Automatic Computation of Sparse Jacobians by Applying the Method of Newsam and Ramsdell*

Uwe Geitner† Jean Utke† Andreas Griewank†

Abstract

The computation of sparse Jacobians is a common subproblem in iterative numerical algorithms. The sparsity structure is not always known a priori and may sometimes change from point to point. The subject of this paper is the automatic detection of the sparsity structure and its exploitation for an efficient computation of Jacobians using automatic differentiation, graph coloring, and fast solution algorithms for Vandermonde systems.

1 Basic Principles and Motivation

Many numerical algorithms for the solution of optimization problems, nonlinear algebraic systems, or differential equations require the computation of Jacobians or other derivatives. In virtually all cases the function of interest $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ is given as an evaluation routine in a computer programming language. Apart from hand-coded derivatives, symbolic differentiation, and divided differences, automatic differentiation (AD) is an efficient way to obtain derivative values. AD uses the execution trace of the evaluation routine to compute derivative values at a given argument point. The computational effort, which depends on the number of dependent and independent variables, is clearly predictable and often less than the effort for divided differences. In addition, automatic differentiation is exact up to the precision of the underlying subroutine; in other words, the truncation error incurred by divided differences is avoided.

The sparsity structure of the Jacobian $\boldsymbol{J} = \boldsymbol{f}'(\boldsymbol{x})$ of $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is determined by the dependency relation between the independent variables $\boldsymbol{x}$ and the dependent variables $\boldsymbol{y}$. The computational effort of symbolic differentiation is dominated by the (typically exponential) expression swell, while hand-coded derivatives are error prone, and their generation is time consuming. Automatic differentiation tools are based on the code of the evaluation routine. Therefore, information about the dependency relation is available. It is a natural idea to ask for this qualitative dependency information before proceeding to the actual evaluation. (We do not intend to explain the theory of automatic differentiation and its application. One may refer to M. Iri's contribution in [1], L. Rall's introduction in this volume, or [13] for more detailed introductions.) Four facts are essential in our context:

(f1) there are two modes of automatic differentiation, the forward and the reverse modes, both of which have predictable temporal and spatial complexity;

(f2) the forward mode can compute linear combinations of columns ($\boldsymbol{J}\boldsymbol{s}$) of the Jacobian[1] with a complexity that does not depend on m;

---

*to appear in: *Computational Differentiation, Proceedings of the Second International Workshop*, M. Berz et.al.eds., SIAM, Philadelphia, Penn. 1996, pp. 161–172

†Technical University Dresden, Institute of Scientific Computing

[1]We can look at these linear combinations as directional derivatives in the direction s.

(f3) the reverse mode can compute linear combinations of rows ($\boldsymbol{w}^T\boldsymbol{J}$) of the Jacobian with a complexity that does not depend on n; and

(f4) the temporal complexity of one linear combination in the forward or reverse mode is a small multiple of the complexity ops($\boldsymbol{f}$) of $\boldsymbol{f}$ itself.

The implementation is based on the package ADOL-C [13]. For the Fortran test codes we used the Fortran 90 front-end ADOL-F [17]. The visualization of the sparsity patterns were done with routines from PETSc [14].

1.1 The Problem

For a given function

$$\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m \quad ; \quad \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}$$

one wants to evaluate the Jacobian

$$\boldsymbol{J} = \left[\frac{\partial f_i}{\partial x_j}\right]_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{1}$$

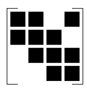
at a certain argument $\boldsymbol{x}_0$. The function $\boldsymbol{f}$ defines a dependency relation

$$j \to i \iff x_j \to y_i \iff y_i \text{ depends on } x_j$$

and therefore defines the sparsity structure of $\boldsymbol{J}$. How do we evaluate $\boldsymbol{J}$? A naive approach using statement (f2) would be to choose the computation of n linear combinations $\boldsymbol{J}\boldsymbol{s}_i$, where the $\boldsymbol{s}_i$ are the columns of the $n \times n$ identity matrix. Similarly, we could perform a computation of m linear combinations $\boldsymbol{w}_j^T\boldsymbol{J}$, where the $\boldsymbol{w}_j^T$ are the rows of the $m \times m$ identity matrix. However, for sparse matrices this concept is inefficient, since there are ways to reduce the number of necessary linear combinations to $p << \min\{n, m\}$.

Example 1: If $n = m = 4$ and the Jacobian $\boldsymbol{J}$ has the sparsity structure

(2)



,

it can be computed by using only 3 instead of 4 linear combinations. The $\boldsymbol{s}_i$ are the columns of the seed matrix $\boldsymbol{S}$, and we get the product $\boldsymbol{J}\boldsymbol{S}$:

$$\boldsymbol{B} = \boldsymbol{J}\boldsymbol{S} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} + \frac{\partial f_1}{\partial x_4} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_1} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_1} + \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_4} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \\ \frac{\partial f_4}{\partial x_4} & 0 & \frac{\partial f_4}{\partial x_3} \end{bmatrix} \quad \text{for} \quad \boldsymbol{S} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} . \tag{3}$$

The full Jacobian is easily reconstructed by computing the two differences $j_{1,4} = b_{1,1} - b_{1,4}$ and $j_{2,3} = b_{2,3} - b_{2,1}$ □

This small example is quite dense. Many large applications have much sparser Jacobians, and therefore the reduction of the number of actually required linear combinations to a smaller p tends to be significant. The next two questions are as follows:

- How one can find a suitable seed matrix $\boldsymbol{S}$ for a given sparsity pattern?
- How complicated is the reconstruction of the Jacobian in general?

The first algorithm to construct $\boldsymbol{S}$ was proposed by Curtis, Powell, and Reid (CPR) in [9] in the context of divided differences, since automatic differentiation was barely known then. Instead of directional derivatives $\boldsymbol{J}\boldsymbol{s}$, divided differences

$$\frac{\boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{s})-\boldsymbol{f}(\boldsymbol{x})}{h} \tag{4}$$

with respect to a certain direction $\boldsymbol{s}$ were used. Nonetheless, all statements about complexity in [9] still hold because of (f4). CPR is based on finding structurally orthogonal columns in the Jacobian. Two columns are structurally orthogonal when no row exists in which both have a nonzero entry. This relation implies that the inner product of the two columns is always zero. The column indices $\boldsymbol{j}_k$ of the Jacobian are split into disjoint sets $\mathcal{S}_r$ so that

$$i, j \in \mathcal{S}_r \quad \Rightarrow \quad \text{the columns } i, j \text{ are structurally orthogonal,}$$

with $\{1, \ldots, n\} = \bigcup_r \mathcal{S}_r$, and $\mathcal{S}_s \cap \mathcal{S}_r = \emptyset$, for $r \neq s$. This construction implies that the calculation of the directional derivatives associated with the columns in one set $\mathcal{S}_r$ can be done with just one linear combination (f2) or one divided difference (4), respectively. The corresponding direction $\boldsymbol{s}_r$ is a 0-1 vector representing the sum of all 0-1 sparsity patterns of the columns $\in \mathcal{S}_r$. If we look again at the example above, we realize that the pattern (2) does not have any structurally orthogonal columns at all. Nevertheless, it seems to be possible to reduce the number of necessary linear combinations to a number less than n.

The major advantage of choosing $\boldsymbol{S}$ according to structural orthogonality is that each element of $\boldsymbol{J}\boldsymbol{S}$ represents exactly one element of $\boldsymbol{J}$; that is, the reconstruction simplifies to a correct placement (spreading) of the elements. When Coleman and Moré [8] and Coleman and Cai [7] investigated the problem, they discovered a relation between finding structurally orthogonal columns and coloring the column incidence graph. Each color corresponds to a set $\mathcal{S}_r$. Thus the number of linear combinations to be computed is the number of colors. The graph coloring problem is known to be $\mathcal{NP}$-hard, but some reliable heuristics have been developed [8]. A lower bound for the number of colors in the column incidence graph is obviously the maximal number of nonzero elements per row nz_r. Though the heuristics typically return coloring numbers similar to the magnitude of the optimal number,[2] it is often much larger than nz_r. For some practical problems the heuristics return increasing coloring numbers with increasing dimension, while nz_r remains constant. In contrast to the original CPR algorithm, graph coloring achieves results that are essentially independent of column permutation.

Recently Steihaug and Hossain [18] and Coleman and Cai [7] proposed a major extension and improvement built upon vector-Jacobian products as in (f3). The principle is to look also at structurally independent rows, perform a coloring of the row incidence graph, and then exploit (f3). For matrices with an arrow-like structure that have only a few (nearly) dense rows and columns, both techniques alone are impracticable. Therefore, the authors of both [18] and [7] suggested a combination of the two techniques that allows the treatment of a larger class of sparse matrices.

In this paper we extend the method of G. N. Newsam and J. D. Ramsdell (NR), which is discussed in the next section. Unlike its competitors, the NR approach is not bounded by the coloring number of the column incidence graph but only by the potentially much smaller nz_r. An idea of how this could work was given in the first example.

[2] This is the so-called chromatic number χ.

2 The Method of Newsam and Ramsdell

In 1983 Newsam and Ramsdell published their paper [16] describing a more efficient method to calculate sparse Jacobians. Since they did not know about automatic differentiation, they had to rely on divided differences, with that method's implicit inaccuracy, but the important complexity results still hold. For the same reason they could treat only vector seed matrix products $\boldsymbol{JS}$, as in the example.

2.1 The One-Way Approach

We reformulate the problem as follows.

Let the Jacobian $\boldsymbol{J}$ have a sparsity structure with at most $p = nz_r$ nonzero elements per row. We wish to find a seed matrix $\boldsymbol{S} \in \mathbb{R}^{n\times p}$ such that $\boldsymbol{J}$ can be reconstructed from

$$\boldsymbol{B} = \boldsymbol{JS} \quad , \quad \boldsymbol{B} \in \mathbb{R}^{m\times p}$$

by solving m linear systems

$$\boldsymbol{b}^i = \boldsymbol{j}^i\boldsymbol{S} \quad , \quad = 1, \ldots, m \quad ,$$

where $\boldsymbol{b}^i$ is the ith row of $\boldsymbol{B}$. Suppose the ith row of the Jacobian has the structure

$$\begin{bmatrix} 0 & \boldsymbol{j}^i_{k_1} & 0 & \cdots & 0 & \boldsymbol{j}^i_{k_2} & \cdots & \boldsymbol{j}^i_{k_{max}} \end{bmatrix} .$$

Then we must solve the system

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{b}^i_{k_1} & \cdots & \boldsymbol{b}^i_{k_{max}} \end{bmatrix} &= \begin{bmatrix} 0 & \boldsymbol{j}^i_{k_1} & 0 & \cdots & 0 & \boldsymbol{j}^i_{k_2} & \cdots & \boldsymbol{j}^i_{k_{max}} \end{bmatrix} \boldsymbol{S} \\
&= \begin{bmatrix} \boldsymbol{j}^i_{k_1} & \cdots & \boldsymbol{j}^i_{k_{max}} \end{bmatrix} \begin{bmatrix} \boldsymbol{e}^T_{k_1}\boldsymbol{S} \\ \boldsymbol{e}^T_{k_1}\boldsymbol{S} \\ \vdots \\ \boldsymbol{e}^T_{k_{max}}\boldsymbol{S} \end{bmatrix} && (5) \\
&= \boldsymbol{z}\widehat{\boldsymbol{S}} \quad , \quad \widehat{\boldsymbol{S}} \in \mathbb{R}^{k_{max}\times p} \quad , && (6)
\end{aligned}$$

with $k_{max} < p$ variables and where $\boldsymbol{e}_k$ is the kth unit vector. The rows of $\widehat{\boldsymbol{S}}$ are the rows $\boldsymbol{s}^{k_1}, \ldots, \boldsymbol{s}^{k_{max}}$ of $\boldsymbol{S}$. Hence, the linear system might be over determined, but it is always consistent and thus solvable by its construction.

Example 2: Take the last line of the product $\boldsymbol{B} = \boldsymbol{JS}$ in (3). We must solve the system

$$\begin{bmatrix} \boldsymbol{b}^4_1 & 0 & \boldsymbol{b}^4_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{j}^4_3 & \boldsymbol{j}^4_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

or its transposed representation

$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{j}^4_3 \\ \boldsymbol{j}^4_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^4_1 \\ 0 \\ \boldsymbol{b}^4_3 \end{bmatrix} .$$

The second equation in the over determined system can obviously be deleted. □

Without loss of generality we may assume[3] that $\widehat{\boldsymbol{S}} \in \mathbb{R}^{p\times p}$. Now the question arises, What kind of matrices $\boldsymbol{S} \in \mathbb{R}^{m\times p}$ can ensure that an arbitrary collection of p different rows forms a submatrix $\widehat{\boldsymbol{S}} \in \mathbb{R}^{p\times p}$ that is regular?

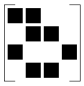
Compared with other sets of matrices with that regularity property, the Vandermonde matrices

$$\boldsymbol{V} = \left[\lambda_i^{j-1}\right], \quad i,j = 1\dots p, \quad \lambda_i \neq \lambda_{i'} ,$$

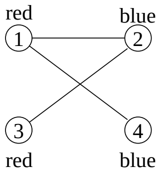
have the advantage that a very fast solution algorithm is applicable [12]. Its complexity is $\frac{5}{2}p^2$ versus $\frac{p^3}{3}$ for the LU-factorization of a dense arbitrary matrix of dimension p. The major disadvantage is the large condition number v_{cond} for which Vandermonde matrices are well known. Hence, Newsam and Ramsdell did not use Vandermonde matrices but another matrix construction based on Chebyshev polynomials. Though their condition numbers are moderate, they cannot compete with Vandermonde systems in terms of efficiency because of the costly general system solver. Fortunately, conditioning is not quite as important here as in the divided differences context, since the right-hand sides are obtained essentially with working precision.

We have investigated ways to reduce v_{cond} by suitable choices of abscissas. Even optimizing the conditioning of a square Vandermonde matrix is a difficult problem, as was shown in [10]. Here we face the additional difficulty that certain square submatrices need to be reasonably conditioned. This idea is currently under investigation. Another concept is the reduction of the number of different λ_i's from n to a number less than n. A small example should illustrate how graph coloring resurfaces.

<u>Example 3:</u> Take a Jacobian with the following sparsity structure:

$\boldsymbol{J} =$  .

It has a column incidence graph that can be colored with two different colors (use a minimal number of colors; attach one color to each node such that no two directly connected nodes have the same color).



One can now observe that rows in $\boldsymbol{S}$ with indices $(i = 1, \dots, n)$ that have the same color in the column incidence graph will never be grouped together in a submatrix $\widehat{\boldsymbol{S}}$ during the reconstruction process and thus can share the same λ_i. Although coloring the graph is rather expensive, this step has to be done only once in an overall calculation that requires the repeated evaluation of Jacobians at different points. Completing the second example, we get the seed matrix

$$\boldsymbol{S} = \begin{bmatrix} \lambda_1^0 & \lambda_1^1 \\ \lambda_2^0 & \lambda_2^1 \\ \lambda_1^0 & \lambda_1^1 \\ \lambda_2^0 & \lambda_2^1 \end{bmatrix}$$

[3] Introduce additional variables that are known to be zero.

and need to solve four linear equation systems with only two different system matrices $\begin{bmatrix} \lambda_1^0 & \lambda_1^1 \\ \lambda_2^0 & \lambda_2^1 \end{bmatrix}$ and $\begin{bmatrix} \lambda_2^0 & \lambda_2^1 \\ \lambda_1^0 & \lambda_1^1 \end{bmatrix}$ with two different right-hand sides for each matrix.

2.2 Columnwise by Forward

In this section we summarize the necessary computational steps for applying NR in the forward direction and add some statements about the complexity.

A1: We start with the detection of the closure of all possible sparsity patterns of the Jacobian $\boldsymbol{J}$. For this paper we use a routine from the AD tool ADOL-C. This routine propagates sparsity structures of gradients of all occurring intermediate quantities w.r.t. the independents by reexecuting a special representation, the *tape*, of the evaluation routine. The gradient sparsity structures are represented as bit patterns and manipulated with built-in C memory routines. Assuming that a logical OR between integers is less expensive than a floating-point operation, we can give the following bounds:

$$\text{time}(\text{bit\_pattern\_routine}) < \text{time}(\boldsymbol{f})\frac{n}{64}\ , \text{ and} \tag{7}$$

$$\text{space}(\text{bit\_pattern\_routine}) \leq \text{space}(\boldsymbol{f})\frac{n}{64}. \tag{8}$$

The $<$ in (7) becomes $\ll$ if $\boldsymbol{f}$ is highly nonlinear. The factor $\frac{n}{64}$ stems from the fact that each `double` variable (64 bits) in the C program is replaced by a bit pattern for the gradient of length n.

A2: Based on the evaluated sparsity structure of the Jacobian, we know the maximal number of nonzeros per row nz_r and set $p = nz_r$. The next task is the coloring of the column incidence graph, which is defined by the sparsity structure as well. The coloring has to be carried out only once, and we use an appropriate heuristic. The number of colors is the number of different λ_i to be chosen. By assigning λ_r to all rows $\boldsymbol{s}^i$, $i \in \mathcal{S}_r$, we construct the seed matrix $\boldsymbol{S}$.

A3: Using another routine from ADOL-C, we perform a first-order vector forward sweep [13]. This routine requires the argument point $\boldsymbol{x}_0$ from (1) and the previously defined seed matrix $\boldsymbol{S}$ as input and returns the product $\boldsymbol{JS}$, which is exact up to machine precision. The temporal complexity for this step is

$$C * p * \text{ops}(\boldsymbol{f}) \quad \text{with} \quad C \in [2, 5]\ .$$

A4: To reconstruct the full Jacobian, one has to solve m linear equation systems with Vandermonde matrices using the fast solver that takes only $\frac{5}{2}p^2$ operations.

As mentioned before, Steps A1 and A2 need be performed only the first time a Jacobian with the given structure is evaluated.

2.3 Rowwise by Reverse

The following method is dual to the method presented in Section 2.2, in the sense that one has to think in terms of columns where rows have been investigated and vice versa. Now we look at products $\boldsymbol{w}^T\boldsymbol{J}$ or $\boldsymbol{W}^T\boldsymbol{J}$ according to (f3). We use a seed matrix $\boldsymbol{W}^T$ that is multiplied from the left side of the Jacobian.

B1: Exactly the same as A1.

B2: Based on the evaluated sparsity structure of the Jacobian, we know the maximal number of nonzeros per column nz_c and set $q = nz_c$. We color the row incidence graph defined by the sparsity structure as well. The number of colors is the number of different λ_i to be chosen. By assigning λ_r to all rows $\boldsymbol{s}^i$, $i \in \mathcal{S}_r$, we construct the seed matrix $\boldsymbol{W}$ in the same fashion as $\boldsymbol{S}$.

B3: Using another routine from ADOL-C, we perform a first-order vector reverse sweep [13]. This routine requires certain information[4] stored during an evaluation at the argument point $\boldsymbol{x}_0$ from (1) (at the cost of $C'' \sim \mathrm{ops}(\boldsymbol{f})$) and the previously defined seed matrix $\boldsymbol{W}^T$ as input and returns the product $\boldsymbol{W}^T\boldsymbol{J}$, which is exact up to machine precision.The temporal complexity for this step is

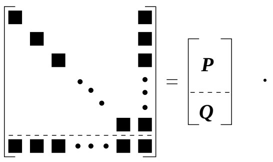
$$C' * q * \mathrm{ops}(\boldsymbol{f}) + C'' \quad \text{with} \quad C' \in [2, 5].$$

B4: Analogous to A4.

3 Combining Both Directions

In Section 1.1 we considered an arrow-like matrix structure as an example for which the forward and the reverse approach alone cannot be applied with reasonable effort. We use this example to explain how both schemes can be combined.

Example 4: Assume a square Jacobian of order $n = m$ that has a nonvanishing diagonal and a dense last row and column



If we split the calculation at the dashed line, we would need just 3 linear combinations, 2 in the forward way and 1 in reverse. For part $\boldsymbol{P}$ we could use ($p = 2$) the $n \times 2$ seed matrix

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

Since the optimal coloring number for the column incidence graph equals 2 ($= nz_r$), we do not need different λ_i. For the part $\boldsymbol{Q}$ ($q = 1$) the $1 \times m$ matrix $\boldsymbol{W}^T$ is

$$\begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}.$$

We do not have to take different λ_i for the analogous reason as stated earlier. The reconstruction of the full $\boldsymbol{P}$ from $\boldsymbol{JS}$ and $\boldsymbol{Q}$ from $\boldsymbol{W}^T\boldsymbol{J}$ is reduced as in CPR to a placement task. □

For arbitrary matrices we must minimize the sum $(p + q)$ of the number of linear combinations p and q in the forward and reverse mode, respectively. Therefore, we try to group all the rows of the Jacobian with at most p nonzero elements per row to part $\boldsymbol{P}$. Then q equals to the maximal number of nonzero elements per column in the remaining part $\boldsymbol{Q}(= \boldsymbol{J} \setminus \boldsymbol{P})$, where we consider matrices as ordered sets of their rows. We start with $p = 1$, calculate q and $p+q$, increase p by 1, and calculate q and $p+q$ again, and so forth.

[4]This is the so-called Taylor file; see [13].

The optimal combination of (p, q) that has been found in the described process will now be used for the splitting of $\boldsymbol{J}$. The whole algorithm can be described as follows:
<u>**C1:**</u> Exactly the same as A1.
<u>**C2:**</u> Detect the optimal pair (p, q) as defined above. Split $\boldsymbol{J}$ according to (p, q) into $\boldsymbol{P}$ and $\boldsymbol{Q}$.
<u>**C3:**</u> Perform A2 for the $(m-q) \times n$ matrix $\boldsymbol{P}$. Perform B2 for the $q \times n$ matrix $\boldsymbol{Q}$. Now we have two seed matrices, one $n \times p$ matrix $\boldsymbol{S}$ for $\boldsymbol{P}$ and one $q \times m$ matrix $\boldsymbol{W}^T$ for $\boldsymbol{Q}$.
<u>**C4:**</u> Perform A3 using $\boldsymbol{S}$. In the reconstruction process ignore all the $(m-p)$ rows that do not belong to part $\boldsymbol{P}$. Perform B3 using $\boldsymbol{W}^T$. In the reconstruction process of the m columns disregard all the row indices for the rows that do not belong to part $\boldsymbol{Q}$.

The complexity results given above still hold. For repeated Jacobian evaluations only C4 needs to be repeatedly performed.

4 Numerical Tests

As an established reference method for the given class of problems, we use the method of Coleman and Moré (CM). We compare CM to the runtime of three variants of Newsam and Ramsdell:
- forward (NRf),
- reverse (NRr), and
- combined (NRc).

Although we implemented the method using ADOL-C, these variants can be performed by any tool that provides forward and reverse mode. Since the compressed matrices are still sparse, one could refine the implementation by applying dynamically sparse data structures as in [4] or the "dirty vectors" introduced by B. Christianson in [6]. The reconstruction of the full Jacobian from the compressed matrix $\boldsymbol{B}$ is completely independent of automatic differentiation. We have already mentioned the general conditioning problem with the Vandermonde systems. The first tests of the precision of the results showed clearly that the graph coloring to reduce the number of different λ_i cannot be omitted. We have used three different heuristics:
- largest first ordering,
- smallest last ordering, and
- incidence degree ordering,

where the smallest coloring number was then taken [8]. The test suite consists of five problems from the MINPACK-2 test collection [3], a molecular distance geometry problem (mdg) [15], and a matrix with arrow structure (mas). The MINPACK problems [4] are
- flow in a channel (fic),
- solid fuel ignition (sfi),
- flow in a driven cavity (fdc),
- swirling flow between disks (sfd), and
- incompressible elastic rods (ier).

4.1 The MINPACK Problems

All problems are square in the sense that the number of independent and dependent variables are equal. They were executed with $n = m = 1000, 1500, 2000, 3000$. The sparsity patterns of the Jacobians are depicted in Figures 1–5.

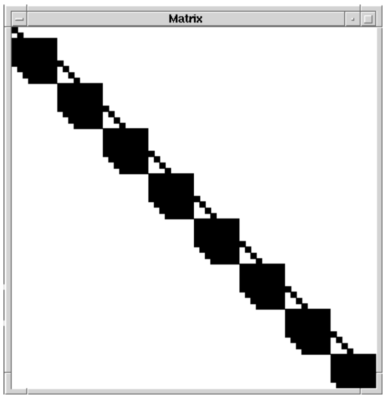


Fig. 1: *Flow in a channel* (64×64), (fic)

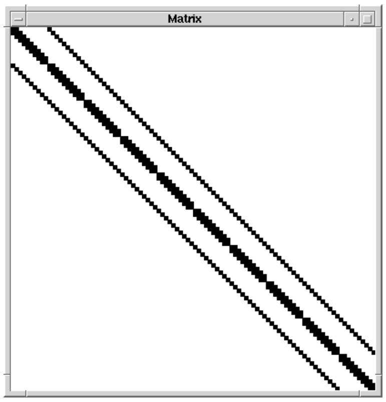


Fig. 2: *Solid fuel ignition* (100×100), (sfi)

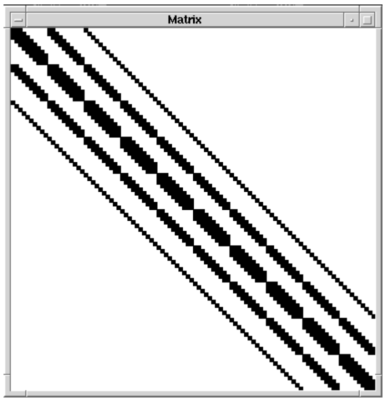


Fig. 3: *Flow in a driven cavity* (100×100), (fdc)

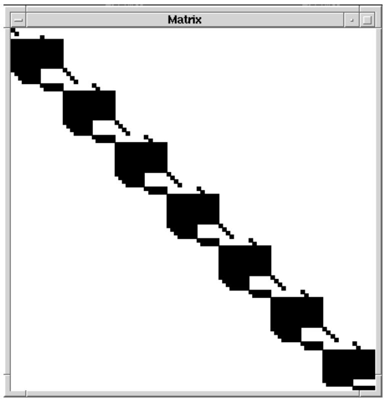


Fig. 4: *Swirling flow between disks* (98×98), (sfd)

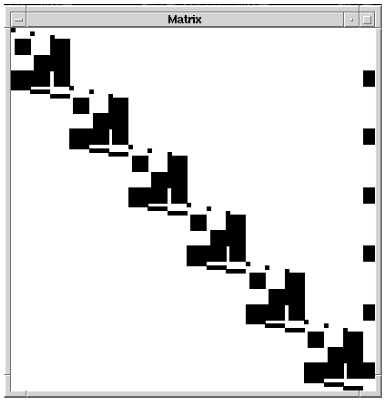


Fig. 5: *Incompressible elastic rods* (93×93), (ier)

The numbers of nonzero elements and colors remained constant for the different values of n, since the sparsity is repetitive, except for (ier). Table 1 contains the coloring numbers for (CM) and nz_r, nz_c, and the coloring numbers of the row/column incidence graphs for the Newsam-Ramsdell approaches. The table shows that the combined method does not yield any advantage on this particular set of test problems. Every attempt to apply the combined method simplifies to forward or (for (sfd)) reverse. For the banded symmetric matrices like these in (sfi) and (fdc), this result is obvious. For (fic) one can see that no splitting of $\boldsymbol{J}$ can produce a sum $p+q<9$.

TABLE 1: *Coloring numbers (co.) and row/column lengths*

Problem	CM	NRf/co.	NRr/co.	NRc=NRf+NRr	f_co./r_co.
fic	9	9/9	9/9	9=9+0	9/-
sfi	6	5/6	5/6	5=5+0	6/-
fdc	**18**	**13/18**	**13/18**	**13=13+0**	**18/-**
sfd	14	14/14	13/13	13=0+13	-/13
ier	17	17/17	□	17=17+0	17/-

Because of the complex structure of (sfd) and (ier) there is no simple explanation for the results that are returned correctly by the program. The □ indicates that (ier) cannot sensibly be treated with NRr. The structure has nonzero blocks in the last columns of $\boldsymbol{J}$, and their number is proportional to n. Hence, nz_c grows rapidly with n, and thus NRr becomes impracticable. The (fdc) problem shows the largest difference between CM and NR, but the coloring number is still in the same range as nz_r.

4.2 The Molecular Distance Geometry Problem

The (mdg) problem was chosen because of its property that the heuristics return coloring numbers proportional to the problem dimension, while nz_r does not change. The underlying function $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ describes interactions between atoms in a molecule. For a given n ($= 3*$ number of atoms), the function detects $m \geq n$. Here m is the number of interactions depending on a given maximal distance parameter d. Figures 6–7 illustrate the sparsity structure.

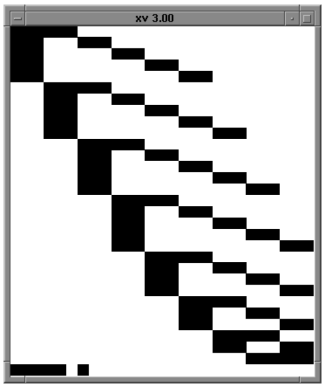


FIG. 6: $d=5$; (31×28)

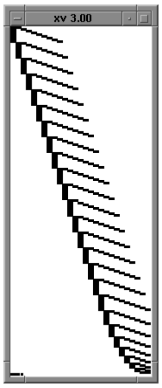


FIG. 7: $d=9$; (199×81)

TABLE 2: *Coloring numbers (co.) and row/column lengths*

n	d	m	CM	NRf/co.	NRr/co.
240	15	1081	48	6/48	30/30
375	20	2291	63	6/63	40/40
450	25	3426	78	6/78	50/50
525	30	4786	93	6/93	60/60

The test calculations were done with large n and m. The increase of d with growing dimension has a direct impact on the coloring number and on nz_c and is in reality caused by the physics of the problem. Unfortunately, this again is not suitable to prove the advantages of the combined mode. We still do not have the situation of some almost dense rows and columns.

4.3 Runtime

The graphs in Figure 8 show time ratios for the MINPACK problems. They illustrate what we have already expected. Since the coloring number is not much larger than nz_r or nz_c respectively, the method of Newsam and Ramsdell cannot prove to be a big advantage. $\boldsymbol{B} \rightarrow \boldsymbol{J}$ denotes the reconstruction of $\boldsymbol{J}$ from $\boldsymbol{B}$.

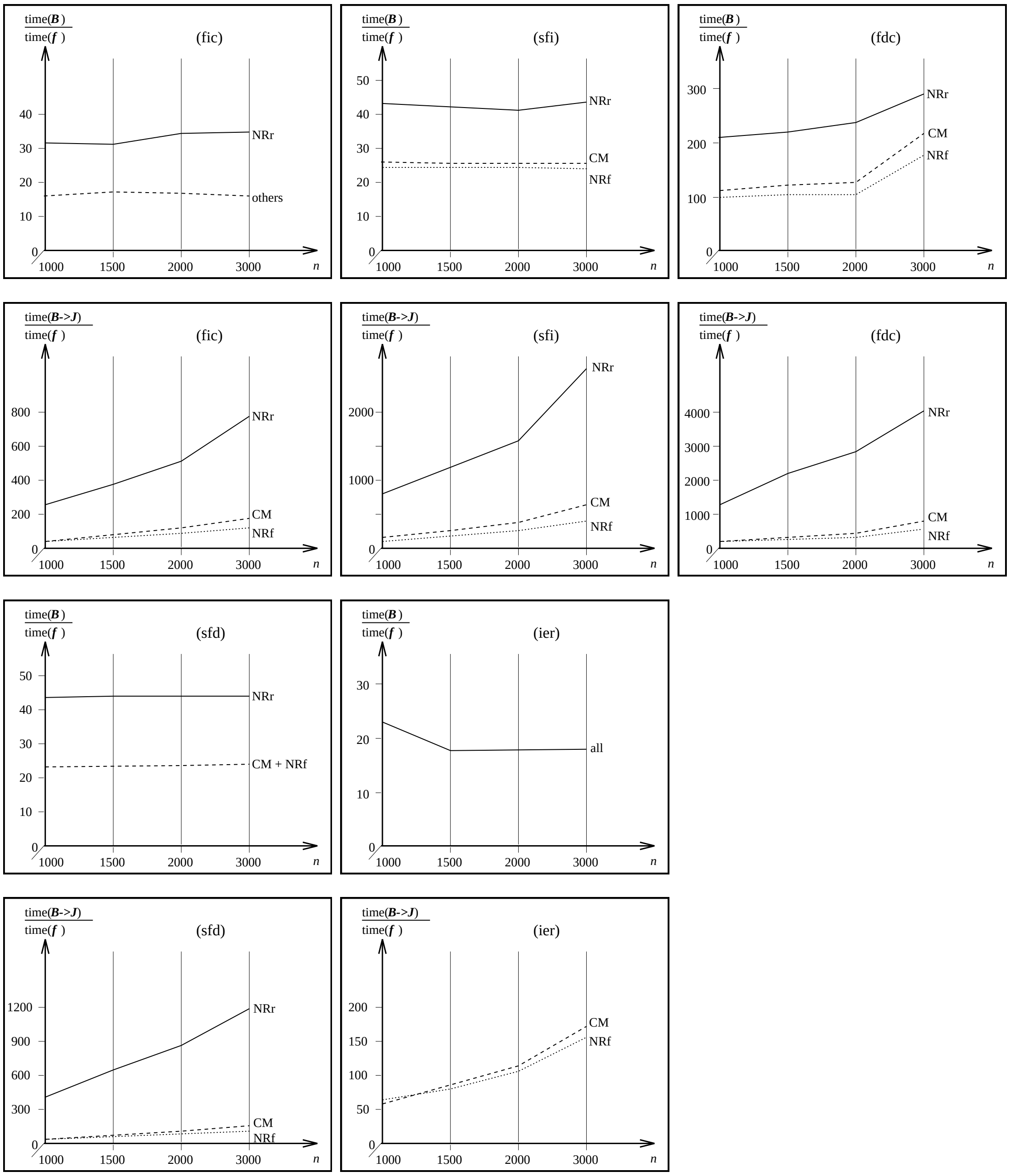


Fig. 8: *Runtime ratios for the MINPACK problems*

Nevertheless, it is remarkable that the solution of the linear systems in the reconstruction process does not require a significantly larger amount of runtime compared with the placement routine of CM. A possible explanation is that the actual computing is dominated by the time for paging. Rather surprising results are the large time ratios for NRr. Besides the influence of C'' in B3 we assume a major slowdown in the reconstruction process B4 because in B4 the implementation performs matrix operations columnwise, while they are stored rowwise (as usual in C). A better adjusted data structure might overcome this effect.

As opposed to the five MINPACK problems, the molecular distance geometry problem makes the benefits of NR obvious.

TABLE 3: *Runtime comparison for evaluating* $\boldsymbol{B}$ *for (mdg)*

$(m \times n)$	CM	NRf	NRr
1081 $\times$240	0.94	0.19	1.18
2291 $\times$375	2.57	0.40	3.18
3426 $\times$450	4.48	0.59	5.79
4787 $\times$525	9.65	0.84	11.29

Table 3 shows the time needed to evaluate $\boldsymbol{B}$. Although NRr still cannot compete with CM, as is clear from the sparsity structure, NRf is superior. Table 4 confirms the previous statements about the runtime for the reconstruction/placement process.

TABLE 4: *Runtime comparison for reconstructing* $\boldsymbol{J}$ *for (mdg)*

$(m \times n)$	CM	NRf	NRr
1081 $\times$240	0.45	0.38	7.57
2291 $\times$375	1.43	1.14	32.86
3426 $\times$450	2.56	1.97	72.86
4787 $\times$525	8.77	9.95	142.07

The last problem, the matrix with arrow structure (mas), satisfied the expectations with respect to the combined method NRc. The minimal value $p + q = 3$ makes all other effects negligible.

4.4 Numerical Stability

As stated earlier, the coloring of the incidence graphs in NR should not be omitted. After some tests, the λ_i for the seed matrices were chosen as the roots of Chebyshev polynomials, for (sfi) and (fdc) and equidistantly in $[-1, +1]$ for (mdg). An optimization algorithm for the selection of the λ_i that takes account of the actual grouping of Vandermonde rows is currently under development. The relative error of $\boldsymbol{J}$ was evaluated in several cases, see Table 5.

TABLE 5: *Relative errors*

problem	dimension	relative error
(sfi)	1000 $\times$ 1000	$6.14 * 10^{-16}$
	2000 $\times$ 2000	$6.03 * 10^{-16}$
(fdc)	1000 $\times$ 1000	$4.59 * 10^{-12}$
	2000 $\times$ 2000	$2.05 * 10^{-11}$
(mdg)	1081 $\times$ 240	$2.72 * 10^{-9}$
	2291 $\times$ 375	$1.28 * 10^{-8}$
	3426 $\times$ 450	$1.47 * 10^{-8}$
	4787 $\times$ 525	$4.63 * 10^{-8}$

In contrast to divided differences, the application of automatic differentiation makes the method of Newsam and Ramsdell with Vandermonde matrices reliable for coloring numbers up to 20. Higher coloring numbers make v_{cond} still critical, and the precision may suffer. We expect a noticeable improvement in numerical stability from the optimization algorithm for the λ_i. Instead of Vandermonde matrices one might use other seed matrices that yield better accuracy at slightly increased reconstruction costs.

References

[1] *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, Proceedings of the First SIAM Workshop on Automatic Differentiation, A. Griewank, G. F. Corliss eds., SIAM 1991.

[2] B. M. Averick, C. H. Bishof, A. Carle, A. Griewank and J. J. Moré, *Computing Large Sparse Jacobian Matrices Using Automatic Differentiation*, SIAM J. Sci. Comp., vol. 15(2), 1994, pp. 258–294.

[3] B. M. Averick, R. G. Carter, J. J. Moré and G. Xue, *The MINPACK-2 Test Problem Collection*, Argonne National Laboratory, preprint MCS-P153-0692, 1992.

[4] C. H. Bishof, A. Bouaricha, A. Carle and P. M. Khademi, *Efficient Computation of Gradients and Jacobians by Transparent Exploitation of Sparsity in Automatic Differentiation*, Argonne National Laboratory, preprint MCS-P519-0595, 1995.

[5] C. H. Bishof, A. Bouaricha, P. M. Khademi and J. J. Moré, *Gradients in Large-Scale Optimization using Automatic Differentiation*, Argonne National Laboratory, preprint MCS-P488-0195, 1995.

[6] B. Christianson, *Sharing Storage Using Dirty Vectors*, in this proceedings

[7] T. F. Coleman and J. Cai, *The Cyclic Coloring Problem and Estimation of Sparse Hessians*, SIAM J. Alg. Disc. Meth., vol. 7(2), 1986, pp. 221–235.

[8] T. F. Coleman and J. J. Moré, *Estimation of Sparse Jacobian Matrices and graph coloring problems*, SIAM J. Numer. Anal., vol. 20(1), 1983, pp. 187–209.

[9] A. R. Curtis, M. J. D. Powell and J. K. Reid, *On the Estimation of Sparse Jacobian Matrices*, SIAM J. Inst. Math. Appl., vol. 13, 1974, pp. 117–119.

[10] W. Gautschi, *Optimally Conditioned Vandermonde Matrices*, Numer. Math., vol. 24, 1975.

[11] U. Geitner, *Automatische Berechnung von dünnbesetzten Jacobimatrizen nach dem Ansatz von Newsam-Ramsdell*, diploma theses, TU Dresden, Inst. of Scientific Computing, 1995.

[12] G. H. Golub and C. F. van Loan, *Matrix Computation*, John Hopkins University Press, 1989.

[13] A. Griewank, D. Juedes and J. Utke, *ADOL-C, A Package for the Automatic Differentiation of Algorithms Written in C/C++*, to appear in TOMS 1996.

[14] W. Gropp, S. Balay, L. C. McInnes, B. Smith, *PETSc 2.0 Users Manual*, Argonne National Laboratory, draft 11/1995.

[15] J. J. Moré and Z. Wu, *ϵ-Optimal Solutions to Distance Geometry Problems via Global Continuation*, Argonne National Laboratory, preprint MCS-P520-0595, 1995.

[16] G. N. Newsam and J. D. Ramsdell, *Estimation of Sparse Jacobian Matrices*, SIAM J. Alg. Disc. Meth., vol. 4(3), 1983, pp. 404–417.

[17] D. Shiriaev, A. Griewank and J. Utke, *A User Guide to ADOL-F: Automatic Differentiation of FORTRAN Codes*, TU Dresden, Inst. of Scientific Computing, preprint 1995.

[18] T. Steihaug and S. Hossain, *Computing a sparse Jacobian Matrix by Rows and Columns*, University of Bergen, Department of Computer Science, 1995.