Homework 7, matrix diagonalization

I have done all parts of the exercise. At least I think part C should count, although I accidentally made my code the optimal way in A and B, so I wrote an unoptimized version just for C. This does not make any difference

PART A

Output: a text file, OutA.txt with the tests I run. the tests are random each time you press make

In part A I implemented both specific functions for jacobi multiplications, and the Jacobi Eigenvalue decompositions. The program in the part A folder runs a number of tests to verify that the decomposition works.

PART B

Output: a text file OutB.txt with lowest energies I found. Also the images wavefunctions.png showing the wavefunctions , and Energies_R.png and Energies_N.png, showing the energies found at different max distance and matrix sizes

In part B I generate the V and K matrices for the hydrogen schrödinger equation, I use this to generate the eigenenergies and eigenfunctions (with my routine from part A).

I investigate the convergence with respect to dr by picking a fixed $R_{max} = 50a$ and running the routine with dozens of different N (where $dr = R_{max}/N$). I investigate the convergence with respect to R_{max} by picking $dr = R_{max}/N = 0.25$ fixed and running with dozens of different $R_{max}.

The convergence is shown simply by plotting the energies as a function of R_{max} or N alongside the bohr energies.

Finally, I run the routine with high resolution $R_{max} = 80$ and $N = 500$, and save the lowest 5 wavefunctions from my routine, against the analytical wavefunction, which I create in another script I made. The agreement is not perfect, but I think it is good enough

NOTE: the sign of the eigenvectors do not matter, my procedure sometimes returns the analytical wavefunction times minus 1, I manually multiply these wavefunctions by -1 whenever that happens, this is NOT an error, if V is an eigenvector, so is -V.

PART C

Output: a png image showing the time cost of the (C.2) optimized and unoptimized (C.1) functions. Fit to n^3

In part C.1, I used the same code as in A, and tried timing the execution time. I use a single-execution on a random symmetric matrix with a size from 25x25 to 400x400, the agreement with O(n^3) is pretty good for larger n (for smaller n the time is dominated by other processes, i.e. generating the matrix).

Unfortunately, I accidentally messed up part C.2 ... by already writing as optimized code as possible. I do not use the suggested timesJ functions, instead I always used the relations given in the notes, and I only ever saved to half the matrix, for this reason, I explicitly had to write a worse version to compare against. The better version is slightly better, but not by much, and for low matrix sizes (100x100 or below) the versions perfor about the same (the time is likely dominated by the time it takes the program to start)