Quantum Information and Computing - Assignment 3

Exercise 1

In the first exercise, it was required to write a python script from which to call the Fortran executable we had built beforehand, changing its parameters with each call.

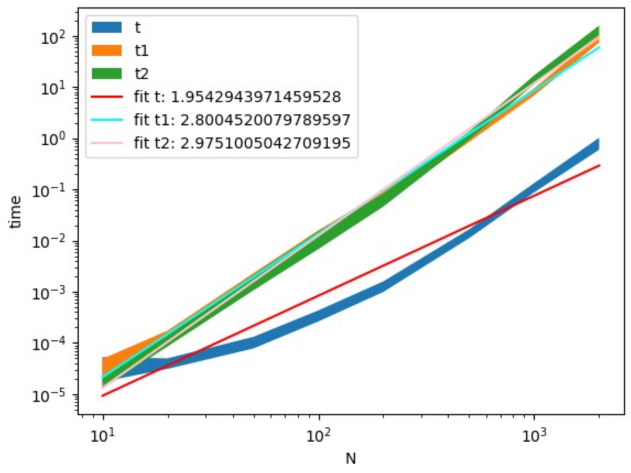
To do so I had to first change the Fortran code to accept the matrix size and number of iterations as command line arguments instead of having the ranges hardcoded, and to print to stdout instead of directly to file. Then, to make the python script run the different cases simultaneously (assuming the resources available are enough) I used a Pool of workers, as created by the multiprocessing library.

The call to Fortran is executed using the subprocess.run function, called by the apply_asinc function of the worker pool with the relative arguments.

The results are then printed to file in a much tidier way than the direct Fortran printing thanks to the easier formatting options, making it possible to print a csv file with the header recording the dimension of the matrix and the elements recording the execution times over different runs.

In this case I only used the executable produced by the standard compilation options, but the same process could easily be applied with a loop over executables generated with all the different options available.

In the graph, the log-log plot of the execution time for different N, along with the fit



Exercises 2 – 3

In the other exercises, it was required to analyze the normalized spacings distribution for a random hermitian matrix, and compare it with the spacings distribution in the case of a real diagonal matrix.

The normalized spacings are found by subtracting each eigenvalue from the next, with the eigenvalues sorted from smallest to biggest, and then dividing by the average spacing.

Given a complex random hermitian matrix, the eigenvalues can be found using scipy.linalg.eig and taking the first element that is returned; after they are ordered using the builtin sorted function, np.diff calculates the differences, and np.mean finds its average value.

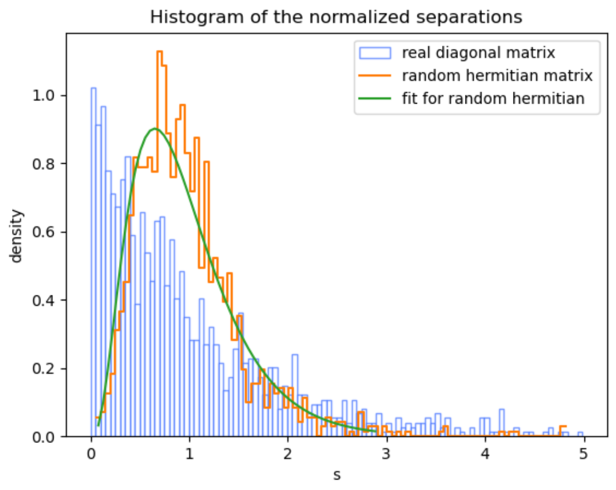
Due to the existence of outliers, the distribution was cut at s=5, which proved to be reliably greater than the values on which the distribution has nonvanishing probability. The histogram values are then copied into other arrays used later for plotting and in others used for fitting.

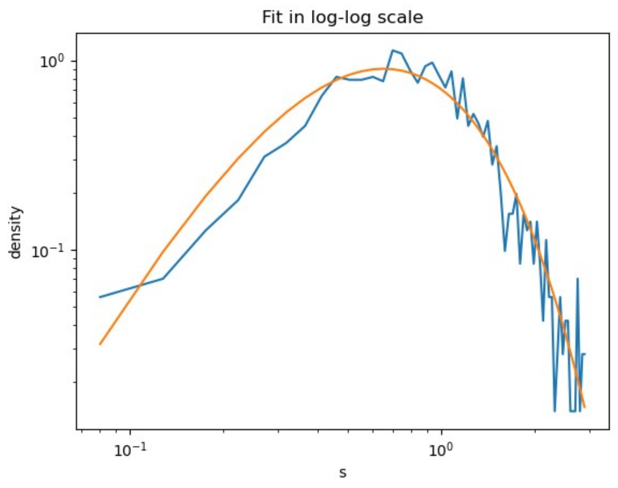
To execute the fitting step, I've used as x the center point of each histogram bin; the distribution is cut again by looking for the last two consecutive bins where there are more than two counts each, to eliminate the noisy tail of the distribution. The actual fitting is done using scipy.optimize.curve_fit over the log of x and y, with the function already defined with this in mind, so that instead of $y = a x^{\alpha} \cdot e^{(b x^{\beta})}$ we have $z = \log(x), w = \log(y) \Rightarrow w = \log(a) + \alpha \cdot z + b \cdot e^{b \cdot z}$.

For the real diagonal case, the eigenvalues are the values of the diagonal itself, so there is no need to use the scipy algorithm; their spacings are exponentially distributed.

In the left graph, the result for a random matrix with N=1500, with each element uniformly distributed between -1 and 1 in both the real and imaginary parts, along with the histogram for the real diagonal matrix with the same size and generating distribution.

In the right graph, an example of the fit as calculated in the log-log scale.


Histogram of the normalized separations
real diagonal matrix
random hermitian matrix
fit for random hermitian
density
s


Fit in log-log scale
density
s
10^0
10^{-1}