Simon Lee
AM 129

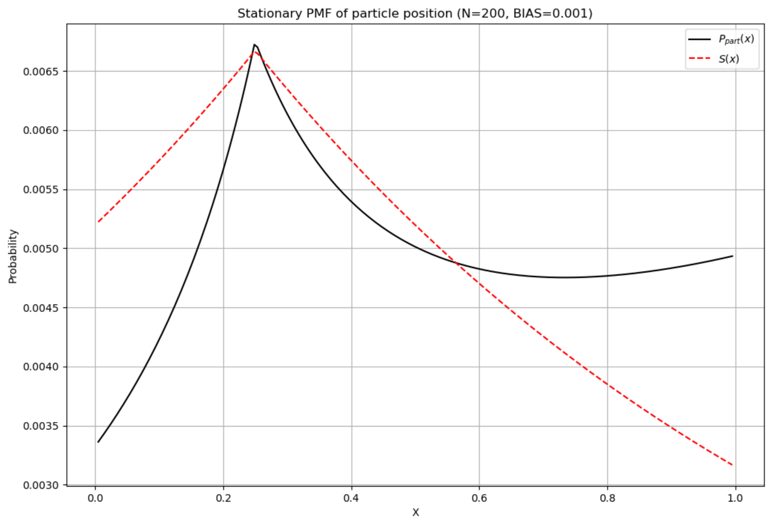
HW 6 Report

Introduction:

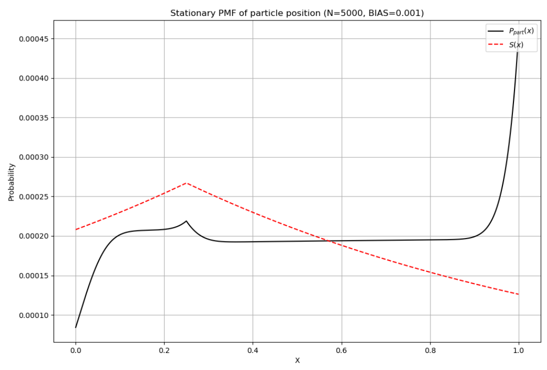
In this assignment we were asked to build some functions for a sparse matrix data structure in C, and visualize the output using Matplotlib in Python. Our objective was to calculate the stationary distribution of a particle moving randomly on a lattice. This concept comes from Markov Chains and the stationary distribution of a markov chain is a probability distribution that remains unaffected in the markov chain as time progresses. In Mathematical terms the stationary distribution of a markov chain with the transition matrix $P$is some vector ψsuch that $\psi P = \psi$. One of its most famous applications are within *Google's* Pagerank algorithm which is a method to rank web pages in their search engine results.

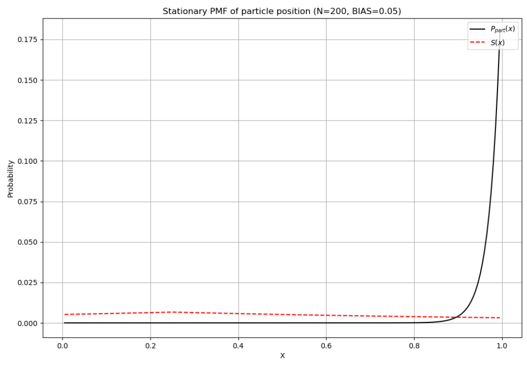
Analysis

Sparse Matrices was an essential portion of this assignment. We were asked to implement parts of our own Sparse Matrix API, and our goal was to have it perform a sparse matrix vector product. Because our matrices have a lot of zero entries, this data structure is essentially supposed to compress and extract the nonzeros (nnz) and perform a matrix vector product of only the nnz values. This is supposed to be more cost effective because this code puts it into compressed-sparse-row format. This helps compress the overall size of the array and extracts only the nnz entries and where they are located. Doing so we were able to obtain this graph that was computed relatively quickly taking a total of 14149 iterations. The diagram below represents 200 grid points and a bias of 0.001.

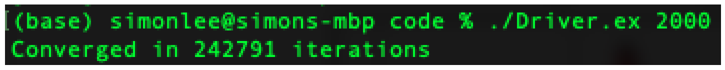


Now to get a better understanding of the graphical representations I was planning to conduct two small experiments by increasing N to a really high number and seeing how a matrix of size N x N performs the larger we go. I was also going to mess with the bias as well which should have some effect over the representation of the graph. The following experiments will be shown:

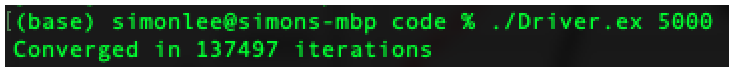




These two plots were generated from my following experiments. On the left we can see what happens given a 5000 x 5000 matrix. And on the right we can see how if we change the BIAS, it has a big effect on the distribution overall. In my first experiment with the 5000 x 5000 Matrix, there was a noticeable difference in time complexity which is expected considering we are working with such a big dimensional space. During my first trial it exceeded the `ITER_MAX` which is placed so if it exceeded this amount of iterations it will automatically shut down the program. Taking a total of 137497 iterations, I was surprised to see that the number overall wasn't too large considering we have significantly increased the `N`. But then I tried running it with `N = 2000` in which I got 242791 iterations, which was much larger than 5000. I also thought this was a relatively interesting observation. Here are a proof of these results:



```
[(base) simonlee@simons-mbp code % ./Driver.ex 2000
Converged in 242791 iterations
```



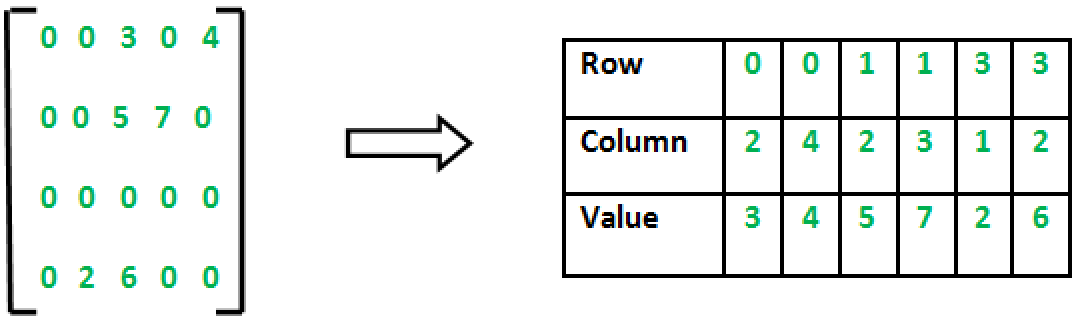
```
[(base) simonlee@simons-mbp code % ./Driver.ex 5000
Converged in 137497 iterations
```

I am guessing there is some maximum value in which the iterations begin to hit a maximum but I am still a little unsure as to why there are more iterations for 2000 than 5000. However this was indeed an interesting behavior that I did discover from this first experiment.

In this second experiment I messed with the `BIAS` and changed it into 0.05. I put in our standard value for N which was 200 and I saw that there was also a significant difference between the two visualizations. Towards the end of the plot it appears that it just begins to blow up exponentially at a certain point. Though it does model a similar behavior to the original graph we can see that there is that sudden spike where it simply begins exponentially increasing. Therefore it is safe to assume that playing around with the BIAS definitely also has to do with how the representation is.

<u>The cost between a matrix vector product and the sparse matrix vector product</u>

It is known that the matrix vector product would not be as efficient at runtime if we were to do this through its standard method. Because of all the zero entries it would simply be a waste to include these arrays of exceeding sizes and perform a matrix vector product of the two. Inside a dense matrix $N\ x\ N$, it requires on the order of N^2operations. However the sparse matrix significantly reduces the cost and takes a $N\ x\ N$ matrix which only requires on the order Noperations. Why is this you may ask? Here is a quick graphical representation as to why this is. In this example we have our sparse matrix which contains a whole bunch of zero entries and some nnz entries. This matrix is converted into condensed row form and we get the following:



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

What is going on in this graphic representation is that only the nnz entries are being extracted during the matrix vector product. So instead of multiplying every entry of the matrix, it only performs it when it needs to. So this sparse matrix is an intelligent optimized form of what we need. It is for this reason that it only requires Noperations vs $N^2$of a matrix vector product.

Conclusion

Overall this assignment showed us how useful the sparse matrix data structure can be. When working with dense matrices that are a lot to handle it can be very useful. In Fact I think I will take what I have learned and apply it to my research project we are currently working on in the Jonsson lab. With having to work with a very large N x N matrix for both gene expression and ATAC-seq data, we worked with binarizing our data which has left a lot of zero entries in our matrix. And to optimize our code a sparse matrix can be used to convert this and extract only the useful information. I am glad I was able to take this assignment and apply to other things and I think it has been cool overall to learn about the backbone of the Pagerank algorithm.