Eigenproblem & Random Matrix Theory

Alberto Chimenti

Exercise n.5
Information Theory and Computation - 2019

Abstract

The following report is intended to discuss the implementation of methods and usage of LAPACK built-in functions to perform eigendecomposition and study some properties of random matrices.

1 Theory

Here we are interested in performing matrix diagonalization and study the distribution of the spacings between the ordered found eigenvalues.

The procedure has been performed for both Hermitian complex random matrices and diagonal real ones.

Given the ordered set of eigenvalues, the spacings were computed as:

$$s_i = \frac{\Delta\lambda}{\overline{\Delta\lambda}} = \frac{\lambda_{i+1} - \lambda_i}{\overline{\Delta\lambda}}$$

Therefore such quantities were counted in bins, normalized and used to fit their probability distribution. Such distribution is in the form:

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

This Poisson distribution parameters are different depending on the chosen random matrix type. In case of hermitian matrices with gaussian distributed random entries, one should obtain the Wigner-Dyson distribution

2 Code Development

2.1 Fortran Module

It uses the already implemented module to define type **cmatrix** which defines a complex matrix and initializes it by computing trace, adjoint and determinant. For computational cost reasons, in this utilization, the computation of the determinant was avoided.

Therefore some other features were added.

Initialization now adds the *matrix%eigenvals* object and allocates it with the right dimension.

Random Hermitian matrix generator was implemented in a function called **random_hermitian**. It generates then a complex two dimensional object with real random diagonal entries and conjugate random entries in the corresponding upper triangular and lower triangular parts. Moreover, an optional logical input was implemented, called **gauss**, which lets the user choose whether to generate random uniformly distributed entries or random gaussian distributed entries.

Gaussian distributed entries have been computed starting from random uniformly distributed numbers using Box-Muller method, therefore using:

$$z_1 = \sqrt{-2lnu_1}cos(2\pi u_2)$$

$$z_2 = \sqrt{-2lnu_1}sen(2\pi u_2)$$

where u_1 and u_2 are uniformely distributed.

Hermitianity check was performed using a subroutine called **check_hermitianity** which stops the program if all of the values, both imaginary and real part, of each entry correspond between the original matrix and its adjoint. Here it was very handy to perform calculations using the predefined type which automatically stores the *matrix%adj* object when initialized.

Eigenvalues were computed using an implemented function called **compute_eigenvals**. This function uses the **zheev** built-in subroutine in LAPACK.

```
call zheev('N', 'U', H%dim(1), temp_mat, H%dim(1), eig, work, lwork, rwork, info)
```

The subroutine call, destroys the original matrix given as an input and replaces it with either the upper or lower triangular component. For this reason, a temporary matrix was used in order to not loose the original matrix stored. The first two entries respectively state that the subroutine should: calculate the eigenvalues (without the eigenvectors), store the upper triangular matrix in place of the original object. The other ones are input and output variables such as the dimension of the matrix, the matrix object itself, its leading order and the vector used to store the eigenvalues. The last four refer to specifications for the computation and especially the last one **info** was useful to check the convergence of the algorithm after every run performed.

An interesting remark is that since we are dealing with hermitian matrices we expect the eigenvalues to be real. A problem found with the LAPACK subroutine was that if one decides to generalize the eigenvalues object to a double complex vector, the output values are stored in a sequential way not discerning the real entries from the imaginary ones but just storing the entries sequentially in the wrong position i.e. storing the second real eigenvalue in the imaginary part of the first one. Therefore it was necessary to define the output object **eig** as a double precision instead of a double complex.

Spacings were computed with **compute_spacings** function. It computes the normalized spacings as follows:

```
do ii=1, (N - 1)
    norm_spacings(ii) = eig(ii+1) - eig(ii) !computes space (NOT normalized)
    mean_spacing = mean_spacing + norm_spacings(ii) !starts computing mean spacing
end do
mean_spacing = mean_spacing /  (N - 1)
norm_spacings = norm_spacings / mean_spacing
```

Random diagonal real matrix generator was lastly implemented in the **random_real_diag** function. It also takes an optional logical input to select the random entries distribution.

2.2 Fortran Program

Finally the **main** program opens the specific files or creates new ones if they are not found, computes random matrices, eigenvalues and spacings for both of the implemented types (hermitian and diagonal) and stores the spacings values in separate files.

2.3 Python Script

The python script handles the optional compilation of the fortran script and the reset if needed of the output files. The latter is performed if the user after the prompted choice selects 'y' to run the fortran executable. In this case, the python script stores the selected matrix dimension in a file called *input.txt* and resets the output files needed to store the spacings. Finally it runs the fortran script using the input file as prompt input as many times as the ones selected by the user.

After obtaining the final output files containing the spacings for both cases, it launches the terminal command to run the gnuplot script which handles the plotting and fitting of the probability distributions.

2.4 Gnuplot Script

In order to compute the normalized counts for the probability distribution binning, I chosed to use the following approach:

```
# HIST
binwidth = 0.2
set boxwidth binwidth
set style fill solid 0.8
bin(x, width) = width*floor(x/width) + width/2


set table "hist_h.txt"
plot h_file u (bin($1,binwidth)):(1.0/(binwidth*N)) smooth freq with boxes
unset table
```

This method defines bin(x,width) as the bin value for x given the binwidth. Successively the set table command saves the output values from the plot function in the deprecated file. The plot statement uses the x's from the h_file which contains the spacings and by means of the **smooth freq** command, stores the counts of the number of x's with the same binning value. The y value is deprecated as $(1.0/(binwidth * N))$ to explicit the normalization in order to get a probability distribution.

Afterwards, the fitting is called over the values stored in the table file.

3 Results

The chosen matrix dimension was N=2000 with 100 iterations. The resulting plots and fits are presented in the following:

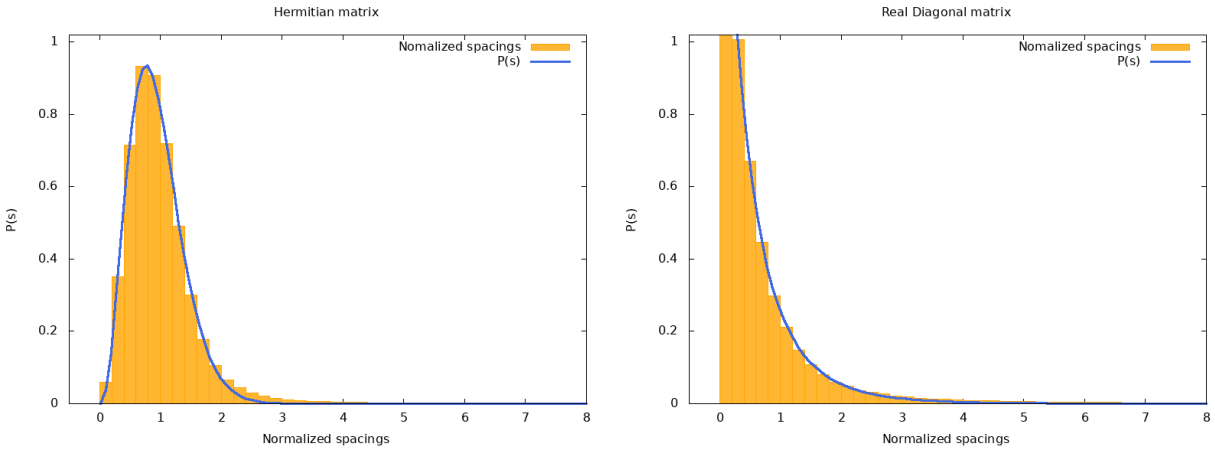

Figure 1: Gaussian Entries Fits

As expected one can see that the mean value of the spacing is around one in the hermitian case, while in the diagonal case the first value diverges.

In the case where the matrices entries where generated from a uniform distribution, the analysis encountered some problems probably concerning the normalization of the histograms. The gnuplot script was unable to perform the fit raising as error that it encountered some undefined values. This is probably due to some very high values of the spacings, but even by neglecting the smallest eigenvalue from the evaluation the problem persisted. It would probably need further analysis.

4 Self-evaluation

In this homework we learned how to deal with fortran libraries and in general getting accustomed with their error handling. Moreover, it was interesting and useful to reason on histogram plotting starting from a set of records and deal with normalization for the probability distribution. Also some new data handling was discovered in gnuplot.

A Gnuplot fit.log

Hermitian Gaussian Fit

```
*******************************************************************************

FIT:    data read from "hist_h.txt" u 1:2
        format = x:z
        #datapoints = 59
        residuals are weighted equally (unit weight)

function used for fitting: P(x)
    P(x) = a*(x**alpha) * exp(-b*(x**beta))
fitted parameters initialized with current variable values

  92 3.5068531630e-03   -4.81e-01   9.08e-04     8.980375e+00    2.347607e+00    2.399980e+00    1.439459e+00

After 92 iterations the fit converged.
final sum of squares of residuals : 0.00350685
rel. change during last iteration : -4.81358e-06

degrees of freedom    (FIT_NDF)                        : 55
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00798505
variance of residuals (reduced chisquare) = WSSR/ndf   : 6.3761e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 8.98037          +/- 1.795        (19.99%)
alpha           = 2.34761          +/- 0.1189       (5.065%)
b               = 2.39998          +/- 0.2025       (8.439%)
beta            = 1.43946          +/- 0.0672       (4.668%)

correlation matrix of the fit parameters:
                a      alpha  b      beta
a               1.000
alpha           0.990  1.000
b               1.000  0.987  1.000
beta           -0.986 -0.956 -0.988  1.000


*******************************************************************************
```

Real Diagonal Gaussian Fit

```
*******************************************************************************

FIT:    data read from "hist_d.txt" u 1:2
        format = x:z
        x range restricted to [-0.500000 : 8.00000]
        #datapoints = 40
        residuals are weighted equally (unit weight)

function used for fitting: P(x)
    P(x) = a*(x**alpha) * exp(-b*(x**beta))
fitted parameters initialized with current variable values

 478 8.2443427537e-04   -8.78e-01   1.10e-03     7.246682e+00    3.089284e-01    3.344235e+00    6.136217e-01

After 478 iterations the fit converged.
final sum of squares of residuals : 0.000824434
rel. change during last iteration : -8.78193e-06

degrees of freedom    (FIT_NDF)                        : 36
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00478549
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.2901e-05
```

```
Final set of parameters            Asymptotic Standard Error
=======================            =========================
a               = 7.24668          +/- 2.752        (37.98%)
alpha           = 0.308928         +/- 0.08676      (28.08%)
b               = 3.34424          +/- 0.3843       (11.49%)
beta            = 0.613622         +/- 0.04674      (7.617%)

correlation matrix of the fit parameters:
                a      alpha  b      beta
a               1.000
alpha           0.999  1.000
b               1.000  0.999  1.000
beta           -0.995 -0.989 -0.994  1.000


*******************************************************************************
```