# Report: Week 3

Quantum Information and Computing (2021/22)
Prof. Simone Montangero

**Samuele Piccinelli**

Università degli Studi di Padova

23 November 2021

### 1) Matrix-matrix multiplication performance

1. $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$: if $C = AB$, $C \in \mathbb{R}^{n\times p}$, $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{jk}$.
2. **Complexity** is $\mathcal{O}(\mathbf{n^3})$ for $n \times n$ matrices.
3. **Nested loop** to iterate e.g. either by rows or by columns, by swapping the order of $i = 1, \ldots, n$ and $j = 1, \ldots, p$.
4. Impact on practical performance: **memory access patterns** and **cache use** of the algorithm.

We study this orders in **`by_row`** and **`by_col`** functions.

We compare their performances with the native **`MATMUL`** function.

### 2) Eigenproblem and random matrix theory

1. For a **random** $n \times n$ **Hermitian** matrix $A$, the eigenvalues $\lambda_i \in \mathbb{R}$ $(i = 1, \ldots, n)$.
2. With $\lambda_1 < \lambda_2 < \cdots < \lambda_n$, the **normalized spacing** between them is $s_i = \frac{\lambda_{i+1} - \lambda_i}{\langle \Delta\lambda \rangle}$.
3. We expect for $P(s)$ to follow the **Wigner surmise**, $P(s) = \frac{32 s^2}{\pi^2} \exp\left(\frac{4 s^2}{\pi}\right)$.

We compute a set of normalized spacing from **generic** and **diagonal** matrices.

We bin the data, fit them with $P(s) = a s^{\alpha} \exp(-b s^{\beta})$ and report the parameters.

# Code development

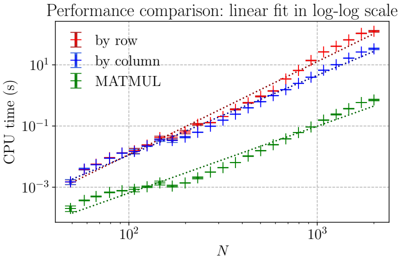
- The modules from the previous weeks are employed.
- A **Python script** is implemented to aid automatization: **`subprocess`** library.

```python
def get_output(N):
    output = sp.Popen([('./' + exe), str(N)], \
                      stdout=sp.PIPE ).communicate()[0]
    times = output.decode('utf-8').split()
    return np.array([float(time) for time in times])
```

- The function **`by_col`** is expected **by implementation** to be the faster since it access consecutive memory elements.
- We make use of the **native functions**:
  - **`GET_COMMAND_ARGUMENT`** to get the dimension of the square input matrices (`N`) as command-line argument;
  - **`CPU_TIME`** to measure the elapsed time.
- For every value of `N` the Fortran executable is called **`N_times`** times: **mean** and **standard deviation** are computed.
- Also: **pre-** and **post-conditions** are considered.

# Results

The performance is monitored for `N` spanning in [`N_min`,`N_max`]=[50, 2000] for 25 points **evenly log-spaced**.




Fit parameters for $y = ax + b$:

| method | $a \pm \sigma_a$ | $b \pm \sigma_b$ |
|---|---|---|
| `by_row` | $3.04 \pm 0.08$ | $-8.0 \pm 0.2$ |
| `by_col` | $2.61 \pm 0.05$ | $-7.2 \pm 0.1$ |
| `MATMUL` | $2.19 \pm 0.09$ | $-7.6 \pm 0.2$ |

- The results `by_row` are in agreement with the expected slope coefficient of $\sim 3$.
- `by_col` and `MATMUL` have better performances (Strassen algorithm/Winograd variation?)

# Code development

- New function **hmat_init_rand** and **hmat_init_diag** is added to the **cmat** type.
- Random matrix elements sampled from $\sim \mathcal{N}(0,1)$ (**random_stdnormal()** function).
- Eigenvalues: **Lapack**'s **ZEEHV** is employed (**hmat_eigs** as wrapper).

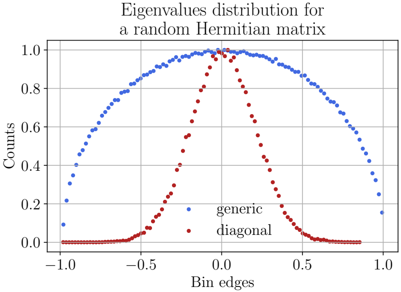
```
allocate(rwork(max(1, 3*dim-2)))

lwork = -1
call ZHEEV('N', 'U', dim, A%elem, dim, eigs, dummy, lwork, rwork, info)

lwork = max((nb+1)*dim, nint(real(dummy(1))))
allocate(work(lwork))

if (.NOT.diagonal) then
    A = .RH.dim
else
    A = .DH.dim
end if

call ZHEEV('N', 'U', dim, A%elem, dim, eigs, work, lwork, rwork, info)
```

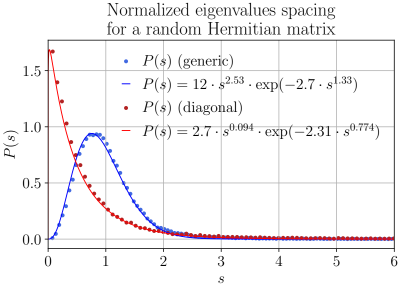
- **Compile flags**: `-llapack -Og -Wextra -fcheck=all`

# Results

**Spacings** for a $1000 \times 1000$ are sampled 50 times for both **generic** and **diagonal** random hermitian matrices.

- **Eigenvalues distribution**: Wigner semicircle, Gauss.
- **Spacing distribution**: Wigner surmise, Poisson.




| type | $\alpha \pm \sigma_\alpha$ | $\beta \pm \sigma_\beta$ | $a \pm \sigma_a$ | $b \pm \sigma_b$ |
|---|---|---|---|---|
| **generic** | $2.53 \pm 0.07$ | $1.33 \pm 0.03$ | $13 \pm 2$ | $2.7 \pm 0.1$ |
| **diagonal** | $0.094 \pm 0.004$ | $0.774 \pm 0.005$ | $2.72 \pm 0.04$ | $2.31 \pm 0.02$ |