POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

Sparse Matrices

K. Ming Leung

Abstract: Sparse matrices are introduced. The handling of these matrices in Matlab is discussed.

Directory

Last Revision Date: November 23, 2004

Table of Contents

1. Introduction

Sparse matrices[1, 2] are matrices whose elements are mostly zeros. Matrices that are not sparse are called full matrices. Of course most common matrices tend to be full, however sparse matrices do exist in many areas in science such as graph theory and in numerical treatments of ordinary and partial differential equations. Special treatment of sparse matrices is needed in order to take advantage of the sparsity. It makes sense to store only the non-zero elements to save memory and storage. Of course special indices are needed in order to reference these elements. Operations and manipulations of sparse matrices can often be done much faster compare with their full counterparts. There are slightly different schemes in how sparse matrices are handled. We will discuss below how they are dealt with in Matlab.

2. Sparse Matrices in Matlab

Suppose we enter the following matrix in Matlab:

```
F = [ 0 0 1 0; 2 0 0 3; 0 0 0 0; 0 4 0 5].
```

This creates a full matrix

$$F = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 5 \end{bmatrix}.$$

This matrix has 11 zero elements out of a total of 16 elements. The sparcity of a matrix, defined as the ratio of zero elements to the total number of elements, is 11/16, which is about 70%. The basic idea for handling sparse matrices is to simply store the values of the non-zero elements, together with indices giving the row and column positions of these elements. This can be done in Matlab by entering

```
S = sparse([1 2 2 4 4],[3 1 4 2 4],[1 2 3 4 5]);
```

Matlab displays a sparse matrix by listing their position (sorted by columns) followed by their values. The above sparse matrix , S, will be displayed as

```
S =
   (2,1)         2
   (4,2)         4
```

```
(1,3)        1
(2,4)        3
(4,4)        5
```

Elements are listed columnwise, it i.e. the first nonzero element in column 1 is listed first followed by other non-zero elements in that column, then the first non-zero element in the second column is listed, followed by other non-zero elements in that column, etc. This columnwise ordering of the elements reflects the internal data structure of the way matrices are handled in Matlab.. The first column of the Matlab output shows the row and column indices of these elements. The values of these elements are shown in the second column.

A full matrix can be converted to a sparse matrix by

```
SF = sparse(F)
```

Of course this is not an efficient way to create a sparse matrix. It is much better to create it directly.

On the other hand, a sparse matrix can be converted into a full matrix by

```
FS = full(S)
```

The number of non-zeros in a sparse matrix can be obtained by

```
n = nnz(S)
```

One can check the amount of storage by matrices S and F

```
whos S F
```

The resulting display is

```
  Name        Size                    Bytes  Class

  F           4x4                       128  double array
  S           4x4                        80  double array (sparse)

Grand total is 21 elements using 208 bytes
```

With increasing sparcity, much larger saving in memory will result.

The `sparse` function accepts three extra parameters:

```
S = sparse(Vi,Vj,Vs,m,n)
```

constructs a sparse $m \times n$ matrix. This form is needed when the last row or column of the matrix is zero.

```
S = sparse(Vi,Vj,Vs,m,n,nzmax)
```

allocates spaces to store `nzmax` non-zeros. A sparse matrix for an $m \times n$ matrix of zeroes is given by

```
sparse(m,n)
```

This is the same as

```
sparse([],[],[],m,n,0)
```

The following ways creates an $n \times n$ sparse identity matrix

```
speye(n)
```

or

```
speye(n,n)
```

Given a sparse matrix S,

```
spones(S)
```

creates a sparse matrix having the same sparcity pattern as S and with ones in the non-zero positions.

The following codes dissects and reconstitutes a sparse matrix S:

```
[Vi,Vj,Vs[ = find(S);
[m,n] = size(S);
S = sparse(Vi,Vj,Vs,m,n);
```

The function `spy` gives an overall view of the location of the non-zero elements of a sparse matrix. For example

```
S = gallery('wathen',8,8);
subplot(121), spy(S), subplot(122), spy(chol(S)),
```

Most Matlab matrix operations and functions can be applied to both full and sparse matrices. The dominant factor in determining the execution time and memory requirements for sparse matrix operations is the number of non-zeros, `nnz(S)`, in the various matrices involved.

As an example, let us consider setting up the adjacency matrix of an undirected graph. A graph is a set of points or nodes with specified connections (represented by a line with no arrows, or with two arrows pointing in both directions) between them. For example, let us consider the case where if node 1 is connected to nodes 2 and 4, node 2 is connected to nodes 1 and 3, node 3 is connected to nodes 2 and 4, and node 4 is connected to nodes 3 and 1. The adjacency

matrix is given by

$$F4 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

This matrix is somewhat sparse, as are most adjacency matrices, since not every node is connected to every other nodes. The use of sparse matrices can be very useful. The sparse matrix for this case can be created by

```
S4 = sparse([1 1 2 2 3 3 4 4],[2 4 1 3 2 4 1 3],1,4,4)
```

Of course the last two parameters can be omitted here. We can check to see that this indeed creates the correct matrix

```
F4 = full(S4)
```

MATLAB's `gplot` function creates a graph based on an adjacency matrix and a related array of coordinates. The columns of gplot's coordinate array contain the Cartesian coordinates for the corresponding

node. For the above diamond example, the xy coordinates are given by the array

```
xy = [1 3; 2 1; 3 3; 2 5];
```

This places the first node at location (1,3), the second at location (2,1), the third at location (3,3), and the fourth at location (2,5). To view the resulting graph, enter

```
gplot(A,xy)
```

One can import sparse matrices from computations outside MATLAB. This can be done by using the `spconvert` function in conjunction with the `load` command to import text files containing lists of indices and nonzero elements.

3. The Bucky Ball

One interesting construction for graph analysis is the Bucky ball. This is composed of 60 points distributed on the surface of a sphere in such a way that the distance from any point to its nearest neighbors is the same for all the points. Each point has exactly three neighbors. The Bucky ball models four different physical objects:

1. The geodesic dome popularized by Buckminster Fuller
2. The C60 molecule, a form of pure carbon with 60 atoms in a nearly spherical configuration
3. In geometry, the truncated icosahedron
4. In sports, the seams in a soccer ball

The Bucky ball adjacency matrix is a 60-by-60 symmetric matrix B. B has three nonzero elements in each row and column, for a total of 180 nonzero values. This matrix has important applications related to the physical objects listed earlier. For example, the eigenvalues of B are involved in studying the chemical properties of C_{60}.

To obtain the Bucky ball adjacency matrix, enter

```
B = bucky;
```

At order 60, and with a density of 5%, this matrix does not require sparse techniques, but it does provide an interesting example.

You can also obtain the coordinates of the Bucky ball graph using

```
[B,v] = bucky;
```

This statement generates v, a list of xyz-coordinates of the 60 points in 3-space equidistributed on the unit sphere. The function gplot uses these points to plot the Bucky ball graph.

```
gplot(B,v)
axis equal
```

It is not obvious how to number the nodes in the Bucky ball so that the resulting adjacency matrix reflects the spherical and combinatorial symmetries of the graph. The numbering used by bucky.m is based on the pentagons inherent in the ball's structure.

The vertices of one pentagon are numbered 1 through 5, the vertices of an adjacent pentagon are numbered 6 through 10, and so on. One can generate a picture showing the numbering of half of the nodes (one hemisphere); the numbering of the other hemisphere is obtained by a reflection about the equator. Use gplot to produce a graph showing half the nodes. You can add the node numbers using a for loop.

```
k = 1:30;
gplot(B(k,k),v);
```

```
axis square
for j = 1:30, text(v(j,1),v(j,2), int2str(j)); end
```

To view a template of the nonzero locations in the Bucky ball's adjacency matrix, use the spy function:

```
spy(B)
```

Spy plots of the matrix powers of B illustrate two important concepts related to sparse matrix operations, fill-in and distance. Spy plots help illustrate these concepts.

```
spy(B^2)
spy(B^3)
spy(B^4)
spy(B^8)
```

Fill-in is generated by operations like matrix multiplication. The product of two or more matrices usually has more nonzero entries than the individual terms, and so requires more storage. As p increases, B^p fills in and their spy plot gets more dense.

The distance between two nodes in a graph is the number of steps on the graph necessary to get from one node to the other. The spy

plot of the p-th power of B shows the nodes that are a distance p apart. As p increases, it is possible to get to more and more nodes in p steps. For the Bucky ball, B^8 is almost completely full. Only the antidiagonal is zero, indicating that it is possible to get from any node to any other node, except the one directly opposite it on the sphere, in eight steps.

4. Computational Considerations

The computational complexity of sparse operations is proportional to `nnz`, the number of nonzero elements in the matrix. Computational complexity also depends linearly on the row size m and column size n of the matrix, but is independent of the product m*n, the total number of zero and nonzero elements.

The complexity of fairly complicated operations, such as the solution of sparse linear equations, involves factors like ordering and fill-in, requires special discussions. In general, however, the computer time required for a sparse matrix operation is proportional to the number of arithmetic operations on nonzero quantities.

5. Matlab's Documentation on sparse matrices

The following is the documentation on sparse matrices. You can get such a documentation by typing

`help sparse`

at the Matlab command prompt.

```
SPARSE Create sparse matrix.
   S = SPARSE(X) converts a sparse or full matrix to
   sparse form by squeezing out any zero elements.

   S = SPARSE(i,j,s,m,n,nzmax) uses the rows of [i,j,s]
   to generate an m-by-n sparse matrix with space
   allocated for nzmax nonzeros.  The two integer index
   vectors, i and j, and the real or complex entries
   vector, s, all have the same length, nnz, which is the
   number of nonzeros in the resulting sparse matrix S .
   Any elements of s which have duplicate values of i and
   j are added together.
```

There are several simplifications of this six argument call.

S = SPARSE(i,j,s,m,n) uses nzmax = length(s).

S = SPARSE(i,j,s) uses m = max(i) and n = max(j).

S = SPARSE(m,n) abbreviates SPARSE([],[],[],m,n,0). This generates the ultimate sparse matrix, an m-by-n all zero matrix.

The argument s and one of the arguments i or j may be scalars, in which case they are expanded so that the first three arguments all have the same length.

For example, this dissects and then reassembles a sparse matrix:

```
[i,j,s] = find(S);
```

```
[m,n] = size(S);
S = sparse(i,j,s,m,n);
```

So does this, if the last row and column have nonzero entries:

```
[i,j,s] = find(S);
S = sparse(i,j,s);
```

All of MATLAB's built-in arithmetic, logical and indexing operations can be applied to sparse matrices, or to mixtures of sparse and full matrices. Operations on sparse matrices return sparse matrices and operations on full matrices return full matrices. In most cases, operations on mixtures of sparse and full matrices return full matrices. The exceptions include situations where the result of a mixed operation is structurally sparse, eg. A .* S is at least as sparse as S . Some operations, such as

```
S >= 0, generate "Big Sparse", or "BS", matrices --
matrices with sparse storage organization but few
zero elements.

See also SPALLOC, SPONES, SPEYE, SPCONVERT, FULL,
FIND, SPARFUN.
```

References

[1] John R. Gilbert, Cleve Moler, and Robert Schreiber, *S*parse Matrices in MATLAB: Design and Implementation, SIAM J. Matrix Anal. Appl., Vol. **13**, No. 1, January 1992, pp. 333-356.

[2] A very good tutorial on Matlab's treatment of sparse functions can be found at *McGill University*.