AM 129 Homework 6 Report Deliverables

Owen Morehead

Dec 4 2020

Overall I found success implementing this program in C language which works with a sparse matrix data structure in which we calculate the stationary distribution of a particle moving randomly on a lattice. The sparse matrix data structure is very useful because the sparse matrix is a matrix that has so many zero entries that it becomes beneficial only to store the non_zero entries. The matrix vector multiplication that we compute is much more computationally efficient with a sparse matrix. When using 100 grid points with a bias of 1e-3 and a threshold of 1e-6, our program takes 15550 iterations to converge. This is a large amount of iterations but the code compiles very fast and this is because we are using a sparse matrix data structure in our program. I also found that for the same bias and threshold values and a grid size of N = 50, the program converged in 5861 iterations, for N = 200 it converged in 14149 iterations, and for N = 300, it converged in 30216 iterations (still in a timely manner!). When I use N = 500, the program fails to converge as this grid size is too large for an ITER_MAX of 100,000. If I increase the ITER_MAX TO 10,000,000, then using N = 500 our program will converge in 107,863 iterations. The code still compiles pretty fast considering there are over 100,000 iterations done in this calculation!

We haven't said much about the internals of the sparse matrix data structure. But at a high level, what the structure actually does is instead of representing all values (non-zero and zero values) as an array, this sparse matrix structure only stores non-zero elements and can store those values for example with triples in the form (row,column,value). We essentially deallocate everything inside the original matrix structure, and then restructure the matrix by capturing only the non-zero matrix entries and their locations in the matrix.

Performing a matrix-vector product with a dense matrix of size N by N requires on the order of N squared operations. Similarly the storage cost scales like N squared. However, doing a sparse matrix-vector product requires only on the order of N operations and this is because the sparse matrix is stored in this new format where there are no zeros taking any data space. Essentially the sparse matrix data can be visualized as being stored in three columns and the number of rows in each is the number of non-zero elements in the matrix.

The columns are then: the row assignment in the matrix, column assignment in the matrix, and the value in the matrix. If we want to conduct a sparse matrix-vector product, we only need to consider the column of element values from the sparse matrix data. So if there are N number of non-zero elements, the the running time complexity in terms of big-O notation goes as $O(3 * N) = O(N)$. While performing a matrix-vector product with a dense matrix of size N x N requires $O(N^2)$.