**Team members**: Nathan Blanken and Nathan Pan

**Project goal and work accomplished so far**

Our goal for the final project is to develop a lossless compression algorithm for a NxM matrix where the resulting compressed matrix uses less storage space than existing formats.

We use the small 500 Human PBMCs, 3' LT v3.1, Chromium X dataset from 10x Genomics for testing. We decided to use the gzipped matrix as it is easier to convert to the full NxM matrix using SciPy and store in a .csv file for reference (while writing this report, we realized that the project says to take in a sparse matrix while we started with the dense matrix - this simply changes how we process the matrix before starting).

To establish a starting point for our work, we decided to store the (row, column, value) 3-tuple in corresponding bit-packed row, column, and value vectors (COO format). This gets us almost as small as the gzipped .mtx file: ~6.1MB vs ~5.7MB. Since we are reading in row-major order, we can make the row vector more efficient by instead storing the number of non-zero values in that row (row[i] returns how many non-zero values row i contains - similar to CSR format). This now reduces our compressed size down to just ~4.3MB, which is smaller than the gzipped matrix but still more than the H5 version provided (~2.5MB). If we then compress our matrix file with gzip we get a smaller file (~1.5MB) but we are not sure if this is allowed. We also tested delta encoding the column vector (with delta encodings resetting to 0 at the start of each row), but this did not make a significant difference on its own.

**Next steps**

Our next steps are to implement a more efficient encoding of the row, column, and value numbers we store. We plan to use Huffman coding to further compress our vectors and will test with and without delta encoding to see where we get the most benefit. We also plan on testing alternative approaches. One idea is to convert the matrix into a 1D array so we only store two bit-packed arrays: one with the index of non-zero values and one with the associated values. These arrays can be delta encoded and Huffman encoded to maximize storage savings. Another idea is to use rank and select enabled bit vectors, where a 1 indicates a non-zero value at that index. We could either use one bit vector for the entire 1D array or one for each row in the matrix (would require a per-row store of values). Run length encoding and/or Huffman encoding of bytes will be necessary to achieve good compression. If time permits, we will update our algorithm to take in a sparse matrix instead of a dense matrix. By the end of the semester we hope to have a method that can compress an NxM matrix smaller than existing methods for all datasets and compression methods we test against.

**Most relevant question**

Where would we find datasets using the other formats (CSR/CSC compressed with GZip and Loom compressed with H5) to compare to?

**Resources:**

1. *Cell ranger*. Feature-Barcode Matrices -Software -Single Cell Gene Expression -Official 10x Genomics Support. (n.d.). https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/output/matrices
2. *Datasets*. 10x Genomics. (n.d.). https://www.10xgenomics.com/resources/datasets?query=&page=1&configure%5BhitsPerPage%5D=50&configure%5BmaxValuesPerFacet%5D=1000&refinementList%5Bproduct.name%5D%5B0%5D=Spatial%2BGene%2BExpression
3. Kourtis, K., Goumas, G., & Koziris, N. (2008). Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression. *2008 37th International Conference on Parallel Processing*, 511–519. https://doi.org/10.1109/icpp.2008.62
4. Lawlor, O. S. (2013). In-memory data compression for sparse matrices. *Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms*, 1–6. https://doi.org/10.1145/2535753.2535758
5. Willcock, J., & Lumsdaine, A. (2006). Accelerating sparse matrix computations via data compression. *Proceedings of the 20th Annual International Conference on Supercomputing*, 307–316. https://doi.org/10.1145/1183401.1183444