Math 343 Lab 4: Sparse Matrices

Objective

A sparse matrix is one where many or most of the entries in the matrix are zero. In this lab, we will examine Matlab's built-in sparse matrix functions.

The `sparse` and `full` commands

Type the following into Matlab's command window

```
>> A = diag([2 3 4])

>> B = sparse(A)

>> C = full(B)
```

Notice that the matrix A has mostly zeros. It's more efficient to represent the matrix only by the non-zero entries. We use the `sparse` command to carry this out. To convert a sparse matrix from a sparse list of non-zero entries back to full matrix form, we use the `full` command.

We remark that if you want to make a sparse diagonal matrix, the best way to do it isn't to use `diag` followed by `sparse`, it's actually better to use the `spdiags` command:

```
>> spdiags([2;3;4],0,3,3)
```

Banded Matrices

A banded matrix is one whose only non-zero entries are diagonal strips. For example, the matrix

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

is banded because there are three nonzero diagonals. This particular type of banded matrix is called a tri-diagonal matrix.

One can naively create banded matrices in Matlab by adding `diag` commands. For example, the matrix A above can be created by entering

```
>> diag([3,6,9],-1) + diag([1 4 7 10],0) + diag([2 5 8],1)
```

Make sure you understand what each part does. Remember that you can always type `help diag` if you're not sure. A better way to create a tri-diagonal is it use the `spdiags` command:

```
>> spdiags([3 1 0;6 4 2;9 7 5;0 10 8],-1:1,4,4)
```

Make sure you understand how this works. Remember that you can always type `help spdiags` if you're not sure. Also, if you want to visualize the matrix, use the `full` command.

Using sparse matrices

Consider the linear system $Ax = b$, where A is a large tri-diagonal matrix, say $100,000 \times 100,000$. To store a full matrix of that size in your computer, it would normally require 10 billion double-precision floating-point numbers. Since it takes 8 bytes to store a "double", it would take roughly 80 giga-bytes to store the full matrix. For most desktop computers, that fact alone makes the system numerically prohibitive to solve. Recall also that the temporal complexity of a linear system is $O(n^3)$. As a result, even if the computer could store an 80GB matrix in RAM, it would still take several weeks to solve the system. However, since we don't have computers with that much available RAM, most of the matrix would have to be stored on the hard drive, thus bumping our estimates run-time to somewhere between 6 months to a year.

The point is that even the next generation of computers will struggle with solving general linear systems of this form in a reasonable period of time. However, if we take advantage of the sparse structure of the tri-diagonal matrix, we can solve the linear system, even with a modest modern computer. Why? Because all of those zeros don't need to be stored and we don't need to do as many operations to row reduce the tri-diagonal system.

Let's first compute the spatial complexity of the above system when considered as a sparse matrix. There are three diagonals that have roughly $100,000$ non-zero entries. That's $300,000$ double-precision floating point

numbers, which is about 2.4 mega-bytes[1]. As a result, it will easily fit into the computer's RAM. What is the temporal complexity? For a tri-diagonal matrix, it is $O(n)$. Let's see how long it takes to solve the system for random data:

```
>> D = rand(100000,3);

>> b = rand(100000,1);

>> A = spdiags(D,-1:1,100000,100000);

>> tic; A \ b; toc
```

Assignment

Problem 1. *Write a Matlab function that returns a full $n \times n$ tri-diagonal matrix with 2's along the diagonal and -1's along the two sub-diagonals above and below the diagonal. Call this function* `triSassy(n)`*. Hint: Use the* `diag` *command.*

Problem 2. *Repeat the above except have the function return a sparse matrix. You must build this as a sparse matrix from the beginning, not just return* `sparse(triSassy(n))`*. Call this function* `spTriSassy(n)`*. Hint: Use the* `spdiags` *command.*

Problem 3. *Solve the linear system $Ax = b$ where A is the $n \times n$ tri-diagonal matrix from the above two problems and b is randomly generated. How high can you go for each method? Make a table for several different values of n and the time it took to solve for each run. What conclusions can you draw?*

[1] Less storage than your favorite .mp3 file.