Sparse Matrices

A matrix is said to be *sparse* if it contains relatively few nonzero entries. There is no formal definition of exactly what constitutes a sparse matrix, but J. H. Wilkinson, a prominient numerical analysist, suggested the informal definition that a matrix is sparse if it contains "enough zeros that it pays to take advantage of them". From a practical point of view, this makes a lot of sense, because we are typically interested in sparsity because it allows us to reduce storage and/or computation time. For example, a diagonal matrix of order n has at least $n^2 - n$ zeros and at most n nonzero entries (the diagonal). Storing only the diagonal entries of such a matrix instead of the full matrix reduces the storage requirement from n^2 elements to n elements. Moreover, if $A = \text{diag}(a_1, \ldots, a_n)$ is a diagonal matrix with diagonal entries $a_1, \ldots, a_n$ and $x = (x_1, \ldots, x_n)$ is a vector, then the matrix–vector multiplication $y \leftarrow Ax$ requires only n scalar multiplications ($y_i = d_i x_i$ for $i = 1, \ldots, n$). This is significantly fewer than the $2n^2 - n$ arithmetic operations that are required if A were a *dense* matrix of order n.

1 Sparse matrix representations

A sparse matrix can be represented in many ways. Here we consider three of the most common representations and discuss their pros and cons. We will use the following matrix to illustrate the representations:

$$A = \begin{bmatrix} 1.0 & 0 & 0 & 4.0 \\ 0 & 2.0 & 3.0 & 0 \\ 5.0 & 0 & 6.0 & 7.0 \end{bmatrix}. \tag{1}$$

This matrix has 7 nonzero entries out of 12, so A hardly qualifies as a sparse matrix according to Wilkinson's informal definition. However, it is useful as a "toy example" to illustrate different sparse representations.

1.1 Coordinate representation

The *coordinate* representation (often abbreviated "COO") of a sparse matrix A stores the nonzero entries of A as a list T of *triplets* of the form (i, j, A_{ij}) along with a tuple $S = (m, n)$ that represents the *shape* of the matrix. Each triplet consists of a row index, a column index, and a value, and the shape consists of the number of rows and columns. A coordinate representation of the matrix A defined in (1) is given by

$$T = \big((1,1,1.0), (3,1,5.0), (2,2,2.0), (2,3,3.0), (3,3,6.0), (1,4,4.0), (3,4,7.0)\big), \quad S = (3,4).$$

We note that the coordinate representation is also sometimes referred to as a *sparse triplet* representation, and its storage requirement is proportional to the number of triplets $|T|$ (i.e., the

length of the list T) rather than the total number of entries which is mn. Generally speaking, this implies that the coordinate representation is more economical than the full representation when $|T| \ll mn$.

There are many ways to map the coordinate representation to a data structure. For example, the row and column indices can be based on either 0-based or 1-based indices, the list T can be stored as a list of tuples or as a three separate lists (one for row indices, one for column indices, and one for the values). Moreover, the triplets can be in an arbitrary order or ordered by rows or columns. Some implementations allow the triplet list T to contain several triplets with the same indices, and such triples are interpreted in an additive manner. For example, if T contains both $(3,1,1.0)$ and $(3,1,4.0)$, then the representation is equivalent to one where $(3,1,1.0)$ and $(3,1,4.0)$ are replaced by their sum $(3,1,5.0)$. This additive interpretation of triplets that have the same indices means that the sum of two sparse matrices represented using two triplet lists T_1 and T_2 is simply the concatenation of their triplet lists, i.e., $T_1 \frown T_2$ or $T_2 \frown T_1$ (assuming that the two matrices are conformable for addition). Clearly, the result of such a sum operation may not yield a minimal coordinate representation since the number of nonzeros may be smaller than the number of triplets. However, a minimal coordinate representation can be obtained by replacing triplets with the same indices by their sum.

1.2 Compressed sparse column

The *compressed sparse column* (CSC) representation (aka *compressed column storage* or "CCS") of a sparse matrix may be motivated by the observation that if the nonzero entries are ordered by column, then the column index need not be stored along with every entry. Instead, the CSC representation of a sparse matrix A of size $m \times n$ consists of its shape $S = (m,n)$ and an ordered list of lists $C = (C_1, \ldots, C_n)$ where C_j denotes a list of tuples of the form (i, A_{ij}), corresponding to the nonzero entries in the jth column of A. The CSC representation of the matrix A defined in (1) is given by

$$C = \Big(\underbrace{\big((1,1.0),(3,5.0)\big)}_{C_1}, \underbrace{\big((2,2.0)\big)}_{C_2}, \underbrace{\big((2,3.0),(3,6.0)\big)}_{C_3}, \underbrace{\big((1,4.0),(3,7.0)\big)}_{C_4} \Big), \quad S = (3,4).$$

The storage requirement does not only depend on the number of nonzero entries, but also the number of columns since C is a list of length n.

1.3 Compressed sparse row

The *compressed sparse row* (CSR) representation (aka *compressed row sparse* or "CRS") of a sparse matrix is closely related to the CSC representation: instead of ordering the entries by column, the entries are ordered by row such that the row indices become superfluous. The resulting representation consists of the shape $S = (m,n)$ and a list of row lists $R = (R_1, \ldots, R_m)$ where R_i denotes a set of tuples of the form (j, A_{ij}), corresponding to the nonzero entries in the ith row of A. The CSR representation of the matrix A defined in (1) is given by

$$R = \Big(\underbrace{\big((1,1.0)(4,4.0)\big)}_{R_1}, \underbrace{\big((2,2.0),(3,3.0)\big)}_{R_2}, \underbrace{\big((1,5.0),(3,6.0),(4,7.0)\big)}_{R_3} \Big), \quad S = (3,4).$$

The storage requirement depends on the number of nonzeros as well as the number of rows in A since R itself is a list of length m. We note that the CSR representation of a matrix A is

essentially the same as the CSC representation of A^T with the row and column dimensions interchanged.

2 Data structures for sparse matrices in C

In the previous section, we introduced different sparse matrix representations using the mathematical convention that row and column indices start at 1. We will now illustrate how the mathematical characterization can be mapped to data structures in C that use 0-based indexing.

2.1 Coordinate format

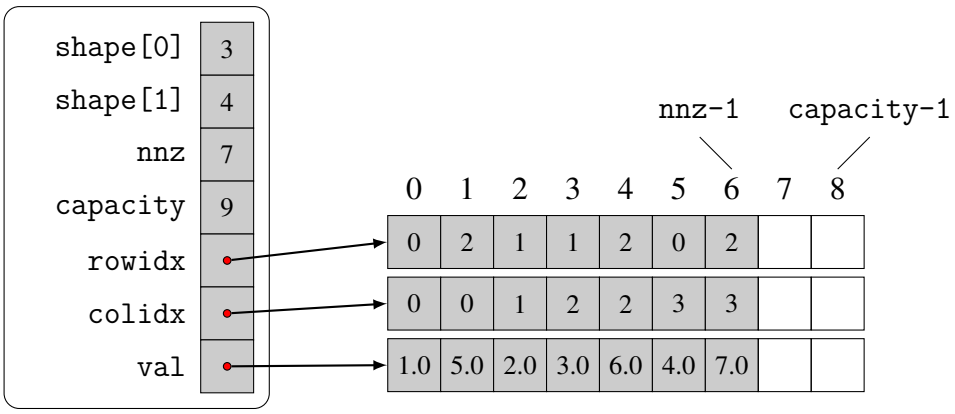
The coordinate representation may be implemented using the following data structure:

```c
typedef struct coo {
    size_t shape[2]; // number of row and columns
    size_t nnz;      // number of nonzeros
    size_t capacity; // array capacity
    size_t *rowidx;  // pointer to array of row indices
    size_t *colidx;  // pointer to array of column indices
    double *val;     // pointer to array of values
} coo_t;
```

To allocate a `coo_t`, we may use the following routine:

```c
coo_t *coo_alloc(const size_t shape[2], const size_t capacity) {
  coo_t *sp = malloc(sizeof(*sp));
  if (sp == NULL) { return NULL; }
  sp->shape[0] = shape[0]; sp->shape[1] = shape[1];
  sp->nnz = 0; sp->capacity = capacity;
  sp->rowidx = malloc(capacity * sizeof(*sp->rowidx));
  sp->colidx = malloc(capacity * sizeof(*sp->colidx));
  sp->val = malloc(capacity * sizeof(*sp->val));
  if (sp->rowidx == NULL || sp->colidx == NULL || sp->val == NULL)
      { coo_dealloc(sp); return NULL; }
  return sp;
}
```

The figure below illustrates how the matrix A defined in (1) may be represented using the `coo_t` data structure.



2.2 Compressed sparse format

The compressed sparse formats may be represented using a single data structure:

```
enum cstype { CSC, CSR };
typedef struct csp {
    size_t shape[2];  // number of row and columns
    enum cstype csx;  // CSC or CSR
    size_t *ptr;      // column and row offsets
    size_t *idx;      // row or column indices
    double *val;      // values
} csp_t;
```

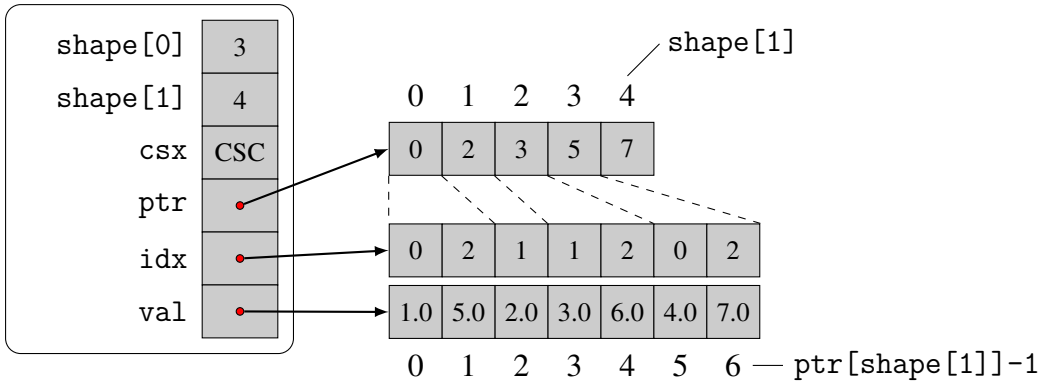
The members `idx` and `val` point to arrays of length equal to the number of nonzeros. The member `csx` determines whether the data structure is a CSC or CSR representation. If the value of `csx` is `CSC`, then `idx` is an array of row indices, and `ptr` points to an array of length `shape[1]+1`. Similarly, if `csx` is `CSR`, then `idx` is an array of column indices, and `ptr` points to an array of length `shape[0]+1`. The `ptr` array has the following meaning: `ptr[k+1]-ptr[k]` is either the number of nonzeros in column (row) `k`, and if `ptr[k+1]-ptr[k]>0`, then `ptr[k]` is the index in `idx` and `val` where the first element of column (row) `k` is stored.

The `csp_t` data structure may be allocated using the following routine:

```
csp_t *csp_alloc(
    const size_t shape[2], const size_t nnz, enum cstype csx) {
  csp_t *sp = malloc(sizeof(*sp));
  if (sp == NULL) { return NULL; }
  size_t N = (csx == CSC) ? shape[1] : shape[0];
  sp->shape[0] = shape[0]; sp->shape[1] = shape[1];
  sp->csx = csx;
  sp->idx = malloc(nnz * sizeof(*(sp->idx)));
  sp->ptr = malloc((N + 1) * sizeof(*(sp->ptr)));
  sp->val = malloc(nnz * sizeof(*(sp->val)));
```

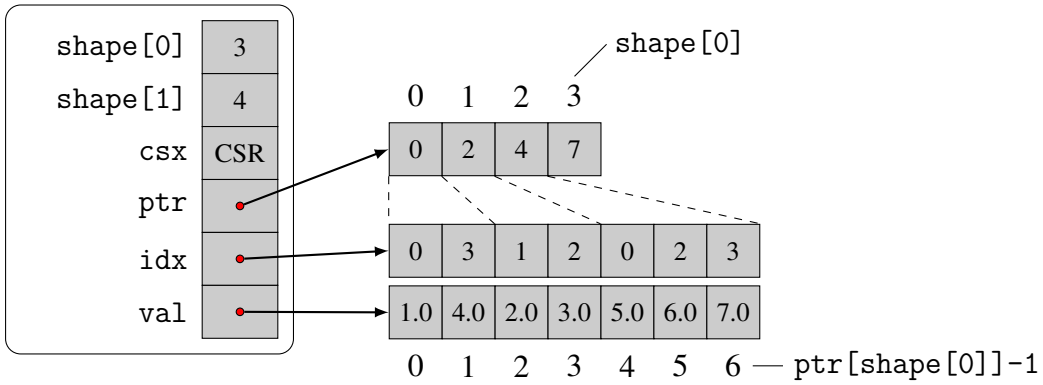
```
  if (sp->idx == NULL || sp->ptr == NULL || sp->val == NULL)
    { csp_dealloc(sp); return NULL; }
  for (size_t k = 0; k < N; k++) sp->ptr[k] = nnz;
  return sp;
}
```

The following figure illustrates the `csp_t` data structure for the matrix A defined in (1) when the compressed sparse column (CSC) storage format is used.



Note that `ptr[k+1]-ptr[k]` is the number of nonzeros in column `k` of the matrix, and `ptr[shape[1]]` is the total number of nonzeros.

The next figure illustrates what the `csp_t` data structure looks like when the compressed sparse row (CSR) storage format is used.



Note that `ptr[k+1]-ptr[k]` is the number of nonzeros in row `k` of the matrix, and `ptr[shape[0]]` is the total number of nonzeros.

3 Reading and writing sparse matrices

We end this note by introducing a file format for sparse matrices known a the Matrix Market (MM) coordinate format. This is one of two Matrix Market Exchange Formats (the other one is for dense matrices), and it is a simple text-based representation in which or matrix A from (1) is stored in following form:

```
%%MatrixMarket matrix coordinate real general
% This is a comment line,
% and so is this one.
3  4  7
1  1  1.0
3  1  5.0
2  2  2.0
2  3  3.0
3  3  6.0
1  4  4.0
3  4  7.0
```

The first line is a so-called "header line" that includes a number of attributes. In the example, the matrix contains a general real-valued matrix in the coordinate format. The header line may be followed by a number of comment lines. These start with character `%` and are optional. The first line that follows the header and subsequent comment lines (if any) contains three numbers: the number of rows, the number of columns, and the number of nonzero entries. This line is then followed by a line for each nonzero in the matrix, and each of these lines contains a triplet (row index, column index, and value). Notice that this format uses 1-based row/column indices.