Random matrices

Abstract

In this work we consider a complex Hermitian or a real diagonal matrix and we compute its eigenvalues and the normalized spacings between them. Then we study the distribution of these spacings.

1 Theory

A random matrix is a matrix in which some or all elements are random variables. From Wigner theory we know that if we consider the ordered sequence of eigenvalues $\{\lambda_i\}$ and we define the normalized spacings

$$s_i = \frac{\Delta\lambda_i}{\overline{\Delta\lambda}},$$

where $\Delta\lambda_i = \lambda_{i+1} - \lambda_i$ and $\overline{\Delta\lambda}$ is the average spacing, then the probability distribution of these spacings is approximately given by

$$P(s) = as^\alpha e^{-bs^\beta}.$$

In particular, we will consider Hermitian random matrices whose entries are independently distributed complex random variables, which constitute the so called Gaussian Unitary Ensemble (GUE). For a matrix belonging to the GUE, $P(s)$ has the following expression:

$$P(s) = \frac{32}{\pi^2} s^2 e^{-\frac{4}{\pi}s^2}.$$

2 Code development

2.1 Eigenproblem

In the first part we consider a Hermitian matrix of size N, we diagonalize it and we store its eigenvalues. Then we compute the normalized spacings between eigenvalues.

In order to initialize the matrix, we define a subroutine `MatrixInit`. One of its input arguments is the character flag `which_matrix`, which is used to choose the type of matrix to initialize:

- if `which_matrix == "h"`, a Hermitian matrix with complex entries will be initialized. Both real and imaginary parts of the entries are independently uniformly distributed between -1 and 1;
- if `which_matrix == "d"`, a real diagonal matrix with uniformly distributed between -1 and 1 entries will be initialized. This type of matrix will be used in the second part of the exercise.

In order to diagonalize the matrix and compute the eigenvalues we use the LAPACK subroutine `cheev`.

```
1    subroutine ComputeEigenvalues(matr, eig, info)
2        complex, dimension(:, :), intent(in) :: matr
3        real*4, dimension(size(matr, 1)) :: eig
4        real*4, dimension(:), allocatable :: rwork
5        complex, dimension(:), allocatable :: work
```

```
        character(1) :: jobz, uplo
        integer :: n, lda, lwork, info
        n = size(matr, 1)
        lda = size(matr, 1)
        lwork = 2*size(matr, 1) - 1
        jobz = "N"
        uplo = "U"
        allocate(rwork(3*size(matr, 1)-2))
        allocate(work(2*size(matr, 1)-1))
        call cheev(jobz, uplo, n, matr, lda, eig, work, lwork, rwork, info)
        deallocate(rwork, work)
        return
    end subroutine ComputeEigenvalues
```

The subroutine `cheev` takes in input many parameters, one of which is `info`. This is an integer which takes non-zero values in case of failure in the computation of the eigenvalues. We make use of it in the main program, to be sure that the computation is done with success.

```
    ...
    info = 1
        do while (info .ne. 0)
            ! initialize the matrix
            M = MatrixInit(N, which_matrix)
            ! call the subroutine to compute the eigenvalues
            call ComputeEigenvalues(M, eig, info)
        end do
    ...
```

The spacings are computed by means of a dedicated subroutine `ComputeSpacings` and then are normalized using another subroutine, `ComputeNormSpacings`. Notice that the `cheev` subroutine returns the eigenvalues sorted in ascending order, so there is no need for further sorting.

Additionally, we compute the average spacing $\overline{\Delta\lambda}$ locally, i.e. over different number of levels around λ_i ($N/100$, $N/50$, $N/10$, $N/5$). This is done using the `ComputeNormSpacingsLocal` subroutine.

2.2 Random Matrix Theory

In the second part of the exercise we want to study the distribution $P(s)$ of the normalized spacings s_i obtained from different random matrices. In order to do that, we compute the s_i for a certain number of number of matrices (`ntrials`, which is specified by the user) and we store all them in a vector in case of global normalization, in a matrix in case of local normalization (the columns corresponding to the different locality levels).

```
    do trial = 1, ntrials
        print *, "Computing the spacings for matrix number", trial
        ! use the info flag of the subroutine 'cheev' to check if the
        ! diagonalization has been successful
        info = 1
        do while (info .ne. 0)
            ! initialize the matrix
            M = MatrixInit(N, which_matrix)
            ! call the subroutine to compute the eigenvalues
            call ComputeEigenvalues(M, eig, info)
        end do
        ! call the function to compute the spacings
        if (which_spacing == "g") then
            norm_spacings = ComputeNormSpacings(eig)
            ! add the spacings we just computed to the vector of all the spacings
            all_s(1+(trial-1)*(N-2):trial*(N-2)) = norm_spacings
```

```
        else if (which_spacing == "l") then
            do ii = 1, size(div)
                local_norm_spacings(:,ii)=ComputeNormSpacingsLocal(eig,N/div(ii))
                all_local_s(1+(trial-1)*(N-2):trial*(N-2),ii)=local_norm_spacings
    (:, ii)
            end do
        end if
    end do
```

The distribution of the spacings is computed using the subroutine `ComputePDF`, in which we use the data to build a histogram and then we normalize it dividing by the area of the bins. The subroutine takes in input the array with the data (`x`), the number of bins (`nbins`) and two arrays to fill, one with the bin centers (`bin_centers`) and one with the points of the distribution (`dist`).

```
    subroutine ComputePDF(x, nbins, dist, bin_centers)
        real*4, dimension(:), intent(in) :: x
        integer, intent(in) :: nbins
        real*4 :: bin_size, bin_increment
        real*4, dimension(:), allocatable :: right_edge, bin_centers
        integer :: ii, jj
        integer, dimension(:), allocatable :: counts
        real*4, dimension(:), allocatable :: dist
        allocate(right_edge(nbins))
        allocate(counts(nbins))
        bin_size = (maxval(x) - minval(x)) / nbins
        bin_increment = minval(x)
        do ii = 1, nbins
            bin_increment = bin_increment + bin_size
            right_edge(ii) = bin_increment
        end do
        bin_centers = right_edge - (bin_size/2)
        counts = 0
        do ii = 1, size(x, 1)
            do jj = 1, nbins - 1
                if (x(ii) .le. right_edge(jj)) then
                    counts(jj) = counts(jj) + 1
                    exit
                end if
            end do
            if (x(ii) .ge. right_edge(nbins-1)) counts(nbins) = counts(nbins) + 1
        end do
        dist = real(counts) / (sum(counts)*bin_size)
        deallocate(right_edge)
        deallocate(counts)
    end subroutine ComputePDF
```

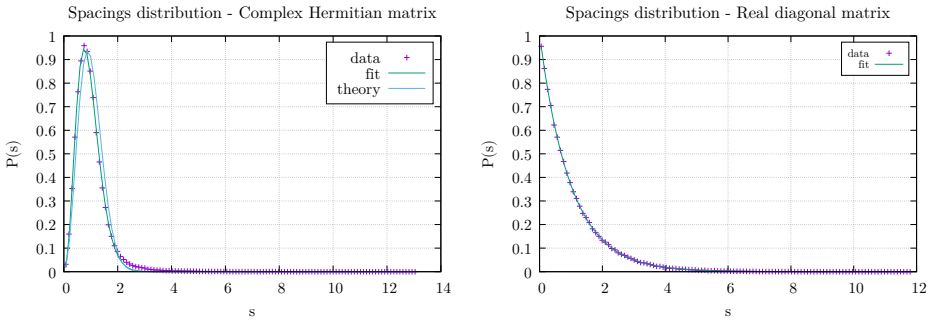
The number of bins to build the histogram is chosen in the main program using the Rice rule: $\text{nbins} = \lceil 2\sqrt[3]{n} \rceil$.
We compute $P(s)$ both for Hermitian matrices and for real diagonal ones, and then we fit the distributions with the function $P(s) = as^{\alpha}e^{-bs^{\beta}}$ using a gnuplot script.

Finally, the computation of $\langle r \rangle$ is implemented in a dedicated subroutine, `ComputeR`.

In the main program we ask the user to enter the dimension of the matrix, its type and if he/she wants to consider global-normalized or local-normalized spacings. Finally, he/she will be asked to enter the number of matrices to generate. According to these choices – and to the locality level – the results will be saved in different output files.

3 Results

The computed distributions, together with the fits, are shown in Figure 1. The fit results are shown in Table 1. All the results are obtained from 100 samples (`ntrials = 100`) of size $N = 2000$.



(a) Complex Hermitian matrices.

(b) Real diagonal matrices.

Figure 1: Distributions of normalized spacings between eigenvalues for complex Hermitian and real diagonal matrices obtained with 100 samples of size $N = 2000$.

Matrix	a	α	b	β
Hermitian	12 ± 2	2.50 ± 0.09	2.6 ± 0.2	1.37 ± 0.05
Diagonal	1.00 ± 0.01	-0.002 ± 0.003	1.00 ± 0.01	0.99 ± 0.07

Table 1: Fit results for global-normalized spacings distributions obtained from 100 samples of size $N = 2000$.

For the Hermitian case, we can compare these results with the known values from theory, which are:

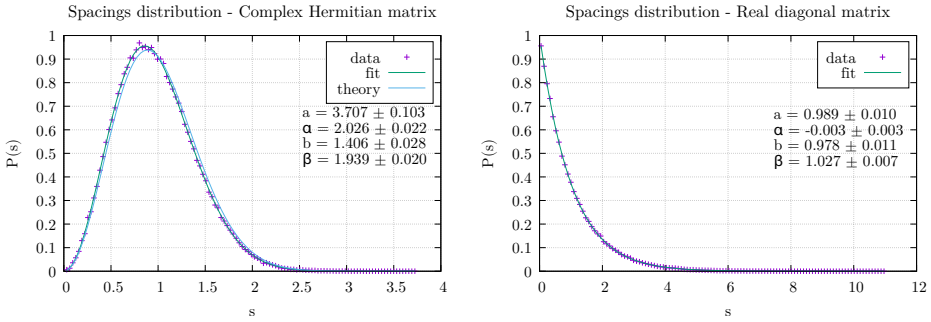
$$a \approx 3.24, \qquad \alpha = 2, \qquad b \approx 1.27, \qquad \beta = 2.$$

We see that the fitted parameter are not exactly compatible with the theory. This is probably due to the fact that we need to consider more samples to get a better fit. However, if we consider the locally-normalized spacings, we get way better results. For example, in Table 2 we report the fit results for a locality level $N/100$.

Matrix	a	α	b	β
Hermitian	3.7 ± 0.1	2.03 ± 0.02	1.41 ± 0.028	1.94 ± 0.02
Diagonal	0.99 ± 0.01	-0.003 ± 0.003	0.98 ± 0.01	1.027 ± 0.007

Table 2: Fit results for locally-normalized spacings distributions obtained from 100 samples of size $N = 2000$, locality level $N/100$.

So, we conclude that considering locally-normalized spacings we get results that better resemble the theory.



(a) Complex Hermitian matrices. (b) Real diagonal matrices.

Figure 2: Distributions of locally-normalized spacings between eigenvalues for complex Hermitian and real diagonal matrices obtained with 100 samples of size $N = 2000$, locality level $N/100$.

Locality level	Hermitian matrix	Diagonal matrix
N	0.596484065	0.375744760
$N/5$	0.596745312	0.376050562
$N/10$	0.596747220	0.375988901
$N/50$	0.596536160	0.375978142
$N/100$	0.596696019	0.375708878

Table 3: Values of $\langle r \rangle$ for different type of matrices and locality levels.

The values of $\langle r \rangle$ for different locality levels are shown if Table 3[1]. Notice that we obtain the same results independently of the locality level. These results are in line with what we expect from theory[2], that is $r_{diag} \approx 0.386$ and $r_{herm} \approx 0.603$.

4 Self evaluation

This exercise was very instructive since it made me learn how solve an eigenproblem in Fortran using LAPACK. I admit that this assignment was quite challenging, partly because the topic – i.e. Random Matrix Theory – was fairly new for me and partly because the writing of the code itself was rather demanding. However, once all worked, I was quite satisfied.

One possible improvement could be to fix one little problem I had with the fit of the hermitian matrix's global-normalized spacings. In fact, to avoid errors, one has to provide an initial guess for the parameters. However, these initial guesses do not work when we fit other data. So, a dedicated gnuplot file for this dataset could be written.

[1] In the program, the values of $\langle r \rangle$ are computed using the normalized spacings. This should not be a problem because if we consider the global normalization we are dividing both the denominator and the nominator for the same quantity, which cancels out. If we instead consider local normalizations we have slightly different values, but since the eigenvalues are a lot and they are in increasing order, it is an educated guess to consider $\overline{\Delta\lambda}_i \sim \overline{\Delta\lambda}_{i+1}$

[2] Y. Y. Atas, E. Bogomolny, O. Giraud, and G. Roux, Phys. Rev. Lett. **110**, 084101 (2013).