1 Introduction

Simulations using partial differential equations (PDE) and PDE-constrained optimizations are employed to attack real-world problems in scientific computing. In both areas, solving systems of linear equations with Jacobian matrices as coefficient matrices is mandatory. To solve these linear systems, sparse Jacobian matrices demand a matrix-free iterative solver with access to (transposed) Jacobian matrix-vector products. Automatic differentiation is a technique to provide not only Jacobian matrices but also these products. The great benefit of these Jacobian matrix-vector products is that there is no need to store the nonzero elements of the Jacobian matrix. If a Jacobian matrix exceeds the main memory, it is essential to use matrix-free iterative solvers. Memory consumption and computational effort are restricting resources in simulation and optimization. In this thesis, the sparsity of Jacobian matrices is exploited to decrease both resources.

Exploiting the sparsity of Jacobian matrices is important to decrease the computational effort. Determining all nonzero elements is denoted as full Jacobian computation, opposed to the partial Jacobian computation where only a subset of the nonzero elements is computed. Determining such a subset is important for employing the preconditioning techniques in the second part of this thesis.

Nonzero elements may be determined row- or column-wise with automatic differentiation. Either rows or columns can be combined to linear combinations to reduce the computational effort. There are special classes of matrices for which the number of linear combinations can be reduced further. Therefore, the nonzero elements can be determined by rows and columns. These rows and columns are combined to linear combinations of rows and linear combinations of columns. The arrow-shaped matrix is a good example for this matrix class. These reductions of linear combinations can be modeled as combinatorial optimization problems, in particular, graph coloring problems. That is, each linear combination is associated to a color. Combining either rows or columns is denoted as one-sided coloring, whereas combining rows and columns is denoted as two-sided coloring.

Reducing the number of colors for full and partial Jacobian computation is studied in the first main part of this thesis. In particular, algorithms to solve the resulting graph coloring problems are developed. Regular Cartesian grids are a graph class occurring in stencil-based computations. A sub-exponential exact coloring algorithm is introduced to determine minimal colorings for full and partial Jacobian computation. This algorithm takes advantage of the grid and stencil properties by employing a divide-and-conquer scheme and separators derived from Lipton-Tarjan separators. Thereafter, since the runtime depends directly on the grid size, a grid size-independent approach with linear time complexity is invented. A small grid is

colored and, then, a coloring information—if available—is extracted which is sufficient to color a larger grid of arbitrary size with the smallest number of colors. Although this approach is not universal, it works for all considered stencils. Following one-sided colorings to exploit Jacobian matrices in stencil-based computations, the next topic is two-sided colorings for general graphs. An algorithm for partial Jacobian computation is introduced. Thereafter, several classes of Jacobian matrices are considered to assess whether there is a two-sided coloring with less colors than a minimal one-sided coloring.

For matrix-free iterative solvers, the access to Jacobian matrix-vector products is sufficient without assembling the full matrix. Unfortunately, standard preconditioning techniques expect that all nonzero elements of a Jacobian matrix are available. Using preconditioners increases the convergence behavior and speeds up solving systems of linear equations. If the available memory is a limiting resource, storing all these elements is often impossible. Cullum and Tuma [21] proposed to restrict the input for the preconditioning method to a subset of the nonzero elements of a Jacobian matrix. However, they do not address how to choose this subset. Gebremedhin, Pothen, and Manne [27] introduced the coloring definition for the partial Jacobian computation and mentioned its usage for preconditioning. When a subset of the Jacobian elements—solely for preconditioning purposes—is determined, these nonzero elements may be stored and, afterwards, the preconditioning techniques are restricted to these elements.

Combining preconditioning techniques with the partial Jacobian computation is the topic of the second half of this thesis. It is assumed that the sparsity pattern of the Jacobian matrix is available. This information is employed to determine a preconditioner. A set of nonzero elements is suggested by domain experts and, then, additional nonzero elements are chosen to speed up solving systems of linear equations. When these elements are chosen, the limited memory and the computational effort must be taken into account. To determine the initial nonzero elements, a coloring with a specific number of colors is needed. Additional nonzero elements are chosen without increasing this number of colors and without exceeding the available memory. The nonzero elements are categorized to different classes. Several algorithms are developed to choose nonzero elements. At the end, a couple of additional nonzero elements are omitted to obtain a structure which is beneficial for solving the preconditioned system of linear equations in parallel. The aim is to reduce the degree of dependence between the employed processors.

The main contributions of this thesis are in the fields of coloring algorithms for sparsity exploitation of Jacobian matrices as well as the combination of preconditioning and partial Jacobian computation. In the first part, a sub-exponential exact coloring algorithm and a linear-time algorithm which is independent of the original grid size are introduced for full and partial Jacobian computation in stencil-based computations. For general graphs, a two-sided coloring algorithm for partial Jacobian computation is given. Thereafter, several classes of Jacobian matrices are considered to assess whether there is a two-sided coloring with less colors than a minimal one-sided coloring. In the second part, the combination of preconditioning and the partial

Jacobian computation is a completely new approach. It includes the categorization of nonzero elements and choosing specific nonzero elements for preconditioning.

This thesis is structured as follows: Already known graph models associated to sparse Jacobian matrices are described in Chap. 2. These models comprise a bipartite graph model for general Jacobian matrices and a regular Cartesian grid for Jacobian matrices arising from stencil-based computations. Furthermore, the optimization problems concerning the exploitation of Jacobian matrices and a first introduction to graph coloring algorithms is given. Moreover a consistent notation for the thesis is introduced. In Chap. 3, an exact sub-exponential coloring algorithm on regular grids for full and partial Jacobian computation in stencil-based computation is presented. Another new coloring algorithm to compute minimal colorings is presented to reduce the complexity of the previous algorithm. The determination of the coloring information is independent of the original grid size. For general graphs, a coloring algorithm for the partial Jacobian computation is described which is also applicable for the full Jacobian computation. Thereafter, for some matrix classes, the reduction in the number of colors by using two-sided colorings compared to one-sided colorings is evaluated. In the following chapter, a preconditioning technique is combined with the partial Jacobian computation. The preconditioning for solving systems of linear equations is introduced and motivated. Afterwards, the nonzero elements of Jacobian matrices are classified. For every class, algorithms are introduced to determine different subsets of required nonzero elements. At the end, solving preconditioned systems of linear equations in parallel is considered. The usage of the required elements for preconditioning is evaluated by the number of matrix-vector products, number of nonzero elements, and number of colors. In Chap. 5, several applications from science and engineering are considered to show the practical relevance for using the previously described techniques, in particular, the graph coloring and preconditioning. This thesis closes with a concluding summary.

2 Exploiting sparsity in Jacobian computation

Before describing the progress in coloring algorithms and partial Jacobian computation for preconditioning, we give a short survey of the state of the art in exploiting the sparsity of Jacobian matrices. After presenting the Jacobian computation for all nonzero elements, we explain how to reduce the computational effort when only a subset of these nonzero elements is determined. Thereafter, these exploitation techniques are modeled as graph coloring problems. Finally, the focus is on Jacobian matrices occurring from discretizing domains with stencil-based methods. In contrast to general graphs, regular grids are employed.

2.1 Full Jacobian computation

Given a program implementing some mathematical function

$$f(x) : \mathbb{R}^n \ \rightarrow \ \mathbb{R}^m, \tag{2.1}$$

the derivative of the vector-valued function f with respect to some vector $x \in \mathbb{R}^n$ in the direction of a vector $s \in \mathbb{R}^n$ is defined by

$$\frac{\partial f}{\partial x} s = \lim_{h \to 0} \frac{f(x + hs) - f(x)}{h}. \tag{2.2}$$

Let $A := \partial f / \partial x$ denote the $m \times n$ Jacobian matrix whose columns are given by

$$A = [a_1 a_2 \cdots a_n].$$

Then, by choosing $s \in \{0, 1\}^n$ as a binary vector, any sum of columns a_j can be computed where the jth entry of s is nonzero, i.e.,

$$As = \sum_{j \text{ with } s_j = 1} a_j.$$

Moreover, the product of the Jacobian matrix A and some $n \times p$ *seed matrix* S can be approximated by $p + 1$ evaluations of the function f using divided differencing. Similarly, the forward mode of automatic differentiation is capable of computing that product, $A \cdot S$, without truncation error using $p+1$ times the time needed to evaluate f. Therefore, p indicates a rough measure of the time needed to compute the Jacobian matrix.

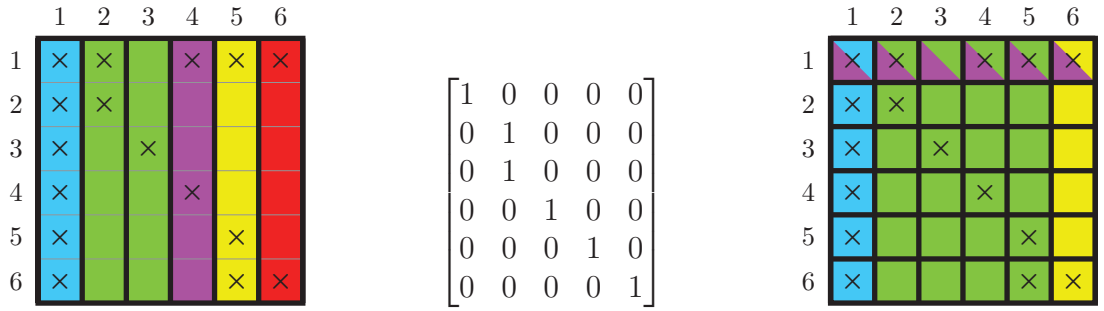

Figure 2.1: (a) Sparsity pattern of Jacobian matrix A with $p = 5$ column groups ({1}, {2,3}, {4}, {5}, {6}). (b) Seed matrix S corresponding to column groups in (a). (c) Bidirectional partition of A with $p = 4$ groups (row: {1}, column: {1}, {2,...,5}, {6}).

Automatic differentiation (AD) [31, 55] comprises a set of techniques to transform the function f into another function which computes the derivative. The two major modes of AD are the forward mode to compute linear combinations of the columns of a Jacobian matrix and the reverse mode to compute linear combinations of the rows. Using AD in a naïve way in the forward or reverse mode, we need $p = n$ or $p = m$ directional derivatives, respectively.

The sparsity pattern of a Jacobian matrix A can be determined beforehand or is known due to the discretization of the underlying physical model. This information can be used to decrease the number of directional derivatives p by combining several columns to a linear combination without losing values. This technique is called *column compression*. The idea to reduce p—and hence the time to compute all nonzero elements of a sparse Jacobian matrix—consists of partitioning the columns of the Jacobian matrix into groups of those columns whose sum contains all the nonzero elements of the columns in that group [22]. The definition and the corresponding problem for row groups is straightforward. The property characterizing such a column group is introduced in the following definition:

Definition 2.1. Two columns a_i and a_j are *structurally orthogonal* if and only if they do not have any nonzero element in the same row, i.e.,

$$a_i \perp a_j \quad :\Longleftrightarrow \quad \nexists k : \; a_{k,i} \neq 0 \wedge a_{k,j} \neq 0.$$

In the example given in Fig. 2.1(a), the columns a_2 and a_3 are structurally orthogonal since there is no row in which both columns have a nonzero element. So, the sum $a_2 + a_3$ contains all nonzero elements of these two columns. The columns a_1 and a_2 are not structurally orthogonal, so-called *structurally non-orthogonal*, because both have nonzero elements in rows 1 and 2. The combinatorial optimization problem to find a minimal p is formulated as follows:

Problem 2.2. *Given a Jacobian matrix A, partition its columns into a minimal number of groups of structurally orthogonal columns. More precisely, find a binary $n \times p$ matrix S such that all nonzero elements of A are contained in the matrix-matrix product $A \cdot S$ and the number of columns p, representing the number of groups, is minimized.*

A solution to that problem, a so-called *unidirectional partition*, may not be unique. For the illustrating example in Fig. 2.1(a), a solution is indicated by using different colors. This unidirectional partition consists of $p = 5$ column groups $\{1\}$, $\{2, 3\}$, $\{4\}$, $\{5\}$, and $\{6\}$. Compared to the naïve way, this is a reduction of one directional derivative. The corresponding seed matrix is given in Fig. 2.1(b).

A further reduction is possible using a combination of rows and columns, a so-called *bidirectional partition*. This is beneficial if there is at least one column and one row which are pretty dense, i.e., there are a lot of nonzero elements in this column or row, respectively. An obvious example for the bidirectional partitioning is the arrow shaped matrix with one full row, one full column, and a full diagonal. The combinatorial optimization problem to find a minimal number of groups is formulated as follows:

Problem 2.3. *Given a Jacobian matrix A, find a binary $p_r \times m$ matrix S_r and a binary $n \times p_c$ matrix S_c such that all nonzero elements of A are contained in the matrix-matrix products $S_r \cdot A$ and $A \cdot S_c$, and the number of rows and columns $p = p_r + p_c$, representing the number of groups, is minimized.*

An illustrating example for the bidirectional partitioning is depicted in Fig. 2.1(c). All nonzero elements in the first row are covered by the row group $\{1\}$. Now, the column groups can be determined without taking the nonzero elements of the first row into account. The columns a_2 and a_4 can be part of the same column group, although the values of the elements $a_{1,2}$ and $a_{1,4}$ are destroyed. The reason is that these elements are determined by the row group $\{1\}$. The bidirectional partition consists of the row group $\{1\}$ and the column groups $\{1\}$, $\{2, \ldots, 5\}$, and $\{6\}$. That is, the bidirectional partition consists of $p = p_r + p_c = 1 + 3$ groups. This is a reduction of one group compared to the unidirectional partition.

2.2 Partial Jacobian computation

Rather than computing all nonzero elements of the Jacobian matrix A, only a proper subset of the nonzero elements should be determined. Computing such a set of *required* nonzero elements R is called *partial Jacobian computation* as opposed to *full* Jacobian computation where all nonzero elements are computed. The remaining nonzero elements are called *non-required* elements. Gebremedhin, Manne, and Pothen [27] introduced the rules for the partial Jacobian computation. In [40–42], the assumption was validated that the partial Jacobian computation reduces the number of groups p compared to the full Jacobian computation.

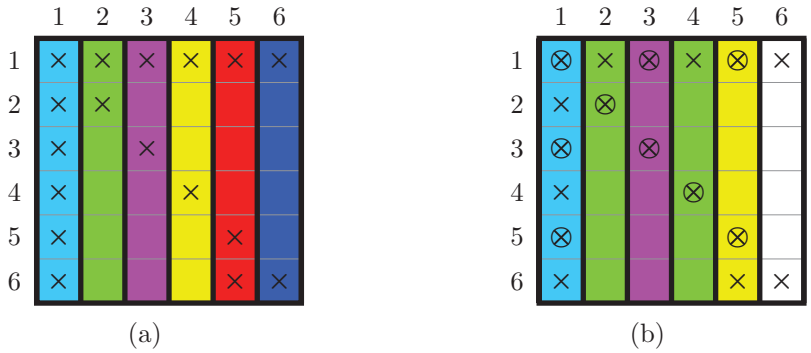

Figure 2.2: (a) Full Jacobian computation for Jacobian matrix with $p = 6$ column groups. (b) Partial Jacobian computation for Jacobian matrix from (a) with $p = 4$ column groups with required elements denoted by symbol $\otimes$ and non-required elements by symbol $\times$.

For the unidirectional partitioning, the property how to combine columns is introduced in the following definition:

Definition 2.4. Two columns a_i and a_j are *partially structurally orthogonal* with respect to the required elements R if and only if they do not have a nonzero element in the same row where at least one nonzero element is required, i.e.,

$$a_i \perp_R a_j \quad :\Longleftrightarrow \quad \nexists k : \; a_{k,i} \neq 0 \wedge a_{k,j} \neq 0 \wedge (a_{k,i} \in R \vee a_{k,j} \in R).$$

We consider the following Jacobian matrix to explain the column compression in partial Jacobian computation: Let the sparsity pattern of the 6×6 Jacobian matrix A be given by an arrow-shaped structure with an additional nonzero element $a_{6,5}$. This pattern is depicted in Fig. 2.2(a). All columns of this Jacobian matrix are structurally non-orthogonal due to row 1 which is full of nonzero elements. Six column groups are needed for the full Jacobian computation. This changes when we define a subset R with the required elements indicated by the symbol $\otimes$ in Fig. 2.2(b). The required elements in the first row are $a_{1,1}$, $a_{1,3}$, and $a_{1,5}$. The non-required elements are indicated by the symbol $\times$. Due to the fact that we are not interested in the non-required elements $a_{1,2}$ and $a_{1,4}$, the columns a_2 and a_4 are partially structurally orthogonal and can be combined into one column group. The column a_6 does not contain a required element and therefore is not part of any column group.

The adaption of the optimization problems 2.2 and 2.3 for the full Jacobian computation to the partial Jacobian computation is straightforward regarding the modified definition of partial structural orthogonality. We skip the definition of the bidirectional partitioning for partial Jacobian computation in this section and refer to the explanation in the next section with the corresponding graph model.

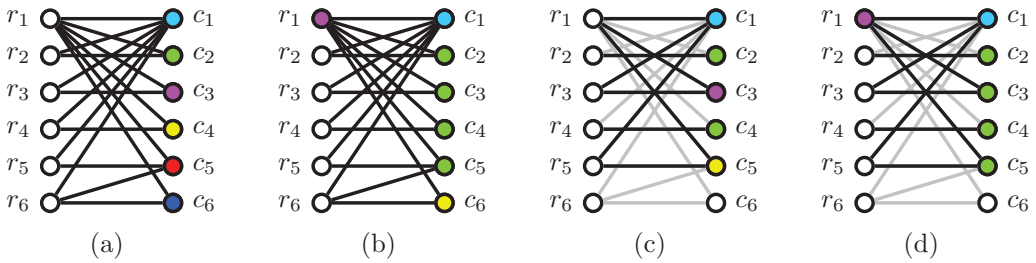

Figure 2.3: (a) Unidirectional partition and (b) bidirectional partition of the bipartite graph associated to Jacobian matrix in Fig. 2.2 for full Jacobian computation. (c) Unidirectional partition and (d) bidirectional partition for partial Jacobian computation with required edges indicated in black and non-required edges in gray.

2.3 Graph representation

Coleman and Moré [18] were the first authors who modeled the computation of sparse Jacobian matrices by graphs. In particular, they introduced the column intersection graph. Since then, various graph models have been used to describe different sparsity-exploiting derivative computations [16, 17, 33–35]. Coleman and Verma [19] introduced a bipartite graph model. In contrast to the other graph models, this model is sufficient for the bidirectional partitioning in full and partial Jacobian computation. Therefore, we mainly consider the bipartite graph throughout this thesis.

The sparsity pattern of a Jacobian matrix A can be represented as an undirected bipartite graph $G = (V_r \uplus V_c, E)$ which consists of m vertices in V_r to represent the rows and n vertices in V_c to represent the columns. The symbol $\uplus$ indicates that the sets V_r and V_c are disjoint. The vertex $r_i \in V_r$ corresponds to row i and the vertex $c_j \in V_c$ to column j. There is an edge $(i, j) \in E$ if and only if a nonzero element $a_{i,j} \neq 0$ exists. The bipartite graph G associated to the matrix in Fig. 2.2(a) consists of 6 vertices in V_r to represent the rows, 6 vertices in V_c to represent the columns, and 17 edges connecting vertices from V_r and V_c representing the nonzero elements. The graph G is depicted in Fig. 2.3.

The unidirectional partitioning of columns can be modeled as a graph coloring problem on the bipartite graph. The following definition is used to partition the column vertices into different groups.

Definition 2.5. Two column vertices c_i and c_j are *structurally orthogonal* if and only if they are not connected by a path of length 2, i.e.,

$$c_i \perp c_j \quad :\Longleftrightarrow \quad \nexists r_k \in V_r : \ (r_k, c_i) \in E \ \wedge \ (r_k, c_j) \in E.$$

Two vertices are *distance-k neighbors*, if they are connected by a path of length k. For column compression, all column vertices have to be colored so that different colors are assigned to any pair of distance-2 neighbors. This rule is given more precisely in the following definition: