Assignment 3

Quantum Information & Computing

Francesco Barone

University of Padua, Department of Physics

November 16, 2022

EXR 1 // Scaling of matrix multiplication

In Homework 1, I had already done a **bash script** to automatize the benchmarks. Nevertheless, I have rewritten the script **in Python**, as requested.

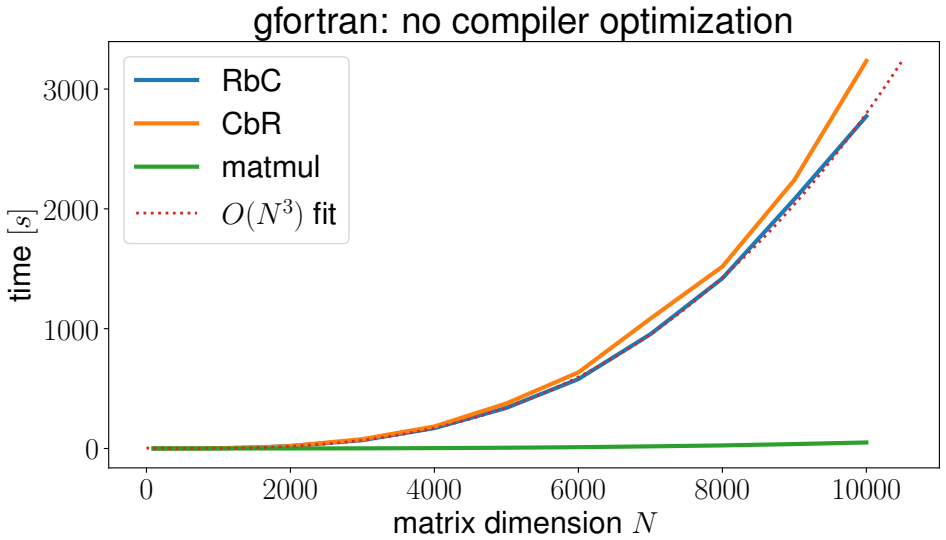
- different optimization flags $(0, 1, 2, 3, 5)$ are probed, each one printing the results on a separate file in `data/ *.bench`
- the size of the matrices N scales from 100 to 10000 with step $10^{\texttt{FLOOR}(\log_{10} N)}$, i.e.

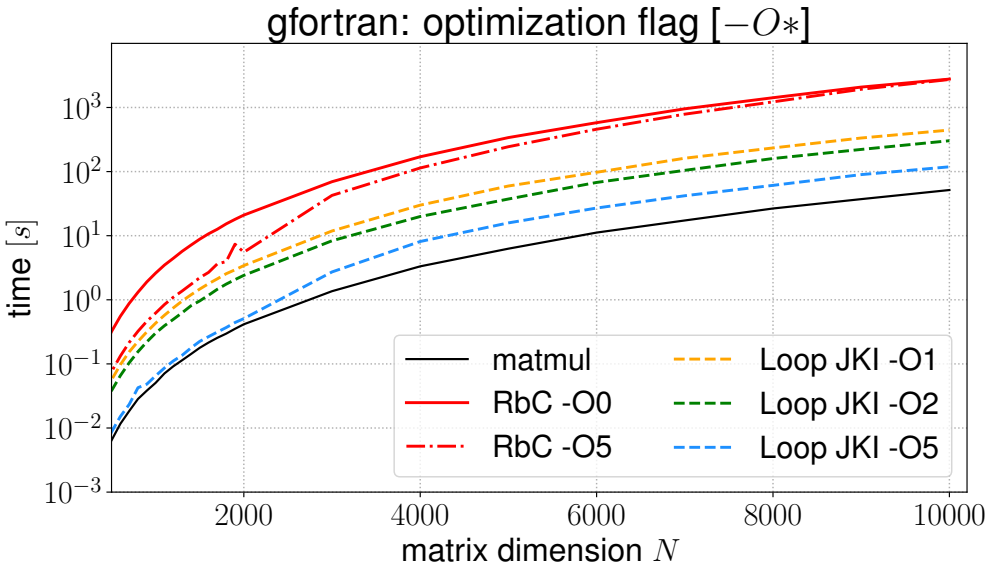
$$N = 100, 200, ..., 900, 1000, 2000, ..., 10000$$

- to improve stability, matrices with $N < 2000$ are benchmarked 3 times, taking the average CPU time

The fit with an order 3 polynomial is very good all over the domain:

$$y(N)_{RbC} \sim (3.69 \pm 0.05) \cdot N^3$$





EXR 2 // Eigenproblem

We wish to compute the eigenvalues of Hermitian matrices of order N.
To accomplish this task, **I have updated the complex matrix module** of Homework 2 (`mod_matrix_c16.f90`) with some new functions:

1) If we neglect the usual operations to initialize the matrix object itself, generating an Hermitian matrix is quite simple:

- The `complex*16` random values are drawn from a distribution (chosen by the user via `dist` arg) with LAPACK's `zlarnv()`
  - the random values are placed column-wise in the upper triangular matrix
  - the conjugate values are copied to the rows of the lower triangular matrix
- diagonal is set to be real

Remark: I know that there exists a function in LAPACK to generate directly the matrix. Nevertheless. for this exercise I wanted to use a more "standard" approach.

complex16 Hermitian

```
function InitHermitian(size, dist) result(cmx)
    integer, intent(in) :: size
    type(complex16_matrix) :: cmx
    character(1) :: dist
    complex*16, dimension(:), allocatable :: tmp

    ...

    do jj = 1,size
        ! generate random numbers to tmp array
        call zlarnv(idist, iseed, jj, tmp)
        ! copy from tmp to columns & rows (conjugate)
        cmx%val(1:jj,jj)   =        tmp( 1:jj )
        cmx%val(jj,1:jj-1) = conjg( tmp( 1:jj ) )
        ! fix diagonal to real
        cmx%val(jj,jj) = realpart(cmx%val(jj,jj))
    end do
    deallocate(tmp)
end function
```

EXR 2 // Eigenproblem

complex16 Hermitian

```
function InitDiagonalReal(size, dist) result(cmx)
    ...
    call dlarnv(idist, iseed, size, tmp)
    do jj=1,size ! placing values in diagonal
        cmx%val(jj,jj) = dcmplx(tmp(jj))
    end do
    deallocate(tmp)
end function
```

2) Since it will be required by EXR3, this function will generate diagonal matrices with real values, using LAPACK's `dlarnv()`.
The matrix is initialized to zero and the random real values are just copied to the diagonal.

3) To compute the eigenvalues, I have wrapped `zheev()` from the LAPACK library in a custom function that returns just the eigenvalues sorted in ascending order.

eigenvalues

```
type(complex16_matrix) :: A
double precision, dimension(:), allocatable :: eigv

eigv = eigenvalues(A)
```

Given the eigenvalues λ_i, we define the **eigenvalue spacing** as

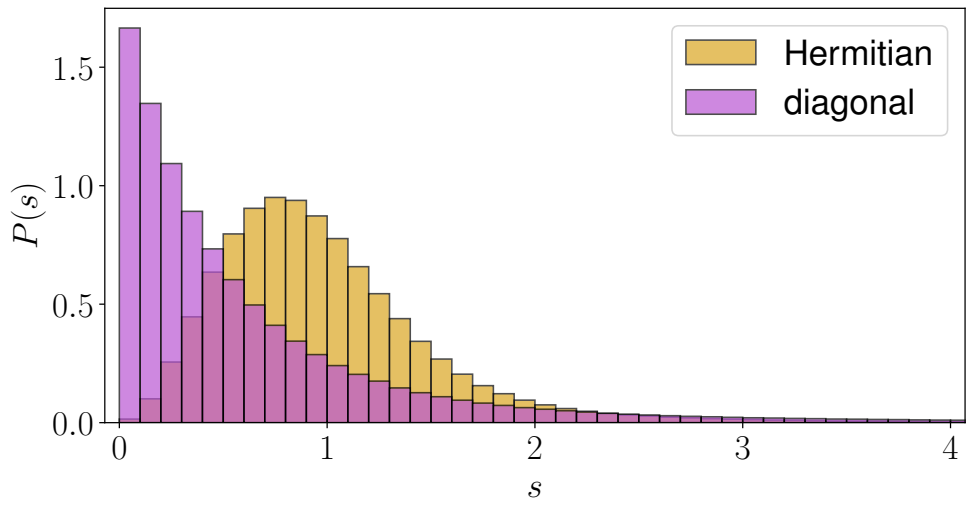
$$\Lambda_i = \lambda_i - \lambda_{i-1}$$

which is eventually normalized to $s_i = \Lambda_i / \bar{\Lambda}$ with $\bar{\Lambda} = \mathrm{avg}(\Lambda_i)$.

EXR 3 // Random matrix theory

In this exercise, we compute the eigenvalues spacing distribution $P(s)$ for Hermitian and real-Diagonal matrices.

The following results are based on eigenvalues sampled from 2000 random matrices of size $N = 1000$.



We are asked to fit the distribution with the generic function of 4 parameters

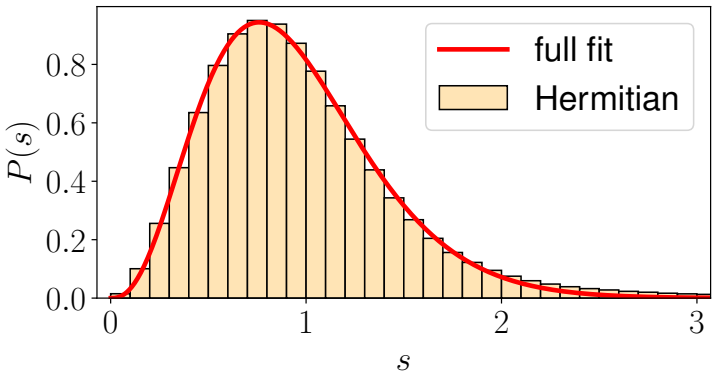
$$P(s) = as^{\alpha} \exp\left(bs^{\beta}\right)$$

Indeed, due to random matrix theory, we expect the eigenvalue spacing to follow the **Wigner surmise distribution** (exact for 2×2 hermitian matrices)

$$P(s) = \frac{32}{\pi^2} s^2 \exp\left(-4s^2/\pi\right)$$

matrix type	a	b	α	β
Hermitian	12.82 ± 1.47	-2.75 ± 0.12	2.55 ± 0.06	1.34 ± 0.03
Diagonal	3.82 ± 0.23	-2.65 ± 0.06	0.164 ± 0.014	0.687 ± 0.012

This table summarizes the parameters found from the function fit.

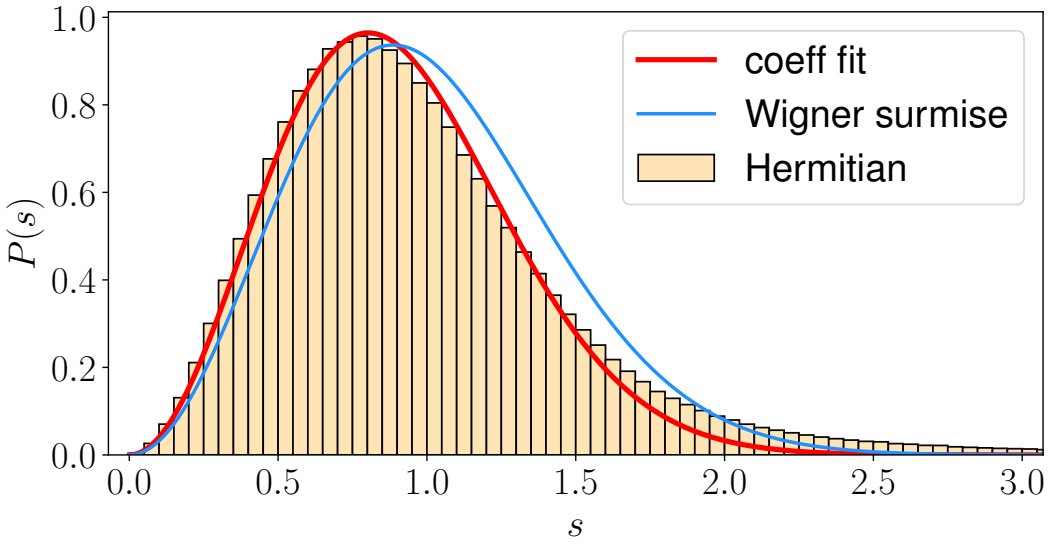


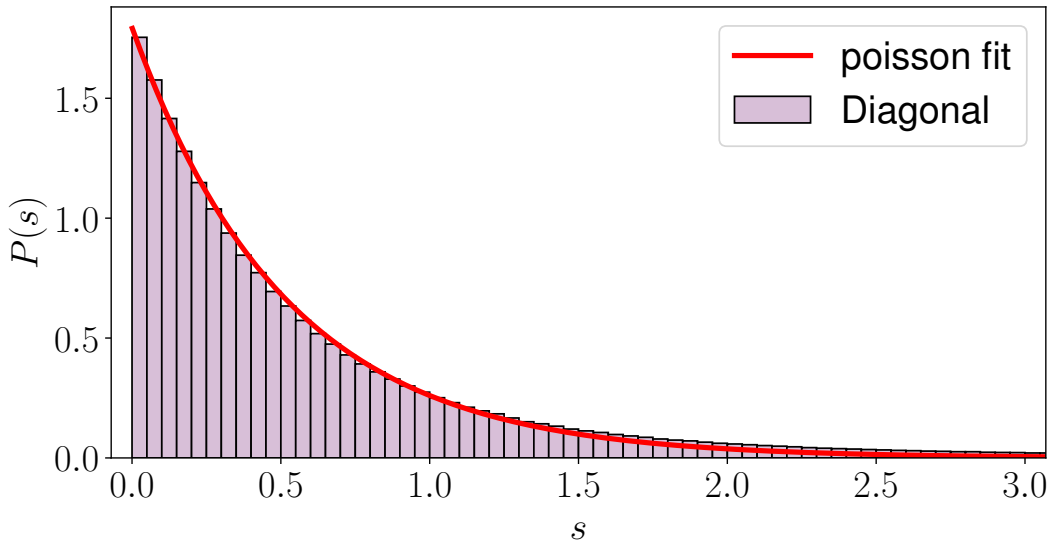
EXR 3 // Random matrix theory

The full-parametric function $P(s) = as^{\alpha} \exp\left(bs^{\beta}\right)$ seems a good fit for both the distributions.
What happens if we bind some parameters?

Reducing the free parameters to a and b brings to a more efficient optimization, as the parameter values get closer to the theoretical values of the Wigner surmise distribution.

The most appropriate fit for the diagonal matrix spacing seems to be an exponential distribution.





matrix type	a	b	α (fixed)	β (fixed)
Hermitian	4.071 ± 0.026	-1.553 ± 0.007	2	2
Diagonal	1.795 ± 0.005	-1.931 ± 0.008	0	1