UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Corso di Laurea in Fisica

Tesi di Laurea

Hilbert curve mapping for 3d-systems tensor network algorithms

Mappature di curve di Hilbert per sistemi 3d tramite algoritmi di tensor network

Thesis supervisor: Prof. Simone Montangero
Thesis co-supervisor: Daniel Jaschke

Grad student: Aurora Costantini

Anno Accademico 2021/2022

Abstract

A possible way to study many-body multi-dimensional quantum systems is by simulating them. In order to simulate many-body quantum systems on a classical computer, it is necessary to use an approximation: tensor network method represent a possible procedure. In some cases, multi-dimensional simulations need to be mapped to one dimension: one can achieve this mapping through the Hilbert curve of up to $8 \times 8 \times 8$ qubits, i.e. 512 qubits.
In this thesis, the focus is to implement a Hilbert curve generator to study the ground state quantum critical point of the quantum Ising model by using tensor network methods, while transitioning from a two-dimensional system to a three-dimensional one.

Un modo possibile per studiare i sistemi a multicorpi a più dimensioni è simulandoli. Per simulare questi sistemi a multi-corpi in un computer classico è necessario approssimarli: i tensor network method sono una procedura possibile. In alcuni casi, le simulazioni multi-dimensionali devono essere mappate in una dimensione: si può ottentere questa mappatura tramite la curva di Hilbert fino a $8 \times 8 \times 8$ qubit, cioè 512 qubit.
In questa tesi, lo scopo è quello di implementare un generatore di curve di Hilbert per studiare il punto critico quantistico dello stato fondamentale del modello di Ising quantistico tramite l'utilizzo di metodi di tensor network, il tutto transizionando da un sistema a due dimensioni ad uno a tre dimensioni.

Index

1 Introduction

The transverse-field Ising model, or quantum Ising model, is one of the most popular quantum models in terms of research, as it represents meaningful physics in a relatively accessible way. The transverse-field Ising model describes the properties of a system made of multiple particles disposed in a lattice, as it considers the spins of the particles projected on the z-axis and studies the energy associated with the agreeing or disagreeing of the two nearest-neighbour interactions' spins projections. The Hamiltonian describing the model includes the case in which there is an external magnetic field perpendicular to the z-axis that interacts with spins projections along the x-axis.
The d-dimensional transverse-field Ising model can be mapped to the $(d + 1)$-dimensional anisotropic classical Ising model: this equivalence is convenient as the classical Ising model has Ising's exact analytic solution for the one-dimensional case and Onsager's exact solution for the two-dimensional case [1]. On the one hand, this model's simplicity marks it as the best candidate for studying new theories applicable to other models, such as the Bose-Hubbard one, as shown in some recent studies [2, 3], and can be used to study long-range interactions [4]. On the other hand, the study of the quantum Ising model has great importance for the development of quantum information technologies in the last years [5, 6]. For example, the Ising model's simplicity can be used to study the effects of high-order correlation functions on the decoherence of qubits, comparing the results with the analytical solutions [7].
In order to properly study the properties of this model, we implemented a code to simulate the system and its Hamiltonian. However, such simulations require a simulation time that increases exponentially with the system's number of sites: this property did not allow us to simulate three-dimensional systems bigger than $2 \times 2 \times 2$ sites, without using an approximation. The solution relies on tensor network method [8–10]: we introduce the Tree Tensor Network (TTN) approximation, that keeps the simulation time under one week for systems up to 512 sites, i.e. an $8 \times 8 \times 8$ system. The system's sides length must be a power of two: in fact, the algorithm we use relies on the binary TTN method, which consists in continuous subdivisions of each tensor into two tensors.
With the aim to study and simulate this model in high-dimensions, we map the 2D and 3D coordinates of the lattice's sites into a one-dimensional curve. The curve we choose for this purpose is the Hilbert curve. The Hilbert curve is used in several cases among different fields, ranging from image rendering [11], to cosmology [12], and to medical studies [13].

This thesis will be focusing on the study of the quantum Ising Model transitioning from a two-dimensional to a three-dimensional lattice, allowing to study the quantum critical point of systems that have more than two dimensions. We confirm the idea that, being the three-dimensional systems described by more interactions between sites, the quantum critical point shifts into regions with bigger magnetic field strength. As a consequence, we study how properties such as the magnetic behaviour and the magnetization value change depending on the system's lattice arrangement and dimension.
Chapter 2 explains the underlying concepts of the tensor network methods, Chapter 3 talks about the Hilbert curve and the coding process, Chapter 4 describes the transverse-field Ising model and the process of the simulations, and Chapter 5 summarizes the conclusions.

2 Tensor network methods

Many-body quantum systems are described by wave functions. In the case covered in this thesis, we work with a system where the configuration of each site i is described by the spin direction along the z-axis. This spin configuration is described by $|\alpha_i\rangle$, where $|\alpha_i\rangle$ can be in any superposition of $|\alpha_i\rangle = |\uparrow\rangle$ and $|\alpha_i\rangle = |\downarrow\rangle$, so that a particular lattice configuration is described by $|\alpha_1\alpha_2...\alpha_N\rangle$, where N is the number of sites of the lattice. This means that the generic state is described by a N-rank tensor as seen in Eq. (1):

$$|\psi\rangle = \sum_{\vec{\alpha}} \psi_{\alpha_1\alpha_2...\alpha_N} |\alpha_1\alpha_2...\alpha_N\rangle\,, \tag{1}$$

where $|\psi\rangle$ is the state describing the linear combination of possible state configurations acquired by the system, and $\psi_{\alpha_1\alpha_2...\alpha_N}$ is the probability amplitude of each possible state configuration. The generic state can also be described by the tensorial form $\psi_{\alpha_1\alpha_2...\alpha_N}$.
We can describe a tensor graphically using the tensor network representation: the tensor itself is represented by a circle and the indexes are represented by lines. Such graphic representation is used to describe different mathematical objects, ranging from scalars to general n-rank tensors, i.e. as in Fig. 1.

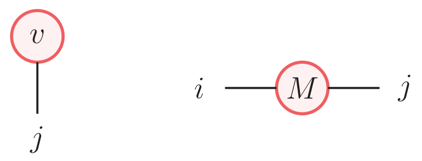


Figure 1: Tensor network representation of a vector v_j and a matrix $M_{i,j}$.

One can use this representation to show the contraction of indexes between tensors in a compact form. Using Einstein's notation where repeated indexes are summed, one can represent the contraction $A_{i,j}B_{j,k,l}C_{l,m}$ as in Fig. 2.

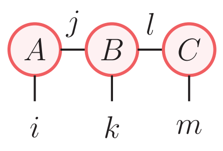


Figure 2: Tensor network representation of the contraction $A_{i,j}B_{j,k,l}C_{l,m}$.

It is important to notice that each line does not have to represent a singular index of dimension one. In fact, by using index fusion and index splitting one can represent an index with bigger dimension. For example, one could re-arrange a matrix into a vector:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \rightarrow \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}. \tag{2}$$

This re-arrangement causes the fusion of the two indexes describing the position in the matrix into a single index describing the position in the vector, e.g. b's position is described in the different representations as $(1, 2)$ in the matrix and (2) in the vector. In this way, any tensor of n indexes can be described in matrix form, with each of the two indexes' dimensions of the matrix being multiplied to n. Using the tensor network representation we can visualize the system's state as in Fig. 3.

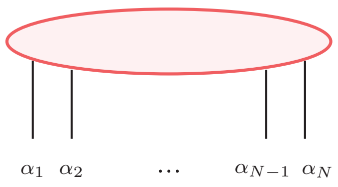


Figure 3: Tensor network representation of the state of the lattice $\psi_{\alpha_1\alpha_2\ldots\alpha_N}$.

When approaching the simulation, it is useful to calculate the computational cost of tensor contraction. This is important because, when computing a system wave function, the execution time is dictated by the number of operations needed to be performed in order to obtain it. One can extract a general rule that tells the number of operations necessary to compute the state, that is, the contraction complexity. Starting from a simple example of a scalar product as in Fig. 4, one can understand that the operations needed correspond to the dimension of the contracted index, m. On a general note, an estimation of the contraction complexity is given by the product of the dimensions of the free indexes and the contracted ones.

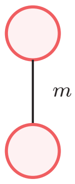


Figure 4: Tensor network representation of a scalar product between two vectors with bond size m.

In general, one could use advanced linear algebra manipulation algorithms in order to reduce the actual complexity, e.g. divide-and-conquer methods for matrices multiplication. As a consequence we can deduce, assuming the index dimension is d, that the contraction complexity for the inner product of the tensor in Fig. 3 is d^N. In the particular case of the transverse-field Ising model (see Chapter 4) $d = 2$, as it represents the two possible configurations of spin $s_z = \frac{1}{2}$.
The computational complexity grows exponentially with N, which leads to limited possibilities of simulations as it increases too quickly when increasing the number of sites. In order to simplify the situation, tensor network methods aim to decompose a general state into different tensor network forms using different useful approximations [14].

2.1 Mean field ansatz

One widely used approximation is the mean field, which can describe the system in a way that is easily computable, even though it is a rough approximation.
The mean field ansatz treats each site as independent from the others and subject to an external mean field, which can be different for every site. This ansatz is described as a tensor product of the N single-body wave functions as in Eq. (3):

$$|\psi^{MF}\rangle = \sum_{\alpha_1=1}^{d} \psi^{[1]}_{\alpha_1} |\alpha_1\rangle \otimes \sum_{\alpha_2=1}^{d} \psi^{[2]}_{\alpha_2} |\alpha_2\rangle \otimes ... \otimes \sum_{\alpha_N=1}^{d} \psi^{[N]}_{\alpha_N} |\alpha_N\rangle \,, \tag{3}$$

where d represents the index dimension, that is to say, the possible configurations that $|\alpha_i\rangle$ can be. In the Ising Model, $d = 2$, because it represents the possible spins configurations that can be $+\frac{1}{2}$ or $-\frac{1}{2}$ ($d = 2 \cdot \frac{1}{2} + 1$). Using the graphical tensor representation, this ansatz is represented by Fig. 5.

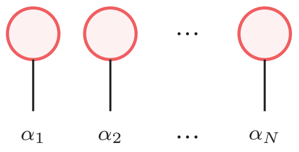


Figure 5: Tensor network representation of the state in mean field ansatz for N sites.

Instead of the exponentially scaling number of coefficients as in the case of a general state, in mean field approximation the memory cost of storing a wave function becomes Nd. Such scaling is linear in system size N, and therefore much more manageable. However, the downside is that it does not accurately represent the system and its behaviour if there is entanglement present in the system. In order to achieve a middle ground between computational complexity and accurate description we use the Matrix Product States (MPS) tensor network method, which is described in Chapter 2.2.

2.2 Matrix Product States

The main difference between the MPS method and the mean field ansatz is that in the former there is a description of the approximate interaction of the sites, while in the latter the sites are independent and are only subject to an external mean field. The tensor representation of the MPS approximation is represented in Eq. (4):

$$\psi_{\alpha_1\alpha_2...\alpha_N} = A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_2 s_1} ... A^{s_{N-1}}_{\alpha_{N-1},s_{N-2}} A_{\alpha_N,s_{N-1}}. \tag{4}$$

Visually, the MPS interaction is represented by the presence of auxiliary indexes between the tensors, as shown in Fig. 6, where the auxiliary indexes' dimension is m.

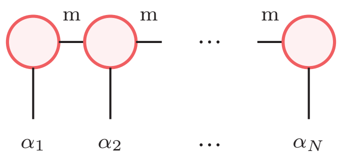


Figure 6: Tensor network representation of the MPS method of N sites with bond dimension m.

Every wave function can be represented by an MPS if one chooses the correct dimension of m, but generally, it is better to profit from the approximation, choosing

a smaller dimension to decrease the computational cost. In this case, the memory cost becomes Ndm^2, which grows linearly in N. The mean field ansatz is a limit of the MPS as $m \to 1$.
The MPS is a particular case of a much wider class of tensor networks called Tree Tensor Network (TTN). The structure of the TTN has each tensor with rank three: the binary TTN (bTTN), which divides each tensor into two other tensors, is shown in Fig. 7.

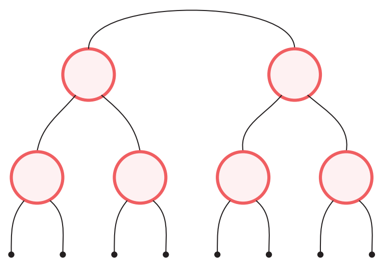


Figure 7: Tensor network representation of the bTTN method of 8 sites.

This type of tensor network has the practical advantage of successfully describing periodic systems [15], and is the method we use for the simulations within this thesis, as it is already proven to be effective for simulating the ground state of the 1D and 2D quantum Ising model [16, 17].
For the case of two-dimensional systems, another worth noting tensor network representation is available, called Projected Entangled Pair States (PEPS) [18, 19]. PEPS represents a natural extension of the one-dimensional MPS to two dimensions and is visualized in Fig. 8.

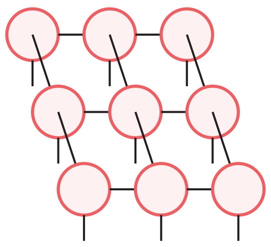


Figure 8: Example of the PEPS tensor network representation for 9 sites.

3 The Hilbert curve

In order to represent a multi-dimensional space we need to find a way to describe it in a complete and useful way. We aim to find a description that covers the entire space while keeping track of the interactions between nearest-neighbour sites, mapping the 2D or 3D system disposition into a one-dimensional curve. The snake curve could serve the purpose, but as it turns out the best choice to preserve locality is the Hilbert curve [13, 17]. We, therefore, use a Hilbert curve, which is defined as a continuous fractal space-filling curve.
Looking at a singular site, we observe that the medium distance between this site and its nearest neighbours in the 1D-curve representation is smaller in the Hilbert curve case with respect to the snake curve: Fig. 9 shows an example of the site at position $(2, 2)$. The projected position of the $(2, 2)$ site and its nearest neighbours on the one-dimensional axis, representing the curve filling order, show that in the Hilbert curve their proximity is preserved better than the snake curve. As studied in Ref. [17], on average this property holds for every site.

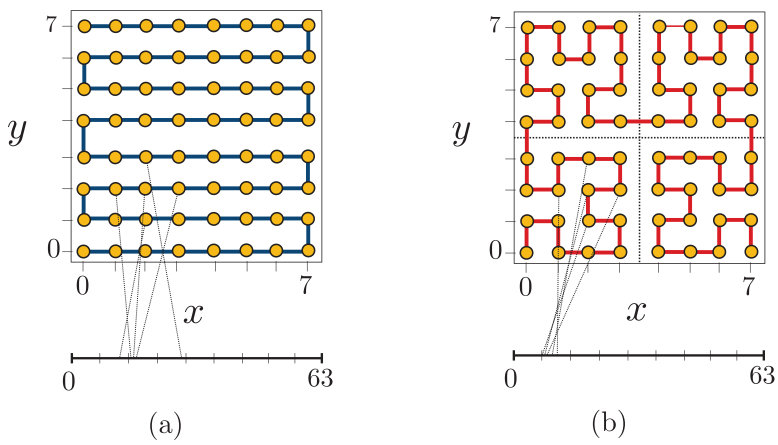


Figure 9: Representation of the preserved locality of a 2D lattice disposed into a) a snake curve and b) a Hilbert curve. The figure is taken from the study cited in [17].

For these reasons, we focus on coding two-dimensional and three-dimensional Hilbert curves on a generic square or cube with a side being a power of two, and then extend the code into a generic rectangle or box, with the sides being different powers of two.

3.1 2D Hilbert curve

In order to code a generic two-dimensional Hilbert curve we must follow a certain set of rules. The first step is to take a square, divide it into four equal squares and decide in which order the curve will cover them. This is the first-order Hilbert curve and it is represented in Fig. 10, where the written numbers order the squares (sites of the lattice) in the covering order.

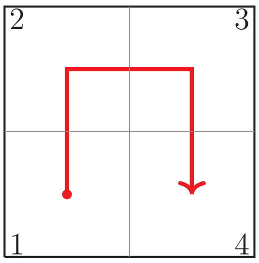


Figure 10: First-order Hilbert curve in a 2×2 square.

Being the Hilbert curve a fractal curve, the following steps to create the i-th order Hilbert curve are repeated. Each of the squares created is divided into other four squares, connected through a first-order Hilbert curve. The steps following the first-order one do not have the choice of the covering sequence, because they must create a continuous curve. While coding the curve, we must follow the same sequence as the first-order case. In Fig. 10, for example, the first square that has to be divided and covered by the second-order Hilbert curve is the one labelled "1". This is necessary in order to create a recursive function that will work regardless of the chosen division of the original square. The second-order Hilbert curve looks like Fig. 11.

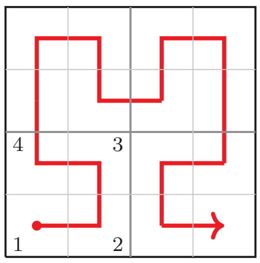


Figure 11: Second-order Hilbert curve in a $2^2 \times 2^2$ square.

It is worth noting that, in order to follow the filling sequence of the first-order Hilbert curve the choice about the subdivision orientation curve is fixed. This

process can be repeated n times in order to divide the original square space into $2^n \times 2^n$ squares.
After these steps, it is straightforward to extend the curve to the case of a general rectangle made of $2^n \times 2^m$ squares. If we follow the initial order presented in the first-order Hilbert curve, we can observe that the curve always ends up in the bottom-right square: if the rectangle's width is greater than its height, it is sufficient to repeat the Hilbert curve as many times as needed. If $m = n + 1$, for example, there will be one repetition, as shown in Fig. 12.

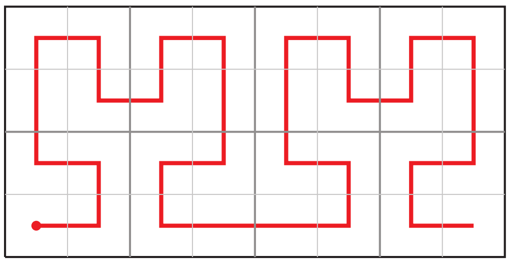


Figure 12: Example of a Hilbert curve covering a rectangular space of $2^2 \times 2^3$.

If the rectangle's width is smaller than its height, instead, it is sufficient to do the same procedure as for the longer case followed by mirroring the result, as one can see in Fig. 13.

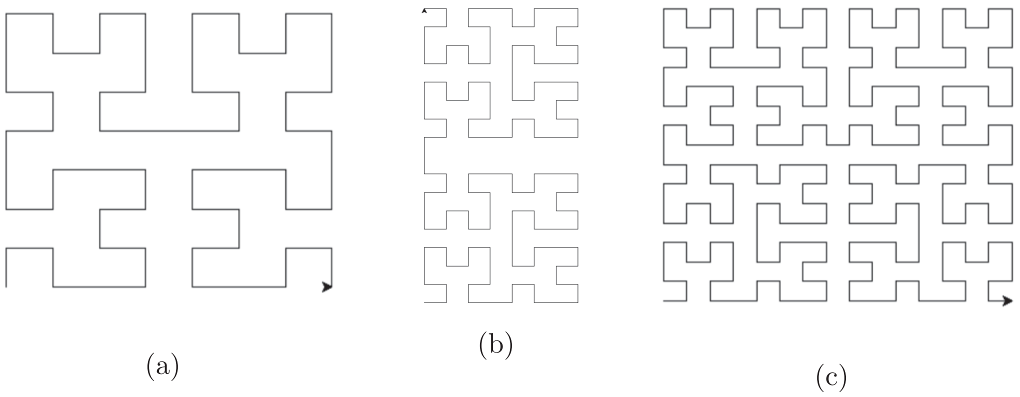

(a) (b) (c)

Figure 13: Representation of the Hilbert curve a) in the $2^3 \times 2^3$ case, b) the $2^3 \times 2^4$ case, and c) the $2^4 \times 2^4$ case.

The central example in Fig. 13 does not end in the bottom-right corner: that is

because the rectangle is higher than the original square, so the code does the final mirroring, as said before, in order to visually represent what we planned.
The code used to achieve this representation used TURTLE out of the Python standard library, which consists in a moving and rotating cursor, i.e. the turtle, that writes down the areas it covers. The recursive function has two variables: "angle" and "k". The first variable represents the direction of rotation of the turtle, and it has value $90°$ or $-90°$ depending on the case. The second variable, k, represents the order of the Hilbert curve and derives from the input-given power of two wanted for the side length (n, m). Listing 3.1 shows the corresponding piece of the code inside the recall function. There are four calls at itself diminishing the k variable by one, representing the recursive divisions of every square into four squares, which are covered by a Hilbert curve with an inferior order.

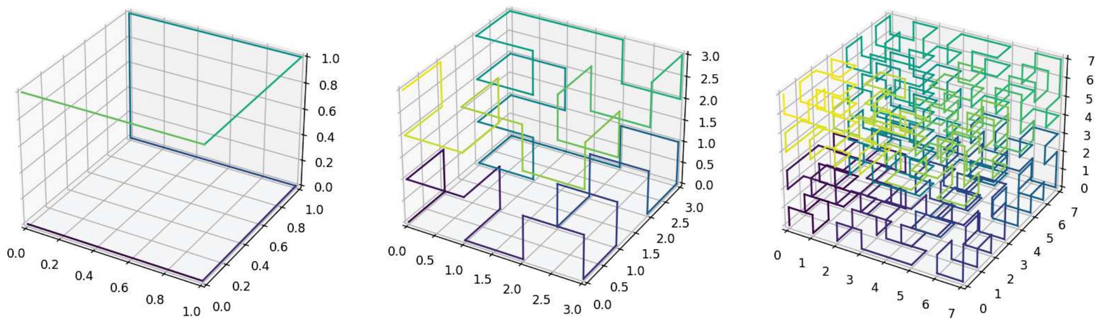
```python
def hilbert_curve(angle, k):

    if k==1:

        forward(0.5)
        right(angle)
        forward(0.5)
        right(angle)
        forward(0.5)

    else:

        hilbert_curve(-angle, k - 1)

        if(k%2==0):
            right(angle)
        forward(0.5)
        if (k % 2 == 1):
            right(angle)
        hilbert_curve(angle, k - 1)

        if (k % 2 == 0):
            right(-angle)
        forward(0.5)
        if (k % 2 == 0):
            right(-angle)
        hilbert_curve(angle, k - 1)

        if (k % 2 == 1):
            right(angle)
        forward(0.5)
        if (k % 2 == 0):
            right(angle)
        hilbert_curve(-angle, k - 1)
```

Listing 1: Code section of the recursive function providing the two-dimensional Hilbert curve.

The recursive calls stop when the input order $k = 1$, providing a special if-clause that returns the procedure of drawing the first-order Hilbert curve with the according direction and rotation.

3.2 3D Hilbert curve

The three-dimensional Hilbert curve is less straightforward to code but follows a similar set of rules. Likewise the initial division of a square into four squares for the 2D case, we start by dividing the initial cube into eight cubes: after this, we can define a recursive function that divides the new cubes into eight more cubes, and so on. In Appendix A we report the code with the recursive calls of the function of itself [20]. The python library used for the visualization is MATPLOTLIB.



(a) (b) (c)

Figure 14: Representation of the Hilbert curve in a) the $2 \times 2 \times 2$ case, b) the $2^2 \times 2^2 \times 2^2$ case, and c) the $2^3 \times 2^3 \times 2^3$ case. The curve starts at the $(0, 0, 0)$ position in purple, and arrives at the endpoint in green-yellow.

Figure 14 shows how the orientation of the curve is chosen: the first-order 3D Hilbert curve starts in point $(0, 0, 0)$, and the following point to be covered could be either $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$. All these alternatives are valid, as we could always find the right path to cover the rest of the cube: the chosen direction is arbitrary.

We approach the second-order 3D Hilbert curve as the 2D case: we follow the sequence of the first-order curve as a path and fill every subdivision keeping in mind the following cube to be filled. For example, the second covered site in Fig. 14's a) case is $(1, 0, 0)$, shifting in the x-direction: this translates in the need to create a cube that ends in a shift in the x-direction for the first 8 covered sites of the second-order Hilbert curve. How the cube achieves the x-shift itself is not important: the other cubes' filling order follows the same logic.
The natural evolution of this curve is to take the general box, with each side being a different power of two. In order to achieve this plotting, we start choosing the biggest possible cube that can be repeated in order to cover the whole space, corresponding to the side with the least sites, and proceed by following the pattern of a two-dimensional Hilbert curve. In Fig. 15, we represent the sequence followed by the code in red, while the thin black line represents the underlying two-dimensional Hilbert curve. Here, we assume that the side with the least sites is n_y.

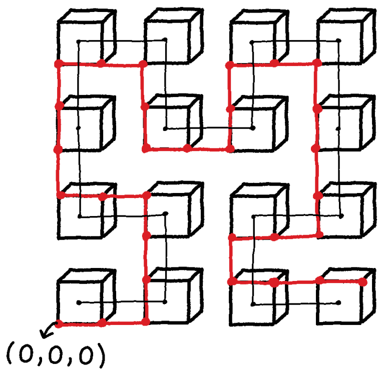


Figure 15: Visualization of the code for a general box: example with $n_x = n_z$. The red line represents the sequence followed by the code and the thin black line represents the underlying two-dimensional Hilbert curve.

The starting point of the curve is always $(0, 0, 0)$ and the two points in two of the eight vertices represent the starting and the finishing point of the cubes' creation. The actual code is presented in Appendix B. The cases where n_x or n_z are the minimum are implemented by switching the values of indexes as needed, running the program assigning the smallest value to n_y and then creating a matrix containing the coordinates in the correct order. For example, the $2^2 \times 2^3 \times 2^4$ looks like Fig. 16.

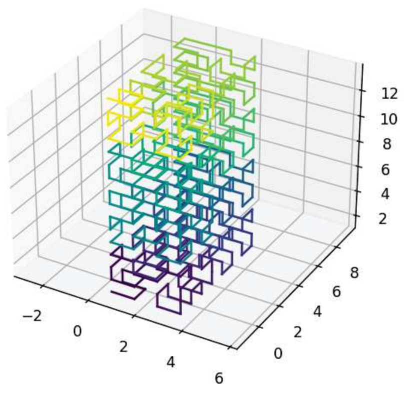


Figure 16: Visualization of the Hilbert curve for the $2^2 \times 2^3 \times 2^4$ rectangular cuboid.

4 The Quantum Ising Model

The transverse-field Ising Model is a quantum version of the classical Ising Model. It consists of a lattice of particles with spin $S = \frac{1}{2}$, and it features a magnetic field along the x-axis. The Hamiltonian is described by Eq. (5):

$$H = -J \sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z - g \sum_j \sigma_j^x = H_0 + H_1, \tag{5}$$

where the notation $\langle i, j \rangle$ represents nearest neighbour sites. The σ_i are operators acting on the i-th site in the 3D system, representing the spin projections along the particular axis, and are described by the Pauli matrices:

$$\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma^y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad \sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{6}$$

On the one hand, H_0 represents the interaction between the sites. J represents the strength of interaction between the sites and for $J > 0$ we are in the ferromagnetic case, while for $J < 0$ we are in an antiferromagnetic case. On the other hand, H_1 represents the sites' interactions with the external magnetic field, whose strength is controlled by the parameter g. In the limit $g \to \infty$ the system behaves paramagnetically.

The main cause of quantum behaviours in the system arises from the fact that the two terms in the Hamiltonian do not commute, that is: $[\sigma_x, \sigma_z] \neq 0$. This means that if the external field is zero, the spins align in the z-direction, in which case the degenerate ground state is:

$$|\psi\rangle = \frac{|\uparrow\uparrow ... \uparrow\rangle \pm |\downarrow\downarrow ... \downarrow\rangle}{\sqrt{2}}. \tag{7}$$

The TTN method helps the algorithm by breaking the symmetry of the system, and decides which sense is the one the spins will take along the z-direction. Adding the magnetic field along the x-direction creates a situation in which there is no common eigenbasis of H_0 and H_1, creating the quantum fluctuations. Increasing the field strength enough makes the quantum fluctuations less noticeable, as the spins align according to the external field, which leads to lower energy in the system. In this case, the ground state is:

$$|\psi\rangle = |\rightarrow\rightarrow ... \rightarrow\rangle\,, \tag{8}$$

along the external magnetic field strength.
In the following, we present the results of the simulations focusing on two studies: the search of the quantum critical point (QCP), explained in detail in Chapter 4.1, and the convergence to the best value by increasing the bond dimension. In both studies, we choose a ferromagnetic scenario by fixing $J = 1$, without loss of generality.

4.1 Quantum critical point

The quantum critical point is a point in the phase diagram where a sudden change in the qualitative properties of the ground state, i.e. a quantum phase transition, occurs. Taking into account the Hamiltonian of Eq. (5), one can show that if $g = 0$, there are no conflicts in terms of commuting operators in the Hamiltonian, meaning that the ground state shows a ferromagnetic order along the z-axis. In the limit $|g| \to \infty$, in contrast, the second term in the Hamiltonian prevails and the spins align along the x-axis.
In this study, we, therefore, look for the QCP by looking at the magnetization along the z-axis by varying the field strength. In a straightforward way. the magnetization would be computed following Eq. (9):

$$M' = \frac{1}{N}\sum_{i=1}^{N}\langle\sigma_i^z\rangle, \tag{9}$$

where N is the total number of sites and the sum is looping over all the sites in the box. The magnetization holds the information about the QCP because the TTN breaks the symmetry described in Eq. (7) and shows a local magnetization in the z-direction, which disappears when the external magnetic field g becomes strong enough and the spins align along the x-axis. But this feature relies on a numerical instability and therefore it is safer to compute the magnetization through the correlations between spins:

$$M = \frac{1}{N^2} \sum_{i,j=1}^{N} \langle \sigma_i^z \sigma_j^z \rangle. \tag{10}$$

Using Eq. (10), we can not achieve a null magnetization even in the paramagnetic limit, because the mean also counts the cases in which $i = j$, that do not correspond to a correlation and return $\langle \sigma_i^z \sigma_i^z \rangle$=1.
The one-dimensional quantum Ising model in the thermodynamic limit faces a phase transition at $\frac{g}{J} = 1$. In order to align all the spins in the x-direction for the two-dimensional and three-dimensional case we expect a stronger external field because each site has more interactions. Here, we study the shift of the g value corresponding to the quantum phase transition, from now on called g^*, starting from a two-dimensional system of size 8×8 and transitioning into three-dimensional systems of size $8 \times 8 \times 2$, $8 \times 8 \times 4$ and $8 \times 8 \times 8$, while using bond dimension of $m = 100$.
In order to simulate the system we used a collection of codes offered by [15, 21, 22], which already had the codes regarding the tensor network simulation and the Hamiltonian construction for various models, including the two-dimensional transverse-field Ising model. We implemented a code that could simulate the three-dimensional transverse-field Ising model and the codes from the Appendixes regarding the three-dimensional Hilbert curve.
Figure 17 shows that, as expected, the value of the external field strength g necessary to set the magnetization M to zero increases as the third dimension increases from $N_3 = 1$ to $N_3 = 8$. In order to find the quantum phase transition we search the value of g for which the magnetization goes to zero: we find this value, g^*, studying the first derivative of the magnetization with respect to g. We choose g^* based on when the first derivative is maximum, indicating that the magnetization is changing the fastest. As stated in Ref. [23], for the $8 \times 8 \times 1$ lattice we expect a quantum phase transition around $g^* \approx 3$: taking into account our restrained amount of time for this thesis' simulations, our results are compatible. The results are:

N_3	1	2	3	4
g^*	2.6	3.7	4.1	4.3

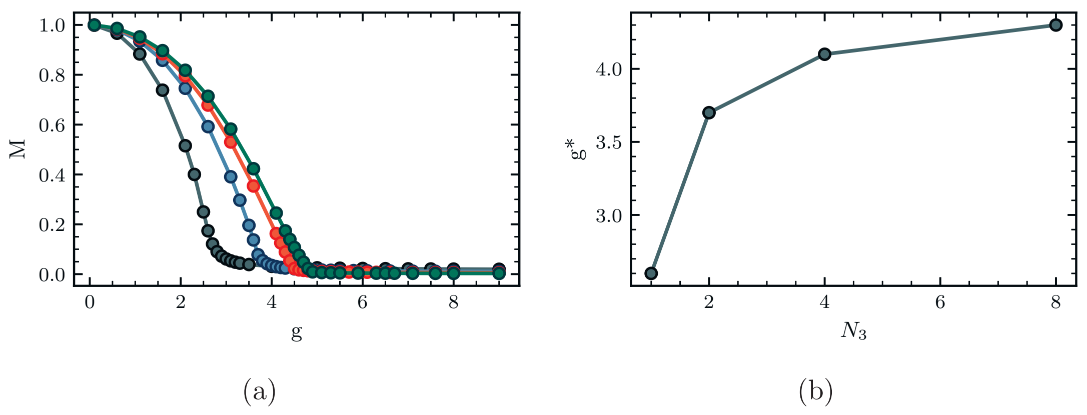


Figure 17: We show the variation of the quantum critical point by increasing the third dimension N_3, where g represents the field strength, M represents the magnetization along the z-axis and N_3 represents the system's third direction's number of sites. The critical value g^* is the g-value corresponding to the quantum phase transition. In a) we can see the variation of the magnetization by varying the magnetic field strength (black for the $8 \times 8 \times 1$, blue for the $8 \times 8 \times 2$, red for the $8 \times 8 \times 4$, and green for the $8 \times 8 \times 8$). The system behaves ferromagnetically before the quantum phase transition occurs, while the system behaves paramagnetically after approaching the QCP. In b) we can see the variation of the quantum critical point field strength increasing the third dimension.

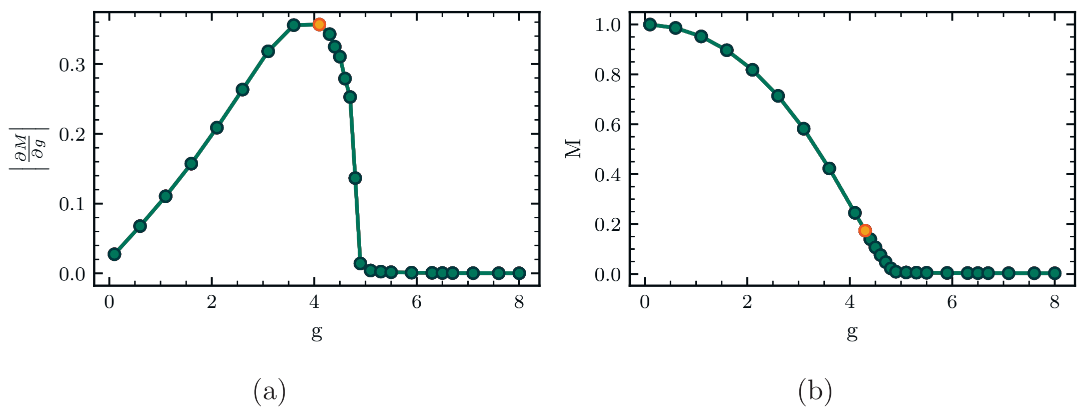
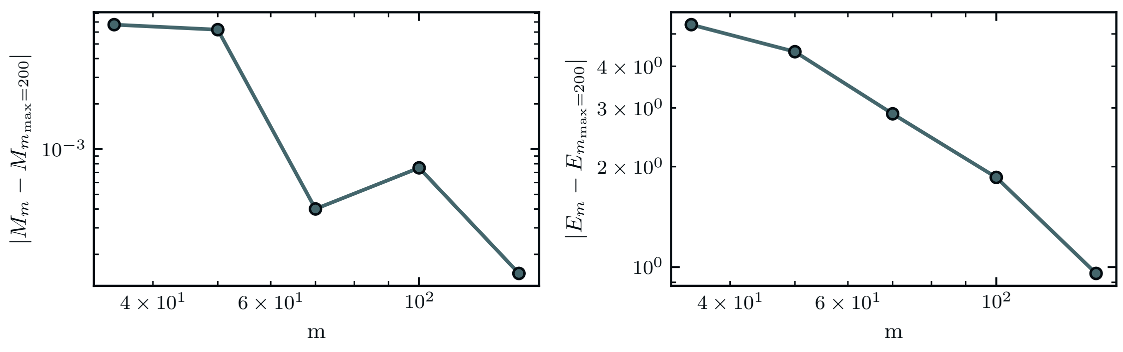


Figure 18: We show how the QCP was chosen, using the $8 \times 8 \times 8$ lattice as an example. In a) is presented the first derivative of the magnetization M with respect to the magnetic field strength, g. In orange is highlighted the max value of the first derivative. In b) is highlighted in orange the correspondent point.

We notice that the biggest shift in g^* occurs when we transition from the two-dimensional into the three-dimensional system: when we increase the third dimension's site numbers in an already three-dimensional system, the shift is not as radical. This result is sensible: the biggest correlations are between closer sites, and on average the two-dimensional sites have 3.5 nearest neighbours, while the $8 \times 8 \times 2$, the $8 \times 8 \times 4$, and the $8 \times 8 \times 8$ have, respectively, 4.5, 5 and 5.25 nearest neighbours on average. The biggest shift in average close neighbours is, in fact, between the two-dimensional system and the $8 \times 8 \times 2$ system.

4.2 Convergence study

As reported in Chapter 2, the simulations need to be approximated via tensor network methods, where the degree of the approximation gives us control over the simulation time. In our case, the capability of our computer allows us to take bond dimensions up to $m = 100$. This approximation is to be checked for convergence, given that the correct bond dimension to describe exactly the system potentially goes, in the worst-case scenario, to $m = d^{\frac{N}{2}}$, but it is a good choice in order to keep both the simulation time and the approximation error under control.



(a) (b)

Figure 19: Subsection a) represents the error of the magnetization M with respect to the magnetization value of the simulation with bond dimension $m = 200$, in a log scale. It clearly shows that from bond $m = 70$ the magnetization is stabilized. Figure b) represents the energy error with respect to the energy value of of the simulation with bond dimension $m = 200$.

We study the magnetization convergence of the lattice of dimension $8 \times 8 \times 8$, studying the case of $g = 4.9$, and considering bond dimensions m of 35, 50, 70,

100, 141, and 200. We choose to study the system at $g = 4.9$ because it is close to the QCP for the $8 \times 8 \times 8$ box, implying that this point is probably a difficult one to converge correctly. In particular, we plot the magnetization difference from the best magnetization value. The best value was chosen based on the energy of the system, which in this case is lower at bond dimension 200. This behaviour is expected: having a larger m dimension implies that the system is described closely to its real, not approximated behaviour, allowing it to settle down to smaller and therefore more stable energy values. In Fig. 19, the results are presented.
We notice that the magnetization begins to stabilize around bond dimension $m = 70$, where the error at $m = 100$ is bigger probably due to fluctuations of the systems caused by the tensor network approximation. The error of the $m = 100$ simulation with respect to the $m = 200$ simulation is eight per cent, and the energy does not converge: this shows that the study could have been better simulated with higher bond dimension, but is not unprofitable.

5 Conclusion

In this work, we studied the properties of the quantum phase transition of the three-dimensional transverse-field Ising model varying the strength of the external magnetic field. In order to approach this study, we implemented a code for the Hilbert curve of a general box having the side lengths being a power of two: the choice of using the Hilbert curve for mapping the system among all possible curves is justified by previous studies of the interaction-preserving properties of the Hilbert curve [17]. In particular, we focused on studying the shift of the quantum critical point varying the force of the external field starting from a two-dimensional square and transitioning into a three-dimensional cube. In order to simulate the system in a controlled amount of time, we used proper tensor network techniques, focusing on the Tree Tensor Network approximation. We also studied the convergence of the magnetization varying the bond dimension of the TTN approximation.
We discovered that, transitioning from an $8 \times 8 \times 1$ to an $8 \times 8 \times 8$ lattice, the QCP increases the most in the shift from a two-dimensional into a three-dimensional system, while it tends to increase less while transitioning from an already three-dimensional system into a bigger one. This result was expected, as the biggest shift of average nearest-neighbour sites occurs in the transition from the 8×8 lattice to the $8 \times 8 \times 2$ lattice. We studied the efficacy of our approximation through the convergence study, which showed us that bond dimension $m = 100$ is not sufficient to consider the system's simulation converged, but the results are nevertheless usable as a first approximation. We would have preferred to run the simulations with bigger bond dimensions, but the time dedicated to this thesis could not allow us to run simulations that took longer than a week.
With this study, we expanded the discussion on the quantum Ising Model. Studying the three-dimensional quantum Ising Model could help in understanding the general properties of different systems by testing approximations and procedures on this model before implementing them in other, more difficult models such as the Bose–Hubbard model, leading the way in new technologies in the quantum information field.
We would like to improve these results using an increased bond dimension: the simulation of bigger systems and higher bond dimensions are beyond the scope and time frame of this thesis. Future studies could also be focused on states other than the ground state, such as the first-excited states, and in simulating systems of bigger size.

A Cube code

In Listing A we present the code used for the filling of a cubic lattice through the Hilbert curve. It is important to notice the eight recursive calls to the function itself on lines 44 – 67.

```
1  def get_3d(s, x, y, z, dx, dy, dz, dx2, dy2, dz2, dx3, dy3, dz3, xs,
       ys, zs, m, n):
2      '''
3      s: number of sites of the side
4  
5      (x, y, z): coordinates of the cube beginning
6      (dx, dy, dz): direction of the first vector of the cube
7      (dx2, dy2, dz2): direction of the second vector of the cube
8      (dx3, dy3, dz3): direction of the third vector of the cube
9      (xs, ys, zs): the arrays that will be returned filled with the
   coordinates in sequence
10 
11     m: keeps track of the position of the arrays, needs to be
   updated in the various recursive calls
12 
13     n: number of total sites of the cube
14     '''
15 
16     if (s == 1):
17         xs[m] = x
18         ys[m] = y
19         zs[m] = z
20         m=m+1
21         return m
22     else:
23         s=s/2
24         if (dx < 0):
25             x -= s * dx
26         if (dy < 0):
27             y -= s * dy
28         if (dz < 0):
29             z -= s * dz
30         if (dx2 < 0):
31             x -= s * dx2
32         if (dy2 < 0):
33             y -= s * dy2
34         if (dz2 < 0):
35             z -= s * dz2
36         if (dx3 < 0):
37             x -= s * dx3
```

```
        if (dy3 < 0):
            y -= s * dy3
        if (dz3 < 0):
            z -= s * dz3
        m = hilbert_curve(s, x, y, z, dx2, dy2, dz2, dx3, dy3, dz3,
            dx, dy, dz, xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx, y + s * dy, z + s * dz,
            dx3, dy3, dz3, dx, dy, dz, dx2, dy2, dz2, xs, ys, zs,
            m, n)
        m = hilbert_curve(s, x + s * dx + s * dx2, y + s * dy +
            s * dy2, z + s * dz + s * dz2, dx3, dy3, dz3, dx, dy,
            dz, dx2, dy2, dz2, xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx2, y + s * dy2, z + s *
            dz2, -dx, -dy, -dz, -dx2, -dy2, -dz2, dx3, dy3, dz3,
            xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx2 + s * dx3, y + s * dy2 +
            s * dy3, z + s * dz2 + s * dz3, -dx, -dy, -dz, -dx2,
            -dy2,-dz2, dx3, dy3, dz3, xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx + s * dx2 + s * dx3, y +
            s * dy + s * dy2 + s * dy3, z + s * dz + s * dz2 + s *
            dz3, -dx3, -dy3, -dz3, dx, dy, dz, -dx2, -dy2, -dz2,
            xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx + s * dx3, y + s * dy +
            s * dy3, z + s * dz + s * dz3, -dx3, -dy3, -dz3,
            dx, dy, dz, -dx2, -dy2, -dz2, xs, ys, zs, m, n)
        m = hilbert_curve(s, x + s * dx3, y + s * dy3, z + s * dz3,
            dx2, dy2, dz2, -dx3, -dy3, -dz3, -dx, -dy, -dz, xs,
            ys, zs, m, n)

        if (m == n and s == 1):
            return  xs[m-1], ys[m-1], zs[m-1], xs, ys, zs
        return m
```

Listing 2: Recursive calls of the function providing the three-dimensional Hilbert curve in a general $2^n \times 2^n \times 2^n$ cube.

B General box code

In Listing B we present the code used for the filling of a general box with the sides being a power of two long. It is presented only the code piece that creates the cube fillings as presented in Fig. 15: in fact, there are several calls to the code of the cube Hilbert curve function.

```
'''
```

```
nmin = the minimum of nx, ny and nz

(x_prior, x_current, x_post): based on the 2d Hilbert curve, this
numbers represent the prior position covered and the following one
on the x-direction
(z_prior, z_current, z_post): based on the 2d Hilbert curve, this
numbers represent the prior position covered and the following one
on the z-direction

hilbert2d: numpy matrix containing the position of the 2d Hilbert
curve covering with sides pow(2, nx) and pow(2, nz)

(xs, ys, zs): temporary arrays
(coordx, coordy, coordz): arrays containing the covering
(pos_x, pos_y, pos_z): current position
'''

min_dim = pow(2, nmin)

x_current = np.where(hilbert2d == 0)[1]
z_current = np.where(hilbert2d == 0)[0]
x_post = np.where(hilbert2d == 1)[1]
z_post = np.where(hilbert2d == 1)[0]

if x_post == x_current + 1:
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, 0, 1, 0, 0, 0, 1, 1, 0, 0, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
    xs, ys, zs)
elif z_post == z_current + 1:
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, 1, 0, 0, 0, 1, 0, 0, 0, 1, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
    xs, ys, zs)

for i in range(1, int(pow(2, (nx - ny)) * pow(2, (nz - ny)) - 1)):
        x_prior = np.where(hilbert2d == i - 1)[1]
        z_prior = np.where(hilbert2d == i - 1)[0]
        x_current = np.where(hilbert2d == i)[1]
        z_current = np.where(hilbert2d == i)[0]
        x_post = np.where(hilbert2d == i + 1)[1]
        z_post = np.where(hilbert2d == i + 1)[0]

        if x_post == x_current + 1:
            if z_prior == z_current + 1:
                pos_z = pos_z - step - 1
                pos_x = pos_x - step
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, -1, 0, 0, 0, 1, 0, 0, 0, -1,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)
            elif (z_prior == z_current - 1) or (x_prior == x_current
            - 1):
                if (z_prior == z_current - 1):
                    pos_z = pos_z + 1
                elif (x_prior == x_current - 1):
                    pos_x = pos_x + 1
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 0, 1, 0, 0, 0, 1, 1, 0, 0,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)

        elif x_post == x_current - 1:
            if z_prior == z_current - 1:
                pos_z = pos_z + 1
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 1, 0, 0, 0, 1, 0, 0, 0, 1,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)
            elif (z_prior == z_current + 1) or (x_prior == x_current
            + 1):
                if (z_prior == z_current + 1):
                    pos_z = pos_z - step - 1
                    pos_x = pos_x - step
                elif (x_prior == x_current + 1):
                    pos_x = pos_x - step - 1
                    pos_z = pos_z - step
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 0, 1, 0, 0, 0, -1, -1, 0, 0,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)

        elif z_post == z_current + 1:
            if x_prior == x_current + 1:
                pos_z = pos_z - step
                pos_x = pos_x - step - 1
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 0, 1, 0, 0, 0, -1, -1, 0, 0,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)
            elif (z_prior == z_current - 1) or (x_prior == x_current
            - 1):
                if (z_prior == z_current - 1):
                    pos_z = pos_z + 1
                elif (x_prior == x_current - 1):
                    pos_x = pos_x + 1
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 1, 0, 0, 0, 1, 0, 0, 0, 1,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)

        elif z_post == z_current - 1:
            if x_prior == x_current - 1:
                pos_x = pos_x + 1
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, 0, 1, 0, 0, 0, 1, 1, 0, 0,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)
            elif (z_prior == z_current + 1) or (x_prior == x_current
            + 1):
                if (z_prior == z_current + 1):
                    pos_z = pos_z - step - 1
                    pos_x = pos_x - step
                elif (x_prior == x_current + 1):
                    pos_x = pos_x - step - 1
                    pos_z = pos_z - step
                pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim,
                pos_x, pos_y, pos_z, -1, 0, 0, 0, 1, 0, 0, 0, -1,
                xs, ys, zs, 0, sites)
                coordx, coordy, coordz = add_coordinates(coordx,
                coordy, coordz, xs, ys, zs)


x_prior = np.where(hilbert2d == int(pow(2, (nx - ny))
* pow(2, (nz - ny)) - 2))[1]
z_prior = np.where(hilbert2d == int(pow(2, (nx - ny))
* pow(2, (nz - ny)) - 2))[0]
x_current = np.where(hilbert2d == int(pow(2, (nx - ny))
* pow(2, (nz - ny))) - 1)[1]
z_current = np.where(hilbert2d == int(pow(2, (nx - ny))
* pow(2, (nz - ny))) - 1)[0]
```

```

if x_prior == x_current + 1:
    pos_x = pos_x - step - 1
    pos_z = pos_z - step
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, 0, 1, 0, 0, 0, -1, -1, 0, 0, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
     xs, ys, zs)
elif z_prior == z_current - 1:
    pos_z = pos_z + 1
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, 1, 0, 0, 0, 1, 0, 0, 0, 1, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
     xs, ys, zs)
elif x_prior == x_current - 1:
    pos_x = pos_x + 1
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, 0, 1, 0, 0, 0, 1, 1, 0, 0, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
     xs, ys, zs)
elif z_prior == z_current + 1:
    pos_z = pos_z - step - 1
    pos_x = pos_x - step
    pos_x, pos_y, pos_z, xs, ys, zs = get_3d(min_dim, pos_x, pos_y,
    pos_z, -1, 0, 0, 0, 1, 0, 0, 0, -1, xs, ys, zs, 0, sites)
    coordx, coordy, coordz = add_coordinates(coordx, coordy, coordz,
     xs, ys, zs)
```

Listing 3: The code of the general box of dimension $2^{n_x} \times 2^{n_y} \times 2^{n_z}$.

References

[1] Zhidong Zhang. “Exact solution of two-dimensional (2D) Ising model with a transverse field: A low-dimensional quantum spin system”. In: *ScienceDirect* (2021). DOI: https://doi.org/10.1016/j.physe.2021.114632.

[2] Ádám Bácsi and Balázs Dóra. “Kibble-Zurek scaling due to environment temperature quench in the transverse field Ising model”. In: *arXiv* (2022). DOI: https://doi.org/10.48550/arXiv.2203.04029.

[3] István A. Kovács. “Quantum multicritical point in the two- and three-dimensional random transverse-field Ising model”. In: *arXiv* (2021). DOI: https://doi.org/10.48550/arXiv.2111.06828.

[4] Jan Alexander Koziol et al. “Quantum-critical properties of the long-range transverse-field Ising model from quantum Monte Carlo simulations”. In: *APS* (2021). DOI: https://doi.org/10.1103/PhysRevB.103.245135.

[5] Alessio Franchi, Andrea Pelissetto, and Ettore Vicari. “Quantum critical behaviors and decoherence of weakly coupled quantum Ising models within an isolated global system”. In: *arXiv* (2022). DOI: https://doi.org/10.48550/arXiv.2209.06523.

[6] B.-W. Li et al. “Probing critical behavior of long-range transverse-field Ising model through quantum Kibble-Zurek mechanism”. In: *arXiv* (2022). DOI: https://doi.org/10.48550/arXiv.2208.03060.

[7] Bobin Li. “Decoherence Effect of Qubits in 1D Transverse Ising Model”. In: *arXiv* (2021). DOI: https://doi.org/10.48550/arXiv.2112.10345.

[8] Kai Zapp and Roman Orus. “Tensor network simulation of QED on infinite lattices: learning from (1+1)d, and prospects for (2+1)d”. In: *APS* (2017). DOI: https://doi.org/10.1103/PhysRevD.95.114508.

[9] Henrik R. Larsson. “Computing vibrational eigenstates with tree tensor network states (TTNS)”. In: *The Journal of Chemical Physics* (2019). DOI: https://doi.org/10.1063/1.5130390.

[10] Mattia Moroder et al. “Metallicity in the Dissipative Hubbard-Holstein Model: Markovian and Non-Markovian Tensor-Network Methods for Open Quantum Many-Body Systems”. In: *arXiv* (2022). DOI: https://doi.org/10.48550/arXiv.2207.08243.

[11] Alexander Keller, Carsten Wächter, and Nikolaus Binder. “Rendering along the Hilbert Curve”. In: *arXiv* (2022). DOI: https://doi.org/10.48550/arXiv.2207.05415.

[12] Qiao Wang et al. "PHoToNs–A Parallel Heterogeneous Threads oriented code for cosmological N-body simulation". In: *IOPScience* (2018). DOI: https://doi.org/10.1088/1674-4527/18/6/62.

[13] E. Estevez-Rams et al. "Visualizing long vectors of measurements by use of the Hilbert curve". In: *arXiv* (2015). DOI: https://doi.org/10.1016/j.cpc.2015.08.019.

[14] Simone Montangero. *Introduction to Tensor Network Methods*. Springer, 2018. URL: https://doi.org/10.1007/978-3-030-01409-4.

[15] Pietro Silvi et al. "The Tensor Networks Anthology: Simulation techniques for many-body quantum lattice systems". In: *SciPost* (2019). DOI: https://doi.org/10.21468/SciPostPhysLectNotes.8.

[16] M. Gerster et al. "Unconstrained tree tensor network: An adaptive gauge picture for enhanced performance". In: *APS* (2014). DOI: https://doi.org/10.1103/PhysRevB.90.125154.

[17] Giovanni Cataldi et al. "Hilbert curve vs Hilbert space: exploiting fractal 2D covering to increase tensor network efficiency". In: *Quantum Journal* (2021). DOI: https://doi.org/10.22331/q-2021-09-29-556.

[18] Cécilia Lancien and David Pérez-García. "Correlation length in random MPS and PEPS". In: *Springer* (2019). DOI: https://doi.org/10.1007/s00023-021-01087-4.

[19] Ian MacCormack, Alexey Galda, and Adam L. Lyon. "Simulating Large PEPs Tensor Networks on Small Quantum Devices". In: *arXiv* (2021). DOI: https://doi.org/10.48550/arXiv.2110.00507.

[20] Herman Haverkort. "An inventory of three-dimensional Hilbert space-filling curves". In: *arXiv* (2016). DOI: https://doi.org/10.48550/arXiv.1109.2323.

[21] Timo Felser, Simone Notarnicola, and Simone Montangero. "Efficient tensor network ansatz for highdimensional quantum many-body problems". In: *APS* (2021). DOI: https://doi.org/10.1103/PhysRevLett.126.170603.

[22] *Quantum TEA: Quantum Tensor-network Emulator Applications*. Last visited August 29th 2022. URL: https://baltig.infn.it/quantum_tea.

[23] Sheng-Hao Li and Guo-Ping Lei. "Quantum phase transition in a two-dimensional quantum Ising model: Tensor network states and ground-state fidelity". In: *IOPscience* (2018). DOI: https://doi.org/10.1088/1742-6596/1087/5/052011.