Complexity in Numerical Optimization
P.M. Pardalos, Editor

Some Bounds on the Complexity of Gradients, Jacobians, and Hessians *

Andreas Griewank
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 USA

Abstract

The evaluation or approximation of derivatives is an important part of many nonlinear computations. The cost of evaluating first- and second-derivative matrices is often assumed to grow linearly and quadratically with the number of independent variables, respectively. It is shown here that much tighter bounds can be achieved through the exploitation of partial function- and argument-separability in combination with the forward and reverse mode of computational, or automatic, differentiation.

The new separability concepts facilitate the reduction of chromatic numbers and maximal row lengths, which determine the complexity of the Curtis-Powell-Reid and Newsam-Ramsdell schemes for estimating sparse derivative matrices. Because of the duality between the forward and reverse modes these techniques can be applied to Jacobians as well as their transposes and the associated row-intersection graphs. In contrast to differencing, computational differentiation yields derivative values free of truncation errors and without any parameter dependence.

A key result presented in this paper is that gradients and Hessians of partially separable functions can also be obtained surprisingly cheaply in the easily implemented forward mode as well as in the more sophisticated reverse mode of computational differentiation.

*This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1 Introduction

Most algorithms for the numerical solution of nonlinear programming problems involve the gradient of the objective function, the Jacobian of the active constraints, and the Hessian of the Lagrangian function. Some of this derivative information may be used only implicitly or in projected form. However, since optimizers are locally characterized by the KKT conditions in terms of objective and constrained gradients, evaluation errors in these first derivatives directly limit the solution accuracy. Moreover, except for algebraically simple test functions and some large, sparse problems, the computational effort of evaluating or approximating the required derivative information may dominate the cost of the numerical linear algebra and other algorithmic overhead.

As in numerical ordinary differential equations and other nonlinear computational fields, designers of optimization methods have usually assumed that derivatives are hard to come by and that their provision belongs to the realm of the user. This is sometimes called the black box model [21], where the optimization algorithm relies exclusively on a subroutine for evaluating objectives and gradients at given arguments, usually with unspecified precision. Unless the user provides additional codes for gradient and Jacobian evaluation, the optimization method must either resort to divided difference approximations or model the optimization problem on the basis of function values alone. The latter approach is rarely advantageous since it typically results in slow and unreliable convergence.

1.1 The Computational Model

In this paper we develop complexity estimates for gradients, Jacobians, and Hessians of functions that are defined by computer programs as compositions of arithmetic operations and intrinsic functions. We will assume that these elementary functions are performed in floating-point arithmetic with fixed precision and that their computational cost is independent of the argument. This scenario applies to the majority of practical optimization calculations, but not to symbolic, interval, or variable-precision computations. The overall temporal complexity is measured by simply counting the number of elementary operations and the number of random or sequential memory accesses. Here, it is assumed that the memory hierarchy can be split into a randomly accessed "core" storage and a sequentially accessed "disk" storage. Because of the increasing lag between processor and memory speed, the distinction between various memory access patterns becomes increasingly important not only on supercomputers but also on modern workstations.

The process of calculating overall derivatives vectors and matrices from the "local" derivatives of the elementary functions has become known as *automatic differentiation* [19],[13] but might be better called *computational differentiation*. Depending on the order in which the chain rule is applied, one obtains the forward, reverse, or

mixed mode of computational differentiation. As was shown in [12], even a sophisticated implementation of the reverse mode of computational differentiation involves a logarithmic increase in the storage requirement, but the extra data can be stored and accessed sequentially in successive forward and reverse sweeps. In contrast, this nearly perfect data locality is lost if one tries to minimize the operations count for Jacobian evaluations by the general vertex elimination scheme described in [14]. Moreover, the combinatorial task of finding an elimination ordering that absolutely minimizes the operations count is conjectured to be NP hard. Therefore, we confine ourselves in this paper to sequential schemes for which the randomly accessed core memory can be a priori restricted to a small multiple of that used by the original function evaluation.

Several other practical aspects of relative efficiency are very hard to quantify, even in an informal discussion of computational complexity. In our context this applies, for example, to the alternative of compilable derivative code versus more interpretive derivative-evaluation schemes. A related issue is the relative efficiency of dynamically sparse vector operations with indirect addressing versus dense vector operations on contiguous arrays. While it is relatively easy to generate compilable code for the forward mode of automatic differentiation (see, e.g., [2]), all current implementations of the reverse mode incur a large number of procedure calls or other interpretive overheads, unless the original evaluation source is very restricted. Therefore, we will emphasize ways of reducing the temporal complexity of the forward mode by exploiting partial separability and other structure, even when the basic reverse mode has a lower operations count, as is the case for gradients.

Fortunately, much of the excellent research that has been conducted regarding the estimation of sparse Jacobians and Hessians by differencing (see, e.g., [10], [9], [18], [20]) carries over to computational differentiation. The main difference is that, instead of approximating Jacobian vector products by divided differences, one obtains them without any truncation errors by the forward mode of automatic differentiation. Moreover, since the reverse mode yields vector Jacobian products accurately and efficiently, one can exploit the sparsity structure not only columnwise but also rowwise. Therefore the well-known coloring techniques can be applied either to the column- or the row-intersection graph. Hence our complexity bounds involve the maximal clique size and the chromatic number of these undirected graphs. We have delayed until the central section of this paper the introduction of the directed *computational graph* that is associated with function evaluation programs. Until then, the results are developed in a more familiar matrix-vector notation. Concepts such as operations counts and intermediate variable will be used in an intuitive way until they are defined rigorously in Section 3.

The paper is organized as follows. In the remainder of the introduction we review the derivative requirements in optimization calculations and the basic properties of the forward and reverse modes of computational differentiation in combination with the CPR (Curtis-Powell-Reid) and NR (Newsam-Ramsdell) approaches to estimating sparse derivative matrices. In Subsections 2.1 and 2.2 we discuss whether and how

partial function-separability and a dual concept of *partial argument-separability* can be used to express the original function as contraction of larger systems, whose Jacobians or their transposes have shorter row lengths and lower chromatic numbers. In Subsection 2.3 we discuss the tearing of functions and arguments, which amounts to a forced function or argument separation that entails a certain duplication of intermediate calculations. The resulting complexities are closely related to those of dynamically sparse implementations of the forward and reverse modes. In Subsection 2.4 we discuss the relation to multicoloring and in Subsection 2.5 we examine a generic binary interaction example. In Section 3 we provide a rigorous foundation and generalization of all previous developments in terms of computational graphs. In Section 4 we compile and discuss complexity bounds for the various methods and decompositions. The paper concludes with a brief summary in Section 5.

1.2 Derivative Requirements in Nonlinear Optimization

As a focal point of our investigation we consider the constrained optimization problem

$$\min w^T x \quad \text{s.t.} \quad f(x) = 0 \quad \text{and} \quad \underline{x} \leq x \leq \bar{x}$$

with $w \in \mathbb{R}^n$ some fixed-price vector and $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ at least twice continuously differentiable. Without loss of generality we have assumed that the objective and the inequality constraints are linear; hence, all nontrivial derivatives occur in the equality constraints. As a consequence of this normalization, some components of x are likely to be slack variables; thus, the corresponding derivatives of f have a special structure as well. Sparsity in the Jacobian $J(x) \equiv J$ and the second-derivative tensor f'' will be a major concern throughout this paper. However, we will not make use of the observation that one need not evaluate rows of J that correspond to "safely" inactive constraints because their slack variable value is very large. Similarly, one could theoretically skip columns of J corresponding to "nonbasic" variables x_i that are certain to remain fixed at their lower or upper bound $\underline{x}_i$ or $\bar{x}_i$ over the current iteration.

Typically, one step of an iterative optimization algorithm proceeds as follows. First, the scalar objective $w^T x$ and the constraint vector $f(x)$ are evaluated at a new trial point x, which may be assumed feasible with respect to the bound constraints. Occasionally, the combination of values $(w^T x, f(x))$ may be deemed unacceptable, and the algorithm backtracks to a previous iterate. Otherwise, one evaluates the Jacobian $J(x)$ and computes a matrix S, whose $n - m$ columns span the null-space of $J(x)$. This is usually done by factoring $J(x)$, for example, into the product of a lower triangular or orthogonal

problem involves (besides the objective gradient w) the constraint Jacobian J and some information about the Hessian of the Lagrangian

$$L(x, u) \equiv w^T x + \sum_{i=1}^{m} u_i f_i(x).$$

Here the Lagrange multipliers u_i are usually estimates from the previous iteration. For one-step quadratic convergence a Newton-like algorithm must know the one-sided projection

$$H \equiv \nabla_x^2 L(x,u)\,S \quad \text{with} \quad J\,S = 0 \in \mathbb{R}^{m\times(n-m)} \quad .$$

For two-step quadratic convergence it suffices to evaluate the two-sided projection $S^T H \in \mathbb{R}^{p\times p}$ with $p = n - m$; but as we will show, this is not necessarily a saving with regard to computational differentiation. Moreover, it is important to recognize that the one-side projection onto the columns of S can be built into the differentiation process so that the full Hessian need never be formed. This implicitly projected approach is likely to be efficient on problems with a comparatively small degree of freedom, p.

1.3 Basic Properties of the Forward and Reverse Modes

Rather than considering $f(x)$ just as a mathematical mapping, we will assume from now on that it is defined by a procedural (sub)program for its evaluation at any given argument $x \in \mathbb{R}^n$. For simplicity we will exclude the possibility that the program takes different branches for various values of x or that some intrinsic function such as square root is evaluated at a point of nondifferentiability. Then one can by hand or automatically (i.e., with the help of some software package) extend the original program to a code that also evaluates the Jacobian-matrix product

$$J(x)\,S \quad \text{for} \quad S \in \mathbb{R}^{n\times p} \tag{1.1}$$

"analytically," that is, without incurring any truncation error. The choice of the *seed matrix* S provides the user with a great deal of flexibility at run time. For example, one might choose $S = I$ with $p = n$ to obtain the full Jacobian J in one sweep. Alternatively, one may have $p = 1$ over several sweeps and let $S = [s]$ range over a sequence of column vectors $s \in \mathbb{R}^n$, possibly as part of an iterative Newton-step calculation. Under realistic assumptions on the computational model, we will establish in Proposition 1 of Section 4 the bounds

$$\begin{aligned} \mathrm{OPS}\{f, JS\} &\le (2+3p)\ \mathrm{OPS}\{f\} && (1.2)\\ \mathrm{RAM}\{f, JS\} &\le (1+p)\ \mathrm{RAM}\{f\} \quad . && (1.3) \end{aligned}$$

Here, OPS denotes a conventional operations count, and RAM represents the number of randomly accessed storage locations required by the function or derivative evaluation program. On a serial machine the run time will be proportional to the operations count, and we will therefore refer to $\mathrm{OPS}\{f, \mathrm{task}\}/\mathrm{OPS}\{f\}$ as the run-time ratio for some additional computational task related to f.

For $p = 1$ the operations count and storage requirement may grow fivefold and twofold, respectively. This situation is somewhat troubling, since a divided difference of the form $[f(x + \varepsilon s) - f(x)]/\varepsilon$ yields an approximation to Js at the cost of one

extra function evaluation and without any increase in storage. Here we have assumed that f has already been evaluated at the base point x. Fortunately, the factor 3 in (1.2) is quite pessimistic. On the other hand, the leading constant 2 does not take account of some of the overhead in the derivative code. Empirically it was found [1] that, when the forward mode is implemented as compilable code and p is in the double digits, then the resulting run-time ratio is typically between half and twice the divided difference ratio $(1 + p)$.

The up to $(1 + p)$-fold increase in memory stems from the fact that a p-vector of derivatives is associated with most or all scalar variables in the original evaluation program. To limit this increase, one could alternatively evaluate one or a few derivative components at a time, but that would increase the accumulated run time significantly, since certain overhead costs are incurred repeatedly. On the other hand, when p reaches into the hundreds, it will probably be better to split the columns of S into groups in order to reduce the number of page faults in accessing intermediate-derivative vectors. Provided these p vectors are allocated as contiguous arrays, the number of memory accesses in the sense of pointer dereferencing does not really grow $(1 + p)$-fold, but merely doubles. On the other hand, if they are stored and manipulated as sparse vectors, the indirect addressing overhead goes up, but the storage requirement may be significantly reduced.

As a first approximation one may view the forward mode as truncation- and parameter-free equivalent of divided differences. The relationship is close enough that the CPR [10] and the NR [18] approaches for estimating sparse Jacobians J from some product $J\,S$ with $p < n$ can be applied virtually unchanged. In the CPR approach the rows of the seed matrix S are Cartesian p vectors, whereas in the NR approach S is usually chosen as a Vandermonde matrix. The potential ill-conditioning of the latter choice may be of less concern here since the projected Jacobian $J\,S$ is obtained essentially to working accuracy. The number p of columns required for the NR method is simply the row-length $\rho(J)$, that is, the maximal number of nonzeros in any row of the Jacobian. The corresponding lower bound for the CPR approach is the chromatic number $\chi(J)$ of the column-intersection graph of the Jacobian. This undirected graph has one node for each column of J, with two of them being connected by an edge exactly if the corresponding column pair shares a nonzero in at least one row. Otherwise the two columns are said to be structurally orthogonal. The row length and chromatic number satisfy the trivial relation

$$\rho(J) \;\leq\; \chi(J) \quad ,$$

which means that the seed matrix for the CPR method has at least as many columns as the minimum needed for the NR approach. However, to improve the conditioning of the linear systems in the latter approach, one may prefer to define S using complex roots of unity, which effectively doubles p to $2\rho(J)$.

From a mathematical point of view, the reverse mode is much more interesting since it has completely different complexity properties compared with differencing.

Instead of multiplying J from the right by a seed matrix S, whose columns represent directions in the domain of f, the Jacobian is now multiplied from the right by a matrix W^T, whose rows represent linear functionals on the range of f. Formally, the reverse mode yields

$$W^T J(x) \quad \text{for} \quad W^T \in \mathbb{R}^{q \times m} \quad .$$

At first sight the corresponding complexity estimates are also neatly transposed in that, for some integer $r \geq 1$,

$$\begin{aligned} \mathrm{OPS}\{f, W^T J\} &\leq (1 + r + 3q)\, \mathrm{OPS}\{f\} && (1.4) \\ \mathrm{RAM}\{f, W^T J\} &\leq (1 + q)\, \mathrm{RAM}\{f\} \quad . && (1.5) \end{aligned}$$

However, there is a crucial difference, namely, that the reverse mode requires the ability to run the evaluation process for f step by step backward. This requirement explains the terminology *reverse mode,* or top-down method, which is preferable to the term *backward differentiation,* a label that invites confusion with a well-established class of methods for the numerical solution of stiff differential equations. The "naive" program reversal based on a full execution trace of the forward evaluation requires temporary storage proportional to $\mathrm{OPS}\{f\}$. Fortunately, the generation and utilization of the trace data occur strictly sequentially in opposite order, so that it makes sense to quantify this storage requirement as SAM (for **S**equentially **A**ccessed **M**emory). Hence we obtain the basic bound

$$\mathrm{SAM}\{f, \overleftarrow{f}\} = \mathcal{O}(\mathrm{OPS}\{f\}) \quad , \qquad (1.6)$$

where $\overleftarrow{f}$ denotes the reversal of the evaluation process for f. Because of the strictly sequential access pattern, this potentially very large data set can be stored on external mass storage devices. When the ratio

$$h\{f\} \equiv \mathrm{OPS}\{f\} / \mathrm{RAM}\{f\} \qquad (1.7)$$

is small or of moderate size, the proportionality relation (1.6) may not be worrisome. This situation applies, for example, for any three-dimensional composite structure without long-range interaction between its components (e.g., static models of buildings or mechanical devices and discretizations of differential operators). However, especially on explicitly time-dependent problems, the ratio $h\{f\}$ may be so large that (1.6) represents an unacceptable increase in memory requirement. This severe limitation of the basic reverse mode can be remedied as follows.

Rather than generating and storing the full execution trace in one piece, one can break it into slices that are (re)generated several times from snapshots taken at judiciously selected checkpoints. A detailed analysis of this recursive reverse mode in [12] shows that there exists a constant c such that for all integers $r \geq 1$, the reversal costs can be limited according to

$$\begin{aligned} \mathrm{OPS}\{\overleftarrow{f}\} &\leq r\ \mathrm{OPS}\{f\} && (1.8) \\ \mathrm{SAM}\{\overleftarrow{f}\} &= c\ \mathrm{RAM}\{f\}\ \sqrt[r]{h\{f\}} \quad . && (1.9) \end{aligned}$$

As an alternative to accepting a constant increase in the operations count and an algebraic growth of the memory requirement with respect to h, one can also limit both increases to order $\log h$. In this paper we will use the algebraic option for the program reversal, whose operations count (1.8) was already included in (1.4). The total complexity of this reverse mode variant is therefore described by the three equations (1.4), (1.5), and (1.9). Despite these rather tight results, the practical reversal of a sizable program is a difficult problem, and no implementation achieving these bounds is currently available.

The key difference between (1.4) and the earlier (1.2) is that the operations count for the reverse mode depends on the number of dependent rather than independent variables. In particular, one obtains gradients (where $m = 1 = q$) at a small multiple of the cost of evaluating the underlying functions. Here, and later, we neglect the additional SAM requirement (1.9), which grows very slowly if r is sizable (say, greater than 5). For vector-valued functions, f, one can apply the CPR or NR approach to the rows rather than the columns of the Jacobian J, so that their respective complexities are determined by

$$\rho(J^T) \;\leq\; \chi(J^T) \quad .$$

It is sometimes mistakenly concluded that Jacobians also have essentially the same complexity as the underlying vector function since each row is a gradient of the corresponding function component. Applying (1.4) with $q = 1$ to each component separately, we have indeed

$$\mathrm{OPS}\{f, I\,J\} \;\leq\; 5 \sum_{i=1}^{m} \mathrm{OPS}\{f_i\} \quad . \tag{1.10}$$

However, the sum of the right hand side can be almost m times larger than $\mathrm{OPS}\{f\}$, as a result of the multiple usage of common intermediates in the simultaneous evaluation of all function components. For example, this will be the case if $m = n$ and f is of the form

$$f_i(x) \quad = \quad x_i * crunch(x_1, x_2, \ldots, x_n),$$

with $crunch(x)$ being a computationally very expensive scalar-valued function. On the other hand, the evaluation of any two components $f_i(x)$ and $f_j(x)$ may not involve any common intermediates, so that the right-hand side of (1.10) is indeed exactly equal to $5\,\mathrm{OPS}\{f\}$. Since this can easily happen even when J is dense, one would certainly wish to do better than applying the standard forward or reverse mode. In fact, this case would be ideal for the more general elimination procedure described in [14], whose optimal application is conjectured to be an NP-hard combinatorial problem. On the downside, even if greedy heuristics are used, the resulting Markowitz-like procedure requires RAM of order $\mathrm{OPS}\{f\}$, which appears to be a serious drawback on larger problems. Therefore, we introduce column- and row-splitting techniques that sometimes yield similar reductions in the operations count without ever violating the storage bounds (1.3) or (1.5), and (1.9).

2 Generalizations of Partial Separability

Rather than applying computational differentiation techniques to calculate the Jacobian J directly, we can first strip it of linear premultiplier and postmultipliers in order to obtain an even sparser central part that contains all nonlinearities. For the purposes of computational differentiation, the aim is to reduce the maximal number of zeros per row or column and the chromatic number of the row- or column-intersection graphs, which determine the cost of the forward and reverse mode, respectively. Especially for large-scale problems, one can expect that not only the Jacobian J but especially the derivative tensor $f'' \in \mathbb{R}^{m \times n \times n}$ is quite sparse. As observed in [15], any scalar function $h \in C(\mathbb{R}^n)$ whose Hessian is sparse can be decomposed into a sum

$$h(x) \quad = \quad \sum_k h_k(P_k x) \quad ,$$

where the projection P_k picks out the subset of components in x on which h_k depends in a nontrivial fashion. Whenever a Hessian entry $\partial^2 h / \partial x_i \partial x_j$ vanishes identically, each P_k annihilates one of the Cartesian basis vectors e_i and e_j or both.

It is very important to understand that this partial separability property does not imply that h is best evaluated by evaluating each additive term h_k separately. For example, we could have a function of the form

$$\begin{aligned} h_1(x_1, \ldots, x_{n-1}) &= x_1 \cdot crunch(x_2, \ldots, x_{n-1}) \\ h_2(x_2, \ldots, x_n) &= crunch(x_2, \ldots, x_{n-1}) \cdot x_n, \end{aligned}$$

with $crunch(x_2, \ldots, x_{n-1})$ a computationally intensive and nonseparable common intermediate. Then $h = h_1 + h_2$ is partially separable because x_1 and x_n do not interact in a nonlinear fashion, so that the $(1, n)$ and $(n, 1)$ entries of the Hessian vanish. However, evaluating h_1 and h_2 separately is clearly not a good idea since it would entail computing $crunch(x_2, \ldots, x_{n-1})$ twice. Instead one should rewrite $h(x)$ as

$$h(x) = (1, 1)\, H(x) \quad \text{with} \quad H(x) \equiv (h_1(x), h_2(x))^T,$$

so that the gradient of h is given by

$$\nabla h(x) \quad = \quad (1, 1)\; H'(x) \quad .$$

The Jacobian of the vector function $H : \mathbb{R}^n \to \mathbb{R}^2$ is sparse and can therefore be evaluated at reduced cost. More generally, we can use the following generalization of partial separability.

2.1 Partial Function Separability and Row Splitting

We will call the vector function f *partially function-separable* whenever one of its components is partially separable in the usual sense and can therefore be split into

two. The corresponding row of the Jacobian is then also split into two so that the number of nonzeros in either of the new rows cannot be greater than the number of nonzeros in the original row. On the other hand, unless the new rows are structurally orthogonal, there is at least one column in which the number of nonzeros grows by one, whereas it is nondecreasing in all others. Hence we find that row splitting is likely to decrease the row length but increase the column length. Similarly, the number of edges in the column-intersection graph cannot grow, whereas the number of nodes and edges in the row-intersection graph must go up. However, the chromatic number of the row intersection can go down, as can be seen in the following example.

Consider a vector function $f : \mathbb{R}^3 \to \mathbb{R}^3$ whose Jacobian has a vanishing diagonal and whose first component function is partially separable. Then the first component can be split into two, and we obtain an vector function $\hat{f} : \mathbb{R}^3 \to \mathbb{R}^4$ such that the Jacobians J and $\hat{J} \equiv \hat{f}'$ have the sparsity pattern

$$J = \begin{bmatrix} 0 & \times & \times \\ \times & 0 & \times \\ \times & \times & 0 \end{bmatrix} \quad \Rightarrow \quad \hat{J} = \begin{bmatrix} 0 & \times & 0 \\ 0 & 0 & \times \\ \times & 0 & \times \\ \times & \times & 0 \end{bmatrix} \quad .$$

The associated row-intersection graphs take the following simple forms:

The column-intersection graphs are both identical to the row-intersection graph of J, so that

$$\chi(\hat{J}) \;=\; 2 \;<\; 3 \;= \chi(J) = \chi(J^T) = \chi(\hat{J}^T) \quad .$$

In this example, row splitting makes no difference for the forward mode but is beneficial for the CPR approach in the reverse mode. The corresponding weight matrix would be simply

$$W^T = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad .$$

We will see later that the dual process of column splitting is sometimes beneficial for the CPR approach in the forward mode.

If the row-splitting process is carried out as far as possible, one obtains a representation of the form

$$f(x) \;=\; B\hat{f}(x) \quad \text{with} \quad \hat{f} : \mathbb{R}^m \to \mathbb{R}^{\hat{m}} \quad \text{and} \quad \mathrm{OPS}\{\hat{f}\} \;=\; \mathrm{OPS}\{f\} \quad , \qquad (2.1)$$

where each column of the matrix $B \in \mathbb{R}^{m \times \hat{m}}$ is a Cartesian basis vector. Here and throughout the paper we consider the computational cost of merging the components

of $\hat{f}$ to those of f by addition as negligible. Applying the arguments given above, by induction one finds that

$$\rho(\hat{J}) \;\leq\; \rho(J) \quad , \quad \chi(\hat{J}) \;\leq\; \chi(J) \quad , \quad \rho(\hat{J}^T) \;\geq\; \rho(J^T) \quad . \tag{2.2}$$

If the first two inequalities hold strictly, it is advantageous in the forward mode first to evaluate $\hat{J}$ and then to multiply it by B in order to obtain $J = B\hat{J}$. It is natural to ask whether one can split off a similar linear factor on the right in order to make the column- rather than the row-incidence graph sparser. In the next subsection, we show how to do this by splitting independent variables, or arguments, rather than dependent variables, or functions. First, however, we will end this subsection with an observation that will be useful regarding the complexity of second derivatives.

For $\hat{J}$ to be nonseparable it is necessary that the nonzeros in the Hessians of each component function $\hat{f}_i$ form a dense square block. Otherwise, $\hat{f}_i$ would still be partially separable and could be split in at least two smaller component functions (increasing $\hat{m}$ by one in the process). Hence we have the following lemma.

Lemma 1 *If $\hat{f}(x) : \mathbb{R}^n \to \mathbb{R}^{\hat{m}}$ is not partially function-separable, its column-intersection graph is identical to the incidence graph G of the symmetric Hessian $\nabla_x^2[\hat{u}^T \hat{f}(x)]$ for a generic multiplier vector $\hat{u} \in \mathbb{R}^{\hat{m}}$.*

Using the terminology of Coleman and Cai [6], we find for the path and cyclic chromatic numbers $\chi_\pi(G)$ and $\chi_0(G)$

$$\rho(\hat{J}) = \chi(G) \leq \chi_0(G) \leq \chi_\pi(G) \leq \chi(G^2) \quad . \tag{2.3}$$

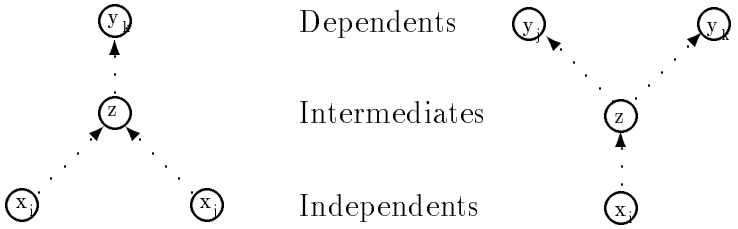
Here G^2 is the column-intersection graph of the Hessian, which does not reflect its symmetry. The chromatic numbers $\chi_\pi(G)$ and $\chi_0(G)$ determine how many gradient evaluations are required to estimate the Hessian by differencing with direct and indirect substitution, respectively. The same techniques have been applied in combination with computational differentiation, which yields Hessian vector products without any truncation errors. Again one may employ either the purely forward mode or a combination with the reverse mode, which has a lower operations count but requires more storage, as shown in Proposition 2 of Subsection 4.2. In any case it is possible to obtain second-derivative information directly from the evaluation program for f, without asking the user to supply a gradient code.

2.2 Partial Argument-Separability and Column Splitting

The row splitting described above proceeded by identifying one dependent variable $y_k = f_k(x)$ and a corresponding set of independent variables $\{x_i\}_{i \in \mathcal{I}}$ that do not interact nonlinearly in the evaluation of y_k. By *not interacting nonlinearly* we mean that $\partial y_k / \partial x_i \not\equiv 0$ for all $i \in \mathcal{I}$ but that

$$\partial^{|\mathcal{I}|} y_k \Big/ \prod_{i \in \mathcal{I}} \partial x_i \quad \equiv \quad 0 \quad ,$$

where $|\mathcal{S}|$ denotes the cardinality of a set $\mathcal{S}$. This effect must occur in particular when y_k is calculated as a sum of intermediate values each of which is constant with respect to at least one x_i with $i \in \mathcal{I}$. In the simplest case two independent variables x_i, x_j and the dependent variable y_k form a triplet (x_i, x_j, y_k) that is *disconnected* in the sense that at least one of the three dependencies between them and any intermediate variable is trivial.



Two triplets that are connected by some intermediate variable z

Similarly, there may be triplets (x_i, y_j, y_k) consisting of one independent x_i and two dependent variables y_j, y_k that are disconnected in the sense that no intermediate that depends on x_i can impact both y_j and y_k. Then we may create a duplicate $x_{i-1/2}$ of x_i and replace for all intermediates that impact y_k their functional dependence on x_i by exactly the same dependence on $x_{i-1/2}$. For example, if $\exp(x_i)$ enters into the evaluation of y_j, and $\sin(x_i)$ enters into the evaluation of y_k but not y_j we can replace the definition of the second intermediate by $\sin(x_{i-1/2})$. Thus we have increased the number of independents by one and obtained a function

$$\check{f}(x_1, x_2, \ldots, x_{i-1}, x_{i-1/2}, x_i, \ldots, x_n) \; : \; \mathbb{R}^{n+1} \to \mathbb{R}^n$$

so that

$$f(x) = \check{f}(\check{I}_i x) \quad \text{and} \quad \mathrm{OPS}\{\check{f}\} = \mathrm{OPS}\{f\} \quad ,$$

where $\check{I}_i \in \mathbb{R}^{(n+1)\times n}$ is obtained from the $(n+1) \times (n+1)$ identity matrix by adding the i-th and $(i+1)$-st column together. The operations count for $\check{f}$ is the same as that for f even if the value for $x_{i-1/2}$ is chosen differently from x_i. This follows from the assumed disconnectedness of the triplet (x_i, y_j, y_k). For example, in the situation discussed above, exp and sin are no longer necessarily evaluated at the same argument, but the kind and number of such intrinsic functions remain exactly unchanged. Since by the chain rule

$$J(x) = \check{J}(I_i x)\, \check{I}_i \quad \text{with} \quad \check{J} = (\check{f})',$$

we see that the i-th column of J has indeed been split.

More generally, we will call x_i separable and f partially argument-separable, if there exists a subset of dependent variables $\{y_k\}_{k\in\mathcal{K}}$ such that $\partial y_k / \partial x_i \not\equiv 0$ for all $k \in \mathcal{K}$ but each intermediate that depends on x_i impacts only a proper subset of the $\{y_k\}_{k\in\mathcal{K}}$. For each of these subsets we may then make a copy of x_i and redefine the dependencies accordingly. The process of column splitting can be continued until

none of the independents is separable any more. The resulting decomposition has the form

$$f(x) \;=\; \check{f}(Ax) \quad \text{with} \quad \check{f} : \mathbb{R}^{\check{n}} \to \mathbb{R}^n \quad \text{and} \quad \mathrm{OPS}\{\check{f}\} \;=\; \mathrm{OPS}\{f\} \quad ,$$

where all rows of the matrix $A \in \mathbb{R}^{\check{n} \times n}$ are Cartesian basis vectors.

Comparing the relation $J = \hat{J}\, A$ with (1.1), we see that column splitting is formally the opposite process of the Jacobian compression used in the CPR approach. As exact transposition of (2.2) we find

$$\rho(\check{J}^T) \;\leq\; \rho(J^T) \quad , \quad \chi(\check{J}^T) \;\leq\; \chi(J^T) \quad , \quad \rho(\check{J}) \;\geq\; \rho(J) \quad , \tag{2.4}$$

but there is no general relationship between $\chi(\check{J})$ and $\chi(J)$. Hence one might generally expect that column splitting improves the situation primarily for the reverse mode. However, as discussed in the Subsection 2.4 the splitting of independents may also be used to implement CPR with so-called multicoloring in the forward mode. But first, let us conclude this section by looking at an ideal situation for column splitting with regard to the reverse mode.

Consider a vector function f where each component f_k is evaluated by a separate computation so that the ratio

$$\check{\gamma}\{f\} \;=\; \left[\sum_{k=1}^{m} \mathrm{OPS}\{f_k\}\right] \Big/ \; \mathrm{OPS}\{f\} \tag{2.5}$$

is exactly equal to one. This situation occurs frequently in ordinary differential equations when the right hand side is given as a set of algebraic expressions. Suppose the k-th of these right hand sides depends nontrivially on $n_k \leq n$ of the variables. Then we can define $\check{f} : \mathbb{R}^{\check{n}} \to \mathbb{R}^m$ with

$$\check{n} \quad = \quad \sum_{i=1}^{n} n_i$$

by making for all component functions f_k one replica of each x_i that they depend on. The resulting extended Jacobian has the form

$$\check{J} \quad = \quad \begin{bmatrix} \times \cdots \times & 0 \cdots 0 & \cdots\cdots\cdots & 0 \cdots\cdots 0 & 0 \cdot 0 \\ 0 \cdots\cdots 0 & \times \cdots \times & \cdots\cdots\cdots & 0 \cdots\cdots 0 & 0 \cdot 0 \\ \cdots\cdots & \cdots\cdots & \cdots\cdots\cdots & \cdots\cdots & \cdots \\ 0 \cdots\cdots 0 & 0 \cdots 0 & \cdots\cdots\cdots & \times \cdots\cdots \times & 0 \cdot 0 \\ \underbrace{0 \cdots\cdots 0}_{n_1} & \underbrace{0 \cdots 0}_{n_2} & \cdots\cdots\cdots & \underbrace{0 \cdots\cdots 0}_{n_{m-1}} & \underbrace{\times \cdot \times}_{n_m} \end{bmatrix} \quad . \tag{2.6}$$

The nonzeros in $\check{J}$ are exactly the same as the nonvanishing entries of J, but now only one nontrivial entry occurs in each column, so that

$$\rho(\check{J}^T) \;=\; 1 \;=\; \chi(\check{J}^T) \quad .$$

Consequently, one reverse sweep with $q = 1$ yields $\check{J}$ and thus its row contraction, J. In other words, we consider each component function separately and evaluate its gradient in the scalar reverse mode. Obviously, this ideal example has an extreme degree of argument separability. Nevertheless, one can expect that significant savings are still possible if many arguments can be split with respect to most of the functions. For example, it can be seen that, if f_i and f_j share common intermediates only if $|i-j|$ is less than some width b, then $\check{f}$ can be defined such that $\rho(\check{J}^T) \leq b$.

2.3 Decomposition by Tearing of Rows and Columns

One may ask whether the complete splitting described above cannot be applied even when the ratio $\check{\gamma}\{f\}$ defined in (2.5) is greater than one. This will be the case if and only if certain intermediate values are shared in the joint computation of f and must therefore be calculated repeatedly if its components f_k are evaluated separately. To indicate that this process involves some losses in efficiency, we will refer to it as tearing of columns or rows. To emphasize the contrast, we will sometimes refer to the row and column splittings discussed in Subsections 2.1 and 2.2 as *exact*. If each column is torn into as many copies as it contains nonzeros, we obtain a function $\check{F}$ with $\mathrm{OPS}\{\check{F}\}/\mathrm{OPS}\{f\} = \check{\gamma}\{f\}$, as defined in (2.5), and a Jacobian of the structure (2.6). This complete column tearing of f amounts to considering each of its component as a completely separate function, and we have $J = \check{F}' A$ with $A^T = (I, \ldots, I) \in \mathbb{R}^{n \times n \cdot n}$. Then the reverse mode yields $\check{F}'$ with the operations count

$$\begin{aligned} \mathrm{OPS}\{I\check{F}'\} &\leq (r+4)\,\mathrm{OPS}\{\check{F}\} \\ &= (r+4)\,\check{\gamma}\,\{f\}\mathrm{OPS}\{f\} \\ &\leq (r+4)\,\rho\{\check{J}^T\}\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

where the last inequality follows from the relation $\check{\gamma}\{f\} \leq \rho\{\check{J}^T\}$, which will be established in Lemma 3 of Section 3.3. By comparison with (1.4) for $q \geq \rho\{J^T\}$ we see that complete column tearing can theoretically not be worse and is likely to have a lower operations count than the reverse NR approach applied directly to $\check{J}$. Moreover, there may also be a benefit for the CPR approach in the forward mode.

In contrast to exact argument splitting, complete column tearing keeps the number of nonzeros in all rows constant. Moreover, the column-intersection graph becomes the union of m disconnected cliques, which contain at most $\rho(J) = \max\{n_k\}$ elements. Hence we have

$$\chi(\check{F}') \;=\; \rho(\check{F}') \;=\; \rho(J) \;\leq\; \chi(J) \quad ,$$

so that in the forward CPR approach

$$\begin{aligned} \mathrm{OPS}\{\check{F}'I\} &\leq [1 + 3\chi(\check{F}')]\mathrm{OPS}\{\check{F}\} \\ &= \check{\gamma}\{f\}\,[1 + 3\rho(J)]\mathrm{OPS}\{f\} \\ &\leq [\rho(J^T) + 3\,\rho(J^T)\rho(J)]\mathrm{OPS}\{f\} \quad . \end{aligned}$$

If the gap between $\chi(J)$ and $\rho(J)$ is sufficiently large, this approach may be more efficient than straightforward CRP. However, in practice it may be hard to extract from the evaluation program for f exactly those calculations needed to obtain one particular dependent value y_k without carrying out any unnecessary calculations. As we will see, essentially the same operations count can be obtained by a dynamically sparse version of the reverse mode.

The "transposed" concept of *row tearing* is somewhat less intuitive, and the resulting vector function depends in particular on the current argument. Suppose we define for each x_i the univariate function

$$f_x^{(i)} \equiv f(\bar{x}_1, \ldots, \bar{x}_{i-1}, x_i, \bar{x}_{i+1} \ldots, \bar{x}_n) \quad , \tag{2.7}$$

where $\bar{x}_j$ represents x_j held constant at its current value. Let $m_i \leq m$ be the number of component functions that depend nontrivially on x_i in that $\partial f_k / \partial x_i \not\equiv 0$. Then we may omit the $m - m_i$ constant components so that $f_x^{(i)} : \mathbb{R} \to \mathbb{R}^{m_i}$. For each i we may theoretically evaluate and store all intermediates occurring in the evaluation of f that do not depend on x_i. These operations need not be counted in the effort for subsequently evaluating $f_x^{(i)}$, so we obtain a generally lower complexity

$$\mathrm{OPS}\{f_x^{(i)}\} \leq \mathrm{OPS}\{f\} \quad .$$

This is exactly the complexity that would arise during differencing if $f(x + \varepsilon e_i)$ could be evaluated without redoing the parts of the evaluation at the base points x that are unaffected by the increment ε in x_i. If each f_k is fully separable (i.e., the sum of univariate scalar functions), then the ratio

$$\hat{\gamma}\{f\} = \left[\sum_{i=1}^{n} \mathrm{OPS}\{f_x^{(i)}\}\right] \Big/ \mathrm{OPS}\{f\} \leq n \tag{2.8}$$

is equal to one. In the presence of joint intermediates, $\hat{\gamma}\{f\}$ may be any number between 1 and n. In any case we can combine the $f^{(i)}$ to the function

$$\hat{F} \equiv [(f^{(1)})^T, (f^{(2)})^T, \ldots, (f^{(n)})^T]^T : \mathbb{R}^n \to \mathbb{R}^{\hat{m}} \quad ,$$

where $\hat{m} = \sum_i m_i$. The associated Jacobian $\hat{F}'$ has the transposed structure of (2.6), so that now

$$\rho(\hat{F}') = 1 = \chi(\hat{J}) \quad .$$

Hence the forward mode yields J with the complexity

$$\begin{aligned} \mathrm{OPS}\{\hat{F}'e\} &\leq 5\,\mathrm{OPS}\{\hat{F}\} \\ &= 5\,\hat{\gamma}\{f\}\,\mathrm{OPS}\{f\} \\ &\leq 5\,\rho\{\hat{J}\}\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

where the last inequality follows from the relation $\hat{\gamma}\{f\} \leq \rho(\hat{J})$, which will be established in Section 3.3.

The superscripts were chosen such that going from f to $\hat{f}$ by row splitting and on to $\hat{F}$ by tearing is generally beneficial for the forward mode, whereas successively increasing the number of columns by going from f to $\check{f}$ and $\check{F}$ is generally beneficial for the reverse mode. The superscripts of the corresponding ratios $\hat{\gamma}$ and $\check{\gamma}$ can also be memorized as representing the average size of predecessor sets and successor sets in the computational graph, respectively. This relation is established in Subsection 3.3. In view of the Jacobian structure (2.6), we may also refer to $\check{f}$ and $\check{F}$ as *horizontal* expansions of f. Analogously $\hat{f}$ and $\hat{F}$ may be called *vertical* expansions of f. Conversely we may refer to f as horizontal and vertical contraction of $\check{f}$ and $\check{F}$ or $\hat{f}$ and $\hat{F}$, respectively.

2.4 Relation to Multicoloring

It has often been observed [20] that for a partitioned vector function

$$f(x) \;=\; \begin{bmatrix} f^{(1)}(x) \\ f^{(2)}(x) \end{bmatrix} \quad : \quad \mathbb{R}^n \to \mathbb{R}^{m_1+m_2}$$

the partial Jacobians $J^{(i)} = \left(f^{(i)}\right)'$ for $i = 1, 2$ may satisfy

$$\chi(J) \;>\; \chi(J^{(1)}) + \chi(J^{(2)}) \quad .$$

Then the CPR approach should be applied to obtain $J^{(1)}$ and $J^{(2)}$ separately, which is more economical even if

$$\mathrm{OPS}\{f\} \;\approx\; \mathrm{OPS}\{f^{(i)}\} \quad \text{for} \quad i = 1, 2$$

because of the presence of many common intermediates in evaluating $f^{(1)}$ and $f^{(2)}$. Formally, this can be interpreted as tearing all columns of J into two copies so that

$$\check{F}\begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix} \quad = \quad \begin{bmatrix} f^{(1)}(x^{(1)}) \\ f^{(2)}(x^{(2)}) \end{bmatrix} \quad : \quad \mathbb{R}^{2n} \to \mathbb{R}^m$$

with $x^{(1)}$ and $x^{(2)}$ duplicates of x. Hence we have the representation

$$f(x) = \check{F}(Ax) \quad \text{with} \quad A^T = [I, I]^T \in \mathbb{R}^{n \times 2n} \quad .$$

The extended Jacobian takes the form

$$\check{F}' \quad = \quad \begin{bmatrix} J^{(1)} & 0 \\ 0 & J^{(2)} \end{bmatrix} \quad ,$$

an expansion that has also been considered in [20]. Since the column-intersection graph for $\check{J}$ is the disconnected union of the two graphs associated with $J^{(1)}$ and $J^{(2)}$, its chromatic number is given by

$$\chi(\check{J}) \;=\; \max\{\chi(J^{(1)}), \chi(J^{(2)})\} \quad .$$

Hence the CPR approach yields $\check{J}$ and thus J at a total cost not exceeding

$$\chi(J^{(1)})\,\mathrm{OPS}\{f^{(1)}\} + \chi(J^{(2)})\,\mathrm{OPS}\{f^{(2)}\} \leq \max\{\chi(J^{(1)}), \chi(J^{(2)})\}[\mathrm{OPS}\{f^{(1)}\} + \mathrm{OPS}\{f^{(2)}\}] \quad .$$

In general, one has to split f into more than two subvectors in order to get the individual $\chi(J^{(i)})$ down to a small number. It then becomes important to ask how the sum of the complexities $\mathrm{OPS}\{f^{(i)}\}$ grows with the number of parts. If there are none or few common intermediates, it will be essentially equal to $\mathrm{OPS}\{f\}$; but if most intermediates are shared by all dependents, the ratio (2.5) can grow proportionally to the number of partitions.

For the NR approach in the forward mode, row partitioning makes no sense since the maximal row length satisfies

$$\rho(J) \;=\; \max\{\rho(J^{(1)}), \rho(J^{(2)})\}$$

and therefore cannot be reduced at all.

2.5 A Binary Interaction Example

Suppose we have an unconstrained problem with an objective function of the form

$$f(x) \quad \equiv \quad \sum_{1 \leq i < j \leq n} f_{ij}(x_i, x_j),$$

where all element functions $f_{ij} : \mathbb{R}^2 \to \mathbb{R}$ are nonseparable. For example, the variables x_i could represent the coordinates of atoms that are aligned along one coordinate axis. The energy of such an arrangement is often modeled as the sum of $n(n+1)/2$ pairwise interactions f_{ij}. This situation has alos been examined in some detail as the *exponential* example in [20].

Now suppose we wish to compute the gradient of f, given a computer program for its evaluation. To illustrate the crucial role of common intermediates, we write each f_{ij} in the form

$$f_{ij}(x_i, x_j) = \tilde{f}_{ij}(prep_i(x_i), prep_j(x_j)) \quad ,$$

where $\tilde{f}_{ij} : \mathbb{R}^2 \to \mathbb{R}$ like f_{ij}, and the n univariate functions $prep_i$ somehow prepare the variables x_i for their involvement in the f_{ij}. Since $m = 1 = \rho(J^T)$, the reverse mode yields the gradient of f at no more than $r + 4$ times the operations count of f itself. However, suppose we wish to evaluate the gradient and Hessian in the forward mode, possibly to avoid the increase in memory requirement and interpretive overhead, or simply because no suitable software for the reverse mode is available.

Using the obvious partial separability, we may rewrite

$$f(x) \;=\; e^T \hat{f}(x) \quad \text{with} \quad \hat{f}(x) \;=\; (f_{ij}(x))_{1 \leq i < j \leq n},$$

where $e^T = A$ is the vector of $n(n+1)/2$ ones and the ordering of the components in $\hat{f}$ does not matter. Because any two columns of $\hat{J}$ share nonzeros in (exactly) one row, the column intersection graph is a clique and therefore only allows the trivial coloring with $\chi(\hat{J}) = n$. Hence the CPR technique for estimating the gradient of f via the Jacobian of $\hat{f}$ would require n full evaluations of $\hat{f}$.

To avoid this unacceptable complexity, we consider a complete column tearing, where each contribution f_{ij} is considered as an independent function from all the others. Then the work ratio defined in (2.5) is given by

$$\begin{aligned} \check{\gamma}\{\hat{f}\} &= \frac{\sum_{1 \le i < j \le n} \left[\mathrm{OPS}\{\tilde{f}_{ij}\} + \mathrm{OPS}\{prep_i\} + \mathrm{OPS}\{prep_j\} \right]}{\sum_{1 \le i \le n} \mathrm{OPS}\{prep_i\} + \sum_{1 \le i < j \le n} \mathrm{OPS}\{\tilde{f}_{ij}\}} \\ &= 1 + \frac{(n-2) \sum_{1 \le i \le n} \mathrm{OPS}\{prep_i\}}{\sum_{1 \le i \le n} \mathrm{OPS}\{prep_i\} + \sum_{1 \le i < j \le n} \mathrm{OPS}\{\tilde{f}_{ij}\}} \quad . \end{aligned}$$

If the preparatory functions are mere identities $pred_i(x_i) = x_i$, then $\hat{f}$ is fully argument separable, and we have $\hat{\gamma}\{\hat{f}\} = 1$. The complete tearing is still advantageous if the preparatory functions $prep_i$ are cheap compared to the actual binary interactions $\tilde{f}_{ij}$. On the other hand, if these univariate transformations are expensive, then one may get an n-fold increase in complexity, which amounts to the same as differencing the gradient of each f_{ij} separately.

One can also apply a complete row tearing to $\hat{f}$. Since each independent impacts $(n-2)$ components of $\hat{f}$ the ratio $\hat{\gamma}$ defined in (2.8) is given by

$$\begin{aligned} \hat{\gamma}\{\hat{f}\} &= \frac{\sum_{1 \le i \le n} \left[\mathrm{OPS}\{prep_i\} + \sum_{j \ne i} \mathrm{OPS}\{\tilde{f}_{ij}\} \right]}{\sum_{1 \le i \le n} \mathrm{OPS}\{prep_i\} + \sum_{1 \le i < j \le n} \mathrm{OPS}\{\tilde{f}_{ij}\}} \\ &= 1 + \frac{\sum_{1 \le i < j \le n} \mathrm{OPS}\{\tilde{f}_{ij}\}}{\sum_{1 \le i \le n} \mathrm{OPS}\{prep_i\} + \sum_{1 \le i < j \le n} \mathrm{OPS}\{\tilde{f}_{ij}\}} \quad . \end{aligned}$$

The row tearing is efficient if the preparatory functions $prep_i$ dominate the computational cost, in which case the column tearing is not so advantageous. If these costs are significant but not dominant both tearings may result in an unacceptable complexity for the CPR approach in the forward mode. In this particular example one may then utilize a mixture between the two in the following way.

First one can order the components of $\hat{f}$ such that the top quarter corresponds to element functions f_{ij} with $1 \le i < j \le n/2$ and the second quarter to those with $n/2 < i < j \le n$. Then the rows in the remaining bottom half correspond to element functions f_{ij} with $1 \le i \le n/2 < j \le n$. The chromatic number of this $(n^2/2) \times n$ or $(n^2-1)/2 \times n$ matrix is 2, since the first $n/2$ and the last $n - n/2$ columns form groups that are pairwise structurally orthogonal. Hence we can estimate the bottom part using two function evaluations or, equivalently, Jacobian-vector products. The two

top quarters represent copies of the original problem with the number of independents cut in half. Hence they can be decomposed recursively in the same way; and since the corresponding columns are structurally orthogonal, each evaluation for the top quarter can be combined with an evaluation for the second quarter. In this way the whole Jacobian can be computed by using only $2 \log_2 n$ Jacobian-vector evaluations.

Using the method of Newsam and Ramsdell [18], one can do even better. Let s be any n-vector whose components σ_i are all different from each other. Then the two Jacobian-vector products

$$\left.\frac{\partial \hat{f}(x + \alpha e + \beta s)}{\partial(\alpha, \beta)}\right|_{\alpha=0=\beta} = \hat{J}(x)[e, s] \in \mathbb{R}^{\hat{m} \times 2}$$

completely determine $\hat{J}(x)$ since the nonzeros in the row corresponding to f_{ij} form a small vector-matrix product of the form

$$\left.\frac{\partial f_{ij}(x + \alpha e + \beta s)}{\partial(\alpha, \beta)}\right|_{\alpha=0=\beta} = \left[\frac{\partial f_{ij}}{\partial x_i}, \frac{\partial f_{ij}}{\partial x_j}\right] \begin{pmatrix} 1 & \sigma_k \\ 1 & \sigma_l \end{pmatrix}$$

for some index pair $k \neq l$. Since by assumption $\sigma_k \neq \sigma_l$, each 2×2 matrix is invertible, and the two nonzero components of ∇f_{ij} can thus be easily recovered. While uniformly spaced σ_k would be acceptable in this case, more sophisticated choices such as roots of unity may be required to make the linear systems reasonably well conditioned when the maximal row length is larger.

Differentiating once more, one obtains with $x(\alpha, \beta) \equiv x + \alpha e + \beta s$

$$\begin{bmatrix} \frac{\partial^2}{\partial \alpha^2} f_{ij}(x(\alpha,\beta)) & \frac{\partial^2}{\partial \alpha \partial \beta} f_{ij}(x(\alpha,\beta)) \\ \frac{\partial^2}{\partial \beta \partial \alpha} f_{ij}(x(\alpha,\beta)) & \frac{\partial^2}{\partial \beta^2} f_{ij}(x(\alpha,\beta)) \end{bmatrix}_0 = \begin{pmatrix} 1 & 1 \\ \sigma_k & \sigma_l \end{pmatrix} \begin{bmatrix} \frac{\partial^2 f_{ij}}{\partial x_i^2} & \frac{\partial^2 f_{ij}}{\partial x_i \partial x_j} \\ \frac{\partial^2 f_{ij}}{\partial x_j \partial x_i} & \frac{\partial^2 f_{ij}}{\partial x_j^2} \end{bmatrix} \begin{pmatrix} 1 & \sigma_k \\ 1 & \sigma_l \end{pmatrix}.$$

Thus we see that all Hessians $\nabla^2 f_{ij}$ can be computed from the $n \times 2 \times 2$ derivative tensor of $f(x + \alpha e + \beta s)$ with respect to α and β.

It is often optimistically assumed that for most square sparsity patterns, not only the difficult to compute chromatic number itself but also a heuristically computed coloring number lies within a factor of two of the maximal row length. It seems unlikely that this property holds for the vertical expansions of partially separable functions, which may have very long columns and thus highly connected intersection graphs. Therefore we presume that the method of Newsam and Ramsdell deserves further investigation in this context. Also, one might hope that sometimes the column intersection graph of the transposed $\hat{J}$ is less connected, even though it is likely to have more nodes. However, on the scalar example above, each column has exactly $n-1$ nonzero entries so that the reverse mode is not cheaper if one wishes to calculate the whole Jacobian J. Nevertheless, it is very efficient for calculating the accumulated gradient

$$\nabla f(x) \quad = \quad e^T \hat{J}(x) \quad ,$$

or the gradient of a Lagrangian, where the vector of ones e would be replaced by a vector of Lagrange multipliers.

3 The Evaluation Program and Its Complexity

In this third section we will describe the computational graph for a given vector function f and develop several complexity estimates at the elemental level. We will also define the following key characteristics of f and its Jacobian J.

RAM$\{f\}$	Bound on the the maximal number of live variables.
OPS$\{f\}$	Sum of all elemental operations counts.
$\chi(\hat{J})$	Chromatic number of the incidence graph of the generic Lagrangian.
$\rho(J)$	Maximal number of independents that impact any dependent variable.
$\rho(\hat{J})$	Maximal number of independents that impact any intermediate variable.
$\hat{\gamma}\{f\}$	Average number of independents on which an intermediate depends.
$\rho(J^T)$	Maximal number of dependent variables impacted by any independent.
$\rho(\check{J}^T)$	Maximal number of dependent variables impacted by any intermediate.
$\check{\gamma}\{f\}$	Average number of dependents impacted by an intermediate variable.

In computational practice, a vector function $f : \mathbb{R}^n \to \mathbb{R}^m$ is evaluated by a sequence of scalar assignments

$$v_j = \varphi_i(v_i)_{i \to j} \quad , \tag{3.1}$$

where the variables v_j may be real or complex. The relation $i \to j$ between integer indices means that v_j *depends directly* on v_i because the latter is an argument of the *elementary function* $\varphi_j \in \mathcal{C}^2$. We will assume that the partial ordering induced by this dependency relation is compatible with the natural ordering of the integers, so that

$$j \to i \Longrightarrow j < i \quad . \tag{3.2}$$

Obviously there may be several such *topological* orderings of the *computational graph* with the vertices v_j. The edges are defined by the dependence relation $\to$, and the vertices can be labeled with the elementary function φ_j. Without loss of generality we may require that the first n variables v_i represent the *independent* variables and the last m represent the *dependent variables*. Then the function $y = f(x)$ can evaluated by executing the elementary assignments (3.1) in one big loop with j ranging from 1 to $o + m$. Here the nonnegative integer o represents the number of intermediate variables, which we expect to be much larger than both n and m for seriously nonlinear problems. Thus we can combine the variables v_i into the three vectors

$$\begin{array}{rcll} x & \equiv & (v_{1-n}, \ldots & \ldots, v_{-1}, v_0) \quad (independent) \\ z & \equiv & (v_1, v_2, \ldots & \ldots, v_o) \quad (intermediate) \\ y & \equiv & (v_{o+1}, \ldots & \ldots, v_{o+m}) \quad (dependent) \end{array} \quad ,$$

so that $x_i \equiv v_{i-n}$ and $y_k \equiv v_{o+k}$. The independent and dependent variables represent the roots and leaves of the computational graph. To make partial function-separability as defined in Subsection 2.1 a special case of partial separability in the usual sense, we have to impose a *final summation* condition, which can always be achieved by minor modifications of the graph. We state this requirement formally.

Final Summation Condition

Elementary function of the additive form

$$v_j \;=\; \varphi_j(v_i)_{i \to j} = \sum_{i \to j} v_i \tag{3.3}$$

must occur for all $i > o$ but cannot occur for any j with $v_j \to y_k$ for some $k \leq m$. The cost of the final summations is considered negligible.

In other words, each final elementary function must be a (possibly unary) addition, but none of the immediate predecessors may be obtained as a sum. When a final elementary function φ_{o+k} is multivariate and nonadditive, we may simply relabel v_{o+k} as an intermediate and use a unary summation to make copy that serves as the k-th dependent variable. On the other hand, immediate predecessors of dependents that are themselves obtained as sums can be eliminated by merging the two summations. The "transposed" condition on the independent variables is that no identical copies are made so that none of the φ_j with $x_i \to v_j$ for some x_i may be the identity function.

3.1 Counting Contemporaries, Ancestors, and Descendants

The v_j considered here are mathematical variables rather than memory locations on the computer. Since the storage requirement would otherwise be at least o, we must allow that v_j overwrites some v_i, provided it is certain that

$$i \to k \implies k < j \quad , \tag{3.4}$$

which means that v_i can no longer occur as an argument once v_j has been computed. In the remainder we will denote by RAM$\{f\}$ an upper bound on the number of *live* variables v_j that must be in storage at any one time during the evaluation process. For Fortran programs, a suitable bound can be determined at compile time since there is no dynamic storage allocation.

We will use the notation

$$i \rightsquigarrow j \quad \text{or} \quad v_i \rightsquigarrow v_j$$

to indicate that v_j depends on v_i either directly in that $i \to j$ or indirectly through intermediates. In graph theoretical terms $i \rightsquigarrow j$ means that a directed path connects the vertices v_i and v_j. For independent and dependent variables we will often write

$$x_i \rightsquigarrow v_j \quad \text{and} \quad v_j \rightsquigarrow y_k$$

in lieu of $i \rightsquigarrow (j-n)$ or $j \rightsquigarrow (o+k)$, respectively. For all v_j we may define the sets

$$X(v_j) \equiv \{x_i : x_i \rightsquigarrow v_j\} \quad \text{and} \quad Y(v_j) \equiv \{y_k : v_j \rightsquigarrow y_k\} \tag{3.5}$$

of independent ancestors and dependent descendants, respectively. By subsuming constants into the definition of elementary functions and eliminating unnecessary calculations, one can ensure that the sets $X(v_j)$ and $Y(v_j)$ are nonempty for all intermediates. In other words, all intermediates lie on a path from an independent to a dependent variable, so that for all $1 \le j \le o$

$$x_i \rightsquigarrow v_j \rightsquigarrow y_k \quad \text{for some} \quad x_i \quad \text{and} \quad y_k \quad . \tag{3.6}$$

The set $X(y_k)$ contains exactly the independent variables on which y_k may be nontrivially dependent. Similarly, $Y(x_i)$ contains exactly those dependent variables that are nontrivially dependent on x_i. Denoting by $|\mathcal{S}|$ the cardinality of a set $\mathcal{S}$ one finds that the maximal row and column lengths of the Jacobian J are given by

$$\rho\{J\} = \max_{k \le m} |X(y_k)| \quad \text{and} \quad \rho\{J^T\} = \max_{i \le n} |Y(x_i)| \quad . \tag{3.7}$$

The concepts of function- and argument-separability used in Subsections 2.1 and 2.2 can now be reintroduced rigorously as follows.

3.2 Function and Argument Splitting on the Graph

Function separability

A dependent variable y_k is called separable if for all $1 \le j \le o$

$$v_j \rightsquigarrow y_k \quad \Rightarrow \quad x_i \not\rightsquigarrow v_j \quad \text{for some} \quad x_i \rightsquigarrow y_k \quad ,$$

in which case f is called partially function-separable.

This condition is equivalent to the property

$$v_j \to y_k \quad \Rightarrow \quad X(v_j) \subset X(y_k) \quad , \tag{3.8}$$

where the symbol $\subset$ excludes equality. Now let $X_{k\,l}$ for $l = 1, \ldots, \bar{l}$ be a numbering of all $\bar{l}$ distinct subsets of the form $X(v_j)$ with $v_j \to y_k$. Then we can split y_k into $\bar{l}$ copies $y_{k\,l}$ defined by

$$y_{k\,l} \;=\; \sum_{X_{k\,l}=X(v_j)} v_j \quad . \tag{3.9}$$

After renumbering some vertices and updating the dependencies accordingly, one has now obtained a computational graph for a function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^{\hat{m}}$ with $\hat{m} = m + \bar{l} - 1$ as originally introduced in Section 2.1. From now on we will assume that $\hat{f}$ is maximal, that is, that it has been obtained from f by performing all possible function splittings.

Argument separability

An independent variable x_i is called separable if for all $1 \le j \le o$

$$x_i \rightsquigarrow v_j \quad \Rightarrow \quad v_j \not\rightsquigarrow y_k \quad \text{for} \quad \text{some} \quad x_i \rightsquigarrow y_k \quad ,$$

in which case f is called partially argument-separable.

This condition is equivalent to the property

$$x_i \to v_j \quad \Rightarrow \quad Y(v_j) \subset Y(x_i) \quad , \tag{3.10}$$

where the symbol $\subset$ still excludes equality. In exact analogy to the function-separable case discussed above we may now number $\bar{l}$ sets $Y_{i\,l} = Y(v_j)$ for some $v_j \leftarrow x_i$ and replace the original assignment $v_j = \phi_j(\ldots, x_{i-1}, x_i, x_{i+1} \ldots)$ by

$$v_j \equiv \phi_j(\ldots, x_{i-1}, x_{k\,l}, x_{i+1}, \ldots) \quad \text{if} \quad Y_{i\,l} = Y(v_j) \quad . \tag{3.11}$$

Whenever all $x_{i\,l} = x_i$ the values of the direct successors $v_j \leftarrow x_i$ as well as all later intermediates are obviously the same. Therefore, the horizontal expansion $\check{f} : \mathbb{R}^{\check{n}} \to \mathbb{R}^m$ defined by the new graph with $\check{n} = n + \bar{l} - 1$ has the properties described in Section 2.2. From now on we will assume that $\check{f}$ is maximal, that is, that it has been obtained from f by performing all possible argument splittings.

Now we can characterize the row and column lengths of the Jacobians $\hat{f}$ and $\check{f}$ directly in terms of the computational graph.

Lemma 2
Under the final summation condition we have for the maximal expansions $\hat{f}$ and $\check{f}$

$$\begin{aligned} \rho(\hat{J}) \;&=\; \max_{1 \le j \le o} |X(v_j)| \;\le\; \rho(J) \quad , \\ \rho\{\check{J}^T\} \;&=\; \max_{1 \le j \le o} |Y(v_j)| \;\le\; \rho(J^T) \quad , \end{aligned}$$

where the $X(v_j)$ and $Y(v_j)$ are defined in (3.5).

Proof. Since we have assumed that all intermediates impact at least one dependent of f, the same is true for $\hat{f}$. Excluding the possibility of accidental cancellation, we must therefore have $\rho\{\hat{J}\} \geq |X(v_j|$ for all $1 \leq j \leq o$. Now suppose that the gradient $\nabla\hat{y}_k$ of some component $\hat{y}_k$ of $\hat{f}$ has more nonzeros than any of the preceding intermediates v_j. Then that $\hat{y}_k$ is in fact separable, which contradicts the definition of $\hat{f}$. Thus we must have equality as asserted, and the bound by $\rho(J)$ is an immediate consequence of (3.7). The second assertion follows analogously. ∎

Finally, we note that the row- and column splitting processes reinforce rather than obstruct each other. More specifically, when some independent x_i is split, all separable dependents y_k maintain that property even if $X(y_k)$ contains x_i and may therefore be enlarged. Similarly, row splittings cannot reduce the number of separable arguments, which can be used for subsequent column splittings. Hence, there must be a function

$$f^\diamond \;:\; \mathbb{R}^{\check{n}} \to \mathbb{R}^{\check{m}}$$

such that

$$f(x) \;=\; Bf^\diamond(Ax) \quad \text{and} \quad \mathrm{OPS}\{f^\diamond\} \;=\; \mathrm{OPS}\{f\} \quad ,$$

where $A \in \mathbb{R}^n \to \mathbb{R}^{\check{n}}$ and $B \in \mathbb{R}^m \to \mathbb{R}^{\hat{m}}$ as before. We can identify the previously discussed vertical and horizontal expansions of f as

$$\hat{f}(x) \;=\; f^\diamond(Ax) \quad \text{and} \quad \check{f}(\check{x}) \;=\; Bf^\diamond(\check{x}) \quad ,$$

where $\check{x} \in \mathbb{R}^{\check{n}}$ is the replicated variable vector. Obviously the only difference between the computational graphs for $\hat{f}, \check{f}, f^\diamond$, and the original f is in the leaves, roots and the way the first layers of intermediates v_j (with $x_i \to v_j$ for some i) are defined as elementary functions of the intermediates. Without ambiguity we may therefore denote the dependents of $\hat{f}$ and $f^\diamond$ by $\hat{y} = (\hat{y}_k)_{1 \leq k \leq \hat{m}}$ and the independents of $\check{f}$ and $f^\diamond$ by $\check{x} = (\check{x}_i)_{1 \leq i \leq \hat{n}}$, respectively. The relation between these vectors and the original independents and dependents is simply $y = B\hat{y}$ and $\check{x} = Ax$. The definition of the first layer will be clear from the context.

Generally speaking, we can expect that the Jacobian of $f^\diamond$ is much larger and sparser than that of the original f. Because of (2.2) and (2.4) it is clear that the NR approach in the forward and reverse modes is best applied to the $\hat{f}$ and $\check{f}$, respectively. This is no longer true when the factors A and B are allowed to be general linear transformations. Such further generalization is useful in cases where the evaluation of f involves linear functionals like the average $e^T x$ of the independent or dependent variables.

3.3 Elemental Complexity Assumptions

Our main restriction on the elementary functions φ_i is that the partial derivatives

$$c_{ij} \equiv \begin{cases} \partial \varphi_i / \partial v_j & \text{if} \quad j \to i \\ 0 & \text{if} \quad j \not\to i \end{cases}$$

and

$$c_{ijk} \equiv \begin{cases} \partial^2 \varphi_i / \partial v_j \partial v_k & \text{if} \quad j \to i \quad \text{and} \quad k \to i \\ 0 & \text{if} \quad j \not\to i \quad \text{or} \quad k \not\to i \end{cases}$$

are well defined and easily evaluated at all arguments of interest. This is certainly the case when the evaluation program is written in a standard high-level language such as Fortran or C. Then the compiler breaks down the evaluation into a sequence of arithmetic operations and intrinsic function calls.

For some purposes it is advantageous to view more complex computational units as elementary building blocks. This approach has the advantage of reducing the interpretive overhead and facilitates some local preaccumulation of derivatives. For example, in the source translator ADIFOR, right-hand sides of assignments are treated as *elementary functions*, whose gradients are computed by the reverse mode in the form of compilable code. This compile-time differentiation can be easily generalized to function and subroutines, especially if their code is tight in that it does not contain variable dimensions or loop lengths. It has proven very efficient for the evaluation of first derivatives. Unfortunately, the trade-offs are more complicated, if one also wishes to compute second or higher derivatives. However, it is clear that linear or bilinear vector-vector and matrix-vector operations should be treated as *elementary functions*, since their first and higher derivatives are easy to store and manipulate, with many of them vanishing altogether. For notational simplicity we will continue to assume that all elementary functions are scalar valued, but we allow the number of local independents $|\{i : i \to j\}|$ to be arbitrarily large.

Our key assumption is that the cost for computing the first and second derivatives of each φ_j is no more than twice that of computing ϕ_j by itself, so that

$$\text{OPS}\{\varphi_j, \nabla \varphi_j, \nabla^2 \varphi_j\} \;\leq\; 2\,\text{OPS}\{\varphi_j\} \quad . \tag{3.12}$$

In fact, this bound is quite pessimistic, since for all linear and bilinear operations the derivatives come virtually for free, and for most intrinsic functions the first two derivatives are easily obtained from the function itself. For sinusoidal functions the bound appears to be sharp, but even there sin and cos are often evaluated in pairs anyway, in which case no extra derivative evaluations are required in theory. In practice, such savings could be realized only if the automatic differentiation tool did some compiler-like dependency analysis and optimization.

The temporal complexity measure OPS{} may account not only for arithmetic operations but also for memory accesses. Naturally, we cannot distinguish the access

costs to different levels of the memory hierarchy and will assume exact additivity so that (at least on a serial machine)

$$\mathrm{OPS}\{f\} \;\equiv\; \sum_{j=1}^{o} \mathrm{OPS}\{\varphi_j\} \quad , \tag{3.13}$$

where we have again assumed that the cost of the final summations φ_{o+k} for $k = 1 \ldots m$ is negligible. Apart from generating the derivatives $\nabla\varphi_j$ and $\nabla^2\varphi_j$, we must also consider the cost of incorporating them into the chain rule. The elementary operations addition and subtraction play a special role, because all first derivatives are 1 or -1 and all second derivatives vanish identically. In these cases no multiplications are required to multiply the local gradients or Hessians by vectors or matrices. In general, we assume that the effort of forming an inner product of the gradient $\nabla\varphi_j$ with a compatible vector, or multiplying the Hessian $\nabla^2\varphi_j$ from the left and right by two vectors, or incrementing a given vector by a multiple of $\nabla\varphi_j$, is bounded according to

$$\max\Big\{\mathrm{OPS}\{(\nabla\varphi_j)^T w\}, \mathrm{OPS}\{u^T\nabla^2\varphi_j w\}, \mathrm{OPS}\{+\omega\nabla\varphi_j\}\Big\} \;\le\; 3\,\mathrm{OPS}\{\varphi_j\} \quad , \tag{3.14}$$

where the + sign indicates that adding the result to a give vector is considered an integral part of the calculation. If a multiplication is no more expensive than an addition, the bound is sharp for the multiplication operator $v_j = \varphi_j(v_1, v_2) = v_1 \cdot v_2$, where $(\nabla\varphi_j)^T w = v_2 w_1 + v_1 w_2$ and $u^T\nabla^2\phi w = u_1 w_2 + u_2 w_1$.

Let us finally perform individual operations counts for the component functions f_k and the univariate functions $f_x^{(i)}$ defined in Subsection 2.3. After discounting all elementary functions φ_j that have no impact on a given f_k, we obtain the operations count

$$\mathrm{OPS}\{f_k\} \;=\; \sum_{j \prec k} \mathrm{OPS}\{\varphi_j\} \quad . \tag{3.15}$$

Similarly, the (re)evaluation of $f_x^{(i)}$ requires only the calculation of the elementary functions that depend on x_i, so that

$$\mathrm{OPS}\{f_x^{(i)}\} \;=\; \sum_{i \prec j} \mathrm{OPS}\{\varphi_j\} \quad , \tag{3.16}$$

where the subscript x indicates that the definition of $f_x^{(i)}$ depends on the "current" point x viewed as a constant. Substituting these expressions into the definitions (2.8) and (2.5), we obtain the following result.

Lemma 3 *The complexity ratios defined in (2.5) and (2.8) satisfy*

$$\check{\gamma}\{f\} \;\le\; \rho(\check{J}^T) \quad \text{and} \quad \hat{\gamma}\{f\} \;\le\; \rho(\hat{J}) \quad .$$

Proof. Summing (3.15) and (3.16) over k and i, we obtain with $Y(v_j)$ and $X(v_j)$ as defined in (3.5) by changing the order of summation:

$$\begin{aligned} \check{\gamma}\{f\}\,\mathrm{OPS}\{f\} &= \sum_{j=1}^{o} \left(\sum_{j \rightsquigarrow k} \mathrm{OPS}\{\varphi_j\} \right) \\ &= \sum_{j=1}^{o} |Y(v_j)|\,\mathrm{OPS}\{\varphi_j\} \;\leq\; \rho(\check{J}^T)\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

where the last inequality follows from (3.7). By interchanging rows and columns we find similarly

$$\begin{aligned} \hat{\gamma}\{f\}\mathrm{OPS}\{f\} &= \sum_{j=1}^{o} \sum_{i \rightsquigarrow j} \mathrm{OPS}\{\varphi_j\} \\ &= \sum_{j=1}^{o} |X(v_j)|\,\mathrm{OPS}\{\varphi_j\} \;\leq\; \rho(\hat{J})\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

which completes the proof. ∎

In this thirs section we have shown that there exist constant matrices $B \in \mathbb{R}^{m \times \hat{m}}$ and $A \in \mathbb{R}^{\hat{n} \times n}$ (whose rows and columns are Cartesian basis vectors, respectively) such that

$$f(x) \;=\; B\hat{f}(x) \;=\; \check{f}(Ax) \;=\; Bf^{\diamond}(Ax)$$

and consequently

$$J(x) \;=\; B\,\hat{J}(x) \;=\; \check{J}(Ax)\,A \;=\; B\,J^{\diamond}(Ax)\,A \quad .$$

Since pre- and post-multiplication by A and B involve only additions, we neglect these costs and assume that

$$\mathrm{OPS}\{f\} = \mathrm{OPS}\{\check{f}\} = \mathrm{OPS}\{\hat{f}\} = \mathrm{OPS}\{f^{\diamond}\}$$

and

$$\mathrm{OPS}\{J\} \leq \min\left\{\mathrm{OPS}\{\check{J}\}, \mathrm{OPS}\{\hat{J}\}, \mathrm{OPS}\{J^{\diamond}\}\right\}.$$

In other words, we view J as a free by-product of any method for calculating $\hat{J}$, $\check{J}$, or $J^{\diamond}$. The same assumption will be made regarding the evaluation of second-derivative matrices or tensors.

4 Results and Discussion

In this final section we formulate rigorous bounds on the complexity of evaluating first and second derivatives of a vector function f in various ways. Similar bounds have been derived repeatedly in the automatic differentiation literature (see, e.g., [16], [17], and [4] as recent references).

4.1 First and Second Derivatives in the Forward Mode

Suppose the independent variables x are considered as linear functions

$$x(d) \equiv x + Sd$$

of the *differentiation parameter* vector $d \in \mathbb{R}^p$. We will refer to $S \in \mathbb{R}^{n \times p}$ as the seed matrix, which may vary from the $n \times n$ identity to a single-direction vector. Then all intermediates v_j have associated gradients and Hessians

$$\nabla_S v_j \in \mathbb{R}^p \quad \text{and} \quad \nabla_S^2 v_j \in \mathbb{R}^{p \times p} \quad .$$

Starting from $\nabla_S x_i = e_i^T S$ and $\nabla_S^2 x_i = 0$, one can propagate these derivatives forward by the chain rules

$$\nabla_S v_j \;=\; \sum_{i \to j} c_{ji} \nabla_S v_i \tag{4.1}$$

and

$$\nabla_S^2 v_j = \sum_{i \to j} \left[c_{ji} \nabla_S^2 v_i + \nabla_S v_i \sum_{k \to j} c_{jik} (\nabla_S v_k)^T \right] \quad . \tag{4.2}$$

At the end one obtains the *reduced* gradients

$$\nabla_S \hat{y}_k \;=\; \left. \frac{\partial f_k(x + Sd)}{\partial d} \right|_{d=0} \;=\; \nabla \hat{f}_k(x) S$$

and the two-sided projected Hessian

$$\nabla_S^2 y_k = \frac{\partial^2 \hat{f}_k(x + Sd)}{\partial d^2} = S^T \nabla^2 \hat{f}_k S \quad .$$

Now we obtain from the elemental complexity assumptions in the preceding section the following result.

Proposition 1 *The forward propagation of first and second derivatives with respect to p differentiation parameters can be achieved with*

$$\begin{aligned} \mathrm{OPS}\{f, \hat{J}\, S\} &\leq (2 + 3p)\, \mathrm{OPS}\{f\} \\ \mathrm{OPS}\{f, \hat{J}\, S, S^T \hat{f}_k'' S \text{ for } k = 1 \ldots \hat{m}\} &\leq [2 + 3p(p+1)]\, \mathrm{OPS}\{f\} \end{aligned}$$

operations. The corresponding memory requirement is bounded by

$$\begin{aligned} \mathrm{RAM}\{f, \hat{J}\, S\} &\leq (1 + p)\, \mathrm{RAM}\{f\} \\ \mathrm{RAM}\{f, \hat{J}\, S, S^T \hat{f}_i'' S \,\text{for}\, k = 1 \ldots \hat{m}\} &\leq (1 + p)(2 + p)/2\, \mathrm{RAM}\{f\} \quad . \end{aligned}$$

Proof. First let us note that the bounds on the randomly accessed memory reflect the fact that the p vector $\nabla_S \varphi_j$ and possibly also the symmetric $p \times p$ matrix $\nabla_S^2 \varphi_j$ must be kept for each scalar variable v_j. These gradient and Hessian objects can also be overwritten when v_j itself is overwritten by a new value.

Equation (4.1) can be interpreted as the multiplication of the row vector $\nabla \varphi_j$ by a matrix with p columns from the right. Hence the computational effort consists of exactly p inner products between $\nabla \varphi_j$ and a generally dense vector w. Thus we derive from (3.12), (3.14) and (4.1)

$$\begin{aligned} \mathrm{OPS}\{f, \hat{J}\, S\} &= \sum_{j=1}^{o} \Big[\mathrm{OPS}\{\varphi_j, \nabla \varphi_j\} + p\, \mathrm{OPS}\{(\nabla \varphi_j)^T w\} \Big] && (4.3) \\ &\leq \sum_{j=1}^{o} (2+3p)\, \mathrm{OPS}\{\varphi_j\} \;=\; (2+3p)\, \mathrm{OPS}\{f\} \quad . && (4.4) \end{aligned}$$

Similarly we see that (4.2) requires the computation of $p(p+1)/2$ inner products in $\nabla \varphi_j$ and exactly the same number of quadratic forms, $u^T \nabla^2 \varphi_j w$, so that by (4.2) and again (3.14)

$$\begin{aligned} &\mathrm{OPS}\{f, \hat{J}\, S, S^T \hat{f}_k'' S \;\; k \leq \hat{m}\} \\ &\quad = \sum_{j=1}^{o} \Big[\mathrm{OPS}\{\varphi_j, \nabla \varphi_j, \nabla^2 \varphi_j\} + \frac{1}{2} p(p+1) \Big(\mathrm{OPS}\{(\nabla \varphi_j)^T w\} + \mathrm{OPS}\{u^T \nabla^2 \varphi_j w\} \Big) \Big] \\ &\quad \leq \sum_{j=1}^{o} [2 + 3p(p+1)]\, \mathrm{OPS}\{\varphi_j\} \;=\; [2 + 3p(p+1)]\, \mathrm{OPS}\{f\} \quad , \end{aligned}$$

which completes the proof. ∎

A key advantage of the forward mode is that no extra sequentiallly accessed storage (SAM) is required and that sweeps of various order can be carried out simultaneously with the function evaluation by compilable code. If J and J' are dense, they can be calculated from one forward sweep with $p = n$ parameters. Alternatively, one can use slicing to obtain the Jacobian f' or the collection of Hessians f'' over several sweeps with S obtained from a partitioning of the identity matrix. For Jacobians the temporal complexity is strictly additive, but for Hessians the operations count may grow by a factor of two as a result of slicing [3]. In the constrained optimization case, one only needs projections of the objective and constraint Hessians to the range space of S anyway.

Even when f is neither function- nor argument-separable and J is dense, it is quite likely that the ratio $\hat{\gamma}$ defined in (2.8) is significantly smaller than n. Then the Jacobian J could theoretically be calculated more efficiently as a contraction of the vertically expanded Jacobian $\hat{F}'$. The difficulty with this approach is that one can, in general, not easily separate the calculations for (re)evaluating the various functions

$f^{(i)}$ by themselves. A similar effect is achieved if one performs the recursion (4.1) with $S = I$ and hence $p = n$, but with $\nabla_S v_j$ and $\nabla^2_S v_j$ stored and manipulated as sparse vectors and matrices, respectively. Since each $\nabla_S v_j$ has at most $|X(v_j)|$ nonzero entries, we obtain the following corollary.

Corollary 1
If the forward propagation of first and second derivatives is carried out using the sparsity of the gradients $\nabla_S v_j$ and Hessians $\nabla^2_S v_j$, then the operations count is

$$\begin{aligned} \mathrm{OPS}\{f, \hat{J}\} &\leq (2 + 3\hat{\gamma}\{f\})\,\mathrm{OPS}\{f\} \\ \mathrm{OPS}\{f, \hat{J}, \hat{f}''_k \text{ for } k = 1 \ldots \hat{m}\} &\leq [2 + 3\hat{\gamma}\{f\}(\rho(\hat{J}) + 1)]\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

and the corresponding RAM requirements are bounded as in Prop. 1 with $p = \rho(\hat{J})$.

Proof. The first inequality follows by definition of $\hat{\gamma}$ from (4.3) with p on the right-hand side replaced by $|X(v_j)|$. To prove the second inequality, we first note that the nonzeros of each Hessian $\nabla^2_S v_j$ form a nonzero square submatrix of order $|X(v_j)|$, so that on the right-hand side of (4.5) the factor p can also be replaced by $|X(v_j)|$. Hence we have instead

$$\begin{aligned} \mathrm{OPS}\{f, J\,S, S^T \hat{f}''_k S,\ k \leq \hat{m}\} &= \sum_{j=1}^{o} [2 + 3\,|X(v_j)|(|X(v_j)| + 1)]\,\mathrm{OPS}\{\varphi_j\} \\ &\leq \left[2 + 3\hat{\gamma}\{f\}(\rho(\hat{J}) + 1)\right]\,\mathrm{OPS}\{f\} \quad , \end{aligned}$$

where we have used Lemma 3 to bound the second factor $|X(v_j)|$. ∎

Since $\hat{\gamma} \leq \rho(\hat{J}) \leq \chi(\hat{J})$, it is clear that the sparse forward mode yields the lowest operations count followed by NR where we may choose $p = \rho(\hat{J})$ and CPR with $p \geq \chi(\hat{J})$. However, the NR and CPR methods may actually have a lower run-time, since on most computing platforms, vectors of fixed length p can be accessed and manipulated much faster than dynamically sparse vectors with a comparable number of nonzeros on average. In comparing the NR and CPR methods, we have so far ignored the fact that the former scheme requires the solution of $\hat{m}$ linear Vandermonde systems. According to [11] this adds

$$\sum_{k=1}^{\hat{m}} 2.5\hat{m}\,|X(\hat{y}_k)|^2 \quad \leq \quad 2.5\hat{m}\rho(\hat{J})^2$$

floating-point operations to complexity. As pointed out in [18], the conditioning of these linear systems can be improved by defining the Vandermonde matrix S using only $\chi(\hat{J})$ distinct real abscissas or defining them as complex roots of unity if the chromatic number is still too large. In the latter case, since all $\nabla_S v_j$ are complex, the arithmetic cost exactly doubles, because no complex multiplications or divisions are required.

4.2 First Derivatives in the Reverse Mode

In this subsection we first consider the complexity bounds for evaluating first derivatives in the reverse mode. Given the weight matrix $W^T \in \mathbb{R}^{q \times n}$, we may associate with each intermediate variable v_j the *adjoint* vector

$$\Delta^W v_j \equiv \frac{\partial W^T y}{\partial v_j} = W^T \frac{\partial y}{\partial v_j} \in \mathbb{R}^q \quad , \tag{4.5}$$

where all v_k with $v_j \not\rightsquigarrow v_k$ are held constant with respect to the differentiation. It is well known that the $\Delta^W v_j$ satisfy the backward recurrence

$$\Delta^W v_j = \sum_{v_j \prec\!\!\sim v_k} c_{kj} \, \Delta^W v_k \quad , \tag{4.6}$$

which can be executed only if the elementary partials $c_{kj} = \partial \phi_k / \partial x_j$ can be provided in reverse order, namely, for $k = o, o-1, \ldots, 1$. As we have mentioned in Subsection 1.3, it was shown in [12] that this program reversal can be performed at the computational costs (1.8) and (1.9) for some integer r, which determines a trade-off between temporal and spatial complexity. At the end of the reverse computation one obtains the adjoint vectors

$$\Delta^W \check{x}_i = W^T \frac{\partial \check{f}(\check{x})}{\partial \check{x}_i} = W^T \hat{J} e_i \quad .$$

The complexity of this reverse sweep is bounded by the following result

Proposition 2
The reverse differentiation of the q functions $W^T \check{f} : \mathbb{R}^{\check{n}} \to \mathbb{R}^q$ with respect to the expanded independents $\check{x}$ can be achieved at the costs

$$\begin{aligned} \mathrm{OPS}\{f, W^T \check{J}\} &\leq (r + 1 + 3q)\mathrm{OPS}\{f\} \\ \mathrm{RAM}\{f, W^T \check{J}\} &\leq (1 + q)\mathrm{RAM}\{f\} \\ \mathrm{SAM}\{f, W^T \check{J}\} &\leq c\,\mathrm{RAM}\{f\}\sqrt[r]{h\{f\}} \quad , \end{aligned}$$

where $h\{f\} \equiv \mathrm{OPS}\{f\}/\mathrm{RAM}\{f\}$ as before.

Proof. The RAM requirement follows from the need to store an adjoint q-vector $\Delta^W v_j$ for each variable that is live during the reverse sweep. Using the third inequality implicit in (3.14), we find that the backward propagation of the q vectors $\Delta^W v_j$ according to (4.6) requires also no more than $3q\,\mathrm{OPS}\{\varphi_k\}$ operations per intermediate node. Together with the cost for evaluating $\nabla \varphi_j$ and that for reversing the program as described in [12], this yields the operations count as well as the SAM requirement. ∎

If one wishes to obtain the whole Jacobian $\check{J}$ in order to compute $J = A\check{J}$, one may use the NR approach with $q = \rho(\check{J}^T)$ and W a Vandermonde matrix or the

CPR approach with $q \leq \chi(\check{J}^T)$ and W a $0-1$ matrix. Similarly, one can also employ a dynamically sparse reverse mode with $W = I$ for which q is effectively replaced by $\check{\gamma}\{f\}$ as defined (2.5). The advantages and disadvantages of these three alternatives are essentially the same as in the forward mode. Again the operations count is highest for CPR and lowest for the dynamically sparse procedure, which does, however, involve more overhead. The NR approach may again suffer from poor conditioning unless the matrix W is chosen carefully, possibly using a coloring or complex roots of unity.

4.3 Combinations of Forward and Reverse Sweeps

In Proposition 1 we have shown that the full second-derivative tensor, $\hat{f}''$, and thus its contraction, f'', can be obtained at a complexity that grows quadratically with $p = \rho(\hat{J})$ or $p = \chi(\hat{J})$, depending on whether one uses the NR or CPR approach. It is interesting to note that, if one were to use CPR in the forward mode to evaluate the gradient of a scalar function f and then to use directional derivatives of this vector function ∇f in an indirect substitution method as described and analyzed in [6], then by (2.3) the complexity would be proportional to $\mathrm{OPS}\{f\}$ times

$$\chi(G)\chi_0(G) \ \geq \chi^2(G) \ = \ \chi^2(\hat{J}),$$

where G is the incidence graph of $\nabla^2 f$, which coincides by Lemma 1 with the column-intersection graph of the expanded Jacobian $\hat{J}$. Consequently, even indirect substitution on a gradient that is evaluated in the forward mode is likely to be less efficient than the calculation of the Hessian by differentiating $\hat{f}$ twice in the forward mode. Even lower complexities can be achieved if the forward and reverse modes are combined (see, for example, [5]). By combining Propositions 1 and 2 we obtain our final result.

Corollary 2
With $f : \mathbb{R}^n \to \mathbb{R}^m$ as before, $u \in \mathbb{R}^m$ a vector of Lagrange multipliers, and $S \in \mathbb{R}^{n \times p}$ the one-sided projection

$$\nabla^2 L(x)\, S \ = \ \sum_{i=1}^{\hat{m}} \hat{u}_i \nabla^2 \hat{f}_i S$$

can be calculated as the complexity

$$\begin{aligned}
\mathrm{OPS}\{\nabla^2 L(x)\, S\} &\leq (4+r)(2+3p)\mathrm{OPS}\{f\} \\
\mathrm{RAM}\{\nabla^2 L(x)\, S\} &\leq 2(1+p)\mathrm{RAM}\{f\} \\
\mathrm{SAM}\{\nabla^2 L(x)\, S\} &\leq c\,(1+p)\,\mathrm{RAM}\{f\}\sqrt[r]{h\{f\}} \quad ,
\end{aligned}$$

where c may be larger by a factor of 3 compared with Proposition 2.

Proof. This result can be achieved by first evaluating the vector function $g(x) \equiv J(x)S : \mathbb{R}^n \to \mathbb{R}^p$. According to Proposition 1 the forward mode yields these values with an operations count no greater than $(2+3p)\mathrm{OPS}\{f\}$ and a RAM requirement no greater than $(1+p)\mathrm{RAM}\{f\}$. Hence the ratio between the operations count and the RAM requirement grows by a factor less than 3. Applying Proposition 2 to this calculation with $W = u$ and $q = 1$, we pick up another factor of $r+4$ for the operations count, a factor of 2 for the RAM requirement, and a factor less than $(1+p)\sqrt[r]{3}$ for the SAM requirement. ∎

Since the number n of independents does not occur in the bounds of Corollary 2, we see that the complexity of the one-sided projected Hessian of the Lagrangian depends only on the degrees of freedom $p = n - m$ in a constrained optimization problem. It also appears that the cheapest way of obtaining the two-sided projection is to multiply the one-sided projection by S. Further cost reductions might be achievable if one exploits sparsity of $\nabla^2 LS$. The columns of S must span the null-space of J and are often defined on the basis of an LU or QR factorization of J. It would appear that these choices may be far from optimal regarding the sparsity of $\nabla^2 LS$, since they tend to introduce dense rows into S and consequently $\nabla^2 LS$. This question deserves further investigation.

5 Summary and Conclusion

In this paper we have shown how function separability and the new concept of argument separability can be exploited to yield first and second derivatives by the forward or reverse mode of computational differentiation with surprisingly low complexity.

The ideal case of function separability is that of a partially separable objective function f, whose gradient and Hessian can be obtained in the forward mode at a complexity of $\rho(\hat{J})$ and $\rho^2(\hat{J})$, respectively. Here $\rho(\hat{J})$ represents the maximal number of variables that are truly intertwined in a nonlinear fashion during the evaluation of f. The ideal case of argument separability is that of a vector function f, whose components f_k are evaluated completely separately from each other. Then one may apply the reverse mode to the horizontal expansion $\check{f}$ and obtain the full Jacobian $J = \check{J}A$ at no more than five times the cost of evaluating f itself. It is likely that substantial savings can be realized in mixed cases, but the implementation in a computational differentiation tool is a nontrivial task.

Rather than just considering additive decompositions with 0-1 matrices A and B, one can generalize the separability concepts, so that arbitrary linear pre-factors B and post-factors A are removed from the given vector functions to facilitate more efficient differentiation on the remaining nonlinear part.

Acknowledgments

The author had the benefit of extended discussions with Jorge Moré and Brett Averick, and he is greatly indebted to Christian Bischof and Paul Plassmann for their careful reading of the first draft.

References

[1] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle, and A. Griewank, *Computing large sparse Jacobian matrices using automatic differentiation*, Preprint MCS-P348-0193, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1993.

[2] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland, *ADIFOR: Generating derivative codes from Fortran programs*, Scientific Programming, 1 (1992), pp. 1–29.

[3] C. Bischof, G. Corliss, and A. Griewank, *Computing second- and higher-order derivatives through univariate Taylor series*, Preprint MCS-P296-0392, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1992.

[4] B. Christianson, *Automatic Hessians by reverse accumulation*, IMA J. of Numerical Analysis, 12 (1992) pp. 135–150 .

[5] B. Christianson, *Reverse accumulation and accurate rounding error estimates for Taylor series coefficients*, Optimization Methods and Software, 1 (1992), pp. 81–94.

[6] T. F. Coleman and Jin-Yi Cai, *The cyclic coloring problem and estimation of sparse Hessain matrices*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 221-235.

[7] T. F. Coleman, B. S. Garbow, and J. J. Moré, *Fortran subroutines for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 346–347.

[8] ——, *Software for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329–345.

[9] 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.

[10] A. R. Curtis, M. J. D. Powell, and J. K. Reid, *On the estimation of sparse Jacobian matrices*, J. Inst. Math. Appl., 13 (1974), pp. 117–119.

[11] G. H. Golub, and C. F. Van Loan, *Matrix Computations*, second edition, The Johns Hopkins University Press, Baltimore (1988)

[12] A. Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optimization Methods and Software, 1 (1992), pp. 35–54.

[13] A. Griewank and G. F. Corliss, eds., *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Philadelphia, 1991.

[14] A. Griewank and S. Reese, *On the calculation of Jacobian matrices by the Markowitz rule*, in **Automatic Differentiation of Algorithms: Theory, Implementation,** and Application (A. Griewank and G. Corliss, eds.), SIAM, Philadelphia, 1991, pp. 126–135.

[15] A. Griewank and Ph.L. Toint *On the unconstrained optimization of partially separable objective functions*, in **Nonlinear Optimization 1981** (M. J. D. Powell, ed.), Academic Press, London, 1981, pp. 301–312.

[16] Masao Iri, *History of automatic differentiation and rounding estimation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. Corliss, eds., SIAM, Philadelphia, 1991, pp. 1–16 .

[17] R. D. Neidinger, *An efficient method for the numerical evaluation of partial derivatives of arbitrary order*, ACM Trans. Math. Software, 18(1992), pp. 159–173 .

[18] G. N. Newsam and J. D. Ramsdell, *Estimation of sparse Jacobian matrices*, SIAM J. Alg. Disc. Meth., 4 (1983), pp. 404–417 .

[19] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, **Lecture Notes in Computer Science,** Vol. 120, Springer-Verlag, Berlin, 1981.

[20] Trond Steihaug and A. K. M. Shahadat Hossain, *Graph coloring and the estimation of sparse Jacobian matrices using row and column partitioning*, Report 72, Department of Informatics, University of Bergen, 1992.

[21] Stephen A. Vavasis, *Nonlinear Optimization, Complexity Issues,* Oxford University Press, Oxford, 1991.