3.4 STORAGE SCHEMES

In order to take advantage of the large number of zero elements, special schemes are required to store sparse matrices. The main goal is to represent only the nonzero elements, and to be able to perform the common matrix operations. In the following, Nz denotes the total number of nonzero elements. Only the most popular schemes are covered here, but additional details can be found in books such as Duff, Erisman, and Reid [77].

The simplest storage scheme for sparse matrices is the so-called coordinate format. The data structure consists of three arrays: (1) a real array containing all the real (or complex) values of the nonzero elements of A in any order; (2) an integer array containing their row indices; and (3) a second integer array containing their column indices. All three arrays are of length Nz, the number of nonzero elements.

Example 3.7 The matrix

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

will be represented (for example) by

AA	12.	9.	7.	5.	1.	2.	11.	3.	6.	4.	8.	10.
JR	5	3	3	2	1	1	4	2	3	2	3	4
JC	5	5	3	4	1	4	4	1	1	2	4	3

In the above example, the elements are listed in an arbitrary order. In fact, they are usually listed by row or columns. If the elements were listed by row, the array JC which contains redundant information might be replaced by an array which points to the beginning of each row instead. This would involve nonnegligible savings in storage. The new data structure has three arrays with the following functions:

- A real array AA contains the real values a_{ij} stored row by row, from row 1 to n. The length of AA is Nz.
- An integer array JA contains the column indices of the elements a_{ij} as stored in the array AA. The length of JA is Nz.
- An integer array IA contains the pointers to the beginning of each row in the arrays AA and JA. Thus, the content of $IA(i)$ is the position in arrays AA and JA where the i-th row starts. The length of IA is $n+1$ with $IA(n+1)$ containing the number $IA(1) + Nz$, i.e., the address in A and JA of the beginning of a fictitious row number $n+1$.

Thus, the above matrix may be stored as follows:

AA	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JA	1	4	1	2	4	1	3	4	5	3	4	5
IA	1	3	6	10	12	13						

This format is probably the most popular for storing general sparse matrices. It is called the *Compressed Sparse Row* (CSR) format. This scheme is preferred over the coordinate scheme because it is often more useful for performing typical computations. On the other hand, the coordinate scheme is advantageous for its simplicity and its flexibility. It is often used as an "entry" format in sparse matrix software packages.

There are a number of variations for the Compressed Sparse Row format. The most obvious variation is storing the columns instead of the rows. The corresponding scheme is known as the *Compressed Sparse Column* (CSC) scheme.

Another common variation exploits the fact that the diagonal elements of many matrices are all usually nonzero and/or that they are accessed more often than the rest of the elements. As a result, they can be stored separately. The *Modified Sparse Row* (MSR) format has only two arrays: a real array AA and an integer array JA. The first n positions in AA contain the diagonal elements of the matrix in order. The position $n+1$ of the array AA is not used, but may sometimes be used to carry other information concerning the matrix. Starting at position $n+2$, the nonzero elements of AA, excluding its diagonal elements, are stored by row. For each element $AA(k)$, the integer $JA(k)$ represents its column index on the matrix. The $n+1$ first positions of JA contain the pointer to the beginning of each row in AA and JA. Thus, for the above example, the two arrays will be as follows:

AA	1.	4.	7.	11.	12.	*	2.	3.	5.	6.	8.	9.	10.
JA	7	8	10	13	14	14	4	1	4	1	4	5	3

The star denotes an unused location. Notice that $JA(n) = JA(n+1) = 14$, indicating that the last row is a zero row, once the diagonal element has been removed.

Diagonally structured matrices are matrices whose nonzero elements are located along a small number of diagonals. These diagonals can be stored in a rectangular array `DIAG(1:n,1:Nd)`, where `Nd` is the number of diagonals. The offsets of each of the diagonals with respect to the main diagonal must be known. These will be stored in an array `IOFF(1:Nd)`. Thus, the element $a_{i,i+\text{ioff}(j)}$ of the original matrix is located in position (i,j) of the array `DIAG`, i.e.,

$$\texttt{DIAG}(i,j) \leftarrow a_{i,i+\text{ioff}(j)}.$$

The order in which the diagonals are stored in the columns of `DIAG` is generally unimportant, though if several more operations are performed with the main diagonal, storing it in the first column may be slightly advantageous. Note also that all the diagonals except the main diagonal have fewer than n elements, so there are positions in `DIAG` that will not be used.

Example 3.8 For example, the following matrix which has three diagonals

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

will be represented by the two arrays

$$\texttt{DIAG} = \begin{array}{|ccc|} \hline * & 1. & 2. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & * \\ 11 & 12. & * \\ \hline \end{array} \qquad \texttt{IOFF} = \boxed{\texttt{-1 0 2}}.$$

A more general scheme which is popular on vector machines is the so-called Ellpack-Itpack format. The assumption in this scheme is that there are at most Nd nonzero elements per row, where `Nd` is small. Then two rectangular arrays of dimension `n` $\times$ `Nd` each are required (one real and one integer). The first, `COEF`, is similar to `DIAG` and contains the nonzero elements of A. The nonzero elements of each row of the matrix can be stored in a row of the array `COEF(1:n,1:Nd)`, completing the row by zeros as necessary. Together with `COEF`, an integer array `JCOEF(1:n,1:Nd)` must be stored which contains the column positions of each entry in `COEF`.

Example 3.9 Thus, for the matrix of the previous example, the Ellpack-Itpack storage scheme is

$$\texttt{COEF} = \begin{array}{|ccc|} \hline 1. & 2. & 0. \\ 3. & 4. & 5. \\ 6. & 7. & 8. \\ 9. & 10. & 0. \\ 11 & 12. & 0. \\ \hline \end{array} \qquad \texttt{JCOEF} = \begin{array}{|ccc|} \hline 1 & 3 & 1 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & 4 \\ 4 & 5 & 5 \\ \hline \end{array}.$$

A certain column number must be chosen for each of the zero elements that must be added to pad the shorter rows of A, i.e., rows 1, 4, and 5. In this example, those integers are selected to be equal to the row numbers, as can be seen in the `JCOEF` array. This is somewhat arbitrary, and in fact, any integer between 1 and n would be acceptable. However, there may be good reasons for not inserting the same integers too often, e.g. a constant number, for performance considerations.