QUANTUM INFORMATION REPORT
Exercise 5

Giovanni Piccolo - 1242806

November 10, 2020

Abstract

This report contains the code development for an hermitian matrix diagonalization. Lapack LU reduction algorithm time scaling is computed as function of matrix dimension and the Lapack diagonalization subroutine is used to analyse the statistical properties of the eigenvalues spectrum of random hermitian matrices and random diagonal matrices. In particular the probability distribution of eigenvalue spacings is computed and fitted.

Theory

In mathematics, an *hermitian matrix* A (or *self-adjoint matrix*) is a complex square matrix that is equal to its own conjugate transpose:

$$A \text{ hermitian} \quad \Longleftrightarrow \quad a_{ij} = \bar{a}_{ji} \tag{1}$$

By the spectral theorem the ordered set $\{\lambda_i\}_{i=1}^{N}$ ($|\lambda_N| > |\lambda_{N-1}| > \dots |\lambda_1|$) of eigenvalues of the hermitian matrix A are real and in this report we will study their behaviour. In particular we are interested in studying the normalized spacing of the eigenvalues s_i:

$$s_i = \frac{\lambda_{i+1} - \lambda_i}{\bar{\Delta\lambda}} \tag{2}$$

where $\bar{\Delta\lambda}$ is the average $\Delta\lambda_i$. We can compute this latter average either *globally* - considering all N eigenvalues - or *locally*, by computing it over a different number of levels around λ_i (e.g. $N/100, N/50 \dots$).

Let $P(s)$ be the distribution of the eigenvalue spacings defined in (2) in the limit of matrices of large dimension. For a large class of random matrices, $P(s)$ is given approximately by the "Wigner surmise":

$$P(s) = as^{\alpha}e^{-bx^{\beta}} \tag{3}$$

However, the validity of (3) has only been proved for matrices whose elements are independent identical distributed gaussians[1]. For this reason we will sample the A matrix entries using the *Box-Muller method*: starting from a couple (U, V) of uniform random variables in $[0, 1]$ we have that the two variables z_1 and z_2 are distributed (and independent) as $\mathcal{N}(0, 1)$ if:

$$z_1 = \sqrt{-2\ln U}\cos(2\pi V) \tag{4}$$

$$z_2 = \sqrt{-2\ln U}\sin(2\pi V) \tag{5}$$

In the end, we define as r_i the following average:

$$r_i = \frac{\min(\Delta\lambda_i, \Delta\lambda_{i+1})}{\max(\Delta\lambda_i, \Delta\lambda_{i+1})} \tag{6}$$

Code Development

In this section we will describe and analyze the codes used to solve the exercise highlighting the most significant lines of the files `Ex5_Piccolo_CODE.f90` and `Ex5_Piccolo_CODE.py`.

Ex5_Piccolo_CODE.f90

This code starts by sampling a bidimensional `z` array which contains the variables z_1 and z_2 described by equations (4), (5) using the `gauss_real(u,v)` function to generate normally distributed variables and the `gauss_complex(u,v)` in order to generate complex numbers whose real and imaginary part are normally distributed.
Then, once the $N \times N$ `hem` hermitian matrix is created and whose property (1) is assured by

```
1    (...)
2    hem(jj,ii)=conjg(hem(ii,jj))
3    (...)
```

we were able to call the Lapack subroutine `cheev`, embedded in the `matrix_diag(matr,nn,eigvs,info)` subroutine, in order to diagonalize the matrix:

```
1    cheev('N','U',nn,matr,nn,eigvs,work,lwork,rwork,info)
```

The first input character `'N'` is used to compute the eigenvalues of A only, while `'U'` imposes to Lapack to store the upper triangular part of A. Since our matrix is hermitian we could have chosen as second input character also `'L'`, in

[1] *Statistical Behavior of the Eigenvalues of Random Matrices*, Yi-Kai Liu, Mathematics Junior Seminar, Spring 2001, Princeton University.

order to store the lower triangular part of A, while the eigenvalues are stored inside the vector `eigvs`.

On the other hand the `lwork` integer variable was set equal to -1 in order to calculate the optimal size of the `work` array: in this way the routine returns immediately and provides the recommended workspace in the first element of the corresponding array (`work`). This operation is called a workspace query. Finally, the `info` parameter indicates if the diagonalization worked correctly: if `info` $=$ 0, the execution is successful. If `info=0` then we can compute the eigenvalues and normalize according to a global or local average, as explained in the *theory* section. To ordinate the eigenvalues (and consequently the normalized spacings described in (2)) we used a *merge-sort* algorithm, while the r_i values were computed in the following function:

```
  function nn_minmax_ratio(x, sz)result(r)
  (...)
  do ii=2,sz
    r(ii-1)=min(x(ii), x(ii-1))/max(x(ii),x(ii-1))
  end do
  end function
```

Once the ordered spacings were computed we were able to create an histogram according to a fixed *cutoff*, which coincides with the biggest value of the spacing to be considered in the binning procedure and according to a given number of bins. The local mean (at position i) was computed by the mean from *i-steps* to *i+steps*:

```
  function localmean(x, sz,steps)result(xm)
  (...)
  do ii=1,sz
    ll = max(1,ii-steps)
    rl = min(sz,ii+steps)
    xm(ii)=sum(x(ll:rl))/(rl-ll+1)
  end do
end function
```

The Fortran program must be compiled using the `-o Ex5_Piccolo_CODE.out -lbals -llapack` flags and the correct syntax to launch it is the following:

`$./Ex5_Piccolo_CODE.out [matrix size] [number of samples] [number of histogram bins] [general matrix==0, diagonal matrix==1] [global average==0, local average==1].`

The program then saves the histograms in a file named

`hh_arg[1]_arg[2]_arg[3]_arg[4]_arg[5].dat`,

where the `arg[i]` ($i = 1 \dots 5$) are the command terminal arguments described previously. The ratios are saved in the

`hr_arg[1]_arg[2]_arg[3]_arg[4]_arg[5].dat`

file indeed.

Ex5_Piccolo_CODE.py

This python script is used to read and load the two output files of the Fortran program, to fit the collected data using function (3) and in order to calculate its fitting parameters and print them on screen with along the mean $\langle r \rangle$. The command line syntax of this program is the same with respect to the previous one: `$ python3 Ex5_Piccolo_CODE.py arg[1] ...arg [5]`.

Results

The simulation was run using a 1500×1500 matrix with 100 sampled matrices and 200 bins for the histogram, computing both using the global and local mean and for diagonal and general hermitian matrices. In Table 1 we have reported the results obtained from fit (3) and of mean calulations.

Table 1: Parameters of the fit shown in Figure 1 and mean $\langle r_i \rangle$. The values presented with (*) are zero in order to be congruent with the significant precision of the data; the error on the mean was computed using the standard deviation and not the mean error.

	Global	**Local**	**Global diagonal**	**Local diagonal**
$\boldsymbol{a}$	13.121	3.729	1.218	1.120
$\boldsymbol{\alpha}$	2.532	2.019	0.000*	0.000*
$\boldsymbol{b}$	2.532	1.435	1.117	1.109
$\boldsymbol{\beta}$	2.775	1.882	1.174	1.032
$\langle \boldsymbol{r} \rangle$	0.600 ± 0.008	0.600 ± 0.008	0.388 ± 0.009	0.388 ± 0.009

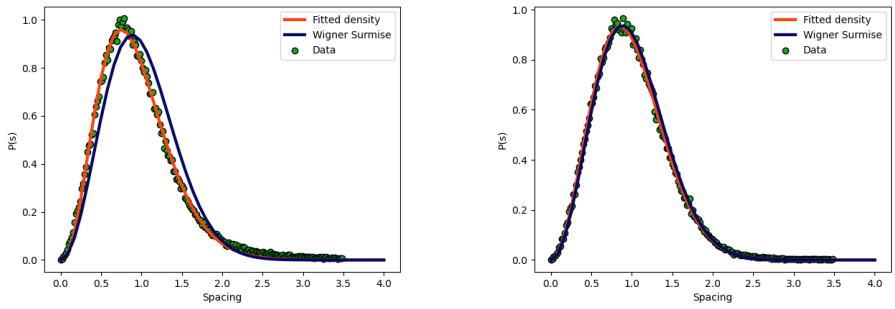
In Figure 1 we show the result we gained plotting the simulated data and its fits. The first row of Figure 1 is referred to the general hermitian matrices case, while the second one represents the diagonal one.

The expected theoretical Wigner sumrise function plotted together with the fitted curve in the general hermitian case is defined as:

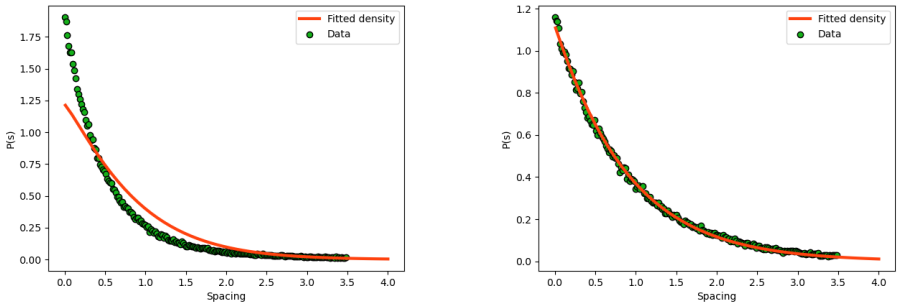
$$P(s) = 2 \left(\frac{4s}{\pi} \right)^2 e^{-\frac{4s^2}{\pi}} \tag{7}$$

As we can clearly see from Figure 1, the use of a local mean describes better the simulated data with respect to a global mean.

Global Local



G-H



D-H

Figure 1: Spacing s as a function of the $P(s)$ curve described in (3). The first two images of this set represent the general hermitian case (G-H) while the other two the diagonal ones (D-H). 'Global' and 'Local' on the top of the figure are referred to the way which the mean was computed.

Self Evalutation

Before doing anything, I tried to make the Lapack routines work in simple trial programs. This has proven extremely useful later on, so I'll use this approach any time I'll need it. I had bad time trying to make Gnuplot doing histograms (and also had problem with a reported bug) and for this reason I decided to use Python instead. Furthermore the histogram fits should have been done using the errors from Poisson distribution and the statistical part should have been implemented in a more thoroughly way.