Framework for universal NMR quantum computing using Heisenberg spin interaction

Master's thesis by
Maximilian Schwetz [1]
Work Group Many-Body-Physics [2]
Phillips-Universität Marburg

August 6, 2020

[1]maximilian.schwetz@physik.uni-marburg.de
[2]supervisor: Prof. Dr. R. M. Noack

Abstract

Quantum computing using the control techniques of nuclear magnetic resonance (NMR) has been one of the first experimental implementations of quantum information processing. By rotating nuclear spins inside molecules with magnetic fields, it is possible to implement any unitary operation on a set of spin-1/2-qubits. Since published work has so far been limited to the Ising spin interaction, this thesis extends the framework of NMR quantum computing to the Heisenberg interaction. In order to find NMR pulse sequences that represent quantum gates in the machine language of NMR quantum computing, magnetic and radio-frequency fields, an algorithm was implemented to examine billions of possible sequences for a universal set of quantum gates. The Python program was optimized to avoid the numerically most expensive calculations so that sequences up to nine pulses could be investigated. The search yielded an NMR pulse sequence for the anisotropic Heisenberg interaction that implements the CNOT quantum gate on an arbitrary input state. However, no such sequence was found for the isotropic Heisenberg interaction for a sequence of up to nine pulses in length and while restricting the single-qubit rotations to a finite set of rotation angles. The framework of NMR quantum computing can thus be extended to the Heisenberg interaction although it is not clear if universal quantum computing is possible using only the isotropic Heisenberg interaction to entangle two qubits.

German Abstract / Deutsche Zusammenfassung

Quantencomputer, die auf Methoden der Kernspinresonanz (NMR) basieren, bildeten die ersten experimentellen Umsetzungen der Quanteninformationstheorie. Durch Drehungen der Kernspins in Molekülen, die durch magnetische Felder verursacht werden, kann man jede mögliche unitäre Operation auf Qubits mit Spin 1/2 ausführen. Da sich bisherige Veröffentlichungen auf die Ising Wechselwirkung zwischen Spins konzentrieren, wird die Theorie von NMR Quantencomputern in dieser Arbeit auf die Heisenberg Wechselwirkung übertragen. Um Pulsfolgen für Quantengatter in der Maschinensprache der NMR Quantencomputer (magnetische Felder) zu finden, wurde eine Methode entwickelt, die Milliarden von möglichen Pulssequenzen auf universelle Sätze von Quantengattern überprüft. Dieses Python Programm wurde optimiert um die kostspieligsten Berechnungen zu vermeiden und somit Folgen bis zu einer Länge von neun Pulsen zu untersuchen. Es wurde eine Pulsfolge gefunden, die das CNOT Quantengatter mit der anisotropen Heisenberg Wechselwirkung implementiert. Bis zur Länge von neun Pulsen und unter der Annahme, dass nur einige diskrete Rotationswinkel möglich sind, wurde jedoch keine Pulsfolge für die isotrope Heisenberg Wechselwirkung gefunden. Es ist also möglich, das Konzept von NMR Quantencomputern auf die Heisenberg Wechselwirkung zu erweitern. Jedoch ist weiterhin unklar, ob man einen universellen Satz von Quantengattern auch im isotropen Falle implementieren kann.

Contents

1. Introduction

Quantum computing has the potential to significantly impact the capability of computers. Should scientists succeed in building a sufficiently large quantum computer, it will be possible to solve problems in a few hours that, using classical computers, would have taken longer than the lifetime of the universe. Supremacy over classical computers for some problems, once achieved, will potentially extend to many problems in computer science, from fields like cryptography to machine-learning methods in medical research. Having such a powerful device has the potential to change our lives, just as the development of the first computers in the middle of the 20th century has.

A quantum computer can be implemented using many different physical systems. Possible architectures range from trapped ions all the way to topological qubits. One of these architectures, nuclear magnetic resonance (NMR) quantum computing, has, arguably, undergone the fastest development of all implementations in the early days of experimental quantum computing. The earliest implemented quantum algorithms of relevant size have all been implemented in NMR experiments.

NMR quantum computing is set apart from other quantum computing implementations by its power to perform coherent quantum computations at room temperature, a feature almost all other methods lack. On the contrary, quantum computing usually involves isolating individual quantum-mechanical particles, which is extremely difficult and energy-intensive. The methods of NMR have been used in material science and medicine for over half a century. While NMR has long been used as a method for imaging substances, researchers have, over time, developed NMR from a read-only sensing method to a method that can read, write, and process information. Since the methods of manipulating nuclear spins have long been highly developed, only the ability to implement coherent quantum states on a sample at room temperature had to be added to make NMR quantum computing possible.

1.1. Historical development of NMR quantum computing

While magnetic interaction with the nuclei of atoms was first described in 1938 [1], techniques to actively manipulate and detect nuclear magnetic moments were developed only about ten years later [2]. This led to the first NMR spectrometers being available in the early 1950s, which were primarily applied to sensing of atoms and molecules in chemistry and biology. In the 1970s, combined with computer imaging technology in order to resolve human tissue structure [3]. This has led to the most commonly known application of NMR in medical science, which today is known as Magnetic Resonance Imaging (MRI).

In 1980, Paul Benioff proved that computing on quantum-mechanical particles is theoretically possible and that it can be at least as powerful as a Turing machine [4]. The Turing machine is a universal mathematical model of a computer; all modern computers are mathematically equivalent to a Turing machine. Two years later Feynman *et al.* [5] realized that information processing and computing on quantum bits (qubits) might be more powerful than classical computing in some cases. In particular, some algorithmic problems that are in the classical non-polynomial-time complexity class fall into a polynomial-time quantum complexity class [6]. This means that some computational problems, like integer factoring [7], experience an exponential speedup in going from classical to quantum computers. The proposal of quantum supremacy led to many quantum algorithms being developed for potential quantum computers in the late 20th century. These algorithms all have the distinct feature of solving problems with fundamentally fewer operations than classical algorithms would need.

In the last decade of the 20th century, the field of NMR-spectroscopy and the field of quantum computing were brought together. In 1993, Seth Lloyd proposed a quantum computer that would use manipulative techniques formerly utilized in NMR imaging to change the nuclear spins inside molecules [8]. If these nuclei were used to represent qubits, one could execute quantum computations on them. This includes methods such as creating superpositions of quantum states, which go beyond the realm of classical information processing.

Lloyd proposed to use a strong magnetic field to split the spin-energy levels of a spin-1/2-particle into two distinct levels, thus implementing a qubit [8]. The magnetic field also causes a precession of the spins magnetic moment. When another magnetic field, rotating at the precession frequency, is applied, the spin can be rotated along axes in the rotating frame of reference other than the initial precession axis. This secondary, manipulative magnetic field oscillates at radio frequency. Using such these techniques, one can implement any rotation of a particular spin.

A sketch of the sample of an NMR experiment is shown in Fig. 1.1a. Simply speaking, each of the molecules is a separate small quantum computer in which the nuclear spins play the role of qubits. The nuclear spins inside the sample must be subjected to a strong homogeneous magnetic field in order to split up the two states. In addition to the homogeneous magnetic field, radio-frequency electromagnetic fields are applied in the transverse direction to manipulate the spins in the sample. A sketch of such an experimental setup is shown in Fig. 1.1b.

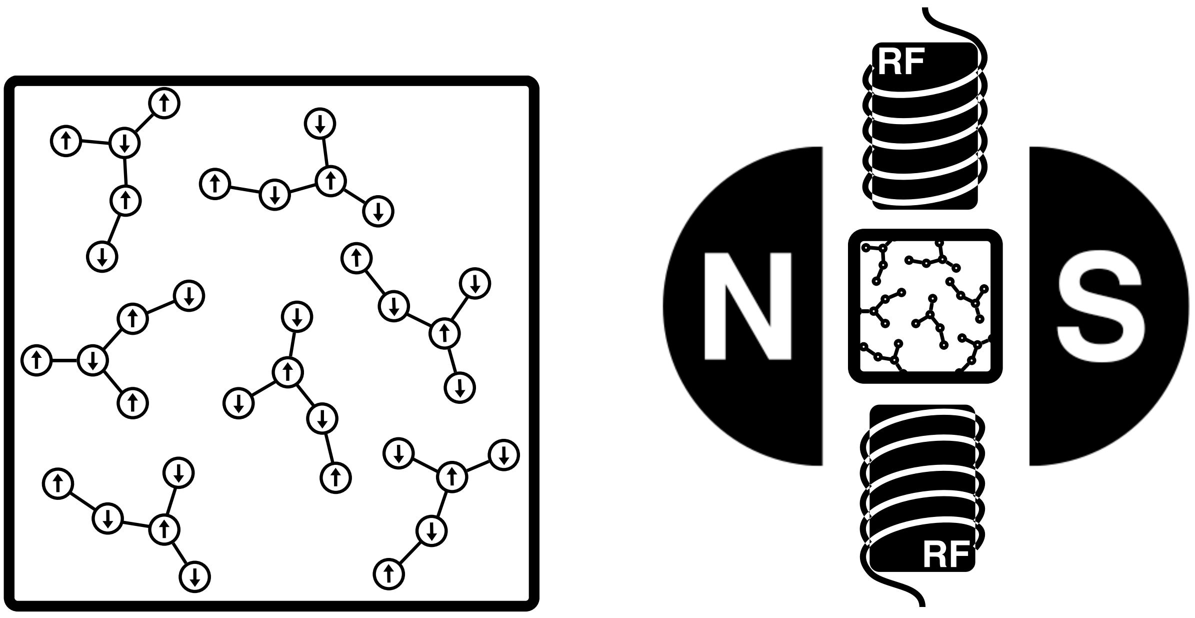


Figure 1.1.: (a) Schematic sample for an NMR quantum computer. Nuclear spins of the atoms inside each molecule act as qubits. All molecules are simultaneously manipulated. (b) Sketch of an NMR quantum computer. A permanent magnet produces a constant homogeneous horizontal field to enforce Zeeman splitting and precession. An electromagnetic radio-frequency field is applied perpendicular to the strong field to manipulate individual spins inside of each molecule via pulses of set length.

1.2. Choosing a spin-interaction model

In published theoretical and experimental work in the field of NMR quantum computing, the interaction between spins in the molecules is simplified. The interaction is essential because, without it, we would not be able to generate entanglement between qubits, a necessary property for universal quantum computing. Recent work treats the interaction between the spins as an Ising interaction. This assumption is reasonable because the J-coupling, which has the form of the Ising interaction, is the dominant type of spin-spin interaction in NMR molecules [9]. The Ising model is a (semi-)classical model that attributes two distinct and discrete possible states to the spins, i.e., up and down and the interaction assigns different energies if the spins are aligned or anti-aligned. Within this simplified regime, it is possible to implement a universal set of quantum gates using a sequence NMR control pulses. This means that it is theoretically possible to apply an arbitrary unitary operator to quantum states using only pairwise Ising interactions between the spins in addition to single-qubit rotations.

The relatively simple Ising model is often suited to model the macroscopic properties of spin systems. However, the classical nature of the Ising model cannot represent all properties of a quantum spin system. As an alternative, the quantum-mechanical Heisenberg model can be a more realistic representation of spin interaction between spin-1/2 quantum spins. The Heisenberg model represents the spins as three-dimensional vectors of quantum-mechanical operators in contrast to the classical interaction of the Ising model. The interaction of spin magnetic moments is then calculated as the vector product of these spin-magnetic moments in the Hamiltonian.

The main difference between the Ising model and the classical Heisenberg model is that the Ising model is symmetric to a reflection of all spins while the Heisenberg model is invariant under simultaneous rotation of all spins. The latter is thus more suited to describe physical systems with just that symmetry property. On the other hand, the Ising model is suited to handle systems, where the interaction is strongly anisotropic. In the quantum-mechanical case, the Heisenberg interaction represents spin components using non-commuting quantum-mechanical spin operators. On the other hand, the Ising model does not have any non-commuting operators because it only consists of one component of the spin.

The quantum versions of the Ising and Heisenberg interactions may lead to different ways of implementing quantum gates via NMR pulse sequences. This includes the order, number, and type of pulses that we need to apply to realize a certain operation. It may be possible to execute transitions between quantum states faster or with fewer pulses with the Heisenberg interaction rather than with the Ising interaction. The opposite is also possible. Another interesting question is: Is the Heisenberg model capable of universal quantum computing at all? After all, the capability to implement all unitary operations using the Ising interaction might arise from its simplifying assumptions, which may not be valid for certain physical systems.

In real NMR experiments, the interaction can usually be approximated by the Ising interaction. This is the case in the weak-coupling regime, where the indirect interaction through chemical bonds is much stronger than the direct dipole-dipole interaction. However, if the spins are strongly coupled, the Heisenberg interaction can be a better description of the physical interaction. In order to fully understand quantum computing with NMR control techniques, we need to know what the consequences of the choice of the coupling regime, and thus of the spin model, are. Since no publication known to the author of this thesis has yet tried to implement NMR quantum computing using the Heisenberg interaction, we may be able to gain insight into general concepts of experimental quantum computing by treating a different spin interaction model. The concepts may be transferable to other quantum computing architectures than NMR.

1.3. Finding a protocol for quantum computing in the Heisenberg model

In order to advance towards a formalism of quantum computing with Heisenberg-NMR methods, we will start by introducing the underlying theoretical concepts. They will then lead us to new methods of propagating spin quantum states and an extensive search of NMR-pulse-sequences. Finally, we will discuss the findings of this search and give an outlook to future work.

In particular, in Chap. 2, we will review the basic features of the Ising model and the Heisenberg model. The different symmetry properties of the models shall be outlined in particular. We will then give a short summary of quantum information processing and why quantum computing can potentially outperform classical computing. The difference between universal sets of gates for classical and quantum computing as well as the scaling of different problems will be discussed. We will then introduce the basic concepts of nuclear magnetic resonance, focusing on the aspects relevant for quantum computing. We will observe that we can not only detect the values of observables with NMR techniques, but also control qubits.

Chap. 3 will then connect the three subjects spin of interaction models, quantum computing, and NMR to form a theory for carrying out quantum computations on spins using NMR control techniques. We will first investigate these concepts in the scope of the Ising model, which is also used in recent published literature. Subsequently, we will develop a similar formalism for the Heisenberg interaction. We will cover both the isotropic and the anisotropic Heisenberg interactions. In particular, we will search for NMR pulse sequences that implement quantum gates when applied to spin qubits.

In Chap. 4, we shall investigate the methods used to visualize and compute the results of the earlier sections, covering the Python modules used for visualization as well as containing an explanation of the program that is used to find NMR pulse sequences in the Heisenberg model. The fundamental algorithm is a search, which grows to be very expensive in time and space for larger problems In order to reduce the computational cost, several optimizations and shortcuts are presented so that it is possible to search a larger problem space.

Further potential optimization to the program are mentioned in Chap. 5. After improvements on the existing program are discussed, the general method of solving a problem of this kind, *quantum optimal control*, is presented. Additionally, some problems that occur in experimental NMR quantum computing are briefly introduced. Finally, a general discussion about how the theoretical ideas of the previous sections can be further developed is included. Here we will revisit the issue of to symmetry considerations and discuss whether or not certain operations can be implemented at all using the Heisenberg interaction.

2. Theoretical Framework for NMR quantum computing

In this thesis I will assume that the reader has a good working knowledge of advanced quantum mechanics. Building on that, this section will review how spin interaction is treated within quantum mechanics. We will derive the two most common models for spin interaction and investigate their symmetry properties and their relation to each other. For a more elaborate introduction to this subject, textbooks such as Ref. [10] are recommended.

We will then give a reasonably short introduction to quantum information theory and quantum computing. We assume that the reader is already familiar with the basic concepts of quantum computing, as this thesis skips most of the derivations and only summarizes the most important statements. For a more thorough treatment of this subject, the reader is referred to standard textbooks like Ref. [6].

The last part in this chapter will give an introduction to the idea of nuclear magnetic resonance (NMR) in general. The introduction to NMR will be more elementary than the other sections. It will be a motivation for the main subject of implementing a quantum computer on a spin system.

In Chap. 3, we will then combine all three of these concepts. By the means of NMR, we will use a spin system to implement a theoretical quantum computer. This computer will be treated with two different forms of spin interactions.

2.1. Interaction of spin-1/2 particles

In this thesis, the interaction of quantum spins will be a key element. We will now introduce different models to describe the interaction between spins. These models are utilized mainly to describe lattices of spins. In these lattices, interaction is often restricted to nearest neighbors because the spin-spin interaction is very weak in magnitude. However, the models can also be extended to cover longer-range interactions.

Since the models each obey different symmetries, they are in terms suited for different problems that also vary in their symmetry properties. Namely, we will cover the Ising and the Heisenberg models of spin interaction. The former obeys discrete symmetry, while the latter is invariant under a continuous transformation. A third model, which lies in between the two, will be briefly described.

In Chap. 3 we will characterize computational sequences on nuclear spins in the Ising and the Heisenberg model. We will see, that the models yield different results. Changing between the models means that we also have to apply different nuclear magnetic resonance methods to implement quantum gates. This can partly be explained by their symmetry properties.

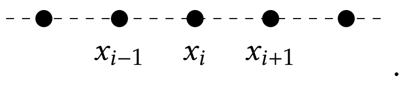
We consider a general lattice as the generalization of a spin cluster. Here, we illustrate the lattice as one-dimensional. In general, it may possess any number of dimensions and may also be finite or infinite in size. If the lattice is finite, we can apply open or periodic boundary conditions as will fit the physical system under investigation.

In the main part of this thesis, we will treat a one-dimensional spin chain. Hence, all content here will concentrate on the one-dimensional case. Higher-dimensional lattices are generally equivalent in their mathematical treatment. They may, however, display new effects and properties.

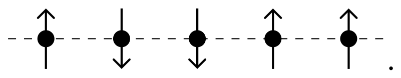
In addition to the aforementioned simplifications, we will restrict this introduction to particles with spin $S = \frac{1}{2}$ as this will be the focus of the algorithms in later chapters. It is possible to extend these models to higher spins.

2.1.1. Ising model

The positions of the lattice sites are labeled with x_i, as depicted in the following sketch:



In the Ising model, the spin S_i at each site x_i can have two possible values. Each spin is thus discrete. The values of the spins are written as $S_i \in \{-1/2, +1/2\}$, where we set $\hbar = 1$. In the sketches we will depict it as $\uparrow$ and $\downarrow$ because this makes the connection to the Heisenberg model simpler. We can identify the two states as the eigenstates of the z-component of the spin operator S^z without loss of generality. An example of a lattice in this notation may look like



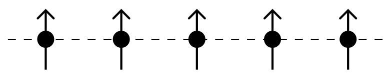
This means that we only allow the spins to have discrete values. Hence, the Ising model treats the spins only (semi-)classically.

We will here restrict ourselves to nearest-neighbor interaction. The Hamiltonian will then consist of the sum of all pairs of spins that are nearest neighbors. We assume for our purposes that the interaction strength J is the same between all lattice pairs, i.e., $J_{ij} = J$. Then the Hamiltonian reads

$$\hat{H}_{\text{Ising}} = J \sum_{\langle i,j \rangle} S_i S_j \,, \tag{2.1.1}$$

where $\langle i, j \rangle$ denotes nearest-neighbor pairs at sites $(x_i, x_j) = (x_i, x_{i+1})$.

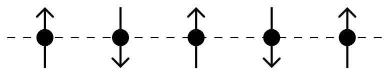
For the ground state of this system, we have to distinguish two cases. For the ferromagnetic case, $J < 0$, the energy is minimized when all spins point in the same direction. The spin distribution of the lattice then looks like


.

The axis, along which all the spins point is irrelevant because the system is invariant with respect to spin reflection. The total magnetization is thus

$$M = \left| \frac{1}{N} \sum_i S_i \right| = \frac{1}{2} \,. \tag{2.1.2}$$

In the antiferromagnetic case, $J > 0$, the energy is minimized when each of the terms in the sum of Eq. (2.1.1) is negative. Hence, the two spins of each pair have to point in opposite directions. The ground state looks like


,

where the total magnetization amounts to

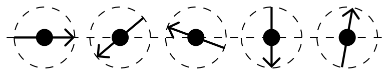
$$M = \frac{1}{N} \sum_i S_i = 0 \,. \tag{2.1.3}$$

To sum up, the Ising model is a simplified approach to spin lattices. It is invariant under reflection and treats the quantum-mechanical spin in a (semi-)classical way.

2.1.2. Isotropic Heisenberg model

We will now introduce a model, that treats spin in a more quantum-mechanical manner. The key difference from the Ising model will be advancing from a discrete symmetry to a continuous symmetry. Additionally, spin operators do not commute anymore, i.e., $[\hat{S}_i, \hat{S}_j] = i\hbar\epsilon_{ijk}\hat{S}_k$. These attributes not only change the ground state of the respective Hamiltonian, but also lead to significantly different behavior for certain lattices.

In the Heisenberg model, each spin is identified as a vector of operators $\hat{\vec{S}}$ with three spatial components. These three components correspond to three spatial dimensions in Euclidian space. Hence, we can identify the spin as a magnetic moment, which has direction $\hat{\vec{S}}/|\hat{\vec{S}}|$ and magnitude $|\hat{\vec{S}}|$. In general, a lattice in the Heisenberg model may look like

 ,

where the circles in this sketch represent spheres in three dimensions.

In any basis, one spin can be expressed as a two-component vector, representing the projection of the spin onto a particular basis. For example, in the S^z basis, some representative spinors are

$$|\uparrow\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \qquad |\downarrow\rangle = \begin{pmatrix}0\\1\end{pmatrix} \qquad \text{, and} \qquad |\Psi\rangle = \langle z|\uparrow_x\rangle\,|z\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}. \tag{2.1.4}$$

They describe the vectors with a spin projection of 1/2 and −1/2 in the +z-direction and the superposition of these two states. The last vector also is the vector with spin 1/2 in the S_x-basis.

Since the transformation of a normalized state has to result in another normalized state, the corresponding transformation matrix must be unitary and must have a determinant of unity according to the basic axioms of quantum mechanics. Operators thus need to be in the special unitary group

$$\mathrm{SU}(2) = \left\{U \in \mathbb{C}^{2\times 2} :\ U^\dagger U = \mathbb{1},\ \det U = 1\right\}. \tag{2.1.5}$$

We can now define the *commutator*

$$[.,.]\ :\ \mathrm{SU}(2)\times\mathrm{SU}(2) \to \mathrm{SU}(2)\ : [P,Q] \to PQ - QP\,. \tag{2.1.6}$$

The commutator vanishes if the two matrices P and Q commute with one another. The Lie group SU(2) and the commutator together form the Lie algebra $\mathfrak{su}(2)$. The generators of this Lie algebra are the Pauli matrices

$$\sigma_x = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix},\ \sigma_y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix} \text{ and } \sigma_z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}. \tag{2.1.7}$$

We can check that they obey the expected commutation relations

$$[\sigma_i, \sigma_j] = 2\epsilon_{ijk}\sigma_k\,. \tag{2.1.8}$$

Using *Cornwell's Theorem*, we can show that there exists a 2-to-1-homomorphism from $\mathfrak{su}(2)$ to SO(3), the group of real orthogonal matrices with the same algebra [11]. A possible mapping m from $U \in \mathrm{SU}(2)$ to $m(U) \in \mathrm{SO}(3)$ with $m(U) = m(-U)$ may be given as

$$m(U)_{ij} = \frac{1}{2}\,\mathrm{Tr}\left(\sigma_i U \sigma_j U^\dagger\right). \tag{2.1.9}$$

Through this homomorphism, we can identify the operations of $\mathfrak{su}(2)$ on the spin states (spinors) of Eq. (2.1.4) with operators of SO(3). Hence, we may interpret the Pauli matrices as generators of rotations around the three spatial axes. A finite rotation of a spinor around an axis $\hat{\phi} = \vec{\phi}/\phi$ by an angle ϕ can then be calculated as

$$R(\vec{\phi}) = \exp\left(\mathrm{i}\,\vec{\phi}\cdot\vec{\sigma}\right), \tag{2.1.10}$$

where $\vec{\sigma} = \hat{e}_x\sigma_x + \hat{e}_y\sigma_y + \hat{e}_z\sigma_z$ is the vector of Pauli matrices and $\hat{e}_i$ are the usual unit basis vectors in three dimensions.

Now that we have seen that the Pauli matrices are a good choice to represent the Lie algebra $\mathfrak{su}(2)$ of spinor operations, it is natural to define the interaction Hamiltonian of two spins as

$$\hat{H}_{\text{Heisenb.}} = \sum_{\langle ij\rangle} J_{ij}\hat{\vec{S}}_i\cdot\hat{\vec{S}}_j\,, \tag{2.1.11}$$

where the spin operators are defined as $\vec{S}_j = \frac{1}{2}\vec{\sigma}_j$ in the respective sub-Hilbert-space of the j-th spin. We have again set $\hbar = 1$. If

$$\mathscr{H} = \bigotimes_i \mathscr{H}_i \tag{2.1.12}$$

is the total Hilbert space of the spin lattice with $\mathscr{H}_i$ the subspaces of the j-th spin, then the operator $\vec{S}_j$ in the full Hilbert space is the tensor product

$$\vec{S}_j = \frac{\hbar}{2}\vec{\sigma}\bigotimes_{i\neq j}\mathbb{1}_{\mathscr{H}_i}\,. \tag{2.1.13}$$

As an example, we can give the interaction Hamiltonian for two spin 1/2 particles with isotropic interaction strength $J_{ij} \equiv J$:

$$\hat{H} = J\hat{\vec{S}}_{\text{A}}\hat{\vec{S}}_{\text{B}} = J\left(\hat{S}_{x,\text{A}}\hat{S}_{x,\text{B}} + \hat{S}_{y,\text{A}}\hat{S}_{y,\text{B}} + \hat{S}_{z,\text{A}}\hat{S}_{z,\text{B}}\right) = \frac{J}{4}\begin{pmatrix}1 & & & \\ & -1 & 2 & \\ & 2 & -1 & \\ & & & 1\end{pmatrix}, \tag{2.1.14}$$

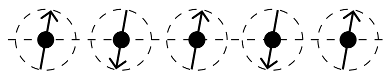
where we have used that $\hat{S}_{x,\text{A}} = \frac{\hbar}{2}\sigma_x\otimes\mathbb{1}_2$. With the spin ladder operators $\hat{S}^{\pm} = (\hat{S}_x \pm \mathrm{i}\,\hat{S}_y)/\sqrt{2}$ we can also rewrite the Hamiltonian as

$$\hat{H} = J\left(\frac{1}{2}(\hat{S}^+_{\text{A}}\hat{S}^-_{\text{B}} + \hat{S}^-_{\text{A}}\hat{S}^+_{\text{B}}) + \hat{S}_{z,\text{A}}\hat{S}_{z,\text{B}}\right) \tag{2.1.15}$$

and see the matrix structure of Eq. (2.1.14) more easily.

The ground state of the Heisenberg Hamiltonian is split into two regimes. Just as for the Ising Hamiltonian, we have to distinguish between the ferromagnetic ($J < 0$)

and the antiferromagnetic ($J > 0$) interaction. As for the Ising case, the ferromagnetic ground state is totally parallel and the antiferromagnetic ground state is totally anti-parallel. The difference is that the spins can point in any direction in space as long as they all remain parallel or anti-parallel. The antiferromagnetic ground state may look like

 .

2.1.3. Anisotropic Heisenberg model

Up to now, we have assumed the Heisenberg interaction to be isotropic, i.e., that the interaction parameters $J_{ij}^{\alpha} \equiv J$, ($\alpha \in \{x, y, z\}$), where i and j are nearest neighbors. Here we consider two ways of easing that assumption. First we can have a spatial distribution of the interaction strength. This may be the case if the spins are not equally spaced inside a lattice. Then, in general, $J_{ij} \neq J_{kl}$, where ij and kl denote different nearest neighbors.

The second generalization would be to have two spins that do not interact with the same strength in all spatial directions of the spin. We could have have a direction in which spins interact more strongly. The case of anisotropy in one interaction parameter is termed the *Heisenberg XXZ model* and its Hamiltonian is

$$\hat{H}_{\mathrm{XXZ}} = \sum_{ij} J\left(\Delta \hat{S}_i^z \hat{S}_j^z + \hat{S}_i^x \hat{S}_j^x + \hat{S}_i^y \hat{S}_j^y\right) = \sum_{ij} J\left(\Delta \hat{S}_i^z \hat{S}_j^z + \frac{1}{2}(\hat{S}_{\mathrm{i}}^+ \hat{S}_{\mathrm{j}}^- + \hat{S}_{\mathrm{i}}^- \hat{S}_{\mathrm{j}}^+)\right), \qquad (2.1.16)$$

with Δ the anisotropic factor.

When Δ gets large, the limit of this model is the Ising model. Naturally, we need to introduce some normalization $J \longrightarrow J/\Delta$ in order to keep the system at finite energies in the limit. For the Ising model, it does not matter which one of the spatial components is taken to be dominant because it does effectively treat the spin as a (classical) bit without direction. This is only true for the interaction term. Other terms than spin-spin interaction may destroy the reflection symmetry of the Hamiltonian in the Ising model.

When Δ becomes small, we arrive at the *XY model*. In this model, the spin is characterized by a two dimensional vector. These two degrees of freedom are usually in a plane perpendicular to the axis that connects two lattice sites. We will not treat the XZ model in this thesis because it is a rather special model only usable for very specific problems.

In general, the interaction strength can vary for all spatial directions, i.e., $J^x \neq J^y \neq J^z$. The general Heisenberg Hamiltonian for the XYZ chain then reads

$$\hat{H}_{\mathrm{XYZ}} = \sum_{ij} \left(J_{ij}^x \hat{S}_i^x \hat{S}_j^x + J_{ij}^y \hat{S}_i^y \hat{S}_j^y + J_{ij}^z \hat{S}_i^z \hat{S}_j^z\right). \qquad (2.1.17)$$

In this thesis, we will not cover the most general case of XYZ interaction and will stick to the XXZ model as the most general assumption because we will find a property of XXZ model and do not have to generalize further as a consequence.

2.1.4. Differences

We have seen that the Ising model and the XY model are both limits of the Heisenberg model. In the same way, the Heisenberg model is only a simplification of real spin interaction.

Finally, we will discuss some differences between the Ising and the Heisenberg model that arise from the different symmetries that each obeys. Recall that the Ising model is invariant under reflection, while the Heisenberg model is invariant under rotations.

Since the symmetry for the Heisenberg model is continuous, we will also obtain continuous excitations. That is, in general, we can excite the system by an arbitrarily small energy $\mathrm{d}E$. There can, of course, still be macroscopic band gaps in the systems energy distribution. On the other hand, in the Ising model, we only have discrete states and thus there is a lowest excitation energy δE with $|\delta E| > 0$.

A consequence of the last point is that a Heisenberg lattice does not show spontaneous breaking of symmetry in one or two dimensions at finite temperature. This follows from the *Mermin-Wagner theorem* [12]. This means that in one or two dimensional lattices, there cannot be a phase transition from a ferromagnetic to a paramagnetic phase. The latter has an unordered distribution of spin directions and hence vanishing magnetization. The Mermin-Wagner theorem does not apply to the Ising model because it does not have a continuous symmetry. The ferromagnetic-paramagnetic phase transition at high temperature is thus allowed even for less than three dimensional lattices.

As we take a look at the structure of the Hamiltonian of the Ising model (Eq. (2.1.1)), we notice that its matrix is diagonal. Because of its diagonal shape, the eigenstates in the S^z basis will always be simple. More precisely, they will always be integers multiplied by the interaction strength J. These facts make the Ising model a very convenient model to treat mathematically.

On the other hand, the Hamiltonian of the Heisenberg model is clearly not diagonal in the S^z basis. Also, the terms $\hat{S}_{x,y,z}$ that contribute to the Hamiltonian do not commute with each other. The last fact leads to eigenvalues that are not simple, i.e., not evenly spaced. Since the Hamiltonian contains one of each generator of the $\mathfrak{su}(2)$ Lie algebra, any other operator with one of these generators will not commute with the Hamiltonian because of the commutation relation in Eq. (2.1.8). This non-commuting property will be of relevance in Chap. 3, when we try to simplify pulse sequences by permuting pulses. These pulses will be exponentials of spin operators $\hat{S}_{x,y,z}$.

In summary, the Ising model is well-suited to model systems that show a highly preferable axis of spin orientation. This is true for systems that are invariant under reflection. In this sense, one can take the Ising model to be a classical limit of the Heisenberg model. The latter is well suited for systems that show rotational invariance. As the reflection symmetry is included in rotational invariance, the Heisenberg model does encompass the Ising-like problems. The Heisenberg model is thus a generalization of the Ising model. The additional degrees of freedom for the spin also lead to other physical properties, as we have seen in this section.

2.2. Summary of Quantum Information Theory

After having discussed the quantum mechanical interaction of spins, we will now introduce one of the many areas of application in which quantum-mechanical spins can be put to use. We will examine ways to represent information in physical systems.

First, we will treat information in a classical way. These are the foundations of computer science. Then, however, we will change gears and introduce quantum mechanical systems as carriers of information. The resulting quantum information theory will lead to a way of computing that is significantly more powerful than classical computing.

Although it has long been theoretically proven that quantum computing is more powerful on both the space and the time scale, as of the writing of this thesis, no real quantum computer has yet been built , that is "supreme" relative to all classical computers. There are, however, several development efforts that aim to achieve such a quantum supremacy. For some very specific problems, quantum supremacy has been demonstrated experimentally.

2.2.1. Classical information processing

If you have any kind of information, be it thoughts, numbers, text, audio, video or anything else, and want to communicate or store it, you will need some encoding to map this information onto physical states. For example, this could be text, written down as a layer of atoms, which reflect in a certain color or the sound of a voice that is carried by oscillations of the surrounding air.

It is essential for good information processing that this encoding will stay the same when we use an encoding over and over again. Coming back to the last example, it is necessary that the sound propagates through air roughly in the same way for all times and all locations. If sufficient consistency is present, reliable and fast procedures can be developed to work with that information. They can be standardized to the point that the main task for a user is the work with the information, not to encode it. An infinite amount of such encodings can be invented, but here we will only cover the classical bit encoding and, as an extension to that, the processing of information on quantum bits in the next subsection.

Classical bits are a digital or boolean encoding of information. A bit is a physical system that can be in one of two states. We can label these two states 0 and 1 or as the boolean values `False` and `True`. It is essential for information processing that the two states are always distinguishable and that we always have the same mapping of the boolean values to the intrinsic physical properties. This is the condition of consistency that we just discussed.

Physical manifestations of bits can be, for example, electrical currents or ferromagnetic solids as used in classical computers. Electrical currents can either be off or on, representing 0 and 1, while in a ferromagnetic solid, the magnetization in certain domains can be either up or down relative to some axis. Of course, there are many more possible implementations of physical bits, but they all share these same mapping properties.

A basic and easy to understand encoding for numbers is the classical *integer bit representation*. An integer k is represented by the bitstring $b_m b_{m-2} \ldots b_1 b_0$ as

$$k = b_0 + b_1 2 + b_2 2^2 + b_3 2^3 + \cdots = \sum_{i=0}^{\infty} b_i 2^i \,, \tag{2.2.1}$$

where $b_i \in \{0, 1\}$. For example, the integer $5 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ would be stored as 101 and needs three bits for its representation. Then, in a ferromagnet, the first and the third domain may have upwards magnetization, while the second one would point downwards. These types of sequences $b_m b_{m-2} \ldots b_1 b_0$ are also used to represent floating point numbers, strings and any other kind of data, albeit in a more elaborate encoding.

It is important to remember that, even if we write k as $k = 5$, then we still are working in a code. In this case, we represent the number in the *decimal* system rather than the *binary* representation of Eq. (2.2.1). From a mathematical point of view, there is no preferred encoding. However, some encodings might be more natural to certain physical systems as the binary system is to electronics. Also, of course, the decimal system has culturally developed as the dominant representation of numbers.

We can manipulate information by flipping the bits. For example, if we flip the third bit in the representation 101 of the integer 5 from above, we yield the representation 100 of the integer 4. This manipulation already was an example of a NOT gate on the third bit. *Gate* is just a name for a manipulative operation that may change the state of a physical system. A NOT gate simply flips a bit from 0 to 1 or from 1 to 0, depending on the initial state. More elaborate are two-bit operations. An example for a two-bit gate is the CNOT gate, which flips the second of two bits if and only if the first bit is in the 1 state. Mathematically speaking, the CNOT-gate is the XOR logic relation $\oplus$ of the

1st bit in	2nd bit in	1st bit out	2nd bit out
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 2.1.: Truth table of the CNOT-gate. The output of the second bit will be the XOR relation of both input bits. Another way of interpretation is that the second bit is flipped if and only if the first bit is 1.

two bits pasted onto the second bit. It can thus be written as

$$b_1 b_0 \xrightarrow{\text{CNOT}} b_1(\text{XOR}(b_1, b_2)) = b_1(b_0 \oplus b_1)\,, \tag{2.2.2}$$

where $\oplus$ denotes the sum modulo 2. The CNOT gate can also be represented as a truth table, which is shown in Table 2.1. Of course, there are many other operations and gates on bits. However, a special property of the CNOT gate is its *universality*. Universality means that one can implement every n-bit operation using a combination of 2-bit CNOT gates. The physical manifestation of the XOR operation is the transistor, which is the key element of all classical computers. A transistor will let an electric current pass if and only if there is a voltage applied to a third contact. One can even see the manifestation of the CNOT gate in the component itself.

It shall be mentioned here that in classical information theory, it is not the CNOT gate, but rather the XOR operation, that is implemented. The difference is that the XOR gate has two input gates and one output gate (the XOR of the inputs) and is thus not reversible. The full CNOT gate is reversible, though.

More elaborate algorithms can be composed of sequences of only CNOT gates because CNOT gates are universal. These Algorithms may range simple arithmetic such as addition all the way to highly sophisticated search methods on chaotic data sets.

The more difficult the problem, the longer an algorithm will usually take to solve it. A similar statement can also be made for the required space in physical memory. The crucial question is usually, how the time or space needed for a solution scales with the size of the system, for example, how much longer it takes to add two four-digit numbers instead of two two-digit numbers. It has always been an important task in computer science to find new algorithms that can solve problems faster than existing ones. This means finding algorithms that scale better with the length of the problems input. Typically, these algorithms are sorted into *complexity classes*, which classify how the algorithms scale with the problem size. An example of a complexity class might be P, that includes all problems which take polynomial time $t = P(n)$ to solve, where n is the size of the system and P is a polynomial.

On the other hand, there are also many problems that scale less quickly than polynomially. An example for a more difficult task would be the factoring problem of finding the prime factors of an integer. For an integer k, we want to find the unique representation

$$k = \prod_i p_i^{n_i} \tag{2.2.3}$$

for which the p_i are prime numbers and n_i are integers. For small integers up to a few digits, this factorization can still be done by hand. Larger integers can be factorized by a computer. However, if the size of the integer further increases, the amount of time needed to solve the problem would quickly rise to more than the age of the universe due to an exponential scaling of the required time to solve this problem, even with the best existing algorithms.

On the other hand, the factorization of an integer is easily verifiable. Given two or more prime factors, they only need to be multiplied in order to prove that they are indeed factors of the given integer, which can be done in a short time. Problems like the factorization problem lie in the nondeterministic polynomial time (NP) complexity class. Such problems are hard to solve, but solutions are easy to verify. Such tasks often used as a basis of cryptography. In the classical *public key* communication, a key is openly communicated to every possible sender of a secret message. This key may, for example, be the product of two primes. A sender then encrypts his or her secret message with a method that is seeded by the public key. Only the person that knows the prime factors will be able to decrypt this message. So if the number, that is used as the key, is large enough, no one will realistically be able to decrypt the message but the generator of the public key.

A wide variety of problems fall into NP. A subclass of these problems can be proven to be the hardest problems to solve within NP (NP-hard). It has not yet been proven, that classical algorithms exist that could solve those problems in polynomial time. If someone were to find a classical polynomial solver for any NP-hard problem, it has been proven that all problems that fall into NP could be solved in polynomial time. This is the famous "P=NP?" problem in computer science.

2.2.2. Quantum information processing

After realizing that there are problems that are hard to solve, one might ask what an approach for finding a better solver might look like. It turns out that there are already several algorithms that use randomness to find the solution of an NP-hard problem with high probability in polynomial time. Although the solution is not correct one hundred percent of the time, it is still easy to verify if the solution is correct or if the algorithm has to be run again.

However, there is another approach: adding quantum mechanics to the physical bits that were so far taken to be classical objects. This could, on the one hand, be done specifically by using very small and quantum mechanical particles as carriers of information. Approaching from a different angle, we observe the fact that physical memory for computing is getting smaller and smaller by the year. This means that, in the near future, we will reach a point where we will have to take quantum mechanical effects into account anyway. Thus, we will naturally need to include quantum mechanics in our models for computing. In the present day, the best and smallest electrical circuits already incorporate quantum effects.

Retaining the analogy to classical information theory, quantum bits (*qubits*) are a physical system that can be in one of two possible states. There are a variety of realizations of qubits, but here we focus on spin systems. If a particle with total spin 1/2 is under the influence of a magnetic field, the initially degenerate eigenenergies corresponding to spin up or down are split by a small margin (Zeeman splitting). The two levels then become distinguishable and thus suitable to represent a bit. This application of a magnetic field for Zeeman splitting will be revised again later in the context of nuclear magnetic resonance.

In contrast to classical mechanics, the properties of a quantum object are determined by its (quantum) state or its density operator. Only a measurement of an observable can retrieve information about the quantum object, but also collapses the state of the object. So the question is: does making the step to quantum mechanics make any difference in information processing? After all, we have thus far only introduced another two-state system acting as a bit.

A classical system of two bits can be in one of the four states, 00, 01, 10, or 11. A classical measurement will obtain the state that the system is in. A quantum system of two qubits, however, can be in any superposition

$$|\Psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \tag{2.2.4}$$

of the aforementioned four states, where $\alpha_i \in \mathbb{C}$. A measurement with respect to this basis will also only yield one of the four results of the classical system, each with probability $|\alpha_i|^2$. Since the measurement is projective, the system will end up in the state $|i\rangle$. As the possible measurement outcomes are the same as in the classical case, we can ask if we have gained anything?

Indeed, the results can only be discrete eigenvalues of the observable. However, the state before measurement can be any superposition of basis states. So, as long as we do not make a measurement, we have a larger space in which we can do computations. The Hilbert space of the quantum states has dimension 2^N, where N is the number of qubits. If we wanted to pinpoint where the dimensional advantage of quantum computing lies, we need to look at the amplitudes α_i of the states. We notice that these are complex valued. So, in addition to the mixing of the moduli of the amplitude, in the quantum case we can also have a relative phase between the components.

In classical computation, a gate can only act on one input state. A quantum gate, however, can act on a superposition of states and, as such, on all possible states at once. This is called *quantum parallelism*. Using this kind of parallel computing on one system, one can compute the manipulation of all possible input states simultaneously. The basic idea behind almost all quantum algorithms is to prepare an initial superposition state and then do a quantum parallel manipulation on it that represents the problem to be solved. Finally, one has to amplify the amplitude of the best solution so that a measurement will yield that result with high probability or with certainty.

To understand the current state of research in the field of quantum computing, we first need to take a look at the historical development. In 1980, Paul Benioff showed that computing on quantum particles is theoretically possible and at least as powerful as a Turing machine [4]. Just two years later, Feynman *et al.* [5] realized that information processing and computing on quantum bits might be more powerful in some cases than classical computing . This led to a number of algorithms that were adapted to the new quantum architecture. Coming back to the example of factorization from above, in 1994, Peter Shor proposed an algorithm for factoring integers that is exponentially faster than the fastest classical factoring algorithms. This is probably the most prominent instance of *quantum supremacy*, the ability to solve some problems in significantly less time on quantum computers than on classical computers.

Since the early 90s, when many new algorithms were developed, the main obstacle was the experimental implementation of a quantum computer. Since then, the work space has been gradually improved from a handful of qubits to tens of qubits nowadays. This is still far from a dimension, though, with which real quantum supremacy over classical computers could be achieved. In 2019, for the first time, something at least close to quantum supremacy was claimed by Google [13]. The problem they addressed was very specific, though, and currently has not interesting application. Simply speaking, they verified part of a random number experiment that would take classical computers orders of magnitude more time to solve. It will still take a long time until real world applications are dominated in performance by quantum computers.

2.3. Principles of nuclear magnetic resonance methods

We will now give an introduction to a field of physics that has long been a pillar of medical imaging procedures. Although this is the most known application, its methods are important in many fields of material science, predating its use in clinics. In this thesis, we will not try to image properties of materials, but actively manipulate a physical system and its information with the same techniques.

2.3.1. NMR spectroscopy

When particles have a non-vanishing spin angular momentum $\vec{S}$, they also possess a magnetic moment $\vec{\mu} = \gamma\vec{S}$, where $\gamma = gq/2m$ with g the gyromagnetic factor, q the charge, and m the mass of the particle. Particles with magnetic moments can be manipulated and controlled using magnetic fields. This is the method used in nuclear magnetic resonance spectroscopy.

Consider a particle with spin S and magnetic moment $\vec{\mu}$. If it is placed in a constant and homogeneous magnetic field with magnetic flux density $\vec{B}$, its Hamiltonian reads

$$\hat{H} = \vec{B}\,\hat{\vec{\mu}} = \vec{B}\gamma\hat{\vec{S}}\ , \tag{2.3.1}$$

where $\hat{\vec{S}} = \frac{\hbar}{2}(\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z)$ is the vector of Pauli matrices.

To find the evolution of a state $|\Psi(t)\rangle$ under the Hamiltonian $\hat{H}$ we need to solve the *time dependent Schrödinger equation*

$$\frac{\mathrm{d}}{\mathrm{d}t}\,|\Psi(t)\rangle = i\hat{H}(t)\,|\Psi(t)\rangle\ . \tag{2.3.2}$$

As the Hamiltonian in Eq. (2.3.1) is not dependent on time, we find the solution to the Schrödinger equation as

$$|\Psi(t)\rangle = \exp i\hat{H}t/\hbar\,|\Psi(t_0)\rangle\ . \tag{2.3.3}$$

We define the *time propagator* as

$$\hat{U}(t, t_0) = \exp i\hat{H}(t - t_0)/\hbar \tag{2.3.4}$$

and it gives the time evolution of the state $|\Psi(t)\rangle = \hat{U}(t, t_0)\,|\Psi(t_0)\rangle$. The operator $\hat{U}$ is unitary because it is the exponential of a Hermitian matrix $\hat{H}$.

We now investigate how the time propagator $\hat{U}$ acts on an arbitrary quantum state. From this point on, we will set $t_0 = 0$ without loss of generality. Then the propagator becomes $\hat{U}(t) \equiv \hat{U}(t, 0)$. Without loss of generality, we choose a direction of the static magnetic field, say $\vec{B} = B\vec{e}_z = (0, 0, B)$. The Hamiltonian in Eq. (2.3.1) then simplifies to $\hat{H} = \gamma B\hat{S}^z$. With that, the time propagator becomes

$$\hat{U}(t) = \exp(i\hat{H}t/\hbar) = \exp\left(-\mathrm{i}\,B_0\gamma_\mathrm{A}t\hat{S}^z_\mathrm{A}\right) = \begin{pmatrix} \mathrm{e}^{-\mathrm{i}\,B_0\gamma_\mathrm{A}t/2} & 0 \\ 0 & \mathrm{e}^{+\mathrm{i}\,B_0\gamma_\mathrm{A}t/2} \end{pmatrix} \equiv R_z(\theta = -B_0\gamma_\mathrm{A}t)\,. \tag{2.3.5}$$

This operator $R_z(\theta) \in \mathfrak{su}(2)$ can directly be mapped to a rotation in SO(3) with the homomorphism in Eq. (2.1.9). Indeed, the corresponding operator in SO(3) is just a rotation around the z-axis by an angle $\theta = B_0\gamma_\mathrm{A}t$ [14]. In a static magnetic field $\vec{B}$, the state vector thus steadily precesses around the z-axis. By applying a magnetic field along an arbitrary axis ϕ along the unit vector $\vec{\phi}$, one can similarly create a precession $R_\phi(\theta)$.

In classical NMR-spectroscopy, this precession is used to detect the type of nuclei inside a sample. The sample is exposed to a strong and homogeneous magnetic field.

The $\vec{B}$-field causes all the nuclei of the atoms to precess that have $S \neq 0$, where S is the total spin of the nucleus. The gyromagnetic ratios γ of the nuclei differ. Hence, they will precess at different angular velocities. Since the precessing spins each create their own magnetic field, resonances in the perpendicularly applied rf-field can be detected with a pickup-coil. With this procedure one can spatially resolve the areas of concentrated atoms of a certain kind. We know this method most prominently from medical science in order to image body tissue (MRI). It is also applied in physical, chemical and biological analysis of materials.

2.3.2. Manipulation of spins using radio frequency electromagnetic fields

Up to now, we have only discussed the behavior of a spin in a strong homogeneous magnetic field. We will now try to manipulate the state of such a spin. More precisely, we want to change the z-projection of the spin. Naturally, if a $\vec{B}$-field is applied, as we have just seen, the x and y projections of the spin do change at all times because the spin precesses around the z-axis. In order to change the z value of the spin, we need to rotate the Bloch vector of the spin around some axis in the x-y-plane (or at least not parallel to z). Since the Bloch vector precesses around the z-axis, we need the second rotation axis to also rotate around the z-axis in resonant frequency. We have just learned in the previous section that a magnetic field induces a rotation around its field axis. So we need an additional magnetic field that rotates around the z-axis in the x-y-plane. The frequency of the rotating field has to be synchronized with the original precession frequency ω_{P}.

An electromagnetic radio frequency field with its Poynting vector in z-direction has its electric and magnetic field vectors in the x-y-plane. If the radio frequency field has frequency ω_{rf}, the electric and magnetic fields rotate around the z-axis with frequency ω_{rf}. We now apply the rotating wave approximation [15] and change to the rotating frame. We then obtain the total magnetic field

$$\vec{B} = \vec{B}_0 + \vec{B}_{\mathrm{rf}} \simeq \begin{pmatrix} B_{\mathrm{rf}} \cos(\phi) \\ B_{\mathrm{rf}} \sin(\phi) \\ B_0 - \omega/\gamma \end{pmatrix} \simeq \begin{pmatrix} B_{\mathrm{rf}} \cos(\phi) \\ B_{\mathrm{rf}} \sin(\phi) \\ 0 \end{pmatrix}, \tag{2.3.6}$$

where ϕ is the phase of the rf-field relative to the precession around $\vec{B}_0$. The z-component $B_0 - \omega/\gamma$ is an expression for the chemical shift and can be neglected for reasonably large B_{rf} [14]. If we drive the radio frequency field in resonance with the precession, the phase ϕ thus determines the direction of the effective magnetic field in the x-y-plane. As we already know, a magnetic field causes a rotation. Hence, the spin rotates around an effective axis determined by ϕ. In the resting frame of reference, the axis and the spin precess around the z-axis as well.

Such a rotation by 90° is shown in Fig. 2.1. The initial state $|\uparrow_x\rangle = (|\uparrow_z\rangle + |\downarrow_z\rangle)/\sqrt{2}$ is

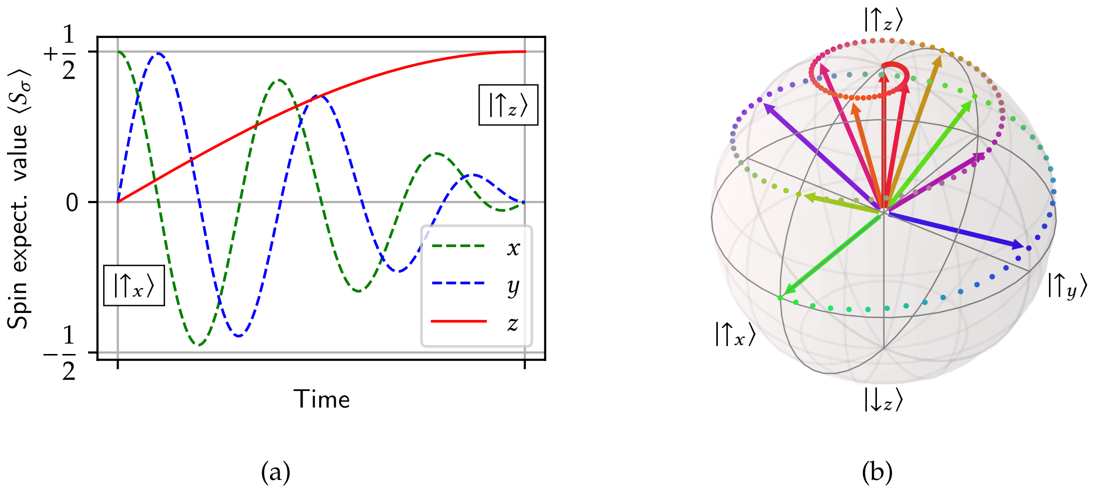


Figure 2.1.: Rotation of a spin by 90° in a strong homogeneous magnetic field, using a radio-frequency pulse that is in resonance with the precession frequency. The initial state $|\uparrow_x\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$ is rotated around the y-axis to the final state $|\uparrow\rangle$. (a) Expectation values for the spin in each of the three directions $\sigma = x, y, z$ during the rotation. (b) Visualization of the combined action of constant magnetic field and rf-field in the bloch sphere. In the resting frame, which is shown here, the Bloch-vector spirals up to the $|\uparrow\rangle$ state. In the rotating frame of reference, which is in resonance with the spin precession, the rotation would look like a constant rotation in the $x-z$-plane.

rotated around the y-axis in the rotating frame to reach the final state $|\uparrow_z\rangle$. In Fig. 2.1a one can see the expectation values $\langle S_\sigma\rangle$ for $\sigma = x, y, z$ during the time of the rotation. It is clearly observable that the x- and y-projections are oscillating with a high precession frequency. Over the period of several revolutions, the state slowly changes towards the final state $|\uparrow_z\rangle$. The latter can be seen as the x- and y-projections losing magnitude and the z projection permanently getting larger. In Fig. 2.1b the same rotation is shown in the Bloch-sphere representation. The state slowly spirals away from the x-y-plane towards the north pole which represents the pure state $|\uparrow_z\rangle$.

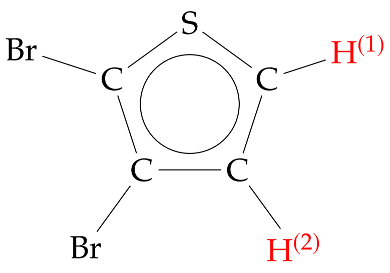
2.3.3. Rotations with more spins

If we have two spins that are uncoupled, they are separable for all times. Mathematically speaking, this means that we can write the state $|\Psi\rangle$ of the system as the tensor product

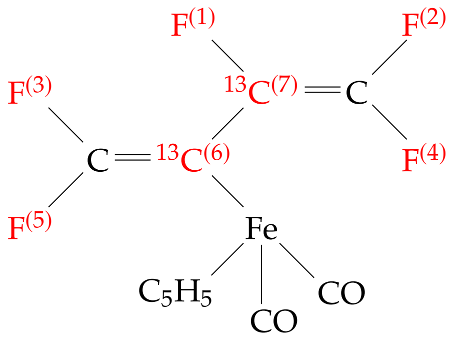
$$|\Psi\rangle = |\Psi_A\rangle \otimes |\Psi_B\rangle \; , \tag{2.3.7}$$

where $|\Psi_A\rangle \in \mathscr{H}_A$ and $|\Psi_B\rangle \in \mathscr{H}_B$ are states in the sub-Hilbert-spaces of the respective spins. Another way of looking at separability is to note that the spins are not entangled. It follows that we can easily adapt Eq. (2.3.5) to the Hilbert space of the two spins, labeled with A and B, by the tensor products

$$R_{\phi,A}(\theta) = R_\phi(\theta) \otimes \mathbb{1} \qquad \text{and} \qquad R_{\phi,B}(\theta) = \mathbb{1} \otimes R_\phi(\theta) \; . \tag{2.3.8}$$



(a)



(b)

Figure 2.2.: Two examples of suitable molecules for liquid NMR quantum computing. The atoms of the qubit nuclei are colored red and labeled with a number in the superscript. (a) (2,3)-dibromothiophene as proposed by Cory et al. for implementation of a CNOT gate [16]. (b) A more complex molecule with seven qubits for calculations. This synthetic molecule was used by Vandersypen et al. to factorize the number 15 using the Shor algorithm [17]. The two center carbon atoms are ^{13}C isotopes, as the more common ^{12}C has zero nuclear spin.

Mathematically, it is clear how to perform this-sub-Hilbert space rotation, but how does it work in practice? In order to rotate spins in a combined system individually, one has to be able to drive individual spins with the radio frequency field but leave out the other spins. Two concepts are required for that.

The first requirement is the *chemical shift*. If the spins are in slightly different chemical environments, then their Zeeman-splitting energies will be differentiated by a tiny shift. This is the case, for example, in asymmetric molecules like the ones shown in Fig. 2.2. The hydrogen atoms in Fig. 2.2a have a different set of neighbors, so they are subject to different chemical potentials. These chemical potentials shift the Zeeman splitting by a few parts per million [14]. Hence, the two hydrogen atoms are distinguishable by their precession frequency. Nuclei of different atoms (heteronuclear) are naturally distinguishable because they have a different magnetic moments.

The second requirement is that we have access monochromatic radio frequency fields. We can then drive the rotation of only one spin because the two spins have distinguishable transition energies (chemical shift). This means we can selectively rotate the spins by a precisely tuned radio frequency, and the rotation is represented by Eq. (2.3.8). Later, we will loosen up this requirement because pulses of a radio frequency field with finite length will always contain a bandwidth of frequencies. This will cause the problem, that spins with similar transition energies will also be driven to some extend by a pulse that was not intended for them. But for now, let us stick with the idealization of quantized fields.

2.3.4. The coupling dance in the Bloch sphere

We will now introduce interaction between the spins in the form of a coupling term in the Hamiltonian. The two-spin state will then be separated as well as possible by partially tracing out the respective other subspace. This will lead to a representation in the Bloch sphere in which we follow the traced-out individual spin states. We have to keep in mind, though, that the evolution takes place in the full four-dimensional Hilbert space of the two spins leading to rotations in the two-spin Hilbert space.

We investigate these evolutions for both the Ising and the Heisenberg models. The Ising model is a good approximation for the spin-spin interaction in NMR experiments in the *weak coupling* regime. Weak coupling means that the difference between the frequencies due to the chemical shift is much larger than the J-coupling strength, i.e.,

$$\left|\frac{J}{2}\right| \ll \left|(\delta_{\mathrm{A}} - \delta_{\mathrm{B}})\frac{\omega_0}{2\pi}\right|, \tag{2.3.9}$$

where δ are the chemical shifts of spin A and spin B, and ω_0 is the Larmor-precession-frequency of the (homonuclear) spin-pair [9]. If this inequality holds, the dipole-dipole interaction and the anisotropic part of the J-coupling average out in a liquid NMR sample due to non-directional inter- and intramolecular motion. The interaction Hamiltonian is then diagonal in the S_z-basis. The weak-coupling regime can always be applied for heteronuclear spins-pairs because their precession frequencies differs greatly.

On the other hand, when the J-coupling and the chemical shift frequency are roughly of the same magnitude, the off-diagonal elements of the general Hamiltonian do not vanish. In this *strong-coupling* regime, the Heisenberg spin model is the better model for the spin interaction. The strong-coupling regime, and thus the Heisenberg model, can be applied to homonuclear spins in relatively large molecules that are not as agile as small molecules.

In both models, the paths of the traced-out subsystems in the Bloch sphere circle around each other. The paths of the individual subspaces are always dependent on each other. Thus, one could visualize their behavior as a kind of choreographed dance in the Bloch sphere.

These rotations are the crucial ingredient for non-trivial two-spin quantum gates. The non-separable nature of these rotations is essential because they can generate entanglement between individual spins. We will derive how to construct quantum gates from these rotations in Chap. 3, where we will make use of the non-separability of the rotations to form manipulations of spin states that are conditional on other spin states.

Ising coupling rotation

The Hamiltonian for the Ising model of two spins is

$$\hat{H} = \hat{H}_{\mathrm{A}} + \hat{H}_{\mathrm{B}} + J\,\hat{S}_{z,\mathrm{A}}\hat{S}_{z,\mathrm{B}}\,, \tag{2.3.10}$$

where $\hat{S}_{z,\mathrm{A}}$ and $\hat{S}_{z,\mathrm{B}}$ are the spin operators, and J is the coupling strength. Here $\hat{H}_{\mathrm{A}}$ and $\hat{H}_{\mathrm{B}}$ are the local spin Hamiltonians of each spin that arise, for example, due to the strong homogeneous magnetic field. As described in the last section, these parts of the Hamiltonian, isolated, lead to free rotations around the axis of the magnetic field in the time evolution. But what does the coupling term contribute to the time evolution of the whole system? We calculate the time evolution operator for the interaction term $J\,\hat{S}_{z,\mathrm{A}}\hat{S}_{z,\mathrm{B}}$ in the S_z-basis, obtaining

$$\begin{aligned}
\hat{U}(t) &= \exp\left(\mathrm{i}\,J\hat{S}_{z,\mathrm{A}}\hat{S}_{z,\mathrm{B}}t/\hbar\right) \\
&= \exp\left(\mathrm{i}\,Jt/4\,(\sigma_z\otimes\mathbb{1})\cdot(\mathbb{1}\otimes\sigma_z)\right) \\
&= \exp\left(\mathrm{i}\,Jt/4\,(\sigma_z\otimes\sigma_z)\right) \\
&= \exp\left(\mathrm{i}\,Jt/4\begin{pmatrix}1&&&\\&-1&&\\&&-1&\\&&&1\end{pmatrix}\right) \\
&= \cos\left(Jt/4\right)\mathbb{1}_4 + \mathrm{i}\sin\left(Jt/4\right)\begin{pmatrix}1&&&\\&-1&&\\&&-1&\\&&&1\end{pmatrix} \\
&\equiv R_{z\mathrm{AB}}(\alpha t)\,,
\end{aligned} \tag{2.3.11}$$

where $\alpha \equiv J\hbar/4$. This can be viewed as a rotatio,n as the sinus and cosinus terms suggest. But what does this operation actually do in the four-dimensional Hilbert space? If we evaluate this operator matrix for representative values of $\theta = \alpha t$, we find that

$$R_{z\mathrm{AB}}(0) = \mathbb{1}_4\,,\quad R_{z\mathrm{AB}}(180^\circ) = \frac{1+\mathrm{i}}{\sqrt{2}}\begin{pmatrix}1&&&\\&-\mathrm{i}&&\\&&-\mathrm{i}&\\&&&1\end{pmatrix},\quad R_{z\mathrm{AB}}(360^\circ) = \mathrm{i}\begin{pmatrix}1&&&\\&-1&&\\&&-1&\\&&&1\end{pmatrix}. \tag{2.3.12}$$

The operator $R_{z\mathrm{AB}}(\theta)$ is, in general, not separable into the two subspaces of spins A and B.

There is no representation like the Bloch sphere for a four-dimensional Hilbert space. We would need six dimensions to represent the behavior. However, we will separate the full product space into two single-qubit states by tracing out the rest of the Hilbert space. For a state $|\Psi\rangle$ in the full Hilbert space $\mathscr{H} = \mathscr{H}_{\mathrm{A}}\otimes\mathscr{H}_{\mathrm{B}}$, we can then plot the traced-out state

$$|\Psi_\mathrm{A}\rangle = \mathrm{Tr}_B\, |\Psi\rangle \;\in \mathscr{H}_\mathrm{A} \tag{2.3.13}$$

in the Bloch sphere of spin A.

We can now investigate the effect of the operation in Eq. (2.3.11) on the total state, broken down into two sub-Hilbert-spaces. In Fig. 2.3 this is done for two different initial states, $|{\uparrow_z}\rangle \otimes |{\uparrow_x}\rangle = |{\uparrow}\rangle \otimes (|{\uparrow}\rangle + |{\downarrow}\rangle)/\sqrt{2} = (|{\uparrow\uparrow}\rangle + |{\uparrow\downarrow}\rangle)\sqrt{2}$ and $|{\uparrow_x}\rangle \otimes (|{\uparrow_z}\rangle + |{\uparrow_z}\rangle)/\sqrt{2}$. The operator $R_{z\mathrm{AB}}(\theta)$ acts as an elliptic rotation of the state of a qubit. That is, both traced-out substates rotate around an ellipse in their respective Bloch spheres, corresponding to the traced-out sub-Hilbert-space $\mathscr{H}_\mathrm{A}$ or $\mathscr{H}_\mathrm{B}$. The eccentricity depends on the z-projection of the other qubits traced-out state. If qubit $\beta \in \{\mathrm{A,B}\}$ is in a projected state with expectation value $S^z_\beta = \pm 1/2$, then the other qubit experiences a circular rotation around the z-axis. For expectation values $0 < S^z_\beta < 1/2$, the other qubit rotates on track of an ellipse. For $S^z_\beta = 0$, the "rotation" flattens out to an oscillation on a line perpendicular to the z-axis. We can already see here the conditional action of this rotational operator. In Chap. 3 we will learn, why this behavior is essential for creating a universal set of quantum gates.

In the Bloch sphere representation, it is clear to see why $R_{z\mathrm{AB}}(\theta)$ is a non-separable unitary operator. For the general case, $0 < S^z_\beta < 1/2$, the curve of the projected state in the Bloch sphere deviates from the surface of the sphere. It can be observed in Fig. 2.3b. That means that the sub-states are changing to mixed states. Since the full two-qubit state will stay pure under the action of a unitary operator, if follows that there is a non-separable action that cannot be resolved by projecting the full state onto the sub-Hilbert-spaces of the individual qubits.

Heisenberg XXX coupling rotation

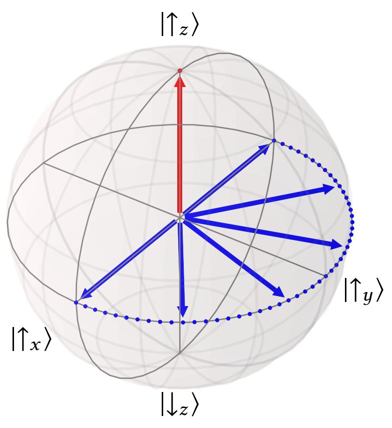
So far, we have investigated interaction in the (semi-)classical Ising model. We will now move on to the more quantum-mechanical case and treat the isotropic Heisenberg Hamiltonian

$$\hat{H} = \hat{H}_\mathrm{A} + \hat{H}_\mathrm{B} + J\,\vec{\hat{S}}_\mathrm{A}\vec{\hat{S}}_\mathrm{B}\;, \tag{2.3.14}$$

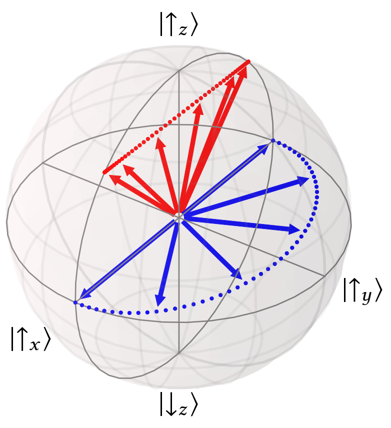
where, again, the first two terms describe the local Hamiltonians of the respective spins. Recall that the interaction term can be expressed as the vector product of Pauli operators:

$$\hat{H}_\mathrm{c} = J\,\vec{\hat{S}}_\mathrm{A}\vec{\hat{S}}_\mathrm{B} = J\left(\hat{S}_{x\mathrm{A}}\hat{S}_{x\mathrm{B}} + \hat{S}_{y\mathrm{A}}\hat{S}_{y\mathrm{B}} + \hat{S}_{z\mathrm{A}}\hat{S}_{z\mathrm{B}}\right) = \frac{J}{4}\begin{pmatrix} 1 & & & \\ & -1 & 2 & \\ & 2 & -1 & \\ & & & 1 \end{pmatrix}. \tag{2.3.15}$$

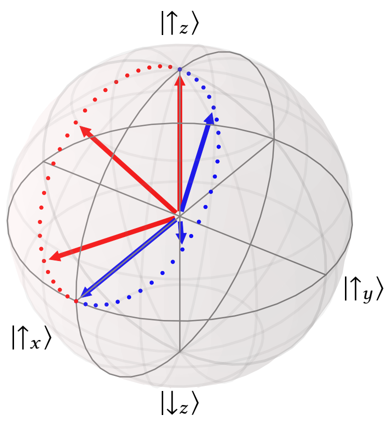
To derive the evolution of the system under the interaction term, we have to take its



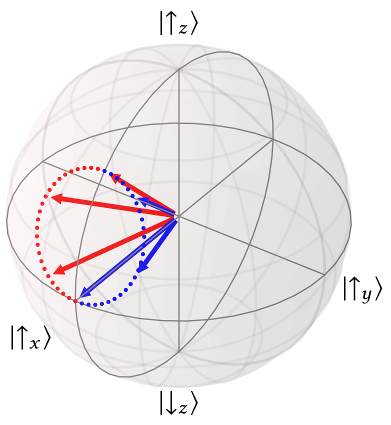
(a) Ising coupling,
$|\Psi_0\rangle = |\uparrow_z\rangle \otimes |\uparrow_x\rangle$



(b) Ising coupling,
$|\Psi_0\rangle = (|\uparrow_z\rangle + |\uparrow_x\rangle)/\sqrt{2} \otimes |\uparrow_x\rangle$



(c) Heisenberg coupling,
$|\Psi_0\rangle = |\uparrow_z\rangle \otimes |\uparrow_x\rangle$



(d) Heisenberg coupling,
$|\Psi_0\rangle = (|\uparrow_z\rangle + |\uparrow_x\rangle)/\sqrt{2} \otimes |\uparrow_x\rangle$

Figure 2.3.: Action of the coupling rotation (Eq. (2.3.11)) on two different initial states. In the representation of separated qubits (red for qubit A, blue for qubit B), the single-qubit states are rotated on the line of an arc of an ellipse around the z-axis. The eccentricity depends on the z-projection of the other state. If the z-projection of one qubit is maximal (1/2), then the other qubit will rotate on a perfect circle on the surface of the Bloch sphere. This is shown in (a) for the initial state $|\uparrow_z\rangle \otimes |\uparrow_x\rangle = (|\uparrow\uparrow\rangle + |\uparrow\downarrow\rangle)\sqrt{2}$. The qubit starting in the $|\uparrow_x\rangle$ state is rotated around z with constant radius. In (b), both other cases are shown. If the z-projection of one qubit (blue) is minimal (0), then the second qubit (red) will oscillate on a straight line perpendicular to the z-axis. If the z-projection of one qubit (red) is somewhere between 0 and 1/2, then the rotation of the second qubit (blue) will be on an ellipse. From the Bloch sphere, it can be seen that this evolution is non-separable in the non-extremal cases because the projected single qubit states evolve over non-pure states (inside of the Bloch sphere). (c) and (d) show an evolution in the Heisenberg model with the same length and the same initial states as in (a) and (b). In the Heisenberg model, the Bloch vectors circle around each other in the shape of an ellipse. The eccentricity is given by the expectation value S_z of the respective other qubit.

exponential and form the propagator

$$e^{\mathrm{i}\hat{H}_c t} = e^{\mathrm{i}J\vec{\hat{S}}_A\vec{\hat{S}}_B t} = e^{-\mathrm{i}Jt/4}\begin{pmatrix} e^{\mathrm{i}Jt/2} & & & \\ & \cos(Jt/2) & \mathrm{i}\sin(Jt/2) & \\ & \mathrm{i}\sin(Jt/2) & \cos(Jt/2) & \\ & & & e^{\mathrm{i}Jt/2} \end{pmatrix} = R_{\mathrm{AB}}^{\mathrm{XXZ}}(Jt)\,. \tag{2.3.16}$$

Comparison with Eq. (2.3.11) shows how the propagator changes if we switch from the Ising to the Heisenberg regime. The structure is no longer diagonal; there are now two off-diagonal elements. These off-diagonal elements lead to evolution that is no longer confined to the x-y-plane. In the Heisenberg model, the two Bloch vectors follow a choreography that forms an ellipse around one common axis. The rotational evolution is shown in Figs. 2.3c and 2.3d for two different initial states.

If we consider the auxiliary vector $|\mathrm{C}\rangle \equiv (|\mathrm{A}\rangle + |\mathrm{B}\rangle)/2$ that lies in between the starting vectors $|\mathrm{A}\rangle$ and $|\mathrm{B}\rangle$ of the respective subspaces A and B, then this vector $|\mathrm{C}\rangle$ marks the center of the ellipse, around which the two subspaces rotate. The eccentricity of the ellipse depends on the length of the vector $|\mathrm{C}\rangle$.
The eccentricity is large when the length of the auxiliary vector is small and vice versa. If the two vectors $|\mathrm{A}\rangle$ and $|\mathrm{B}\rangle$ are parallel, then the auxiliary vector has maximum length (the same as the two vectors), and the path of the two state vectors is a circle. Of course, when the two vectors and the auxiliary vector are the same, the circle reduces to a point on the Bloch sphere. Thus, if the two vectors are the same, the propagator is the identity operator for all times. This is the case when the full system is the tensor product of two identical states.
On the other hand, when the two initial subspace vectors are antiparallel, then the eccentricity of the ellipse is maximal. This means that the ellipse has become a straight line in the Bloch sphere, along which the two traced-out subspace vectors evolve.

Representative propagators at special rotation angles are

$$R_{\mathrm{AB}}^{\mathrm{XXX}}(0) = \mathbb{1}_4\,,\ R_{\mathrm{AB}}^{\mathrm{XXX}}(\pi/2) = \begin{pmatrix} 1+\mathrm{i} & & & \\ & 1 & \mathrm{i} & \\ & \mathrm{i} & 1 & \\ & & & 1+\mathrm{i} \end{pmatrix},\ R_{\mathrm{AB}}^{\mathrm{XXX}}(\pi) = \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix}, \tag{2.3.17}$$

where the overall phase factors were omitted because they are irrelevant for quantum measurements. We see that the free evolution in the Heisenberg XXX model represents the swapping of information for the two qubits if $Jt = \pi$.

Heisenberg XXZ coupling rotation

If we now assume that the Heisenberg interaction is not isotropic, we then obtain an extra degree of freedom in the coupling term. First, we investigate the case that the coupling strength in one of the three spatial coordinates deviates from the two others.

Without loss of generality, we assume that the z-term of the coupling deviates from the x- and y-terms by a factor Δ, that is,

$$\hat{H}_\mathrm{c} = J\Big(\Delta \hat{S}^z_\mathrm{A}\hat{S}^z_\mathrm{B} + \frac{1}{2}(\hat{S}^+_\mathrm{A}\hat{S}^-_\mathrm{B} + \hat{S}^-_\mathrm{A}\hat{S}^-_\mathrm{B})\Big) = \frac{J}{4}\begin{pmatrix} \Delta & & & \\ & -\Delta & 2 & \\ & 2 & -\Delta & \\ & & & \Delta \end{pmatrix}, \tag{2.3.18}$$

with $J_z = \Delta J_x = \Delta J_y$ and the ladder operators $\hat{S}^\pm = \hat{S}^x \pm \mathrm{i}\,\hat{S}^y$. Here, $\Delta = 1$ represents the isotropic Heisenberg model. As $\Delta \to \infty$, we approach the Ising model. Free evolution under the coupling term in the Heisenberg XXZ model is then given by

$$\mathrm{e}^{\mathrm{i}\hat{H}_\mathrm{c}t} = \mathrm{e}^{-\mathrm{i}J\Delta t}\begin{pmatrix} \mathrm{e}^{2\mathrm{i}J\Delta t} & & & \\ & \cos(Jt) & \mathrm{i}\sin(Jt) & \\ & \mathrm{i}\sin(Jt) & \cos(Jt) & \\ & & & \mathrm{e}^{2\mathrm{i}J\Delta t} \end{pmatrix} \equiv R^\mathrm{XXZ}_\mathrm{AB}(Jt)\,. \tag{2.3.19}$$

Exemplarily, we can give one special combination of Δ and Jt that we will use later to find a sequence for a quantum gate:

$$R^\mathrm{XXZ}_\mathrm{AB}(\pi/4, \Delta = 2) = \begin{pmatrix} -1 & & & \\ & \frac{1}{\sqrt{2}} & \frac{\mathrm{i}}{\sqrt{2}} & \\ & \frac{\mathrm{i}}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \\ & & & -1 \end{pmatrix}. \tag{2.3.20}$$

As the final and most complex case, we note that the anisotropy Δ does not necessarily need to be constant over time. For example, the structure of the molecule and its environment could change and influence the way two spins interact with each other. We now want to investigate the case in which the anisotropy Δ gets larger, starting from the isotropic Heisenberg model. We assume that the anisotropy increases linearly as $\Delta(t) = 1 + t$. The Hamiltonian for the coupling then reads

$$\hat{H}_\mathrm{c}(t) = J\Big(\Delta(t)\hat{S}^z_\mathrm{A}\hat{S}^z_\mathrm{B} + \frac{1}{2}(\hat{S}^+_\mathrm{A}\hat{S}^-_\mathrm{B} + \hat{S}^-_\mathrm{A}\hat{S}^-_\mathrm{B})\Big) = \frac{J}{4}\begin{pmatrix} 1+t & & & \\ & -1-t & 2 & \\ & 2 & -1-t & \\ & & & 1+t \end{pmatrix}. \tag{2.3.21}$$

Solving for the propagator $\hat{U}(t)$ generally requires a time-ordered exponential if the Hamiltonian is time-dependent. However, in the specific case of a linear ramp, treated here, the Hamiltonians at different times t_i commute with each other, namely, $[\hat{H}_\mathrm{c}(t_i), \hat{H}_\mathrm{c}(t_j)] = 0\ \forall i, j$. Because of this, we can omit the time ordering and integrate the

Hamiltonian directly to yield the propagator

$$\hat{U}(t) = \exp\left\{ \mathrm{i} \int_0^t \mathrm{d}t' \hat{H}_\mathrm{c}(t')/\hbar \right\} \simeq \begin{pmatrix} \mathrm{e}^{i(t+t^2)/2\hbar} & & & \\ & \cos(t/2\hbar) & \mathrm{i}\sin(t/2\hbar) & \\ & \mathrm{i}\sin(t/2\hbar) & \cos(t/2\hbar) & \\ & & & \mathrm{e}^{i(t+t^2)/2\hbar} \end{pmatrix}, \tag{2.3.22}$$

where an overall phase factor was discarded because it has no relevance for quantum measurements. We will not take this case into account in the upcoming search for quantum gates, but mention it to inspire future work.

This concludes the short introduction into the theoretical aspects on which the work in the upcoming chapters is based. While we will neither develop the spin interaction models nor the quantum information theory any further, the theory of NMR quantum operations will be extended in the next chapter. We will then be at a point where we will be able to execute quantum computations on the architecture of a molecule inside and NMR experiment, at least, theoretically.

3. Forming quantum gates from NMR qubit rotations

We will now build on the theoretical introduction of the Chap. 2. The ideas of nuclear magnetic resonance will be put to use, not in an imaging process, but to manipulate qubits. We will see that we can choose the manipulations so that we can implement quantum gate operations. By combining these manipulations to form a sequence of pulses, we will be able to form more complex gates. Finally, we will show that these gates are sufficient to carry out universal quantum computing on an NMR architecture, the quantum gates just need to form a universal set. A key element of such a set is the CNOT gate. Our goal will thus be to search for NMR pulse sequences that have the effect of a CNOT gate on an arbitrary initial state. In this work, our primary goal will be to generalize these sequences from the Ising to the Heisenberg spin model. While there has been extensive work on sequences based on the Ising interaction, we know of no known pulse sequences in the literature, that implement a CNOT gate using a Heisenberg interaction between the two spin qubits.

3.1. Elementary single qubit gates

As a first step to realize quantum computations by applying NMR methods, we will limit our focus to just one qubit. This qubit would typically be one of the nuclear spins of a molecules in an NMR sample.

As shown in Chap. 2, if we apply a magnetic field to quantum spins, they will precess around the axis of the applied B-field. We can use this rotation to implement single qubit gates if we apply an rf-pulse for the appropriately chosen period of time. By adjusting the time of the pulse, we can rotate the spin by a set angle θ.

The NOT-gate on a singe qubit is represented by the Pauli matrix σ_x. It is straightforward to prove that the NOT-gate is a rotation around the x-axis by an angle of π. We compute it by inserting $\theta = \pi$ into Eq. (2.3.5), the operation for rotations around the x-axis. This means, that in order to negate a qubit, one has to apply a magnetic field in the x-direction for the time

$$\tau = \frac{\theta}{B_0 \gamma}, \tag{3.1.1}$$

so that the state rotates by $\theta = \pi$ during that time. Then, up to an irrelevant global phase factor i, the operation is the NOT-gate

$$U_{\mathrm{NOT}} = R_x(\pi) = \mathrm{i}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} . \tag{3.1.2}$$

Similarly, one can define the Y and Z gates (Pauli matrices) for rotations around the y- and z-axes, respectively, as

$$U_{\mathrm{Y}} = R_y(\pi)\,, \quad U_{\mathrm{Z}} = R_z(\pi)\,. \tag{3.1.3}$$

The generalization of the Z gate is the phase gate, often denoted by P. It is just the rotation

$$U_{\mathrm{P}} = R_z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\,\theta} \end{pmatrix} . \tag{3.1.4}$$

It adds a set relative phase $\mathrm{e}^{\mathrm{i}\,\theta}$ to the state.

The NOT gate and its y and z counterparts are operations that we also know from classical computation. We will now consider a quantum gate that has no classical counterpart. The Hadamard gate can be viewed as the single-qubit version of the quantum Fourier transform. Its action is equivalent to the basis transform from the S^z to the S^x basis. One way of writing the matrix representation is $H = (U_{\mathrm{X}} + U_{\mathrm{Z}})/\sqrt{2}$. In this representation, it is the sum of two rotations around two cartesian axes, but we can instead rotate around the median axis, i.e., an axis between the $x-$axis and the z-axis. This axis has the direction $\vec{\phi} = (1, 0, 1)/\sqrt{2}$ in cartesian coordinates. Hence, the Hadamard gate can be implemented by the rotation

$$U_{\mathrm{H}} = R_{\hat{\phi}}(\pi) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} , \tag{3.1.5}$$

where the rotation axis $\hat{\phi} = (1, 0, 1)/\sqrt{2}$ is defined by the polar and azimuthal angles $(\theta, \varphi) = (\pi/4, 0°)$.

We assume that rf-fields are the source of the magnetic field that rotates the spin. If we take its mean field vector to be perpendicular to the strong homogeneous B-field, then the magnetic field vectors will be in the x-y-plane. Because the axis of rotation is confined to a plane in this image, we need to apply a sequence of rotations instead of the single rotation around a tilted axis $\hat{\phi}$ that does not necessarily lie in that plane. The sequence for the Hadamard gate can then be formed as

$$U_{\mathrm{H}} = R_x(\pi)\,R_y(-\pi/2) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} . \tag{3.1.6}$$

However, finding a sequence that is equivalent to some arbitrary quantum gate is, in general, not easy.This searching difficulty is a clear hint that its problem probably lies in the complexity class NP, which we have treated in Sec. 2.2.1. We will come to more complex sequence searches later and will discuss the methods of finding such sequences in Chap. 4.

3.2. Two-qubit gates

One-qubit gates alone are obviously not sufficient to form a universal set of quantum gates. In order to do this, we need to find at least one gate, that acts on two or more qubits. The controlled NOT (CNOT) gate the canonical example of such a two-qubit gate. We will show that we can implement the CNOT gate from rotations of individual qubits and from a coupling rotation.

As mentioned in Chap. 2, we can investigate implementing two-qubit rotations using an Ising or a Heisenberg interaction. Which one is experimentally relevant, depends on the characteristics of the systems spin-arrangement and on the magnitudes of the different interactions. In the following, we will study pulse sequences to implement two-qubit gates for both of the aforementioned interaction types. We will begin with the Ising interaction. We can then build on these results to search for gates in the case of the Heisenberg interaction.

We will learn that the Ising and Heisenberg models also differ in the way in which they can be used for quantum computation, that is, the pulse sequences for quantum gates are, in general, not the same for the two models. For the Ising interaction, it is rather easy to find a sequence of rotational pulses that implements the CNOT gate. On the other hand, for the Heisenberg interaction, we will be able to implement the SWAP gate as a single evolution of the two-qubit coupling.

3.2.1. Ising model

As shown in Chap. 2, the Ising interaction between two quantum spins A and B has the form

$$\hat{H}_\mathrm{c} = J\,\hat{S}^z_\mathrm{A}\hat{S}^z_\mathrm{B}\,, \tag{3.2.1}$$

where J is the coupling strength, and we choose the z-axis as our preferred axis of spin angular momentum without loss of generality. We will confine ourselves to a system of two spins here for the sake of simplicity. Additional qubits can be treated in an analog manner; one just needs to enlarge the Hilbert space to take into account the additional spin degrees of freedom.

We now want to find a sequence of rotations that allows us to apply the CNOT operation to an arbitrary input state. For this, we will incorporate two different types of evolution: first, the single qubit-gates that we have covered in the last section, and second, the interaction evolution that arises from Eq. (3.2.1). The propagation rotation for the Ising interaction in matrix form is

$$R_{z\mathrm{AB}}(Jt/\hbar) = \mathrm{e}^{\mathrm{i}\,\alpha \hat{H}_{\mathrm{c}}/\hbar} = \begin{pmatrix} \mathrm{e}^{\mathrm{i}\,Jt/2\hbar} & & & \\ & \mathrm{e}^{-\mathrm{i}\,Jt/2\hbar} & & \\ & & \mathrm{e}^{-\mathrm{i}\,Jt/2\hbar} & \\ & & & \mathrm{e}^{\mathrm{i}\,Jt/2\hbar} \end{pmatrix}, \tag{3.2.2}$$

where t is the duration time of the pulse, and J is the interaction strength.

We do make a few simplifying assumptions here. Firstly, we assume that either a single qubit manipulation or an interaction evolution takes place, but never both at the same time. This is actually not strictly true - we cannot just switch off the interaction when we want to manipulate single spins. However, the rotation speed due to interaction is, in general, much slower than for singe-qubit rotations. Hence, we may assume that, during the time of a single-qubit operation, the interaction does not cause a significant amount of decoherence. In addition, we assume that the shape of the pulse amplitudes is rectangular. That is, we switch the magnetic fields on and off in infinitesimal time. We will discuss these simplifications further in Chap. 5.

CNOT and CPHASE gate

Taking into account the aforementioned restriction, one known sequence for a CNOT gate is given by

$$U_{\mathrm{CNOT}} = R_{y\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{B}}(\pi/2)\,. \tag{3.2.3}$$

This sequence for the CNOT gate with the Ising interaction can be found in most of the papers on NMR quantum computing, for example in Ref. [18]. We can calculate the effect of this expression by applying the matrix representations of the respective rotations. Note that, as we are working with operators, the first pulses applied chronologically are on the right side of the sequence. This means that, in Eq. (3.2.3), the first action on the state is $R_{y\mathrm{B}}(\pi/2)$. To apply the sequence, we compute

$$\begin{aligned}
U_{\mathrm{CNOT}} &= \xi \begin{pmatrix} 1 & -1 & & \\ 1 & 1 & & \\ & & 1 & -1 \\ & & 1 & 1 \end{pmatrix} \begin{pmatrix} 1-\mathrm{i} & & & \\ & 1-\mathrm{i} & & \\ & & 1+\mathrm{i} & \\ & & & 1+\mathrm{i} \end{pmatrix} \begin{pmatrix} 1-\mathrm{i} & & & \\ & 1+\mathrm{i} & & \\ & & 1-\mathrm{i} & \\ & & & 1+\mathrm{i} \end{pmatrix} \\
&\quad \cdot \begin{pmatrix} 1+\mathrm{i} & & & \\ & 1-\mathrm{i} & & \\ & & 1-\mathrm{i} & \\ & & & 1+\mathrm{i} \end{pmatrix} \begin{pmatrix} 1 & 1 & & \\ -1 & 1 & & \\ & & 1 & 1 \\ & & -1 & 1 \end{pmatrix} \\
&= \xi \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{pmatrix},
\end{aligned} \tag{3.2.4}$$

which is the CNOT gate up to an irrelevant global phase factor $\xi = (1-\mathrm{i})/\sqrt{2}$ [14]. The sequence is a rotation of the target bit B, followed by a free evolution of the coupled spins and then three consecutive rotations of the target qubit, the control qubit and the target qubit, respectively. There is no such representation like the Bloch sphere for a four-dimensional Hilbert space, but we can still trace out and plot the directions of the two spins over the course of the sequence. In Fig. 3.1, the initial state $|{\downarrow\downarrow}\rangle = |11\rangle$ is transformed to the final state $|{\downarrow\uparrow}\rangle = |10\rangle$ via the CNOT gate. For this particular initial state, spin A does not change at all.

Related to the CNOT gate is the controlled Z gate, which applies a phase flip to the second qubit if and only if the first qubit value is 1. The CZ gate can be implemented by changing the first and last operations of the CNOT gate to be z-axis- rather than y-axis-rotations. The sequence then reads

$$U_{\mathrm{CZ}} = R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{z\mathrm{B}}(\pi/2)\ . \tag{3.2.5}$$

By adjusting the time of interaction-only evolution, we can also implement a more general controlled-phase gate of the CZ-gate. That is, we are not limited to the full symmetric phase flip. The sequence for the controlled phase gate (CP) then reads

$$U_{\mathrm{CP}} = R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\alpha)\, R_{z\mathrm{B}}(\pi/2)\ , \tag{3.2.6}$$

where the angle α parameterizes the phase applied, and its matrix representation is

$$U_{\mathrm{CP}} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & e^{\mathrm{i}\,\alpha/2} \end{pmatrix}. \tag{3.2.7}$$

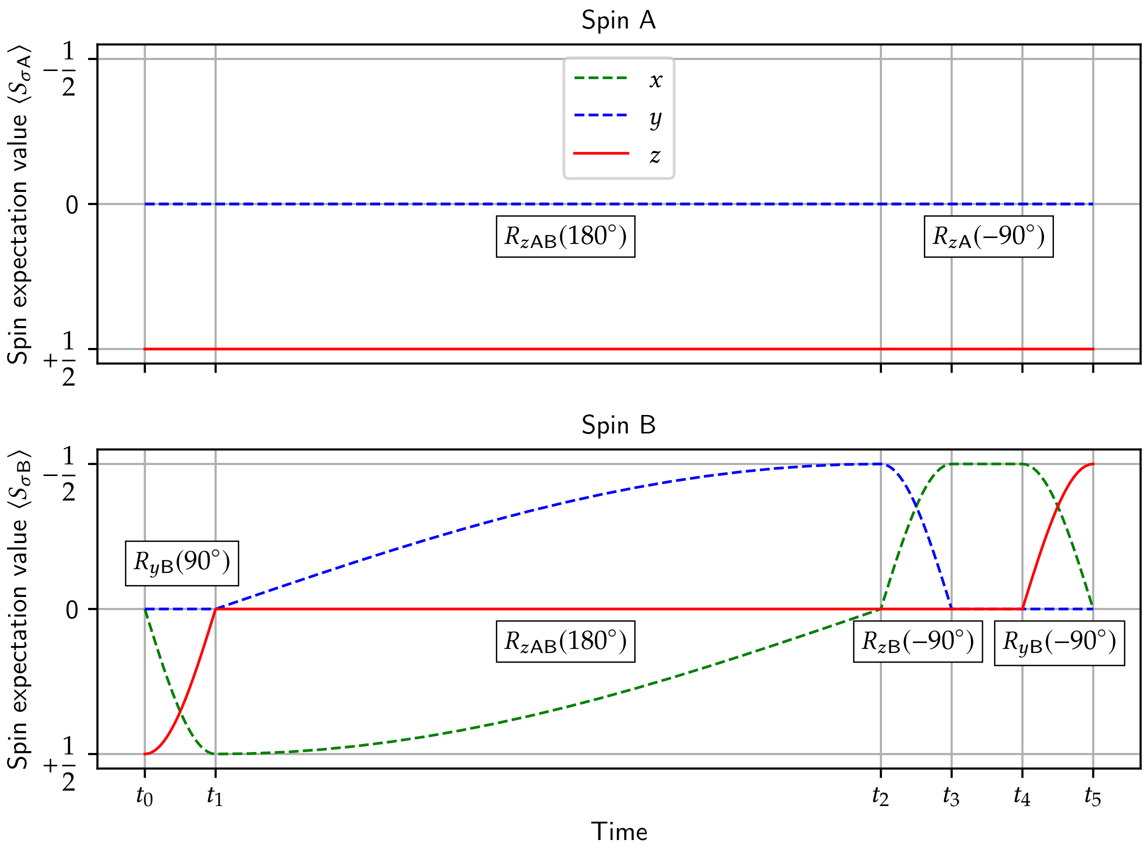


Figure 3.1.: Action of the sequence for the CNOT gate on the initial state $|{\downarrow\downarrow}\rangle$. Shown are the expectation values for all spatial projections S^σ, $\sigma = x, y, z$, for the two qubits over time. At the end of the sequence, the second qubit has been flipped so that the product states reads $|{\downarrow\uparrow}\rangle$.

We already know from Chap. 2 that the CNOT gate, combined with all single-qubit rotations, forms a universal set of quantum gates. This means that, with this sequence and the implementations of the single qubit gates described in the last subsection, we can form all possible unitary gates on two qubits. This set is also universal for larger numbers of qubits. Interactions that extend further than the nearest neighbor can be simulated by sequences of pairwise two-qubit operations on neighboring sites.

We have noted in Sec. 3.1 that single-qubit rotations cannot form a universal set of quantum gates because they cannot create or annihilate entanglement for two or more qubits. Now we will check this requirement for the CNOT gate, as made up of the operations in Eq. (3.2.3). In Fig. 3.2, the effect of this sequence is shown for the maximally entangled initial state $|\Psi_0\rangle = |{\uparrow\downarrow}\rangle + |{\downarrow\uparrow}\rangle$. Only the evolution of the spin B is shown because the spin A does not change during this operation. This can clearly be observed in Fig. 3.1. The total state is changed to the final state $|{\uparrow\downarrow}\rangle + |{\downarrow\downarrow}\rangle = |{\uparrow_x}\rangle \otimes |{\downarrow_z}\rangle$.

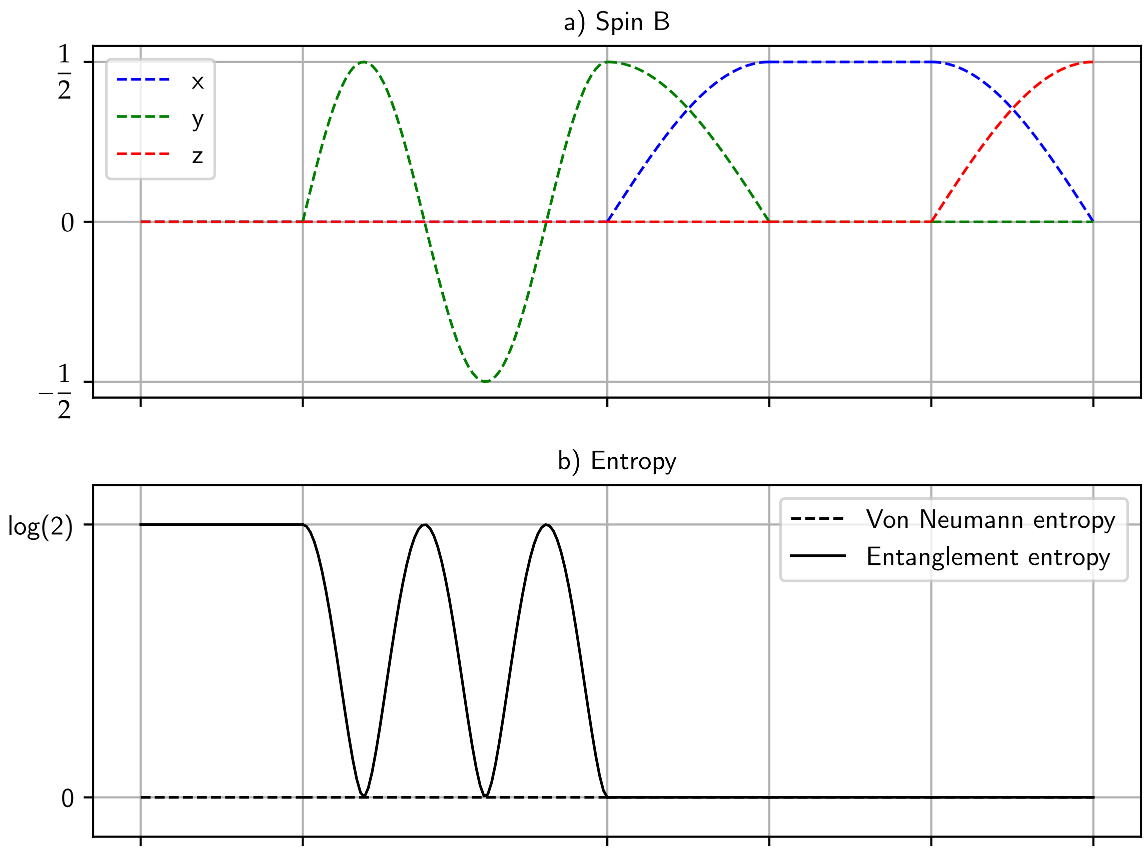


Figure 3.2.: Entropy evolution during the CNOT pulse sequence. The maximally entangled initial state $|\Psi_0\rangle = |\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle$ is changed to the non-entangled final state $|\uparrow\downarrow\rangle + |\downarrow\downarrow\rangle = |\uparrow_x\rangle \otimes |\downarrow_z\rangle$. a) Spin expectation values during the sequence. Only the expectation values of the spatial components of the second spin are shown here because the first spin only acts as control qubit. b) Entropy evolution during the sequence. The von Neumann entropy vanishes for all time; thus, the state remains pure. However, the entanglement entropy changes. Hence, the interaction between the two spins can create and annihilate entanglement.

The first thing we note is that the Von Neumann entropy

$$\mathcal{S}_{\mathrm{vN}} = -\operatorname{Tr}\left(\rho \log \rho\right) , \tag{3.2.8}$$

where ρ is the density operator of the two-spin system, vanishes over the full time of the CNOT operation because the initial state is pure (with $\rho = |\Psi_0\rangle\langle\Psi_0|$) and only unitary operations are applied. Hence, the state remains pure over the full length of the sequence.

The second observation is that the entanglement entropy of the system,

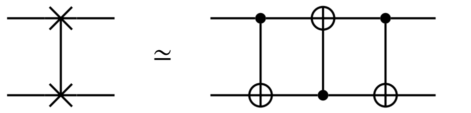
$$\mathcal{S}_{\mathrm{E}} = -\operatorname{Tr}\left(\rho_{\mathrm{A}} \log \rho_{\mathrm{A}}\right) = -\operatorname{Tr}\left(\rho_{\mathrm{B}} \log \rho_{\mathrm{B}}\right) , \tag{3.2.9}$$

which is the Von Neumann entropy of a traced-out single-qubit state, is not constant.

Here $\rho_A = \mathrm{Tr}_B\, \rho$. The entanglement entropy is a measurement for the entanglement between subsystems A and B. For two coupled two-level subsystems, it can range from 0 to log 2, where the latter value is for maximal entanglement. Indeed, we observe in Fig. 3.2 that the entanglement entropy of the initial state is maximal. During the interaction pulse, we see that the entropy oscillates up and down and ends up in a non-entangled final state. We clearly observe that the inter-qubit interaction is responsible for creating and annihilating entanglement. So we see that this was the missing ingredient needed for universal quantum computing.

SWAP gate

Another useful gate is the SWAP gate, which switches the information of the two qubits. The gate can be constructed as



,

where the left side represents the SWAP gate, and the right side consists of three CNOT gates with alternating orientation [19]. It is easy to prove via matrix multiplication that

$$U_{\mathrm{SWAP}} = U^{\mathrm{AB}}_{\mathrm{CNOT}}\, U^{\mathrm{BA}}_{\mathrm{CNOT}}\, U^{\mathrm{AB}}_{\mathrm{CNOT}} = \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix}, \tag{3.2.10}$$

where the superscript of U_{CNOT} denotes which qubit is the control bit. The first letter represents the control qubit and the second letter represents the target qubit. We now need to translate this into the machine language of NMR quantum computing, which are rotations caused by magnetic fields. The straightforward way to do this would be to apply the definition of the sequence for CNOT (Eq. (3.2.3)) three times, resulting in

$$\begin{aligned} U_{\mathrm{SWAP}} = & R_{y\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{B}}(\pi/2) \\ & \cdot R_{y\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{A}}(\pi/2) \\ & \cdot R_{y\mathrm{B}}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\, R_{z\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{B}}(\pi/2)\,. \end{aligned} \tag{3.2.11}$$

This, however, is a long sequence of pulses for a two-qubit gate. After all, the CNOT gate only needed one third as many pulses. The question arises: can we implement the SWAP gate with fewer pulses?

In general, this question is the quantum analog of the brachistochrone problem of classical analytical mechanics [20]. We are given an initial state $|\Psi_i\rangle$ and a final state $|\Psi_f\rangle$. The task is to find a Hamiltonian operator $\hat{H}$ that fulfills two conditions: a) The time evolution of $|\Psi_i\rangle$ will yield $|\Psi_f\rangle$ after some time t and b) the time t is minimal in

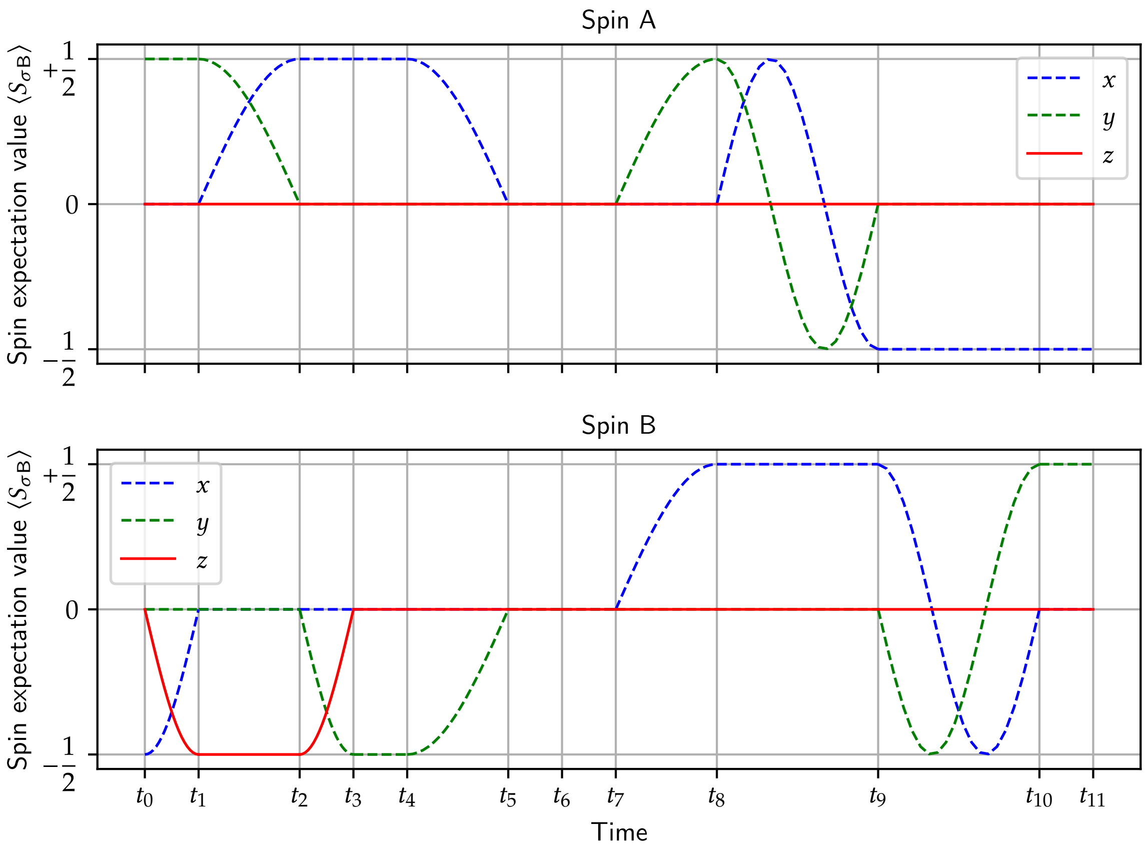


Figure 3.3.: Action of the SWAP gate pulse sequence Eq. (3.2.12) on the initial state $|\uparrow_y\rangle \otimes |\downarrow_x\rangle$. Shown are the expectation values for all spatial projections S_σ, ($\sigma \in \{x, y, z\}$), for the two qubits over time. At the end of the sequence, the information on the two qubits is swapped and they form the state $|\downarrow_x\rangle \otimes |\uparrow_y\rangle$.

the sense that there exists no other Hamiltonian that can connect the two states in less time. We will discuss quantum optimal control later in Chap. 5 as a method for finding NMR pulse sequences.

However, there is an easy way of shortening longer sequences. The method is to sort the pulses so that two pulses that correspond to the same direction and subspace end up next to each other. We can then join these two operations by carrying out just one pulse with a longer or shorter application time. In order to sort these sequences, we have to compute the pairwise commutators of the matrix representation of single pulses. We can swap two matrices only if they commute. Lee *et al.* [21] automate this procedure and show that Eq. (3.2.11) can be reduced to the eleven-pulse sequence

$$\begin{aligned} U_{\mathrm{SWAP}} = & R_{y\mathrm{B}}(-\pi/2)\, R_{z\mathrm{B}}(\pi/2)\, R_{z\mathrm{A}}(\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{B}}(\pi/2) \\ & \cdot R_{y\mathrm{A}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{x\mathrm{A}}(\pi/2) \\ & \cdot R_{x\mathrm{B}}(-\pi/2)\, R_{z\mathrm{AB}}(\pi)\, R_{y\mathrm{B}}(\pi/2)\,. \end{aligned} \tag{3.2.12}$$

They find that, for a sequence of this length, this is not the unique solution. Although this is the shortest-length sequence that they could find, they are not able to prove that none exists. Both the quantum optimal control variational ansatz and the simpler method based on commutators are further discussed in Chap. 4 and Chap. 5. The effect of the given SWAP pulse sequence in Eq. (3.2.12) is shown in Fig. 3.3 for the initial state

$$|\Psi_\mathrm{i}\rangle = |\uparrow_y\rangle \otimes |\downarrow_x\rangle = |\uparrow\uparrow\rangle - |\uparrow\downarrow\rangle + \mathrm{i}\,|\downarrow\uparrow\rangle - \mathrm{i}\,|\downarrow\downarrow\rangle \ . \tag{3.2.13}$$

If one traces the expectation values for the three spatial spin projections through the sequence, one indeed finds that the final state

$$|\Psi_\mathrm{f}\rangle = |\downarrow_x\rangle \otimes |\uparrow_y\rangle = |\uparrow\uparrow\rangle + \mathrm{i}\,|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle - \mathrm{i}\,|\downarrow\downarrow\rangle \tag{3.2.14}$$

is the initial state with the two qubits swapped, just as expected.

3.2.2. Heisenberg model

Since the Ising interaction is only a limit of the Heisenberg interaction between spins, we will now try to generalize qubit operations to the more general interaction. When three spatial components in the spin-spin interaction are included, the pulse sequences derived in Sec. 3.2.1 no longer hold. Thus, we have to develop a new approach to implement quantum gates with Heisenberg interaction via NMR control techniques. The first and simplest interaction model will be the isotropic Heisenberg interaction. Later, we will treat the more general case including anisotropy.

We remark here that the sequences and pulses based on the Ising interaction are well known and are documented by a number of authors. However, to the knowledge of this works author, no one has yet generalized NMR quantum computing pulse sequences to the Heisenberg interaction or any other, more general, spin interaction. Thus, this will be the central subject of my thesis. The remaining chapters will concentrate on the methods and ideas behind developing NMR quantum computing with a Heisenberg spin-spin interaction.

Isotropic Heisenberg model

The isotropic Heisenberg model interaction for two spins has the form

$$\hat{H}_\mathrm{c} = J\,\vec{\hat{S}}_\mathrm{A}\vec{\hat{S}}_\mathrm{B} = J\left(\hat{S}^x_\mathrm{A}\hat{S}^x_\mathrm{B} + \hat{S}^y_\mathrm{A}\hat{S}^y_\mathrm{B} + \hat{S}^z_\mathrm{A}\hat{S}^z_\mathrm{B}\right) , \tag{3.2.15}$$

where J is the coupling strength as in Chap. 2. It was shown that the evolution of a two-spin system under this coupling Hamiltonian can be described by the propagator

$$R_{\mathrm{AB}}^{\mathrm{XXX}}(Jt/\hbar) = \mathrm{e}^{\mathrm{i}\,\hat{H}_{\mathrm{c}}t/\hbar} = \mathrm{e}^{-\mathrm{i}\,Jt/4\hbar}\begin{pmatrix} \mathrm{e}^{\mathrm{i}\,Jt/2\hbar} & & & \\ & \cos(Jt/2\hbar) & \mathrm{i}\sin(Jt/2\hbar) & \\ & \mathrm{i}\sin(Jt/2\hbar) & \cos(Jt/2\hbar) & \\ & & & \mathrm{e}^{\mathrm{i}\,Jt/2\hbar} \end{pmatrix}. \tag{3.2.16}$$

We can omit the overall phase factor $\mathrm{e}^{-\mathrm{i}\,\omega t/4}$ because it has no physical consequences. We can see immediately that, for $Jt/\hbar = \pi$, this matrix is exactly the SWAP gate

$$R_{\mathrm{AB}}^{\mathrm{XXX}}(\pi) = U_{\mathrm{SWAP}} = \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix}, \tag{3.2.17}$$

where we have again omitted an overall phase factor $\mathrm{e}^{\mathrm{i}\,\pi/2} = \mathrm{i}$ on the right side. It is very convenient that we only need to let the two-spin system freely evolve for a time $t = \pi\hbar/J$ to implement the SWAP gate. Remember that it took eleven pulses to implement the same gate using the Ising interaction. Thus, using the Heisenberg interaction looks very promising at first sight. However, we will soon realize that things are not as simple as they seem at first sight, as motivated by the simple implementation of the SWAP gate.

The SWAP gate, alone or combined with single qubit rotations, does not form a universal set of gates for quantum computing. This is easy to see as neither the SWAP gate nor single qubit rotations can create entanglement. We have seen in the Sec. 3.1, that this is a crucial property of a universal set of quantum gates. Thus, the SWAP gate is not sufficient. While it is trivial to construct the SWAP gate from three CNOT gates, as shown in Eq. (3.2.10), the inverse operation is not possible. Therefore, we have to search for a way to find a universal set of quantum gates that can be implemented by NMR pulse sequences using the Heisenberg interaction.

As mentioned before, the search for such a sequence is a very demanding problem. Considering the large number of degrees of freedom, the space of possible sequences increases drastically with the length of the sequence. Even if we restrict the search to discrete angles of rotation, as seen in Eq. (3.2.3), the possibilities still grow quickly with the length. Imagine that we limit the rotation angles to $\pm\pi$ and $\pm\pi/2$. Even then, for a five-pulse sequence, there are 1.5 million different sequences. For an eight-pulse sequence, this rises to 20 million. Realizing that, in order to check a sequence for desired properties, we have to calculate the product of multiple matrices for each sequence, this amounts to a large amount of effort. We will discuss methods for finding certain sequences in Chap. 4.

For the time being, it shall just be noted, that I, the author of this work, was not successful in finding any sequence of length eight or lower that implements the CNOT gate using the Heisenberg interaction. This search was limited to the above-mentioned discrete angles. Therefore, it cannot be concluded that no such sequences in that range of length exist. It cannot be ruled out that there might be a sequence with odd combinations of angles that represents the CNOT gate, or at least some operation that is sufficiently close to it.

The set of CNOT and single-qubit gates is not the only universal set of quantum gates in the two-qubit regime. As we have already found, implementing the SWAP gate is quite trivial using the Heisenberg interaction. It is thus natural to search for a supplemental gate that can enlarge the set to a universal set. Ramelow *et al.* [22] show in their work that there is a set of gates called matchgates that form a universal set of quantum gates if combined with the SWAP gate. Matchgates are two-qubit unitary transformations

$$M = \begin{pmatrix} a_{11} & 0 & 0 & a_{12} \\ 0 & b_{11} & b_{12} & 0 \\ 0 & b_{21} & b_{22} & 0 \\ a_{21} & 0 & 0 & a_{22} \end{pmatrix}, \tag{3.2.18}$$

where the singe-qubit unitary transformations

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \tag{3.2.19}$$

are themselves in U(2) and obey $\det A = \det B$. These sub-matrices A and B act on the even- and odd-parity subspaces, respectively. As with the CNOT gate, I, the author, was not successful in finding NMR sequences using the Heisenberg interaction that can simulate the full set of matchgates.

The failure in finding working pulse sequences raises the suspicion that none might exist at all. These thoughts will be discussed in Chap. 5. One idea is to examine the symmetry that the Ising and Heisenberg interactions each obey. The continuous rotational symmetry of the isotropic Heisenberg interaction might be the reason for not being able to construct the aforementioned gates. One idea is thus to examine using an interaction that breaks this symmetry.

XXZ interaction

Taking one step away from an isotropic interaction, we will first take one direction of the spin interaction to deviate from the other two directions in strength. From Eq. (2.3.18) we recall the Heisenberg XXZ coupling Hamiltonian for two spins,

$$\hat{H}_{\mathrm{c}} = J\left(\Delta \hat{S}^z_{\mathrm{A}} \hat{S}^z_{\mathrm{B}} + \frac{1}{2}\left(\hat{S}^+_{\mathrm{A}} \hat{S}^-_{\mathrm{B}} + \hat{S}^-_{\mathrm{A}} \hat{S}^-_{\mathrm{B}}\right)\right) = \frac{J}{4} \begin{pmatrix} \Delta & & & \\ & -\Delta & 2 & \\ & 2 & -\Delta & \\ & & & \Delta \end{pmatrix}, \tag{3.2.20}$$

where Δ parameterizes the anisotropy. The coupling rotation propagator is then

$$R_{\mathrm{AB}}^{\mathrm{XXZ}}(Jt/\hbar) = \begin{pmatrix} \mathrm{e}^{2\,\mathrm{i}\,J\Delta t/\hbar} & & & \\ & \cos(Jt/\hbar) & \mathrm{i}\sin(Jt/\hbar) & \\ & \mathrm{i}\sin(Jt/\hbar) & \cos(Jt/\hbar) & \\ & & & \mathrm{e}^{2\,\mathrm{i}\,J\Delta t/\hbar} \end{pmatrix}, \tag{3.2.21}$$

where we have again omitted an overall phase factor because it has no physical relevance. If we compare Eq. (3.2.21) to Eq. (3.2.16), we see that we now have an extra degree of freedom in the first and last diagonal entries. We can interpret the anisotropy Δ as a factor that shifts the even parity subspace relative to the odd parity subspace. We remember that this was one of the features of the matchgates in Eq. (3.2.18).

Indeed, by choosing the anisotropy to be $\Delta = 2$, it is possible to find a sequence for the CNOT gate that only uses single qubit rotations and the XXZ rotation. Here $\Delta = 2$ means that the z components of the spin interact twice as strongly as the other components. In some sense, this is an interaction intermediate between the Ising model and the isotropic Heisenberg model. The sequence found for two qubits reads

$$U_{\mathrm{CNOT}} = R_{x\mathrm{B}}(\pi/2)\, R_{\mathrm{AB}}^{\Delta=2}(\pi/2)\, R_{y\mathrm{A}}(\pi/2)\, R_{z\mathrm{B}}(\pi/2)\, R_{\mathrm{AB}}^{\Delta=2}(-\pi/2)\, R_{z\mathrm{A}}(-\pi/2)\,. \tag{3.2.22}$$

The matrix representation is calculated just as in Eq. (3.2.4). We notice several differences to the Ising sequence of the CNOT in Eq. (3.2.3). First, this sequence needs six pulses. It is not possible to construct a sequence of less than six pulses if we only include the aforementioned discrete rotation angles. Secondly, we need to apply the interaction rotation twice. Again, it is not possible to reduce it to fewer interaction pulses. It is interesting to note that, for $\Delta = 2$, it is also not possible to find a sequence for the CNOT gate with seven or eight pulses that makes use of only one coupling rotation pulse.

The finding of this sequence shows that it is indeed possible to have a universal set of quantum gates, implemented as NMR control sequences with a generalized Heisenberg interaction. Whether or not it is possible to find such a set of sequences in the isotropic case remains unclear. This will be discussed more extensively in Chap. 5.

As a side note, in the search for a CNOT sequence in the isotropic model, a sequence using a mixture of the isotropic and anisotropic interactions was found. Namely, the sequence

$$U_{\mathrm{CNOT}} = R_{y\mathrm{B}}(-\pi/2)\, R_{\mathrm{AB}}^{\Delta=2}(\pi/2)\, R_{\mathrm{AB}}^{\Delta=1}(\pi/2)\, R_{y\mathrm{B}}(\pi/2)\, R_{z\mathrm{A}}(\pi/2)\, R_{x\mathrm{B}}(\pi/2) \tag{3.2.23}$$

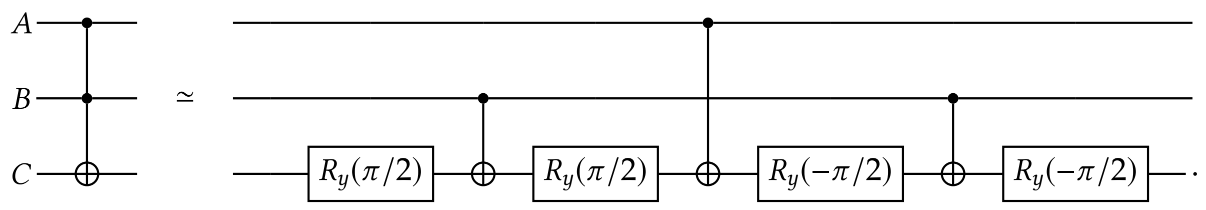
uses one isotropic interaction and one anisotropic one with $\Delta = 2$. The question arises, if this combination is relevant for real physical systems. After all, by having two different interactions, we assume that the systems environment has changed from one interaction pulse to the next. However, this is actually something that can be realized by, for example, manipulating a lattice structure externally. Note here that the quench-like sequence $R_{\mathrm{AB}}^{\Delta=2}(\pi/2)\,R_{\mathrm{AB}}^{\Delta=1}(\pi/2)$ does not have the same effect as gradually increasing the anisotropy like in Eq. (2.3.21).

3.3. Universal three-qubit-gate

Having found sequences for the implementation of a universal set of quantum gates with two qubit operations, we will now take one step further and investigate three qubit operations to close out this chapter.

In contrast to classical computing, there is no universal two-qubit quantum gate. This means that a universal set of gates that is capable of all quantum computations on two qubits has to have more than one member. An example of a universal set of gates for two qubits is the set of the CNOT gate combined with the single qubit rotations.

There is, however, a universal three-qubit quantum gate. It forms a universal set consisting of just one member: the *Toffoli gate*, which can also be described as a controlled-controlled-NOT-gate (CCNOT). It flips qubit C if and only if both qubits A and B are in state 1. One can implement any quantum circuit using networks of the Toffoli gate. The Toffoli gate can be written in terms of two and one qubit gates as [23]



It requires only CNOT gates and single-qubit rotations $R_y(\theta)$. The sequence of pulses can, of course, be assembled by simply concatenating the single-qubit and two-qubit sequences of the respective gates. However, Lee *et al.* [21] show that for the Ising interaction, the sequence can be reduced to

$$\begin{aligned} U_{\mathrm{TOFF}} = & R_{z\mathrm{A}}(-\pi/4)\,R_{z\mathrm{B}}(-\pi/4)\,R_{z\mathrm{AB}}(\pi/2)\,R_{y\mathrm{C}}(\pi/2) \\ & \cdot R_{z\mathrm{C}}(\pi/4)\,R_{z\mathrm{AC}}(-\pi/2)\,R_{z\mathrm{BC}}(-\pi/2)\,R_{y\mathrm{C}}(\pi/2)\,R_{z\mathrm{AC}}(\pi/2) \\ & \cdot R_{x\mathrm{C}}(\pi/2)\,R_{z\mathrm{BC}}(-\pi)\,R_{x\mathrm{C}}(-\pi/2)\,R_{z\mathrm{AC}}(-\pi) \end{aligned} \tag{3.3.1}$$

using only 13 rotations. The effect of this sequence on the initial state $|{\downarrow\downarrow\uparrow}\rangle$ is depicted in Fig. 3.4. After the sequence is finished, the last qubit ends up in the opposite state because both the first and the second qubit initially were in the 1 state ($|{\downarrow\downarrow}\rangle_{\mathrm{AB}}$).

For the XXZ interaction with $\Delta = 2$, it is easy to assemble the sequence for the Toffoli gate by combining sequences for the CNOT gate with single qubit rotations.

To summarize, we have seen that it is fairly easy to devise a universal set of quantum gates using NMR control techniques using the Ising interaction. In particular, the key

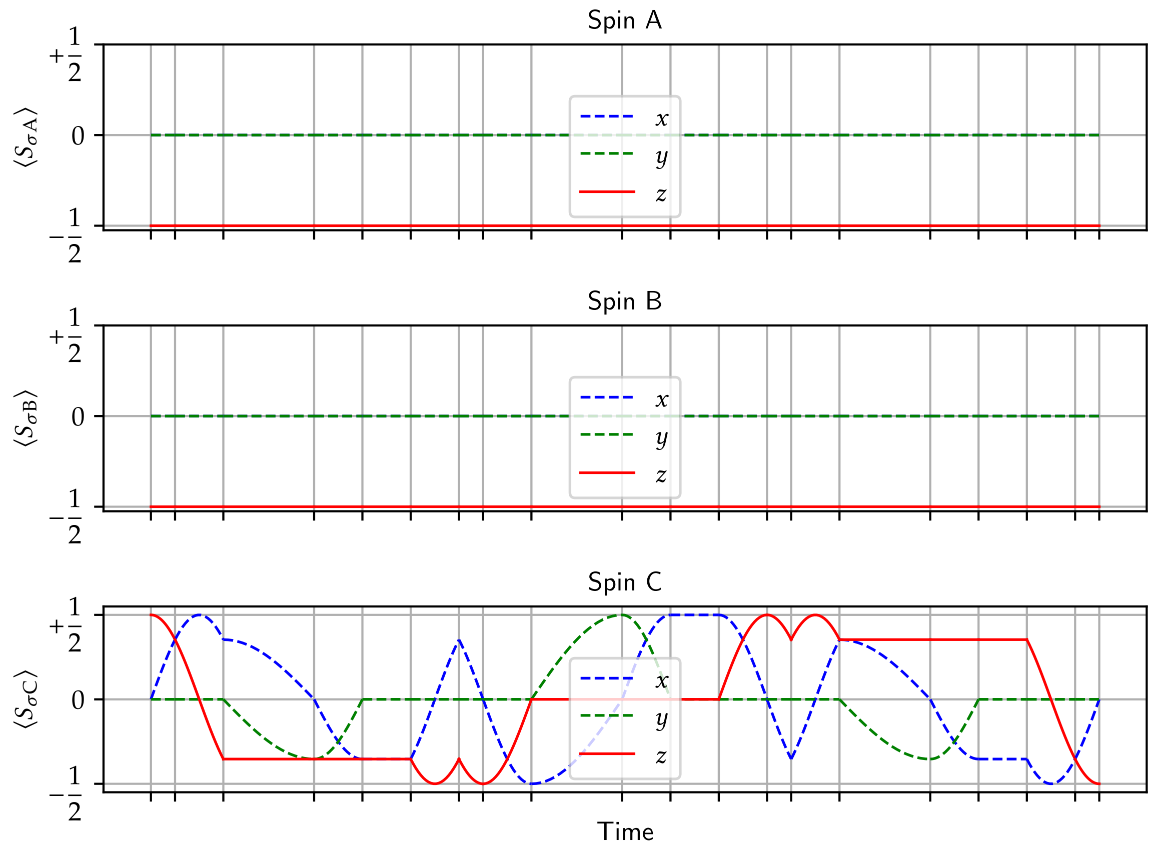


Figure 3.4.: Evolution of the expectation values during the sequence of the Toffoli-gate. The $|{\downarrow\downarrow\uparrow}\rangle$ initial state is rotated to $|{\downarrow\downarrow\downarrow}\rangle$ because both the first and the second qubit are in the state 1 ($|{\downarrow\downarrow}\rangle_{AB}$).

element of the set, the CNOT gate, can be encoded by five NMR pulses, one of them an evolution due to the spin-spin interaction.

For the isotropic Heisenberg model, I, the author, have not found a pulse sequence, either for the CNOT gate or for matchgates. It remains unclear if symmetry or some other property excludes such a sequence.

For the anisotropic Heisenberg interaction, however, we have seen that a pulse sequence that implements the CNOT gate exists. In combination with single-qubit rotations, this forms a universal set of quantum gates. The generalized Heisenberg interaction can thus be used to implement universal quantum computing. In Chap. 4, we will discuss in detail the methods that were used to obtain these results. The results themselves will be discussed in Chap. 5.

4. Methods

In the last chapter, we have found that we can implement quantum gates using NMR pulse sequences. In addition, we have shown that universal quantum computing is possible within this scheme. However, a universal set of quantum gates was only found for the Ising and the anisotropic Heisenberg interaction. While the pulse sequences for the Ising model can be found in the literature, in this chapter, we will discuss the algorithms that I developed for finding analogous sequences for the Heisenberg interaction.

Before we come to the algorithms, we will briefly cover the methods used to visualize the propagation of the quantum states. This was done with the *QuTiP* Python library [24], which offers a convenient framework for quantum calculations. The program uses an ordinary differential equation solver to calculate the time evolution of quantum states.

We will then discuss my approach to finding possible NMR pulse sequences to implement quantum gates. First, we will have an introduction by covering a very simple and general algorithm that searches though all possible sequences. This is not very efficient but well-suited to illustrate the general idea of the search.

Finally, we will learn about the optimizations, that I have added to the base algorithm. These optimizations include both algorithmic and numeric improvements. On the one hand, the optimized algorithm does not have to carry out all matrix multiplications, the most costly part of the program. On the other hand, fast data structures such as hash tables have been implemented. Finally, the full method has been adapted to utilize multi-threading. Multi-threading is not natively supported by Python, but there are workarounds to nevertheless use its advantages.

In the next chapter, an alternative approach to these tasks is mentioned. This highly advanced technique would however have far exceeded the scope of this thesis, so it was not implemented here.

4.1. Propagation and visualization of quantum systems using QuTiP

The visualization of the effect of NMR pulses on quantum states is a key element of understanding how they might be used to form quantum gates. In particular, if we know what the effect of a quantum gate such as CNOT is, we can try to reverseengineer its code to form NMR pulse sequences. If we also know how each pulse changes the quantum state, we can then try to find the desired sequence by combining pulses in a visualization scheme such as the Bloch sphere. The Bloch sphere is a three-dimensional representation of a spin and is much more intuitively accessible than a matrix representation.

For the visualizations of quantum systems in this thesis (e.g., Fig. 3.1), I have chosen the Python module QuTiP, which provides a complex environment for the simulation of quantum systems [24]. QuTiP is an open-source framework that has proven its versatility in many fields of quantum mechanics. In particular, simulation of quantum computing and plotting in the Bloch sphere is what it was mainly used it for here.

The main task was to visualize how the Hamiltonian of a certain NMR pulse or pulse sequence propagates an arbitrary quantum state. The computationally difficult problem in simulating quantum objects is solving the Schrödinger equation

$$\frac{\mathrm{d}}{\mathrm{d}t} |\Psi(t)\rangle = \mathrm{i}\,\hat{H}\,|\Psi(t)\rangle \ . \tag{4.1.1}$$

The Schrödinger equation is an ordinary differential equation (ODE). It is thus natural that QuTiP uses a long-established ODE solver to find the time evolution of quantum states. The QuTiP Schrödinger equation solver `qutip.sesolve()` calls the SciPy generic ODE solver `scipy.integrate.ode()` [25]. The latter uses the C-method *zvode*, which is the complex valued version of the method *vode*. Essentially, it all boils down to *vode*, which is a solver that uses variable-coefficient Adams-Moulton and backward differentiation formula methods in Nordsieck form to find the evolution of the state [26]. We will not discuss the intrinsic workings of the QuTiP module any further because it is only peripherally relevant to the key elements of this thesis.

A convenient way of tracking the evolution of a spin quantum state, as described in Chap. 2, is to plot the expectation values of hypothetical measurements along all spin axes. The brute-force method would be to compute the evolved state for all of a set number of time steps. Then one could compute the expectation values of this state along all axes. However, this is very costly computationally. Hence, the time evolution method `qutip.sesolve()` only computes the expectation values and not the states, as they are not required for visualization.

The second great feature of QuTiP is that one can directly take the results from the Schrödinger equation solver and feed them to another method, for example, one that plots them in a Bloch sphere representation. The data can then be visualized as Bloch vectors or points inside the sphere. This Bloch sphere plotting was extremely helpful for understanding the action of both the Ising and the Heisenberg interactions propagation, as depicted in Fig. 2.3.

While the QuTiP module is very versatile and can be used for many problems in quantum mechanics, I chose not to use it to implement the search algorithm in this thesis. The main reason for this is that, because of its versatility, QuTiP uses a lot

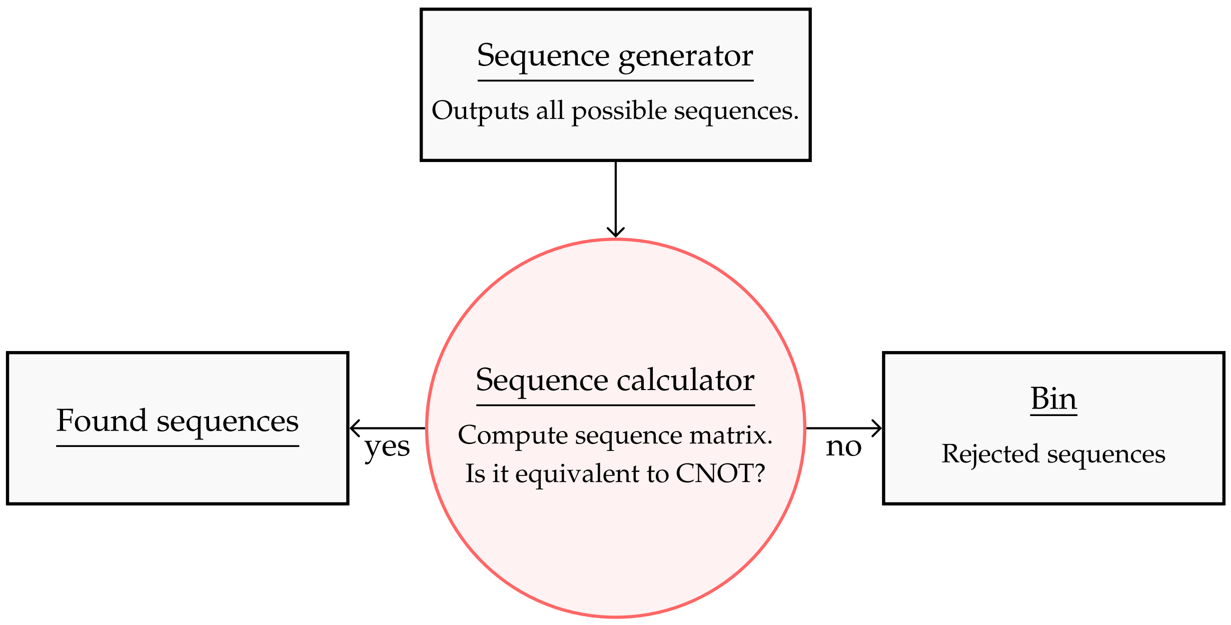


Figure 4.1.: Simplest idea of the algorithm to find CNOT sequences. A python generator generates an encoding of every possible sequence one by one. The encoding is then transformed into the matrix representation via multiple matrix multiplication. If the matrix is the CNOT gate, then the sequence is saved under "found sequences".

of data overhead to represent quantum states and operators. In order to search as many possible NMR sequences as possible, I used slimmer built-in methods of Python, which are more efficient in both time and space. The disadvantage is a less intuitive representation of the physics within the program.

4.2. The search for quantum gates in terms of NMR pulse sequences

We will now discuss how the new NMR pulse sequences in Chap. 3 were found. For that, we will walk through a series of steps that check whether or not a given sequence has the desired properties. A very simple summary of the algorithm is given in Fig. 4.1. The upper method encodes all possible sequences in a way that is easy to process. The sequence encoding is then translated to a matrix representation using multiple matrix multiplication. The resulting matrix is the operator that represents the full sequence of NMR pulses. It is saved if and only if it fulfills the requirement of being equivalent to the CNOT gate. The method was completely analogous for finding matchgates.

In this simplest version of the search algorithm, the execution of the matrix multiplications takes over 99% of the computing time. Although it is only a multiplication of up to nine 4×4-matrices, the sheer number of possible sequences makes this part very expensive. It is thus natural to search for ways to avoid having to compute the matrix representation for as many sequences as possible.

4.2.1. Encoding of sequences

The first thing we have to ask ourselves is how we translate the pulse sequences such as that in Eq. (3.2.3) efficiently into code. I implemented each pulse as a tuple of parameters, i.e.,

```
        ('y', 0, 3.14),
```

where the first entry denotes the direction or type of the pulse's rotation, and the second entry represents which qubit which the pulse acts on. Note that this itself can be a tuple when describing a coupling rotation. The last element is the angle of rotation, which can be positive or negative. For example, the rotation $R_{\mathrm{AB}}^{\mathrm{XXX}}(-90°)$ would be encoded as

```
        ('XXX', (0, 1), -1.57) .
```

A full sequence consists of the consecutive application of its pulses. Thus, the encoding of a full sequence is just a tuple of single-pulse tuples. The causality runs from left to right, that is, its direction is the opposite of the matrix representation. The pulse in the first entry of the sequence is applied first. The full sequence of the CNOT gate on spins A and B in the Ising model (Eq. (3.2.3)) is given by

```
        (('y', 1, pi / 2.), ('z', (0, 1), pi), ('z', 1, -pi / 2.), ('
    z', 0, -pi / 2.), ('y', 1, -pi / 2.)) ,
```

where the variable `pi` contains the number π.

Tuples were chosen as a data structure because it is important in a later part of the algorithm to have the sequences saved as immutable data structures. In Python, tuples are, in their functionality, essentially the same as lists. However, lists can be changed after their creation, while tuples remain unchanged until annihilation.

When creating sequences, one could simply create and save all possible permutations of pulses of the length under investigation. However, saving such a large amount of data is unnecessary and very expensive with regards to memory. Python does provide a way to avoid these shortcomings. Python *generators* are functions that yield elements one by one at each call. They do not have all the elements saved, but construct them one at a time, following the rules given to them. They are very light in memory because they only need to save the rules and not the elements themselves. A simple example is the generator of natural numbers

```
        def natural_numbers():
                n = 0
                while True:
                        yield n
                        n += 1
```

which yields the next natural number each time `natural_numbers()` is called. In this way, one could, in principle, do calculations on each integer, without ever saving more than one integer.

Using an analogous generator, the permutations of NMR pulse sequences are generated and passed on one by one. The generator only saves the set of pulses to be permuted and is hence much more memory-efficient than code that saves a full array of sequences. It is implemented with the Python-module *itertools*, which makes easy implementation of generator methods like permutations or combinations possible.

4.2.2. Testing sequences for universal gate application

The key element of the program is the method for testing the sequence. For each of the possible pulses, we already know the corresponding matrix representation from Chap. 2 and Chap. 3. One can then simply take the matrix for each pulse and execute the full multi-matrix multiplication, resulting in an operator matrix that represents the action of the full pulse sequences in the chosen basis.

After having obtained the resulting matrix, one only needs to check if the matrix does indeed represents the required quantum gate, be it the CNOT gate or a matchgate. If one is searching for the CNOT gate, it is useful to normalize all entries of the matrix so that the first entry is unity. This is allowed because an overall phase does not have any physical meaning in quantum mechanics. It is also worth noting that the comparison of two floating point numbers is a procedure which computers are quite bad at. Because of small fluctuations in the floating-point representation, one should test for strong similarity rather than exact equality.

Matrix multiplication is, in general, a problem that scales very badly. For the multiplication of two $n\times n$-matrices, the number of elementary operations (flops) is n^3 [27]. As we are treating 4×4-matrices, this means 64 flops for each multiplication. Hence, we need $64l$ flops to compute the full matrix, where l is the length of the sequence. Even though n is small in this case, computational difficulty still arises because we have to execute so many multi-matrix multiplications. After all, we have to carry out many millions of these multiplications, as shown in Chap. 3.

4.2.3. Optimizing the program for performance

Because of the inefficiency of the matrix multiplication, it would be desirable to carry out fewer of these operations, but without missing any matrices that we are searching for. We will now discuss several ways of trimming the program to execute fewer matrix multiplications. The general idea is that sequences are ignored before carryig out the multiplication if they are not a valid candidate for a CNOT or a matchgate. Fig. 4.2 displays a diagram of the optimized program. Two new procedures have been inserted before the sequence check compared to the algorithm in Fig. 4.1.

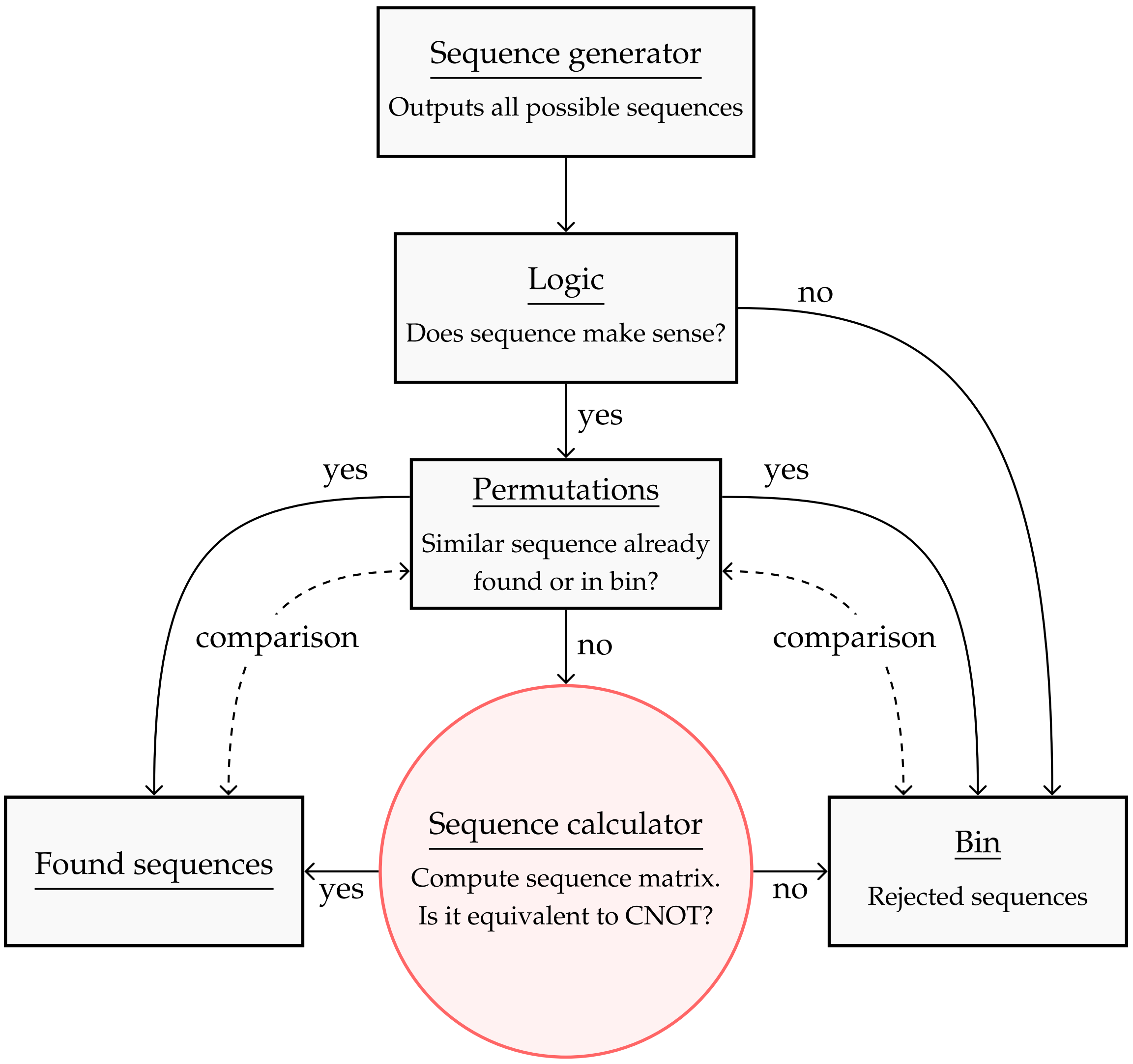


Figure 4.2.: Advanced algorithm for the finding of a CNOT sequence. The generator generates all possible sequences. Each sequence is then checked for logical attributes and may be rejected. Then, all known permutations of the sequence are calculated using the commutator relations. If one of these was already treated, the matrix of the sequence itself need not be calculated. The rest of the algorithm is the same as in Fig. 4.1

Logical examination

The first way to reduce the number of executed matrix multiplications is by checking the sequences for logic. That means we want to exclude sequences that we know do not yield the desired matrix by examining the sequence.

Consider the example sequence

$$R_{\mathrm{AB}}^{\mathrm{XXX}}(-90^\circ)R_{\mathrm{yA}}(90^\circ)R_{\mathrm{yA}}(180^\circ)R_{\mathrm{zB}}(-90^\circ) \tag{4.2.1}$$

and notice that the second and the third pulse are both rotations around the y-axis in the subspace of spin A. Two pulses of the same kind, albeit with different rotation angles, just represent a longer-acting pulse. This means that this sequence, though written down in terms of four pulses, is effectively just three pulses long. Because the approach of this thesis' search was increasing the length of the sequence, it is justified to discard this sequence. If it had been relevant, it would have already been found in the search for three-pulse sequences.

Consider a further example sequence,

$$R_{\mathrm{AB}}^{\mathrm{XXZ}}(-90^\circ)R_{\mathrm{zA}}(90^\circ)R_{\mathrm{zB}}(-90^\circ)R_{\mathrm{yA}}(90^\circ)\,, \tag{4.2.2}$$

where the first pulse is a rotation of spin A around the y-axis. Subsequently, there are only z and interaction rotations. For the CNOT gate in the S^z basis, we expect the spin A to return to the same substate after the sequence. However, if we assume that spin A does not start in the y direction, then it is rotated away from its initial position in the first pulse. As it is never rotated back, this cannot yield the CNOT gate for arbitrary initial states. Similar thoughts apply to the other axis directions.

A sequence can also be discarded for one of the above reasons without directly meeting these criteria. If we find consecutive pulses in a sequence that commute with each other, then we can switch the pulses and still have the same net action. By consequently permuting pulses, we may discover that equivalent sequences fall under the aforementioned dismissal criteria. After excluding all of these sequences, I was able to reduce the number of unnecessary matrix multiplications by about 25%.

Checking permutations of the pulse

The main reason for the program to have taken a long time was that a lot of the matrices that were computed had similar sequences, that were rejected and discarded before. The same applies for sequences that were put into the "found sequences" stack before. In the first version of the program, only those sequences that yielded the correct matrix were saved; rejected sequences were discarded. As we have just seen, many sequences may effectively be the same, albeit with permuted pulses. In order to compare to the already found or discarded sequences, they shall now be saved in some form.

Again, a number of different data structures can be used to save the found and discarded sequences. I selected Python's `set()` class, which is the best structure for quick lookups. The set class is the raw version of a dictionary in Python. While a dictionary contains keys and allocated values, the set only contains keys. Since we want to look for similar sequences in the set, that has already been saved, this look up has to be as efficient as possible. Sets (like dictionaries) are implemented using a hash function, i.e., they are hash tables by nature. Hashable structures are amongst the most efficient ways to look up elements because the computer only has to check one address in memory. Simply speaking, hash tables work similarly to a checklist. A set theoretically consists of all possible data structures that can be saved and a checkbox for each of the elements. If an element is contained in the set, there is a check in the element's entry. If one searches for an element inside a set, all one needs to do is to see if there is a check at the corresponding entry. A simple consequence of this architecture is that hash tables cannot have duplicate entries. In contrast, lists (or tuples) work on a pointer- or reference-based architecture. That is, each entry is a pointer to a place in memory with the corresponding object. If one wants to search for an element inside a list, in the worst case, one has to look up every single element's reference in memory.

Here we come back to the choice of tuples to encode the sequences. Eventually, we will want to save the sequences inside the set of found sequences or the set of discarded sequences. Sets only support immutable objects as entries. The reason for this is, simply speaking, that the entries are not saved as objects, but rather as hashes or hash values. These codes do only correspond to a particular data structure and cannot be altered. In contrast, the objects which the pointers of a list point to can be changed. There will then just be a different object at the pointer's address in memory. In summary, hash tables are the ideal choice for look ups and they only support non-mutable entries.

The workflow, which is sketched in Fig. 4.2, is then as follows. For each incoming sequence, all known permutations are calculated taking into account the commutation relations. Each of these permutations is then looked up in the found and discarded stacks. If it is already in one of these, then the sequence is itself discarded. Calculating the permutations of the tuple sequences is faster than executing the matrix multiplications by many orders of magnitude.

After having implemented all of the above optimizations, the execution time decreases dramatically. Initially, testing all pulse sequences of length six took around 30 minutes. With the aforementioned optimizations, this was reduced to around three minutes, i.e., by a factor of ten. However, for longer sequences, a different obstacle arose. Remember that I chose to implement the creation of the sequences as a Python generator in order to avoid saving all sequences beforehand. Saving the sequences to the discarded and found stacks recreates this problem of space. While keeping all those sequences in memory is no problem for short sequences, for the sequence length of eight pulses, the set of discarded sequences could no longer be saved within the random access memory (RAM) of 64 gigabytes. In order to compute longer sequences,

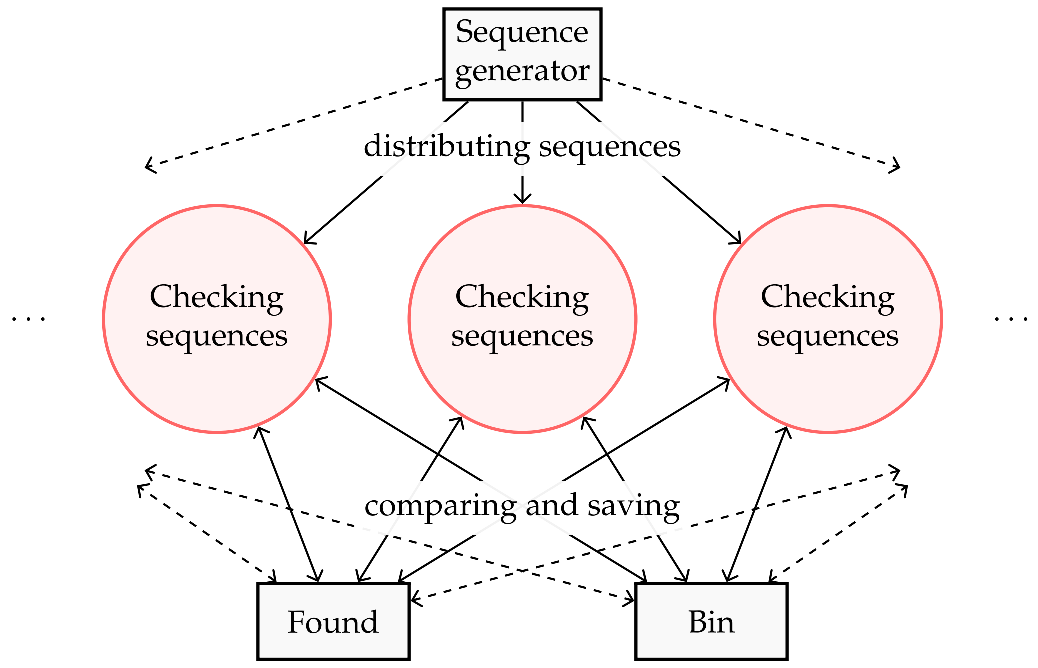


Figure 4.3.: Sketch of multiprocessing, applied to the program of Fig. 4.2. The generator distributes the sequences to many different processes, one on each processor. Each of the processes then calculates a portion of all sequences. All the processes run in parallel. If permutation checking is implemented, then the comparison with the found and discarded stacks leads to a cluttered communication. The latter can drastically diminish the performance of the program.

one has to find a good-trade off between the memory limitation and the execution time.

Multiprocessing

In order to expand the search for a Heisenberg XXX sequence by one more pulse, I eventually introduced multiprocessing into the program. Inherently, Python does not provide multiprocessing or even multithreading. That is, Python starts one thread by default and executes all the computations in this single thread.

However, there does exist a way to implement multithreading and also multiprocessing. The Python module `multiprocessing` offers ways to start more than one process on the CPU and is even able to distribute and manage those processes among all kernels. In this way, tasks can be executed in parallel on many processors as opposed to just one thread. The multiprocessing module provides manager objects that receive data, in this case, sequences, and then send it to the different processes for calculation. This distribution is done, so that the one process that is idle is always fed first. Using this procedure, one can minimize the idle time of all the processors. After computation has finished, each process communicates its results back to the manager, which resides in a parent process.

If one only needs to execute independent calculations, then multiprocessing is the ideal approach to increase performance. However, in the last subsection, we have seen that the search program was extended by incorporating some communication features that ensure that no two similar sequences are calculated twice. Communication is typically something that multiprocessing is not good at. We will now see the reason why.

When processes have to communicate with each other, they typically do it in a send-message-receive-message way. That is, the receiving process has to be idle for processing the input. If the process still has work to do, it may not receive the input for a long time. If the sender then wants to send another message, the communication channel would still be blocked by the first message. As the sender now has to wait until the channel is free again, it cannot continue with its own tasks. In this way, communication can significantly slow down the overall performance of a program.

In my program, each processor receives one or many sequences from the sequence generator and is then given the task to decide whether or not this sequence yields the desired matrix. In the last subsection, we have seen that one step of this checking is to look up all permutations of the sequence in the found and the discarded stacks. The sequence will actually be calculated only if it is not contained in those sets. In order to check against all found or discarded sequences, these sets have to be saved in a parent process, for example, the one which also houses the sequence distribution manager. Every process then has to communicate with the parent process, for each sequence to check, if a similar sequence has already been computed. A sketch of the multiprocessing evolution of the aforementioned program can be found in Fig. 4.3.

The amount of communication that is needed in such a simple implementation renders all the performance gains that we discussed so far void. The program would become even slower than the very first and rough version of the program. To understand this problem, we note the clutter of arrows in Fig. 4.3 that represent communication. The amount of communication can be reduced in the following way: the first step is to distribute larger chunks of sequences to the processes so that they do not have to communicate inputs and results so often. Secondly, each process can create its own intermediate found and discarded stacks and only compare its sequences to these sets. It will then receive the full stacks of found and discarded sets only once at the beginning of the process. In general, this will lead to more matrix computations because the process cannot check against all sequences calculated. We have to realize that, during the processes' work, other processes are constantly finding or discarding sequences that cannot be accessed by any other process. However, it lies in the nature of the sequence generator that successive sequences will be similar in shape. That is, they are created by a few permutations at most. This is exactly what the permutation optimization tries to sort out. Only after completion of the process will the intermediate sets be added to the global found and discarded stacks.

This modification of multiprocessing does indeed bring a performance enhancement compared to the single-threaded version of the program. In summary, we have two trade-offs between between factors that can enhance or diminish the performance of the sequence search. With regards to the found and discarded sets, these are: saving as few sequences as possible for memory efficiency versus saving as many sequences as possible for time efficiency. For the multiprocessing method, we have to weigh calculating and comparing as up-to-date as possible for maximum efficiency against communicating as little as possible to reduce the idle time. By carefully adjusting these factors, I was eventually able to extend the search for a Heisenberg XXX CNOT sequence to nine pulses. Nonetheless, under the assumption of the above discussed restrictions of the angles, no sequences of up to nine pulses were found that implement the CNOT operation.

In Chap. 5 we will debate wether the methods in this chapter are sufficient to find the desired sequences. We will find that the simplifications used for the sequence search are reasonable. Nevertheless, there are other, more advanced, methods that may make this search more efficient. One of them will be described briefly in the next chapter.

5. Discussion

We have now discussed all the implementations and approaches for finding NMR sequences for quantum computing that are part of this work. One important achievement was the discovery that universal quantum computing is possible using the anisotropic Heisenberg interaction. Specifically, I found a sequence of NMR pulses that implements the CNOT gate using the Heisenberg XXZ interaction with $\Delta = 2$. However, I was not able to find a CNOT sequence for the isotropic Heisenberg model.

In this chapter, we will discuss these results as well as possible methods to refine this search. First of all, we will investigate real NMR quantum computing experiments and discuss why they will, most likely, never work on architectures larger than 15 qubits. Subsequently, we will discuss wether or not it is even possible to implement a CNOT gate in using the Heisenberg XXX interaction using only NMR manipulation techniques. Finally, several ideas to improve the performance of the search algorithm described in Chap. 4 will be presented, including various small improvements as well as a variational approach that could form the basis for a new search algorithm.

5.1. Application in real experiments

In the previous chapters, we have discussed NMR quantum computing in a rather simplified and theoretical manner. We will now discuss some obstacles and restrictions that occur in real experimental NMR quantum computing.

5.1.1. Ensemble quantum computing

While other potential quantum computers rely on isolating individual ions or photons from its environment to overcome decoherence, a different approach is used in an NMR quantum computer. The nuclear spins are well-suited for quantum computing because their coherence time is typically much longer than the time scale of decoherence that arises from thermal motion. The coherence time can reach up to thousands of seconds. Even at room temperature, this property can be used to realize a quantum computer.

Since it is very difficult to isolate single molecules, an NMR quantum computer uses a method called *ensemble quantum computing*. Typically, it will run on a sample of $\sim 10^{23}$ molecules, for example, molecules dissolved in a liquid. Each of these molecules represents a separate quantum computing unit with a certain number of qubits, embodied as nuclear spins (see Fig. 1.1a for a sketch).

The actual quantum information is not directly realized as the state of one molecule. Such a state would not be coherent because of the thermal motion of the molecules at room temperature. Since an NMR measurement can only detect macroscopic magnetic fields, the magnetization of the molecules would average to zero. The quantum states are instead coded in small deviations from the mean value of the total spin magnetization that arises due to an external magnetic field. This procedure makes pseudo-pure states at room temperature possible. The disadvantage of this technique is that one needs to have a larger number of spins with only a part of them acting as the computational qubits.

Typically, the number of computational qubits is less than half of the number of the physical qubits, which are spins [14]. For larger systems, there are methods to gain more computational qubits at the cost of losing signal strength. The signal strength is the biggest weakness of ensemble quantum computing. It roughly scales as 2^{-N}, where N is the number of (computational) qubits. The conseqence is that with modern signal-detection methods, it is only possible to resolve of the order of ten qubits. This corresponds to a state space with dimension $2^{10} \approx 10^3$. Beyond this, other elaborate information packing schemes exist that can enlarge the computational space, but only with polynomial scaling. In contrast, the state space grows exponentially with the number of qubits.

NMR quantum computing has both advantages and disadvantages compared to other implementations. On the one hand, no other method grew so fast in qubit size in the late 1990's and early 2000's, which was the beginning of the age of real implementations of quantum computers. NMR quantum computing experiments led to the first experimentally implemented quantum gates and algorithms, such as the Shor prime factorization algorithm in 2001 [17]. On the other hand, the potential scaling of NMR quantum computers is so unfavorable that it is unlikely that any experiments with more than 15 qubits will be realized unless some revolutionary new method is discovered. Consequently, scientists are presently focusing on other architectures such as cold trapped ions or topological qubits.

Recall that some of the ideas and theories that were presented in chapters 2 and 3 are greatly simplified; therefore we should discuss their validity for real experiments. In the following, we will discuss two obstacles to real implementation as well as solutions to overcome them.

5.1.2. Refocusing

In the preceding chapters, we have assumed that we can switch the rotational pulses on and off at will. For a single qubit-rotation via rf-pulses, this is indeed true to some extent, but, as we will see, there is a limitation to this idealization. However, we cannot switch the interaction term, $J\hat{S}_z\hat{S}_z$, off because we cannot break the molecule up and distance the nuclear spins. Thus, we need a way to control the interaction term in the Hamiltonian so that we can still implement the aforementioned sequences. If we cannot turn off the interaction, an alternative is to reverse it repeatedly so that we have a vanishing interaction evolution on average. We note that a time inversion of the interaction

$$R_{\phi\mathrm{A}}(180^\circ)\, R_{z\mathrm{AB}}(\theta)\, R_{\phi\mathrm{A}}(-180^\circ) = R_{z\mathrm{AB}}(-\theta) \tag{5.1.1}$$

can be achieved with the Ising interaction by a 180° pulse along any axis $\vec{\phi}$ [14]. By repeatedly applying reversing pulses, we can let the system evolve back and forth with respect to the interaction term in the Hamiltonian. On average, the state will remain unchanged. This technique is called *refocusing* and is essential to implementing circuits such as the CNOT gate (Eq. (3.2.3)) in practice.

5.1.3. Pulse shaping

We have learned how to effectively eliminate the coupling evolution via refocusing, but what happens if we are simultaneously applying rotations to the individual nuclei A and B? After all, the interaction is still present during the single qubit rotations in Eq. (5.1.1). Will the coupling term destroy the ability to implement perfect single-qubit rotations such as $R_{x\mathrm{A}}(90^\circ)$? In Fig. 5.1, the rotation due to the rf-field is illustrated with and without the coupling term in the Hamiltonian. The coupling strength J is highly exaggerated for illustrative purposes. We can observe that the two evolutions deviate by only a small amount. We assume, without loss of generality, that we want to apply a rotation around the y-axis. If we apply an rf-field with its effective B-field in the y-direction, the Hamiltonian, including the Ising interaction, reads

$$\hat{H} = \gamma B \hat{S}_{y\mathrm{A}} + J \hat{S}_{z\mathrm{A}} \hat{S}_{z\mathrm{B}} = \frac{\hbar}{2} \begin{pmatrix} J\hbar/2 & 0 & -\mathrm{i}\gamma B & 0 \\ 0 & -J\hbar/2 & 0 & -\mathrm{i}\gamma B \\ \mathrm{i}\gamma B & 0 & -J\hbar/2 & 0 \\ 0 & \mathrm{i}\gamma B & 0 & J\hbar/2 \end{pmatrix}, \tag{5.1.2}$$

where the diagonal elementy induce only a small deviation from $R_{x\mathrm{A}}(90^\circ)$ because $J\hbar/2 \ll \gamma B$.

What if it is necessary to eliminate this small disturbance? Analytically, it is not possible to discard the interaction or separate it from the single qubit pulses. However, there are certain techniques that can improve the pulse in practice. The most effective method is to shape the pulse of the rf-field in a particular way. There are several different shapes that have different effects and may be used for a variety of corrections, for example, instead of the rectangular shape that was implicitly assumed up to now, the pulse could be shaped by a gaussian envelope, which is much more realistic in general. Vandersypen *et al.* [18] give a very good overview of these *pulse shaping* techniques and many other methods for practical NMR quantum computing.

We have seen so far that the two last mentioned methods are a kind of error correc-

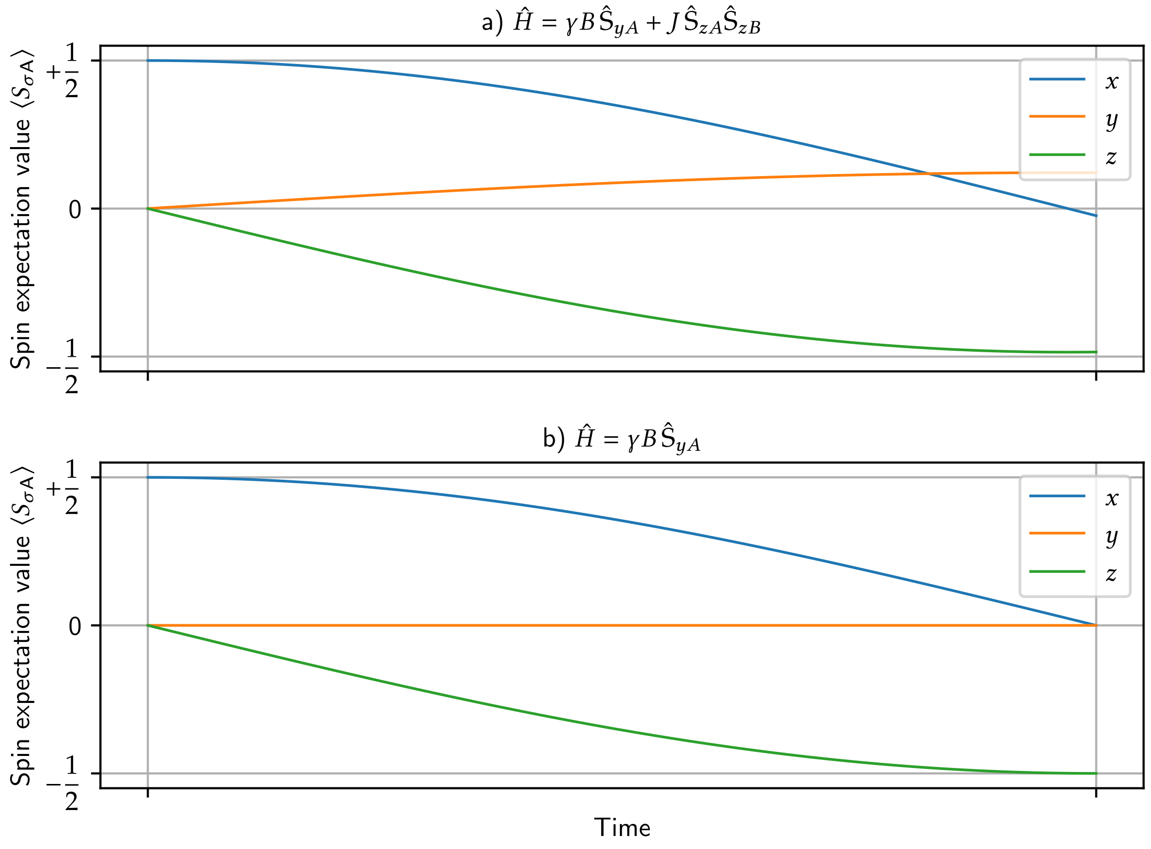


Figure 5.1.: Comparison of an undisturbed y-rotation of qubit A and the same rotation in the presence of interaction between the spins. The initial state $|\uparrow_x\rangle \otimes |\uparrow_z\rangle = (|\uparrow\uparrow\rangle + |\downarrow\uparrow\rangle)/\sqrt{2}$ is rotated by 90° around the y-axis. a) In the presence of interaction, the final state is a small deviation from $|\downarrow\uparrow\rangle$. Due to the coupling disturbance, the state experiences a nutation on top of the y-rotation, that leads to a deviation from the theoretically predicted final state. b) For comparison, the evolution without the interaction term in the Hamiltonian is shown. The largest deviation can be observed in expectation value of S_z.

tion in the machine language of NMR pulse sequences. However, we might also take them as opportunity to find interesting new pulse sequences. Remember that I did not succeed in finding pulse sequences with the Heisenberg interaction that can be used to form a universal set of quantum gates. The disturbance of single-qubit pulses by the spin-spin interaction could be used to change the symmetry properties of the available pulses in order to overcome symmetry dependent obstacles. We will discuss some ideas along these lines in Sec. 5.2.

5.2. Symmetry considerations

The fact that an exhaustive search of pulse sequences of up to nine pulses could not find a sequence implementing the CNOT gate, raises the question of wether it is at all possible to find such a sequence with the isotropic Heisenberg interaction. It would thus be useful to prove that NMR quantum computing in the isotropic Heisenberg model is not capable of implementing a CNOT gate. A more comprehensive statement would be that universal NMR quantum computing cannot be implemented using the Heisenberg interaction. Several approaches to find proofs for these statements were tried in the scope of this thesis. Two of them shall be presented here.

The first approach is to examine the subspaces of the Hilbert space for two qubits of different parity. The Heisenberg interaction propagator,

$$R_{\mathrm{AB}}^{\mathrm{XXX}}(Jt) = \mathrm{e}^{\mathrm{i}\,J\vec{S}_{\mathrm{A}}\vec{S}_{\mathrm{B}}t} \simeq \begin{pmatrix} \mathrm{e}^{\mathrm{i}\,Jt/2} & & & \\ & \cos(Jt/2) & \mathrm{i}\sin(Jt/2) & \\ & \mathrm{i}\sin(Jt/2) & \cos(Jt/2) & \\ & & & \mathrm{e}^{\mathrm{i}\,Jt/2} \end{pmatrix} \tag{5.2.1}$$

does not mix the subspaces of different parity. This means that it cannot change the quantum state from the odd-parity subspace of the Hilbert space, spanned by $|\uparrow\downarrow\rangle$ and $|\downarrow\uparrow\rangle$, to the even parity subspace, spanned by $|\uparrow\uparrow\rangle$ and $|\downarrow\downarrow\rangle$. In contrast, the CNOT gate can do this. This can directly be seen in the structure of the matrix

$$U_{\mathrm{CNOT}} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{pmatrix}, \tag{5.2.2}$$

which does not have block-diagonal structure with block sizes 1-2-1. The reasoning is that, since single-qubit rotations alone cannot implement the CNOT gate, so the missing properties must come from the interaction term. The CNOT gate and the interaction propagator should then obey the same rules. However, it can easily be seen that the Ising interaction propagator does not mix the parity subspaces either because it is diagonal. However, single-qubit rotations can clearly connect the parity subspaces. Thus, definite statements about the mixing of the subspaces of different parity cannot be made for sequences of pulses which include either Heisenberg or Ising interaction propagators and single-qubit pulses.

A similar idea uses the symmetry of the interaction term as an argument. As mentioned in Sec. 2.1, the distinct difference between the Ising and the Heisenberg spin interactions are their symmetry properties. While the Ising model is invariant under reflection, the Heisenberg model is invariant under rotations. The Heisenberg interaction Hamiltonian is thus invariant under the $\mathfrak{su}(2)$ Lie algebra. The line of reasoning is similar to the previous approach. If the Heisenberg interaction propagator is invariant under rotations, can we implement the CNOT gate, which is clearly not invariant under rotations? If both the interaction propagator and the single-qubit rotations were rotationally invariant, we could prove that a sequence containing only such pulses could not implement the CNOT gate. However, similarily to the previous approach, single-qubit rotations are not invariant under rotations. Specifically, rotational invariance means that rotating all the spins by the same angle would leave the system's energy unchanged. Single-qubit rotations do not fulfill this criterion. In fact, it is exactly their action of rotating just one qubit that leads to them not being invariant to full rotations.

These ideas are described here as a starting point for future work. While I was not able to find a proof that a CNOT gate cannot be implemented using the isotropic Heisenberg interaction, I was also not able to find such an implementation using an exhaustive search of pulse sequences of up to length nine.

5.3. Further numerical improvement

If it should prove impossible to show that no Heisenberg XXX pulse sequence can be used to implement the CNOT gate, then it is useful to discuss how to improve the numerical approach to find such a sequence. In order to make this search more efficient, a few ideas to further improve the search algorithms beyond the optimizations in Chap. 4 are discussed here.

The first step of the algorithm in Chap. 4 is the generation of all the sequences. This is implemented by means of a Python generator in order to avoid saving all the sequences at once, say in a list. For convenience, methods of the Python module `itertools` were used. After their generation, all of the sequences had to be passed on to the part of the program that checked them for logic. Theoretically, one could implement the logic directly into a custom generator. The sequences that are illogical would then not have to be passed on. However, the passing of the sequences takes next to no CPU time, and the built-in modules are usually implemented very efficiently. Hence, such an optimization will not implement a big improvement in performance.

It was mentioned in Chap. 4 that the search for the sequences was limited to a very small set of rotation angles. This limitation to certain discrete angles is based on the experience that the CNOT sequence with the Ising interaction only requires angles of 90° or 180°. Of course, it is possible to rotate through any angle with NMR manipulation methods. This restriction might have been the reason for overlooking a potential CNOT sequence. We want to discuss two ways to loosen that limitation here.

The first idea is to use the Bloch sphere representation directly. We have seen in Sec. 2.3 how single-qubit rotations and interaction propagations change a state in the Bloch sphere. We could work in spherical coordinates to represent the current state in the Bloch sphere. With two qubits, we need two sets of spherical coordinates and, thus, we have to work with six degrees of freedom. The next step is to assume that we can rotate the state around any axis with NMR techniques. This is true because a rotation around an arbitrary axis can always be simulated by subsequent rotations around the cartesian axes. The task is to find a path in spherical coordinates through the Bloch sphere, so that it implements the CNOT operation. While more illustrative than the matrix representation, the Bloch-sphere representation does still not offer any numerical advantage. We still have the same number of degrees of freedom as with matrices because a unitary 4×4 matrix also has six degrees of freedom when leaving out the global phase. Also, calculating the new spherical coordinates after a rotation around an arbitrary axis is not easy. This method of searching would be advantageous if one were to search by hand and needed to keep track of the effect of the operation.

The aforementioned idea of varying continuous angles in the Bloch sphere representation is only a simplified application of the general method of solving such a problem. *Quantum optimal control* is a variational method and resembles the brachistochrone problem in classical mechanics by searching for a path of minimal action. The task is the following: given an initial state $|\Psi_i\rangle$ and a final state $|\Psi_f\rangle$, for what Hamiltonian $\hat{H}$ does the evolution $\hat{U}[\hat{H}](t)$ connect two states in the shortest amount of time, that is

$$|\Psi_f\rangle = \hat{U}\big[\hat{H}\big](t)\;|\Psi_i\rangle\;, \qquad \delta t\big(\delta\hat{H}\big) = 0\,. \tag{5.3.1}$$

The variation is, in general, not limited to time but can be extended to any type of action, for example, the amount of energy used in during the operation. Carlini *et al.* [28] give a good introduction to quantum optimal control. They develop a general variational theory for finding such a Hamiltonian and also present two examples with exact solutions. This method could potentially be used to determine wether a CNOT sequence using the Heisenberg interaction exists. The implementation of quantum optimal control would have exceeded the scope of this Master's thesis and is left as a topic for future work.

6. Conclusion

In this thesis, we have extended the concept of NMR based quantum computing to utilize the Heisenberg spin interaction to entangle the qubits. By changing the symmetry properties of the spin interaction, we have also changed the evolution of the spin states due to interaction. With this different interaction evolution, the sequences for quantum gates that are known from already published work are no longer applicable. We had to search for alternate sequences to also validate universal quantum computing based on the Heisenberg interaction.

I was not able to find a pulse sequence using the isotropic Heisenberg interaction that represents the CNOT gate or the set of matchgates with the numerical methods in this thesis. Thus, it was not possible to prove that universal quantum computing is possible using the isotropic Heisenberg interaction. However, I have derived a sequence for the CNOT gate using the Heisenberg XXZ interaction. This proves that universal quantum computing is possible using a Heisenberg interaction in general. While I have not found a sequence for the isotropic interaction, this does not mean that it is impossible to find such a sequence for all instances of the Heisenberg interaction.

The most interesting remaining question is: does a set of sequences using the isotropic Heisenberg interaction exist that can implement universal quantum computing? One possibility is that it could be proven to be impossible to implement such a sequence. Another possibility is that the methods of finding sequences could be improved, perhaps leading to such a sequence being found. Possible improvements include numerical optimizations of the program described in Chap. 4 or the application of quantum optimal control. With these methods, one could search for more rotation angles and for longer sequences without knowing in advance if a result exists.

It has indeed been possible to extend the framework of NMR quantum computing to the Heisenberg spin interaction. While universal quantum computing has been proven to be possible using a specific instance of the anisotropic Heisenberg interaction, further work is needed to show or disprove whether this can be done for the isotropic Heisenberg interaction.

Appendices

A. Python code for the sequence search

```python
import numpy as np
from itertools import product, chain, permutations, islice
from cProfile import run
from math import factorial
from multiprocessing import Pool, cpu_count, set_start_method, freeze_support
from multiprocessing.managers import SyncManager, BaseManager
from sys import stdout, getsizeof
from os import getpid
from time import sleep
from guppy import hpy


def mat_Ising(ang=np.pi):
    """returns the Ising models interaction rotational matrix"""
    return np.cos(ang / 2.) * np.eye(4) + 1j * np.sin(ang / 2.) * np.diag([1, -1, -1, 1])


def mat_XXX(ang=np.pi/2.):
    """returns the Heisenberg XXX models interaction rotational matrix"""
    return np.array([[np.exp(1j * ang), 0, 0, 0],
                     [0, np.cos(ang), 1j * np.sin(ang), 0],
                     [0, 1j * np.sin(ang), np.cos(ang), 0],
                     [0, 0, 0, np.exp(1j * ang)]])


def mat_XXZ(ang=np.pi/2., delta=2):
    """returns the Heisenberg XXZ models interaction rotational matrix"""
    return np.array([[np.exp(2j * ang * delta), 0, 0, 0],
                     [0, np.cos(ang), 1j * np.sin(ang), 0],
                     [0, 1j * np.sin(ang), np.cos(ang), 0],
                     [0, 0, 0, np.exp(2j * ang * delta)]])


single_mats = np.load('single_mats.npy')    # external file with all the single qubit rotation matrices


dir_which = (('x', 0), ('x', 1), ('y', 0), ('y', 1), ('z', 0), ('z', 1))    # allowing for positive and negative rotation for each axis
single_dict = {i: j for (i, j) in zip(dir_which, single_mats)}
single_dict_expanded = dict()
for key in single_dict:
    single_dict_expanded[(*key, -np.pi / 2)] = single_dict[key][0]
    single_dict_expanded[(*key, np.pi / 2)] = single_dict[key][1]
double_dict = dict()          # creating a dictionary of all two-pulse matrices
for key1 in single_dict_expanded:
    for key2 in single_dict_expanded:
        double_dict[(key1, key2)] = np.dot(single_dict_expanded[key1], single_dict_expanded[key2])


def sequence_to_operator(sequence):
    """
    Takes an encoded NMR pulse sequence an calculates its matrix operator
    :param sequence: [('x' or 'y' or 'z', 0 or 1 or (0, 1), angle), (.,.,.), ...]
    :return: matrix operator of the full sequence
    """
    dim = 0
    for i in sequence:
        if isinstance(i[1], int):
            temp = i[1]
        else:
            temp = max(i[1])
        if temp > dim:
            dim = temp
    #dim += 1
    dim = 2
    matrices = []
    global single_dict_expanded
    global double_dict
    last_was_single_rotation = False
```

```python

    for op in sequence:
        if last_was_single_rotation:
            if isinstance(op[1], int):
                matrices.insert(0, double_dict[(
   last_was_single_rotation, op)])
            else:
                matrices.insert(0, single_dict_expanded[
   last_was_single_rotation])
            last_was_single_rotation = False
        if isinstance(op[1], int):
            last_was_single_rotation = op
        elif op[0] == 'I':
            matrices.insert(0, mat_Ising(ang=(int(op[2]) - .5) * np.
   pi))
        elif op[0] == 'XXX':
            matrices.insert(0, mat_XXX(ang=(int(op[2]) - .5) * np.pi)
   )
        elif op[0] == 'XXZ':
            matrices.insert(0, mat_XXZ(ang=(int(op[2]) - .5) * np.pi)
   )

    # is there still a single rotation on the hold?
    if last_was_single_rotation:
        matrices.insert(0, single_dict_expanded[
   last_was_single_rotation])

    if len(matrices) > 1:
        mat = np.linalg.multi_dot(matrices)
    else:
        mat = matrices[0]

    # simplify
    mat = mat / mat[0, 0]
    return mat


def seq_is_logical(seq):
    """
    Checks, if a given sequence is trivial in some way
    :param seq:
    :return:
    """
    # limit how many rotations of each type are in a sequence
    counts = {'x': 0, 'y': 0, 'z': 0, 'XXX': 0, 'XXZ': 0, 'ramp': 0}
    for i in seq:
        if i[0] not in counts:
            counts[i[0]] = 1
        else:
            counts[i[0]] += 1
    if counts['XXX'] > 1 or counts['XXZ'] > 1 or (counts['y'] % 2 and counts['x'] % 2):
        pass

    # no two adjacent rotations can be of same type
    for i in range(len(seq) - 1):
        if seq[i][0] == seq[i + 1][0]:
            return False
    return True


dirs = {'x', 'y', 'z'}
single_angs = {False, True}
coupling_angs = {False, True}
coupling_dirs = {'XXX'}
nos = {0, 1}


def tot_comb(length):
    t = len(dirs) * len(nos) * len(single_angs) + len(coupling_dirs) * len(coupling_angs)
    return int(factorial(t) / factorial(t - length))


def sequence_commutations(seq):
    index = 0
    singles = {'x', 'y', 'z'}
    doubles = {'XXX'}
    yield seq
    # compare nearest neighbours
    while index < len(seq) - 1:
        if all((dir in singles for dir in (seq[index][0], seq[index + 1][0])))\
                or all((dir in doubles for dir in (seq[index][0], seq[index + 1][0]))):
            if index == 0:
                new_seq = (seq[index + 1],
                           seq[index],
                           *seq[index + 2:])
            elif index == len(seq) - 2:
                new_seq = (*seq[:index],
                           seq[index + 1],
                           seq[index])
            else:
                new_seq = (*seq[:index],
                           seq[index + 1],
                           seq[index],
                           *seq[index + 2:])
            yield new_seq

            if index < len(seq) - 2:
                if all((dir in singles for dir in (seq[index][0], seq
   [index + 1][0], seq[index + 2][0])))\
                        or all((dir in doubles for dir in (seq[index
   ][0], seq[index + 1][0], seq[index + 2][0]))):
                    if index == 0:
                        new_seq = (seq[index + 2],
                                   seq[index + 1],
                                   seq[index],
                                   *seq[index + 3:])
                    elif index == len(seq) - 3:
                        new_seq = (*seq[:index],
                                   seq[index + 2],
                                   seq[index + 1],
                                   seq[index])
                    else:
                        new_seq = (*seq[:index],
                                   seq[index + 2],
                                   seq[index + 1],
                                   seq[index],
                                   *seq[index + 3:])
                    yield new_seq

        index += 1


# noinspection PyStatementEffect
def process_combination(combs, already_found, tossed, count, tot,
   found_count, jumped_count, mat_count):

    newly_found = set()
    newly_tossed = set()
```

```python
    newly_jumped = 0
    newly_mult_mats = 0
    newly_found_count = 0
    i = 0
    for comb in combs:
        # do not calculate for already handled sequences and their equivalences
        if comb in already_found:
            newly_jumped += 1

        # do nothing if an equivalent sequence was already tossed before
        elif not tossed.isdisjoint(set(sequence_commutations(comb))):
            newly_jumped += 1

        # do not calculate matrix if sequence is trivial in some way
        elif not seq_is_logical(comb):
            newly_tossed.add(comb)
            newly_jumped += 1
        else:
            # create matrix
            mat = sequence_to_operator(comb)
            newly_mult_mats += 1

            # test for CNOT
            if all(abs(m) < 1e-6 for m in {mat[0, 1], mat[0, 2], mat[0, 3], mat[1, 0], mat[1, 2], mat[1, 3],
                                           mat[2, 0], mat[2, 1], mat[2, 2], mat[3, 0], mat[3, 1], mat[3, 3]}) \
                    and all(np.isclose(m, 1) for m in {mat[1, 1], mat[2, 3], mat[3, 2]}):
                newly_found_count += 1

                # add more commutations of the found sequence to the already found set
                newly_found.update(sequence_commutations(comb))
                #already_found.update(sequence_commutations(comb))

            else:
                newly_tossed.update(comb)

        if (count + i) % 1000 == 0:
            print('\r{0:} of {1:} sequences checked'.format(count + i, tot), end='')
            stdout.flush()
        if (count + i) % 1000000 == 0:
            print(’Size of tossed after’, count, ’steps:’, round(getsizeof(tossed._getvalue()) / 1000000, 1), ’MByte’)
            stdout.flush()
        i += 1

    already_found.update(newly_found)
    tossed.update(newly_tossed)
    jumped_count + newly_jumped
    mat_count + newly_mult_mats
    found_count + newly_found_count

    return


def callback_procession(args):
    return


class Counter(object):
    def __init__(self, i=0):
        self.value = int(i)

    def __add__(self, other):
        self.value += other

    def __mod__(self, other):
        return self.value % other

    def __str__(self):
        return str(self.value)

    def __repr__(self):
        return str(self.value)


def search_sequence(seq_len):
    print(’Sequence length: ’, seq_len)

    # create all possible sequences
    single_seqs = product(dirs, nos, single_angs)
    H_seqs = product(coupling_dirs, {(0, 1)}, coupling_angs)
    all_seqs = chain(single_seqs, H_seqs)
    all_perm = permutations(all_seqs, r=seq_len)

    # Proxies for multiprocessing
    SyncManager.register('set', set, exposed=('add', '__iter__', 'isdisjoint', 'update').0

    SyncManager.register('int', Counter, exposed=('__add__', '__mod__', '__repr__', '__str__'))
    manager = SyncManager()
    manager.start()
    tot = manager.int(tot_comb(seq_len))
    found_count = manager.int()
    jumped_count = manager.int()
    mat_count = manager.int()
    already_found = manager.set()
    tossed = manager.set()

    # multiprocessing
    p = Pool(maxtasksperchild=1000)
    chunk_len = 1000
    count = 0
    while True:
        try:
            count += 1
            combs = set(islice(all_perm, chunk_len))
            if not combs:
                raise StopIteration
            p.apply_async(process_combination,
                          args=(combs.copy(), already_found, tossed, count, tot, found_count, jumped_count, mat_count),
                          callback=callback_procession)
        except StopIteration:
            count -= 1
            break
    sleep(15)
    print(hpy().heap())
    p.close()
    p.join()

    print('\n\nsequences checked:\t', count, '/', tot)
    print('found:\t\t\t\t', found_count)
    print('jumped:\t\t\t\t', jumped_count)
    print('matrices multiplied:', mat_count)
    return set(already_found)
```

```

if __name__ == '__main__':
    set_start_method('forkserver')
    freeze_support()

    seq_len = 6
    run('found_seqs = search_sequence(seq_len)', sort='cumtime')
    with open('found_sequences_length_{}.txt'.format(seq_len), 'w')
   as file_handler:
        for seq in iter(found_seqs):
            file_handler.write('{}\n'.format(seq))
        if not found_seqs:
            file_handler.write('nothing found')


    quit()
```

B. Personal notes and acknowledgements

First of all, I want to thank my supervisor Prof. Dr. R. Noack for guiding me through the process of creating this work. I look back to many fruitful discussions and advice that has helped me grow as a theoretical physicist. I encourage everyone interested in numerical many-body-physics to apply for a thesis in the work group *Vielteilchenphysik* at *Universität Marburg*.

As an orientation for any future masters student I want to lay open the amount of work put into this thesis. I have logged my work hours on a daily basis. From the beginning of the browsing for background literature to handing in the thesis it required a total of 565 hours of work time. The total can be split up into 120 hours of background reading, 255 hours of implementing the program code in *Python*, and 185 hours for actually writing down the thesis with LaTeX. This time does only include actual work time and does not represent all the time spent at work.

C. Declaration of originality

English

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

German

Ich versichere hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, ganz oder in Teilen noch nicht als Prüfungsleistung vorgelegt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet. Mir ist bewusst, dass es sich bei Plagiarismus um akademisches Fehlverhalten handelt, das sanktioniert werden kann.

August 6, 2020

Date/Datum	Signature/Unterschrift (Maximilian Schwetz)

References

[1] I. I. Rabi, J. R. Zacharias, S. Millman, and P. Kusch, "A New Method of Measuring Nuclear Magnetic Moment", Phys. Rev. **53**, 318–318 (1938).

[2] E. M. Purcell, H. C. Torrey, and R. V. Pound, "Resonance Absorption by Nuclear Magnetic Moments in a Solid", Phys. Rev. **69**, 37–38 (1946).

[3] P. C. Lauterbur, "Image Formation by Induced Local Interactions: Examples Employing Nuclear Magnetic Resonance", Nature **242**, 190–191 (1973).

[4] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines", Journal of Statistical Physics **22**, 563–591 (1980).

[5] R. P. Feynman, "Simulating physics with computers", International Journal of Theoretical Physics **21**, 467–488 (1982).

[6] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2010).

[7] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring", in Proceedings 35th Annual Symposium on Foundations of Computer Science (1994), pp. 124–134.

[8] S. Lloyd, "A Potentially Realizable Quantum Computer", Science **261**, 1569–1571 (1993).

[9] M. H. Levitt, *Spin Dynamics*, Second (John Wiley & Sons Ltd, 2007).

[10] A. Auerbach, *Interacting Electrons and Quantum Magnetism*, edited by J. Birman, H. Faissner, and J. W. Lynn (Springer Verlag, 1994).

[11] J. Dr. Wells, *Lecture on the Standard Model*, tech. rep. (CERN, 2013).

[12] N. Goldenfeld, *Lectures on phase transitions and the renormalization group* (CRC Press, 2018).

[13] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman,

M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor", Nature **574**, 505–510 (2019).

[14] N. A. Gershenfeld and I. L. Chuang, "Bulk Spin-Resonance Quantum Computation", Science **275**, 350–356 (1997).

[15] Y. Wu and X. Yang, "Strong-Coupling Theory of Periodically Driven Two-Level Systems", Phys. Rev. Lett. **98**, 013601 (2007).

[16] D. G. Cory, A. F. Fahmy, and T. F. Havel, "Ensemble quantum computing by NMR spectroscopy", Proceedings of the National Academy of Sciences **94**, 1634–1639 (1997).

[17] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of Shor′s quantum factoring algorithm using nuclear magnetic resonance", Nature **414**, 883–887 (2001).

[18] L. M. K. Vandersypen and I. L. Chuang, "NMR techniques for quantum control and computation", Reviews of Modern Physics **76**, 1037–1069 (2005).

[19] R. Khalaf and A. Abdullah, "Novel Quantum Encryption Algorithm Based on Multiqubit Quantum Shift Register and Hill Cipher", Advances in High Energy Physics **2014**, 1–5 (2014).

[20] A. C. C. de Albornoz, J. Taylor, and V. Cărare, "Time-optimal implementations of quantum algorithms", Physical Review A **100** (2019).

[21] S. Lee, S.-J. Lee, T. Kim, J.-S. Lee, J. Biamonte, and M. Perkowski, "The cost of quantum gate primitives", Journal of Multiple-Valued Logic and Soft Computing **12**, 561–573 (2006).

[22] S. Ramelow, A. Fedrizzi, A. M. Steinberg, and A. G. White, "Matchgate quantum computing and non-local process analysis", New Journal of Physics **12**, 083027 (2010).

[23] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation", Physical Review A **52**, 3457–3467 (1995).

[24] J. Johansson, P. Nation, and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems", Computer Physics Communications **183**, 1760–1772 (2012).

[25] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. 1. 0. Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", Nature Methods **17**, 261–272 (2020).

[26] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, "VODE: A Variable-Coefficient ODE Solver", SIAM Journal on Scientific and Statistical Computing **10**, 1038–1051 (1989).

[27] G. H. Golub and C. F. V. Loan, *Matrix Computations*, Fourth Edition (The Johns Hopkins University Press, 2013).

[28] A. Carlini, A. Hosoya, T. Koike, and Y. Okudaira, "Time-Optimal Quantum Evolution", Phys. Rev. Lett. **96**, 060503 (2006).