Albert “AJ” Iglesias
8.26.19

Page Rank Using Numpy

- FORMATTED AS JUPYTER NOTEBOOK

Part 1)

To begin my program I created a function to create a random matrix that included only 1s and 0s, but also initiated 0s along the diagonal of the matrix. It was important to establish a while loop based on minval because it was crucial that none of the columns added up to be a total of zero. If this happened the whole process would not work. This was established at the end of the loop by obtained the column sum and then using the numpy function of .amin which obtains the minimum value and if a column sum adds up to zero then it will loop again and should generate a new matrix. If by chance it creates a column with a sum of 0.0 then the code will not run in the next step and the random_matrix should be re-compiled. Once the random matrix is created it is crucial to do a column sum of the matrix by using np.sum and then dividing the Matrix by the obtained column_sum. This should create a modified adjacency matrix that will be used to solve system using Direct and Power methods.

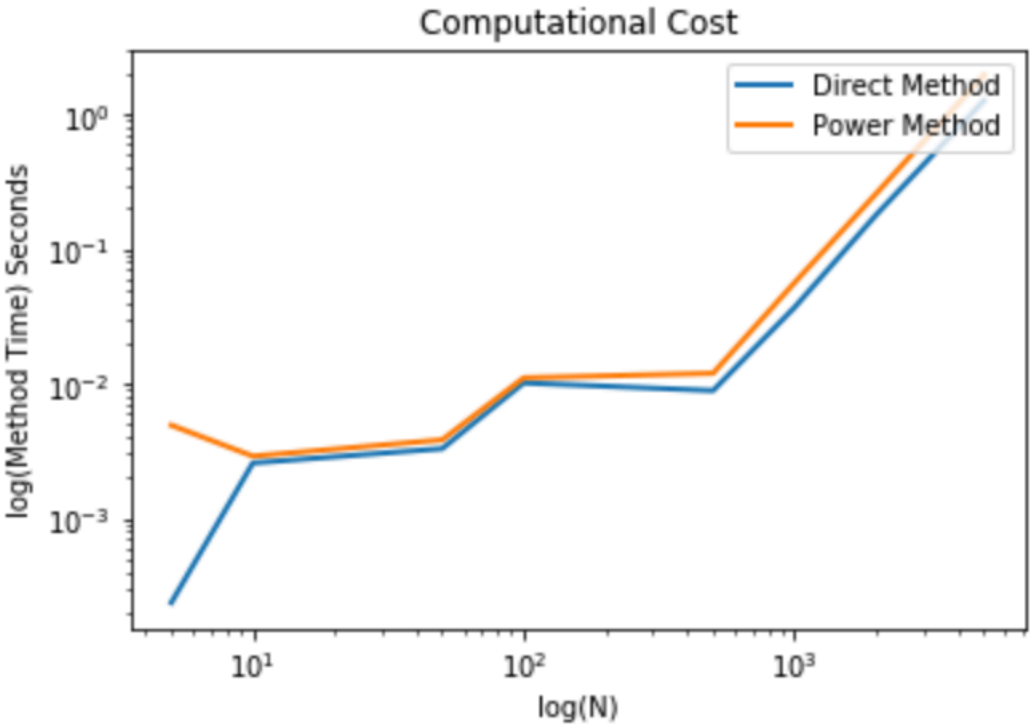
Part 2)

In number two my code was compiled within a function defined as Method1 that made up my Direct Method. This method was less complex than the Power Method. In this function I assigned 4 parameters, d,N,M,coeff respectively. My d was my damping variable which is normally set to .85, N was dependent on the size of the different matrices, M was my modified

adjacency matrix obtained from my random matrix in part 1, and finally my coeff parameter represent the n x 1 array of ones for the linear operation. Here I simply ran it through the formula in the Lab Manuel and then Normalized the R vector I got. This in turn gave me the correct answer for my R vector of pageranks. ***(The Power Method should provide the same answer in a different format)

Part 3)

In three my code was compiled by creating a method within the jupyter notebook called (Method2) that determined my power method. I gave my function 4 parameters, d,N,M,E respectively. Lowercase d never changed throughout both methods and stayed at .85, N was determined by the Lab 2 manual which told us to test 8 N values. My M in my code was whatever my modified adjacency matrix was that I compiled after spawning my random matrix created in Part 1 of this Lab. It was important to change my M matrix for the direct method and power method for each N because if not an error in array size would occur within Jupyter notebook because it would not match my E parameter. With that being said, my E parameters was used here to establish my n x n array of 1s for the Power Method function. Once I initiated some crucial values for the power method, I created a while loop since we have chosen to include a termination criterion. A for loop could be used if using number of interations, I establish a super large error of 1.0 before the loop started to ensure it be less than 1E-6. Using numpy dot product, and numpy's linear algebra normalization techniques I obtained an error of less than 1E-6, which stopped my while loop and correctly gave me an xn array matching the answer from the Direct Method used in part 2.



Part 4)

The graph above details on a loglog scale the computational cost (time it took to run each method) when I compiled my code. The y axis for time is not hardcoded so the graph structure may change each time I run the code however the power method should have higher computational cost on loglog scale as my log(n) gets higher.

Part 5)

I tested my power method code multiple times with n values that were higher than 50 (i.e 75,85,225,600,3000). In this case I played with the termination criterion and set out to look for a way to function my error norm in a way that would still match up pageranks or “our top ten sites.” I determined after experimentation that we can get away with an error norm aggressive function of 1/n for n values greater than 50. In this case our error norm should get lower as our n value increases which in the case of the power method is correct, improving accuracy of the ‘pageranks’.