UNIVERSITY OF PADUA
PHYSICS AND ASTRONOMY DEPARTMENT

# Homework #5

Student name: *Beatrice Segalini* – 1234430

Course: *Quantum Information and Computing 2020* – Professor: *S. Montangero*
Due date: *November 10th, 2020*

## Abstract

In this report, a possible solution of the eigenproblem for hermitian random matrices is displayed. Functions and subroutines of `Fortran` library `LAPACK` are applied to diagonalise the matrices, compute their eigenvalues, and to obtain the normalised spacings between eigenvalues. The normalisation factor is computed as both the local and global average on the differences between eigenvalues. Furthermore, a study of $P(s)$, the distribution of the previously defined spacings, is included, for random matrices of size at least $N = 1000$. The matrices considered are both hermitian with complex entries and diagonal with real ones.

## Theory

A diagonal matrix $D$ is a matrix whose elements are all equal to 0 but the ones of the diagonal. Its eigenvalues are, of course, its non-zero elements.

A matrix $A$ is Hermitian if $A = A^\dagger$, and if its size is $N$ it has exactly $N$ real eigenvalues.

The normalised spacings between eigenvalues $s_i$ is defined as:

$$s_i = \frac{\Delta\lambda_i}{\Delta\bar{\lambda}} \tag{1}$$

with $\Delta\lambda_i = \lambda_{i+1} - \lambda_i$ and $\Delta\bar{\lambda}$ is the average $\Delta\lambda_i$ computed either globally on all the spacings or locally, in an arbitrary neighbourhood of each $\Delta\lambda_i$.

The so-defined spacings are distributed according to a probability distribution $P(s)$ such that:

$$P(s) = as^\alpha \exp\left(-bs^\beta\right) \tag{2}$$

## Code development

All the following analysis is performed on matrices of size $N = 1500$, with histograms of 100 bins, in the $s-$range $s \in [0, 5]$. Each matrix entry different from 0 is drawn uniformly in $[-1, 1]$ with the intrinsic `Fortran` function `RAND`.

**Computation of eigenvalues.** The first required step is the computation of eigenvalues. This is trivial for a diagonal matrix, since its diagonal elements are also its eigenvalues. For a Hermitian matrix with complex entries, on the other hand, the computation is more complicated.

To derive the eigenvalues, one needs to diagonalise the matrix and then to solve a $N$ dimensional system of equations.

The `LAPACK` library provided by `Fortran` allows to automatically do so, thanks to the subroutine `ZHEEV`. The subroutine diagonalise the matrix, compute its eigenvalues and stores them in ascending order. The code implementation is shown in Listing 1.

Listing 1: Eigenvalues derivation with LAPACK-provided subroutine ZHEEV

```
!   compute optimal size of workspace
LWORK = -1
call ZHEEV('N', 'U', N, A, N, evl, WORK, LWORK, RWORK, INFO)
LWORK = min(LWMAX, int(WORK(1)))

!   compute eigenvalues
call ZHEEV('N', 'U', N, A, N, evl, WORK, LWORK, RWORK, INFO)
```

In lines $1-4$, the LAPACK subroutine computes the optimal workspace size, given the dimension $N$ of the problem to be solved. After that, the array `evl` is updated with the eigenvalues of $A$ in ascending order, if the computation converged (i.e., if the parameter `INFO` is equal to 0).

**Computation of spacings.** After properly obtaining a sorted array of eigenvalues, the normalised spacings can be computed. To do so, a function `spacings` is defined and displayed in Listing 2.

Listing 2: Code for calculating the normalised eigenvalue spacings.

```
function spacings(evl, step)result(ss)

implicit none
double precision, dimension(:), intent(in)  :: evl
integer, intent(in)                         :: step
integer                                     :: N, left, right, INFO
double precision                            :: avg
double precision, dimension(:), allocatable :: ss, devl

N = size(evl)

allocate(ss(N-1))
allocate(devl(N-1))

devl = evl(2:N) - evl(1:(N-1))

do aa = 1, (N-1)
        left = max(1, aa - step)
        right = min(N-1, aa + step)
        avg = sum(devl(left:right))/(right-left+1)
        ss(aa)= devl(aa)/avg
end do

call dlasrt('I', N-1, ss, INFO)

deallocate(devl)

end function
```

The function takes as input a vector of eigenvalues `evl` and an integer `step`. This last variable defines the range around which to compute the average $\Delta\bar{\lambda}$, which in this way can also be computed locally around each $\Delta\lambda_i$. The `devl` support variable computes the various $\Delta\lambda_i$, which are then normalised with the proper average value. Finally, the spacings are sorted in increasing order with the `LAPACK` subroutine `dlasrt`.

**Making histograms.** The normalised spacings need now to be represented graphically through normalised histograms. The number of bins is arbitrarily set to 100 and the function in Listing 3 is

defined to achieve this goal.

Listing 3: Code for binning the histograms.

```
function make_hist(values, nbins, bottom, top)result(counts)

implicit none
real*8, dimension(:), intent(IN)    :: values
integer, intent(IN)                 :: nbins
real*4, intent(IN)                  :: bottom, top
integer                             :: N, jj, ii

real*8                              :: temp, limit, dx
integer, dimension(:), allocatable  :: counts
logical                             :: flag

flag = .True.
N = size(values)

allocate(counts(nbins))

dx = (top - bottom)/nbins

counts = 0
limit = bottom + dx
jj = 1

do ii = 1, nbins
        flag = .True.
        do while((jj <= N) .AND. flag)
                temp = values(jj)
                if (temp > limit) then
                        limit = limit + dx
                        flag=.FALSE.
                else
                        counts(ii) = counts(ii) + 1
                        jj = jj + 1
                end if
        end do
end do

end function
```

Thanks to this function, the data are binned and counted properly and just need to be normalised. The normalisation factor can be obtained by summing the total number of counts and multiply it by the bin width `dx`.

After performing 20 iterations of the previously described procedures (matrix generation, diagonalization, computation of eigenvalues and normalised spacings, bin division), the obtained `total_hist` array is normalised and the histograms in Figure 1, 2 are obtained.

The two graphs seem to have very different shapes: the first one is clearly peaked around the value $s \approx 1$, while the second one is similar to an exponential decay.

Moreover, one can notice that changing the normalisation factor causes a shift in the peak of the distribution for the hermitian matrix one, while for the diagonal this effect not relevant.

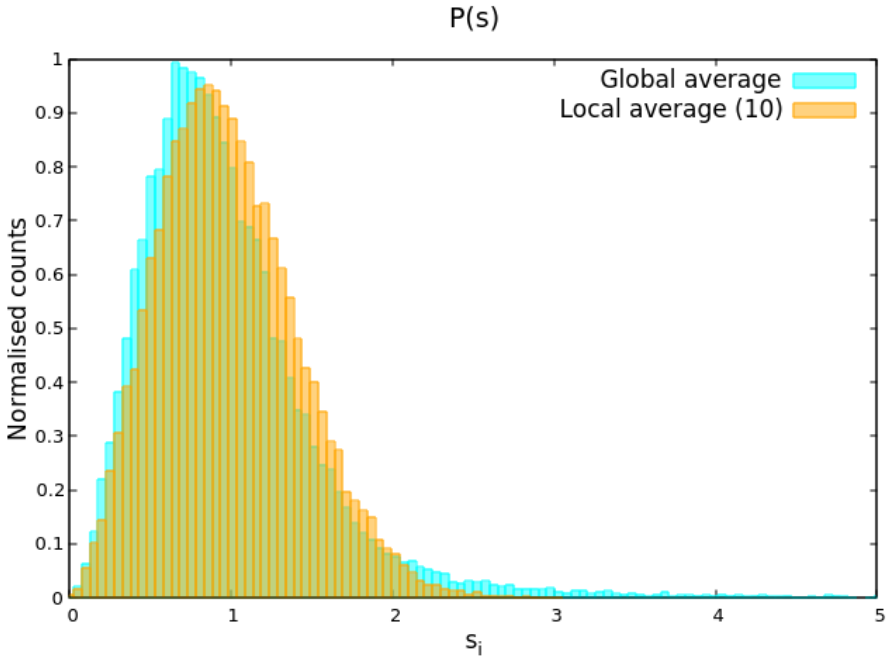


Figure 1: Distribution of normalised eigenvalue spacings for Hermitian matrices with random complex entries. The orange set is obtained normalising with a local average in a neighbourhood of each spacing of 20 elements, for the cyan dataset a global average is considered.

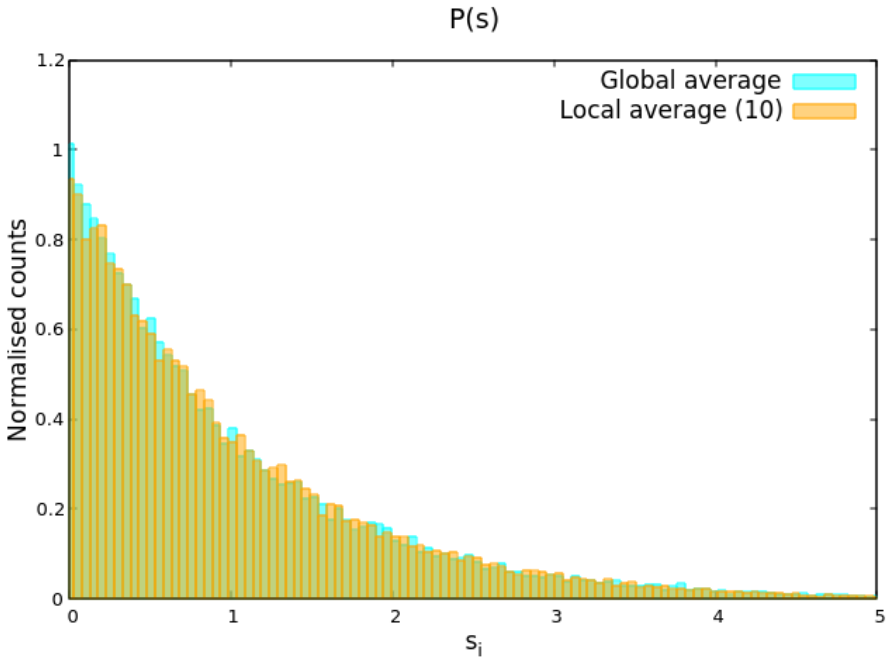


Figure 2: Distribution of normalised eigenvalue spacings for diagonal matrices with random real entries. The orange set is obtained normalising with a local average in a neighbourhood of each spacing of 20 elements, for the cyan dataset a global average is considered.

**Fitting the probability distribution.** The differences qualitatively described in the previous subsection can be quantified by fitting the distribution with the function $P(s)$ of Equation 2.

$$P(s) = as^{\alpha} \exp\left(-bs^{\beta}\right)$$

This is done through `gnuplot` and its `fit` function; the results are displayed in Figures 3, 4, 5, 6.

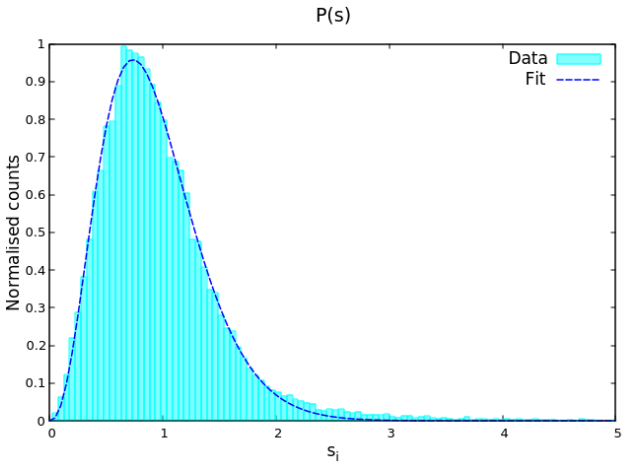


Figure 3: Fit of the distribution of normalised eigenvalue spacings for Hermitian matrices with random complex entries, with global average as normalising factor.

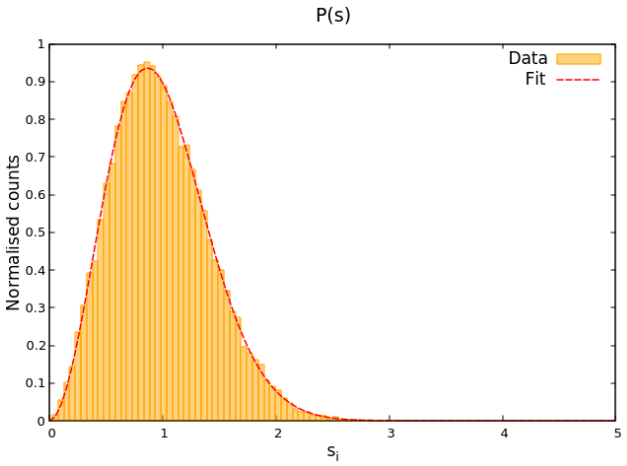


Figure 4: Fit of the distribution of normalised eigenvalue spacings for Hermitian matrices with random complex entries, with local average as normalising factor.

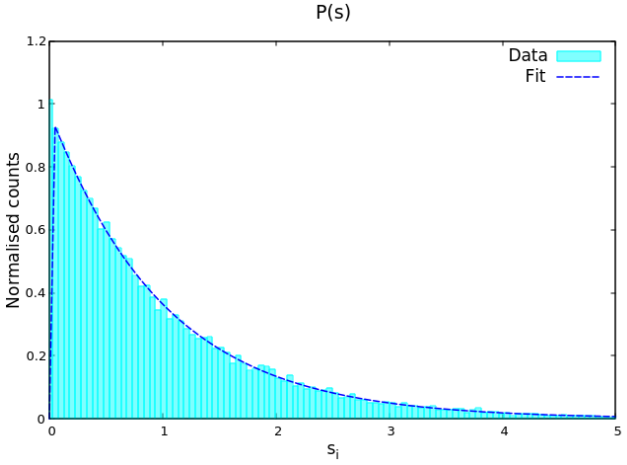


Figure 5: Fit of the distribution of normalised eigenvalue spacings for diagonal matrices with random real entries, with global average as normalising factor.

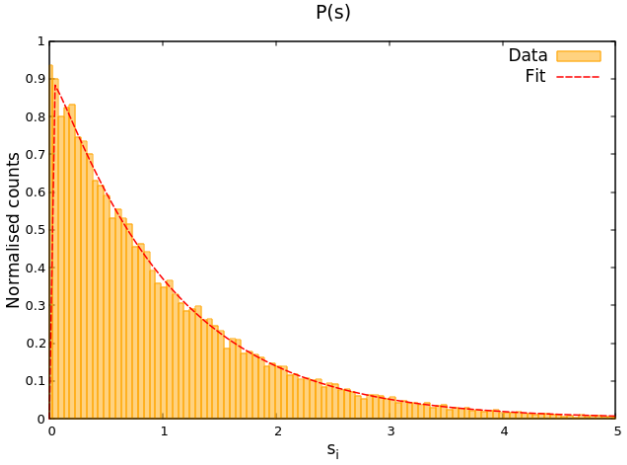


Figure 6: Fit of the distribution of normalised eigenvalue spacings for diagonal matrices with random real entries, with local average as normalising factor.

## Results

The fit results are reported in Table 1. $A$ represents parameters estimated for hermitian matrices, $D$ for diagonal ones, the pedices points out the type of normalization adopted. The second column displays some theoretical results valid for hermitian matrices.

| Parameter | $A_{th}$ | $A_{glob}$ | $A_{loc}$ | $D_{glob}$ | $D_{loc}$ |
|---|---|---|---|---|---|
| $a$ | $32/\pi^2 \approx 3.242$ | $13 \pm 3$ | $3.2 \pm 0.2$ | $1.0 \pm 0.5$ | $1.0 \pm 0.5$ |
| $b$ | $4/\pi \approx 1.273$ | $2.8 \pm 0.3$ | $1.28 \pm 0.06$ | $1.0 \pm 0.6$ | $1.0 \pm 0.5$ |
| $\alpha$ | 2 | $2.5 \pm 0.1$ | $1.90 \pm 0.04$ | $0.0 \pm 0.2$ | $0.0 \pm 0.2$ |
| $\beta$ | 2 | $1.32 \pm 0.07$ | $1.97 \pm 0.04$ | $1.0 \pm 0.4$ | $1.0 \pm 0.3$ |

Table 1: Fit parameters and associated errors computed by `gnuplot`.

Looking at the fit results, one easily notices the difference between hermitian and diagonal ma-

trices. For the latter, the parameters fitted suggest that the distribution is simply an exponential one, for both the normalisation techniques.

As far as hermitian matrices are concerned, on the other hand, it can be observed that the better method for normalising proved to be the local average: in fact, the parameters derived are significantly closer to the theoretical ones, the fit also shows better convergence properties and the errors on the estimated parameters are smaller.

## Self evaluation

This exercise allows to learn about solving an eigenproblem using `LAPACK`, which is a powerful and efficient tool for linear algebra problems.

The main issues observed in this exercise were about the fitting process: without a proper parameter tuning, `gnuplot` is unable to give correct results.

In conclusion, some interesting results were derived about random matrix theory, consistent with the known theory and with the code implementation.