{c}$, i.e., the Jacobian-vector product, is given by the calculation

$$\begin{aligned}
&d\mathbf{u}^{-1} = d\mathbf{u}^0 = 0\\
&\texttt{for } k = 0 : m-1\\
&\quad d\mathbf{u}^{k+1} = \partial_1 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})\, d\mathbf{c} + \partial_2 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})\, d\mathbf{u}^k + \partial_3 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})\, d\mathbf{u}^{k-1} \qquad (10)\\
&\quad dh_{k+1} = \mathbf{e}_1^T\, d\mathbf{u}^{k+1}\\
&\texttt{end}
\end{aligned}$$

The matrices $\partial_1 F$, $\partial_2 F$, and $\partial_3 F$ are Jacobians of the function F with respect to the first, second, and third variables. Therefore, they are $(n+1)$-by-$(n+1)$ matrices. In a computer program, we would simply define $F(\mathbf{c}, \cdot, \cdot)$ and use AD to either compute these matrices or produce subprograms that calculate the action of these matrices on given vectors. The desired directional derivative (Jacobian times vector $d\mathbf{c}$) is $d\mathbf{h} = (dh_1, dh_2, \ldots, dh_m)^T$.

4.1. *Adjoint Computation via Linear Algebra*

The above calculation can be defined as a set of matrix equations through the use of the extended Jacobian framework [3, 4]. Let

$$dU = \begin{bmatrix} d\mathbf{u}^1 \\ d\mathbf{u}^2 \\ \vdots \\ d\mathbf{u}^m \end{bmatrix}.$$

Define the $m(n+1) \times m(n+1)$ matrix

$$M = \begin{bmatrix}
-I & 0 & \ddots & \ddots & 0 & 0\\
\partial_2 F(\mathbf{c}, \mathbf{u}^1, \mathbf{u}^0) & -I & 0 & \ddots & \ddots & 0\\
\partial_3 F(\mathbf{c}, \mathbf{u}^2, \mathbf{u}^1) & \partial_2 F(\mathbf{c}, \mathbf{u}^2, \mathbf{u}^1) & -I & \ddots & \ddots & 0\\
0 & \ddots & \ddots & \ddots & -I & 0\\
0 & \ddots & \ddots & \partial_3 F(\mathbf{c}, \mathbf{u}^{m-1}, \mathbf{u}^{m-2}) & \partial_2 F(\mathbf{c}, \mathbf{u}^{m-1}, \mathbf{u}^{m-2}) & -I
\end{bmatrix}$$

and $m(n+1) \times (n+1)$ matrix

$$B = \begin{bmatrix} \partial_1 F(\mathbf{c}, \mathbf{u}^0, \mathbf{u}^{-1}) \\ \partial_1 F(\mathbf{c}, \mathbf{u}^1, \mathbf{u}^0) \\ \vdots \\ \partial_1 F(\mathbf{c}, \mathbf{u}^{m-1}, \mathbf{u}^{m-2}) \end{bmatrix}$$

and $m \times m(n+1)$ matrix

$$T = \begin{bmatrix} \mathbf{e}_1 \mathbf{e}_1^T & \mathbf{e}_2 \mathbf{e}_1^T & \dots & \mathbf{e}_m \mathbf{e}_1^T \end{bmatrix}.$$

Then

$$-M\, dU = B\, d\mathbf{c}, \qquad \text{and} \qquad d\mathbf{h} = T\, dU.$$

From the above, we can solve for dU and write

$$d\mathbf{h} = -T M^{-1} B\, d\mathbf{c}, \tag{11}$$

which encapsulates the Jacobian-vector product calculation in (10).

To obtain a formula for the adjoint-vector product calculation, we start by formally taking the adjoint of (11). Let $\mathbf{p}$ be the result of multiplying vector $\mathbf{q}$ by the adjoint of the Jacobian. Then from (11)

$$\mathbf{p} = -B^T M^{-T} T^T \mathbf{q}. \tag{12}$$

The matrices B, M, and T should not be computed explicitly, but rather, this formalism is used to generate an efficient algorithm for finding $\mathbf{p}$ given $\mathbf{q}$.

Let $\mathbf{q}$ be an m-vector. Then $Q = T^T \mathbf{q}$ is an $m(n+1)$-vector. From (12) if we let $Y = -M^{-T} T^T \mathbf{q}$, then

$$-M^T Y = Q \qquad \text{and} \qquad \mathbf{p} = B^T Y. \tag{13}$$

By exploiting the structures of M and B, we can come up with an efficient algorithm to find $\mathbf{p}$ for a given $\mathbf{q}$. Because of the lower-triangular structure of M, we never need to invert any matrices. The algorithm starts by chopping up Q into m separate pieces,

$$Q = \begin{bmatrix} \mathbf{q}^1 \\ \mathbf{q}^2 \\ \vdots \\ \mathbf{q}^m \end{bmatrix}$$

and similarly for Y. Then, according to (13), we can calculate $\mathbf{p}$ by

$$
\begin{aligned}
&\mathbf{y}^m = \mathbf{q}^m;\ \mathbf{p} = \partial_1 F(\mathbf{c}, \mathbf{u}^{m-1}, \mathbf{u}^{m-2})^T \mathbf{y}^m \\
&\mathbf{y}^{m-1} = \mathbf{q}^{m-1} + \partial_2 F(\mathbf{c}, \mathbf{u}^{m-1}, \mathbf{u}^{m-2})^T \mathbf{y}^m \\
&\mathbf{p} = \mathbf{p} + \partial_1 F(\mathbf{c}, \mathbf{u}^{m-2}, \mathbf{u}^{m-3})^T \mathbf{y}^{m-1} \\
&\texttt{for } k = m-2 : -1 : 1 \\
&\quad \mathbf{y}^k = \mathbf{q}^k + \partial_2 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})^T \mathbf{y}^{k+1} + \partial_3 F(\mathbf{c}, \mathbf{u}^{k+1}, \mathbf{u}^k)^T \mathbf{y}^{k+2} \\
&\quad \mathbf{p} = \mathbf{p} + \partial_1 F(\mathbf{c}, \mathbf{u}^{k-1}, \mathbf{u}^{k-2})^T \mathbf{y}^k \\
&\texttt{end}
\end{aligned}
\tag{14}
$$

Note that we have adjoints/transpose of $\partial_1 F(\mathbf{c}, \cdot, \cdot)$, $\partial_2 F(\mathbf{c}, \cdot, \cdot)$ and $\partial_3 F(\mathbf{c}, \cdot, \cdot)$. These adjoints can be computed explicitly if we have matrices $\partial_1 F$, $\partial_2 F$, and $\partial_3 F$ or we can resort to AD to produce subprograms that compute their action on given vectors.

Note that in the algorithm (14), we need to have available values of the fields $\mathbf{u}^k$ for all indices k. Depending on the size of the problem, it may be more efficient to store only values of $\mathbf{u}^k$ for some indices $k \in K$ and use (9) to generate the field for other indices $k \notin K$. An efficient method to do this is discussed in [8].

4.2. *Adjoint Computation via Adjoint Variables*

We give an alternate derivation of the algorithm in (14) which is based on using adjoint variables. Consider a simple calculation involving the following 3 steps. The input variable is c and the output variable is u_3; u_0 is a parameter. The steps are

$$
\begin{aligned}
u_1 &= f(c, u_0, 0) \\
u_2 &= f(c, u_1, u_0) \\
u_3 &= f(c, u_2, u_1).
\end{aligned}
$$

We can view this as an extended function. The Jacobian calculation is

$$
\begin{aligned}
du_1 &= \partial_1 f(c, u_0, 0)\, dc \\
du_2 &= \partial_1 f(c, u_1, u_0)\, dc + \partial_2 f(c, u_1, u_0)\, du_1 \\
du_3 &= \partial_1 f(c, u_2, u_1)\, dc + \partial_2 f(c, u_2, u_1)\, du_2 + \partial_3 f(c, u_2, u_1)\, du_1.
\end{aligned}
$$

Therefore, the Jacobian (in this case, derivative) can be identified as J from the output $du_3 = J dc$. This is a forward mode computation.

Let the adjoint variables be p and v_3 so that we formally have $p = J^T v_3$. If we view du_1 and du_2 as intermediate variables, then we can associate to them adjoint variables v_1 and v_2. From the third equation in the adjoint calculation, we can formally write

$$
\begin{bmatrix} p \\ v_2 \\ v_1 \end{bmatrix} = \begin{bmatrix} \partial_1 f(c, u_2, u_1) \\ \partial_2 f(c, u_2, u_1) \\ \partial_3 f(c, u_2, u_1) \end{bmatrix} v_3
$$

and from the second,

$$\begin{bmatrix} p \\ v_1 \end{bmatrix} = \begin{bmatrix} \partial_1 f(c, u_1, u_0) \\ \partial_2 f(c, u_1, u_0) \end{bmatrix} v_2,$$

and from the first,

$$p = \partial_1 f(c, u_0, 0) v_1.$$

The contributions to each of the adjoint variables are summed over each operation, hence

$$\begin{aligned} v_2 &= \partial_2 f(c, u_2, u_1) v_3 \\ v_1 &= \partial_3 f(c, u_2, u_1) v_3 + \partial_2 f(c, u_1, u_0) v_2 \\ p &= \partial_1 f(c, u_2, u_1) v_3 + \partial_1 f(c, u_1, u_0) v_2 + \partial_1 f(c, u_0, 0) v_1. \end{aligned}$$

This is the reverse computation [9].

We can generalize this concept to the 1-D wave propagation problem. In (10), we identify adjoint variables $\mathbf{p}$ with $d\mathbf{c}$ for the input, and $\mathbf{q}$ with $d\mathbf{h}$ for the output. To the intermediate variables $d\mathbf{u}^k$, we associate adjoint variables $\mathbf{v}^k$. Performing the reverse mode calculation, we must start at index $k = m - 1$. Let q_k, for $k = 1 : m$, be the elements of $\mathbf{q}$. The adjoint-times-vector algorithm is

$$\begin{aligned} &\texttt{set } \mathbf{v}^m = 0 \\ &\texttt{for } k = m - 1 : 0 \\ &\quad \mathbf{v}^{k+1} = \mathbf{v}^{k+1} + \mathbf{e}_1 q_{k+1} \\ &\quad \mathbf{v}^k = \mathbf{v}^k + \partial_2 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})^T \mathbf{v}^{k+1} \\ &\quad \mathbf{v}^{k-1} = \mathbf{v}^{k-1} + \partial_3 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})^T \mathbf{v}^{k+1} \\ &\quad \mathbf{p} = \mathbf{p} + \partial_1 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})^T \mathbf{v}^{k+1} \\ &\texttt{end} \end{aligned} \tag{15}$$

At the end of the calculation, we can identify $\mathbf{p} = J^T \mathbf{q}$.

4.3. *Discussion*

The foregoing methodology, while limited to the 1-D problem, can be adapted to solve the more complicated 2-D problem. What we wish to emphasize here is the conciseness of the extended Jacobian framework and how to exploit the underlying problem structure. The algorithms in (10), (14), and (15) can be viewed as code templates for Jacobian and adjoint vector product calculations. AD is deployed in computing the Jacobian and adjoint of the subproblem described by the time stepping process (4).

We recall that the adjoint (reverse product) $\partial_1 F(\mathbf{c}, \cdot, \cdot)^T y$, etc., can be computed using the adjoint (reverse) mode of an AD tool. For large problems like this, computing the adjoint product of the timestep routine (4) can be very expensive, since the size of $\mathbf{c}$, $\mathbf{u}^k$, and $\mathbf{u}^{k-1}$ can be large. An AD tool would be default assume that every element of $\partial_1 F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ depends on every element of $\mathbf{c}$, $\mathbf{u}^k$, and $\mathbf{u}^{k-1}$. This assumption on dependence generates a "table" which is used in computing intermediate values in the reverse product mode. For example, ADOL-C [10] implements this lookup by creating a tape, which

it will write on the disk if the problem size is large. When it does this, it becomes unacceptably inefficient.

This concern brings us to the main idea of this paper, i.e., that of AD applied to the finite difference stencil. Our approach is to use AD on the smallest component of the calculation—a kind of "microscopic" structure exploitation. We discuss how this is done in the next section.

In principle, what we are exploiting is a specific sparsity structure that is inherent in the finite difference scheme. A general approach for exploiting sparsity in AD is described in [2].

5. EXPLOITING THE STENCIL STRUCTURE

The finite difference method that we used in the 1-D case can be written as indicated in (4) which we rewrite here

$$\mathbf{u}^{k+1} = F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1}), \qquad \text{with } \mathbf{u}^{-1} = \mathbf{u}^0 = \mathbf{0}.$$

This shorthand notation does not reveal the stencil structure given by the explicit formulas in (3). For the jth component of $\mathbf{u}^{k+1}$, j not equal to 0 or n, from (3a) we can write

$$u_j^{k+1} = f\left(c_j, u_{j-1}^k, u_j^k, u_{j-1}^k, u_j^{k-1}\right). \tag{16}$$

The above expression spells out clearly that the dependence of $\mathbf{u}^{k+1}$ on $\mathbf{c}$, $\mathbf{u}^k$, and $\mathbf{u}^{k-1}$, is very *sparse*. This is best visualized by studying Fig. 2. Thus, we need only to deal with f which is a function of only 5 variables. From (3c–3d), we have two more such functions but they depend only on 4 and 3 variables, respectively, and are given by

$$\begin{aligned} u_0^{k+1} &= f_L\left(c_0, u_0^k, u_1^k, u_0^{k-1}\right), \\ u_n^{k+1} &= f_R\left(c_n, u_{n-1}^k, u_n^k\right). \end{aligned}$$

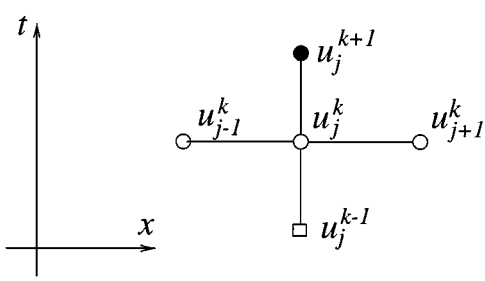

FIG. 2. The stencil for the 1-D problem for $j \neq 0, n$. Boundary nodes are slightly different and require separate treatment.

The function $F(\cdot,\cdot,\cdot)$, representing a time-step, is now replaced with the pseudo-code

$$\begin{aligned}
&\texttt{function } \mathbf{u}^{k+1} = F(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1}) \\
&u_0^{k+1} = f_L\left(c_0, u_0^k, u_1^k, u_0^{k-1}\right) \\
&u_n^{k+1} = f_R\left(c_n, u_{n-1}^k, u_n^k\right) \\
&\texttt{for } j = 1 : n-1 \\
&\qquad u_j^{k+1} = f\left(c_j, u_{j-1}^k, u_j^k, u_{j-1}^k, u_j^{k-1}\right) \\
&\texttt{end}
\end{aligned} \tag{17}$$

It is to these "small" functions of a few variables that we want to apply automatic differentiation. The benefits are that we will have efficient codes which explicitly exploit the structure of the problem. The cost is that the derivative and adjoint codes will be slightly more complicated to assemble. We discuss this next.

5.1. *Sparse Jacobian*

Due to the sparsity afforded by the stencil structure, it is feasible to calculate the full Jacobian (rather than the Jacobian vector product). To see this we introduce projection matrices. Let $\mathbf{e}_j$ be the jth unit vector (we will let j run from 0 to n for convenience). Then (16) can be rewritten in terms of vectors, $\mathbf{c}$, $\mathbf{u}^k$, and $\mathbf{u}^{k-1}$ as

$$u_j^{k+1} = f\left(\mathbf{e}_j^T\mathbf{c}, \mathbf{e}_{j-1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^k, \mathbf{e}_{j+1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^{k-1}\right). \tag{18}$$

In computing the Jacobian, we needed the derivatives of $F(\cdot,\cdot,\cdot)$ with respect to the 3 variables. We next derive procedures to do this using the stencils.

The components of $F_1(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ are

$$F_1 = \begin{bmatrix} \left(\nabla_{\mathbf{c}} u_0^{k+1}\right)^T \\ \left(\nabla_{\mathbf{c}} u_1^{k+1}\right)^T \\ \vdots \\ \left(\nabla_{\mathbf{c}} u_{n-1}^{k+1}\right)^T \\ \left(\nabla_{\mathbf{c}} u_n^{k+1}\right) \end{bmatrix}.$$

The gradients are easily obtained by differentiating (18) with respect to $\mathbf{c}$. We obtain, for $j \neq 0, n$,

$$\nabla_{\mathbf{c}} u_j^{k+1} = \partial_1 f\left(\mathbf{e}_j^T\mathbf{c}, \mathbf{e}_{j-1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^k, \mathbf{e}_{j+1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^{k-1}\right)\mathbf{e}_j,$$

which is an $(n+1)$-vector with a single nonzero entry at j. Thus, it can be seen that $F_1(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ is a *diagonal* matrix. This property is not apparent to state-of-the-art automatic differentiation programs.

The Jacobian $F_3(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ will also be diagonal for the same reason and will be computed by applying AD to (16). The Jacobian $F_2(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ will be slightly more complicated. The components of the Jacobian are similar to those of $F_1(\cdot,\cdot,\cdot)$ except that the gradient will be with respect to $\mathbf{u}^k$. Directly differentiating (16) with respect to $\mathbf{u}^k$

yields

$$\begin{aligned}\nabla_{\mathbf{u}^k} u_j^{k+1} = {} & \partial_2 f\left(\mathbf{e}_j^T\mathbf{c}, \mathbf{e}_{j-1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^k, \mathbf{e}_{j+1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^{k-1}\right)\mathbf{e}_{j-1} \\ & + \partial_3 f\left(\mathbf{e}_j^T\mathbf{c}, \mathbf{e}_{j-1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^k, \mathbf{e}_{j+1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^{k-1}\right)\mathbf{e}_j \\ & + \partial_4 f\left(\mathbf{e}_j^T\mathbf{c}, \mathbf{e}_{j-1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^k, \mathbf{e}_{j+1}^T\mathbf{u}^k, \mathbf{e}_j^T\mathbf{u}^{k-1}\right)\mathbf{e}_{j+1}.\end{aligned}$$

Thus, the matrix $F_2(\mathbf{c}, \mathbf{u}^k, \mathbf{u}^{k-1})$ is a *tridiagonal* matrix.

We can summarize the steps in a MATLAB pseudo-code[2]

$$\begin{aligned}& F_1 = \left[\partial_1 f_L \mathbf{e}_0^T\right]; \\ & F_2 = \left[\left(\partial_2 f_L \mathbf{e}_0^T + \partial_3 f_L \mathbf{e}_1^T\right)\right]; \\ & F_3 = \left[\partial_4 f_L \mathbf{e}_0^T\right]; \\ & \texttt{for } j = 1 : n-1 \\ & \qquad F_1 = \left[F_1; \partial_1 f \mathbf{e}_j^T\right]; \\ & \qquad F_2 = \left[F_2; \left(\partial_2 f \mathbf{e}_{j-1}^T + \partial_3 f \mathbf{e}_j^T + \partial_4 f \mathbf{e}_{j+1}^T\right)\right]; \\ & \qquad F_3 = \left[F_3; \partial_5 f \mathbf{e}_j^T\right]; \\ & \texttt{end} \\ & F_1 = \left[F_1; \partial_1 f_R \mathbf{e}_n^T\right]; \\ & F_2 = \left[F_2; \partial_2 f_R \mathbf{e}_{n-1}^T + \partial_3 f_R \mathbf{e}_n^T\right]; \\ & F_3 = [F_3; \texttt{zeros}(1, n+1)];\end{aligned}$$

Once the matrices F_1, F_2, and F_3 are obtained, we can use the code in (10) to compute the forward derivatives and the code in (14) to compute the adjoint. The codes for the partial derivatives of f, f_L, and f_R are easily obtained using AD. These codes are expected to be very efficient because of the simplicity of the stencil formula and because of the small number of independent variables involved. We have gained efficiency in the AD computation by applying AD at the stencil level. The cost to the user is performing some detail "hand" coding.

We can employ a similar approach for the more complicated 2-D example. We note that a typical stencil for interior nodes is given by

$$u_{ij}^{k+1} = f\left(c_{ij}, u_{i-1,j}^k, u_{i+1,j}^k, u_{i,j-1}^k, u_{i,j+1}^k, u_{ij}^k, u_{ij}^{k-1}\right).$$

The stencil is displayed in Fig. 3. Boundary node and corner nodes, because of the absorbing boundary conditions described in (7c), result in slightly more complex stencils. The key observation is that the stencil embodies the sparsity structure of the Jacobian and is a feature that should be exploited.

5.2. *Stencil in Forward and Reverse Mode*

We can also exploit stencil structure without explicitly computing the Jacobian. This results in procedures to compute the Jacobian times vector and adjoint times vector. Suppose we are given $d\mathbf{c}$ and we wish to calculate the vector $d\mathbf{h}$ as outlined in (10). The approach we take will make use of stencil formulas such as (16). Assume that we have used AD

[2] One would use sparse utilities to implement this in MATLAB.

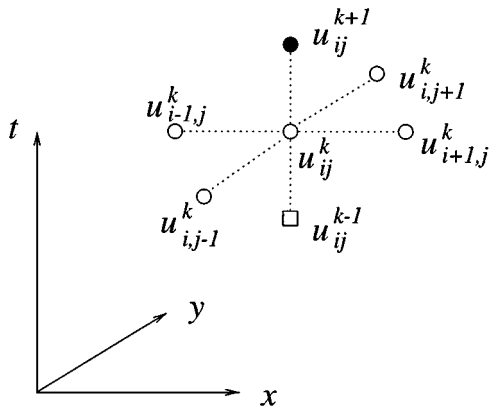

FIG. 3. The stencil for the 2-D problem for an interior node. Boundary and corner nodes are slightly different and require separate treatment.

to generate an algorithm to compute the gradient of $f(\cdot,\cdot,\cdot,\cdot,\cdot)$ times a 5-vector; that is, given X and dX, we have a procedure to find

$$f(X) \qquad \text{and} \qquad \nabla f(X)\cdot dX.$$

Here X stands for a 5-vector with components $X=[c_j, u^k_{j-1}, u^k_j, u^k_{j+1}, u^{k-1}_j]^T$. Then it is easy to see that from (16) if $dX=[dc_j, du^k_{j-1}, du^k_j, du^k_{j+1}, du^{k-1}_j]^T$, then

$$du^{k+1}_j = \nabla f(X)\cdot dX.$$

We would have similar formulas for $j=0$ and $j=n$ with the difference that the vector of independent variables would be 4 and 3 dimensional, respectively. We can therefore assemble the du^{k+1}_j within an outer loop which corresponds to the time steps. The pseudo-code would take the form

$$\begin{aligned}
&d\mathbf{u}^{-1} = d\mathbf{u}^0 = 0\\
&\texttt{for } k = 0 : m-1\\
&\quad X = \left[c_0, u^k_0, u^k_1, u^{k-1}_0\right]^T;\ dX = \left[dc_0, du^k_0, du^k_1, du^{k-1}_0\right]^T\\
&\quad du^{k+1}_0 = \nabla f_L(X)\cdot dX\\
&\quad X = \left[c_n, u^k_{n-1}, u^k_n\right]^T;\ dX = \left[dc_n, du^k_{n-1}, du^k_n\right]^T\\
&\quad du^{k+1}_n = \nabla f_R(X)\cdot dX\\
&\quad \texttt{for } j = 1 : n-1\\
&\qquad X = \left[c_j, u^k_{j-1}, u^k_j, u^k_{j+1}, u^{k-1}_j\right]^T;\ dX = [dc_j, du^k_{j-1}, du^k_j, du^k_{j+1}, du^{k-1}_j]^T\\
&\qquad du^{k+1}_j = \nabla f(X)\cdot dX\\
&\quad \texttt{end}\\
&\quad dh_{k+1} = du^{k+1}_0\\
&\texttt{end}
\end{aligned} \tag{19}$$

The adjoint codes generated by AD on the stencil formula (16) computes the following. Given a scalar v and a vector X, the adjoint code calculates a 5-vector

$$v\nabla f(X).$$

We have similar procedures for $f_L(\cdot)$ and $f_R(\cdot)$. In reverse mode, we want to perform a calculation similar to (15). We start with a vector $\mathbf{q} = [q_1, q_2, \ldots, q_m]^T$, and we wish to compute $\mathbf{p} = J^T q$. The pseudo-code for this is

$$
\begin{aligned}
&\texttt{set } \mathbf{v}^m = 0\\
&\texttt{for } k = m-1:0\\
&\quad v_0^{k+1} = v_0^{k+1} + q_{k+1}\\
&\quad X = \left[c_0, u_0^k, u_1^k, u_0^{k-1}\right]^T\\
&\quad Y = \left[p_0, v_0^k, v_1^k, v_0^{k-1}\right]^T\\
&\quad Y = Y + dv_0^{k+1}\nabla f_L(X)\\
&\quad \texttt{for } j = 1:n-1\\
&\qquad X = \left[c_j, u_{j-1}^k, u_j^k, u_{j+1}^k, u_j^{k-1}\right]^T\\
&\qquad Y = \left[p_j, v_{j-1}^k, v_j^k, v_{j+1}^k, v_j^{k-1}\right]^T\\
&\qquad Y = Y + dv_j^{k+1}\nabla f(X)\\
&\quad \texttt{end}\\
&\quad X = [c_n, u_{n-1}^k, u_n^k]^T\\
&\quad Y = [p_n, v_{n-1}^k, v_n^k]^T\\
&\quad Y = Y + dv_n^{k+1}\nabla f_R(X)\\
&\texttt{end}
\end{aligned}
\tag{20}
$$

Algorithm (19) for the forward product calculation and algorithm (20) for the reverse product calculation will be extremely efficient because the codes produced by AD for calculating the derivative of f and its adjoint will be nearly as short and simple as the function calculation. The number of independent variables is small, and there are no loops as can be seen in (3a). The algorithm (20) can be easily understood by noticing that the input is the vector $\mathbf{q}$. The algorithm goes backwards in time computing adjoints of each state by using the previously computed adjoints. At completion, this algorithm returns the adjoints of the independent variables.

In 2-D, the stencil is a bit more complex as already pointed out, but the general principle described here applies. Indeed, we have coded a version of algorithms (19) and (20) for the 2-D problem. We discuss the results of our numerical calculations next.

6. NUMERICAL RESULTS

We present some results from our numerical computations. In both examples, we use TAMC [7] to obtain derivative and adjoint codes from Fortran sources. All the Fortran codes were "wrapped" as MATLAB `mex`-files and used in conjunction with MATLAB codes.

Our goal in this paper is to demonstrate the use of extended Jacobian framework together with exploitation of stencil in Jacobian and adjoint calculations. In a subsequent work, we apply our approach to solve a 2-D inverse problem arising in acoustic imaging.

6.1. *The 1-D Problem*

In our example, we choose $\Delta x = 1$ and $\Delta t = 0.8$. The domain is of length $L = (N-1)\Delta x$. We will use several N in our calculations. The initial boundary value problem for the 1-D wave equation is discretized according to (3). The number of time steps is m, which will be varied as well. For excitation $f(t)$, we choose the derivative of the Gaussian. The graph of f is shown in Fig. 4a. We take two sound speeds $c_1(x)$ and $c_2(x)$, shown in Fig. 4b when $n = 100$. The resulting boundary data are $h(t) = u(0, t)$. When the medium is $c_1(x)$

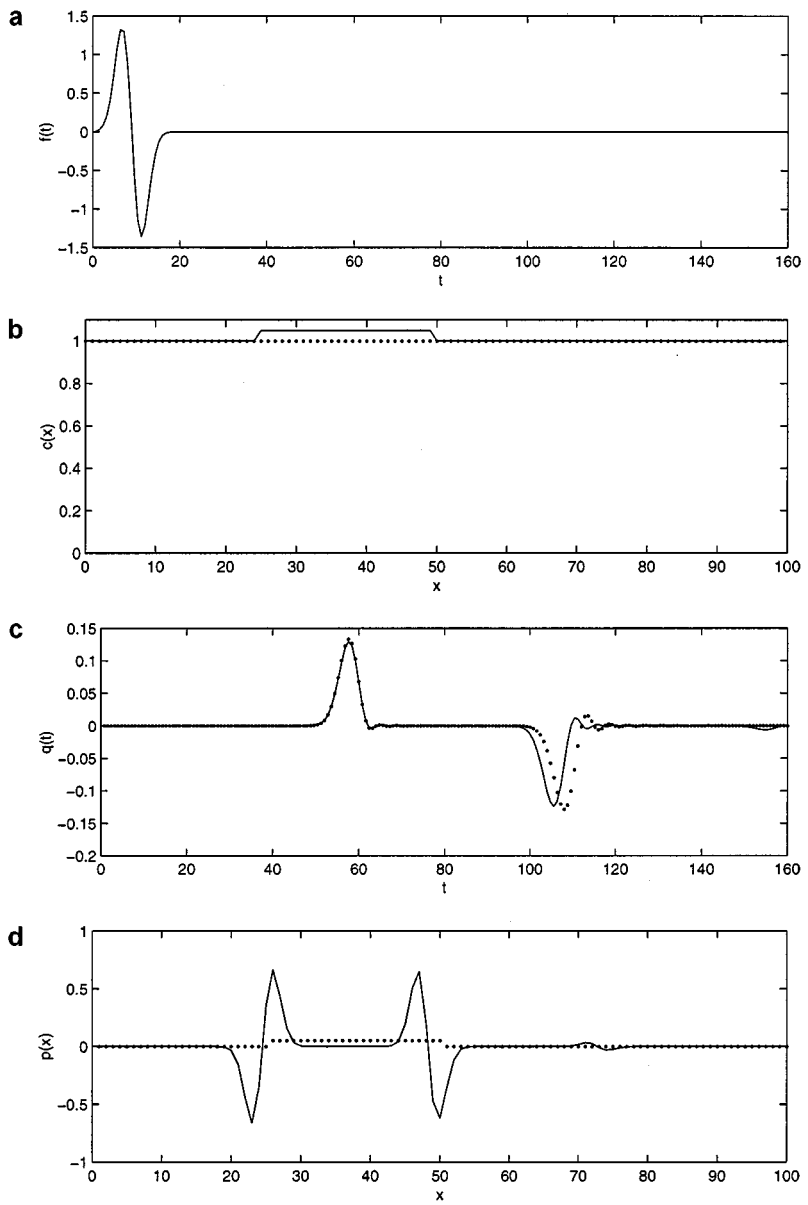

FIG. 4. (a) The excitation used in the examples. (b) The two sound speed profiles, $c_1(x)$ in dots, $c_2(x)$ in solid. (c) The graphs of $(\mathbf{h}_2 - \mathbf{h}_1)$ and $J(\mathbf{c}_1)(\mathbf{c}_2 - \mathbf{c}_1)$. (d) The graph of $\mathbf{p} = J(\mathbf{c}_1)^T(\mathbf{h}_2(t) - \mathbf{h}_1)$, shown for comparison, is the graph of $\mathbf{c}_2 - \mathbf{c}_1$.

the boundary data are $h_1(t)$, and $h_2(t)$ when the medium is $c_2(x)$. Let $q = h_1 - h_2$; the graph of $q(t)$ is displayed in Fig. 4c for $m = 200$.

We will first compute the Jacobian at $c_1(x)$ times the difference $c_2(x) - c_1(x)$. The resulting output vector should be very close to $q(t)$. A comparison of $p(t)$ with $J(\mathbf{c}_1)(\mathbf{c}_2 - \mathbf{c}_1)$ is shown in Fig. 4c. Next we calculate the adjoint times $q(t)$; i.e.,

$$\mathbf{p} = J(\mathbf{c}_1)^T(\mathbf{h}_2 - \mathbf{h}_1).$$

The output of this calculation will be the steepest descent direction corresponding to the nonlinear least-squares functional in (6). This direction would be similar to $c_2(x) - c_1(x)$. The graph of $p(t)$ is shown in Fig. 4d. One can see that the 2 big signals, which are scaled versions of f, are reproduced near the places where $c_2(x) - c_1(x)$ take jumps. Unfortunately, the similarity ends there; the result shows that the inverse problem is not very well posed. However, we did check that the Jacobian and the adjoint are correctly computed by evaluating

$$\mathbf{q}^T J\, d\mathbf{c} \qquad \text{and} \qquad d\mathbf{c}^T J^T \mathbf{q} \tag{21}$$

and comparing their values for any choice of $\mathbf{c}$, $\mathbf{q}$, and $d\mathbf{c}$. The agreement is usually 14 digits. Additionaly, the result is also confirmed by the computation of $\mathbf{q}^T J d\mathbf{c}$ using finite differences, where we usually get about 6 digits right. We show a typical run in Fig. 5.

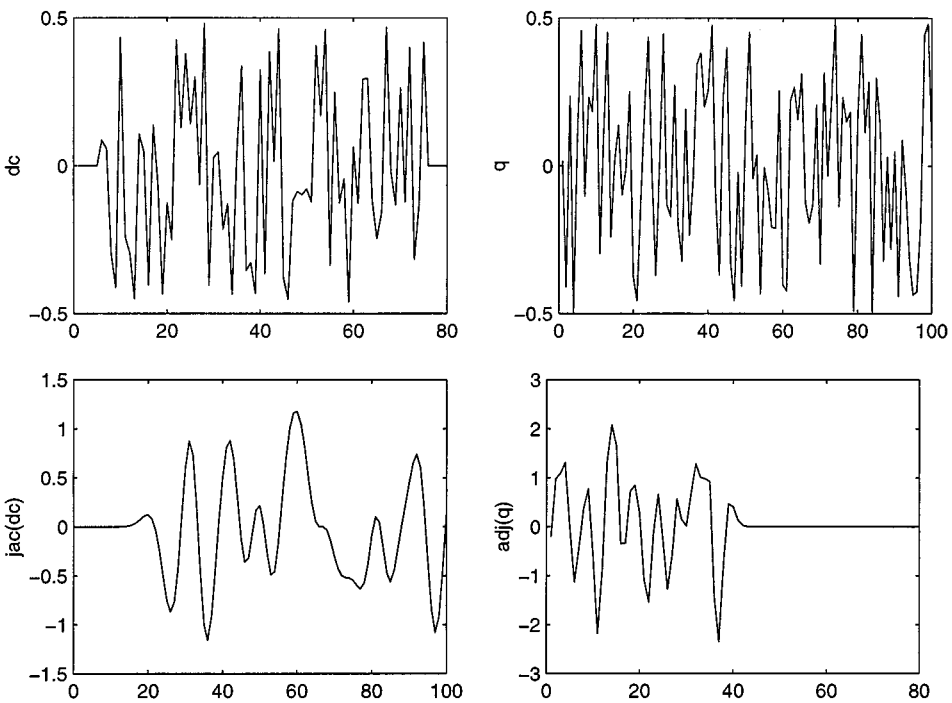

FIG. 5. A test of the correctness of Jacobian and adjoint calculations. In this example, $N = 80$ and $m = 100$. We choose at random 2 vectors $d\mathbf{c}$ and $\mathbf{q}$ displayed on the first row. On the second row, we show $J\, d\mathbf{c}$ and $J^T\mathbf{q}$. The inner products $\mathbf{q}^T J\, d\mathbf{c}$ and $d\mathbf{c}^T J^T \mathbf{q}$ are evaluated. They agree to 14 digits.

Computation time is linear in the number of x nodes for a fixed number of t nodes. There is no difficulty with memory as the stencil codes are very simple with a small number of independent variables.

6.2. *The 2-D Problem*

In the 2-D problem, we set up a grid of 161-by-161 node points. The computational domain is $[-80, 80] \times [-80, 80]$, thus $\Delta = 1$. For interior nodes, we use a second order accurate discretization of the wave equation (7a). On the boundary nodes, we use a second order discretization of the Enquist–Majda boundary condition (7c)–(7d). The corner nodes, and the 2 nodes adjacent to the corner on the boundary, require special stencils. The stencils are obtained by requiring that the discrete wave equation be satisfied at the node while at the same time satisfying the discrete version of the absorbing boundary condition.

For excitation, we choose a point source. To model this, if the source is at node (i_s, j_s), i.e., located at position $(i_s\Delta, j_s\Delta)$, we assume that $f(x, y, t)$ is

$$f(x, y, t) = \begin{cases} \phi(t) & \text{at } (i_s\Delta, j_s\Delta) \\ 0 & \text{otherwise.} \end{cases}$$

The time-dependent function $\phi(t)$ is chosen to be a Gaussian and will be sampled at the time increments $\Delta t = 0.55$, which is the time step chosen for the finite difference scheme.

Data will be collected at 64 stations located at node points. These points are nodes that lie close to a set of points distributed evenly at 64 places on the circumference of a circle of radius 72. We will take 381 times steps. A window of size $[-70, 70] \times [-70, 70]$ represents where $c(x, y)$ is allowed to vary. Thus the mapping from sound speed c to data at the receiver is from $\mathbb{R}^{141\times141}$ to $\mathbb{R}^{64\times381}$.

In Fig. 6a we display the sound speed distribution in the domain. The receivers are marked with circles; receiver 1 is at 0° from the positive x-axis. The source is located by a $\star$. Next, in Fig. 6b, we display the receiver data when the medium has the two cylinders shown. The difference between the previous data and those when the domain is homogeneous is shown in Fig. 6c. In Fig. 6d, we show the result of applying the adjoint on the difference data in Fig. 6c. This process is often refered to as back-propagation, and corresponds to the steepest descent direction for the nonlinear least squares functional in (8). The resulting vector should resemble the image of the two cylinders. Indeed this is the case if one compares. Figs. 6d and 6e, the latter displayed for comparison.

In numerous experiments with random vectors, we were able to get the inner products similar to (21) to agree 14 digits. The adjoint calculations take approximately 115 seconds on a 4-processor SGI Challenge L. The hand-coded gradient calculation takes approximately 90 seconds. The fully AD-generated gradient (using the reverse mode) takes almost 5 minutes to compute. In another run, the size of the sound speed c (independent variables) was increased from 141×141 to 201×201. A window of size $[-100, 100] \times [-100, 100]$ represents where $c(x, y)$ could vary. The receivers are placed on the circumference of a circle of radius 90. The grid of 241×241 node points is set up corresponding to the domain $[-120, 120] \times [-120, 120]$. The adjoint calculations scale very well and take 225 seconds to compute. The hand-coded gradient calculation is marginally efficient and takes about 200 seconds. However, the fully AD-generated gradient scheme becomes intractable because the volume of data to be stored for the reverse mode is much larger and some extra

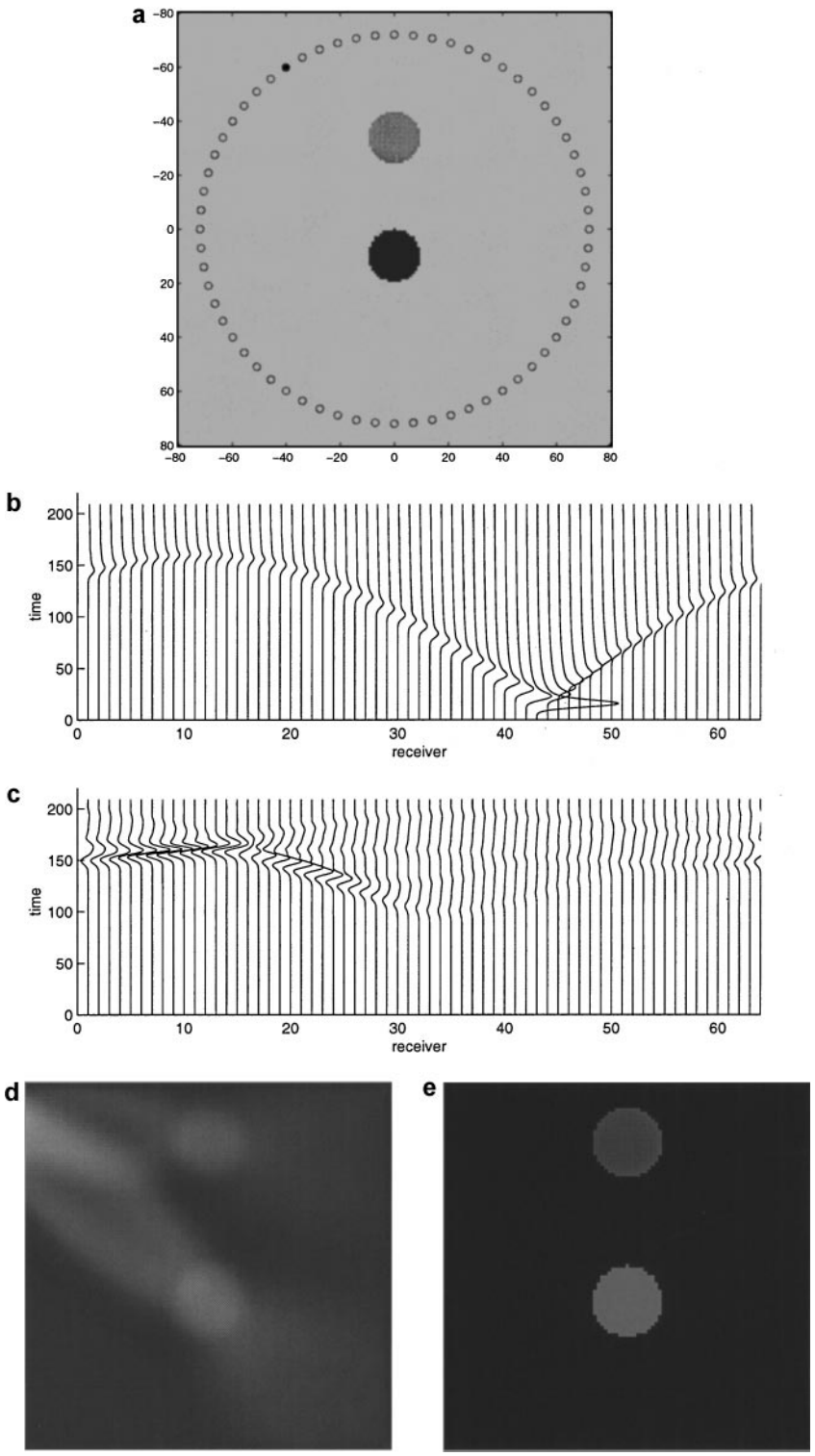

FIG. 6. (a) The setup for the numerical experiment. The two cylinders represent sound speed anomalies. The darker cylinder is 2% faster while the lighter cylinder is 1% faster than the background medium. Shown in circles are the receiver locations. A star marks the location of the point source. (b) The receiver data when the two cylinders are present. (c) The difference between (b) and receiver data when the medium is constant. (d) The adjoint applied to (c). (e) The two cylinders plotted on the same scale as in (d) for comparison.

time is spent in memory paging and accessing secondary storage. This takes nearly 20 minutes to finish. This illustrates that the adjoint scheme presented in this paper is automatic but still much like traditional hand-coding in performance, while the naive use of AD doesn't provide a reasonably efficient solution.

7. CONCLUSIONS

We have described an inverse problem arising in wave propagation and how the need arises for efficient computation of Jacobian and adjoint products when the problem is posed as a least-squares problem. In this work, we describe the extended Jacobian framework which gives a high level description of Jacobian and adjoint vector product calculation. The framework is particularly appropriate for functions whose evaluation involve some type of time stepping, such as those that arise in discretizing the wave equation.

We show further that the stencil structure of the finite difference scheme that provides the underlying function evaluation can be exploited. Automatic differentiation is applied at the stencil level, and the resulting subprograms fit nicely within the extended Jacobian framework. The framework provides a guide for building highly efficient codes for Jacobian and adjoint vector product evaluations. The one drawback of the approach is that we have given up some "automation" for efficiency. A small amount of hand-coding is required to assemble the programs. Nevertheless, our approach provides a way to overcome memory problems associated with present AD technology.

One important extension of the 2-D problem we discussed is the case when the receivers don't lie on the grid points. The proposed methodology can be easily extended to handle the interpolation between the grid values required in this case. It poses no difficulty for the adjoint computation as the interpolated values are a linear combination of the values at close grid points. This can be handled through definition of general (linear) projection operators which when applied to the values at all grid points results in the the values at the receivers. This linear operator is trivial to handle in the adjoint computation via AD.

Overall, the idea of exposing the stencil structure is very promising and can lead to an order of magnitude improvement in the adjoint code, as our numerical results show.

REFERENCES

1. G. Chavent, F. Clement, and S. Gomez, Waveform inversion by MBTT formulation, in *Mathematical and Numerical Aspects of Wave Propagation*, edited by Cohen, *et al.* (SIAM, Philadephia, 1995), p. 713.
2. T. Coleman and A. Verma, The efficient computation of sparse Jacobian matrices using automatic differentiation, *SIAM J. Sci. Comput.* **19**, 1210 (1998).
3. T. Coleman and A. Verma, Structure and efficient Hessian calculation, in *Advances in Nonlinear Programming*, edited by Yuan (Kluwer Academic, Boston, 1996).
4. T. Coleman and A. Verma, Structure and efficient Jacobian calculation, in *Computational Differentiation: Techniques, Applications, and Tools*, edited by Berz *et al.* (SIAM, Philadelphia, 1996), p. 149.
5. T. Coleman, F. Santosa, and A. Verma, Semi-automatic differentiation, in *Proceedings of Optimal Design and Control Workshop, VPI, 1997*.
6. B. Enquist and A. Majda, Absorbing boundary conditions for the numerical simulation of waves, *Math. Comp.* **31**, 629 (1977).
7. R. Giering, *Tangent Linear and Adjoint Model Compiler* (User Manual, TAMC Version 4.7, 1997).
8. A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, *Optim. Methods Software* **1**, 35 (1992).

9. A. Griewank, Some bounds on the complexity of gradients, Jacobians, and Hessians, in *Complexity in Nonlinear Optimization*, edited by Pardalos (World Scientific, Singapore, 1993).
10. A. Griewank, D. Juedes, and J. Utke, ADOL-C, a package for the automatic differentiation of algorithms written in C/C++, *ACM Trans. Math. Software* **22**, 131 (1996).
11. T. Mast, A. Nachman, and R. Waag, Focussing and imaging using eigenfunctions of the scattering operator, *J. Acous. Soc. Am.*, in press.
12. F. Santosa and W. Symes, *An Analysis of Least-Squares Velocity Inversion* (Society of Exploration Geophysicists, Tulsa, 1989).
13. F. Santosa and W. Symes, Computation of the Hessian for least-squares solutions of inverse problems of reflection seismology, *Inverse Problem* **4**, 211 (1988).
14. W. Symes, A differential semblance criterion for inversion of multioffset seismic reflection data, *J. Geophys. Res.* **98**, 2061 (1993).
15. W. Symes and C. Zhang, *A Finite Difference Time Stepping Class*, Rice University TRIP Report, 1997.
16. A. Tarantola, *Inverse Problem Theory* (Elsevier, Amsterdam, 1987).