Presentation of Assignment 3

Quantum Information and Computing, A.Y. 2022/2023

Due date: 22/11/2022

Paolo Zinesi
paolo.zinesi@studenti.unipd.it

Matrix multiplication scalings

EX3a_script.py

```python
import os
import math as m

# dimension grid parameters
N_min, N_max = 100, 12000
mult_factor = m.sqrt(2.0)

# number of repetitions (for statistics)
N_rep = 3

# list of dimensions to try
dim_list = [int(N_min * mult_factor**exp) for exp in range(m.floor(m.log(N_max/N_min, mult_factor))+1)]

# list of different methods
method_list = ['naive', 'opt', 'builtin']

# run for different times the desired program (a.out)
for dim_ in dim_list:
    for method_ in method_list:
        for rep_ in range(N_rep):
            os.system(f'./a.out {dim_} {dim_} {dim_} {dim_} {method_}')
```

Execution cycle

EX3a_Zinesi_CODE.f90

```fortran
        IF (TRIM(matmul_method) .EQ. "naive") THEN
            ! naive function testing
            CALL CPU_TIME(ti)
                CALL my_MatMul_naive(AM,BM,CM)
            CALL CPU_TIME(tf)


            ! print performances string
            WRITE (str, "(I5,'; ',I5,'; ',I5,'; ',I5,'; ',ES15.8E2)") &
                SIZE(AM,1), SIZE(AM,2), SIZE(BM,1), SIZE(BM,2), tf-ti

            ! write data into a dedicated file
            OPEN(unit=10, file='results/'//TRIM(matmul_method)//'_performances.dat', access='APPEND')
                WRITE(10, '(A)') TRIM(str)
            CLOSE(unit=10)
```

```
paolozinesi@MBP-di-Paolo Assignment3/EX3a » ./a.out 200 200 200 200 opt
paolozinesi@MBP-di-Paolo Assignment3/EX3a » ./a.out 300 300 300 300 builtin
```

The Python script "**EX3a_script.py**" executes for multiple times the code compiled from "**EX3a_Zinesi_CODE.f90**". The result of each execution is appended to an output file based on the matrix multiplication method. This approach allows to freely choose the number of repetitions in the Python script without recompiling the source code.

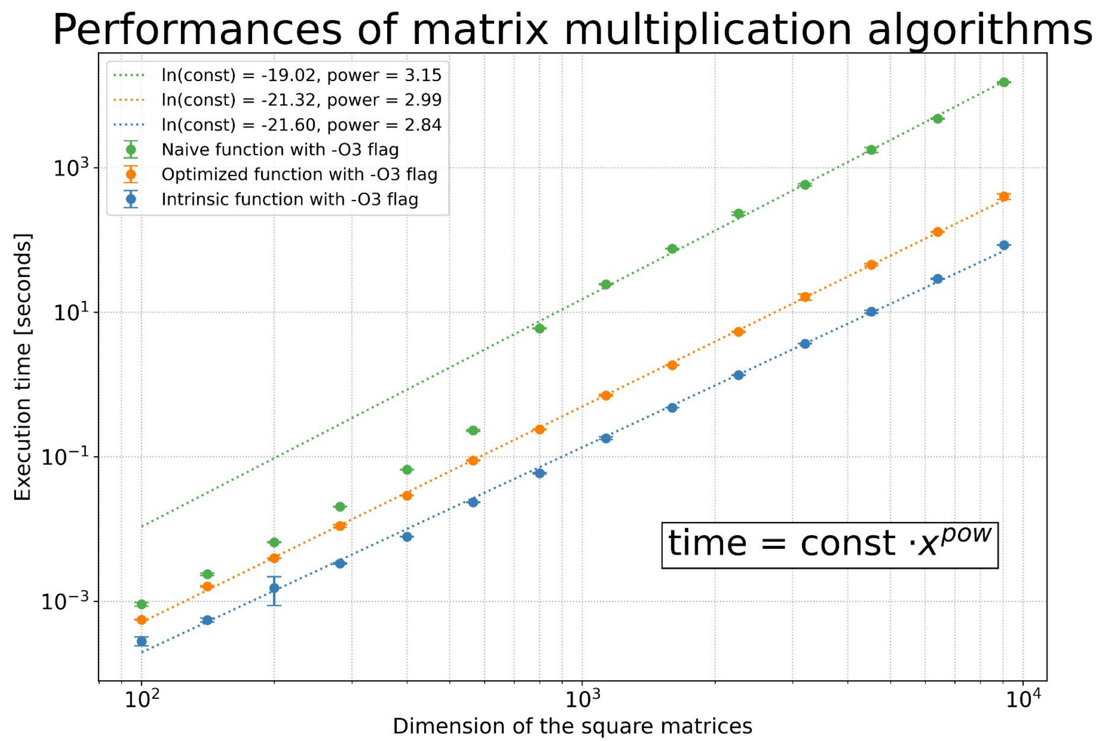
Matrix multiplication scalings

UNIVERSITÀ DEGLI STUDI DI PADOVA

The Python script "**EX3a_plots.py**" plots and fits the execution times obtained previously.

Only sizes ≥ 800 for the naive method have been considered in the linear fit.

The optimization performed by the compiler causes the observed difference in the naive method scalings for small and large matrices, even if the two regimes only differ by a multiplicative constant.



EX3a_plots.py

```python
# fit
if type_=='naive':
    # restrict domain
    popt, pcov = curve_fit(f=pow_func, xdata=np.log(plot_df.loc[plot_df['size']>=800, 'size']),
                           ydata=np.log(plot_df.loc[plot_df['size']>=800, 'time_mean']))
else:
    popt, pcov = curve_fit(f=pow_func, xdata=np.log(plot_df['size']), ydata=np.log(plot_df['time_mean']))
ln_const, power = popt

# plot
x0 = np.linspace(min(plot_df['size']), max(plot_df['size']), 1000).reshape(-1,1)
ax.plot(x0, np.exp(pow_func(np.log(x0), ln_const=ln_const, pow=power)), ls='dotted', c=col_dict[type_],
        label=f"ln(const) = {ln_const:.2f}, power = {power:.2f}")
```

Improvements to DCmatrix

DCmatrix_mod.f90

```fortran
! This subroutine stores into the variable 'M_rand' a random DCmatrix following a given input distribution.
! The LAPACK library ZLATMR is employed to generate such random matrix.
! Only double complex square matrices can be generated with this function.
!
! inputs:
! - M_rand [DCmatrix]: DCmatrix to fill with random values
```

```fortran
SUBROUTINE RandMat(M_rand, dist, iseed, sym)
```

```fortran
    ! filling of DCmatrix using LAPACK routine ZLATMR
    CALL ZLATMR(size, size, dist, iseed, sym, diag, 6, 1.D0, 1.D0, 'F', &
                'N', DL, 0, 1.D0, DR, 0, 1.D0, 'N', ipivot, size, size, 0.D0, -1.0D0, 'N', &
                M_rand%elem, size, iwork, info)
```

```fortran
SUBROUTINE eigenvaluesMat(M, eigenvals)
```

```fortran
    ! call eigenvalues subroutine
    CALL ZHEEV('N', 'U', size, M%elem, size, eigenvals, work, lwork, rwork, info)
```

Two subroutines has been added to DCmatrix module to wrap and simplify calls to **ZLATMR** and **ZHEEV** LAPACK subroutines. In particular, the eigenvalue subroutine for Hermitian matrices might be reused in future code. “**EX3b_Zinesi_CODE.f90**” tests these new functionalities.

```
paolozinesi@MBP-di-Paolo Assignment3/EX3b » ./a.out 10
Normalized spacings printed on 'results/norm_spac.dat'
```

Eigenvalues spacings

UNIVERSITÀ DEGLI STUDI DI PADOVA

$$\{\lambda_i\} : \text{Eigenvalues of a random matrix in increasing order}$$

$$\Lambda_i = \lambda_{i+1} - \lambda_i : \text{Eigenvalue spacings}$$

$$s_i = \frac{\Lambda_i}{\langle \Lambda_i \rangle} : \text{Normalized eigenvalue spacings}$$

“**EX3c_Zinesi_CODE.f90**” uses the DCmatrix routines “**RandMat**” and “**eigenvaluesMat**” to compute the normalized spacings distribution for random Hermitian matrices. The normalized spacings distribution for random real diagonal matrices are computed, instead, by using directly **DLARN** and **DLASRT** LAPACK subroutines. Results are appended to a dedicated output file.

The real and imaginary parts of the complex Hermitian matrix entries are chosen to be **uniform in [-1,1]**. Similarly, the elements of the diagonal real random matrix are uniformly distributed in [-1,1]. Since the mean value of these random entries is zero, there is no need to neglect the largest eigenvalue.

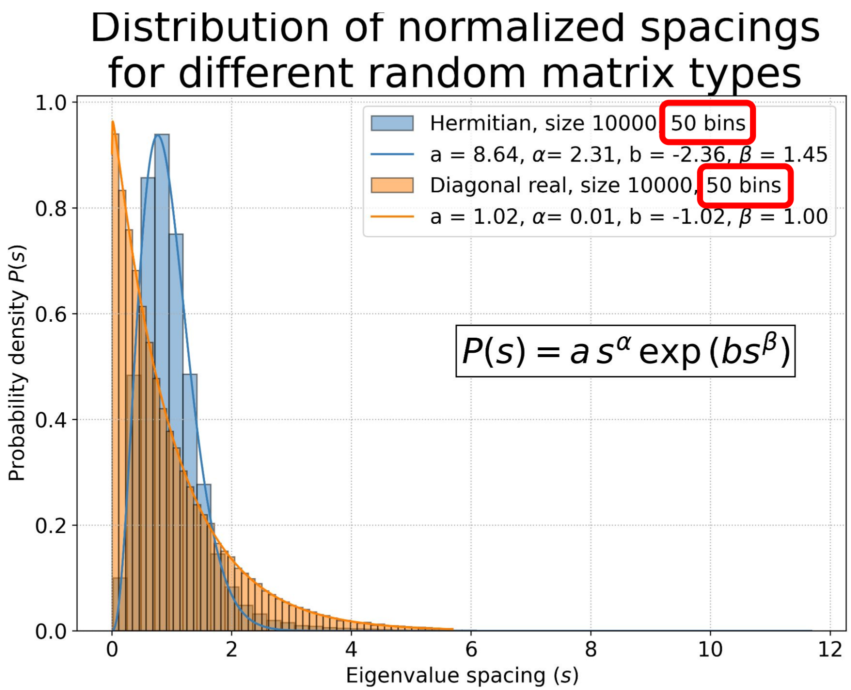
```
paolozinesi@MBP-di-Paolo Assignment3/EX3c » ./a.out 2000 HS
paolozinesi@MBP-di-Paolo Assignment3/EX3c » ./a.out 2000 DS
```

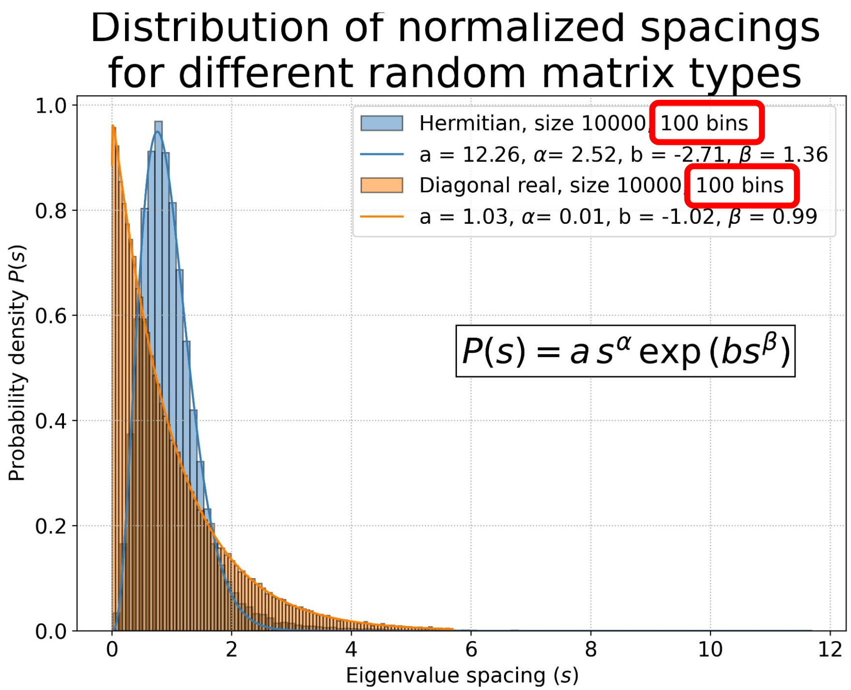
Hermitian

Diagonal real

Eigenvalues spacings

UNIVERSITÀ DEGLI STUDI DI PADOVA





The normalized spacings distribution is plotted and fitted with the Python script “**EX3c_plots.py**”. Different bin choices lead to different optimal fit parameters in the Hermitian case.

100 bins	**a**	**α**	**b**	**β**
Hermitian	12 ± 2	2.52 ± 0.09	-2.71 ± 0.17	1.36 ± 0.05
Real diag.	1.03 ± 0.02	0.013 ± 0.006	-1.02 ± 0.02	0.99 ± 0.01