Semi-Automatic Differentiation[*]

Thomas F. Coleman
Computer Science Department and Center for Applied Mathematics
Cornell University, Ithaca NY 14850

Fadil Santosa
School of Mathematics
University of Minnesota
Minneapolis, Minnesota 55455

Arun Verma
Computer Science Department
Cornell University
Ithaca NY 14850

We demonstrate how the structure that arises in inverse and optimal design problems can be used to aid in the efficient application of automatic differentiation ideas. We discuss the program structure of generic inverse problems and then illustrate, with two examples (one example involves the heat equation, the other involves wave propagation) how structure can be used in combination with automatic differentiation. Finally, we report numerical results and describe the ADMIT-2 software package which enables efficient derivative computation of structured problems.

1. Introduction

Effective use of automatic differentiation (AD) software for realistic large-scale problems is often not "automatic." Indeed, performance gains of several orders of magnitude can sometimes be achieved by using AD in a selective manner (as opposed to straightforward use of AD software). In particular, large-scale problems typically exhibit structure: for AD to be used efficiently (or even feasibly) it is crucial that the dominant problem structure be understood and exploited. This is certainly true for optimal design (and inverse) problems. Direct application of AD can be unbearably expensive on such problems due to their density and complexity. On the other hand, if the use of AD is integrated with the problem structure then there is significant potential for effective calculation of derivatives using AD software.

A large number of engineering design problems are posed as inverse problems. An inverse problem, in discretized form, can be described as follows.

[*] This research was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-97ER25013 and DE-FG02-94ER25225 and also by National Science Foundation grant DMS 9503114 and by the Air Force Office for Scientific Research F49620-95-I-0305.

We begin with a *forward* computation: Given the value of parameters $x = \bar{x} \in \Re^n$, solve for the state $y \in \Re^m$,

$$F(\bar{x}, y) = 0 \tag{1}$$

where function F is typically nonlinear and is assumed to be differentiable. Often F represents a finite difference method for the approximate solution of a differential equation. In that case, the parameter set x typically consists of design variables and y is the solution. Typically, $m \geq n$, and F is a square system w.r.t. y.

Now the *inverse* problem corresponding to this forward computation can be phrased as follows. Given a target $\bar{y}$, determine values for the parameters x such that the forward computation yields a value *close* to the target $\bar{y}$.

There are two popular ways of solving a (discretized) inverse problem. In the case $m = n$, we can try to look for an exact match by solving the nonlinear equation,

$$y(x) - \bar{y} = 0. \tag{2}$$

Alternatively, when $\bar{y}$ is not in the range or when $m > n$, we look for a "close" match by solving the nonlinear least squares problem,

$$min \quad \|y(x) - \bar{y}\|_2. \tag{3}$$

where $y(x)$ is implicitly defined by the forward process (1). Many inverse problems exhibit a time-stepping structure which can be exploited by applying the EASE (Extended functions And Structure Exploitation) scheme of Coleman and Verma [2, 1]. EASE provides an efficient way to compute the Newton step, and other derivative-related information, by exploiting the structure of the problem at hand. In the next section we briefly review EASE and its benefits.

2. Review of EASE

To illustrate the design of EASE, we consider the following very simple time-stepping structured computation shown in Equation (4):

$$y_i = T(y_{i-1}), i = 1, \ldots, N. \tag{4}$$

Based on (4), we want to solve the following boundary value problem :

Problem : Estimate y_0 given the boundary condition $y_N = y_{final}$

This problem can be viewed as the following non-linear equations problem :

$$F(y_0) = y_N(y_0) - y_{final} = 0$$

The Jacobian J of $F(y_0)$, w.r.t. y_0, is given by the product, $J = J_N \cdot J_{N-1} \cdots J_1$, where J_i denotes the derivative of y_i w.r.t y_{i-1}, i.e., $J_i = T'(y_{i-1})$.

Observation : Typically the Jacobians J_i's are sparse (e.g. due to local nature of the "stencils" in finite difference schemes that define the function T in PDE settings), but the overall Jacobian J might be completely dense! This means if we try to compute the Jacobian matrix J ignoring the structure completely, the work required will be equivalent to n function evaluations, where n is the size of the problem, i.e., the size of vector y.

Here is how EASE can help: EASE highlights the inherent structure by using the "extended" form shown in Figure 1.

1) "Solve" $y_1 - T(y_0) = 0$ (i.e., $y_1 := T(y_0)$)
2) "Solve" $y_2 - T(y_1) = 0$
$\vdots$
N) "Solve" $y_N - T(y_{N-1}) = 0$
Compute $F := y_N - y_{final}$

Figure 1: Extended function form in EASE

The extended function in Figure 1 is not just a function of independent variables $x = y_0$, but also depends on the intermediate variables $y_1, \ldots y_{N-1}$. The second step of EASE is to differentiate the extended function w.r.t all its arguments, i.e., $y_0, y_1, \ldots y_{N-1}$, to get the extended Jacobian shown in Equation (5)

$$J_E = \begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_N & I \\ & & & & I \end{pmatrix}. \qquad (5)$$

Features of EASE
There are several benefits to using EASE.

- The extended Jacobian matrix J_E is often sparse, hence J_E can be computed very efficiently using sparsity exploiting ideas [3].
- The Newton step $s = -J^{-1}F$, can be directly computed from J_E via a sparse solve as shown in Equation (6).

$$J_E \begin{pmatrix} \delta x \\ \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ -F \end{pmatrix} \qquad (6)$$

- If the true Jacobian matrix J is required, it can be formed from J_E using elementary linear algebra.
- If a solution of the Newton step is desired using conjugate gradients, the product JV can be formed from J_E with out explicitly forming J. In some cases this method might be more cost effective than the direct solve.

EASE can be extended to Hessian computations, and allows for efficient computation of the Newton step ($\Delta x = H^{-1}\nabla f$) and the true Hessian matrix from a (sparse) extended Hessian matrix [1].

3. Heat Conductivity Inverse Problem

To illustrate the structured application of AD in a concrete way, we consider a very simple inverse problem involving heat transfer. The problem is to find conductivity properties of a 1-dimensional bar whose predicted temperature evolution matches desired (or measured) behavior.

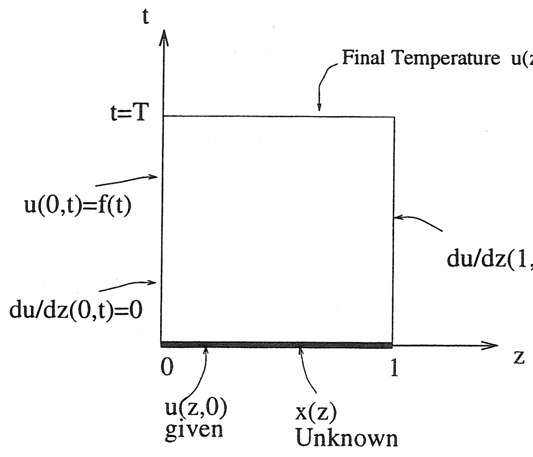

Figure 2: 1-D inverse problem for the heat equation

The setup of the 1-D heat equation is shown in Figure 2. There is a thin bar with ends at $z = 0$ and $z = 1$. The governing partial differential equation is shown:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial z}(x(z)\frac{\partial u}{\partial z}). \tag{7}$$

Function $u(z, t)$ represents the temperature of the bar at position z and time t. Function $x(z)$ represents the *unknown* conductivity of the bar. We aim to determine $x(z)$, to closely match the measured behavior. Suppose that initial temperature distribution $u(z,0)$ is known and both ends of the bar are insulated while the left end temperature is prescribed; the temperature over the bar is let to evolve based on equation (7). The target (or measured) temperature distribution over the bar $\phi_{tar}(z)$ is specified at time $t = T$. Now we wish to solve for the conductivity function $x(z)$ which results in a close match between the target temperature $\phi_{tar}(z)$ and the model temperature distribution $\phi(z) = u(z, T)$. A related inverse problem in heat transfer is described in Mukherjee et. al. [6]. The problem of solving for $x(z)$ is phrased as a least-squares problem:

$$min_{x(z)} \|\phi_{tar}(\cdot) - u(\cdot, T)\|_2.$$

Computationally, this problem is solved using a discretization of spatial and time domains, and employing a suitable finite difference method. For example we can use the following discretization:

$$z_j = (j-1)\Delta z,\ j = 1:N,\ \Delta z = \frac{1}{N-1}$$
$$t_k = k\Delta t,\ k = 0:M,\ \Delta t = \frac{T}{M}$$
$$u_j^{(k)} = u(z_j, t_k).$$

One suitable method for solving 1-D heat equation is the following:

$$\frac{u_j^{(k+1)} - u_j^{(k)}}{\Delta t} = \frac{1}{2}\left(D_+\left(x(z)D_-(u)\right) + D_-\left(x(z)D_+(u)\right)\right).$$

Here D_+, D_- are the spatial difference operators. Expanding:

$$u_j^{(k+1)} = c_j u_j^{(k)} + c_{j+1}u_{j+1}^{(k)} + c_{j-1}u_{j-1}^{(k)} \tag{8}$$

where

$$c_j = (1 - \tfrac{\lambda}{2}(2x_j + x_{j+1} + x_{j-1})), \quad c_{j+1} = \tfrac{\lambda}{2}(x_{j+1} + x_j), \quad c_{j-1} = \tfrac{\lambda}{2}(x_j + x_{j-1}).$$

Equation (8) holds for $j = 2 : N-1$. For boundary cases $j = 1, j = N$, we use boundary conditions along with the following approximations at the boundary.

$$x_{N+1} := 2x_N - x_{N-1} \tag{9}$$
$$x_0 := 2x_1 - x_2 \tag{10}$$
$$u_0^k := f(k\Delta t) \tag{11}$$
$$u_{N+1}^k := 2u_N^k - u_{N-1}^k \tag{12}$$

Approximations made above are linear, e.g. $x_0 := 2x_1 - x_2$ comes from $x_1 = \frac{x_0+x_2}{2}$. Hence we get special equations for $j = 1$ and $j = N$:

$$u_1^{(k+1)} = (1 - 2\lambda x_1)u_1^{(k)} + \frac{\lambda}{2}(x_1 + x_2)u_2^{(k)} + f(k\Delta t)\frac{\lambda}{2}(3x_1 - x_2) \tag{13}$$
$$u_N^{(k+1)} = (1 - \lambda(x_{N-1} - x_N))u_N^{(k)} + \lambda(x_{N-1} - x_N)u_{N-1}^{(k)}. \tag{14}$$

where function $f(t) \equiv u(0,t)$ is given. In the above formulation $\lambda = \frac{\Delta t}{\Delta z^2}$.

Equations (8), (13) and (14) together can be written in a vector form as

$$u^{k+1} = K(x)u^k + h^k$$

where u^k denotes the discretized temperature at time $t = k\Delta t$, i.e., $u^k = [u_1^{(k)}, \dots, u_N^{(k)}]$, and

$$h^k = \begin{pmatrix} \frac{\lambda}{2} \cdot f(k\Delta t)(3x_1 - x_2) \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The matrix $K(x)$ is a tridiagonal matrix of the form:

$$K(x) = \begin{pmatrix} 1 - 2\lambda x_1 & \frac{\lambda}{2}(x_1 + x_2) & & \\ \frac{\lambda}{2}(x_1 + x_2) & 1 - \frac{\lambda}{2}(2x_2 + x_3 + x_1)) & \frac{\lambda}{2}(x_3 + x_2) & \\ & \ddots & \ddots & \ddots \\ & & \lambda(x_{N-1} - x_N) & 1 - \lambda(x_{N-1} - x_N) \end{pmatrix}$$

The inverse problem is typically solved by solving the nonlinear equation :

$$F(x) = u^M(x) - \phi_{tar} = 0$$

3.1. Exploiting structure in computation

Define function $u^+ = G(x,u) = K(x)u + h$. Applying EASE, the "structured" computation of $F(x)$ in 1-D heat equation can be written as shown in Figure 3.

```
Solve for $u^1$ : $u^1 = G(x, u^0)$
Solve for $u^2$ : $u^2 = G(x, u^1)$
$\vdots$
Solve for $u^M$ : $u^M = G(x, u^{M-1})$
Compute $F := u^M - \phi_{tar}$
```

Figure 3: Heat equation extended function

Differentiating, the extended Jacobian is given by:

$$J_E = \left(\begin{array}{c|ccccc} G_x(x,u^0) & -I & & & & \\ G_x(x,u^1) & K(x) & -I & & & \\ \vdots & & \ddots & \ddots & & \\ G_x(x,u^{M-2} & & & K(x) & -I & \\ G_x(x,u^{M-1} & & & & K(x) & -I \\ \hline 0 & & & & & I \end{array}\right)$$

Using the above formulation, we get all the benefits provided by EASE. In particular, the Newton step $\Delta x = -J^{-1}(u^M(x) - \phi_{tar})$ can be computed very cheaply.

3.2. Numerical results

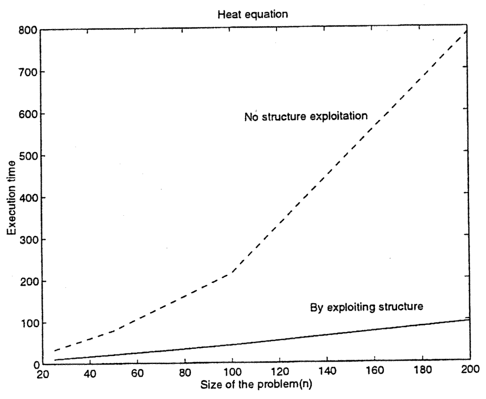

Figure 4: Numerical results for heat equation problem

The graph in Figure 4 shows the effectiveness of EASE compared to the unstructured way of computing the Jacobian and Newton step. The size of the problem is N, the number of points in the discretization of spatial interval [0, 1]. It is easy to show that the unstructured method is quadratic in the dimension of the problem, but the EASE method is linear. Hence we can show an order of magnitude improvement over the dense, unstructured method. In particular, we can show that the unstructured method is $O(MN^2)$ in complexity, while the EASE scheme is $O(MN)$ in complexity. For the purpose of this experiment, the sparse solve on the extended Jacobian matrix was done using the MATLAB's backslash (\) operator. ADMIT-2 software, which is described in Section 6, was employed.

4. Reflection Seismology Inverse Problem

Our second example involves the wave equation and a generic problem in seismology. Reflection seismology inverse problems can be viewed as very large nonlinear data-fitting problems, the desired solutions being the sound speeds of earth's subsurfaces and the data being the reflection seismograms collected at the surface. In other words, we seek a model (or design) of earth which fits the data (or target) in the least squares sense. For a description of a complex reflection seismology inverse problem setting and more details on the nature of the problem, refer to Santosa and Symes [5]. In this paper, we consider a simplified setup of the reflection seismology problem as shown in Figure 5.

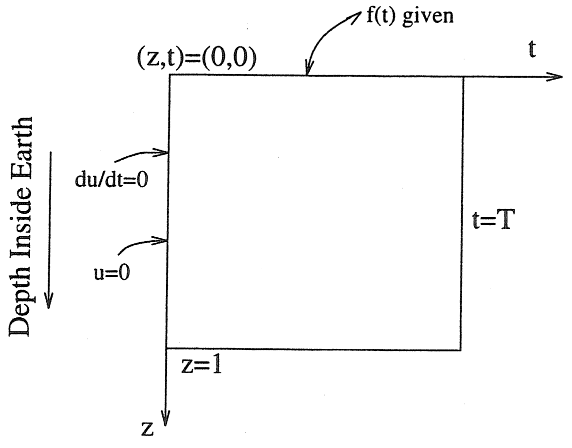

Figure 5: Reflection seismology problem

The problem can be described by the governing PDE (a wave equation) shown below:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial z}(x(z)\frac{\partial u}{\partial z}). \tag{15}$$

The boundary conditions are as follows:

$$\begin{aligned} u(z,0) &= 0 && (16)\\ \frac{\partial u}{\partial t}(z,0) &= 0 && (17)\\ x(0)\cdot\frac{\partial u}{\partial z}(0,t) &= f(t). && (18)\end{aligned}$$

Here z denotes the depth co-ordinate, and $x(z)$ relates directly to sound speeds at depth z inside the earth. Function $u(z,t)$ represents the medium particle displacement at depth coordinate z and time t. Function $f(t)$ denotes the excitation force, in the form of traction applied at the surface ($z = 0$). We chose T sufficiently small so that at time T, the disturbance has not reached $z = 1$, hence we need not specify any boundary condition at $z = 1$.

Discretizing the time and the spatial domains, a suitable finite element method is:

$$u^{k+1} = -u^{k-1} + A(x)u^k + h^k$$

where ($\lambda = \frac{(\Delta t)^2}{(\Delta z)^2}$), and

$$h^k = \begin{pmatrix} f(k\Delta t) \cdot \frac{(\Delta t)^2}{\Delta z} \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The vector u^{k+1} represents the displacement at times $t = k\Delta t$ at discrete node points $z = j\Delta z$ as in the heat equation example. The matrix $A(x)$ is given by:

$$A(x) = \begin{pmatrix} 2 - \lambda(x_1 + x_2) & \lambda(x_1 + x_2) & & \\ \frac{\lambda}{2}(x_1 + x_2) & 2 - \frac{\lambda}{2}(x_1 + 2x_2 + x_3) & \frac{\lambda}{2}(x_2 + x_3) & \\ & \ddots & \ddots & \\ & & \lambda(x_{N-1} + x_N) & 2 - \lambda(x_{N-1} + x_N) \end{pmatrix}.$$

The first iteration can be started by using the boundary condition $u^0 = 0$, also u^{-1} is assumed to be identically zero.

The measurements are made at earth's surface ($z = 0$), i.e., the measured data is the vector $u^1(0), u^2(0), \ldots u^M(0)$, and can be represented by vector $Z = e_1 e_1^T u^1 + e_2 e_1^T u^2 + \ldots + e_M e_1^T u^M$. We assume that $M \geq N$ to avoid an underdetermined system for the solution of x.

If $M = N$, the inverse problem is solved by solving the nonlinear equation,

$$F(x) = Z(x) - z_{target} = 0,$$

but if $M > N$, we look for a solution which minimizes the error by solving the following least squares problem,

$$min_x \quad f(x) = \|Z(x) - z_{target}\|_2.$$

We shall discuss only the nonlinear equation setting in this section. For an illustration of application of EASE for nonlinear least squares setting, please refer to the Appendix.

5. Exploiting Structure in Computation

Define function $u^+ = F(x, u, u^-) = -u^- + A(x)u + h$. Applying EASE, it is seen that the extended function can be written as shown in Figure 6.

The extended Jacobian is given by:

$$J_E = \left(\begin{array}{c|ccccc} F_x(x, u^0, u^{-1}) & -I & & & & \\ F_x(x, u^1, u^0) & A(x) & -I & & & \\ \vdots & \ddots & \ddots & \ddots & & \\ F_x(x, u^{M-2}, u^{M-3}) & & -I & A(x) & -I & \\ F_x(x, u^{M-1}, u^{M-2}) & & & -I & A(x) & -I \\ \hline 0 & e_1 e_1^T & e_2 e_1^T & \cdots & & e_M e_1^T \end{array}\right)$$

$$\boxed{\begin{array}{l} \textit{Solve for } u^1 : \ u^1 = F(x, u^0, u^{-1}) \\ \textit{Solve for } u^2 : \ u^2 = F(x, u^1, u^0) \\ \vdots \\ \textit{Solve for } u^M : \ u^M = F(x, u^{M-1}, u^{M-2}) \\ \textit{Final } Z = e_1 e_1^T u^1 + e_2 e_1^T u^2 + \ldots + e_M e_1^T u^M - z_{target} \end{array}}$$

Figure 6: Wave equation extended function

Again, we get all the benefits provided by EASE. In particular, it is relatively inexpensive to compute the Newton step Δx.

5.1. Numerical results

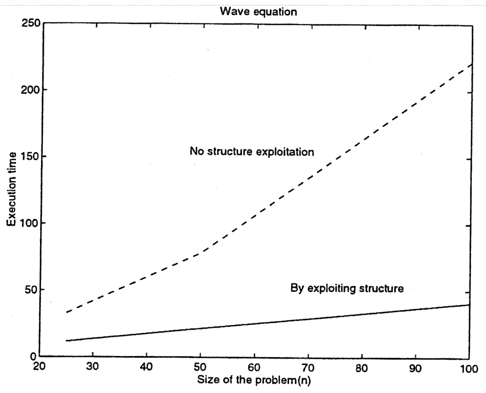

Figure 7: Numerical Results for the wave equation problem

The graph in Figure 7 shows the effectiveness of EASE compared to the unstructured way of computing the Jacobian and Newton step. We get results very similar to the heat conductivity problems results. The experiment was performed employing the ADMIT-2 software which is described briefly in the next section.

6. ADMIT-2 Toolbox

As a part of project ADMIT (Automatic Differentiation and MATLAB Interface Toolbox), we are currently developing two MATLAB toolboxes, ADMIT-1 and ADMIT-2[1]. ADMIT-1 allows for efficient computation of sparse Jacobian/Hess-

[1]see `http://www.cs.cornell.edu/Info/People/verma/AD/research.html`

ian matrices using the ADOL-C tool [4] for Automatic differentiation. ADMIT-2 builds on ADMIT-1, and provides efficient methods to compute the extended derivative matrices and the Newton step implementing EASE. ADMIT-2 recognizes different classes of real-world structured problems, such as inverse problems, partially separable problems, discrete-time optimal control problems etc. A block-diagram of ADMIT-2 is shown in Figure 8.

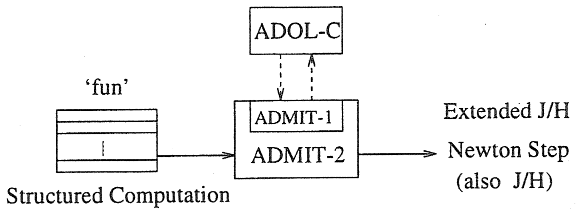

Figure 8: Block Diagram of ADMIT-2 MATLAB toolbox

Sample Usage : ADMIT-2 has a very simple and easy-to-understand interface. evalJExt is an ADMIT-2 function to compute the extended Jacobian and the Newton step. The first argument of evalJExt is the class of the structured computation, GI in this case stands for generalized inverse problems.

```
>>
>>x=ones(100,1);              <- the initial conductivity
>>initval=ones(100,1);        <- the initial temperature
>>yfinal= 2*ones(100,1);      <- the final (desired) temp. at t=1
>>fdata = MakeExtFdata('GI','heq',initval,yfinal);
>>[f,eJ,ns]=evalJExt('GI','heq',x,fdata);
>>
```

`eJ` denotes the extended Jacobian output, `ns` denotes the Newton step. The "user-computation", the time stepping procedure is assumed to be written in file heq.m.

7. Conclusions

We have demonstrated the efficient application of automatic differentiation to inverse design problems. The basic idea is to recognize and use the dominant program structure and then selectively apply AD. Preliminary numerical results indicate that these ideas can lead to an order of magnitude improvement in the computation of the Newton step.

In both the examples discussed in this paper the structure exploitation was coarse-grained: the overall time stepping procedure defined the level of granularity which we used to intertwine with the AD process. However, in some cases in may be pragmatic to apply AD at a finer level of granularity, for example by exploiting the stencil update. We are currently developing a detailed example of this sort.

References

[1] T. F. Coleman and A. Verma. *Structure and efficient Hessian calculation*, Tech. Report TR96-258, Cornell Theory Center, Cornell University, September 1996.

[2] T. F. Coleman and A. Verma. *Structure and efficient Jacobian calculation*, in Computational Differentiation: Techniques, Applications, and Tools, M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds., SIAM, Philadelphia, Penn., 149–159, 1996.

[3] T. F. Coleman and A. Verma. *The efficient computation of sparse Jacobian matrices using automatic differentiation*, SISC, (1997 (to appear)).

[4] A. Griewank, D. Juedes and J. Utke. *ADOL–C, a package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. On Math. Software, 22:131–167, 1996.

[5] F. Santosa and W. W. Symes. *Computation of the Hessian for least-squares solutions of inverse problems of reflection seismology*, Inverse problems, 4:211–233, 1988.

[6] N. Zabaras, S. Mukherjee and O. Richmond. *An analysis of inverse heat transfer problems with phase changes using an integral method*, Transactions of American Society of Mechanical Engineers (ASME), 110:554–561, 1988.

Appendix

A. Applying EASE in the Least Square's Setting

We illustrate how to solve the Newton step in the nonlinear least squares setting, using EASE. In particular, we'll consider the reflection seismology inverse problem, but the ideas are applicable to inverse problem in general.

The problem has the form

$$min_x f(x) = \|Z(x) - z_{desired}\|_2^2$$

where $Z = e_1 e_1^T u^1 + e_2 e_1^T u^2 + \ldots + e_M e_1^T u^M$, and u^i represents the state vector at timestep i.

The gradient of this function w.r.t x is given by

$$\nabla f(x) = J^T (Z(x) - z_{desired})$$

where J is the Jacobian matrix of u^M w.r.t x. The gradient can be computed using application of reverse mode of automatic differentiation. ADMIT-2 provides a method to compute the gradient efficiently.

A.1. Gauss Newton step

The Gauss-Newton step is solved by

$$J^T J s = -\nabla f(x) \tag{19}$$

Equation (19) can be efficiently solved for, without forming J or $J^T J$ explicitly, using the extended Jacobian matrix J_E. EASE tells us that the extended function is as shown in Figure 9.

$$\boxed{\begin{array}{l}
\textit{Solve for } u^1 : \; u^1 = F(x, u^0, u^{-1}) \\
\textit{Solve for } u^2 : \; u^2 = F(x, u^1, u^0) \\
\vdots \\
\textit{Solve for } u^M : \; u^M = F(x, u^{M-1}, u^{M-2}) \\
\textit{Final } Z = e_1 e_1^T u^1 + e_2 e_1^T u^2 + \ldots + e_M e_1^T u^M - z_{desired}
\end{array}}$$

Figure 9: Wave equation extended function

The extended Jacobian is given by :

$$J_E = \left(\begin{array}{c|ccccc} F_x(x,u^0,u^{-1}) & -I & & & & \\ F_x(x,u^1,u^0) & A(x) & -I & & & \\ \vdots & \ddots & \ddots & \ddots & & \\ F_x(x,u^{M-2},u^{M-3}) & & -I & A(x) & -I & \\ F_x(x,u^{M-1},u^{M-2}) & & & -I & A(x) & -I \\ \hline 0 & e_1e_1^T & e_2e_1^T & \cdots & & e_Me_1^T \end{array}\right) = \left(\begin{array}{c|c} A & L \\ \hline B & M \end{array}\right)$$

Then the Gauss-Newton step can be solved by solving system (20)

$$\begin{pmatrix} A & L & 0 & 0 \\ B & M & -I & 0 \\ 0 & 0 & M^T & L^T \\ 0 & 0 & B^T & A^T \end{pmatrix} \begin{pmatrix} s \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -\nabla f(x) \end{pmatrix} \tag{20}$$

It is easy to show that s solves $J^T Js = -\nabla f(x)$. Hence to solve the Newton step we solve the (sparse) extended system (20). Expectedly, it turns out that system (20) can be made symmetric after permutations as shown in equation (21).

$$\begin{pmatrix} -I & 0 & M & B \\ 0 & 0 & L & A \\ M^T & L^T & 0 & 0 \\ B^T & A^T & 0 & 0 \end{pmatrix} \begin{pmatrix} v_2 \\ v_3 \\ v_1 \\ s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -\nabla f(x) \end{pmatrix} \tag{21}$$

A.2. Complete Newton step

For the complete Newton step, we also want to take into account the 2nd derivative information, i.e., solve

$$Hs = -\nabla f(x),$$

where H denotes the Hessian matrix. EASE allows us to do this efficiently, without forming the true Hessian, using the extended Hessian formulation [1].