Week 5: Eigenproblem and Random Matrix Theory

Alice Pagano

(Dated: November 17, 2020)

In this Report, we compute the distribution of normalized spacing between eigenvalues for a random hermitian matrix and a diagonal matrix. We fit the obtained distribution with a curve and we analyze their behavior as a function of different level constants used to compute the local average between eigenvalues.

I. THEORY

A **diagonal matrix** is a matrix in which the entries outside the main diagonal are all zero; the term usually refers to square matrices.

A **hermitian matrix** is a complex square matrix that is equal to its own conjugate transpose, that is:

$$A \text{ Hermitian} \iff a_{ij} = \bar{a}_{ji} \tag{1}$$

The finite-dimensional **spectral theorem** says that any Hermitian matrix can be diagonalized by a unitary matrix, and that the resulting diagonal matrix has only real entries. This implies that all *eigenvalues* of a Hermitian matrix A with dimension n are *real*, and that A has n *linearly independent eigenvectors*. Moreover, a convenient way to store in a computer an Hermitian matrix is by exploiting the so called **packed storage matrix**. It is a more compact way than an n-by-n rectangular array by exploiting the special structure of the hermitian matrix: the lower triangle of A is stored column-by-column in vector AP.

Let us diagonalize matrix A and let us suppose to obtain n eigenvalues λ_i stored in *crescent order*. The **normalized spacing** between eigenvalues s_i is defined as:

$$s_i = \frac{\Delta\lambda_i}{\langle \Delta\lambda_i \rangle} \qquad \text{with} \quad \Delta\lambda_i = \lambda_{i+1} - \lambda_i \tag{2}$$

where the average spacing $\langle \Delta\lambda_i \rangle$ can be computed *globally* (by averaging over all spacing $\Delta\lambda_i$) or *locally*, i.e. over a different number of levels around λ_i (as $n/100$, $n/50$, $n/10$ and so on).

The **distribution** $P(s)$ of normalized spacing s_i for a random Hermitian matrix, or for a random diagonal matrix, can be fitted with the following function:

$$P(s) = a s^{\alpha} \exp\left(-b s^{\beta}\right) \tag{3}$$

Moreover, for each type of matrix we can compute the average $\langle r \rangle$ of:

$$r_i = \frac{\min(\Delta\lambda_i, \Delta\lambda_{i+1})}{\max(\Delta\lambda_i, \Delta\lambda_{i+1})} \tag{4}$$

and compare the results for the two different cases.

II. CODE DEVELOPMENT

In order to study both hermitian and diagonal matrix of size n, we develop one different program for each type of matrix. In particular, the two programs differ only for the matrix initialization, as:

- in "hermitian.f90" we define a complex random vector `AP` of size $np = n(n+1)/2$ which is initialized trough Lapack `SUBROUTINE` **`zlarnv`** and which stores the lower triangle of `A`. Then, we unpack the vector `AP` to matrix `A`.

```
! initialize random vector AP
call zlarnv(3,iseed,np,AP)

! unpacking matrix AP to A
do jj=1,n
    k = jj*(jj-1)/2
    A(1:jj,jj) = AP(1+k:jj+k)
    A(jj,1:jj-1) = conjg(AP(1+k:jj-1+k))
end do
```

- in "diagonal.f90" we define a real random vector `AP` of size n which is initialized trough Lapack `SUBROUTINE` **`dlarnv`** and which stores the diagonal of `A`. Then, we unpack the vector `AP` to matrix `A`.

```
! initialize random vector AP
call dlarnv(3,iseed,np,AP)

! unpacking matrix AP to A
do ii=1,n
    A(ii,ii) = cmplx(AP(ii),0)
end do
```

Then, the structure of the two main programs is exactly the same. In order, when each program is executed:

1. the dimension of the matrix `n`, the number of bins for the histogram `nbins`, the number of times the matrix is computed `ntime` and the fixed `level` for the local average are taken in input;
2. then, for `ntime` times:
   (a) matrix `A` is random initialized with a **normal distribution** in $(0, 1)$ (as explained before);
   (b) eigenvalues are computed and ordered in *ascending order* using Lapack `SUBROUTINE` **`zheev`**;
   (c) eigenvalues spacing are computed (Eq. (2)) and stored in array `delta_eig` of dimension `n-1`;
   (d) then, we compute the global average `aver_delta_eig` and the corresponding global normalized spacing `norm_delta_eig`. We add the normalized spacing into an array `si` of dimension `ntime(n-1)` which stores the results of global normalized spacing for each execution;
   (e) after that, we compute the local average of spacing array `local_aver_delta_eig` of dimension `n-1`. In particular, let us consider a generic eigenvalue λ_i, to compute its local average, we average over the eigenvalue itself, `level` eigenvalues at its left and `level` eigenvalues at its right, for a total of $2\texttt{level}+1$ elements. We have to pay attention to the eigenvalues at the extremes, which do not have enough elements at their left or right. To solve that problem, for each eigenvalue whose index is lower than `level`, or higher than `n-1-level`, we assume their average to be equal to the average of the first, or the last, `level+1` elements. Then, we compute the normalized spacing `local_norm_delta_eig` and we add them into an array `local_si` of dimension `ntime(n-1)` which stores the results of local normalized spacing for each execution;

```
! compute local average spacing
aver_sx = (eig(2*level+1+1)-eig(1)) / (2*level+1)
aver_dx = (eig(n)-eig(n-2*level-1)) / (2*level+1)

do ii=1,n-1
    if (ii <= level) then
        local_aver_delta_eig(ii) = aver_sx
    else if (ii > n-1-level) then
        local_aver_delta_eig(ii) = aver_dx
    else
        local_aver_delta_eig(ii) = (eig(ii+level+1)-eig(ii-level)) / (2*level+1)
    endif
end do
```

   (f) at the end, we compute $\langle r \rangle$ (Eq. (4)) and we write the result for each execution into a file;
3. at the end, we compute the probability function for both the global and local normalized spacing by using `SUBROUTINE` **`create_histogram`** of the user-defined `MODULE` **`histogram`**. This subroutine, which will explain in more details later, takes as input the entries of the histogram, the fixed number of bins `nbins` and a specified range $[\min, \max]$ which we fix to $[0, 5]$. The bin centers and the normalized entries are printed into a file.

In the file "histogram.f90", we define `MODULE` **`histogram`** which contains the `SUBROUTINE` **`create_histogram`**. In particular, when the last subroutine is called, the following operations occurs in order:

1. we fix the bin width `dx` (equal for all the bins) to the range `(max-min)` divided by the number of bins `nbins`;
2. we compute bin centers `bin_centers` and left bin edges `bin_edges` (plus the most right edge);

3. we fill bins with events and we store the number of events for each bin in the array `hist`;

4. we normalize the histogram by dividing the entries of `hist` by the total histogram area `tot_area`, obtaining the array `norm_hist`;

5. the obtained results are printed into a file whose name is dictaed by `file_name` variable in a folder called `folder_name`.

```
subroutine create_histogram(events,n_bins,min,max,file_name,folder_name)
    ...
    real(8), dimension(n_bins) :: bin_centers
    real(8), dimension(n_bins+1) :: bin_edges
    real(8), dimension(n_bins) :: hist, norm_hist
    ...
    ! compute bin width
    dx = (max-min) / n_bins

    ! compute bin centers
    do ii=1,n_bins
        bin_centers(ii) = min + dx/2 + (ii-1)*dx
    end do

    ! compute left bin edges (plus the most right edge)
    do ii=1,n_bins+1
        bin_edges(ii) = min + (ii-1)*dx
    end do

    ! compute histogram
    do ii=1,n_bins
        ! fill bins with events
        do jj=1,size(events,1)
            if (events(jj)>=bin_edges(ii) .and. events(jj)<=bin_edges(ii+1)) then
                hist(ii) = hist(ii) + 1
            end if
        end do
    end do

    ! compute total histogram area
    tot_area = size(events) * dx

    ! compute normalized histogram
    do ii=1,n_bins
        norm_hist(ii) = hist(ii)/tot_area
    end do

    ...

    ! write data into file
    do ii=1,n_bins
        write(file,*) bin_edges(ii), bin_centers(ii), hist(ii), norm_hist(ii)
    end do

end subroutine create_histogram
```

After that, we develop a Python script "script.py" in which:

- we fix the matrix size `N`, the number of times `Ntime` the matrix is computed, the number of bins for the histogram `Nbins` and a list of `level` we want to study;

- then, for each `lev` in `level`, we execute "hermitian" and "diagonal" executables.

- finally, for both diagonal and hermitian matrices, we call a Gnuplot script "plot_hist.p" which produces a plot of normalized spacing distribution for global and local average (for the different levels) with the corresponding fit (Eq. (3)).

```
executable = ['hermitian','diagonal']

# Compile programs
make_command = ["make","all"]
make_proc = subprocess.run(make_command)

# Define variables
N      = 5000
Ntime  = 10
Nbins  = 100

level = [10,50,100,250,1000]

for lev in level:
    for exe in executable:
        print("Type  : ", exe)
        print("N     : ", N)
        print("Nbins : ", Nbins)
        print("Level : ", lev)
        print("Ntime : ", Ntime,'\n')

        result = subprocess.run(['./'+exe,str(N),str(Nbins),str(Ntime),str(lev)])

# Plot histogram
for exe in executable:
    result = subprocess.run(['gnuplot','-e',"file_name='"+exe,'plot_hist.p'])
```

III. RESULTS

In order to have enough statistics for the normalized spacing distribution, we choose a large matrix size of `N=5000` and we compute it `Ntime=10` times. In this way, we obtain 50000 events for the histogram for which we fix `Nbins=100`. The plots obtained are illustrated in Fig. 1. We note that normalized spacing s_i, for both hermitian and diagonal matrices, distributes accordingly to Eq. (3), but with fit parameters quite different. This result is in accordance with the theory since both of them belong to the same ensamble of symmetric matrices. For hermitian matrices, we note that there is only a slightly difference between the distribution for different levels. Instead, for diagonal matrices the difference is more evident.

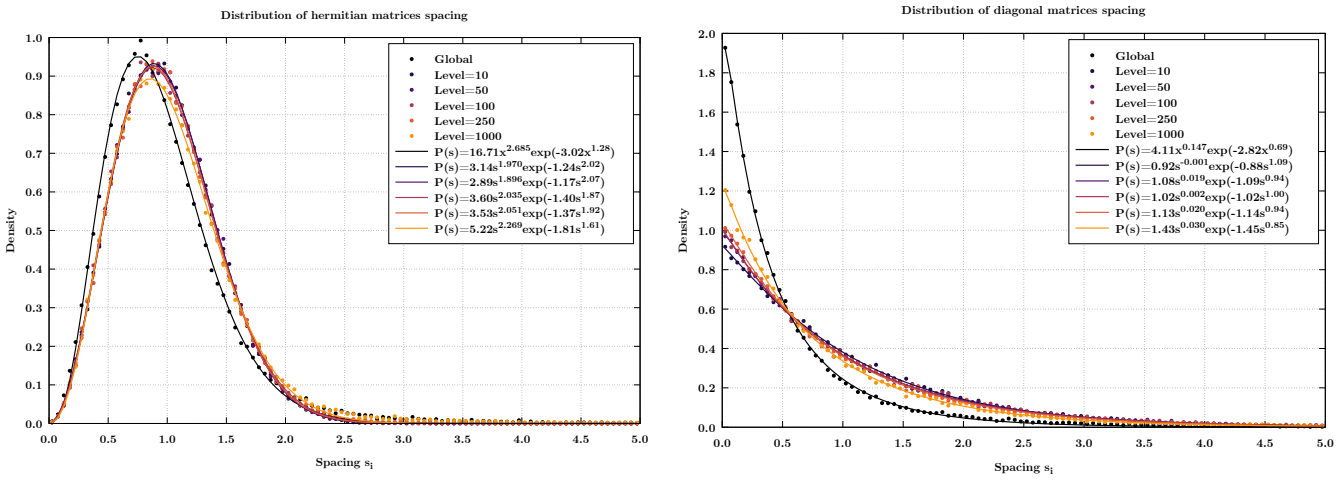

FIG. 1 Plots of normalized spacing distribution for global and local average (with different levels choosen). **Left:** case of hermitian matrices. **Right:** case of diagonal matrices.

Moreover, we compute the $\langle r \rangle$ for the different matrices. The results are:

$$\langle r \rangle_{\text{hermitian}} = 0.600 \pm 0.004, \qquad \langle r \rangle_{\text{diagonal}} = 0.387 \pm 0.003 \tag{5}$$

IV. SELF-EVALUATION

In a further development of the code, it would be interesting to study the behavior of the spacing distribution for matrices which are random initialized with different distribution (not only normal one) to see if their distribution is in accordance with Eq. (3). Moreover, it would be interesting to study other types of random matrices to explore in more details random matrix theory.