Quantum Information and Computing

prof: *Simone Montangero*

Report of exercise 5
Random-matrix theory

Alberto Bassi

November 9, 2020

Abstract

The purpose of this week is to study random-matrix theory. In particular, we will compute the eigenvalues of hermitian and real diagonal random matrices. Then, we will study the distributions of the normalized difference between neighbour (ordered) eigenvalues.

Theoretical introduction

A hermitian matrix $H \in M_{n,n}(\mathbb{C})$ is such that $H^\dagger = \left(H^T\right)^*$, where $*$ indicates the complex conjugate. It is easy to show that its eigenvalues are all real. Matrices with entries random distributed according to some probability distribution are known as random matrices. In particular, for any random square complex matrix $C \in M_{n,n}(\mathbb{C})$, whose entries are i.i.d. random variables with mean zero and unitary variance, the circular law holds: as $n \to \infty$ the eigenvalues of $C/\sqrt{n}$ are almost surely uniformly distributed in the unitary circle in the complex plane. If, however, there are dependences among entries, the law becomes semicircular. That is the case of symmetric matrices, for which the real eigenvalues are distributed following a probability density function whose graph is a semicircle.
In this report, we are going to analyse the properties of the normalized spacings between eigenvalues, for both random hermitian and real diagonal random matrices, with entries taken from a uniform distribution in $[-1, 1]$. Given a vector of real eigenvalues $(\lambda_1, \lambda_2, \dots, \lambda_n)$, in increasing order, we define:

$$\Delta\lambda_i = \lambda_{i+1} - \lambda_i \ , \qquad i = 1, 2, \dots, n-1 \ . \tag{1}$$

We want to study the distribution of the random variable

$$s \equiv \Delta\lambda_i / \bar{\Delta\lambda_i} \ , \tag{2}$$

where $\bar{\Delta\lambda_i}$ is the mean among the differences. In particular, if we take the average along all the interval, we get $\bar{\Delta\lambda_i} = (\lambda_n - \lambda_1)/(n-1), \forall i$. We will also take into account the case in which the average is computed locally, e.g. $\bar{\Delta\lambda_i} = (\lambda_{i+k} - \lambda_{i-k})/(k+1)$, $2 \leq k \leq n$, with the opportune adjustments when the limits are exceeded.
We shall study also the random variable

$$r \equiv \frac{min(\Delta\lambda_{i+1}, \Delta\lambda_i)}{max(\Delta\lambda_{i+1}, \Delta\lambda_i)} \in [0, 1] \ , \qquad i = 1, 2, \dots, n-2 \ . \tag{3}$$

We suppose that $s \sim P_s(s)$ and $r \sim P_r(r)$. It is difficult to compute these distribution analytically (it involves the FT of a Wigner's semicircular distribution), so we shall compute them numerically (i.e. we will compute the normalized histograms). Then, we will fit $P_s(s)$ with the function

$$f(s) = as^\alpha \exp(-bs^\beta) \ . \tag{4}$$

In the following H and D represents respectively the hermitian and the diagonal matrix.

Code Development

We have rearranged the code of previous week in more ".f" files. The program has become complex, so it is impossible to report it entirely here due to the limited space. However, we will try to give a general idea.
All the files are contained in the folder "fortran", which also includes a folder "temp" wherein .txt temporary files are stored (useful to dialogue between programs while running). We have written a new module, called "random_matrix", which contains subroutines to study random matrices and we transferred it in the file "modules.f" along with the module "debugging.f". The program debug_program in "debugging.f" initialize the debug variable. We have separated it from the rest, because we will call the main programs several times later, so we do not want our debugging initialization to be called every time. The program eigenvalues in "norm_spacings.f" computes the eigenvalues of a hermitian random matrix thorough Lapack's *cheev* and of a real diagonal one through Lapack's *ssyev*. Then, the program calculates the normalized spacings s and the variable r, saving them in .txt files. The data are saved in the folder "data". The program prob_distribution in "prob _distribution.f" computes the probability distributions of the random variables s and r for both the hermitian and the diagonal matrix, saving them in .txt files. The program is installed by means of a python script "install.py". Every time we compile a program, we have to compile also the modules. A typical call is the following:

```
subprocess.call("gfortran -O3 fortran/modules.f fortran/norm_spacings.f -o norm_spacings.out -
    llapack",shell=True)
```

The execution of the various programs is controlled by another python script, "exe.py", here not reported due to the limited space. This way of working with python, which is pre-compiled, is easier, because otherwise we should have to compile fortran programs for several time. Instead, we can easily compile with the optimization flag -O3 in a brief time. Every step of the program, computation of normalized spacings, computation of the probability distribution and plots are asked to the user. All of them are independent and the user can run one of these task each time. The computation of normalized spacings is done four matrices at once (with the library multiprocessing), with dimension required to the user. In this version, the division k is n, i.e. $\Delta\lambda_i = (\lambda_n - \lambda 1)/(n-1), \forall i$. In fact the program eigenvalues in "norm_spacings.f" easily allows us to modify the division k. Every time new values are appended. Then, the user is asked to computed the probability density functions (normalized histograms) $P_s(s)$ and $P_r(r)$. In the end, the program plots $P_r(r)$ and plots and fits $P_s(s)$ by calling the gnuplot's script "plot.plt". The output is redirected in the folder "tex", in which the report has been written.
The most important subroutines we have written so far for this exercise are *norm_ spacings* and *get_ prob*. With regards to the first we have

```
subroutine norm_spacings(n_spac,eig_val,dmn,div)
      implicit none
      integer :: dmn !dimension of the matrix
      integer :: div !how large is the interval around lambda_i
      integer :: ii    !iterators integers
      real :: delta             !average spacing local or not
      integer :: ii_min, ii_max
      real,dimension(dmn) :: eig_val !eigenvalues vector
      real, dimension(dmn-1) :: n_spac !output of normalized spacings
      if(div>dmn) div=dmn
      if(div<2) div=2
      do ii=1,dmn-1
         if(mod(div,2)==1) then
            ii_max=ii+(div-1)/2
            ii_min=ii-(div-1)/2
         else
            ii_max=ii+div/2-1
            ii_min=ii-div/2
         endif
         if(ii_max>dmn) then
            ii_max=dmn
            ii_min=dmn-div+1
         else if(ii_min<1) then
            ii_min=1
            ii_max=div

         endif
         delta=(eig_val(ii_max)-eig_val(ii_min))/(div-1)
         n_spac(ii)=(eig_val(ii+1)-eig_val(ii))/delta
      enddo
      end
```

This subroutine computes the normalized spacings *n_ spac* given a vector of eigenvalues *eig_ val*. So, it is usable

by either the hermitian or the real diagonal matrix. Here division k, i.e. the interval to calculate the average of normalized spacings, is denoted by "div". We notice that in this way every time we normalize over an interval of size k.
Then, the subroutine to get probability density functions has two versions, one for the distribution of$s$ and one for the distribution of$r$. That is because we did not succeed in opening a file whose name is given as input, so we managed to pass the file's name thorough an integer "inter". To clarify it, if we want to get the probability density function of s for the hermitian matrix, then we set $inter = 0$, so the subroutine opens the file "data/H_spac.txt" in which the normalized spacings are saved and accumulated. Instead, if inter=1, then it opens "data/D_spac.txt". We point out that that has been a "desperate" manoeuvre to get the program compiled. It is always easier to pass directly the filename, but that case the compiler gave a message of error, because it expected a scalar (file=filename), even tough for the call file='file.txt" worked without problems. Moreover, before computing the probability density function, we had to insert a stopping value in the file to prevent other errors (here -1, because the normalized spacing of a ordered vector are always of the same sign, here positive).
So, the subroutine *get_prob* computes the normalized histograms by the values read from a file, accepting as inputs interval and the number of bins (we have always chosen nbins=100, xmin=0, xmax=5 for the distribution of s, while xmax=1 for r) and counting the number of points in each bin. We divide the number of each bin for the total number of points and and the binwidth, so that it is properly normalized. The output is a vector *prob* of dimension nbins.

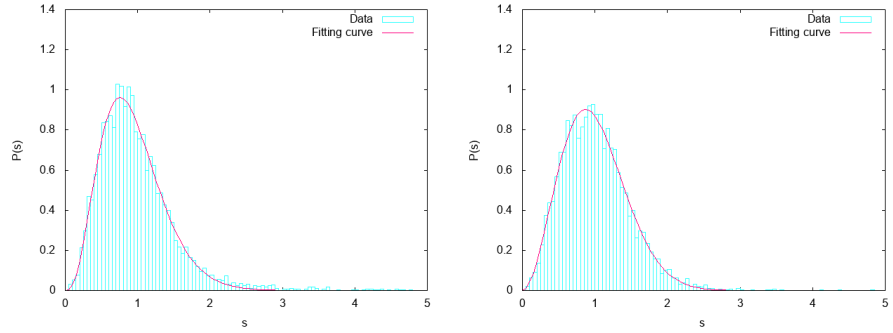
```
subroutine get_prob(inter,prob,nbins,xmin,xmax) !compute the probability densiy function
     implicit none
     integer :: nbins !number of bins
     real, dimension(nbins) :: prob !Prob distribution discretized
     real :: xmin,xmax, bwidth !interval and width of bins
     real :: val !value to be read from file
     integer :: ii,jj             !iterator
     integer :: count          !frequency counting
     integer,intent(in) :: inter
     bwidth=(xmax-xmin)/nbins*1.0!Print and end flag in every file.Here -1.
     if(inter==0) then
        open(10,file="data/H_spac.txt",status='old',access='append')
        write(10,*) -1.0
     elseif(inter==1) then
        open(10,file="data/D_spac.txt",status='old',access='append')
        write(10,*) -1.0
     else
        stop
     endif
     close(10)!Compute the probability density function
     do jj=1,nbins
        if(inter==0) then
           open(10,file="data/H_spac.txt",status='old')
        elseif(inter==1) then
           open(10,file="data/D_spac.txt",status='old')
        else
           stop
        endif
        count=0
        ii=1
        do
           read(10,*) val
        if(xmin+bwidth*(jj-1)<=val.and.val<xmin+bwidth*jj) then
           count=count+1
        endif
        ii=ii+1
        if(val<0) exit
        enddo
        close(10)
        prob(jj)=1.0*count/(bwidth*ii)
     enddo
     end
```

Results

At first, we computed the probability density function[1] for a $n = 5000$ matrix with different values of division k and fitted with Equation 4. So, every time the probability is computed with a single matrix dataset. In Table 1

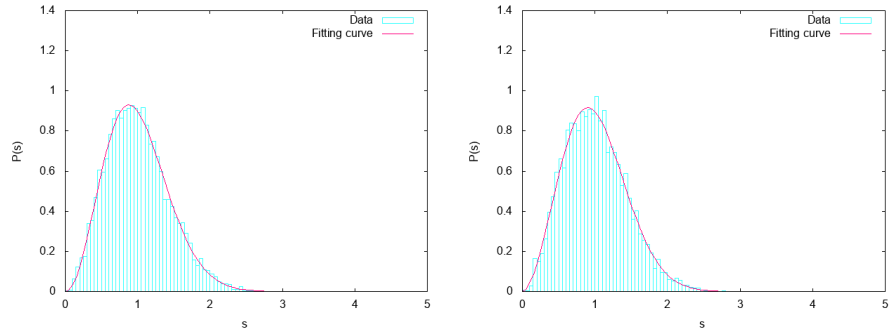
[1] All simulations performed by a MSI Leopard Pro(2017).

the fitting parameters are reported, while the graphs are reported in Figure 1 for hermitian matrices. We do not report the graphs for diagonal matrices due to the limited space.



(a) n/k=5000/5000=1

(b) n/k=5000/500=10



(c) n/k=5000/20=250

(d) n/k=5000/5=1000

Figure 1: The probability distribution of s for different divisions and $n = 5000$ for hermitian matrices.

We can notice that the smaller n/k is, the more stable the distribution becomes. Moreover, the tails tends to disappear, so the variance becomes smaller. The average and the shape of distribution seems to be almost equal.
In the end, we ran a simulation accumulating values from 16 matrices of size $n = 5000$, four processes each time, with division $k = 5000$. In the following we report the terminal's output to show the features of the whole program. Also the mean values of r are reported on terminal. For computing r, we did not take the maximum eigenvalue.

k	a_H	b_H	α_H	β_H	a_D	b_D	α_D	β_D
5000	15.08	2.90	2.66	1.31	1.05	1.07	0.001	0.94
500	2.94	1.22	1.84	1.97	1.18	1.15	0.07	0.93
20	3.46	1.34	2.03	1.93	0.79	0.75	-0.06	1.13
5	2.86	1.15	1.92	2.06	0.56	0.25	-0.07	2.09

Table 1: The fitting parameters for different divisions.

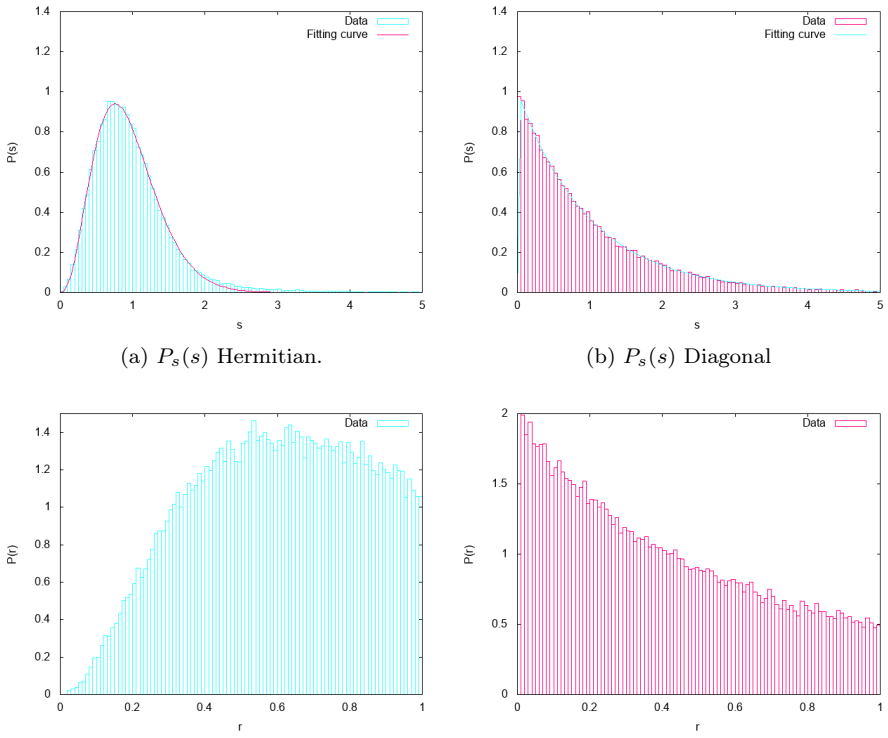
```
ab77@alberto:~/Documents/Quantum_Information/Report_week5/ex5$ python3 exe.py
This program computes the normalized spacings for random matrices
Do you want to make a simulation?[y]/[n]
y
Insert the dimension
5000
 Do you want to start the debugging mode?[y]/[n]
y
 Debugging mode: ON
 Do you want  to print checkpoints?[y]/[n]
n
 Error encountered:            0 #message repeated 16 times
Do you want to compute the probability distributions?[y]/[n]y
Insert number of bins
100
Insert xmin
0.
Insert xmax
```

```
5.
 Average of r(H):  0.597880721
 Average of r(D):  0.384872407
Total computation time is: 762.7492728233337
Do you want to plot and fit the results?[y]/[n]y
```

The fitting parameters are reported on Table 2, while the graphs are reported in Figure 2.

a_H	b_H	α_H	β_H	a_D	b_D	α_D	β_D
10.87	2.59	2.43	1.37	1.01	1.01	0.001	0.98

Table 2: The fitting parameters for the simulation.



(a) $P_s(s)$ Hermitian.

(b) $P_s(s)$ Diagonal

(c) $P_r(r)$ Hermitian

(d) $P_r(r)$ Diagonal

Figure 2: The probability distribution of s and r for 16 matrices of $n = 5000$.

We can notice that now the distributions are more stable and the fits better. Finally, we get an average of 0.59 and 0.38 for the variable r for the hermitian matrix and the diagonal one respectively.

Self Evaluation

In this week we have learned how to use Lapack for computing the eigenvalues of random matrices. Moreover we have learned how to deal with random matrices, which are of the main importance in many branches of Physics. The results of the simulations are good overall, we have obtained what expected in many cases. With regards to the options of lapack's *cheev* and *ssyev*, we can guess that "work" and "rwork" are vectors involved in the efficiency of the algorithm used to compute the eigenvalues.
From the computational point of view we have learned how to divide tasks among different programs and how to reorganize different files in the main folder. Clearly, we have been keeping using tools from last week's task too, for instance parallelizing method. There are things to be improved. First, we have not found a way to automatize the intervals of graphs in gnuplot. That is something very useful to speed up the simulations in future. The fact of dividing the tasks of the program among many different executable has been successful. In this way, it is easier to write complex programs that "speak" to each other. However, we may find a better way to deal with the debugging program, in order not to print too messages on the screen while running them in parallel. Moreover, we shall find a better way to pass the inputs to fortran's executable (we remind that here we have done it by using .txt temporary file in the folder temp).