Assignment 3

Quantum Information and Computing Course 2022/2023

Massimo Colombo

Department of Physics and Astronomy "Galileo Galilei"
Degree in Physics

November 21, 2022

0em

Exercise 1

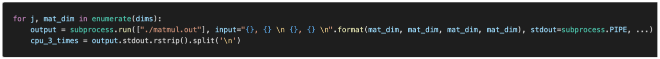
Matrix multiplication program:
$$\begin{tikzcd}\substack{\textbf{STDIN:}\\ \text{matrix dimensions}} \arrow[r] & \textbf{MATMUL} \arrow[r] & \substack{\textbf{STDOUT:}\\ \text{3 Cpu Times}}\end{tikzcd}$$

PYTHON SCRIPT:

- It creates an *array* with values evenly spaced in in the log. scale of $[N_{min}, N_{max}]$.

$$a = int(log_2(N_{min})) \quad b = int(log_2(N_{max})) \quad c = step$$
$$(a, a+c, a+2c, \ldots, b-c, b) \quad \longrightarrow \quad array = (N_{min}, 2^{a+c}, 2^{a+2c}, ..., 2^{b-c}, N_{max})$$

- *Array*'s values are used as **STDIN** in a for-loop thanks to the *subprocess* library.



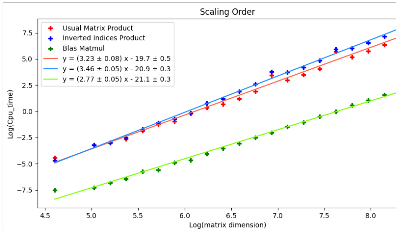
```python
for j, mat_dim in enumerate(dims):
    output = subprocess.run(["./matmul.out"], input="{}, {} \n {}, {} \n".format(mat_dim, mat_dim, mat_dim, mat_dim), stdout=subprocess.PIPE, ...)
    cpu_3_times = output.stdout.rstrip().split('\n')
```

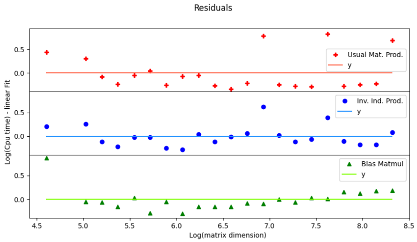
- All the **STDOUT**s are saved in 3 different files, one for each algorithm, containing matrix dimensions and relative Cpu time.

To simply utilize the python script, a logical variable "script" has been added to the Fortran program. If set to .true., the program returns only the 3 Cpu times as STDOUT.

Exercise 1, Part. 2

- The script reads those files and fits the data (for each algorithm) with a 1*st* order polynomial: the logarithm of the Cpu time vs. matrix dimension's logarithm. In this way, the angular coefficient results the complexity's order of the algorithm.



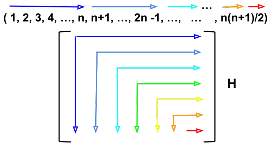


On the left, Linear fit: Logarithm of the Cpu time vs. Matrix dimension's logarithm. On the right, Residuals.

Exercise 2, Part. 1

A random hermitian matrix "H" has the following properties:

- $\frac{n(n+1)}{2}$ independent complex components.
- $H = H^\dagger \Rightarrow H_{ij} = H^*_{ji}$.



(a) Graphic allocation.

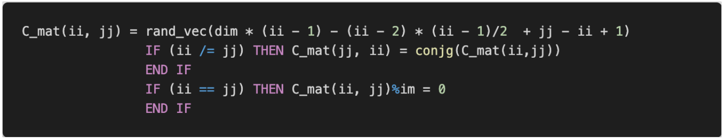
To initialize one, a subroutine "*random_hermitian*" is called:

- The subroutine receives as input a complex $n \times n$ matrix H.
- Initializes a random complex vector of dimension $\frac{n(n+1)}{2}$ through zlarnv( )[1].
- Associates the vector's components to the matrix ones as in Fig.(a).
- Returns matrix H.

[1] LAPACK's intrinsic function

Exercise 2, Part. 2

To achieve the "algorithm" in (a), the following code has been used inside 2 for-loops in the ii and jj variables, going from 1 to dim(**H**).



```
C_mat(ii, jj) = rand_vec(dim * (ii - 1) - (ii - 2) * (ii - 1)/2  + jj - ii + 1)
                IF (ii /= jj) THEN C_mat(jj, ii) = conjg(C_mat(ii,jj))
                END IF
                IF (ii == jj) THEN C_mat(ii, jj)%im = 0
                END IF
```

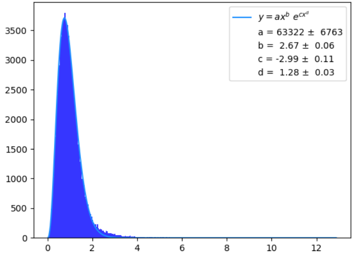
Then, LAPACK intrinsic function zheev( ) allows diagonalizing the matrix **H**, storing the eigenvalues in ascending order, leading to the calculation of normalized spacing s_i, using the given formula: $s_i = \frac{\Delta_i}{\tilde{\Delta}}$, $\Delta_i = \lambda_{i+1} - \lambda_i$, where λ_i is the i-th eigenvalue and $\tilde{\Delta}$ is the average Δ_i.
Same normalized spacings have been calculated starting from an array of random real numbers, organized in ascending order thanks to 2 nested loops.

CHECKPOINT: Comparing **H**'s Trace and the sum of the collected eigenvalues λ_i allows to check the correctness of the procedure.

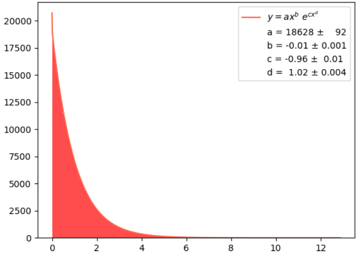
Exercise 2, Part. 3

RESULTS:

In the following, histograms concerning the distribution of the normalized spacing for the hermitian matrix and the diagonal real matrix are presented. The data have been collected iterating the procedure 40 times, running the program with matrix of dimension 1000[2].



(a) Hermitian matrix's norm. spacing distribution



(b) Real diagonal matrix's norm. spacing distr.

[2] This value has not been increased due to the complexity of the sorting algorithm for the diagonal matrix.