Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

November 22, 2021

Theory

- **Exercise 1**. Fit of the scaling of matrix-matrix multiplication CPU Time vs N

  $$\forall N \in [N_{min}, N_{min}+20, \dots, N_{max}] \Rightarrow \sum_{kk=1}^{N} A_{ii,kk} B_{kk,jj}$$

  where $N_{min} = 20$ and $N_{max} = 2020$ with 3 methods:

  - `matmul_byrow`: *ii* (row of mat_A) ,*kk* (col of mat_B), *jj* (col of mat_A)
  - `matmul_bycol`: *jj* (**col** of mat_A) ,*kk* (**col** of mat_B), *ii* (row of mat_A)
  - `MATMUL`: FORTRAN intrinsic subroutine (the fastest!)

  Since we use 3 for loops on N values we **expect a scaling** of $\mathcal{O}(N^3)$

- **Exercise 2**. The **spacing between eigenvalues** stored in **crescent order** *s* of a **generic** hermitian matrix (both of size $N = 2500$ in our case) follows the *Wigner surmise* distribution:

  $$P(s) = as^{\alpha} e^{-bs^{\beta}} \approx \frac{32s^2}{\pi^2} e^{-\frac{4s^2}{\pi}} \qquad \text{Exact for } N = 2. \tag{1}$$

Code development (Ex.1)

- Automatized the launch of the program `w3_ex1.f90` via a Python script `script.py` and saved the results in a `.txt` file for the three methods.

```
!part of FORTRAN code to read from command line
CHARACTER(10) :: command_line_arg
CALL GET_COMMAND_ARGUMENT(1, command_line_arg)
READ(command_line_arg, FMT="(I8)") rows_mat_A
```

```
#Python code to read from output of FORTRAN code
sp.run(['gfortran', src , 'my_mat_mul.f90','error_handling.f90','-o', exe])
for nn, N in enumerate(N_vector):
    output = sp.Popen( [('./' + exe), str(N)], stdout=sp.PIPE ).communicate()[0]
    times = output.decode('utf-8').split()
```

- Fitted a polynomial function and determined the scaling of CPU time vs size of the matrices (using `np.polyfit` from `numpy` and `mean_squared_error` from `sklearn.metrics`).

Code development (Ex.2)

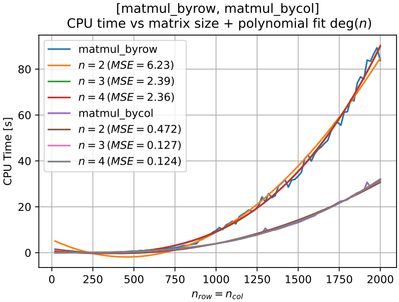
- Expanded the module dc_matrix defined for Assignment 2 including: `hermitian_init`, `diagonal_init`, `eigenvalues`, `norm_eigenvalues_spacing`.

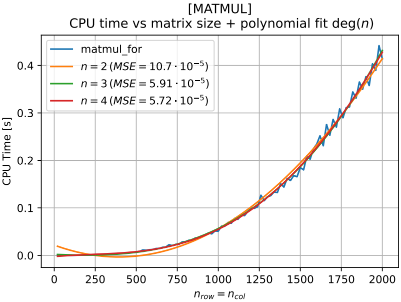
```fortran
FUNCTION hermitian_init(input_dims) RESULT(matrix_out)
    ! [...]
    DO jj = 1, input_dims(1)
        DO ii = 1, input_dims(2)
            IF (ii.EQ.jj) THEN
                matrix_out%m_el(ii, jj) = COMPLEX(-1+2*RAND(),0d0)
            ELSE IF (jj.GT.ii) THEN
                matrix_out%m_el(ii, jj) = COMPLEX(-1+2*RAND(),-1+2*RAND())
                matrix_out%m_el(jj, ii) = CONJG(matrix_out%m_el(ii, jj))
            ELSE
```

```fortran
FUNCTION eigenvalues(matrix_in) RESULT(vector_output)
    ! [...]
    IF (matrix_in%m_dims(1).EQ.matrix_in%m_dims(2)) THEN
        N = matrix_in%m_dims(1)
        ALLOCATE(vector_output(N))
        !ZHEEV computes the eigenvalues and eigenvectors for HE matrices
        CALL ZHEEV('N','U', N, matrix_in%m_el, N, vector_output, work, ...)
        !DLASRT sorts numbers in increasing or decreasing order
        CALL DLASRT('I', N, vector_output, INFO)
    ! [...]
```

CPU Time vs matrix size for three methods (Ex.1)

We report in the following graphs the **results** obtained with the **three methods**, considering the **polynomial** function (order n) for the fit: $y = \sum_{i=1}^{n} a_i x^i + a_0$



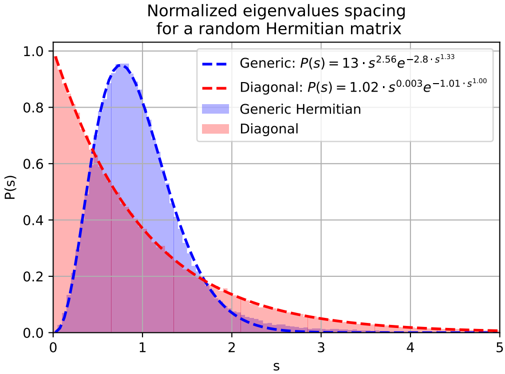


$o(4)$ polynomial fit	a_3	a_4
`matmul_bycol`	6.4e-09	−6.2e-13
`matmul_byrow`	3.6e-09	2.3e-12
`MATMUL`	1.2e-10	−1.8e-14

Table: a_3 and a_4 for the $n = 4$ polynomial fit.

The MSE for $n = 4$ is smaller than the one for $n = 3$ but the coefficients for the fits a_4 and a_3 are more than 3 order of magnitude apart in all cases. For **Occam's razor** we can state that the divergence is $\mathcal{O}(N^3)$.

Normalized eigenvalues spacing (Ex.2)



Sampling is done by running 80 times the code for a $N = 2500$ matrix. The eigenvalues λ_i are stored in **crescent order** and their **normalized spacing** is:

$$s_i = \frac{\lambda_{i+1} - \lambda_i}{\overline{\Delta\lambda}} \tag{2}$$

We use the same fitting function in Eq. 1 for both the Hermitian and the diagonal case. The parameters for the former distribution **differ** from the theoretical ones, but we are able to **distinguish** the two curves.

	a	b	α	β
Generic	13 ± 2	2.8 ± 0.2	2.56 ± 0.09	1.33 ± 0.04
Diagonal	1.02 ± 0.02	1.01 ± 0.02	0.003 ± 0.005	1.00 ± 0.01

Table: Results for the fits on 100 points.

Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

November 29, 2021

Time independent Schrödinger equation

The Hamiltonian operator of the **quantum harmonic oscillator** in one dimension is

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2} m\omega^2 \hat{q}^2 \underset{m=\omega=1}{=} \frac{\hat{p}^2}{2} + \frac{\hat{q}^2}{2} \tag{1}$$

where m is the **mass** of the particle, and $\omega = \sqrt{k/m}$ the **angular frequency**. We work in **coordinate representation** $\psi_n(x) = \langle x|\psi_n\rangle$ and **natural units** $\hbar$=1. We want to solve the time-independent Schrödinger equation:

$$\hat{H}\,|\psi_n\rangle = E_n\,|\psi_n\rangle \rightarrow \frac{1}{2}\left(-\frac{d^2}{dx^2} + x^2\right)\psi_n(x) = E_n\psi_n(x) \tag{2}$$

- A class of solutions for the **eigenvectors** is that of Hermite functions:

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{m\omega x^2}{2\hbar}} H_n\left(\frac{m\omega}{\hbar}x\right) \quad n = 0, 1, 2, \ldots \tag{3}$$

- The energy levels corresponding to each $\psi_n(x)$, are specified by the **eigenvalues**:

$$E_n = \omega\left(n + \frac{1}{2}\right) \tag{4}$$

Code development

1. We discretized a space interval $[L_{min}, L_{max}]$, into N intervals of size $\Delta x = L/N$. We call $x_i = L_{min} + (ii - 1)\Delta x$ for $ii \in [1, N+1]$ the i-th point of the discretized grid.
2. We use the **finite difference method** for the second derivative of the k-th eigenfunction:

$$\frac{\partial^2 \psi_k(x)}{\partial x^2} = \frac{\psi_{k+1} - 2\psi_k + \psi_{k-1}}{\Delta x^2} + \mathcal{O}(\Delta x^4) \tag{5}$$

In this way we will end up with a **tridiagonal**, **real** and **symmetric** hamiltonian H^d

$$H^d_{ij} = H^d_{ji} = -\frac{\hbar^2}{2\Delta x^2} \quad H^d_{ii} = \frac{\hbar^2}{\Delta x^2} + \frac{1}{2} m\omega^2 {x_i}^2 \tag{6}$$

3. We **initialize** the hamiltonian operator $H^d_{ii,jj}$ for $ii, jj \in [1, N+1]$

```
!filling the matrix
        IF(ii.EQ.jj) THEN
            ham_op(ii,jj) = (h_bar**2)/(dx**2)
        ELSE IF(ii.EQ.jj+1) THEN
            ham_op(ii,jj) = -(h_bar**2)/(2*dx**2)
            ham_op(jj,ii) = ham_op(ii,jj)
!adding the lattice-position dependent value
DO ii = 1, N+1
    ham_op(ii,ii) = ham_op(ii,ii) + 0.5*m*omega**2*((Lmin+(ii-1)*dx)**2)
```

Code development

- The subroutine we exploit to diagonalize the matrix and find its eigenvalues λ_n and eigenfunctions is `DSTEV`. We implement it in the `diagonalize_H_op(ham_op, eigs)` subroutine inside the `qm_utils.f90` module.

```fortran
    !compute eigenvalues and eigenvectors
    CALL DSTEV('V', N, diag_A, subd_A, ham_op, N, work, info)
```

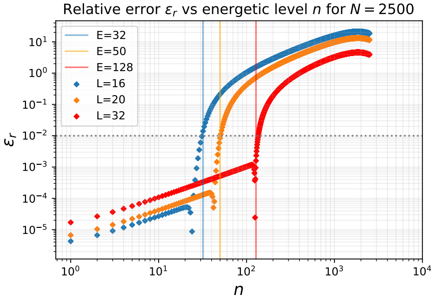
- We can run the code and specify simulation parameters:
  `./w4.out -custom 2500 -10 10 2500`
- We **format** output file name to increase code flexibility.

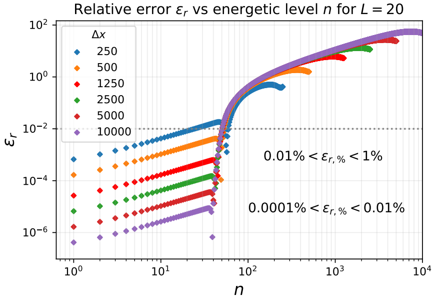
```fortran
CHARACTER(len=1024) :: filename
![...]
WRITE(filename, "(ai4af4.1af4.1a)") "data/eigenvalues_N", N,  "_Lmax", ...
```

Eigenvalues

We analyze the **relative error** between the numerical λ_n and theoretical eigenvalue λ_{th}

$$\varepsilon_r = \frac{\lambda_n - \lambda_{th}}{\lambda_{th}} \qquad \varepsilon_{r,\%} = 100 \cdot \varepsilon_r \tag{7}$$

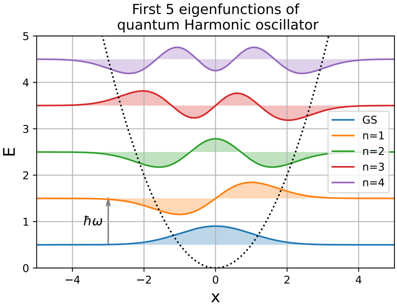

Relative error ε_r vs energetic level n for $N = 2500$


Relative error ε_r vs energetic level n for $L = 20$

For a fixed number of discretization points $N = 2500$, we observe how by **increasing the width** of the space range we can describe systems with **higher energy**. We find that: $\lambda_{n,max} = L_{max}^2 \cdot 0.5$. For $\lambda > \lambda_{n,max}$ the problem becomes the *particle in a box*.

Once the **space range has been set** between $L_{min} = -10$ and $L_{max} = +10$, we can observe that as the number of discretization points increases (different colors), greater precision is achieved. For a fixed n: $\varepsilon_r \propto \mathcal{O}(\Delta x^2)$.

Considerations on the code



We report here the first 5 numerical eigenfunctions, with a spacing equal to $\hbar\omega = 1$ for the choice of the two parameters. The energetic levels E are the one computed in the simulation.

- **Correctness**. We have included some pre and post conditions. A special mention should be made of the one placed after the DSTEV subroutine, careful about the INFO variable ($\neq 0$ if errors are present).
- **Stability**. Variations in the input could result in inaccurate and nonphysical values, but not in code crashes (at least, up to now).
- **Accurate Discretization**. We can in principle achieve arbitrary accuracy up to the $k-th$ eigenvalue.
- **Flexibility**. The user can run the code with -custom flag and specify all the simulation parameters (m, ω, and the lattice quantities).
- **Efficiency**. The critical point of the simulation lies in the diagonalization of the matrix. By using the state-of-the-art DSTEV subroutine, we are able to calculate the eigenvalues and eigenfunctions of our system with good accuracy and in acceptable times. We didn't compute the algorithmic complexity.

Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

December 6, 2021

Theory

We study the solution of a time-dependent Schrödinger 1D equation that obeys the following Hamiltonian

$$H = \frac{\hat{p}^2}{2m} + \frac{1}{2} m\omega^2 (\hat{q} - q_0(t))^2 \tag{1}$$

where $q_0(t) = t/T$ is a **time dependent shift** ($t \in (0, T]$). Making use of the work done last week, we solve for the ground state and we let him evolve exploiting the **split operator method**. Starting from the application of the time evolution operator

$$|\psi(x, t + \Delta t)\rangle = e^{-i\Delta t \hat{H}} |\psi(t)\rangle \overset{(a)}{\approx} e^{-\frac{1}{2} i\Delta t \hat{V}} e^{i\Delta t \hat{T}} e^{-\frac{1}{2} i\Delta t \hat{V}} |\psi(x, t)\rangle \tag{2}$$

we use in (a) the Baker-Campbell-Hausdorff formula. Noticing that $\hat{T}$ is diagonal in the **momentum basis** and $\hat{V}$ is diagonal in the **position basis**, we perform proper Fourier transformations of the wave function.

$$|\psi(x, t + \Delta t)\rangle = e^{-i\Delta t \hat{H}} |\psi(t)\rangle \approx e^{-\frac{1}{2} i\Delta t \hat{V}} \mathscr{F}^{-1} e^{i\Delta t \hat{T}} \mathscr{F} e^{-\frac{1}{2} i\Delta t \hat{V}} |\psi(x, t)\rangle \tag{3}$$

After the application of each Fourier transform we normalize the wave functions. The last equation propagates the state for `dt`, so we apply it **iteratively** to reach the T_{max} value.

Code development - Two main functions

- Initialization of the **kinetic part** of the hamiltonian. Given a discretization with width $L = L_{max} - Lmin$, we populate the momentum vector (after reading `FFTW3`'s manual) with **positive frequencies** $\left[0, \frac{2\pi}{L} \cdot aa\right]$ in the first half $aa \in [0, N/2]$, and with **negative frequencies** $\left[-\frac{2\pi}{L} \cdot (aa - N - 1), 0\right]$ in the second half $aa \in [N/2+1, N+1]$:

```fortran
    FUNCTION H_kinetic_part(N, Lmin, Lmax, m) RESULT(K)
    ![...]
    DO aa=1,INT(N/2)
        K(aa) = (0.5d0/m) * (4.0d0*ASIN(1.0d0)*(aa)/(Lmax-Lmin))**2
    ENDDO
    DO aa=INT(N/2)+1,N+1
        K(aa) = (0.5d0/m) * (4.0d0*ASIN(1.0d0)*(aa-N-1)/(Lmax-Lmin))**2
    ENDDO
    ![...]
```

- Initialization of the **potential part** of the hamiltonian.

```fortran
    FUNCTION H_potential_part(N, Lmin, Lmax, m, omega, time, T) RESULT(V)
    ![...]
    dx = (Lmax-Lmin)/N
    DO aa = 1, N+1
        V(aa) = 0.5*m*omega**2*(Lmin+(aa-1)*dx-time/T)**2
    ENDDO
    ![...]
```

Code development - Two main functions

- The *delicate* `myFFTW(input)` function (`myAFFTW(input)` for the inverse is similar):

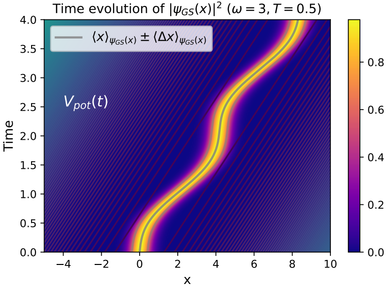
```
    INTEGER(8)                                   :: plan
    INTEGER                                      :: FFTW_FORWARD
    PARAMETER(FFTW_FORWARD=-1)
    INTEGER                                      :: FFTW_MEASURE
    PARAMETER(FFTW_MEASURE=64)
    !initialize the array once the plan is created
    temp = input
    CALL dfftw_plan_dft_1d( plan, N, input, output, FFTW_FORWARD, FFTW_MEASURE)
    CALL dfftw_execute_dft( plan, temp, output)
    output = output/SQRT(1.0d0*N)
    CALL dfftw_destroy_plan(plan)
    ![...]
```

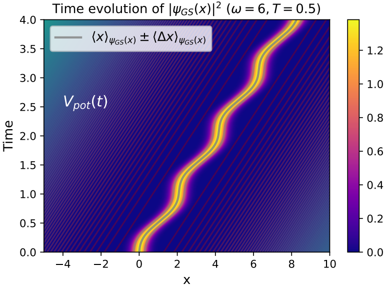
- The function `evolve_dt(wf, dt, Lmin, Lmax, m, omega, time, T)`:

```
    DO aa = 1, N+1
        wf(aa) = EXP(COMPLEX(0.0d0, - 1.0d0 / 2.0d0 * dt * V(aa))) * wf(aa)
    END DO
    wf = myFFTW(wf)
    DO aa = 1, N+1
        wf(aa) = EXP(COMPLEX(0.0d0, -1.0d0 * dt * K(aa))) * wf(aa)
    END DO
    wf = myAFFTW(wf)
    ![...]
```

Time evolution

We report in the following graphs the results obtained by computing the **probability density function** $|\psi(x)|^2$ at each time step and at each point of the discretized space lattice. For this simulations we set the time of the potential equal to $T = 0.5$ and $m = 1$.



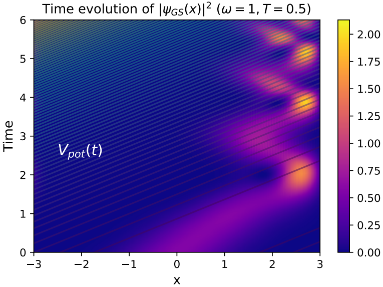


The harmonic potential moves to the right at a **constant speed**, and the position expectation value $\langle x \rangle_{\psi_{GS}(x)}$ does the same but with an **oscillatory component**.

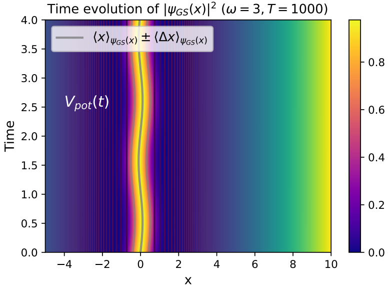
The oscillatory behavior is accentuated as the **angular frequency increases**. We did not investigate further but, at first sight, the fluctuation seems decreased in this case.

Two special cases

We now discuss two special cases achievable with our simulation: where we **extend the time window** (left graph) and where we recover the case in which the harmonic potential is **not practically moving** in a small time window (right graph).



We set $T_{max} = 6$. It is interesting to notice that the wave function hits the right border, goes in the opposite direction and the potential forces it to invert again the direction. The wave function **bounces off the right-hand side** of the interval.



We try to emulate what happens when $T = 1000$. In this case, in a $T_{max} = 4$ time window we are still seeing the ground state **oscillating at the bottom of the potential** (contrary to expectations). This may be due to some errors inserted in the code, we did not investigate further.

Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

December 12, 2021

Theory - Quantum states

We consider a **quantum system** made of N subsystem, each of them represented by its own wave function $|\psi_i\rangle \in \mathcal{H}^D$. We call D **local** and N **system** dimensions respectively. By rewriting each wave function as a linear combination of basis vectors $|\alpha_j\rangle \quad j = 1, \ldots, D-1$, we can define a **generic state** of the system $|\psi_G\rangle \in \mathcal{H}^{D^N}$ as

$$|\psi_G\rangle = \sum_{\alpha_1,\ldots,\alpha_N} C_{\alpha_1,\ldots,\alpha_N} |\alpha_1, \ldots, \alpha_N\rangle \tag{1}$$

and a **separable state** with a simple form

$$|\psi_S\rangle = \bigotimes_{i=1}^{N} |\psi_i\rangle = \bigotimes_{i=1}^{N} \left(\sum_{\alpha_i} C_{\alpha_i} |\alpha_i\rangle \right) \tag{2}$$

The generic state seeks for D^N complex coefficients, while the separable only for $N \cdot D$ of them. For what concerns the code implementation, the **functions** `sep_wf_init` and `gen_wf_init` in the `qm_utils.f90` module are similar (only the dimension of the output state changes) and they **initialize** (and **normalize**) a state with `DOUBLE COMPLEX` in $[-1, 1]$. We are not going to report them in the slides.

Theory - Density matrix

Given a generic state, we also define its $D^N \times D^N$ **density matrix** as

$$\rho = \left|\psi_G\right\rangle \left\langle\psi_G\right| = \sum_{\alpha_1,\ldots,\alpha_N} C_{\alpha_1,\ldots,\alpha_N} C^*_{\alpha_1,\ldots,\alpha_N} \left|\alpha_1,\ldots,\alpha_N\right\rangle \left\langle\alpha_1,\ldots,\alpha_N\right| \tag{3}$$

We then consider a system where $N = 2$ with local dimension D, where the generic state of the composite system is $\left|\psi_{12}\right\rangle \in \mathcal{H}^{D^2}$. We can compute the **reduce density matrices** for each one of the two subsystems by starting from $\rho_{12} = \left|\psi_{12}\right\rangle \left\langle\psi_{12}\right|$ and **tracing out** the system we are not interested in

$$\rho_1 = Tr_2[\rho_{12}] = \sum_{j=1}^{D} \left\langle j\right|_2 \rho_{12} \left|j\right\rangle_2 \quad \rho_2 = Tr_1[\rho_{12}] = \sum_{j=1}^{D} \left\langle j\right|_1 \rho_{12} \left|j\right\rangle_1 \tag{4}$$

where we index by j the different basis elements. The reduced matrices have dimension $D^{N-1} \times D^{N-1}$. For this simplified situation we graphically show how to compute them for $D = 2$. The colors mean that for example, $a_1 = a + f$, or $d_2 = f + p$:

$$\rho_1 = \begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \Longleftrightarrow \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \qquad \rho_2 = \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix} \Longleftrightarrow \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

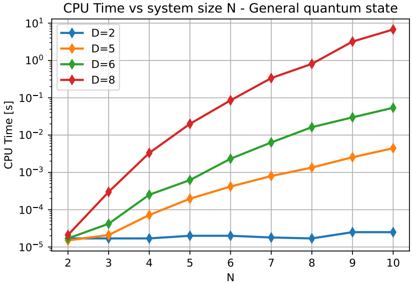
Partial trace

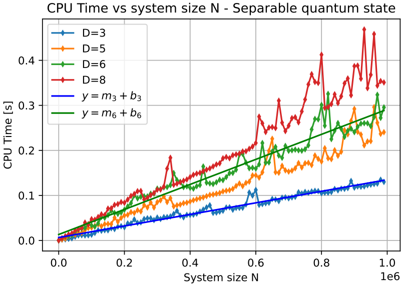
- We report only the function for the partial trace computation, since it is the most important one.

```fortran
FUNCTION p_trace(local_dimension, dmatrix, i_subsystem) RESULT(red_dmatrix)
![...]
dN = SIZE(dmatrix,1)
d = local_dimension
![...]
IF (i_subsystem == 2) THEN
    DO aa=1, d
        row(aa) = 2*aa-1 !useful vector of indices
        col(aa) = 2*aa-1 !useful vector of indices
    ENDDO
    DO aa = 1, SIZE(row)
        DO bb = 1, SIZE(col)
            DO cc = 1, d
                !this computes the trace of each minor
                red_dmatrix(aa, bb) = red_dmatrix(aa, bb) + dmatrix(row(aa)+(cc-1), row(bb)+(cc-1))
    ELSE IF(i_subsystem == 1) THEN
        DO aa = 1, dN/d
            DO bb = 1, dN/d
                DO cc = 1, d
                    red_dmatrix(aa, bb) = red_dmatrix(aa, bb) + dmatrix(aa+(cc-1)*d, bb+(cc-1)*d)
    ELSE
        call checkpoint(.TRUE., 'The function is not built for more than two subsystems', 'e')
ENDIF
```

Results - CPU_TIME

For matters of convenience, we make use of a **Python script** named `script.py` to launch the Fortran code and save the results of the CPU_TIME required for the initialization of the states with different N and D.





The initialization requires D^N coefficients: the graph in fact shows an **exponential behavior** for the CPU_TIME. Already for $N = 10$, the time required is quite a lot, so it is **unfeasible** to initialize huge quantum system in this machine.

In the separable case things are different. The initialization time scales linearly, $m_6/m_3 \approx 2$. The results are also unstable and this may be due to the fact that the times involved are very small and can be influenced by the **instantaneous performance** of the laptop.

Results - Partial trace

We now show some results for the implementation of the partial trace operation in the case of a $N = 2$ system with local dimension $D = 2$. To obtain the partial trace one must **run the code** with `./w6.out -custom 2 2 -trace 2`. With the last line we initialize a density matrix for **two spins one half** ($N = 2$ the first, $D = 2$ the second) and we trace out (`-trace`) the subsystem 2.

$$\rho_{12} = \begin{pmatrix} 0.252025, -i0.000000 & 0.236731, +i0.043350 & 0.274330, +i0.142117 & 0.076515, +i0.171109 \\ 0.236731, -i0.043350 & 0.229821, -i0.000000 & 0.282127, +i0.086306 & 0.101304, +i0.147564 \\ 0.274330, -0.142117 & 0.282127, -i0.086306 & 0.378749, -i0.000000 & 0.179776, +i0.143106 \\ 0.076515, -i0.171109 & 0.101304, -i0.147564 & 0.179776, -i0.143106 & 0.139403, +i0.000000 \end{pmatrix}$$

The two **reduced density matrices** obtained with the partial trace are the following:

$$\rho_1 = \begin{pmatrix} 0.481846, -i0.000000 & 0.375634, +i0.289682 \\ 0.375634, -i0.289682 & 0.518153, +i0.000000 \end{pmatrix}$$

$$\rho_2 = \begin{pmatrix} 0.630774, -i0.000000 & 0.416507, +i0.186456 \\ 0.416507, -i0.186456 & 0.369225, -i0.000000 \end{pmatrix}$$

If we trace with respect to a system different from 1 or 2, the code **raises an error**. **Further checks** can be implemented, exploiting the properties of a density matrix to verify the correctness of this simple operations.

Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

December 19, 2021

Theory

The system we investigated in this assignment is the **transverse-field Ising model** (TFI), which describes a sequence of N spins under the presence of a transverse magnetic field. The Hamiltonian for such a system is the following:

$$\hat{H} = -J \sum_{i=1}^{N-1} \hat{\sigma}_x^i \hat{\sigma}_x^{i+1} + \lambda \sum_{i=1}^{N} \hat{\sigma}_z^i \tag{1}$$

where the $\hat{\sigma}$ are the **Pauli operators**, λ is a coupling constant indicating an **external field** and we set $J = 1$. When we write the interaction term or the single site operator, acting on the whole sequence of spins, what is actually meant is that:

$$\hat{\sigma}_z^i = \mathbb{1}_1 \otimes \cdots \otimes \mathbb{1}_{i-1} \otimes \hat{\sigma}_z^i \otimes \mathbb{1}_{i+1} \otimes \cdots \otimes \mathbb{1}_N \tag{2}$$

$$\hat{\sigma}_x^i \hat{\sigma}_x^{i+1} = \mathbb{1}_1 \otimes \cdots \otimes \mathbb{1}_{i-1} \otimes \hat{\sigma}_x^i \otimes \hat{\sigma}_x^{i+1} \otimes \mathbb{1}_{i+2} \otimes \cdots \otimes \mathbb{1}_N \tag{3}$$

i.e. the Hamiltonian is a **sum over tensor products** of operators (identities and Pauli matrices) acting on all the sites. In the code we are going to **reuse** the double complex matrix `TYPE` defined in week 3. With the newly implemented subroutines and functions we are able to investigate the **spectrum** of the first $k = 4$ eigenvalues, highlighting the behavior of the system near the **quantum critical point** (QCP) at $\lambda = 1$

Two main code snippets

- We report the function to compute the tensor product between two generic matrices (left), and the one for the initialization of the Hamiltonian operator (right). For the former we recall that, being m_1 and m_2 two matrices:

$$m_1 \otimes m_2 = \begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \otimes \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 \cdot a_2 & a_1 \cdot b_2 & b_1 \cdot a_2 & b_1 \cdot b_2 \\ a_1 \cdot c_2 & a_1 \cdot d_2 & b_1 \cdot c_2 & b_1 \cdot d_2 \\ c_1 \cdot a_2 & c_1 \cdot b_2 & d_1 \cdot a_2 & d_1 \cdot b_2 \\ c_1 \cdot c_2 & c_1 \cdot d_2 & d_1 \cdot c_2 & d_1 \cdot d_2 \end{pmatrix}$$

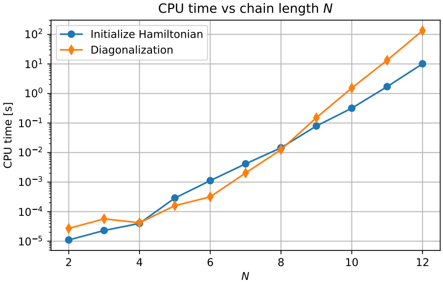
The indices `ii`, `jj` run over m_1, and we call (`rf`,`cf`) and (`rl`,`cl`) as the indices on the bigger matrix for the output of $m_1(ii, jj) \cdot m_2$.

```fortran
FUNCTION tp(m1 ,m2) RESULT(output)
![...]
ALLOCATE(output(n_row1*n_row2,n_col1*n_col2))
![...]
!cycle on the elements of the first matrix
DO ii = 1, n_row1
    DO jj = 1, n_col1
        !creating indices of the output matrix
        rf = (ii - 1) * n_row2 + 1
        cf = (jj - 1) * n_col2 + 1
        rl = ii * n_row2
        cl = jj * n_col2
        output(rf:rl,cf:cl) = m1(ii,jj)*m2
```

```fortran
FUNCTION TFI_ham_init(N, lambda) RESULT(output)
![...]
DO ii = 1, N
    !single site component
    output = output + tp(tp(id_mat_init(ii-1),
             sigma_z()),id_mat_init(N-ii))
ENDDO
output = output * lambda* COMPLEX(1.0d0, 0.0d0)
DO ii = 1, N-1
    output = output - tp(tp(tp(id_mat_init(ii-1),
    sigma_x()),sigma_x()),id_mat_init(N-ii-1))
```

Results - Efficiency

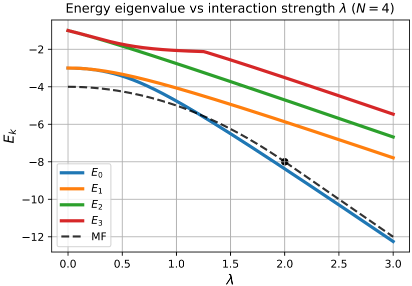
We used several Python **scripts** to **automatize** the work and produce the following graphs. Here we fix $\lambda = 1$ and we study how the **time required** for the initialization of the Hamiltonian, and for the diagonalization routine scales with N.

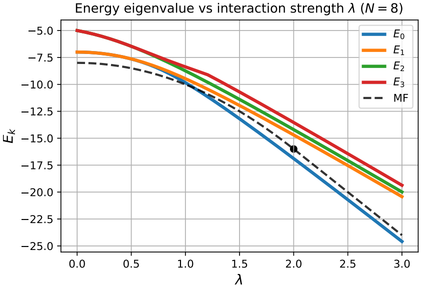


Both of them have an **exponential** behavior, and in particular the diagonalization subroutine requires more time than the initialization of the Hamiltonian. We can surely gain some performances with a better fine tuning of the parameter of the `ZHEEV` subroutine, but we will still have to wait a long time. In the end, this forces us to work with $N < 10$ systems and so to be far away from the behaviors expected in the thermodynamic limit $N \to \infty$.

Results - Energy spectrum

We run the code for $N = 4, 8$, $\lambda \in [0, 3]$, and we plot the first 4 energy eigenvalues, and the results obtainable with a **mean-field** approach (MF): $E/N = 1 - \lambda^2/4$ for $\lambda \in [-2, 2]$ and $E/N = -|\lambda|$ for $\lambda \notin [-2, 2]$.

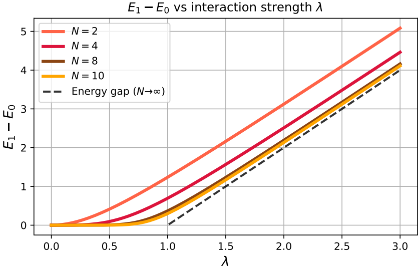




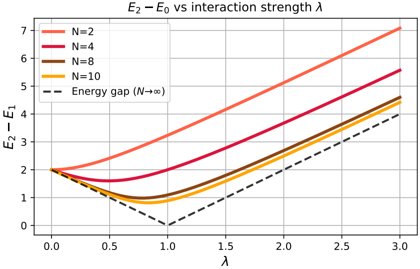
For a **weak field** $\lambda \to 0$ the two energy levels are degenerate, the interaction term is dominating and all the spins are anti-aligned to minimize the energy ($N = 4 : |\uparrow\downarrow\uparrow\downarrow\rangle$ or $|\downarrow\uparrow\downarrow\uparrow\rangle$ and $E = -3$). MF gives a wrong estimate of the GS energy: $-N \langle\psi| \sigma_x |\psi\rangle^2 = -4$, and of the QCP at $\lambda = 2$, rather than $\lambda = 1$. For a **strong field** $\lambda \to \infty$ the external field is dominating, and $E_0 \approx -N\lambda$. Furthermore, we also notice how the energy gap becomes narrower as N increases (comparing it to the left graph). Moreover, the **bifurcation point** (at $\lambda < 1$ in our cases) of the quantum phase transition approaches $\lambda = 1$ for $N \to \infty$.

Results - Energy gap

We plot here the energy gap between the first and the ground state eigenvalue $E - 1 - E_0$, and the one between the lowest excited state and the ground state $E_2 - E_0$.



$E_1 - E_0$ vs interaction strength λ



$E_2 - E_0$ vs interaction strength λ

Taking into account the two graphs of the previous slide, we see how below $\lambda = 1$ (the quantum critical point), the energy difference is 0, since the ground state is **two-folded** (degenerate). The solution approaches the theoretical one in the thermodynamic limit $N \to \infty$.

The lowest excited state E_2 has an energy higher than the ground state E_0 by a nonzero amount (non-vanishing for $N \to \infty$), that is $2(1 - \lambda)$ in the thermodynamic limit for $\lambda < 1$. The energy gap for $\lambda > 1$ and $N \to \infty$ is $2(\lambda - 1)$. The solution approaches the theoretical one in the thermodynamic limit $N \to \infty$.

Quantum Information and Computing (QIaC)

Prof. Simone Montangero

Tommaso Faorlin

Università degli Studi di Padova - Physics of Data

tommaso.faorlin@studenti.unipd.it

December 28, 2021

Theory

The system under investigation is the **transverse-field Ising model** (TFI), which describes a sequence of N spins under the presence of a transverse magnetic field, with Hamiltonian:

$$\hat{H}_N = -J \sum_{i=1}^{N-1} \hat{\sigma}_x^i \hat{\sigma}_x^{i+1} + \lambda \sum_{i=1}^{N} \hat{\sigma}_z^i \tag{1}$$

The **Real Space Renormalization Group** algorithm (RSRA) has been developed in order to efficiently approximate the diagonalization process for Hamiltonians of big dimensions. Starting from a system A (size N), the ground state of the bigger system $A + B$ (size $2N$) is composed of **few low energy eigenstates** of each subsystem of size N. The algorithm works as follows:

1. Double the system size $N \to 2N$ and consider the Hamiltonian of the full system

$$\tilde{H}_{2N} = \hat{H}_N^A \otimes \mathbb{1}_N^B + \mathbb{1}_N^A \otimes \hat{H}_N^B + \mathbb{1}_{N-1}^A \otimes \hat{\sigma}_x^A \otimes \hat{\sigma}_x^B \otimes \mathbb{1}_{N-1}^B \in \mathcal{H}^{2N \times 2N} \tag{2}$$

2. Diagonalize $\tilde{H}$ and build the matrix $\hat{P}$ out of its first N smallest eigenvectors. Project then the full matrix $\tilde{H}_{2N}$, and the two full interaction matrices H_A, H_B into the subspace spanned by its first N eigenvalues:

$$\hat{H}_N^T = \hat{P}^T \tilde{H}_{2N} \hat{P} \quad \hat{H}_N^{A,T} = \hat{P}^T (H_A \otimes \mathbb{1}_N^B) \hat{P} \quad \hat{H}_N^{B,T} = \hat{P}^T (\mathbb{1}_N^A \otimes H_B) \hat{P} \tag{3}$$

We use the last matrices as inputs to create another $\tilde{H}_{2N}$ and iterate the algorithm up until the **variation** of the ground state energy is smaller than $\delta = 10^{-12}$.

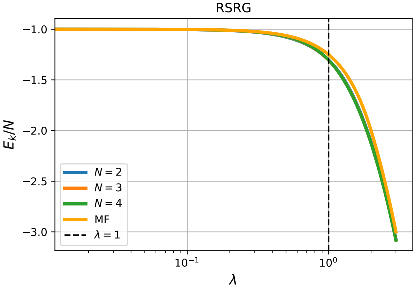
Main code

- We report the code snippet used in order to perform the iterations required by the RSRA algorithm. The idea behind is that we do a RSRA iteration whenever the **difference** between the ground state energy estimated at the previous iterations is smaller than $\delta = 10^{-12}$.

```fortran
H_N_T = TFI_ham_init(N, lambda)                      !TFI hamiltonian for N sites
HabA_T = tp(id_mat_init(N-1), sigma_x())             !sigma_x on N-th site
HabB_T = tp(sigma_x(), id_mat_init(N-1))             !sigma_x on (N+1)-th site

DO WHILE(ABS(gs-est_gs)>delta)
    gs = est_gs
    !initialize the extended Hamiltonian, accounting also for the interaction between N and N+1 sites
    H_2N = tp(H_N_T, id_mat_init(N)) + tp(id_mat_init(N), H_N_T) + tp(HabA_T, HabB_T)
    HabA_2N = tp(HabA_T, id_mat_init(N)).            !sigma_x on N-th site for total system
    HabB_2N = tp(id_mat_init(N),HabB_T)              !sigma_x on (N+1)-th site for total system
    eigvct = H_2N

    CALL ZHEEV('V','U', 2**(2*N), eigvct, 2**(2*N), eigvls, work, lwork, rwork, info)

    sys_size = sys_size * 2                          !the systems size is doubled
    est_gs = eigvls(1)/DBLE(sys_size)
    DO ii=1,SIZE(P,2)
        P(:,ii) = eigvct(:,ii)
    ENDDO
    H_N_T   =  MATMUL(MATMUL(TRANSPOSE(P),H_2N),P)   !projection of H_2N with matrix P
    HabA_T =  MATMUL(MATMUL(TRANSPOSE(P),HabA_2N),P) !projection of HabA_2N with matrix P
    HabB_T =  MATMUL(MATMUL(TRANSPOSE(P),HabB_2N),P) !projection of HabB_2N with matrix P
```

Results - Iterations

We used a Python **script** to **automatize** the work and produce the following graphs. Here we show how the ground-state energy (the value when the algorithm converges), normalized to the system size, varies when we change the interaction strength in the range $\lambda \in [0, 3]$.



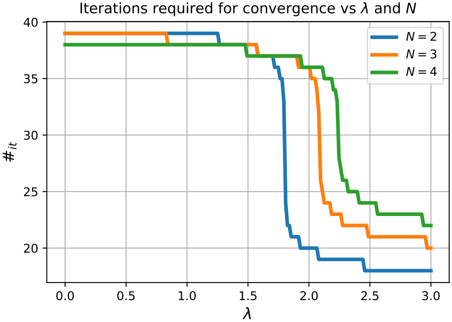
$$E_k/N = \begin{cases} -1 - \frac{\lambda^2}{4}, & \lambda \in [-2, 2] \\ -\lambda, & \lambda \notin [-2, 2] \end{cases} \tag{4}$$

The first thing we notice is that the curves for different N are practically **overlapped**, and this can be explained by the fact that we can diagonalize exactly the Hamiltonian for $2N$ spins. For example, the ground state with $\lambda = 0$ is in modulus equal to the size of the system ($2^{\#_{it}+1}$), and by normalizing it to the size itself we obtain -1.

The results are in perfect accordance with MF for small values of λ, while we can spot a discrepancy as this parameter increases.

Results - Energy

We now want to investigate how the **number of iterations** required by the RSRA for the convergence scales with the interaction strength λ for different N.



A striking feature of this plot is that the number of iterations $\#_{it}$ suddenly drops at a certain point, and that we observe different behaviors for the different sizes. In particular, for $\lambda = 2.5$, we reach the convergence faster with a smaller chain of spins.