Cornell Theory Center Technical Report CTC96-TR258

Structure and Efficient Hessian Calculation *

Thomas F. Coleman† Arun Verma‡

August 26, 1996

Abstract

Modern methods for numerical optimization calculate (or approximate) the matrix of second derivatives, the Hessian matrix, at each iteration. The recent arrival of robust software for automatic differentiation allows for the possibility of automatically computing the Hessian matrix, and the gradient, given a code to evaluate the objective function itself. However, for large-scale problems direct application of automatic differentiation may be unacceptably expensive. Recent work has shown that this cost can be dramatically reduced in the presence of sparsity. In this paper we show that for *structured* problems it is possible to apply automatic differentiation tools in an economical way – *even in the absence of sparsity in the Hessian.*

Keywords: Hessian matrix, automatic differentiation, structured computation, sparsity.

1 Introduction

Calculation or approximation of the matrix of second derivatives, the Hessian matrix, is an important part of modern methods for continuous minimization. Approximation schemes have been particularily popular, e.g., quasi-Newton methods and finite differencing, partly because they do not require, from the user, a code to evaluate the Hessian matrix. Conversely, methods that use exact second derivatives are less popular, despite stronger convergence support, partly due to an (apparently) onerous demand on the user: supply a code to evaluate the n-by-n Hessian matrix, H.

However, with the advent of automatic differentiation (AD) tools this balance is now being challenged. It is now possible to have first and second derivatives automatically computed given a code that computes the objective function. The difficulty is one of computational cost: straightforward application of automatic differentiation tools may be inordinately expensive for large problems. Results obtained in [1, 6] show that for the related *sparse Jacobian* problem, the cost can be dramatically reduced if sparsity is exploited. In principle similar techniques [5] can be applied to the sparse

*Presented at the *International Conference on Nonlinear Programming, Beijing, September, 1996.* 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-FG02-90ER25013, and in part by the Advanced Computing Research Institute, a unit of the Cornell Theory Center which receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of its Corporate Research Institute.

†Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca NY 14850.

‡Computer Science Department, Cornell University, Ithaca NY 14850.

Hessian determination problem in a straightforward manner. An extension of the sparse techniques to problems with dense but "structured" Jacobian matrices is given in [7].

What can be said of the case where H is both large and dense? The point of this paper is to indicate that if the computation of the objective function $f(x)$ is a "structured" computation it is possible to compute H, or perhaps the Newton step $s = -H^{-1}\nabla f(x)$, using automatic differentiation and the computation can be done economically. Moreover, many (if not most) large-scale optimization problems are the result of structured computations.

First we briefly review our work [7] on determining Jacobian matrices of structured vector-valued mappings $F : \Re^n \rightarrow \Re^m$. In this case we are interested in computing the m-by-n Jacobian matrix $J(x)$, or perhaps the Newton step $s = -J^{-1}F(x)$. We begin with an illustrative example.

Suppose that the evaluation of $F(x)$ represents a *composite* computation:

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

where y is the solution to a large sparse positive definite system,

$$Ay = \tilde{F}(x), \tag{2}$$

and $A = A(x)$. Notice that the Jacobian of $F(x)$, $J(x)$, will likely be dense even when matrices $\bar{J}_x, \bar{J}_y, \tilde{J}$, and $A_x y$ are sparse (which is typical) where $\bar{J}_y$ is the Jacobian of $\bar{F}$ with respect to y, $\bar{J}_x$ is the Jacobian of $\bar{F}$ with respect to x, $\tilde{J}$ is the Jacobian of $\tilde{F}$, and $A_x y$ is the Jacobian of the mapping $A(x)y$ (with respect to x). To see this consider that

$$J = \bar{J}_x + \bar{J}_y A^{-1}[\tilde{J} - A_x y]. \tag{3}$$

In general, the application of A^{-1} causes matrix J to be dense.

So, direct application of *sparse AD* techniques offers no advantage in this case. However, it is possible to exploit the structure of this composite function and apply the sparse AD techniques at a deeper level. To see this consider the following "program" to evaluate $z = F(x)$, given x:

$$\boxed{\begin{array}{lrcl} \text{“Solve” } for\ y_1: & y_1 - \tilde{F}(x) & = & 0 \\ Solve\ for\ y_2: & Ay_2 - y_1 & = & 0 \\ \text{“Solve” } for\ z: & z - \bar{F}(x, y_2) & = & 0. \end{array}}$$

But this program can be viewed as a nonlinear system of equations in (x, y_1, y_2) with corresponding Newton equations,

$$J_E \begin{pmatrix} \delta_x \\ \delta_{y_1} \\ \delta_{y_2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -F(x) \end{pmatrix}, \tag{4}$$

where

$$J_E = \begin{bmatrix} -\tilde{J} & I & 0 \\ A_x y_2 & -I & A \\ -\bar{J}_x & 0 & -\bar{J}_y \end{bmatrix}. \tag{5}$$

The point here is that the "extended" Jacobian matrix J_E is sparse and clearly sparse AD techniques, e.g., [1, 6, 9] can be applied with respect to

$$F_E(x, y) = \begin{pmatrix} y_1 - \tilde{F}(x) \\ A(x)y_2 - y_1 \\ -\bar{F}(x, y_2) \end{pmatrix} \tag{6}$$

to efficiently determine J_E. For example, the work required by the bi-coloring technique developed in [6] is of order $\chi \cdot \omega(F_E) = \chi \cdot \omega(F)$ where χ is a "bi-chromatic number" dependent on the sparsity of J_E, and $\omega(\cdot)$ denotes the work required to evaluate the argument. Typically, $\chi << \min(m,n)$. Additional linear algebra work is needed to extract J from J_E: compute the Schur complement (introducing zero matrices in positions $(3,2),(3,3)$) and obtain,

$$J = \bar{J}_x + \bar{J}_y A^{-1}[\tilde{J} - A_x y].$$

If it is the Newton step $\delta_x = -J^{-1}F(x)$ that is required then it is not necessary to explicitly form J. For example, the extended system (4) can be solved directly. It is also possible to compute an approximate Newton step, without forming J, using an iterative solver. Specifically, if a sparse factorization of A is computed, an iterative solver involving only matrix-vector products can be applied to:

$$(\bar{J}_x + \bar{J}_y A^{-1}[\tilde{J} - A_x y])s = -F(x). \tag{7}$$

In addition to the composite function example given above, many important classes of large-scale problems are naturally programmed in the structured fashion illustrated in Figure 1, e.g., dynamical systems, partially separable functions, systems related to boundary value problems, neural network evaluations, and product functions [7]. The crucial observation here is that while the Jacobian of F is often dense in these cases, the Jacobian of the extended function F_E, J_E, is typically very sparse. Hence, the sparse AD techniques developed in [6], for example, can be applied in combination with AD software to compute J_E in an efficient manner.

$$\boxed{\begin{array}{l}
\textit{Solve for } y_1: \; F^1(x, y_1) = 0 \\
\textit{Solve for } y_2: \; F^2(x, y_1, y_2) = 0 \\
\vdots \\
\textit{Solve for } y_p: \; F^p(x, y_1, y_2, \ldots, y_p) = 0 \\
\textit{``Solve'' for output } z: \; z - F^{p+1}(x, y_1, y_2, \ldots, y_p) \; = \; 0
\end{array}}$$

FIG. 1. *A General Structured Computation*

We conclude this section with three remarks. First, given that J_E is computed, a standard linear algebra computation can yield J, if required. Alternatively, if it is the Newton step $s = -J^{-1}F(x)$ that is required, or perhaps an approximation, then it is possible to work with J_E directly. Second, the structural ideas discussed above can, of course, be applied to the special case of gradient computation. This can be particularily useful when only the forward mode of AD is available, e.g., [2]. Third, the structural ideas discussed above can also be applied to the case where $F(x)$ is a gradient function, $\nabla f(x)$, of a scalar-valued function $f(x)$. In this case the computed Jacobian matrix of F corresponds to the Hessian matrix of f. However, in general it is not convenient to supply a structured program to evaluate $\nabla f(x)$ - it is preferable to work directly with the f-evaluation program, if possible.

1.1 Example: A Composite Function

Suppose that the evaluation of $z = f(x) = \bar{f}(x,y)$ is a composite function:

$$\boxed{\begin{array}{rrcl}
\textit{Solve for } y: & Ay - \tilde{F}(x) & = & 0 \\
\textit{``Solve'' for } z: & z - \bar{f}(x,y) & = & 0.
\end{array}}$$

It is easy to see that the gradient of f, with respect to x, is given by,

$$(\nabla f)^T \;=\; \nabla_x \bar{f}^T + \nabla_y \bar{f}^T A^{-1}[\tilde{J} - A_x y].$$

Therefore, the code to evaluate the gradient can be written in "extended" form, $GF_E(x, y, w) = 0$, i.e.,

$$\boxed{\begin{array}{lll} \text{“Solve” } for\ y: & Ay - \tilde{F}(x) \;=\; 0 \\ \text{“Solve” } for\ w: & \nabla_y \bar{f} + A^T w \;=\; 0 \\ \text{“Solve” } for\ \nabla f: & \nabla f - [(A_x y) - \tilde{J}]^T w - \nabla_x \bar{f} \;=\; 0. \end{array}}$$

Note that $GF_E(x, y, w)$ can be differentiated with respect to all variables to yield an extended Hessian matrix H_E:

$$H_E \;=\; \begin{bmatrix} A_x y - \tilde{J} & A & 0 \\ A_x^T w + \nabla^2_{yx} \bar{f} & \nabla^2_{yy} \bar{f} & A^T \\ w^T[(A_x y) - \tilde{J}]_x + \nabla^2_{xx} \bar{f} & w^T A_x + \nabla^2_{xy} \bar{f} & (A_x y - \tilde{J})^T \end{bmatrix}. \tag{8}$$

It is quite likely that the extended matrix H_E will be sparse. Moreover, a symmetric form can be obtained with a simple permutation:

$$H_E^S \;=\; \begin{bmatrix} 0 & A & A_x y - \tilde{J} \\ A^T & \nabla^2_{yy} \bar{f} & A_x^T w + \nabla^2_{yx} \bar{f} \\ (A_x y - \tilde{J})^T & w^T A_x + \nabla^2_{xy} \bar{f} & w^T[A_x y - \tilde{J}]_x + \nabla^2_{xx} \bar{f} \end{bmatrix}. \tag{9}$$

The matrix $w^T[A_x y - \tilde{J}]_x$ is symmetric because it represents the second derivative matrix, with respect to x, of the function $w^T(Ay - \tilde{F}(x))$.

The Hessian matrix with respect to the the original variables x, can be derived from the 3-by-3 block matrix H_E by eliminating, via block Gauss row-transformations, blocks $(3, 2)$ and $(3, 3)$. This yields:

$$\begin{aligned} H = w^T[(A_x y) - \tilde{J}]_x + \nabla^2_{xx} \bar{f} - [(A_x^T w + \nabla^2_{yx} \bar{f})^T A^{-1}(A_x y - \tilde{J}) - \\ (A_x y - \tilde{J})^T A^{-T} \nabla^2_{yy} \bar{f} A^{-1}(A_x y - \tilde{J}) + \\ (A_x y - \tilde{J})^T A^{-T}(A_x^T w + \nabla^2_{yx} \bar{f})]. \end{aligned}$$

There are three important observations to make about this example. First, the matrix H is likely to be dense, due to the action of A^{-1}, whereas under reasonable assumptions H_E will be sparse. Second, matrix H_E can be obtained using automatic differentiation applied to function $GF_E(x, y, w)$. Sparse AD techniques [6] can be applied to $GF_E(x, y, w)$ to allow for the economical calculation of H_E. Third, it is also possible to obtain matrix H_E without *explicitly* applying a sparse AD technique to the structured gradient function GF_E. We have in mind the following recipe:

1. "Solve" for y, differentiate : $\quad Ay - \tilde{F}(x) \;=\; 0$
2. Solve for w : $\quad \nabla_y \bar{f} + A^T w \;=\; 0$
3. Determine the sparse Hessian matrix of $\bar{f} \;+\; w^T[Ay - \tilde{F}(x)]$ with respect to x, y.

This last observation indicates how to use sparse AD techniques to compute H_E given a structured program to evaluate f (as opposed to a structured representation of $\nabla f(x)$). We will see in Section 2.2 that this recipe can be generalized.

1.2 Numerical Experiments

In this section we report on a small experiment to illustrate the advantages of our proposed aproach. We consider a composite function of the form described above. In particular, the function $\tilde{F}$ is defined to be the Broyden [4] function (the Jacobian is tridiagonal). Function $\bar{f}$ is chosen to be a simple scalar-valued function with a tridiagonal Hessian matrix. The structure of A is based on the 5-point Laplacian defined on a regular $\sqrt{n}$-by-$\sqrt{n}$ grid. Each nonzero element of $A(x)$ depends on x in a trivial way such that the structure of matrix $A_x \cdot v$, for an arbitrary vector v, is equal to the structure of matrix A. In particular, for all (i,j), $i \neq j$ where A_{ij} is nonzero the function $A_{ij}(x)$ is defined, $A_{ij} = x_j$.

Experiments reported below were performed on a Sun Sparc 5 under the Solaris operating system in a MATLAB [10]/C environment.

Experiment 1 **: Computing H_E versus H**

In Figure 2 we compare the time to compute H directly, i.e., applying automatic differentiation directly to the function f to obtain the Hessian matrix H, versus the sparse AD computation of H_E using the bi-coloring technique proposed in [6]. Experiments were performed using the AD-software package ADOL-C [8].

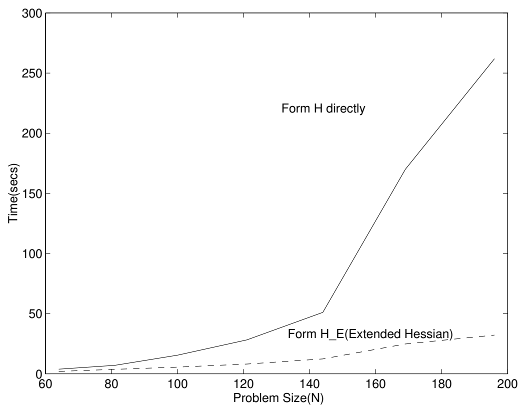

FIG. 2. *ADOL-C experiment*

Clearly, exploiting sparsity is a big win and the advantage grows with problem size. Of course the matrix H_E is not usually an end in itself - if matrix H is required then a standard block elimination computation can be applied to H_E to yield H. However, often matrix H is not really required either – the objective may be to compute (or approximate) the Newton step, $s = H^{-1}\nabla f(x)$. In this case it may be advantageous to work with H_E directly. This remark is explored in the next experiment.

Experiment 2 **: Computing the Newton step, given H_E, in two ways**

Figure 3 plots the time required by two different ways of calculating the Newton step, given H_E. Method 1 – the dashed line – corresponds to first computing the Hessian matrix from H_E using standard block elimination and then doing a system solve with the dense matrix H. The second method involves solving a sparse system with matric H_E directly (using the MATLAB "backslash" function).

Clearly in this case a direct solve using the extended matrix H_E is preferable for all sufficiently large n.

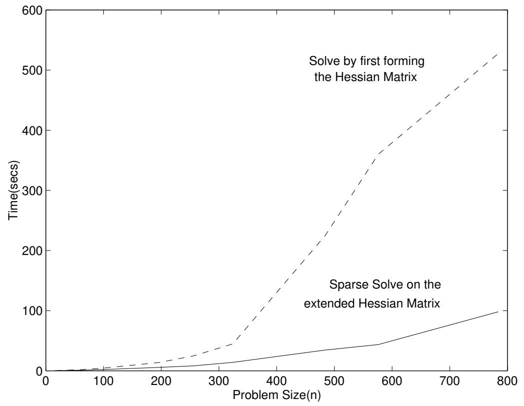

FIG. 3. *Comparison of two approaches to calculate the Newton step*

2 Structure

We believe that most large-scale objective functions in optimization are naturally expressed, at a high-level, in a structured form. In particular, the program to evaluate $z = f(x)$ can be written in the form given in Figure 1, where equation i *uniquely* determines (intermediate) vector y_i. So, a compact program to evaluate f is given in Figure 4

Solve for y : $\tilde{F}^E(x, y) = 0$
"Solve" for output z : $z - \bar{f}(x, y_1, y_2, \ldots, y_p) \;=\; 0$

FIG. 4. *Structured f-Evaluation in Compact Form*

where

$$\tilde{F}^E = \begin{pmatrix} F^1 \\ F^2 \\ \vdots \\ F^p \end{pmatrix}.$$

The component functions of $\tilde{F}^E$, F^i, $i = 1 : p$, defined in Figure 1, are usually conveniently available to the user. It is important to note that intermediate vector y_i is (uniquely) determined by component function F^i, a function of $x, y_1, ..., y_i$. Clearly the composite function example described in Section 1 is a special case of this general form using just one intermediate vector y. Other examples are considered in Section 3.

Differentiation of this program with respect to the original variables x as well as the intermediate variables y yields an "extended" Jacobian matrix:

$$J_E = \begin{pmatrix} \tilde{F}^E_x & \tilde{F}^E_y \\ \nabla_x \bar{f}^T & \nabla_y \bar{f}^T \end{pmatrix}. \tag{10}$$

Typically the Jacobian matrix $\tilde{J}^E = (\tilde{F}^E_x, \tilde{F}^E_y)$ is sparse and so sparse AD techniques can be applied to function $\tilde{F}^E$ to obtain this derivative information efficiently. Note also that $\tilde{F}^E_y$ is block lower-triangular, and, due to the assumption that intermediate vectors y are uniquely determined, $\tilde{F}^E_y$ is nonsingular.

The first question we address in this section is how to write a structured program to evaluate the gradient of f, $\nabla_x f$, such that automatic differentiation can be applied to yield second-derivative information in an efficient way. Once we have sorted this out, we take a step backwards and consider the more practical concern: how do we apply automatic differentiation directly to the structured program that evaluates f to yield the Hessian matrix, or perhaps the Newton step, in an efficient way.

2.1 How to Differentiate the Gradient Function

To answer the first question, the gradient of the structured function f can be evaluated as illustrated in Figure 5.

1. Differentiate $\tilde{F}^E$ *yielding* $\tilde{J}^E = (\tilde{F}^E_x, \tilde{F}^E_y)$
2. Solve $(\tilde{F}^E_y)^T w = -\nabla_y \bar{f}$.
3. Set $\nabla_x f = \nabla_x \bar{f} + (\tilde{F}^E_x)^T \cdot w$.

FIG. 5. *A Structured Gradient program*

The derivation of this program is simple: First differentiate the extended function F_E to obtain J_E; then, eliminate the $(2,2)$-block, $\nabla_y \bar{f}^T$, to define vector w. Finally, modify the $(2,1)$-block of matrix J_E using w to get $\nabla_x f$. In other words, form matrix J_E (10) and then eliminate the $(2,2)$-block using a block Gaussian transformation.

Inspired by this simple program to evaluate the gradient of f, we define an "extended" gradient GF_E, a vector function of the triple (x, y, w):

$$GF_E(x,y,w) = \begin{pmatrix} \tilde{F}^E \\ (\tilde{F}^E_y)^T w + \nabla_y \bar{f} \\ \nabla_x \bar{f} + (\tilde{F}^E_x) w \end{pmatrix}.$$

In principle the vector function GF_E can be differentiated, with respect to (x, y, w) to yield a Newton system,

$$H_E \begin{pmatrix} \delta x \\ \delta y \\ \delta w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -\nabla_x f \end{pmatrix}$$

where

$$H_E = \begin{pmatrix} \tilde{F}^E_x & \tilde{F}^E_y & 0 \\ (\tilde{F}^E_{yx})^T w + \nabla^2_{yx} \bar{f} & (\tilde{F}^E_{yy})^T w + \nabla^2_{yy} \bar{f} & (\tilde{F}^E_y)^T \\ (\tilde{F}^E_{xx})^T w + \nabla^2_{xx} \bar{f} & (\tilde{F}^E_{xy})^T w + \nabla^2_{xy} \bar{f} & (\tilde{F}^E_x)^T \end{pmatrix}.$$

Typically matrix H_E exposes more sparsity than matrix H – the composite function described in Section 1 is a good illustration. Moreover, additional sparsity gains can often be achieved, in principle, if the structure in Step 3. of the gradient-evaluation program is exploited. In particular, notice that the computation $\nabla_x f = \nabla_x \bar{f} + (\tilde{F}^E_x)^T \cdot w$ exhibits "partially separable" form. As illustrated in [7] it is often worthwhile to further break down this step:

3.1 Compute $u_i = (\tilde{F}^E_x(i,:))^T \cdot w_i, for \quad i = 1:p.$
3.2 Assign $\nabla_x f = \nabla_x \bar{f} + \sum u_i.$

Here $\tilde{F}^E_x(i,:)$ represents the ith "block" row of the Jacobian $\tilde{F}^E_x$, i.e $\tilde{F}^E_x(i,:) \equiv F^i_x$. Vectors $w_i, i = 1, \ldots, p$ form a *partition* of vector w.

The result of differentiating this “refined” program leads to a larger, sparser, extended Hessian matrix H_{EE} ,

$$H_{EE} = \begin{pmatrix} \tilde{F}_x^E & \tilde{F}_y^E & 0 & 0 & 0 & 0 & 0 & 0 \\ (\tilde{F}_{yx}^E)^T w + \nabla_{yx}^2 \bar{f} & (\tilde{F}_{yy}^E)^T w + \nabla_{yy}^2 \bar{f} & (F_y^1)^T & \dots & (F_y^p)^T & 0 & 0 & 0 \\ (F_{xx}^1)^T \cdot w_1 & (F_{xy}^1)^T \cdot w_i & (F_x^1)^T & 0 & 0 & -I & 0 & 0 \\ \vdots & \vdots & 0 & \ddots & 0 & 0 & \ddots & 0 \\ (F_{xx}^k)^T \cdot w_k & (F_{xy}^k)^T \cdot w_k & 0 & 0 & (F_x^k)^T & 0 & 0 & -I \\ \nabla_{xx}^2 \bar{f} & \nabla_{xy}^2 \bar{f} & 0 & 0 & 0 & I & \cdots & I \end{pmatrix}$$

where the differentiation is done with respect to the (ordered) variables $x, y, w_1, \dots, w_p, u_1, \dots, u_p$.

The point here is that due to increase in sparsity, it is often more economical to directly compute the “super-extended” matrix H_{EE}.

2.2 How to Differentiate f (Twice)

If we define $g(x, y, w) = \bar{f} + w^T \tilde{F}^E(x, y)$ then H_E can be written:

$$H_E = \begin{pmatrix} \tilde{F}_x^E & \tilde{F}_y^E & 0 \\ \nabla_{yx}^2 g & \nabla_{yy}^2 g & (\tilde{F}_y^E)^T \\ \nabla_{xx}^2 g & \nabla_{xy}^2 g & (\tilde{F}_x^E)^T \end{pmatrix}. \tag{11}$$

This is an important observation because it yields the answer to the second major question of Section 2: How do we apply automatic differentiation directly to the f-evaluation code to yield the extended Hessian H_E in an efficient way? The recipe follows from (11) and the definition of w:

1. Using the sparse AD techniques developed in [7] compute the extended Jacobian $(\tilde{F}^E)$
2. Solve the block lower triangular system for w: $(\tilde{F}_y^E)^T w + \nabla_y \bar{f} = 0$.
3. Using sparse AD techniques, twice differentiate $g(x, y, w) = \bar{f} + w^T F^E(x, y)$, with respect to x, y, to determine the Hessian matrix, i.e., H_E. As indicated in Section 2.1, it can be advantageous to exploit the partially separable structure in $g(x, y, w) = \bar{f} + \sum_{i=1}^k w_i^T F_i^E$: i.e., compute the Hessian matrix of each component function $w_i^T F_i^E$ in turn.

We conclude this section with two simple observations. First, the (reduced) Hessian matrix H is available from H_E through a simple block-elimination procedure. For, example if we partition H_E,

$$H_E = \left(\begin{array}{c|c} A & L \\ \hline B & M \end{array}\right) \tag{12}$$

then $H = B - ML^{-1}A$.

Second, symmetry in the extended form H_E can be achieved with (block) permutations:

$$H_E^S = \begin{pmatrix} 0 & \tilde{F}_y^E & \tilde{F}_x^E \\ (\tilde{F}_y^E)^T & \nabla_{yy}^2 g & \nabla_{yx}^2 g \\ (\tilde{F}_x^E)^T & \nabla_{xy}^2 g & \nabla_{xx}^2 g \end{pmatrix}.$$

3 Examples

In this section we illustrate the application of the structural ideas developed in the previous section with two common classes of structured optimization problems.

3.1 General Composite Functions and Dynamical Systems

A general composite function $f : \Re^n \to \Re^1$ can be written

$$z = f(x) = \bar{f}(T_p(T_{p-1}(\dots(T_1(x))\dots)), \tag{13}$$

where, in general, functions $T_i, i = 1 : p$ are vector maps, while $\bar{f}$ is a scalar map. This formulation is very common, for example, in weather simulations. A natural high-level program to evaluate F is given below, where we let y_0 denote x:

$$\boxed{\begin{array}{l} \textit{for } i = 1:p \\ \qquad \textit{“Solve” for } y_i\textit{:}\ y_i - T_i(y_{i-1}) \;=\; 0 \\ \textit{“Solve” for } z\textit{:}\ z - \bar{f}(y_p) \;=\; 0. \end{array}}$$

Clearly this is a special case of the general extended form given in Figure 1, where we define

$$\tilde{F}^E = \begin{pmatrix} T_1(x) - y_1 \\ T_2(y_1) - y_2 \\ \vdots \\ T_k(y_{k-1}) - y_k \end{pmatrix}.$$

The general recipe given in Section 2.2 can be applied here, followed by a simple permutation, to yield the (symmetric) extended Hessian matrix H_E^S. If we define $g = \bar{f} + w^T\tilde{F}^E$ then

$$H_E^S = \begin{pmatrix} 0 & \tilde{F}_y^E & \tilde{F}_x^E \\ (\tilde{F}_y^E)^T & \nabla^2_{yy} g & \nabla^2_{yx} g \\ (\tilde{F}_x^E)^T & \nabla^2_{xy} g & \nabla^2_{xx} g \end{pmatrix}.$$

Additional structure can be gleaned by closer examination of the submatrices comprising H_E^S, i.e., exploiting the partial separable structure of $w^T F^{\tilde{E}}$:

$$H_{EE}^S = \begin{pmatrix} 0 & \cdots & \cdots & 0 & -I & & & & J_1 \\ \vdots & & & \vdots & J_2 & -I & & & \\ \vdots & & & \vdots & & \ddots & \ddots & & \\ 0 & \cdots & \cdots & 0 & & & J_k & -I & \\ -I & (J_2)^T & & & \nabla^2_{y_1,y_1} g & & & & \\ & -I & \ddots & & & \nabla^2_{y_2,y_2} g & & & \\ & & \ddots & (J_k)^T & & & \ddots & & \\ & & & -I & & & & \nabla^2_{y_k,y_k} & \\ (J_1)^T & & & & & & & & \nabla^2_{x,x} g \end{pmatrix}$$

Due to the high degree of sparsity, sparse AD techniques can be very effective in the direct determination of H_{EE}^S.

3.2 Generalized partial separability

We define a *generalized* partially separable vector-valued function,

$$f(x) = \bar{f}(y_1, y_2, \ldots, y_p); \qquad y_i = T_i(x), \quad i = 1, 2, \ldots, p. \tag{14}$$

Note that if function $\bar{f}$ is simply a summation then f reduces to the usual notion of partial separability.

Following the general form given in Figure 1, F can be computed with the following program,

$$\boxed{\begin{array}{l} \textit{for } i = 1 : p \\ \qquad \textit{“Solve” for } y_i\textit{: } y_i - T_i(x) \;=\; 0 \\ \textit{“Solve” for } z : z - \bar{f}(y_1, y_2, \ldots, y_p) \;=\; 0. \end{array}}$$

Of course this program can be inefficient if some of the functions T_i share common sub-expressions. Therefore a more general program can be written if we define a “stacked” vector $Y^T = (y_1^T, \ldots y_p^T)$ and a corresponding vector function

$$\tilde{F}(x) = \begin{pmatrix} T_1(x) \\ T_2(x) \\ \vdots \\ T_p(x) \end{pmatrix}.$$

This yields the simple 2-liner:

$$\boxed{\begin{array}{l} \textit{“Solve” for } Y : \; Y - \tilde{F}(x) = 0 \\ \textit{“Solve” for } z : \; z \;=\; \bar{f}(y_1, y_2, \ldots, y_p). \end{array}}$$

Therefore the structured program to evaluate f is a particular case of the general form illustrated in Figure 1 and the general recipe given in Section 2.2 can be applied.

Note that if $g = \bar{f} + w^T \tilde{F}^E$ then $\nabla^2_{xy} g = \nabla^2_{yx} g = 0$; therefore, there is additional structure in the extended Hessian matrix:

$$H_E^S = \begin{pmatrix} 0 & -I & \tilde{F}_x \\ -I & \nabla^2_{yy} g & 0 \\ (\tilde{F}_x)^T & 0 & \nabla^2_{xx} g \end{pmatrix}.$$

4 Conclusions

The arrival of robust, reliable automatic differentiation tools, e.g., [3, 8], is a major new development in scientific computing. The potential impact on numerical optimization is enormous.

This paper is concerned with the efficient determination of Hessian matrices, and Newton steps, in large-scale optimization problems. If there is sparsity in the Hessian matrix then graph coloring techniques [5, 6] can be used to guide the use of AD software – the efficiency gains can be significant. However, our thesis is that many large-scale problems exhibit structure at a high, accessible, level. Such problems often have dense Hessian matrices, rendering direct application of sparse AD techniques impotent. However, differentiation of a *structured* program to evaluate the objective function often exposes sparsity, at a deeper level, and thereby allows for the efficient application of sparse AD technology.

References

[1] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle, and A. Griewank, *Computing large sparse Jacobian matrices using automatic differentiation*, SIAM Journal on Scientific Computing, 15 (1994), pp. 285–294.

[2] C. H. Bischof, A. Bouaricha, P. M. Khademi, and J. J. Moré, *Computing gradients in large-scale optimization using automatic differentiation*, Preprint MCS–P488–0195, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., January 1995.

[3] C. H. Bischof, A. Carle, G. F. Corliss, and A. Griewank, *ADIFOR: Automatic differentiation in a source translation environment*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, P. S. Wang, ed., New York, 1992, ACM Press, pp. 294–302.

[4] C. Broyden, *A class of methods for solving nonlinear simultaneous equations*, Mathematics of Computation, 19 (1965), pp. 577–593.

[5] T. F. Coleman and J. Y. Cai, *The cyclic coloring problem and estimation of sparse Hessian matrices*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 221–235.

[6] T. F. Coleman and A. Verma, *The efficient computation of sparse Jacobian matrices using automatic differentiation*, Tech. Report TR95–1557, Computer Science Department, Cornell University, November 1995.

[7] ——, *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., 1996, pp. 149–159.

[8] 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 (1996), pp. 131–167.

[9] A. K. M. Hossain and T. Steihaug, *Computing a sparse Jacobian matrix by rows and columns*, Tech. Report 109, Department of Informatics, University of Bergen, Bergen, June 1995.

[10] MATLAB 4.2c for UNIX, The Mathworks, Inc., 24 Prime Park Way, Natick, Massachusetts 01760.