

ARGONNE NATIONAL LABORATORY  
9700 South Cass Avenue  
Argonne, Illinois 60439

**AUTOMATIC DIFFERENTIATION TOOLS IN OPTIMIZATION  
SOFTWARE**

**Jorge J. Moré**

Mathematics and Computer Science Division

Preprint ANL/MCS-P859-1100

November 2000

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the the National Science Foundation (Information Technology Research) grant CCR-0082807.



# Automatic Differentiation Tools in Optimization Software

Jorge J. Moré

## Abstract

We discuss the role of automatic differentiation tools in optimization software. We emphasize issues that are important to large-scale optimization and that have proved useful in the installation of nonlinear solvers in the NEOS Server. Our discussion centers on the computation of the gradient and Hessian matrix for partially separable functions and shows that the gradient and Hessian matrix can be computed with guaranteed bounds in time and memory requirements.

## 1 Introduction

Despite advances in automatic differentiation algorithms and software, researchers disagree on the value of incorporating automatic differentiation tools in optimization software. There are various reasons for this state of affairs. An important reason seems to be that little published experience exists on the effect of automatic differentiation tools on realistic problems, and thus users worry that automatic differentiations tools are not applicable to their problems or are too expensive in terms of time or memory. Whatever the reasons, few optimization codes incorporate automatic differentiation tools.

Without question, incorporating automatic differentiation tools into optimization is not only useful but, in many cases, essential in order to promote the widespread use of state-of-the-art optimization software. For example, a Newton method for the solution of large bound-constrained problems

$$\min \{f(x) : x_l \leq x \leq x_u\},$$

where  $f : \mathbb{R}^n \mapsto \mathbb{R}$  and  $x_l$  and  $x_u$  define the bounds on the variables, requires that the user provide procedures for evaluating the function  $f(x)$  and also the gradient  $\nabla f(x)$ , the sparsity pattern of the Hessian matrix  $\nabla^2 f(x)$ , and the Hessian matrix  $\nabla^2 f(x)$ . The demands on the user increase for the constrained optimization problem

$$\min \{f(x) : x_l \leq x \leq x_u, c_l \leq c(x) \leq c_u\},$$

where  $c : \mathbb{R}^n \mapsto \mathbb{R}^m$  are the nonlinear constraints. In this case the user must also provide the sparsity pattern and the Jacobian matrix  $c'(x)$  of the constraints. In some cases the user may even be asked to provide the Hessian matrix of the Lagrangian

$$L(x, u) = f(x) + \langle u, c(x) \rangle \tag{1.1}$$

---

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439 ([more@mcs.anl.gov](mailto:more@mcs.anl.gov)). This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the the National Science Foundation (Information Technology Research) grant CCR-0082807.

of the optimization problem. The time and effort required to obtain this information and verify their correctness can be large even for simple problems. Clearly, any help in simplifying this effort would promote the use of the software.

In spite of the advantages offered by automatic differentiation tools, relatively little effort has been made to interface optimization software with automatic differentiation tools. Dixon [16, 15] was an early proponent of the integration of automatic differentiation with optimization, but to our knowledge Liu and Tits [24] were the first to provide interfaces between a general nonlinear constrained optimization solver (FSQP) and automatic differentiation tools (ADIFOR).

Modeling languages for optimization (for example, AMPL [3] and GAMS [18]) provide environments for solving optimization problems that deserve emulation. These environments package the ability to calculate derivatives, together with state-of-the-art optimization solvers and a language that facilitates modeling, to yield an extremely attractive problem-solving environment.

The NEOS Server for Optimization [25] is another problem-solving environment that integrates automatic differentiation tools and state-of-the-art optimization solvers. Users choose a solver and submit problems via the Web, email ([neos@mcs.anl.gov](mailto:neos@mcs.anl.gov)), or a Java-enabled submission tool. When a submission arrives, NEOS parses the submission data and relays that data to a computer associated with the solver. Once results are obtained, they are sent to NEOS, which returns the results to the user. Submissions specified in Fortran are processed by ADIFOR [6, 7], while C submissions are handled by ADOL-C [21]. Since the initial release in 1995, the NEOS Server has continued to add nonlinear optimization solvers with an emphasis on large-scale problems, and the current version contains more than a dozen different nonlinear optimization solvers.

Users of a typical computing environment would like to solve optimization problems while only requiring that the user provide a specification of the problem; all other quantities required by the software (for example, gradients, Hessians, and sparsity patterns) would be generated automatically. Optimization modeling languages and the NEOS Server provide this ability, but as noted above, users of nonlinear optimization solvers are usually asked to provide derivative information.

Our goal in this paper is to discuss techniques for using automatic differentiation tools in large-scale optimization software. We highlight issues that are relevant to solvers in the NEOS Server. For recent work on the interface between automatic differentiation tools and large-scale solvers, see [1, 23]. We pay particular attention to the computation of second-order (Hessian) information since there is evidence that the use of second-order information is crucial to the solution of large-scale problems. The main concern is the cost of obtaining second-order information. See [2, 19, 20] for related work.

We note that at present most optimization software for large-scale problems use only first-order derivatives. Indeed, of the nonlinear solvers available in the NEOS Server, only LANCELOT, LOQO, and TRON accept second-order information. We expect this situation to change, however, as automatic differentiation tools improve and provide second-order information with the same reliability and efficiency as are currently available for first-order information.

## 2 Partially Separable Functions

We consider the computation of the gradient and Hessian matrix of a partially separable function, that is, a function  $f : \mathbb{R}^n \mapsto \mathbb{R}$  of the form

$$f(x) = \sum_{k=1}^m f_k(x), \quad (2.1)$$

where the component functions  $f_k : \mathbb{R}^n \mapsto \mathbb{R}$  are such that the *extended* function

$$f_E(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}$$

has a sparse Jacobian matrix. Our techniques are geared to the solution of large-scale optimization problems. For an extensive treatment of techniques for computing derivatives of general and partially separable functions with automatic differentiation tools, we recommend the recent book by Griewank [20].

Partially separable functions were introduced by Griewank and Toint [22]. They showed, in particular, that  $f : \mathbb{R}^n \mapsto \mathbb{R}$  is partially separable if and only if the Hessian matrix  $\nabla^2 f(x)$  is sparse. Partially separable functions also arise in systems of nonlinear equations and nonlinear least squares problems. For example, if each component of the mapping  $r : \mathbb{R}^n \mapsto \mathbb{R}^m$  is partially separable, then

$$f(x) = \frac{1}{2} \|r(x)\|^2$$

is also partially separable. As another example, consider the constrained optimization problem

$$\min \{f(x) : x_l \leq x \leq x_u, c_l \leq c(x) \leq c_u\},$$

where  $c : \mathbb{R}^n \mapsto \mathbb{R}^m$  specifies the constraints. For this problem, the Lagrangian function  $L(\cdot, u)$  defined by (1.1) is partially separable if  $f$  and all the components of the mapping  $c$  are partially separable. For specific examples note that the functions  $f$  and  $c$  in the parameter estimation and optimal control optimization problems in the COPS [17] collection are partially separable.

We are interested in computing the gradient and the Hessian of a partially separable function with guaranteed bounds in terms of both computing time and memory requirements. We require that the computing time be bounded by a multiple of the computing time of the function, that is,

$$T\{\nabla f(x)\} \leq \Omega_{T,G} T\{f(x)\}, \quad T\{\nabla^2 f(x)\} \leq \Omega_{T,H} T\{f(x)\}, \quad (2.2)$$

for constants  $\Omega_{T,G}$  and  $\Omega_{T,H}$ , where  $T\{\cdot\}$  is computing time. We also require that

$$M\{\nabla f(x)\} \leq \Omega_{M,G} M\{f(x)\}, \quad M\{\nabla^2 f(x)\} \leq \Omega_{M,H} M\{f(x)\} \quad (2.3)$$

for constants  $\Omega_{M,G}$  and  $\Omega_{M,H}$ , where  $M\{\cdot\}$  is memory.

These are important requirements for large-scale problems. In particular, if the constants in these expressions are small and independent of the structure of the extended function  $f_E$ , then the computational requirements of an iteration of Newton's method are comparable with those of a limited-memory Newton's method.

The constants in (2.2) and (2.3) can be bounded in terms of a measure of the sparsity of the extended function. We use  $\rho_M$ , where

$$\rho_M \equiv \max\{\rho_i\},$$

and  $\rho_i$  is the number of nonzeros in the  $i$ th row of  $f_E'(x)$ . We can also view  $\rho_M$  as the largest number of variables in any of the component functions.

Decompositions (2.1) with the number  $m$  of element functions of order  $n$ , and with  $\rho_M$  small and independent of  $n$ , are preferred. Since the number of nonzeros in the Hessian  $\nabla^2 f(x)$  is no more than  $m\rho_M$ , decompositions with these properties are guaranteed to have sparse Hessian matrices. Discretizations of parameter estimation and optimal control problems, for example, have these properties because in these problems each element function represents the contributions from an interval or an element in the discretization.

One of the aims of this paper is to present numerical evidence that we can compute the gradient  $\nabla f(x)$  and the Hessian matrix  $\nabla^2 f(x)$  of a partially separable function with

$$\Omega_{T,G} \leq \kappa_1 \rho_M, \quad \Omega_{T,H} \leq \kappa_2 \rho_M^2, \quad (2.4)$$

where  $\kappa_1$  and  $\kappa_2$  are constants of modest size and independent of  $f_E$ . We normalize  $\Omega_{T,G}$  by  $\rho_M$  because the techniques in Section 3 require at least  $\rho_M$  functions evaluations to estimate the gradient. Similarly, the number of gradient evaluations needed to estimate the Hessian matrix by the techniques in Section 4 is at least  $\rho_M$ . Thus, these techniques require at least  $\rho_M^2$  function evaluations to estimate the Hessian matrix.

### 3 Computing Gradients

We now outline the techniques that we use for computing the gradients of partially separable functions. For additional information on the techniques in this section, see [5, 8].

Computing the gradient of a partially separable function so that the bounds (2.2) and (2.3) are satisfied is based on the observation, due to Andreas Griewank, that if  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is partially separable, then

$$f(x) = f_E(x)^T e,$$

where  $e \in \mathbb{R}^m$  is the vector of all ones, and hence

$$\nabla f(x) = f_E'(x)^T e. \quad (3.1)$$

We can then compute the gradient by computing the Jacobian matrix  $f_E'(x)$ .

At first sight the approach based on (3.1) does not look promising, since we need to compute a Jacobian matrix and then obtain the gradient from a matrix-vector product. However, the key observation is that the Jacobian matrix is sparse, while the gradient is dense. Thus, we can use sparse techniques for the computation of the extended Jacobian.

We could also use the reverse approach of automatic differentiation to compute the gradient of  $f$ . The reverse approach works directly on  $f$  and does not require the partial separability structure of  $f$ . Moreover, for the reverse approach, (2.2) holds with  $\Omega_{T,G}$  small and independent of  $\rho_M$ . Theoretically  $\Omega_{T,G} \leq 5$ , but practical implementations may not satisfy this bound. However, the memory requirements of the reverse approach depend on the number of floating point operations needed to compute  $f$ , and thus (2.3) can be violated. A careful comparison between the reverse approach and the techniques described below would be of interest.

In this section we consider two methods for computing the gradient of a partially separable function via (3.1). In the *compressed* AD approach, automatic differentiation tools are used to compute a compressed form of the Jacobian matrix of the extended function  $f_E$ , while in the *sparse* AD approach, automatic differentiation tools are used to compute a sparse representation of the Jacobian matrix of the extended function.

In the compressed AD approach we assume that the sparsity pattern of the Jacobian matrix  $f_E'(x)$  is known. Given the sparsity pattern, we partition the columns of the Jacobian matrix into groups of *structurally orthogonal* columns, that is, columns that do not have a nonzero in the same row position. Given a partitioning of the columns into  $p$  groups of structurally orthogonal columns, we determine the Jacobian matrix by computing the *compressed Jacobian* matrix  $f_E'(x)V$ , where  $V \in \mathbb{R}^{n \times p}$ . There is a column of  $V$  for each group, and  $v_{i,j} \neq 0$  only if the  $i$ th column of  $f_E'(x)$  is in the  $j$ th group. Software for this partitioning problem [11] defines the groups with an array `ngrp` that sets the group for each column.

The extended Jacobian can be determined from the compressed Jacobian matrix  $f_E'(x)V$  by noting that if column  $j$  is in group  $k$ , then

$$\langle e_i, f_E'(x)V e_k \rangle = v_{i,j} \partial_{i,j} f_E(x).$$

Thus  $\partial_{i,j} f_E(x)$  can be recovered directly from the compressed Jacobian matrix.

We note that for many sparsity patterns, the number of groups  $p$  needed to determine  $A \in \mathbb{R}^{m \times n}$  with a partitioning of the columns is small and independent of  $n$ . In all cases there is a lower bound of  $p \geq \rho_M$ . We also know [13] that if a matrix  $A$  can be permuted to a matrix with bandwidth  $band(A)$ , then  $p \leq band(A)$ .

The sparse AD approach uses a sparse data representation, usually in conjunction with dynamic memory allocation, to carry out all intermediate derivative computations. At present, the SparsLinC library in ADIFOR [7] is the only automatic differentiation tool with this capability. The main advantage of the sparse AD approach over the compressed AD approach is that no knowledge of the sparsity pattern is required. On the other hand, the sparse AD approach is almost always slower, and can be significantly slower on vector machines.

In an optimization setting, a hybrid approach [9] is the best approach. With this strategy, the sparse AD approach is used to obtain the sparsity pattern of the Jacobian matrix of the extended function at the starting point. See Section 4 for additional information on techniques for computing the sparsity pattern of the extended function. Once the sparsity pattern is determined, the compressed AD approach is used on all other iterations. The

hybrid approach is currently the best approach to compute gradients of partially separable functions, and is used in all solvers installed on the NEOS Server.

We conclude this section with some recent results on using the sparse AD approach to compute the gradients of partially separable functions drawn from the MINPACK-2 [4] collection of test problems. We selected ten problems; the first five problems are finite element formulations of variational problems, while the last five problems are systems of nonlinear equations derived from collocation or difference formulations of systems of differential equations.

Table 3.1 provides the value of  $\rho_M$  for the ten problems in our performance results. For each of the problems we used three values of  $n$ , usually  $n \in \{1/4, 1, 4\} \cdot 10^4$ , to observe the trend in performance as the number of variables increases. The results were essentially independent of the number of variables, so our results are indicative of the performance that can be expected in large-scale problems.

Table 3.1: Data for MINPACK-2 test problems

	PJB	MSA	ODC	SSC	GL2	FIC	SFD	IER	SFI	FDC
$\rho_M$	5	4	4	4	5	9	14	17	5	13

We want to show that the bounds (2.4) for  $\Omega_{T,G}$  holds for these problems. For these results we used the sparse approach to compute the Jacobian matrix  $f_E'(x)$  of the extended function, and then computed the gradient of  $f$  with (3.1). For each problem we computed the ratio  $\kappa_1$ , where

$$T\{\nabla f(x)\} = \kappa_1 \rho_M \max T\{f(x)\}.$$

Table 3.2 presents the quartiles for  $\kappa_1$  obtained on a Pentium 3 (500 MHz clock, 128 MB of memory) with the Linux operating system.

Table 3.2: Quartiles for  $\kappa_1$  on the MINPACK-2 problems

min	$q_1$	$q_2$	$q_3$	max
1.3	2.8	4.5	5.3	7.8

The results in Table 3.2 show that the bound (2.4) for  $\Omega_{T,G}$  holds for the MINPACK-2 problems, with  $\kappa_1$  small.

These results are consistent with the results in [5], where it was shown that  $\kappa_1 \in [3, 15]$  on a SPARC-10 for another set of test problems drawn from the MINPACK-2 collection. Note that in [5] the ratio  $\kappa_1$  was computed with  $\rho_M$  replaced by the number of columns  $p$  in the matrix  $V$ . Since  $p \geq \rho_M$ , the ratios in Table 3.2 would decrease if we replaced  $\rho_M$  by  $p$ . The advantage of using  $\rho_M$  is that the ratio  $\kappa_1$  is then dependent only on the structure of the function.

## 4 Computing Hessian Matrices

We have already shown how automatic differentiation tools can be used to compute the gradient of a partially separable function. We now discuss the tools that are needed to compute the Hessian of a partially separable so that the requirements (2.2) on computing time and (2.3) on memory are satisfied.

The techniques that we propose require the sparsity pattern of the Hessian matrix and that the Hessian-vector products  $\nabla^2 f(x)v$  be available. In our numerical results we approximate the Hessian-vector product with a difference of gradient values, but in future work we expect to compute Hessian-vector products with ADIFOR.

We now show how to compute the sparsity pattern of the Hessian matrix from the sparsity pattern of  $f_E'(x)$ . We define the sparsity pattern of a matrix-valued mapping  $A : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$  in a neighborhood  $N(x_0)$  of a point  $x_0$  by

$$\mathcal{S}\{A(x_0)\} \equiv \left\{ (i, j) : a_{i,j}(x) \neq 0, x \in N(x_0) \right\}. \quad (4.1)$$

We are interested in the sparsity pattern of the extended Jacobian and the Hessian matrix of a partially separable function  $f : \mathbb{R}^n \mapsto \mathbb{R}$  in a region  $\mathcal{D}$  of the form

$$\mathcal{D} = \{x \in \mathbb{R}^n : x_l \leq x \leq x_u\}.$$

Given  $x \in \mathcal{D}$ , we evaluate the sparsity pattern  $\mathcal{S}\{f_E'(x)\}$  by computing  $f_E'(\bar{x}_0)$ , where  $\bar{x}_0$  is a random, small perturbation of  $x_0$ , for example,

$$\bar{x}_0 = (1 + \varepsilon)x_0 + \varepsilon, \quad |\varepsilon| \in [10^{-6}, 10^{-4}].$$

Then we can reliably let  $\mathcal{S}\{f_E'(x)\}$  be the set of  $(i, j)$  such that  $\partial_{i,j} f_E(\bar{x}_0) \neq 0$ . We should not obtain the sparsity pattern of the Jacobian matrix by evaluating  $f_E'$  at the starting point  $x_0$  of the optimization process because this point is invariably special, and thus the sparsity pattern of the Jacobian matrix is unlikely to be representative.

The technique that we have outlined for determining the sparsity pattern is used by the solvers in the NEOS Server and has proved to be quite reliable. The sign of  $\varepsilon$  must be chosen so that  $\bar{x}_0 \in \mathcal{D}$ , and special care must be taken to handle the case when  $x_l$  and  $x_u$  agree in some component.

Given the sparsity pattern of the Jacobian matrix of the extended function, we determine the sparsity pattern for the Hessian  $\nabla^2 f(x)$  of the partially separable function  $f$  via

$$\mathcal{S}\{\nabla^2 f(x)\} \subset \mathcal{S}\{f_E'(x)^T f_E'(x)\}. \quad (4.2)$$

Note that (4.2) is valid only in terms of the definition (4.1) for a sparsity pattern. For example, if  $f : \mathbb{R}^2 \mapsto \mathbb{R}$  is defined by

$$f(x) = \phi(\xi_1 \xi_2)$$

for a function  $\phi$  such that  $\phi'(0) \neq 0$ , then  $\partial_{1,2} f(0) \neq 0$ , but  $\partial_1 f(0) = \partial_2 f(0) = 0$ . However, (4.2) holds because  $\partial_2 f(x) \neq 0$  and  $\partial_1 f(x) \neq 0$  in a neighborhood of the origin.

In most cases equality holds in (4.2). This happens, in particular, if  $f$  does not depend linearly on the variables, and

$$\bigcup_{k=1}^m \mathcal{S} \{ \nabla^2 f_k(x) \} \subset \mathcal{S} \{ \nabla^2 f(x) \}. \quad (4.3)$$

If  $f$  depends linearly on some variables, say,

$$f(x) = \xi_1 + \phi(\xi_2, \dots, \xi_n),$$

then equality does not hold in (4.2). Assumption (4.3) implies that there is no cancellation in the computation of the Hessian  $\nabla^2 f(x)$ . This assumption can fail in some cases, for example, when  $f_1 \equiv -f_2$ , but holds in most cases.

Since we are able to estimate the sparsity pattern of the Hessian matrix via (4.2), we could use the compressed AD approach described in Section 3 to compute the Hessian matrix from a compressed Hessian  $\nabla^2 f(x)V$ . However, these techniques ignore the symmetry of the Hessian matrix and thus may require an unnecessarily large number of columns  $p$  in the matrix  $V$ . For example, an arrowhead matrix requires  $p = n$  if symmetry is ignored, but  $p = 2$  otherwise.

Powell and Toint [26] were the first to show that symmetry can be used to reduce the number  $p$  of columns in the matrix  $V$ . They proposed two methods for determining a symmetric matrix  $A$  from a compressed matrix  $AV$ . In the direct method the unknowns in  $A$  are determined directly from the elements in the compressed matrix  $AV$ . In this method unknowns are determined independently of each other. In the substitution method the unknowns are determined in a given order, either directly or as a linear combination of elements that have been previously determined.

These definitions of direct and substitution methods are precise but do not readily yield algorithms for determining symmetric matrices. Coleman and Moré [14] and Coleman and Cai [10] extended [26] by interpreting the problem of determining symmetric matrices in terms of special graph coloring problems. This work led to new algorithms and a deeper understanding of the estimation problem.

Software for the symmetric graph coloring problem is available [12] for both direct and substitution methods. Numerical results in [14] suggest that a direct method yields a 20% improvement over methods that disregard symmetry, and that the substitution method yields about a 30% reduction over the direct method.

We use Algorithm 4.1 to compute the Hessian matrix from a user-supplied extended function  $f_E$ . This algorithm uses static memory allocation so that it is first necessary to determine the number of nonzeros in  $f_E'(x_0)$  by computing  $f_E'(x_0)$  by rows, but not storing the entries. Once this is done, we allocate space for  $f_E'(x_0)$  and compute  $f_E'(x_0)$  and the sparsity pattern. Another interesting aspect of Algorithm 4.1 is that we compute the number of nonzeros in  $f_E'(x_0)^T f_E'(x_0)$  directly from the sparsity pattern of  $f_E'(x_0)$ . In view of (4.2), we then have an accurate idea of the amount of memory needed to store the Hessian matrix. The final step is to compute the Hessian matrix from the the compressed Hessian matrix  $\nabla^2 f(x_0)V$  by either a direct or a substitution method.

- ◊ Evaluate  $f_E(x_0)$  and obtain  $m = \text{size } f_E(x_0)$ .
- ◊ Compute  $\text{nnz}\{f_E'(x_0)\}$ .
- ◊ Allocate space for  $f_E'(x_0)$ .
- ◊ Compute the sparsity pattern  $\mathcal{S}\{f_E'(x_0)\}$ .
- ◊ Compute  $\text{nnz}\{f_E'(x_0)^T f_E'(x_0)\}$ .
- ◊ Allocate space for  $\nabla^2 f(x_0)$ .
- ◊ Compute  $\nabla^2 f(x_0)$  from the compressed Hessian matrix  $\nabla^2 f(x_0)V$ .

Algorithm 4.1: Computing the Hessian matrix for a partially separable function.

We consider both direct and substitution methods to determine the Hessian matrix from the compressed Hessian. In both cases we are interested in the ratio  $\kappa_2$ , where

$$T\{\nabla^2 f(x)\} = \kappa_2 \rho_M^2 T\{f(x)\},$$

since this provides a measure of the cost of evaluating the Hessian matrix relative to the cost of the function. The  $\kappa_2$  quartiles for both direct and substitution methods on the MINPACK-2 problems used in Section 3 appear in Table 4.1.

Table 4.1: Quartiles for  $\kappa_2$  on MINPACK-2 problems

Method	min	$q_1$	$q_2$	$q_3$	max
Direct	1.6	5.1	11.2	15.2	46.4
Substitution	1.5	4.1	9.0	12.5	30.2

Direct and substitution methods usually require more than  $\rho_M$  gradient evaluations to determine the Hessian matrix, and thus the increase in the value of  $\kappa_2$  relative to  $\kappa_1$  in Table 3.2 was expected. Still, it is reassuring that the median value of  $\kappa_2$  is reasonably small. The largest values of  $\kappa_2$  are due to one of the problems; if this problem is eliminated, then the maximal value drops by at least a factor of two. In general, problems with the longest computing times yield the smallest values of  $\kappa_2$  since these problems tend to mask the overhead in the automatic differentiation tools and in determining the Hessian matrix. Moreover, these results are based on using a gradient evaluation that relies on sparse automatic differentiation tools; the use of the hybrid approach mentioned in Section 3 should reduce  $\kappa_2$  substantially.

## Acknowledgments

Paul Hovland merits special mention for sharing his considerable knowledge of automatic differentiation tools. Liz Dolan used a preliminary implementation of the techniques in this paper to install TRON on the NEOS Server, and in the process sharpened these techniques. Gail Pieper provided the final touches on the paper with her careful editing.

## References

- [1] J. ABATE, S. BENSON, L. GRIGNON, P. HOVLAND, L. MCINNES, AND B. NORRIS, *Integrating automatic differentiation with object-oriented toolkits for high-performance scientific computing*, in Automatic Differentiation: From Simulation to Optimization, 2000.
- [2] J. ABATE, C. BISCHOF, A. CARLE, AND L. ROH, *Algorithms and design for a second-order automatic differentiation module*, in International Symposium on Symbolic and Algebraic Computing (ISSAC), SIAM, 1997, pp. 149–155.
- [3] *AMPL*. See <http://www.ampl.com/>.
- [4] B. M. AVERICK AND J. J. MORÉ, *User guide for the MINPACK-2 test problem collection*, Technical Memorandum ANL/MCS-TM-157, Argonne National Laboratory, Argonne, Illinois, 1991. Also issued as Preprint 91-101 of the Army High Performance Computing Research Center at the University of Minnesota.
- [5] C. BISCHOF, A. BOUARICHA, P. KHADEMI, AND J. J. MORÉ, *Computing gradients in large-scale optimization using automatic differentiation*, INFORMS J. Computing, 9 (1997), pp. 185–194.
- [6] C. BISCHOF, A. CARLE, P. KHADEMI, AND A. MAUER, *ADIFOR 2.0: Automatic differentiation of Fortran 77 programs*, IEEE Computational Science & Engineering, 3 (1996), pp. 18–32.
- [7] C. BISCHOF, A. CARLE, P. KHADEMI, A. MAUER, AND P. HOVLAND, *ADIFOR 2.0 user's guide (Revision C)*, Technical Report ANL/MCS-TM-192, Argonne National Laboratory, Argonne, Illinois, 1995.
- [8] C. BISCHOF, P. KHADEMI, A. BOUARICHA, AND A. CARLE, *Efficient computation of gradients and Jacobians by dynamic exploitation of sparsity in automatic differentiation*, Optim. Methods Software, 7 (1996), pp. 1–39.
- [9] A. BOUARICHA AND J. J. MORÉ, *Impact of partial separability on large-scale optimization*, Comp. Optim. Appl., 7 (1997), pp. 27–40.
- [10] T. F. COLEMAN AND J.-Y. CAI, *The cyclic coloring problem and estimation of sparse Hessian amtrices*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 221–235.
- [11] T. F. COLEMAN, B. S. GARBOW, AND J. J. MORÉ, *Software for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329–345.
- [12] ———, *Software for estimating sparse Hessian matrices*, ACM Trans. Math. Software, 11 (1985), pp. 363–377.
- [13] T. F. COLEMAN AND J. J. MORÉ, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

- [14] ———, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, 28 (1984), pp. 243–270.
- [15] L. C. W. DIXON, *On the impact of automatic differentiation on the relative performance of parallel truncated Newton and variable metric algorithms*, SIAM J. Optim., 1 (1991), pp. 475–486.
- [16] L. C. W. DIXON, *Use of automatic differentiation for calculating Hessians and Newton steps*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 114–125.
- [17] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with COPS*, Technical Memorandum ANL/MCS-TM-246, Argonne National Laboratory, Argonne, Illinois, 2000.
- [18] *GAMS*. See <http://www.gams.com/>.
- [19] R. GIERING AND T. KAMINSKI, *On the performance of derivative code generated by TAMC*, manuscript, FastOpt, Hamburg, Germany, 2000.
- [20] A. GRIEWANK, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2000.
- [21] 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.
- [22] A. GRIEWANK AND P. L. TOINT, *On the unconstrained optimization of partially separable functions*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, 1982.
- [23] P. D. HOVLAND, D. E. KEYES, L. C. MCINNES, AND W. SAMYONO, *Using automatic differentiation for second-order methods in PDE-constrained optimization*, in Automatic Differentiation: From Simulation to Optimization, 2000.
- [24] M. D. LIU AND A. L. TITS, *User's guide for ADIFFSQP Version 0.9: A utility program that allows the user of the FFSQP constrained nonlinear optimization routines to conveniently invoke the computational differentiation preprocessor ADIFOR 2.0*, technical report, University of Maryland, Systems Research Center, College Park, MD, USA, 1997.
- [25] *NEOS Server for Optimization Problems*. See <http://neos.mcs.anl.gov/>.
- [26] M. J. D. POWELL AND P. L. TOINT, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060–1074.