Exercise 5

Abstract

This project consists in evaluating eigenvalues of random diagonal and hermitian matrices. Then the the normalized spacing of the eigenvalues have been computed to study their PDF and other interesting quantities.

1 Theory

Diagonal Matrix

Given a square matrix A with n columns and n rows, and let a_{ij} be its elements. This is diagonal if and only if

$$a_{ij} = 0 \qquad \forall i, j \in \{1, 2, ..., n\}, i \neq j \tag{1}$$

From the secular equation we can compute its eigenvalues:

$$det(A - \lambda \mathbb{I}) = \prod_{i=1}^{n} (a_{ii} - \lambda) = 0 \tag{2}$$

and we get

$$\lambda_i = a_{ii} \qquad \forall i \in \{1, 2, ..., n\} \tag{3}$$

Hermitian Matrix

Let A be a matrix, this is hermitian if and only if

$$A = \bar{A}^T \tag{4}$$

where $\bar{A}^T$ represent the complex conjugate transposed matrix of A. It is possible to show that the eigenvalues of an hermitian matrix are real. In fact given an

eigenvalue λ_i, and its respective eigenvector $\mathbf{x}$ we can write

$$\begin{aligned} A\mathbf{x} = \lambda_i \mathbf{x} \qquad &\rightarrow \qquad \bar{\mathbf{x}}^T A\mathbf{x} = \bar{\mathbf{x}}^T \lambda_i \mathbf{x} \\ \bar{\mathbf{x}}^T (A\mathbf{x}) = \lambda_i \|\mathbf{x}\| \qquad &\rightarrow \qquad (A\mathbf{x})^T \bar{\mathbf{x}} = \lambda_i \|\mathbf{x}\| \end{aligned}$$

Applying the complex conjugate operation on both sides we obtain

$$\mathbf{x}^T A^T \bar{\mathbf{x}} = \lambda_i \|\mathbf{x}\| \qquad \rightarrow \qquad \bar{\mathbf{x}}^T \bar{A}^T \mathbf{x} = \bar{\lambda}_i \|\mathbf{x}\|$$

And since A is hermitian

$$\begin{aligned} &\bar{\mathbf{x}}^T A\mathbf{x} = \bar{\lambda}_i \|\mathbf{x}\| = \lambda_i \|\mathbf{x}\| \\ &\bar{\lambda}_i = \lambda_i \end{aligned}$$

Probability density function from a dataset of measures

Les $S = \{s_1, s_2.., s_N\}$ be the set of N measures of a variable $s \in A \subset \mathbb{R}$. If we divide the space A in M fixed length intervals m_i, where $1 < i < M$ we can compute $L_i = S \cap m_i$ and recover then $n_i = \#L_i$ the number of measure of s inside each interval m_i. From the definition of probability we can get $P_i \approx \frac{n_i}{N}$ (the sign $=$ can be used just for infinitely large S), the probability that a measure of s lies inside m_i. Given $\rho(s)$ the PDF of s this can be evaluated numerically as follows.

$$\int_{m_i} \rho(x)dx = P_i \qquad \rho(s_i^*) \approx \frac{P_i}{\Delta s} = \frac{n_i}{N\Delta s} \tag{5}$$

where s_i^* is the center of the i-th interval m_i , and Δs the size of the interval.

2 Code Development

As first the code generates random matrices, using two different functions for the hermitian and diagonal cases. Once the matrices are built their eigenvalues are computed and sorted in ascending order. In the hermitian matrix case, this is performed trough the **zheev** subroutine present in the *lapack* library, while for the diagonal matrix, since the operation was trivial, I wrote a custom function in which the diagonal elements of the matrix are sorted using the bubble-sort algorithm. Its code follows: (**tmphelp** module, **deigs** function).

```
1 !################################################
2 function deigs(a) result(eigs)
3 !################################################
```

```
        implicit none
        real*8, dimension(:,:) :: a
        real*8 :: eigs(size(a, dim=1)), tmp
        integer :: ii , flag, dim
        !compute eigenvalues
        dim = size(a, dim=1)
        call breakifn("Square Matrix is needed " , size(a, dim=1)
            .eq. size(a, dim=2))
        do ii = 1, dim
               eigs(ii) = a(ii,ii)
        end do
        !sort
        flag =1
        do while (flag .eq. 1)
            flag = 0
            do ii = 1, dim-1
                if (eigs(ii) .ge. eigs(ii+1)) then
                    tmp = eigs(ii)
                    eigs(ii) = eigs(ii+1)
                    eigs(ii+1)= tmp
                    flag=1
                    !print*, ii
                end if
            end do
        end do
    end function
```

Then I derived the normalized spacings $s_i = {}^{(\lambda_{i+1}-\lambda_{i+1})}/_{\bar{\Delta\lambda}}$, and for the optional request also the ones normalized with the moving average. In particular these have been evaluated with 4 periods: 10, 50, 100, 150 steps.

To obtain a consistent dataset of the quantities (absolute and moving average) for both diagonal and hermitian matrices, I performed the previous operation on 20 matrix of 1000 rows/columns.

The spacings of the eigenvalues, for both moving average and absolute mean normalization, are merged in arrays obtaining 10 sets of 19800 values: 4 cases for the moving average and 1 for the absolute mean, for both diagonal and hermitian matrices.

The discrete points of the probability density function of this data-sets have been evaluated using a fortran function **density**, which returns a matrix with two columns representing the center of the bins, and the value of the PDF.

```
!############################################################
function density(array, nob, minv, maxv) result(data)
```

```
!#############################################################
    real*8 , dimension(:) :: array
    integer ii, nob, _where,cnter
    real*8 maxv, minv, data(nob,2),delta
    delta = (maxv-minv)/dble(nob)
    cnter =0
    do ii = 1,size(array)
        _where = floor((array(ii)/delta) -minv +1)
        if ((_where .ge. 1) .and.(_where .le. nob)) then
            cnter = cnter +1
            data(_where,1) = data(_where,1)+1
        end if
    end do
    do ii = 1,nob
        data(ii,1) = data(ii,1)/(delta*cnter)
        data(ii,2) = minv+delta*(ii-1+0.5)
    end do
end function
```

The PDF data then have been fitted with the following function using *gnuplot*.

$$\rho(s) = as^{\alpha} exp(-bs^{\beta}) \tag{6}$$

The last operation performed consisted in the evaluation of $< r_{\alpha} >$, that represent the average value for all the r_i^{α} obtained from different matrices.

$$r_i^{\alpha} = \frac{min(\Delta\lambda_i^{\alpha}, \Delta\lambda_{i+1}^{\alpha})}{max(\Delta\lambda_i^{\alpha}, \Delta\lambda_{i+1}^{\alpha})} \tag{7}$$

The attribute α distinguish between diagonal and hermitian cases.

3 Results

In Fig.1 we can appreciate data, and fit performed on the PDF of diagonal and hermitian matrices using the absolute mean normalization. At first sight the results seems good but looking at the following table we notice that the error in the estimate of the parameter a [6] has a magnitude $\approx 65\%$ of the value for the hermitian case.

$\times$	*Hermitian*	*Diagonal*
a	121783 +/- 7.93e+04	1.03312 +/- 0.05005
α	2.38993 +/- 0.1461	0.0176987 +/- 0.01436
b	130.46 +/- 16.95	1.01593 +/- 0.05278
β	1.42486 +/- 0.08197	1.0024 +/- 0.03519

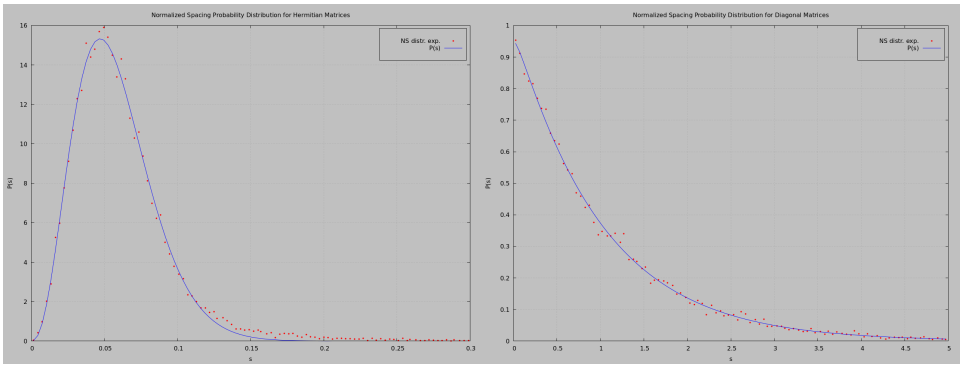

Figure 1: Absolute mean normalization and fit

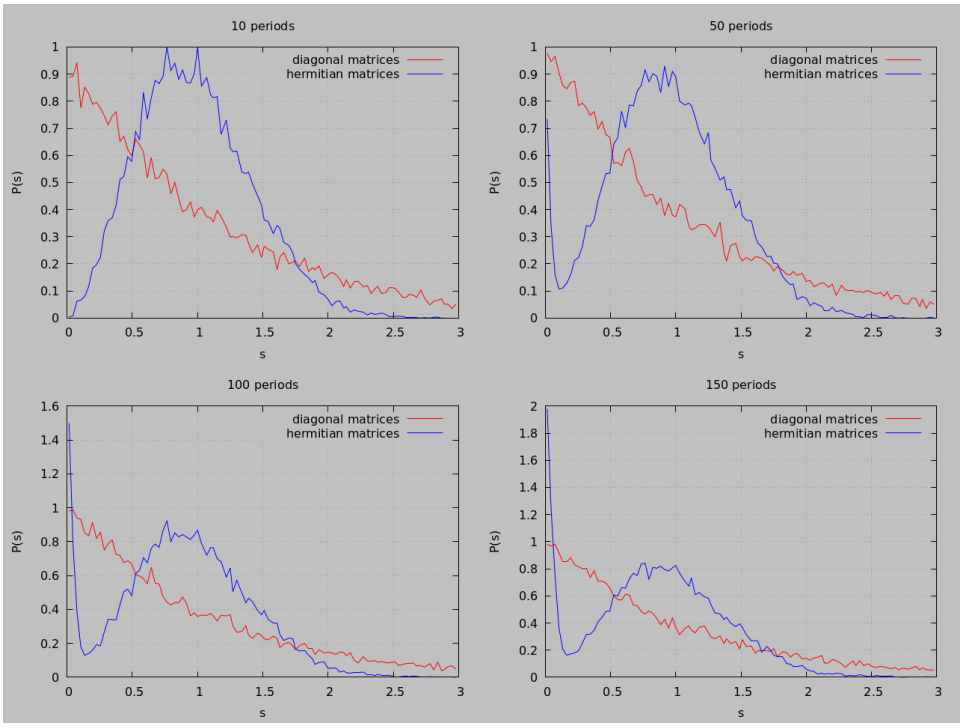

Figure 2: Moving average normalization

The reasons are unclear but a clue of this behaviour could be the use of the least-square algorithm, that assumes a priori that the uncertainty in the counts of the histogram is the same for every bin. This in general is not true, and a χ^2 fit should be preferred. For the moving average normalization the results can be seen in Fig.1, and we notice that increasing the period an edge near the 0 value start to appear. This probably it is caused by the fact that are present large spacings that increase the magnitude of the denominator, leading many spacings near 0.

About the quantity $< r_\alpha >$ the results can be seen in the following table. What we see is that the values of the quantity differ taking in account hermitian and diagonal cases.

$\times$	*Hermitian*	*Diagonal*
$< r >$	0.599	0.384

4 Self Development

In this project I decided to use a different approach in the compilation of the code. In fact I found myself facing an issue of multi-declaration of the **debugqic** module, this was due to the fact that this one is intended as a sort of environment for the other modules, which exploit its debug functionalities. Since in every of the modules using **debugqic**, appears an *include* statement I replaced it with c++ precompiler *#include* allowing conditional compiling of the code trough *#ifndef*, *#define*.