The Solution of Large-Scale Optimization Problems
(using MATLAB)

Thomas F. Coleman[1]
Computer Science
Cornell University
Ithaca , New York

April 14, 1999

[1]coleman@tc.cornell.edu http://www.tc.cornell.edu/~ coleman

Contents

Chapter 1

Computing Derivatives

A large percentage of the computing time to solve optimization problems is consumed by derivative computation. An optimization "client" can spend a distressing amount of time implementing and testing codes for computing and approximating derivatives. For large-scale problems this effort, human and machine, can become overwhelming. Fortunately there are ways to exploit structure in large-scale problems, and new tools to help in this process. Moreover, if the single purpose of computing a Jacobian or Hessian matrix is to solve for the Newton step, then it is sometimes possible to do this directly (or semi-directly) thus avoiding the complete, and expensive, calculation of a Jacobian or Hessian matrix. In this section we will discuss the computation/approximation of derivatives for large-scale optimization and show how the structure of large-scale optimization problems can be used to greatly enhance efficiency.

We divide this chapter into two main sections. The first section, §1.1, is concerned with the determination of *sparse* Jacobian and Hessian matrices. Within this context we consider the use of two approaches: sparse finite differencing and automatic differentiation. Section ?? is about *structured* problems: we develop a very general view of "structure" and show how structure can be used (with particular emphasis on automatic differentiation).

1.1 Sparse Jacobian and Hessian Matrices.

Many large-scale problems exhibit sparsity in their Jacobian (or Hessian matrices). Recognition of this fact can avoid wasting time calculating zero quantities. The techniques we describe for the determination of sparse Jacobians and Hessians by finite differences are quite similar to the methods, described subsequently, for automatic differentiation (AD). The finite differencing approach has the advantage that no additional software is required and optimized hand-coding can be used. However, the sparse AD approach is often a better choice for one of several reasons:

(i) Differencing requires the choice of a differencing parameter, a tricky business. Moreover, sparse AD has greater accuracy due to the lack of truncation error.

(ii) In the unsymmetric case sparse differencing techniques are expensive in the presence of a dense row; sparse AD techniques, such as the bi-coloring method, §1.1.3, are much less sensitive to dense rows.

(iii) The application of sparse finite differencing requires knowledge of the structure of the Jacobian (or Hessian); however, sparse AD software packages can pre-compute the sparsity pattern automatically.

1.1.1 Sparse Finite-differences for Jacobian matrices.

Consider first the unsymmetric case: approximate the Jacobian of $F : \Re^n \to \Re^m$. Partition the columns of J into a set of groups G_C, where the columns in each group $C \in G_C$ are *structurally orthogonal*[1]. The number of groups in G_C is denoted by $p = |G_C|$. Each group $C \in G_C$ defines a column v of a matrix V: $v_i \neq 0$ if and only if column i is in group C. It is clear that the nonzeros of J can be immediately "identified" from the product $\bar{J} \approx JV$. Graph coloring techniques applied to $G_I(J)$, the column intersection graph (see Figures 1.1, 1.2) of J, can be used to try and produce a partition G_C with low cardinality. This, in turn, induces a thin matrix V, i.e., $V \in \Re^{n \times p}$, where p is "small".

$$
\mathrm{J} = \begin{pmatrix} x & 0 & 0 & 0 & 0 & x & x & x \\ 0 & x & 0 & 0 & x & 0 & x & x \\ 0 & 0 & x & 0 & x & x & 0 & x \\ 0 & 0 & 0 & x & x & x & x & 0 \end{pmatrix}
$$

Figure 1.1: A Sparse Jacobian Structure

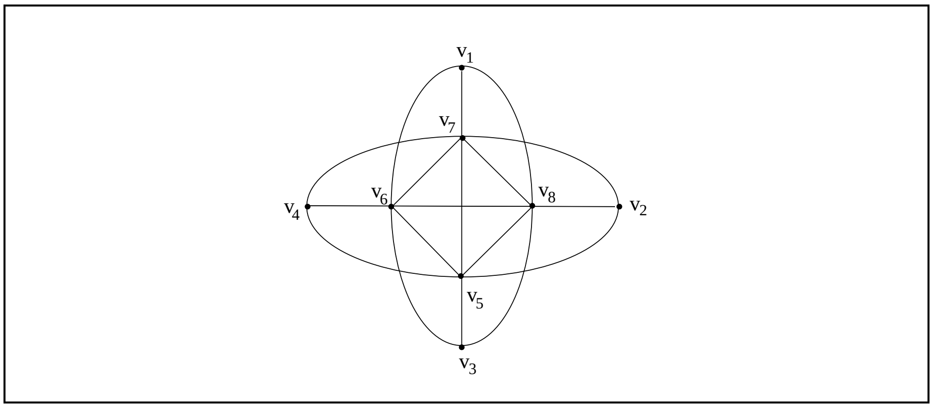

Figure 1.2: Column Intersection Graph of J

The key point is this: computation of each column of $\bar{J}$ requires one evaluation of F. Therefore, $\bar{J}$ can be obtained in exactly p function evaluations (in addition to the evaluation of F at x). Finite-difference approximations to the nonzero elements of J can be resolved directly from $\bar{J}$.

For example, let the structure of the Jacobian matrix be indicated in Figure 1.1. Clearly the corresponding intersection graph, Figure 1.2, requires 4 colors: each of the pairs of vertices $(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_4, v_8)$ can be assigned a separate color, c_1, c_2, c_3, c_4. Each color, in turn, corresponds to a differencing direction; in this case each direction has two nonzero components - the nonzero components of direction d_i correspond to the the columns in the group corresponding to color c_i. So, for example, $d_2^T = (0, x, 0, 0, 0, x, 0, 0)$, and $V = (d_1, d_2, d_3, d_4)$.

[1]Two nonzero n-vectors v, w are *structurally orthogonal* if $v_i * w_i = 0,\ i = 1 : n$.

In general, a p-coloring of $G_I(J)$ induces a partition of the columns of J into groups, $C_1, ..., C_p$ where the columns in group C_k correspond to the vertices assigned color c_k. If we associate a steplength h_j with column j then differencing direction d_k, corresponding to group C_k, is defined

$$d_k = \sum_{j \in C_k} h_j e_j$$

where e_j is the j^{th} column of the identity matrix. The finite-difference approximations to the nonzero elements of J are easy to recover. For example if column $j \in C_k$ and $y_k = F(x+d_k) - F(x)$ has been computed, then for each nonzero J_{ij}:

$$J_{ij} \approx \frac{(y_k)_i}{h_j}. \tag{1.1}$$

Heuristic graph coloring methods can be used to obtain a p-coloring of $G_I(J)$ with a small value of p (usually). It is important to realize that for expensive functions a small value of p is very much desired since the cost of approximating J by this scheme is proportional to $p \cdot \omega(F)$ where $\omega(F)$ is the cost of evaluating F. Note that

$$\eta^* \leq \chi(G_I(J)) \leq p$$

where $\chi(G_I(J))$ is the chromatic number[2] of the intersection graph $G_I(J)$ and η^* is the maximum number of nonzeros in any row of J.

This approach can be very effective. For more background see [5, 7, 8, 9, 10, 14]. However, there are three potential weaknesses. First, optimal graph coloring is NP-hard and a graph coloring heuristic allows for the possibility of using many more colors (groups) than necessary, i.e., $p >> \chi(G_I(J))$. Second, the optimal coloring may be much larger than the lower bound, i.e., $\chi(G_I(J)) >> \eta^*$. Finally, the lower bound η^* may be unacceptably large, i.e., J has at least one very dense row. Of the three concerns the third is the most troublesome from a practical point of view. The first two difficulties, while theoretically possible, do not seem to arise very often in practise. The third problem is more common and finite differencing of F does not offer a solution. One possibility is to try and design the subroutine that evaluates F so that components corresponding to the denser rows can be evaluated separately. In this case a secant approximation can be used to approximate the dense rows. A second solution, quite general, is to use the bi-coloring automatic differentiation scheme discussed in §1.1.3

The approach we have described is a *direct* method. In other words the nonzeroes of the Jacobian matrix can be ordered (call the resulting vector u), and the components of the differencing vectors d_i can be arranged into a *diagonal* matrix D so that $Du = y$, where y is a re-arrangement of the componets of the difference vectors y_i. (See §1.2, Question 1.). Interestingly, this idea can be readily generalized: choose vectors d_i such that a rearrangement yields $Du = y$, where D is a *block* diagonal matrix. It turns out that this can always be done with maximum block size η^* and using η^* vectors d_i. Therefore the lower bound for direct methods, η^*, can always be achieved with a more general approach (see §1.2, Question 2.). Of course the down side is that a matrix factorization will be required to recover J. And there are accuracy concerns, e.g., [4].

[2]The *chromatic number* of a graph is the minimum number of colors required for a valid coloring of the vertices.

1.1.2 Sparse Finite-differences for Hessian matrices.

Symmetry can be used to further reduce the number of differences required. For example, suppose the (symmetric) Hessian matrix H of a scalar value function f has an arrowhead structure:

$$H = \begin{pmatrix} x & x & x & x & x \\ x & x & & & \\ x & & x & & \\ x & & & x & \\ x & & & & x \end{pmatrix}. \tag{1.2}$$

It is clear, due to the dense first row, that a direct partitioning approach that ignores symmetry requires n groups. However, only 2 groups are required if symmetry is used; i.e., $d_1^T = (x, 0, 0, 0, 0)$, $d_1^T = (0, x, x, x, x)$. Differencing the gradient in direction d_1 yields an approximation to the first column of H and, by symmetry, the first row. The remaining diagonal elements are obtained from the gradient difference along d_2.

A general direct method that exploits symmetry can be based on the idea of a path coloring [5], [9] Let $G_{adj}(H) = (V, E)$ be the adjacency graph of H (a sparse symmetric matrix). A *path p-coloring* of G_{adj} is an assignment of colors to vertices, $\phi \rightarrow \{1, ..., p\}$ such that ϕ is a p-coloring with the additional stipulation that every path in G_{adj} of length four vertices uses at least three colors. In Figure 1.3, $v_h - v_i - v_j - v_k$ is an arbitrary path of length four vertices in G_{adj}. The (r, s) color assignment illustrated on the left is invalid since a path coloring demands that at least 3 colors be used on very path of length 4 (vertices); the assignment illustrated in the right graph in Figure 1.3 is valid.

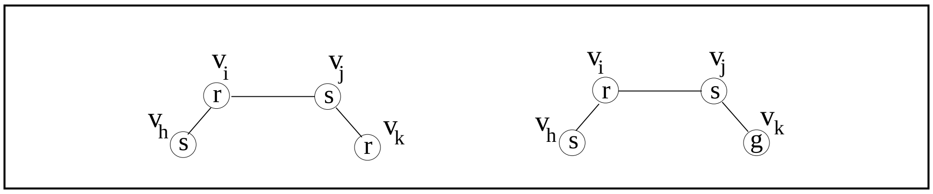

Figure 1.3: A Path Coloring

Coleman and Moré [9]. provide the following characterization of direct determination of a symmetric matrix H.

Theorem 1.1.1 *The mapping ϕ is a path p-coloring if and only if ϕ induces a partition of the columns of H consistent with direct determination.*

Finding an optimal path p-coloring is an NP-hard problem [9]. However, effective heuristics exist based on general graph coloring procedures [9], [18].

Substitution methods and cycling coloring. Symmetric direct methods do not exploit symmetry to the fullest. For example it is easy to show [9] that a symmetric direct method (i.e., a path p-coloring) does not improve on a direct unsymmetric method when applied to a symmetric band matrix. Therefore, it is natural to think that it is possible to better exploit symmetry in some other way. Indeed this is the case if we relax the restriction that all elements of H be determined directly.

A partition of columns of a symmetric matrix induces a *substitution method* if there is an ordering of the matrix nonzeros (unknowns) so that they can be resolved, in that order, using symmetry and previously solved elements. For example, it is easy to verify that a symmetric tridiagonal matrix can be detemined using two gradient differences with substitutions. This can be achieved by forming one group of even-numbered columns and one group of odd-numbered columns. (Note that three groups are required to determine a symmetric tridiagonal H directly.)

Coleman and Cai [5] provide a graph-theoretic characterization of substitution methods for symmetric matrices based on the notion of a *cyclic coloring*. A mapping $\phi : V \rightarrow \{1, 2, ...p\}$ is a *cyclic p-coloring* if ϕ is a p-coloring and ϕ uses at least three colors in every cycle of $G_{adj}(H)$. Figure 1.4 illustrates a cyclic 3-coloring.

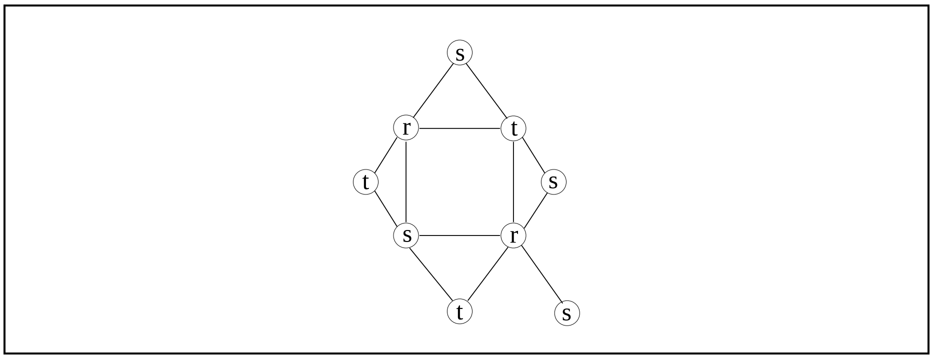

Figure 1.4: A Cyclic Coloring

Theorem 1.1.2 *Let H be a symmetric matrix with a zero-free diagonal. The mapping ϕ induces a substitution method if and only if ϕ is a cyclic coloring of $G_{adj}(H)$.*

Why does a cyclic coloring do the job? Consider any subgraph in a cyclic-colored graph $G_{adj}(H)$ induced by a pair of colors. Clearly this subgraph must be a forest, i.e., a collection of trees. Example 1.5 illustrates the subgraph of the cyclic-colored graph in Figure 1.4 induced by colors

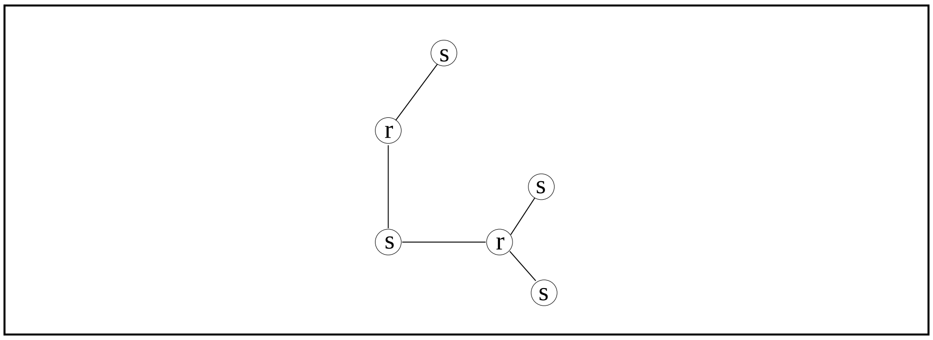

Figure 1.5: The Tree Induced By Colors r,s

(r, s). It is now easy to show that the nonzeros (unknowns) of H corresponding to the edges in the induced tree can be resolved, by substitution, working inward along the tree branches from the leaves [5].

Effective cyclic graph coloring heuristics (optimal cyclic coloring is NP-hard) have been developed by modifying general graph coloring heuristics [5],[9], [18]. Cyclic coloring heuristic generally use fewer colors than path colorings; hence, fewer gradient finite difference are required for substitution methods versus direct methods. However, there is greater danger of roundoff error growth in a substitution method, especially when the stepsizes associated with the different columns vary significantly [18],[5]. Therefore, there can be a trade-off between accuracy and cost.

We have discussed three basic partition-based approaches to the determination of a sparse Hessian matrix by finite differencing of the gradient function; direct (unsymmetric) determination based on coloring the intersection graph defined by the columns of H, direct symmetric determination based on a path-coloring of the adjacency graph of H, and a symmetric substitution method defined by a cyclic coloring of the adjacency graph of H. Let $\chi(G_I(H))$ be the optimal coloring of the intersection graph of H; let $\chi_\pi(G_{adj}(H))$ be the optimal path coloring, and let $\chi_o(G_{adj}(H))$ be the optimal cyclic coloring of $G_{adj}(H)$. Then the following string of inequalities is easy to establish

$$\chi(G_{adj}(H)) \leq \chi_o(G_{adj}(H)) \leq \chi_\pi(G_{adj}(H)) \leq \chi(G_I(H)) = \chi(G_{adj}(H^2)). \tag{1.3}$$

1.1.3 Automatic Differentiation for Sparse Jacobian matrices.

Automatic differentiation is a chain rule based technique for evaluating derivatives of functions defined by high-level computer programs. In this section we briefly review the basics of automatic differentiation: see [15, 16] for more detail.

A program that computes the function $z = F(x), F : \Re^n \rightarrow \Re^m$, can be viewed as a sequence of scalar assignments $v_i = \psi < v_j >_{j \rightarrow i}$, where the vector v can be thought of as set of ordered variables such that $v_i, i \rightarrow j$, is computed before v_j using the set of variables $\{v_k | k \rightarrow i\}$. Here ψ_j represent elementary functions, which can be arithmetic operations and/or univariate transcedental functions.

Ordering the variables as above, we can partition variables v_j into three vectors.

$$x \equiv (v_1, v_2, \dots, v_n) \text{ (independent)}$$
$$y \equiv (v_{n+1}, v_{n+2}, \dots, v_{n+p}) \text{ (intermediate)}$$
$$z \equiv (v_{n+p+1}, v_{n+p+2}, \dots, v_{n+p+m}) \text{ (dependent)}$$

In general, the number of intermediate variables is much larger than the dimensions of the problem, i.e., $p \gg m, n$.

Assume that all these elementary functions ψ_i are well defined and have continuous elementary partials $c_{ij} = \frac{\partial \psi_i}{\partial v_j}, j < i$. Assuming with out loss of generality that the dependent variables y do not themselves occur as arguments of elementary functions, we can combine the partials c_{ij} into the $(p + m) \times (n + p)$ matrix

$$C = \left(\ c_{n+i,j} \ \right)_{1 \leq i \leq p+m}^{1 \leq j \leq n+p} \tag{1.4}$$

Unless elementary functions with more than two arguments are included in the library, each row of C contains either one or two non zero entries. We define a number

$$q \equiv nnz(C) \leq 2p \tag{1.5}$$

Also, since work involved in an elementary function is proportional to number of arguments, it follows that

$$\omega(F) = \sum_{i=1}^{p+m} \omega(\psi_{n+i}) \sim q \tag{1.6}$$

where $\sim$ indicates proportionality. Because of the ordering relation the square matrix C is upper trapezoidal with $n-1$ nontrivial superdiagonals. Thus C can be partitioned as

$$C = \left(\begin{array}{c|c} A & L \\ \hline B & M \end{array}\right) \tag{1.7}$$

where

$$A \in \Re^{p\times n}, B \in \Re^{m\times n}, L \in \Re^{p\times p}, M \in \Re^{m\times p}, \tag{1.8}$$

and L is strictly lower triangular. Application of the chain rule yields:

$$\begin{pmatrix} A & L-I \\ B & M \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 0 \\ -z \end{pmatrix}. \tag{1.9}$$

If we eliminate the intermediate vector Δy from above (1.9), we get an expression(the Schur complement) for the Jacobian:

$$J \equiv B - M(L-I)^{-1}A = B + M(I-L)^{-1}A \tag{1.10}$$

Since $I - L$ is a unit lower triangular matrix, the calculation of the matrix products $\tilde{A} \equiv (I-L)^{-1}A$ and $\tilde{M} \equiv M(I-L)^{-1}$ leads to two natural ways to compute J:

$$J = B + M\tilde{A} \quad \textit{or} \quad J = B + \tilde{M}A. \tag{1.11}$$

The alternative expressions for J given in (1.11) define the two basic *modes* of automatic differentiation, *forward* and *reverse*.

The *forward mode* corresponds to computing the rows of $\tilde{A}$, one by one, as the corresponding rows of $[A, L-I]$ are obtained from successive evaluation of elementary functions. Since this amounts to solutions of n linear systems with lower-triangular matrix $[I-L]$, followed by multiplication of dense columns of $\tilde{A}$ by M, the total computational effort is roughly $n \cdot q$ or $n \cdot \omega(F)$.

The *reverse mode* corresponds to computing $\tilde{M}$ as solution to linear system $(I-L)^T\tilde{M}^T = M^T$. This back-substitution process can begin only after all elementary functions and their partial derivatives have been evaluated. Since this amounts to the solution of m linear systems with lower triangular matrix $[I-L]$, followed by multiplication of dense rows of $\tilde{M}$ by A, total computational effort is roughly $m \cdot q$ or $m \cdot \omega(F)$.

We are interested in computing products of the form JV and W^TJ. Product JV can be computed:

$$JV \quad = \quad BV + M[(I-L)^{-1}(AV)]$$

which can clearly be done in time proportional to $t_V \cdot \omega(F)$ when $V \in \Re^{n\times t_V}$. Analagously, product W^TJ can be computed:

$$W^TJ \quad = \quad W^TB + [(W^TM)(I-L)^{-1}]A$$

which can be done in time proportional to $t_W \cdot \omega(F)$ assuming $W \in \Re^{m \times t_W}$.

An important subcase worthy of special attention is when F is a scalar function, i.e., $m = 1$. In this case the Jacobian matrix corresponds to the transpose of the gradient of F and is a single row vector. Note that the complexity arguments applied to this case imply that the reverse mode of AD yields the gradient in time proportional to $\omega(F)$ whereas the forward mode costs $n \cdot \omega(F)$. The efficiency of the forward mode evaluation of the gradient can be dramatically increased - i.e., the dependence on n is removed - if F has structure that can be exploited [3].

Partition problems and graph theory

Our basic task is to efficiently determine thin matrices V, W so that the nonzero elements of J can be readily extracted from the information $(W^T J, JV)$. The pair of matrices $(W^T J, JV)$ is obtained from the application of both modes of automatic differentiation: matrix $W^T J$ is computed by the reverse mode, the forward mode determines JV. The purpose of this section is to more rigorously formulate the question of determining suitable matrices V, W, first in the language of "partitions", and then using graph theoretic concepts.

We begin with an example illustrating the usefulness of simultaneously applying both modes of AD, forward and reverse. Consider the following n-by-n Jacobian, symmetric in structure but not in value:

$$J = \begin{pmatrix} \square & \triangle & \triangle & \triangle & \triangle \\ \square & \diamond & & & \\ \square & & \diamond & & \\ \square & & & \diamond & \\ \square & & & & \diamond \end{pmatrix}. \tag{1.12}$$

It is clear that a partition of columns consistent with the direct determination of J requires n groups. This is because a "consistent column partition" requires that each group contain columns that are structurally orthogonal and the presence of a dense row implies each group consists of exactly one column. Therefore, if matrix V corresponds to a "consistent column partition" then V has n columns and the work to evaluate JV by the forward mode of AD is proportional to $n \cdot \omega(F)$. By a similar argument, and the fact that a column of J is dense, a "consistent row partition" requires n groups. Therefore, if matrix W corresponds to a "consistent row partition" then W has n rows and the work to evaluate $W^T J$ by the reverse mode of AD is proportional to $n \cdot \omega(F)$.

A *bi-partition* of a matrix A is a pair (G_R, G_C) where G_R is a row partition of a *subset* of the rows of A , G_C is a column partition of a *subset* of the columns of A.

In this example the use of a bi-partition dramatically decreases the amount of work required to determine J. Specifically, the total amount of work required is proportional to $3 \cdot \omega(F)$. To see this define $V = (e_1, e_2 + e_3 + e_4 + e_5)$; $W = (e_1)$, where we follow the usual convention of representing the i^{th} column of the identity matrix with e_i. Clearly elements $\square, \diamond$ are directly determined from the product JV; elements $\triangle$ are directly determined from the product $W^T J$.

The basic idea is to partition the rows into a set of groups G_R and the columns into a set of groups G_C, with $|G_R| + |G_C|$ as small as possible, such that every nonzero element of J can be directly determined from *either* a group in G_R or a group in G_C.

A bi-partition (G_R, G_C) of a matrix A is *consistent with direct determination* if for every nonzero a_{ij} of A, either column j is in a group of G_C which has no other column having a nonzero in row i, or row i is in a group of G_R which has no other rows having a nonzero in column j.

Clearly, given a bi-partition (G_R, G_C) *consistent with direct determination*, we can trivially construct matrices $W \in \Re^{m \times |G_R|}, V \in \Re^{n \times |G_C|}$ such that A can be directly determined from $(W^T A, AV)$.

If we relax the restriction that each nonzero element of J be determined directly then it is possible that the work required to evaluate the nonzeroes of J can be further reduced. For example we could allow for a "substitution" process when recovering the nonzeroes of J from the pair (W^TJ, JV). Figures 1.6, 1.7 illustrate that a substitution method can win over direct determination: Figure 1.6 corresponds to direct determination, Figure 1.7 corresponds to determination using substitution.

$$\begin{pmatrix}
\square_{11} & \triangle_{12} & \triangle_{13} & & & & & & \\
\square_{21} & & & & & & & & \\
\square_{31} & & & & & & & & \\
\square_{41} & & & \square_{44} & \triangle_{45} & \triangle_{46} & & & \\
 & & & \square_{54} & & & & & \\
 & & & \square_{64} & & & & & \\
 & & & \square_{74} & & & \square_{77} & \triangle_{78} & \triangle_{79} \\
 & & & & & & \square_{87} & & \\
 & & & & & & \square_{97} & & \\
 & & & & & & \square_{10,7} & &
\end{pmatrix}$$

Figure 1.6: Optimal partition for direct method

$$\begin{pmatrix}
\square_{11} & \triangle_{12} & \triangle_{13} & & & & & & \\
\square_{21} & & & & & & & & \\
\square_{31} & & & & & & & & \\
\square_{41} & & & \triangle_{44} & \triangle_{45} & \triangle_{46} & & & \\
 & & & \square_{54} & & & & & \\
 & & & \square_{64} & & & & & \\
 & & & \square_{74} & & & \triangle_{77} & \triangle_{78} & \triangle_{79} \\
 & & & & & & \square_{87} & & \\
 & & & & & & \square_{97} & & \\
 & & & & & & \square_{10,7} & &
\end{pmatrix}$$

Figure 1.7: Optimal partition for substitution method

In both cases elements labelled $\square$ are computed from the column grouping, i.e., calculated using the product JV; elements labelled $\triangle$ are calculated form the row groupings, i.e., calculated using the product W^TJ. The matrix in Figure 1.6 indicates that we can choose G_C with $|G_C| = 2$ and G_R with $|G_R| = 1$ and determine all elements directly. That is, choose $V = (e_1 + e_7, e_4)$; choose $W = (e_1 + e_4 + e_7)$. Therefore in this case the work to compute J satisfies $\omega(J) \sim 3 \cdot \omega(F)$. Note that some elements can be determined twice, e.g., J_{11}.

However, the matrix in Figure 1.7 shows how to obtain the nonzeroes of J, using substitution, in work proportional to $2 \cdot \omega(F)$. Let $V = W = (e_1 + e_4 + e_7)$. Let p_1 be the (forward) computed vector $p_1 = JV$; let p_2^T be the (reverse) computed row vector $p_2^T = W^T J$. Then

$$p_1 = \begin{pmatrix} \Box_{11} \\ \Box_{21} \\ \Box_{31} \\ \Box_{41} + \triangle_{44} \\ \Box_{54} \\ \Box_{64} \\ \Box_{74} + \triangle_{77} \\ \Box_{87} \\ \Box_{97} \\ \Box_{10,7} \end{pmatrix}, \quad p_2^T = \begin{pmatrix} \triangle_{11} + \Box_{41}, \triangle_{12}, \triangle_{13}, \triangle_{44} + \Box_{74}, \triangle_{45}, \triangle_{46}, \triangle_{77}, \triangle_{78}, \triangle_{79} \end{pmatrix}.$$

Most of the nonzero elements are determined directly (no conflict). The remaining elements can be resolved,

$$\Box_{74} = p_1(7) - p_2(7); \qquad \triangle_{44} = p_2(4) - \Box_{74}; \qquad \Box_{41} = p_1(4) - \triangle_{44}.$$

It is easy to extend this example so that the difference between the number of groups needed, between substitution and direct determination, increases with the dimension of the matrix. For example, a block generalization is illustrated in Figure 1.8: if we assume $l > 2w$ it is straightforward to verify that in the optimal partition the number of groups needed for direct determination will be $3w$ and determination by substitution requires $2w$ groups.

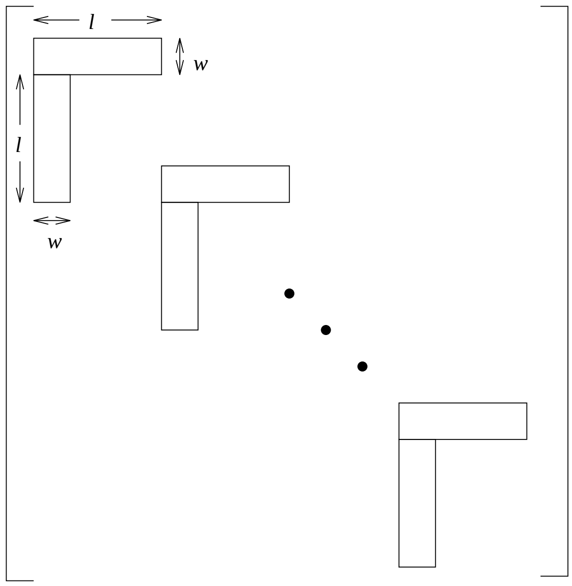

Figure 1.8: Block example

A bi-partition (G_R, G_C) of a matrix A is *consistent with determination by substitution*, if there exists an ordering π on elements a_{ij}, such that for every nonzero a_{ij} of A, either column j is in a group where all nonzeros in row i, from other columns in the group, are ordered lower than a_{ij}, or row i is in a group where all the nonzeros in column j, from other rows in the group, are ordered lower than a_{ij}.

In the usual way we can construct a matrix V from the column grouping G_C and a matrix W from the row grouping G_R: for example, to construct the columns of V associate with each group in G_C a boolean vector, with unit entries indicating membership of the corresponding columns. We can now state our main problem(s) more precisely:

The bi-partition problem (direct) : Given a matrix A, obtain a bi-partition (G_R, G_C) consistent with direct determination, such that total number of groups, $|G_R| + |G_C|$, is minimized.

The bi-partition problem (substitution) : Given a matrix A, obtain a bi-partition (G_R, G_C) consistent with determination by substitution, such that total number of groups, $|G_R| + |G_C|$, is minimized.

Bi-coloring

The two combinatorial problems we face, corresponding to direct determination and determination by substitution, can both be approached in the following way. First, permute and partition the structure of J: $\tilde{J} = P \cdot J \cdot Q = [J_C|J_R]$, as indicated in Figure 1.9. The construction of this partition is crucial; however, we postpone that discussion until after we illustrate its' utility. Assume $P = Q = I$ and $J = [J_C|J_R]$.

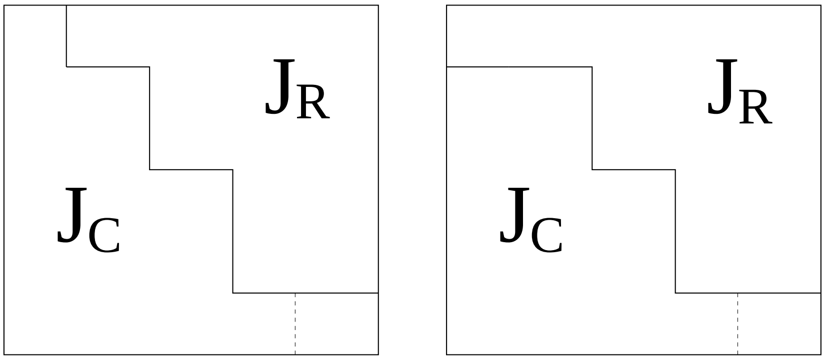

Figure 1.9: Possible partitions of the matrix $\tilde{J} = P \cdot J \cdot Q$

Second, define appropriate intersection graphs $\mathcal{G}_C^I, \mathcal{G}_R^I$ based on the partition $[J_C|J_R]$; a coloring of $\mathcal{G}_C^I$ yields a partition of a subset of the columns, G_C, which defines matrix V. Matrix W is defined by a partition of a subset of rows, G_R, which is given by a coloring of $\mathcal{G}_R^I$. We call this double coloring approach *bi-coloring*. The difference between the direct and substitution cases is in how the intersection graphs, $\mathcal{G}_C^I, \mathcal{G}_R^I$, are defined, and how the nonzeroes of J are extracted from the pair $(W^T J, JV)$.

Direct determination

In the direct case the intersection graph $\mathcal{G}_C^I$ is defined: $\mathcal{G}_C^I = (\mathcal{V}_C^I, \mathcal{E}_C^I)$ where

- Vertex $j \in \mathcal{V}_C^I$ if nnz[3](column $j \cap J_C) \neq 0$.
- $(r,s) \in \mathcal{E}_C^I$ if $r \in \mathcal{V}_C^I, s \in \mathcal{V}_C^I$, $\exists k$ such that $J_{kr} \neq 0, J_{ks} \neq 0$ and *either* $(k,r) \in J_C$ *or* $(k,s) \in J_C$.

The key point in the construction of graph $\mathcal{G}_C^I$, and why $\mathcal{G}_C^I$ is distinguished from the usual column intersection graph, is that columns r and s are said to intersect if and only if their nonzero locations overlap, in part, in J_C: i.e., columns r and s intersect if $J_{kr} \cdot J_{ks} \ = 0$ and *either* $(k,r) \in J_C$ *or* $(k,s) \in J_C$ for some k.

The "transpose" of the procedure above is used to define $\mathcal{G}_R^I = (\mathcal{V}_R^I, \mathcal{E}_R^I)$. Specifically, $\mathcal{G}_R^I = (\mathcal{V}_R^I, \mathcal{E}_R^I)$ where

- Vertex $i \in \mathcal{V}_R^I$ if nnz(row $i \cap J_R) \neq 0$.
- $(r,s) \in \mathcal{E}_R^I$ if $r \in \mathcal{V}_R^I, s \in \mathcal{V}_R^I$, $\exists k$ such that $J_{rk} \neq 0, J_{sk} \neq 0$ and *either* $(r,k) \in J_R$ *or* $(s,k) \in J_R$.

In this case the reason graph $\mathcal{G}_R^I$ is distinguished from the usual row intersection graph is that rows r and s are said to intersect if and only if their nonzero locations overlap, in part, in J_R: rows r and s intersect if $J_{rk} \cdot J_{sk} \ = 0$ and *either* $(r,k) \in J_R$ *or* $(s,k) \in J_R$ for some k.

The bi-partition (G_R, G_C), induced by coloring of graphs $\mathcal{G}_R^I$ and $\mathcal{G}_C^I$, is consistent with direct determination of J. To see this consider a nonzero element $(i,j) \in J_C$: column j is in a group of G_C (corresponding to a color) with the property that no other column in G_C has a nonzero in row i: hence, element (i,j) can be directly determined. Analagously, consider a nonzero element $(r,s) \in J_R$: row r will be in a group of G_R (corresponding to a color) with the property that no other row in G_R has a nonzero in column s: hence, element (r,s) can be directly determined. Since every nonzero of J is covered, the result follows.

Example: Consider the example Jacobian matrix structure shown in Figure 1.10 with the partition (J_C, J_R) shown.

The graphs $\mathcal{G}_C$ and $\mathcal{G}_R$ formed by the algorithm outlined above are given in Figure 1.11. Coloring $\mathcal{G}_C$ requires 3 colors, while $\mathcal{G}_R$ can be colored in two. Boolean matrices V and W can be formed in the usual way: each column corresponds to a group (or color) and unit entries indicate column (or row) membership in that group:

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad JV = \begin{pmatrix} J_{11} & 0 & \times \\ 0 & 0 & \times \\ J_{31} & \times & \times \\ 0 & \times & 0 \\ 0 & J_{52} & J_{53} \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad W^T J = \begin{pmatrix} \times & J_{32} & J_{13} & J_{14} & J_{35} \\ \times & J_{42} & J_{23} & J_{44} & 0 \end{pmatrix}.$$

[3] If M is a matrix or a vector then "nnz(M)" is the **n**umber of **n**on**z**eroes in M

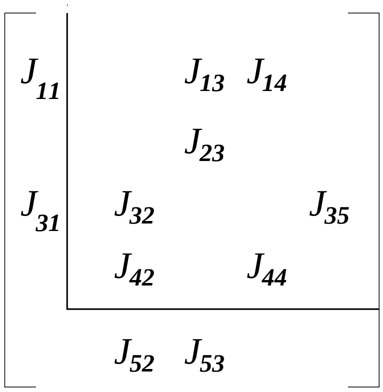

Figure 1.10: Example Partition

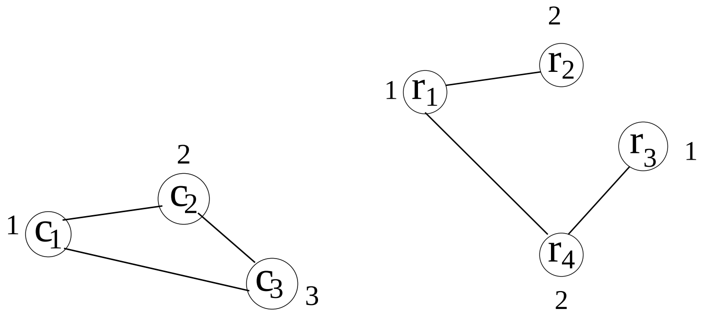

Figure 1.11: Graphs $\mathcal{G}_C$ and $\mathcal{G}_R$ (direct approach)

Clearly, all nonzero entries of J can be identified in either JV or $W^T J$.

Determination by substitution

The basic advantage of determination by substitution in conjunction with partition $J = [J_C | J_R]$ is that sparser intersection graphs $\mathcal{G}_C^I, \mathcal{G}_R^I$ can be used. Sparser intersection graphs mean thinner matrices V,W which, in turn, result in reduced cost.

In the substitution case the intersection graph $\mathcal{G}_C^I$ is defined: $\mathcal{G}_C^I = (\mathcal{V}_C^I, \mathcal{E}_C^I)$ where

- Vertex $j \in \mathcal{V}_C^I$ if nnz (column $j \cap J_C) \neq 0$.
- $(r, s) \in \mathcal{E}_C^I$ if $r \in \mathcal{V}_C^I, s \in \mathcal{V}_C^I$, $\exists k$ such that $J_{kr} \neq 0, J_{ks} \neq 0$ and **both** $(k, r) \in J_C$, $(k, s) \in J_C$.

Note that the intersection graph $\mathcal{G}_C^I = (\mathcal{V}_C^I, \mathcal{E}_C^I)$ captures the notion of two columns intersecting if there is overlap in nonzero structure in J_C: columns r and s intersect if $J_{kr} \cdot J_{ks} \;\; = 0$ and **both**

$(k, r) \in J_C$, $(k, s) \in J_C$ for some k. It is easy to see that $\mathcal{E}_C^I$ is a subset of the set of edges used in the direct determination case.

The "transpose" of the procedure above is used to define $\mathcal{G}_R^I = (\mathcal{V}_R^I, \mathcal{E}_R^I)$. Specifically, $\mathcal{G}_R^I = (\mathcal{V}_R^I, \mathcal{E}_R^I)$ where

- Vertex $i \in \mathcal{V}_R^I$ if row $i \in J_R$ and $nnz(\text{row } i \cap J_R) \neq 0$.
- $(r, s) \in \mathcal{E}_R^I$ if $r \in \mathcal{V}_R^I$, $s \in \mathcal{V}_R^I$, $\exists k$ such that $J_{rk} \neq 0, J_{sk} \neq 0$ and **both** $(r, k) \in J_R$, $(s, k) \in J_R$.

The intersection graph $\mathcal{G}_R^I = (\mathcal{V}_R^I, \mathcal{E}_R^I)$ captures the notion of two rows intersecting if there is overlap in nonzero structure in J_R: rows r and s intersect if $J_{rk} \cdot J_{sk} \ \ = 0$ and $(r, k) \in J_R$ **and** $(s, k) \in J_R$ for some k. It is easy to see that $\mathcal{E}_R^I$ is a subset of the set of edges used in direct determination.

All the elements of J can be determined from $(W^T J, JV)$ by a substitution process. This is evident from the illustrations in Figure 1.12.

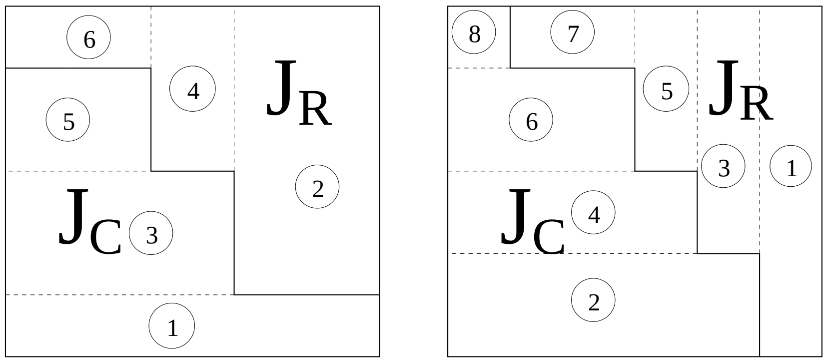

Figure 1.12: Substitution Orderings

Figure 1.12 illustrates two of four possible nontrivial types of partitions. In both cases it is clear that nonzero elements in the section labelled "1" can be solved for directly – by the construction process they will be in different groups. Nonzero elements in "2" can either be determined directly, or will depend on elements in section "1". But elements in section "1" are already determined (directly) and so, by substitution, elements in "2" can be determined after "1". Elements in section "3" can then be determined, depending only on elements in "1" and "2", and so on until the entire matrix is resolved.

Example. Consider again the example Jacobian matrix structure shown in Figure 1.10. Column and row intersection graphs corresponding to substitution are given in Figure 1.13. Note that $\mathcal{G}_C$ is disconnected and requires 2 colors; $\mathcal{G}_R$ is a simple chain and also requires 2 colors.

The coloring of $\mathcal{G}_C$ and $\mathcal{G}_R$ leads to the following matrices V, W and the resulting computation of JV, $W^T J$:

$$V = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad JV = \begin{pmatrix} J_{11} + J_{13} & 0 \\ \times & 0 \\ J_{31} & \times \\ 0 & \times \\ J_{53} & J_{52} \end{pmatrix}$$

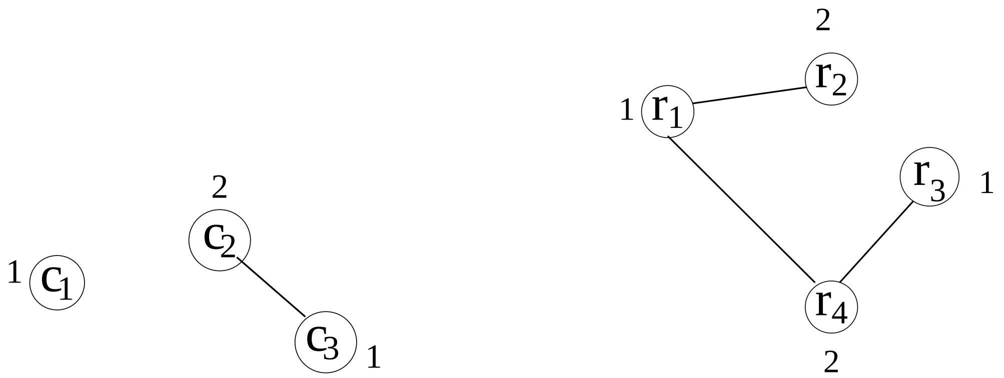

Figure 1.13: Graphs $\mathcal{G}_C$ and $\mathcal{G}_R$ for substitution process

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad W^T J = \begin{pmatrix} \times & J_{32} & J_{13} & J_{14} & J_{35} \\ \times & J_{42} & J_{23} & J_{44} & 0 \end{pmatrix}$$

It is now easy to verify that all nonzeroes of J can be determined via substitution.

1.1.4 How to partition J.

We now consider the problem of obtaining a useful partition $[J_C|J_R]$, and corresponding permutation matrices P,Q, as illustrated in Figure 1.9. A simple heuristic is proposed based on the knowledge that the subsequent step, in both the direct and the substitution method, is to color intersection graphs based on this partition.

Algorithm **MNCO** builds partition J_C from bottom up, and partition J_R from right to left. At the k^{th} major iteration either a new row is added to J_C or a new column is added to J_R: the choice depends on considering a lower bound effect:

$$\rho(J_R^T) + max(\rho(J_C), nnz(r)) < (\rho(J_C) + max(\rho(J_r^T), nnz(c)),$$

where $\rho(A)$ is the maximum number of nonzeroes in any row of matrix A, r is a row under consideration to be added to J_C, c is a column under consideration to be added to J_R. Hence, the number of colors needed to color $\mathcal{G}_C^I$ is bounded below by $\rho(J_C)$; the number of colors needed to color $\mathcal{G}_R^I$ is bounded below by $\rho(J_R^T)$.

In algorithm **MNCO**, matrix $M = J(R, C)$ is the submatrix of J defined by row indices R and column indices C: M consists of rows and columns of J not yet assigned to either J_C or J_R.

Minimum Nonzero Count Ordering (MNCO)

1. Initialize $R = (1 : m)$, $C = (1 : n)$, $M = J(R, C)$
2. Find $r \in R$ with fewest nonzeros in M
3. Find $c \in C$ with fewest nonzeros in M

4. Repeat Until $M = \emptyset$

    if $\rho(J_R^T) + max(\rho(J_C), nnz(r)) < (\rho(J_C) + max(\rho(J_R^T), nnz(c))$ (LB)

        J_C=$J_C \cup$ (r $\cap$ C)

        R=R-{r}

    else

        J_R=$J_R \cup$ (c $\cap$ R)

        C=C-{c}

    end if

    $M = J(R, C)$.

end repeat

Note that, upon completion, J_R, J_C have been defined; the requisite permutation matrices are implicitly defined by the ordering chosen in **MNCO**.

Accuracy

In general, the substitution approach requires fewer colors and therefore is more efficient[4], in principle, than direct determination. However, there is a possibility of increased round-off error due to the substitution process. Nevertheless, Coleman and Verma [12] show that the error growth is limited and considerably better than finite differencing.

Limited error growth is partly due to the observation, proven in [12], that the number of substitutions required to determine any nonzero of J using the bi-coloring approach described above is at most $m + n - 2$. Second, the substitution methods for symmetric finite differencing, described in §1.1.2 experience extreme error growth when the stepsizes associated with different columns vary significantly. This is not an issue in the AD substitution approach since there is no stepsize! (From an analytic point of view this is equivalent to associating unit stepsizes with all columns.)

1.2 Exercises

1. A direct sparse finite-difference method (based on a coloring of the intersection graph) for determining a sparse Jacobian matrix can be viewed as solving a diagonal system of equations. Illustrate the system for the general case.

2. Let A be an arbitrary sparse m by n matrix and let vectors $d_1, ..., d_p$ be such that A is uniquely determined from the products $Ad_1, ..., Ad_p$.

a) Show that

$$p \geq \rho_* \overset{def}{=} max\{\rho_i : 1 \leq i \leq m\}$$

where ρ_i is the number of nonzeroes in row i of A. (Note: This is a general result and is not restricted to **direct** determination.)

b) Show that it is possible to determine directions $d_1, ..., d_{\rho_*}$ such that A is uniquely determined from $Ad_1, ..., Ad_{\rho_*}$. (Note: In general this will not be a direct determination - it may involve solving a system of equations.)

[4] Of course a substitution method does incur the extra cost of performing the substitution calculation. However, this can be done very efficiently and the subsequent cost is usually negligible.

3. Show that there exist graphs that do not correspond to the intersection graph of any n-by-n matrix. Show that every graph is the intersection graph of some m-by-n matrix.

4. Illustrate with an example that it is sometimes possible to approximate sparse **unsymmetric** Jacobian matrices using a **substitution** process in fewer function evaluatio ns than an optimal direct determination based on a consistent partition (i.e., a coloring). Hint: Consider grouping columns but allow columns to belong to more than one group.

5. Illustrate with an example that it is sometimes better (in terms of function evaluations) to approximate sparse **unsymmetric** Jacobian matrices using a **direct** method which does not correspond to a partition of the columns - i.e., which allows columns to belong to more than one group, and therefore does not correspond to a coloring of the intersection graph. If possible give a dramatic example.

6. Put together a simple subspace trust region code to solve a large sparse nonlinear minimization code. You can use a 2-D subspace approach, where the current subspace is defined by an approximate conjugate gradient solution to the current Newton system. The Hessian matrix can be obtained by sparse finite differences.

The problem to solve is tbroy.m,

$$\text{[value, grad]} = \text{tbroy(x)},$$

starting at $x = (-1, -1, ..., -1)$. The structure of the Hessian matrix is available in fbox3.mat . You can use M-files: cg.m, color.m, sfd.m, trust.m. To learn about these functions in MATLAB try both help and Appendix II.

1.3 Structured Problems

Some large-scale problems are sparse, some are not. However, when a large-scale problem is not sparse it is usually structured; recognition of this fact can significantly reduce the cost of computing Jacobian and Hessian matrices (and even gradients). "Structure" is sometimes thought of as a problem-specific feature that can only be "exploited" on a case-to-case basis. Alternatively, in large-scale optimization "structure" is sometimes associated with a particular kind of form, partial separability. In this section we indicate that both views are severely limited: we present a general but practical notion of structure (that includes partial separability as a special case). For more deatils see ??

1.3.1 Structured Jacobian Matrices

Unfortunately, not all large systems exhibit sparse Jacobian matrices. For example, the following composite structure is common in large-scale problems:

$$F(x) \;=\; \bar{F}(y) \tag{1.13}$$

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

$$Ay \;=\; \tilde{F}(x), \tag{1.14}$$

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

$$J \;=\; \bar{J}A^{-1}[\tilde{J} - A_x y]. \tag{1.15}$$

It is the application of A^{-1} that causes matrix J to be dense – this will almost surely be the case unless A^{-1} is very special, e.g., diagonal.

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

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

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

$$J_E \begin{pmatrix} \delta_x \\ \delta_{y_1} \\ \delta_{y_2} \end{pmatrix} \;=\; \begin{pmatrix} 0 \\ 0 \\ F(x) \end{pmatrix}, \tag{1.16}$$

where

$$J_E \;=\; \begin{bmatrix} -\tilde{J} & I & 0 \\ A_x y_2 & -I & A \\ 0 & 0 & -\bar{J} \end{bmatrix}. \tag{1.17}$$

Here is a key point: the "extended" Jacobian matrix J_E is sparse and clearly sparse AD-techniques, e.g., [1, 13, 17], can be applied with respect to

$$F_E(x, y) = \begin{pmatrix} y_1 - \tilde{F}(x) \\ A(x)y_2 - y_1 \\ -\bar{F}(y_2) \end{pmatrix}. \tag{1.18}$$

to efficiently determine J_E. For example, the work required by the bi-coloring technique developed in [13] is $\chi \cdot \omega(F_E) \;=\; \chi \cdot \omega(F)$ where χ is a "bi-chromatic number" dependent on the sparsity of J_E. Typically, $\chi << \min(m, n)$. Additional linear algebra work is needed to extract J from J_E: compute the Schur complement (introducing zero matrices in positions $(3, 2), (3, 3)$) and obtain,

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

If it is the Newton step $\delta_x = -J^{-1}F(x)$ that is required then it is not necessary to explicitly form J. For example the extended system (1.16) can be solved directly.

It is also possible to compute an approximate Newton step, without forming J, using an iterative solver. Specifically, if a sparse factorization of A is computed, an iterative solver involving only matrix-vector products can be applied to

$$(\bar{J}A^{-1}[\tilde{J} - A_x y])s \;=\; -F(x). \tag{1.19}$$

Large-scale nonlinear systems $F(x) \;=\; 0$ often exhibit a natural lower Hessenberg form. Usually, an easy programmatic way to describe F, at a high level, is to state this lower Hessenberg description, or program, F_E. The corresponding Jacobian of F_E, J_E, is typically a sparse matrix: sparse AD techniques can be applied to efficiently compute J_E. The Jacobian of F and/or the Newton step $\delta_x = -J^{-1}F(x)$ can subsequently be computed from J_E. Two key points are:

- The matrix J_E, though larger than J, is usually considerably sparser.
- The high level program F_E is usually readily available to the user: it is often the most natural way of expressing F.

To be more precise, a natural way to evaluate the nonlinear systems $z = F(x)$ is via the lower Hessenberg program illustrated in Figure 1.14 where we assume equation i *uniquely* determines y^i, $i = 1 : p$. We take the point of view that the function F_E is explicitly available, where

$$\boxed{\begin{array}{l}
\textit{Solve for } y_1 : \ F_1(x, y_1) = 0 \\
\textit{Solve for } y_2 : \ F_2(x, y_1, y_2) = 0 \\
\vdots \\
\textit{Solve for } y_p : \ F_p(x, y_1, y_2, \ldots, y_p) = 0 \\
\textit{``Solve'' for output } z : \ z - F_{p+1}(x, y_1, y_2, \ldots, y_p) \ = \ 0
\end{array}}$$

Figure 1.14: A General Structured Computation

$$F_E = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_p \end{pmatrix}.$$

Indeed, usually the component functions of F_E, F_i, $i = 1 : p$, are conveniently available to the user.

Many well-known structured problems are covered by this view: e.g., partially-separable functions, dynamical (recursive) systems, composite functions, various gradient computations. Examples are illustrated in Section §4.

The program in Figure 1.14 can be differentiated to give the extended Newton system:

$$J_E \begin{pmatrix} \delta x \\ \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ -F \end{pmatrix} \tag{1.20}$$

where

$$J_E = \left(\begin{array}{c|cccc}
\frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y_1} & & & \\
\frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y_1} & \frac{\partial F_2}{\partial y_2} & & \\
\vdots & \vdots & & \ddots & \\
\frac{\partial F_p}{\partial x} & \frac{\partial F_p}{\partial y_1} & \cdots & & \frac{\partial F_p}{\partial y_p} \\
\hline
\frac{\partial F_{p+1}}{\partial x} & \frac{\partial F_{p+1}}{\partial y_1} & \cdots & & \frac{\partial F_{p+1}}{\partial y_p}
\end{array}\right). \tag{1.21}$$

If each "interior vector" y_i is uniquely determined then the the super-diagonal of (1.21) has nonsingular blocks, $\frac{\partial F_i}{\partial y_i}$, and δ_x is the Newton step $\delta_x = -J^{-1}F(x)$.

We contend that J_E will likely be sparse, considerably sparse than J; hence, sparse AD techniques can be used to obtain J_E. The Jacobian matrix J can be computed from J_E by zeroing the

$(2,2)$-block row in J_E using block Gauss transformations. Matrix J shows up in the $(2,1)$ location after elimination of the $(2,2)$-block row.

If it is the Newton step that is required, and not matrix J per se, then there are two natural alternatives to the explicit computation of J, given J_E. First, system (1.20) can be solved directly using a direct sparse factorization[5]: this is clearly an attractive option in some cases, e.g., Figure **??**. A second possibility is to perform the $(2,1)$-elimination symbolically to produce a "product form" expression for J, in the $(2,1)$ location, which could then be used in any iterative linear solver requiring only matrix-vector products. An example of this latter possibility is given in the penultimate paragraph in §1.

Examples.

We discuss three common classes of structured nonlinear systems. In each case the Jacobian matrix is potentially dense; however, by differentiating the natural high-level program to compute F, as discussed in §3, underlying sparsity can be exploited in the use of *AD* tools. The result is often a dramatic increase in efficiency.

Composite Functions and Dynamical Systems A composite function $F : \Re^n \rightarrow \Re^m$ can be written

$$F(x) = \bar{F}(T_p(T_{p-1}(\ldots(T_1(x))\ldots), \tag{1.22}$$

where, in general, functions $\bar{F}, T_i, i = 1 : p$ are vector maps. Recursively applying the chain rule yields J, the Jacobian of $F(x)$:

$$J = \bar{J} \cdot J_p \cdot J_{p-1} \cdot \ldots \cdot J_1$$

where J_i is the Jacobian of T_i evaluated at $T_{i-1}(T_{i-2}(\ldots(T_1(x))\ldots)$; $\bar{J}$ is the Jacobian of $\bar{F}$ evaluated at argument $T_p(T_{p-1}(\ldots(T_1(x))\ldots)$.

A natural high-level program to evaluate F is given below where we let y_0 denote x:

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

Clearly this program is a special case of the general form F_E given in Figure 1.14, with corresponding Jacobian matrix J_E:

$$J_E = \begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_p & I \\ & & & & -\bar{J} \end{pmatrix}$$

If a Newton step is required, it can be much more costly to determine J directly via *AD* compared to determining J_E using *AD* and then solving the extended system (1.20) to determine the Newton step.

[5]Griewank [15] has proposed a similar idea in a more extreme form: F_E is defined with respect to *all* intermediate variables. The resulting extended Jacobian matrix J_E is huge, but very sparse.

For example, consider the special case of a dynamical system: $T_i = T, \quad i = 1:p-1$, $\bar{F} = T$, $T : \Re^n \to \Re^n$ is a square nonsingular mapping, and

$$F(x) \;=\; T(T(\ldots(T(x))\ldots)).$$

Assume that the Jacobian of T is a tri-diagonal Jacobian matrix.

For all p sufficiently large the Jacobian matrix J will be dense; hence determination of J by direct application of automatic differentiation requires $O(n \cdot \omega(F)) \;=\; O(n \cdot p \cdot \omega(T))$ flops. Therefore direct determination of J followed be a direct solve requires $O(n \cdot p \cdot \omega(T) \;+\; n^3)$ flops. However, the determination of J_E requires $O(p \cdot \omega(T))$ flops and solution of the banded extended system takes $O(n \cdot p)$ flops for a total of $O(p \cdot \omega(T) \;+\; n \cdot p)$ flops – generally, a much more attractive order of work.

Generalized partial separability

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

$$F(x) = \bar{F}(y_1, y_2, \ldots, y_p); \qquad y_i = T_i(x), \quad i = 1, 2, \ldots, p. \tag{1.23}$$

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

Following the general form given in Figure 1.14, F can be computed with the following program,

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

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

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

This yields the simple 2-liner:

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

The general extended system (1.20) reduces to

$$\left(\begin{array}{c|cccc} -\tilde{J}_1 & I & & & \\ -\tilde{J}_2 & & I & & \\ \vdots & & & \ddots & \\ -\tilde{J}_p & & & & I \\ \hline 0 & \bar{J}_1 & \bar{J}_2 & \cdots & \bar{J}_p \end{array}\right) \begin{pmatrix} \delta x \\ \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -F \end{pmatrix} \tag{1.24}$$

where $\bar{J}$ is broken into different components $\bar{J}_i$ corresponding to variables y_i. If F is partially separable in the usual fashion, i.e., function $\bar{F}$ is a summation, then each matrix $\bar{J}_i$ is an identity matrix. It is clear that the computation of J_E will, in general, be considerably more economical than than the straightforward application of AD to detemine $J = \sum_{i=1:p} \tilde{J}_i \cdot \bar{J}_i$ via AD. For example, Jacobians $\tilde{J}$, $\bar{J}$ can be determined by applying the bi-coloring technique [13] to $\tilde{F}$ and $\bar{F}$ respectively. And of course if $\bar{F}$ is simple enough, $\bar{J}$ will be constant (and trivially available).

If the function $\bar{F}$ is itself non-trivial it may be advantageous to exploit this by expanding the 2-line program. For example, let I_j be an index set indentifying a subset of $\{1,\dots p\}$, $j = 1,\dots,t$. Further, let $\hat{F}_j$ be a vector function, $j = 1,\dots,t$ and suppose $\bar{F}(Y) = \sum_{j=1}^{t} \hat{F}_j(\sum_{I_j} y_j)$. So the program to evaluate $\bar{F}(Y)$ can be written

$$\boxed{\begin{array}{l} \textit{for } j = 1:t \\ \qquad w_j = \sum_{I_j} y_j \\ \qquad v_j = \hat{F}_j(w_j) \\ z = \sum_{j=1}^{t} v_j \end{array}}$$

Clearly the evaluation of this "group partially separable" function is easily expressed in the lower Hessenberg form given in Figure 1.14; the corresponding Jacobian matrix J_E is likely to be sparse and economically computable via the AD techniques in [13]

Product function

A special case of the generalized partially separable form is a product function: Suppose that $F : \Re^n \to \Re^n$ is a component-wise product of functions:

$$F(x) = F_1(x). * F_2(x). * \ \dots\ . * F_p(x),$$

where $F_i : \Re^n \to \Re^n$, $i = 1 : n$, and the notation ".*" indicates component-wise multiplication (following the MATLAB convention). A natural way to evaluate F at an argument x is to execute the program,

$$\boxed{\begin{array}{l} \textit{for } i = 1:p \\ \qquad \textit{“Solve” for } y_i\textit{: } y_i - F_i(x) = 0 \\ \textit{“Solve” for } z : z - y_1. * y_2. * \ \dots\ . * y_p = 0 \end{array}}$$

The extended Newton system (1.20) simplifies to:

$$\left(\begin{array}{c|cccc} -J_1 & I & & & \\ -J_2 & & I & & \\ \vdots & & & \ddots & \\ -J_p & & & & I \\ \hline 0 & D_1 & D_2 & \cdots & D_p \end{array}\right) \begin{pmatrix} \delta x \\ \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -F \end{pmatrix} \tag{1.25}$$

where D_i is a diagonal matrix, $D_i = \prod_{j=1,j\neq i}^{p} diag(y_i)$, and "$diag(y_i)$" is the diagonal matrix with the vector y_i defining the diagonal in the natural order. Note that the Jacobian of F reduces to

$$J = \sum J_i(x) \cdot D_i.$$

Clearly it is entirely possible that J is relatively dense even when each component Jacobian J_i is very sparse; hence, direct determination of J via AD is usually unattractive compared to the calculation of the extended Jacobian, J_E, via AD (e.g., using the bi-coloring approach [13]).

A remark

Our two main examples, composite functions and generalized partially separable functions, are complementary in a structural sense. The evaluation of a composite function is a *depth* computation: each subsequent intermediate variable y_i depends on the previous intermediate y_{i-1}. The computational graph of a composite function is a long chain: See Figure 1.15.

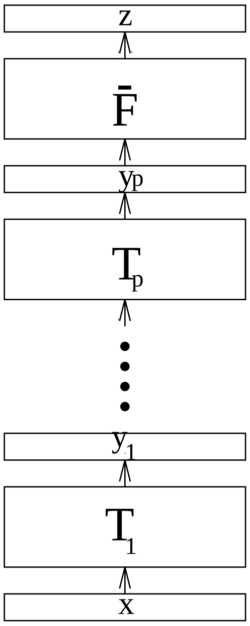

Figure 1.15: Composite Function

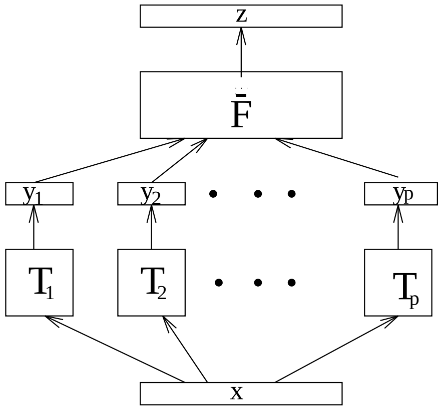

Figure 1.16: Generalized Partially Separable Function

In contrast, generalized partially separable functions are primarily breadth computations: typically, the intermediate variables y_i are relatively independent of each other: it is the final computation $z = \bar{F}(y_1, y_2, \ldots, y_p)$ that binds the intermediates together. The computational graph is short and fat; see Figure 1.16.

Despite such grossly different structures the lower Hessenberg format, illustrated in Figure 1.14, is applicable in both cases: this view represents a convenient way to program the evaluation of F and allows for the efficient application of AD tools to determine the Jacobian matrix and/or the Newton step.

1.3.2 Structured Hessian Matrices

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

However, with the advent of automatic differentiation (AD) tools this balance is now being challenged. It is now possible to have first and second derivatives automatically computed given a code that computes the objective function. The difficulty is one of computational cost: straightforward application of automatic differentiation tools may be inordinately expensive for large problems. Results obtained in [2, 12] show that for the related *sparse Jacobian* problem, the cost can be dramatically reduced if sparsity is exploited. In principle similar techniques [6] can be applied to the sparse Hessian determination problem in a straightforward manner.

Example: A Composite Function

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

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

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

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

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

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

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

$$H_E = \begin{bmatrix} A_x y - \tilde{J} & A & 0 \\ A_x^T w + \nabla^2_{yx} \bar{f} & \nabla^2_{yy} \bar{f} & A^T \\ w^T[(A_x y) - \tilde{J}]_x + \nabla^2_{xx} \bar{f} & w^T A_x + \nabla^2_{xy} \bar{f} & (A_x y - \tilde{J})^T \end{bmatrix}. \tag{1.26}$$

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

$$H_E^S = \begin{bmatrix} 0 & A & A_x y - \tilde{J} \\ A^T & \nabla^2_{yy} \bar{f} & A_x^T w + \nabla^2_{yx} \bar{f} \\ (A_x y - \tilde{J})^T & w^T A_x + \nabla^2_{xy} \bar{f} & w^T[A_x y - \tilde{J}]_x + \nabla^2_{xx} \bar{f} \end{bmatrix}. \tag{1.27}$$

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

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

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

There are three important observations to make about this example. First, the matrix H is likely to be dense, due to the action of A^{-1}, whereas under reasonable assumptions H_E will be sparse. Second, matrix H_E can be obtained using automatic differentiation applied to function $GF_E(x, y, w)$. Sparse AD techniques [12] can be applied to $GF_E(x, y, w)$ to allow for the economical calculation of H_E. Third, it is also possible to obtain matrix H_E without *explicitly* applying a sparse AD technique to the structured gradient function GF_E. We have in mind the following recipe:

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

This last observation indicates how to use sparse AD techniques to compute H_E given a structured program to evaluate f (as opposed to a structured representation of $\nabla f(x)$). We will see in Section 2.2 that this recipe can be generalized. We believe that most large-scale objective functions in optimization are naturally expressed, at a high-level, in a structured form. In particular, the program to evaluate $z = f(x)$ can be written in the form given in Figure **??**, where equation i *uniquely* determines (intermediate) vector y_i. So, a compact program to evaluate f is given in Figure 1.17

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

Figure 1.17: Structured f-Evaluation in Compact Form

where

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

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

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

$$J_E = \begin{pmatrix} \tilde{F}_x^E & \tilde{F}_y^E \\ \nabla_x \bar{f}^T & \nabla_y \bar{f}^T \end{pmatrix}. \tag{1.28}$$

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

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

How to Differentiate the Gradient Function

To answer the first question, the gradient of the structured function f can be evaluated as illustrated in Figure 1.18.

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

Figure 1.18: A Structured Gradient program

The derivation of this program is simple: First differentiate the extended function F_E to obtain J_E; then, eliminate the $(2,2)$-block, $\nabla_y \bar{f}^T$, to define vector w. Finally, modify the $(2,1)$-block of matrix J_E using w to get $\nabla_x f$. In other words, form matrix J_E (1.28) and then eliminate the $(2,2)$-block using a block Gaussian transformation.

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

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

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

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

where

$$H_E = \begin{pmatrix} \tilde{F}_x^E & \tilde{F}_y^E & 0 \\ (\tilde{F}_{yx}^E)^T w + \nabla_{yx}^2 \bar{f} & (\tilde{F}_{yy}^E)^T w + \nabla_{yy}^2 \bar{f} & (\tilde{F}_y^E)^T \\ (\tilde{F}_{xx}^E)^T w + \nabla_{xx}^2 \bar{f} & (\tilde{F}_{xy}^E)^T w + \nabla_{xy}^2 \bar{f} & (\tilde{F}_x^E)^T \end{pmatrix}.$$

Typically matrix H_E exposes more sparsity than matrix H – the composite function described in Section 1 is a good illustration. Moreover, additional sparsity gains can often be achieved, in principle, if the structure in Step 3. of the gradient-evaluation program is exploited. In particular, notice that the computation $\nabla_x f = \nabla_x \bar{f} + (\tilde{F}_x^E)^T \cdot w$ exhibits "partially separable" form. As illustrated in [11] it is often worthwhile to further break down this step:

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

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

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

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

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

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

How to Differentiate f (Twice)

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

$$H_E = \begin{pmatrix} \tilde{F}_x^E & \tilde{F}_y^E & 0 \\ \nabla_{yx}^2 g & \nabla_{yy}^2 g & (\tilde{F}_y^E)^T \\ \nabla_{xx}^2 g & \nabla_{xy}^2 g & (\tilde{F}_x^E)^T \end{pmatrix}. \tag{1.29}$$

This is an important observation because it yields the answer to the second major question of Section 2: How do we apply automatic differentiation directly to the f-evaluation code to yield the extended Hessian H_E in an efficient way? The recipe follows from (1.29) and the definition of w:

1. Using the sparse AD techniques developed in [11] compute the extended Jacobian $(\tilde{F}^E)$
2. Solve the block lower triangular system for w: $(\tilde{F}_y^E)^T w + \nabla_y \bar{f} = 0$.

3. Using sparse AD techniques, twice differentiate $g(x, y, w) = \bar{f} + w^T F^E(x, y)$, with respect to x, y, to determine the Hessian matrix, i.e., H_E. As indicated in Section 2.1, it can be advantageous to exploit the partially separable structure in $g(x, y, w) = \bar{f} + \sum_{i=1}^{k} w_i^T F_i^E$: i.e., compute the Hessian matrix of each component function $w_i^T F_i^E$ in turn.

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

$$H_E = \left(\begin{array}{c|c} A & L \\ \hline B & M \end{array} \right) \tag{1.30}$$

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

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

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

Bibliography

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

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

[3] C. H. Bischof, A. Bouaricha, P. M. Khademi, and J. J. Moré, *Computing gradients in large-scale optimization using automatic differentiation*, Tech. Rep. MCS-P488-0195, MCS, June 1995.

[4] T. F. Coleman, *Large Sparse Numerical Optimization*, Lecture Notes in Computer Science, Volume 165, Springer-Verlag, 1984.

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

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

[7] T. F. Coleman, B. S. Garbow, and J. J. Moré, *Software for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329–345.

[8] ——, *Software for estimating sparse Hessian matrices*, ACM Trans. Math. Software, 11 (1985), pp. 363–377.

[9] T. F. Coleman and J. J. Moré, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, 28 (1984), pp. 243–270.

[10] ——, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. on Numerical Analysis, 20 (1984), pp. 187–209.

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

[12] ——, *The efficient computation of sparse Jacobian matrices using automatic differentiation*, SIAM Journal on Scientific Computing, 19 (1998), pp. 1210–1233.

[13] ——, *The efficient computation of sparse Jacobian matrices using automatic differentiation*, SIAM Journal on Scientific Computing, 19 (1998), pp. 1210–1233.

[14] 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.

[15] A. Griewank, *Direct calculation of Newton steps without accumulating Jacobians*, in Large-Scale Numerical Optimization, T. F. Coleman and Y. Li, eds., SIAM, Philadelphia, Penn., 1990, pp. 115–137. Also appeared as Preprint MCS–P132–0290, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., February 1990.

[16] ——, *Some Bounds on the Complexity of Gradients, Jacobians, and Hessians*, in Complexity in Nonlinear Optimization, P. Pardalos, ed., World Scientific Publishers, 1993, pp. 128–161.

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

[18] M. J. D. Powell and P. L. Toint, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060–1074.