MATH 270A – NUMERICAL LINEAR ALGEBRA – PROGRAMMING HOMEWORK 2

Due Friday, November 23th. Please show your teaching assistant. Groups of up to three people.

Exercise 1 (Sparse Linear Algebra and Richardson Iteration)
When we save every entry of an $n \times n$ matrix A explicitly, then we use n^2 entries in memory. If the vast majority of entries in A is zero, then more memory-efficient formats are possible.

The idea is to describe the non-zero structure of a matrix and its entries. Describing the non-zero structure structure explicitly incurs a little overhead, but the savings in terms of memory are dramatic if the number of non-zero entries is much less than n^2. Furthermore, knowing the non-zero structure of a matrix can dramatically speed up an algorithm.

In the following exposition and exercises we use zero-based indexing for arrays, as in C, Java, or Python, but unlike Fortran or Matlab, which use one-based indexing. However, you can translate the idea into any indexing-convention of your choice.

(1) The *coordinate format* saves a matrix in the following form: two arrays of indices `R` and `C` and an array of floating-point numbers `V`, each of length ℓ, where ℓ is the number of non-zero entries of A. For the i-th entry, the matrix has a non-zero value `V`$[\ell]$ in the entry at row `R`$[i]$ and column `C`$[i]$.

For example, the matrix

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

would be described in a C data structure as

```
int R[10] = { 0, 0, 0, 1, 1, 2, 2, 3, 3, 5 };
int C[10] = { 0, 1, 2, 1, 2, 4, 5, 0, 2, 4 };
int V[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Write a method that on input ℓ, `R`, `C`, `V`, and `x` $\in \mathbb{R}^n$ outputs the product Ax. You may use the above matrix as an example.

(2) We generate a class of matrices in coordinate format that appears in practical applications. Let $m \in \mathbb{N}$ and $n = m^2$. We define a matrix A as follows. The diagonal entries of A have the value $4n$. Furthermore, for each row with index $k = i \cdot m + j$ with $0 \leq i, j < m$, we have the off-diagonal entries at column indices

$$\{ \quad (i+1) \cdot m + j, \quad (i-1) \cdot m + j, \quad i \cdot m + j + 1, \quad i \cdot m + j - 1 \quad \} \cap \{0, \dots, n-1\}$$

being set to $-n^2$.

Write an algorithm that produces a matrix of this form in coordinate format.

(3) The *compressed sparse row format* is even more, well, compressed. Letting n denote the number of rows of the matrix $A \in \mathbb{R}^{n \times n}$, we let `R` be an array of integers of length $n+1$, and we let `C` be an array of integers and `V` be an array of floating-point numbers, both of which having length `V`$[n]$. Their significance is as follows.

The last entry of `V` is the number of non-zero entries in the matrix A. Similar as with the coordinate format, the array `C` saves the column index of each entry, and the array `V` saves the numerical value of each entry. The array `R`, however, is read differently from the Coordinate format.

For any $0 \le i < n$, every index $\mathtt{R}[i] \le k < \mathtt{R}[i+1]$ corresponds to a non-zero entry at row i and column $\mathtt{C}[k]$ with value $\mathtt{V}[k]$.

For instance, the matrix A used as an example above would be described in a C data structure as

```
int R[7]  = { 0, 3, 5, 7, 9, 9, 10 };
int C[10] = { 0, 1, 2, 1, 2, 4, 5, 0, 2, 4 };
int V[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Write a method that on input n, `R`, `C`, `V`, and `x` outputs the product Ax. You may use the above matrix as an example.

(4) In practice, it is convenient to assemble a matrix first in coordinate format and then convert it into compressed sparse row format for faster matrix multiplication. Write an algorithm that transforms a matrix in coordinate format into a matrix in compressed sparse row format.

(5) A sparse matrix A arising in numerical partial differential equations may have k non-zero entries per row. Suppose that integers/indices are saved with 4 bytes each and that floating-point values occupy 8 bytes. If matrix has size $N \times N$, how much memory does it occupy for each of the previous three formats? Give an example with $N = 10^3, 10^6, 10^9$ and state with decimal (or binary) prefixes (Kilo,Mega,Tera,...).

Remark 1

The sparse matrix formats require some overhead to describe the non-zero structure of the matrix. This overhead pays off in many applications because the number of non-zero entries is magnitudes smaller than the number of total entries. For example, for $N \times N$ matrices in finite element methods, the number of non-zero entries grows only linearly in N instead of quadratically.

Different sparse matrix formats have different advantages and disadvantages. The coordinate format occupies a bit more memory than the compressed sparse rows format, but it is rather easy to append (or erase) matrix entries. A matrix in compressed sparse rows format is less easy to modify, but the format is rather compact.

A major advantage of the compressed sparse row format is that the matrix-vector product can be parallelized over the rows, which gives a considerable speed up on many-core CPUs.