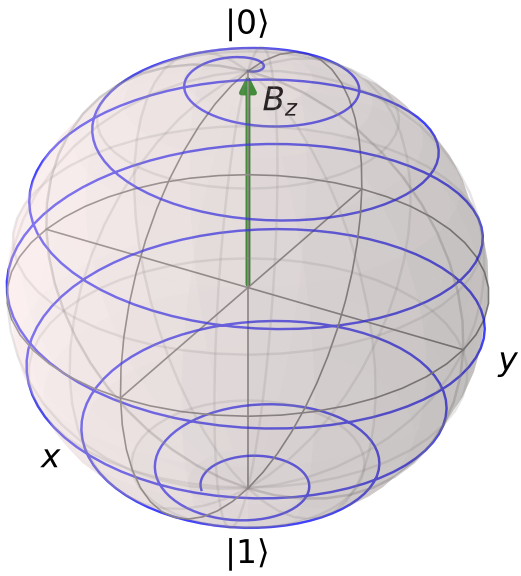
Modelling Magnetic Resonance using QuTiP



Aarts Sanne
Coelen Patrick
Gevers Nicolas
Haagsma Iris
Kušar Stela
Shaju Vignesh
Valenbreder Nina

Supervisor: Blackman Claire

Maastricht Science Programme
Maastricht University
25 January 2022

Contents

1 Introduction

Magnetic resonance is a quantum phenomenon used in different fields for a variety of purposes, most notably in Magnetic Resonance Imaging (MRI) and Nuclear Magnetic Resonance (NMR) spectroscopy. The former has allowed clinical professionals to deepen their *in-vivo* understanding of the physiology of living creatures in a non-invasive manner, while the latter has been used regularly to analyse chemical compounds in otherwise challenging settings. Over the years, magnetic resonance has been studied and put to use extensively, even though the phenomenon is unobservable to the naked eye.

The key to understanding scientific subjects is visualisation. This is especially true for teaching physics, where abstract concepts are often central to the comprehension [1]. Numerous studies have found that the extent to which such abstract subjects are understood, greatly increased through association of the concept to a student's personal experience [1, 2, 3].

It is only natural that physical demonstrations have grown to become a big part of the current academic environment. They are essential to create an intuitive impression of the subject while simultaneously explaining the relevant issue. Despite the progress made over the last few decades, accessibility, safety, and cost limitations still restrict the possibility for such demonstrations. This leaves the more abstract concepts in science classrooms poorly understood [4].

An analogue interpretations of their results are more often restricted to closed form solutions by purely mathematical constraints[1]. Quantum mechanics is one of the most, if not the most, abstract branch within physics because of its characteristic applicability to the sub-microscopic and the counter-intuitive behaviours of quantum systems. One concrete example is that the sheer act of observation on a quantum system alters its properties [5]. Because such behaviours do not obey the laws of classical physics, the visual demonstration possibilities are greatly limited. It should be noted that within quantum mechanics, the aforementioned mathematical restraints are very prominent[6] [7]. In particular, the Schrödinger equation, due to its complex form, only very few cases permit an analytical solution for example [8].

The concept of computational modelling is not new in physics. The early 1970s saw the introduction of models created using the now considered primitive programming languages such as 'Basic' and FORTRAN' into physics classrooms [9]. Since then, the available programs and software have greatly evolved, and their potential has increased significantly. In spite of this, computational modeling remains a vastly underused tool in the physical sciences [4, 1]. Usage of computational modelling allows us to surpass these limitations and simulate the system in a manner that whilst correct, is not more complicated than it needs to be [1].

The benefits and possible applications of computational modelling within science have been widely reported in numerous papers [10]. Yet very few of these papers provide an approach for implementation of computational modelling in the educational environment of Quantum mechanics. To fill this gap, this report aims to develop an understanding of the current possibilities for the usage of QuTiP and the Jupyter notebook in both the demonstration and the understanding of magnetic resonance with to explore their demonstrative potential.

2 Theoretical background

2.1 Formalism

The cornerstone of quantum mechanics is the Schrödinger equation:

$$i\hbar\frac{\partial}{\partial t}\left|\psi\right\rangle = \hat{H}\left|\psi\right\rangle .$$

It describes the behavior of a quantum system, where its quantum state is defined as a vector and denoted with $|\psi\rangle$. Acting on the state vector on the right hand side of the equation is the Hamiltonian operator $\hat{H}$. Since operators are mathematical objects that allow us to determine characteristics of a state, applying them to a quantum system is equivalent to making a measurement on it [11]. In the case of applying the Hamiltonian operator, the result returns the total energy of the quantum system.

The Schrödinger equation is a linear differential equation, meaning that any linear combination of solutions will itself be a solution. By finding two state vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ which are solutions to the Schrödinger equation, we can find any solution which is a linear combination of them. This concept is referred to as the superposition of quantum states [12, 11, 13]:

$$\left|\psi\right\rangle = a\left|\psi\right\rangle + b\left|\psi\right\rangle .$$

2.2 Magnetic Moment

Elementary particles such as electrons and protons have a spin, which is an intrinsic form of angular momentum. Even though there is no actual rotation of the particle around its own axis, it behaves as if there is [12, 14, 11, 13]. This leads to the induction of a magnetic moment, which refers to the orientation and magnitude of the magnetic field resulting from angular momentum of a moving charge [14].

Magnetic moment (μ) defines the interaction of a particle with an external magnetic field [14]. For an atom the total magnetic moment is a sum of the magnetic moment arising as the electron orbits the nucleus and the magnetic moment resulting from the spin of its elementary particles. When particles are not subjected to an external magnetic field, the magnetic moment of atoms is randomly oriented. However, when subjected to a magnetic field, a torque is exerted on the magnetic moment, to align it with the magnetic field, which changes its angular momentum [12, 14, 11, 13]. In a non-homogeneous magnetic field, this gives rise to Larmor precession, the rotation of the magnetic moment around the directional axis of the field with an angular frequency:

$$\Omega_0 = \gamma B_0.$$

This angular frequency, often referred to as the Larmor frequency, is a function of the magnetic field strength and the gyro-magnetic ratio γ, the quantitative relation between the magnetic moment and the angular momentum [15, 16].

In an non-homogeneous magnetic field, there is an additional net force acting on the magnetic moment which not only induces precession, but also changes the trajectory of the particle depending on the orientation of the magnetic moment. In the Stern-Gerlach experiment, a beam of atoms is passed through such an non-homogeneous magnetic field, after which the location of impact for each particle is measured [17, 13]. Classically, we would expect these locations to be distributed randomly, however, the experiment found that there were only two sites of impact, symmetrically separated. This demonstrates the quantization of angular momentum. The Stern-Gerlach experiment was performed using atoms with one unpaired electron. Since paired electrons with opposite spins cancel each other out, the magnetic moment of the atom is a result of the spin angular momentum of the remaining unpaired electron, with an insignificantly small contribution from the nuclear magnetic moment. Hence, the total value is a direct result of the particles' spin [17, 13].

Electrons, protons and other fermions are found to be spin-1/2 particles (I=$\frac{1}{2}$), meaning that their spin angular momentum can only take on the value $\pm\frac{1}{2}\hbar$ [15, 16, 13, 11, 14]. In the absence of a magnetic field, these states are degenerate, meaning that there is no energy difference between them. Only by applying an external magnetic field can this degeneracy be removed [15, 16]. The spin of a particle will then cause the magnetic moment to either align itself along (α-spin states) or against (β-spin states) the direction of the magnetic field, where the former decreases the energy of a particle and the latter increases it [16]. The separation of these energy levels allows for the monitoring of particles' transitions between them, characterised by the absorption or emission of electromagnetic radiation[15, 16].

In NMR, a static magnetic field B_0 is applied to a sample of atoms, we generally take the direction of B_0 to be along the z axis. The application of this magnetic field causes splitting of the energy levels, resulting in the Boltzmann distribution of nuclei between α-spin and β-spin states:

$$\frac{N^\beta}{N^\alpha} = e^{\frac{-\Delta E}{kT}} \quad \text{where} \quad \Delta E = \gamma B_0.$$

For any non-zero magnetic field the ratio $\frac{N^\beta}{N^\alpha} \leq 1$, this means more nuclei will be in the α-state than in the β-state. This results in a net magnetization in the direction of B_0, in NMR we describe the bulk magnetization of the nuclei in our sample with a magnetization vector $\mathbf{M}$ which describes the direction and magnitude of this net magnetization [16].

A second, circularly oscillating field, B_1, is then applied to the sample, orthogonal to B_0, this causes $\mathbf{M}$ to rotate about B_1 at the rate:

$$\Omega_1 = \gamma B_1$$

In NMR this B_1 field is associated with a radiofrequency (rf) pulse, the frequency of which corresponds to the difference in energy induced by B_0 [15, 16].

The resulting motion of $\mathbf{M}$ induces an rf signal, this signal can be detected by the same rf coil that generates B_1. The rf pulse is only applied for a short period of time, allowing $\mathbf{M}$ to realign to B_0. During this process of realignment, the induced rf pulse slowly decays, the signal produced by the decay is referred to as the free induction decay (FID). The FID signal is composite of the frequencies of all the target nuclei in the original sample. Fourier transformation of this signal allows us to convert it into a frequency dependent spectrum. [15, 16].

2.3 Magnetic Resonance in Quantum Mechanics

For the quantum mechanical description of NMR of a single hydrogen atom it is assumed that the atom is stationary and the external magnetic fields are very large and thus essentially unaffected by the disturbances caused by the atom. We define the magnetic field B with an oscillating x-component and a static z-component as:

$$\vec{B} = B_z\hat{u}_z + B_x\cos(\omega t)\hat{u}_x.$$

The Hamiltonian of such a quantum system can therefore be approximated to only depend on the potential of the external magnetic field and magnetic moment [13]:

$$\hat{H} = -\hat{\mu}\cdot B = -\gamma\hat{S}_z B_z - \gamma\hat{S}_x B_x\cos(\omega t),$$

where $\hat{S}_z$ and $\hat{S}_x$ are the spin angular momentum operators in the z- and x-direction, respectively.

Assuming a static magnetic field in z-direction the proton will align its spin with it, giving rise to two possible spin states χ_+ and χ_- with corresponding energy eigenstates:

$$E_\pm = \mp\frac{\hbar}{2}\gamma B_z.$$

The time-dependent state of $\hat{S_z}$ can be expressed as the linear combination of both states, each multiplied by their time-dependence.

$$\chi(t) = a\chi_+ e^{-iE_+/\hbar} + b\chi_- e^{-iE_-/\hbar}$$

In this case a and b are probability amplitudes, so the sum of their squares needs to be 1. This can most easily be achieved by assigning them the values $a = cos(\alpha/2)$ and $b = sin(\alpha/2)$, where α is an arbitrary angle. At this point we can calculate the expectation values of the three angular momentum operators $\hat{S}_x$, $\hat{S}_y$ and $\hat{S}_z$. Expectation values are the most likely outcomes of a measurement of an observable. If we were to repeatedly measure the spin in z-direction of a proton following the wave equation $|\psi\rangle$, the outcomes would average to the expectation value calculated by $\langle\psi| S_z |\bar{\psi}\rangle$. In this manner we can calculate all three expectation values of our case:

$$\begin{aligned}
\langle S_x\rangle &= \frac{\hbar}{2}\sin(\alpha)\cos(\gamma B_z t),\\
\langle S_y\rangle &= -\frac{\hbar}{2}\sin(\alpha)\sin(\gamma B_z t),\\
\langle S_z\rangle &= \frac{\hbar}{2}\cos(\alpha).
\end{aligned}$$

We can imagine that the spin state of our system is located on the surface of a sphere centered at the origin of a Cartesian coordinate system with axes S_x, S_y, S_z. The time dependence of $\langle S_x\rangle$ and $\langle S_y\rangle$ are $cos(\gamma B_z t)$ and $-sin(\gamma Bzt)$ respectively and thus describe a circular precession. $\langle S_z\rangle$ has no time dependence and through trigonometric considerations we can see that the system precedes at a constant angle α relative to the z-axis with the previously defined Larmor frequency [11].

Coming back to our original Hamiltonian we have two magnetic field components and thus two Larmor frequencies $\Omega_0 = \gamma B_z$ and $\Omega_1 = \gamma B_x$ we need to consider. Expressing the Hamiltonian in terms of these two frequencies yields

$$\hat{H} = -\Omega_0\hat{S}_z - \Omega_1\hat{S}_x\cos(\omega t).$$

As was the case for the Larmor precession, we can again write the spin states as a linear combination with probability amplitudes a and b:

$$|\psi\rangle = a(t)\,|\chi_+\rangle + b(t)\,|\chi_-\rangle\,.$$

The Schrödinger equation can thus be expressed as

$$\frac{d}{dt}\begin{pmatrix}a(t)\\b(t)\end{pmatrix} = \frac{i}{2}\begin{pmatrix}\Omega_0 & \Omega_1\cos(\omega t)\\ \Omega_1\cos(\omega t) & -\Omega_0\end{pmatrix}\begin{pmatrix}a(t)\\b(t)\end{pmatrix}.$$

In the case of magnetic resonance experiments $B_z >> B_x$ is often the case, which means that it can be assumed that Ω_1 is relatively unimportant and terms containing it can be dropped in the first approximation. Doing this results in

$$\begin{aligned}
a(t) &= a(0)e^{i\Omega_0 t/2},\\
b(t) &= b(0)e^{-i\Omega_0 t/2}.
\end{aligned}$$

These equations describe the spin state of our proton depending on time. If we don't assume that the oscillating magnetic field can be neglected we can simply replace the initial constant values $a(0)$ and $b(0)$ with new time-dependent variables $c(t)$ and $d(t)$. Rearranging the Schrödinger equations in these terms gives

$$\begin{aligned}
\frac{d}{dt}c &= \frac{i\Omega_1}{4}\left(e^{i(\omega-\Omega_0)t} + e^{-i(\omega+\Omega_0)t}\right)d(t),\\
\frac{d}{dt}d &= \frac{i\Omega_1}{4}\left(e^{i(\omega+\Omega_0)t} + e^{-i(\omega-\Omega_0)t}\right)c(t),
\end{aligned}$$

and reveals a new dependency on the combined rotating frequencies with a rapidly oscillating term $\Omega_0 + \omega$ and a slowly oscillating term $\Omega_0 - \omega$. According to the rapid wave approximation the rapidly oscillating term can be neglected near resonance, so when $\Omega_0 \approx \omega$:

$$\begin{aligned}
\frac{d}{dt}c &= \frac{i\Omega_1}{4}e^{i(\omega-\Omega_0)t}d(t),\\
\frac{d}{dt}d &= \frac{i\Omega_1}{4}e^{-i(\omega-\Omega_0)t}c(t).
\end{aligned}$$

Both terms of the Schrödinger equation now depend on c(t) and d(t). To decouple the equations we can differentiate the first equation. Through rearranging and substitution we find

$$\frac{d^2}{dt^2}c(t) - i(\omega-\Omega_0)\frac{d}{dt}c(t) + \frac{{\Omega_1}^2}{16}c(t) = 0.$$

We can assume c(t) to be oscillatory with a new frequency ω'

$$c(t) = e^{i\omega' t}$$

where

$$\omega'_\pm = \frac{(\omega-\Omega_0) \pm \sqrt{(\omega-\Omega_0)^2 + (\Omega_1/2)^2}}{2}.$$

The square root in this equation is called the Rabi frequency:

$$\Omega_R = \sqrt{(\omega-\Omega_0)^2 + (\Omega_1/2)^2}\,,$$

which allows us to calculate the probability of a spin flip. Assuming that the system is initially in spin-up state, the probability to flip state depending on

time is simply $|b(t)|^2$, or the probability amplitude of the spin-down state squared:

$$P(-z,t) = \frac{\left[(\omega-\Omega_0)^2 - {\Omega_R}^2\right]^2}{{\Omega_1}^2{\Omega_R}^2} 4\sin^2\left(\frac{\Omega_R t}{2}\right).$$

The term

$$P_{max} = \frac{4\left[(\omega-\Omega_0)^2 - {\Omega_R}^2\right]^2}{{\Omega_1}^2{\Omega_R}^2}$$

characterizes the amplitude of this oscillatory function and thus its maximum probability. P_{max} is evidently largest when $\omega = \Omega_0$, so at resonance [13].

3 Methods

3.1 Jupyter Notebook

Jupyter notebook is an open source platform for developing a coding environment. This was used for designing the final product in order to make the code as accessible and reproducible as possible. To achieve that, both for people with and without prior coding experience, it is important to explain thoroughly how each different part of the code affects the outcome [18]. This allows for demonstration of the physics concepts involved while also showing their implementation within the code [19]. Each notebook consists of a number of cells, allowing the code to be divided into smaller blocks which could be run individually or sequentially. In addition to code cells, a notebook can also contain "Markdown" cells which display the input in the form of text. This combination allows designers to be flexible in their implementation of the code, explanatory text, equations, and user input. This platform has grown to become a popular tool for teaching and sharing purposes amongst computer scientists, and thus a tool we took advantage of to meet our aims. [20].

3.2 Quantum Toolbox in Python (QuTiP)

Theoretical and numerical tools are necessary to take into consideration the interactions between the system and its environment, without introducing external influence (e.g. by experimental measurement). In order to solve most non-trivial Hamiltonians, numerical simulations of the equations of motion are required. The underlying Hilbert space's dimensionality then increases exponentially which quickly renders the computation impractical. It is possible however to design systems from oscillators and spin components, excited by a limited number of quanta, that limit the size of the space and allow for a classical simulation. For the calculations and simulations of this work, the programming language python was used in conjunction with the Quantum Toolbox in Python, i.e. QuTiP [21]. This open-source framework builds on the Numpy [22], SciPy [23] and Cython [24] libraries to provide the necessary algorithms to solve the dynamics of Hamiltonians effectively. Additionally, it supports parallel computing, which means that the code can leverage all of the performance from the multi-core processors found in modern computers.

3.3 Installation setup

The first task at hand was to setup the coding environment. Since the aim of the project was modeling using QuTiP, both Python and QutiP needed to be installed. This was done through Anaconda and the conda tool using the pip command. QuTiP's documentation is the recommended source of installation instructions, providing a step-by-step guide for any operational system. This project was carried out on Windows and macOS processing systems. When setting up the new QuTiP coding environment, it became apparent that multiple other modules needed to be implemented in order for QuTiP to function. To run and execute the code product of this project, the following versions were installed:

```
QuTiP Version:        4.6.2
Numpy Version:        1.22.0
SciPy Version:        1.7.3
Cython Version:       0.29.25
Matplotlib Version:   3.4.3
Python Version:       3.9.7
```

For the purpose of collaborating and creating a Jupyter notebook end-product, a Python 3.9 interpreter in PyCharm was used by all project members in conjunction with the QuTiP 4.6.2 toolbox. Different versions of Numpy, SciPy, and Python were used due to compatibility constraints. Matplotlib was notably downgraded to Version 3.4.3 for compatibility with the Bloch sphere class in QuTiP.

3.4 Visualisation

Two notebooks, a main notebook and a supplementary one, were created, both of which depend heavily on visualisation techniques. The supplementary notebook creates animations which depend on user input through widgets, which are imported from the IPywidgets package. As such, the animations are generated by creating a time loop. The main notebook, on the other hand, uses the `FuncAnimation()` function from Matplotlib which requires additional FFmpeg package to create and save animations in mp4 format. This methodology was adapted from [25].

4 Results and Discussion

4.1 Quantum Mechanics and The Hydrogen Atom

The starting point for this research was to improve our collective abilities to use Python in visualising quantum mechanics concepts related to the Hydrogen atom. We realised users may benefit from an interactive notebook (see *Supplementary_Notebook.ipynb* from Appendix A: GitHub) as supplementary material for understanding quantum-related concepts. Before presenting samples of the corresponding code, it is important to note that this notebook did not require using QuTiP. It first uses the Laguerre polynomials to plot the radial and radial probability plots for the wavefunction. The code developed for this notebook was adapted from [26] and [27], respectively. To begin, features necessary for the visualisations were imported to the stock python environment.

```
import numpy as np
from pylab import *
import matplotlib.pyplot as plt
from matplotlib import cm, colors
from mpl_toolkits.mplot3d import Axes3D
import ipywidgets as widgets
import scipy.special as spe
from mpl_toolkits import mplot3d
```

Listing 1: Module imports for code samples from `Supplementary_Notebook.ipynb` notebook.

Using the Laguerre special function from SciPy, a new function was defined in order to calculate the radial component of the Schrödinger Equation. The user is able to change the principle `(n)` and orbital `(l)` quantum numbers to generate their corresponding radial plots (Figure 1, 2). Users may find the radial probability distribution plot particularly useful for understanding how position is determined by probability, and how this can further relate to applications such as density plots in quantum chemistry.

```
def psi_R(r,n,l):
    w = np.sqrt((2.0/n)**3 * spe.
    factorial(n-l-1) /(2.0*n*spe.
    factorial(n+l)))
    x = spe.assoc_laguerre(2.0*r/n,n-l
    -1,2*l+1)
    y = np.exp(-r/n)
    z = (2.0*r/n)**l
    answer = w*x*y*z
    return answer

def radial_function (n, l=0):
    x = np.linspace(0,10,100)
    y = psi_R(x,n,l)
    E_y = np.absolute((y**2))*(x**2)


widgets.interact(radial_function, n
    =(1,5,1), l=(0,5,1))
```

Listing 2: The radial part of the Hydrogen wavefunction.

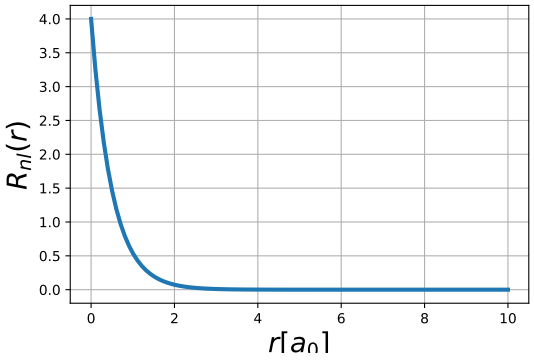


Figure 1: The radial plot output from running the code in Listing 2.

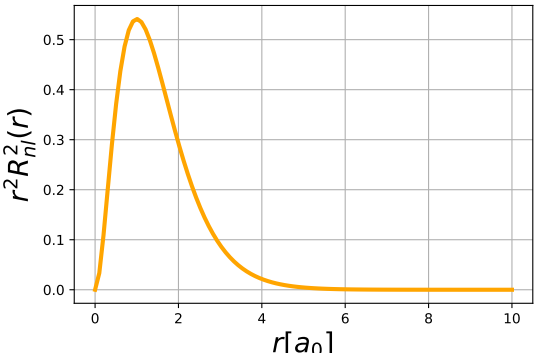


Figure 2: The radial density plot output from running the code in Listing 2.

In a similar manner, it is also possible for the user to compute the angular part of the Hydrogen wavefunction by using the spherical harmonics special function from SciPy. In the code below, the user is able to change the principle `(n)` and magnetic `(m)` quantum numbers within the notebook.

```
def psi_ang(m, n, theta, phi):
    sphHarm = spe.sph_harm(m, n, theta,
    phi)
    #theta in [0, 2pi] and phi in [0, pi]
    #computes sphetical harmonics using
    associated Legendre functions (Pnm)
    print(sphHarm.real)
    return sphHarm.real
```

Listing 3: The angular part of the Hydrogen wavefunction.

As shown in the figure below, the final output of this notebook is a 3D visualisation of the atomic orbitals for Hydrogen.

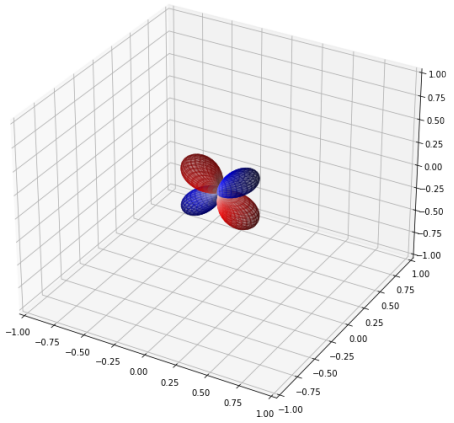


Figure 3: The atomic orbital shape visual produced after running the code in Listing 3.

The second half of this notebook was designed for users to visualise how the spin-1/2 particle experiences a magnetic field, in parallel with the text in [13]. Users are first able to observe the effect of the strength of the static magnetic field `Bz` via widgets.

```
A = float(input("Bz = "))
def magneticfield_z(Bz): #Function to
    plot our static magnetic field
    lines_plotted = plt.plot([]) #Empty
    plot
    plt.ylim(0,(A + 1)) #Want y-limit to
    always be slightly higher than the
    input max, so that we can see the
    field line
    line_plotted = lines_plotted[0]
    plt.xlim(0,2*np.pi)
    x = np.linspace(0,2*np.pi,100)
    line_plotted.set_data((x, Bz))

widgets.interact(magneticfield_z, Bz
    =(0.0, A, 1.0))
```

Listing 4: The static magnetic field along the z-axis.

This is followed by a similar interactive visual which allows users to observe the effect of a changing oscillating magnetic field strength `Bx`.

```
print("What should the maximum field
    strength along the x-axis be?")
C = float(input("Bx = "))

print("What should the maximum of the
    angular frequency be?")
D = float(input("The angular frequency is
    : "))

def plot (Bx=4.0, w=4.0, t=0.0):

    lines_plotted = plt.plot([])

    plt.ylim(-10,10)
    line_plotted = lines_plotted[0]
    plt.xlim(0,2*np.pi)

    x = np.linspace(0,2*np.pi,100)
    B_x=0
    B_x=Bx*np.cos(x+w*t)
    line_plotted.set_data((x,B_x))


# Widget loops over time. It can be used
    to adjust amplitude and frequency.
# w = increments of half pi
# click on loop button to loop over time
    otherwise it lasts 30 sec

widgets.interact(plot, Bx=(0.0, C, 1.0),
    w=(0.0, D, np.pi/2), t=widgets.Play(
    min=0.0, max=30.0))
```

Listing 5: The oscillating magnetic field along the x-axis.

In the final visual both the static and the oscillating magnetic field are plotted. It is possible for the user to change the strength of both the magnetic fields as well as the angular frequency. As this animation loops over time, the effect of these changes on the magnetic field can easily be observed.

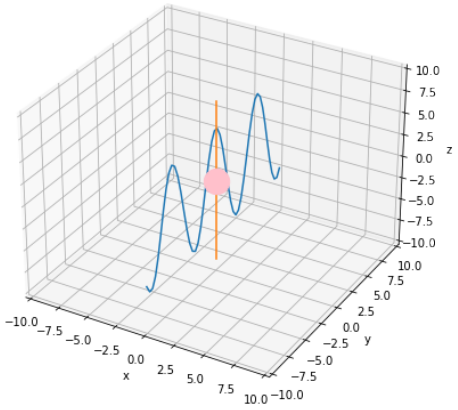


Figure 4: Static and oscillating magnetic field

The interactivity permitted in this notebook gives users the ability to follow content from Chapter 13.4 in [13] chapter with the complementary visualisations.

4.2 Magnetic Resonance

The overarching aim of this research was to develop a comprehensive understanding of QuTiP and its potential as a tool for understanding magnetic resonance. A Jupyter notebook was developed for this purpose, which allows users to explore magnetic resonance and its relation to a spin-1/2 particle via animations on the Bloch sphere. This is intended as a complementary resource for chapter 9.4 and complement 9.A from [13]. In the following subsections, we will outline how quantum theory influenced the code developed in *Magnetic_Resonance.ipynb* from Appendix A: GitHub.

4.2.1 Spin States, Operators and Expectation Values

A state of a quantum mechanical system, including all known information, is represented by a normalized ket, $|\psi\rangle$ [13, 11]. This notation can be used to express all quantum states as in the Hilbert space. In the aforementioned Stern-Gerlach experiments, it was found that the orientation of a spin-1/2 particle is either spin-up $|+z\rangle$ or spin-down $|-z\rangle$. These two spin states form a complete set of basis vectors which are expressed as

$$|+z\rangle = \begin{pmatrix}1\\0\end{pmatrix}_z \quad , \quad |-z\rangle = \begin{pmatrix}0\\1\end{pmatrix}_z .$$

They form the eigenvectors of the Hermitian spin operator $\hat{S}_z$ and correspond to eigenvalues of $\pm\hbar/2$. $\hat{S}_x$ and $\hat{S}_y$ behave analogously. A measurement of spin along the z-axis is mathematically represented by applying $|\psi\rangle$ to $\hat{S}_z$. The result of this operation $\langle\hat{S}_z\rangle$ is called the expectation value. The operator $\hat{S}_z$ can therefore be expressed using the Pauli matrix representation σ_z, which forms a basis of the spin space, according to the axis it is operating on (z in this case):

$$\hat{S}_z = \frac{\hbar}{2}\begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} = \frac{\hbar}{2}\sigma_z .$$

Similarly, S_x and S_y can be defined using:

$$\sigma_x = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix} \quad , \quad \sigma_y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix} .$$

After importing the necessary packages, the fundamental variables pertaining to the spin-1/2 particle were defined by calling functions from QuTiP, as shown in Listing 6.

```
#Spin states
pz = qt.Qobj([[1],[0]]) # +z
mz = qt.Qobj([[0],[1]]) # -z

#Pauli matrices
sx = qt.sigmax()
sy = qt.sigmay()
sz = qt.sigmaz()

#Expectation values - observable spin
    components
Sx = 1/2.0 * qt.sigmax()
Sy = 1/2.0 * qt.sigmay()
Sz = 1/2.0 * qt.sigmaz()
```

Listing 6: Defining the spin states, Pauli matrices, and expectation values using QuTiP.

4.2.2 Plotting on the Bloch Sphere

The step from quantum theory to quantum computing requires an understanding of qubits [28]. Classical information in computing is stored in a binary digit, or bit, which can take the logical values "0" or "1". Its analogue in quantum computing is the qubit, which instead takes the form of a linear combination of these logical values. Physically, this is represented as two distinct eigenstates:

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \quad \text{or} \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix} .$$

Within the notebook, users can note the equivalence of these eigenstates with those defined earlier for measurements along the z axis. The subsequent *computational* basis in the Hilbert space is represented as the set of these eigenstates:

$$[|0\rangle , |1\rangle] .$$

This basis allows for the expression of other orthonormal basis vectors, including:

$$|+\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \quad , \quad |-\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} .$$

Using this notation allows one to express the generic state of the qubit as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle .$$

The complex conjugates α and β, are related by $|\alpha|^2 + |\beta|^2 = 1$, which allows a two-state system to be parameterized by θ and ϕ as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\psi}\sin\left(\frac{\theta}{2}\right)|1\rangle .$$

This geometric representation visualises the dynamics of quantum states on the unit radius Bloch sphere. The `qutip.Bloch` class, and its associated functions, was central in the notebook design to visualise a spin-1/2 particle exhibiting magnetic resonance. As shown in the listing below, users were therefore first introduced to plotting a point, state and multiple states on a Bloch sphere.

```
b = qt.Bloch() #create a Bloch sphere

#plotting a single state:
st1 = qt.Qobj(1/2*np.array([np.sqrt(3)
    ,-1])) #input [a,b]
b.add_states(st1) #transforms state into
    Bloch representation -> calculates
    angles to plot on sphere

#plotting a single point
north_pole = [0,0,1]
b.add_points(north_pole)

#plotting two or more different states:
st_arr = [qt.Qobj(1/2*np.array([np.sqrt
    (3),-1])), qt.Qobj(1/2*np.array([-np.
    sqrt(3),-1]))] #array of quantum
    states
b.add_states(st_arr, kind='point') #you
    can also plot it as points by
    changing kind (kind='point' or '
    vector')

b.render() #render figure
```

Listing 7: Plotting on the Bloch Sphere.

4.2.3 Spin-1/2 Particle in a Magnetic Field

After allowing the user to become more familiar with the Bloch sphere visualisation using QuTiP, the notebook discusses the model of a spin-1/2 particle in a magnetic field. Beginning with the *static* magnetic field, initial and time-dependent spin states $a(t)$ and $b(t)$ are defined. This includes defining values of the gyromagnetic ratio γ, static magnetic field strength in the z-direction B_z, angle of precession α, and time period. The code in the subsequent section in the notebook (see Listing 8) calculates the time evolution of state of the spin-1/2 particle through the following calculation:

$$\chi(t) = \begin{pmatrix} a(t) \\ b(t) \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) e^{i\gamma B_z t/2} \\ \sin\left(\frac{\alpha}{2}\right) e^{-i\gamma B_z t/2} \end{pmatrix} .$$

```
b = qt.Bloch()
xi_0 = qt.Qobj([[math.cos(alpha/2)], [
    math.sin(alpha/2)]])
xi_t = [qt.Qobj(np.array([math.cos(alpha
    /2)*cmath.exp(1j*gamma*Bz*time/2),
    math.sin(alpha/2)*cmath.exp(-1j*gamma
    *Bz*time/2)])) for time in t]
b.add_states(xi_0, kind='point')
b.add_states(xi_t, kind='vector')
b.add_vectors([0,0,1]) # magnetic field
b.add_annotation([0,0.1,0.8],'$B_z$')
b.render()
```

Listing 8: Manually computed time evolution and its plotting on Bloch sphere.

The same time-dependent dynamics can be achieved by the `sesolve()` function in QuTiP. This requires defining the time-independent Hamiltonian (`H`) as a function of γ, $\hat{S}_z$, and static magnetic field strength. Additionally, the Hamiltonian, initial state, time and operator parameters are used as inputs, upon which it returns the evolution of the system." This QuTiP function is a more efficient and practical way to solve the time-independent and time-dependent Schrödinger equation than a manual calculation, as we will see later on. The accompanying visual generated via both approaches is equivalent and shown in Figure 5.

```
#Hamiltonian
H = -gamma*Sz*Bz

#starting state
psi0 = qt.Qobj(np.array([math.cos(alpha
    /2),math.sin(alpha/2)]))

#time evolution
result = qt.sesolve(H, psi0, t, [sx,sy,sz
    ])
b = qt.Bloch()
b.add_points([x, y, z], 'l') #'m', 's' or
     'l'
b.add_states(psi0, kind='vector')
b.add_vectors([0,0,1]) # magnetic field
b.add_annotation([0,0.1,0.8],'$B_z$')
b.render()
```

Listing 9: Using `sesolve()` for spin-1/2 particle time evolution in static magnetic field.

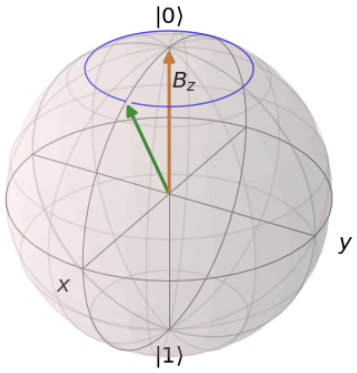


Figure 5: Bloch sphere representation of time-evolution in a static magnetic field

Next, time evolution of a proton is modeled in an *oscillating* magnetic field along the x-axis:

$$\overrightarrow{B} = B_x \cos(\omega t)\overrightarrow{u_x}.$$

As explained before, the best approach is the usage of `sesolve()`, which in this case requires a time-dependent Hamiltonian. This can be achieved through defining a time-dependent term as a user-defined function `H_coeff` (see listing below), which is then paired with an operator on the list `H`. The visual output of this computation is shown in Figure 6.

```
omega = gamma * Bz # Larmor frequency

#initial state
psi0 = pz #spin-up

#Hamiltonian
def H_coeff(t, args):
    return np.cos(w*t)
H = [-omega * Sx, H_coeff]

times = np.arange(0, np.pi/2, 0.1) #time
    in sec

result = qt.sesolve(H, psi0, times, [sx,
    sy, sz]) #evolve in time
x, y, z = result.expect #extract
    expectation values

#Plot on Bloch sphere
b = qt.Bloch()
b.add_points([x,y,z], 'm')
b.render()
```

Listing 10: Using `sesolve()` for the time evolution of a spin-1/2 particle in an oscillating magnetic field.

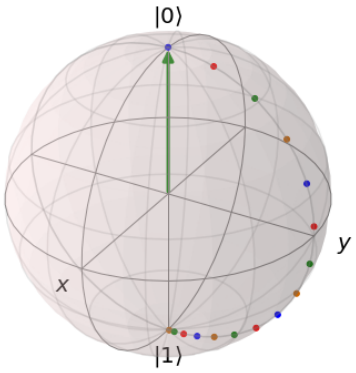


Figure 6: Bloch sphere representation of time-evolution in an oscillating magnetic field

The following section combines the static and oscillating magnetic field components to model magnetic resonance of a proton. This required accounting for the magnetic field strength separately as B_z and B_x, and hence for the Hamiltonian ($\hat{H}$) with a time-independent and time-dependent components. While this would be an extremely challenging task for manual calculation, `sesolve()` only required adjusting the input variables and the resulting values obtained through `result.expect` could be plotted on the Bloch sphere to represent magnetic resonance as state time evolution. By establishing equivalence `w = omega0`, conditions that result in magnetic resonance were satisfied. For the convenience in the subsequent use of the same methods, a new function was defined as shown in the listing below.

```
#arbitrarily chosen values
gamma = 1000 #proton gyromagnetic ratio (my value but *10**6 --> would need many tiny steps)
Bz = 1      #static mag field strength in z-axis
Bx = 10e-6   #oscillating mag field strength in x-axis
t = np.arange(0.0, 1000, 0.2) #arbitrary choice for time
psi0 = pz #initial state

#STATE TIME EVOLUTION FUNCTION
def time_evol_states(gamma, Bz, Bx, t, psi0):
    #Larmor frequencies
    omega0 = gamma*Bz
    omega1 = gamma*Bx

    #frequency of oscillating field
    w = omega0 # this determines if resonance happens (w = Omega0)

    #Hamiltonian
    def H_coeff(t, args):
        return np.cos(w*t)
    H = [-omega0 * Sz, [-omega1 * Sx, H_coeff]]

    #Evolve in time & extract expectation values
    result = qt.sesolve(H, psi0, t, [sx, sy, sz])
    exp0, exp1, exp2 = result.expect
    return exp0, exp1, exp2

x, y, z = time_evol_states(gamma, Bz, Bx, t, psi0)
```

Listing 11: Using `sesolve()` for the time evolution of a spin-1/2 particle in magnetic resonance.

With the following code, the results of the newly-defined function were plotted (see Figure 7).

```
n = 6 #takes every nth value
style = 's' #use 'l', 'm' or 's'

#Plot on Bloch Sphere
b = qt.Bloch()
expt_arr = [x[::n], y[::n], z[::n]]
b.add_points(expt_arr, style)
Bz_vec = [0,0,1]
b.add_vectors(Bz_vec)
b.add_annotation([0,0.15,0.8],'$B_z$')
b.render()
```

Listing 12: Plotting the results of state evolution in magnetic resonance on the Bloch sphere.

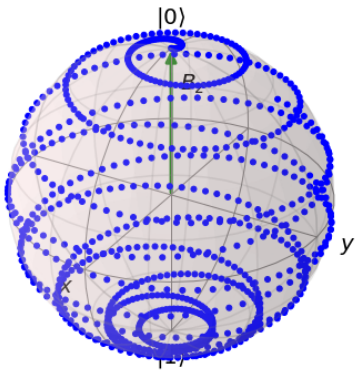


Figure 7: Bloch sphere representation of magnetic resonance.

4.2.4 Rabi Oscillations

As the quantum state of the spin-1/2 particle is oscillating between $|0\rangle$ and $|1\rangle$, it is also possible to visualise its associated Rabi oscillation as a plot of state probability versus time (Figure 8), where state probability can be obtained simply by specifying `mz*mz.dag()` (`mz` was defined as a spin-down state) as one of the operators in the last entry of `sesolve()`.

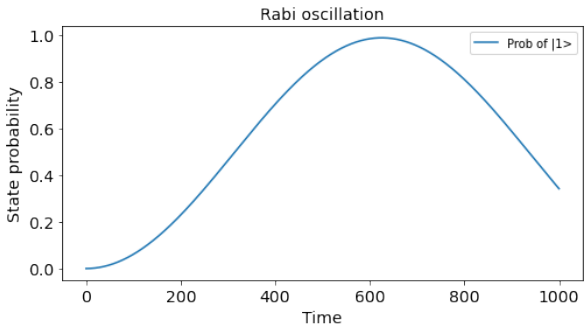


Figure 8: Rabi oscillation, corresponding to the magnetic resonance time evolution from the previous section.

To simulate a single state flip with 100% chance and end in the opposite spin state we need to apply an external magnetic field for a time $t = \pi/\Omega_R$, which is also called a π-pulse and can be easily calculated after computing the Rabi frequency:

$$\Omega_R = \sqrt{(\omega - \Omega_0)^2 + (\Omega_1/2)^2}$$

Using a π-pulse as the time input for the `sesolve()` solver, we can model exactly one spin flip, which is half of a Rabi cycle, whose plot is shown on the Figure 9. The corresponding Bloch sphere model is shown on Figure 10.

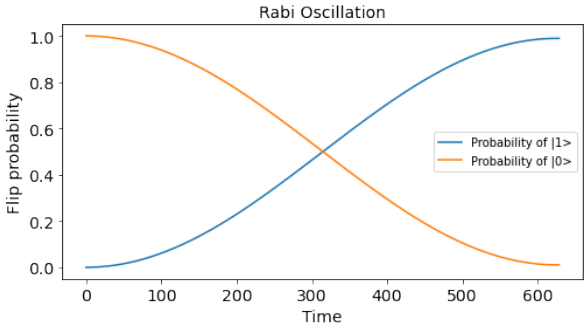


Figure 9: Rabi oscillation of exactly one spin flip.

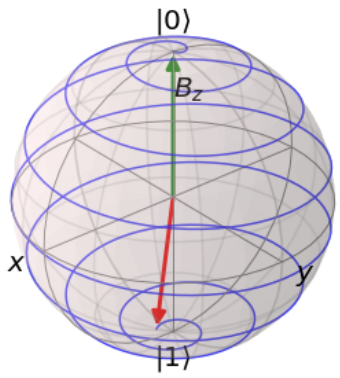


Figure 10: Single flip Bloch sphere representation.

4.2.5 Animations

The notebook then encodes two animations with the aim of tying all previous visualisations together. These are both saved as mp4 files within the user's local directory. The first visualisation is a simulation of state evolution under magnetic resonance as the spin-1/2 particle undergoes a single flip (see listing 13). This required defining a new function to animate the state vectors in the Bloch sphere.

```
t = np.arange(0.0, t_rabi, 0.02) #pi-pulse in seconds
x, y, z, H = time_evol_states(gamma, Bz, Bx, t, psi0)

n = 60 #takes every n-th value - ##all multiples of 60 work well for the shape
exp0 = x[::n]
exp1 = y[::n]
exp2 = z[::n]

#First set up the figure, the axis, and the plot element we want to animate
fig1 = plt.figure()
ax = Axes3D(fig1, azim=-40, elev=30, auto_add_to_figure=False)
fig1.add_axes(ax)
b = qt.Bloch(axes=ax)

#Animation function (called sequentially)
def animate(i): #generates new image each iteration
    b.clear()
    vector = [[exp0[i], exp1[i], exp2[i]]] #new vector added where new point appears
    b.add_vectors(vector)
    b.add_points([exp0[:i+1], exp1[:i+1], exp2[:i+1]],'l') #new point added
    b.add_vectors(Bz_vec)
    b.vector_color = ['r','g']
    b.add_annotation([0,0.15,0.8],'$B_z$')
    b.make_sphere()
    return ax

#Initialization function: plot the background of each frame
def init(): #keep vectors same color
    return ax

#Animate and save
ani1 = animation.FuncAnimation(fig1, animate, init_func=init, repeat=False, save_count=len(exp0))

ani1.save('bloch_sphere_1Flip.mp4', fps=20, extra_args=['-vcodec', 'libx264'])
```

Listing 13: Animating single flip magnetic resonance.

The second animation simulates a complete Rabi cycle and therefore visualizes a double spin flip, which ends with the state vector in its initial position. This is achieved simply by doubling the time to `t = np.arange(0.0, 2*t_rabi, 0.02)` and running the same code. The resulting

final image of the animation is shown on Figure 11.

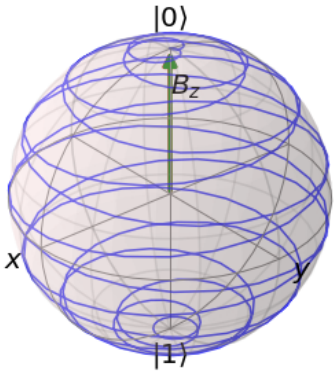


Figure 11: Final image of the double spin flip animation.

4.2.6 Test Initial Conditions

The notebook concludes by giving the user the opportunity to explore the influence of different initial conditions (namely γ, B_z, B_x and t). Prior to running the code, the user can manually change the initial conditions of two individual Bloch spheres, which are then plotted side by side for a practical visual comparison of the outcomes. It should be pointed out that not every input will be adequate for the code to run due to limitation of a time step size nor will every combination of initial conditions generate a useful plot. Furthermore, the user should be aware that with increases of values, particularly γ, B_z and the number of time steps, the code's execution time might also escalate drastically.

5 Conclusion

The aim of this project was achieved through creation of a Jupyter notebook, which calculates and models the magnetic resonance of a spin-1/2 particle, including its resulting Rabi oscillations. Additionally, a supplementary notebook was developed to help users further understand the quantum properties of a hydrogen atom. To create these notebooks, Python and the necessary dependencies were installed to successfully set up the QuTiP environment.

Both notebooks were created while keeping the user in mind. In the supplementary notebook this was achieved through the use of widgets, which take the user input, where in the main notebook parts of the code (mainly initial conditions) can be adjusted to produce different results. This enables a more interactive learning experience which aids in knowledge acquisition. The main notebook relies on QuTiP built-in classes and functions to gradually compute and simulate the model of magnetic resonance.

These notebooks are intended as a supplemental resource for students to help better grasp the understanding of quantum mechanics and magnetic resonance. The theory presented in this paper should enhance the comprehension of background knowledge required for complete understanding of these notebooks, however, depending on the reader's prior knowledge, additional reading of other literature might be needed. This is why the notebook should be used to complement the theory and not replace it. It should also be noted that in order to use this notebook, a QuTiP environment needs to be created. This could cause limitations of the use of the notebook, if the user is not able to achieve that. One way this has been overcome is through the use of Binder in conjunction with GitHub, which can be used to pull the required dependencies from the *requirements.txt* file on GitHub to build a functional python environment in which the Jupyter notebooks can be run.

Although the project focused on the use of the QuTiP module for computation and visualisation of quantum mechanics concepts, it is apparent that in the supplementary notebook other Python modules allowed for a better introduction to magnetic fields. However, using these modules to demonstrate the effect of these fields on the spin-1/2 particle was not as straightforward. The main notebook was able to demonstrate these effects on the particle using QuTiP, where users were even able to change these conditions, however, visualisations of the magnetic field are missing. To overcome this separation, these notebooks could be combined, using user input to first visualise the magnetic field and subsequently showing the effect of this magnetic field on the spin-1/2 particle.

References

[1] Jannis Weber and Thomas Wilhelm. "The benefit of computational modelling in physics teaching: a historical overview". In: *European Journal of Physics* 41.3 (2020), p. 034003. ISSN: 0143-0807. DOI: 10.1088/1361-6404/ab7a7f.

[2] Bruce Sherin. "Common sense clarified: The role of intuitive knowledge in physics problem solving". In: *Journal of Research in Science Teaching* 43 (Aug. 2006), pp. 535–555. DOI: 10.1002/tea.20136.

[3] John Clement. "Use of physical intuition and imagistic simulation in expert problem solving." In: (1994).

[4] G. Battimelli et al. *Computer Meets Theoretical Physics: The New Frontier of Molecular Simulation*. The Frontiers Collection. Springer International Publishing, 2020. ISBN: 9783030393991. URL: https://books.google.nl/books?id=VCbsDwAAQBAJ.

[5] Massimiliano Sassoli de Bianchi. "The Observer Effect". In: *Foundations of Science* 18 (Sept. 2011). DOI: 10.1007/s10699-012-9298-3.

[6] Drummond Brian. "Understanding quantum mechanics: a review and synthesis in precise language". In: *Open Physics* 17.1 (2019), pp. 390–437. DOI: doi:10.1515/phys-2019-0045. URL: https://doi.org/10.1515/phys-2019-0045.

[7] Nicolay V. Lunin. *"Nobody understands quantum mechanics." Why?* 2017. arXiv: 1707.02857 [physics.gen-ph].

[8] Miquel Bosch, Guillem Megıas, and Gerard Pascual. "Accurate numerical approach to Quantum Mechanics". In: (2018).

[9] Alfred Bork. "Computers as an aid to increasing physical intuition". In: *American Journal of Physics* 46.8 (1978), pp. 796–800. DOI: 10.1119/1.11189. eprint: https://doi.org/10.1119/1.11189. URL: https://doi.org/10.1119/1.11189.

[10] Priscilla W. Laws, Maxine C. Willis, and David R. Sokoloff. "Workshop Physics and Related Curricula: A 25-Year History of Collaborative Learning Enhanced by Computer Tools for Observation and Analysis". In: *The Physics Teacher* 53.7 (2015), pp. 401–406. DOI: 10.1119/1.4931006. eprint: https://doi.org/10.1119/1.4931006. URL: https://doi.org/10.1119/1.4931006.

[11] David J. Schroeter Darrell F. Griffiths. *Introduction to quantum mechanics*. 2018. ISBN: 9781107189638.

[12] D. Bohm. *Quantum Theory*. Dover books in science and mathematics. Dover Publications, 1989. ISBN: 9780486659695. URL: https://books.google.nl/books?id=-vhCqN2twGQC.

[13] Mark Beck. "Quantum mechanics : theory and experiment". In: (2012). URL: http://site.ebrary.com/id/10578493.

[14] H.D. Young et al. *University Physics with Modern Physics, Global Edition*. Pearson Education, Limited, 2019. ISBN: 9781292314730. URL: https://books.google.nl/books?id=T7OoxgEACAAJ.

[15] I. Campbell. *Biophysical Techniques*. OUP Oxford, 2012. ISBN: 9780199642144. URL: https://books.google.nl/books?id=4bvcMXDFrwYC.

[16] John C Edwards. "Principles of NMR". In: *Process NMR Associates LLC, 87A Sand Pit Rd, Danbury CT* 6810 (2009).

[17] Jean van Huele and Jared Stenson. "Stern-Gerlach experiments: past, present, and future". In: (Jan. 2004).

[18] Helen Shen. "Interactive notebooks: Sharing the code". In: *Nature* 515.7525 (2014), pp. 151–152. ISSN: 0028-0836. DOI: 10.1038/515151a. URL: https://dx.doi.org/10.1038/515151a.

[19] Bernadette M. Randles et al. "Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study". In: *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)* (2017), pp. 1–2.

[20] João Felipe Pimentel et al. "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (2019), pp. 507–517.

[21] J.R. Johansson, P.D. Nation, and Franco Nori. "QuTiP 2: A Python framework for the dynamics of open quantum systems". In: *Computer Physics Communications* 184.4 (Apr. 2013), pp. 1234–1240. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2012.11.019. URL: http://dx.doi.org/10.1016/j.cpc.2012.11.019.

[22] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[23] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[24] Stefan Behnel et al. “Cython: The best of both worlds”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39.

[25] Jake VanderPlas. *Matplotlib Animation Tutorial*. Aug. 2012. URL: https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/.

[26] Poscha McRobbie and Eitan Geva. *Wolfram Demonstrations Project*. Jan. 2010. URL: https://demonstrations.wolfram.com/HydrogenAtomRadialFunctions/.

[27] Davit Potoyan and Zachery Crandall. *Visualizing H atom wavefunctions*.

[28] Ivan S. Oliveira et al. “3 - Fundamentals of Quantum Computation and Quantum Information”. In: *NMR Quantum Information Processing*. Ed. by Ivan S. Oliveira et al. Amsterdam: Elsevier Science B.V., 2007, pp. 93–136. ISBN: 978-0-444-52782-0. DOI: https://doi.org/10.1016/B978-044452782-0/50005-1. URL: https://www.sciencedirect.com/science/article/pii/B9780444527820500051.

Appendices

Appendix A: GitHub

All the relevant files were uploaded on GitHub and can be accessed through the following link: `https://github.com/GerbOise/Magnetic_Resonance-QuTiP`

Additionally, the notebooks can be easily run through binder, through the following links:

Magnetic_Resonance	`https://mybinder.org/v2/git/https%3A%2F%2Fgithub.com%2FGerbOise%2FMagnetic_Resonance-QuTiP/HEAD?labpath=Magnetic_Resonance.ipynb`
Supplementary_Notebook	`https://mybinder.org/v2/git/https%3A%2F%2Fgithub.com%2FGerbOise%2FMagnetic_Resonance-QuTiP/HEAD?labpath=Supplementary_Notebook.ipynb`

Appendix B: Contributions

Introduction	Sanne Aarts, Vignesh Shaju
Theoretical background	Patrick Coelen, Sanne Aarts, Vignesh Shaju
Methods	Nicolas Gevers
Results and Discussion	Nina Valenbreder, Stela Kušar
Conclusion	Iris Haagsma
Supplementary notebook	Nina Valenbreder, Iris Haagsma
Main notebook	Stela Kušar
Presentation	Iris Haagsma, Nina Valenbreder, Nicolas Gevers, Patrick Coelen, Sanne Aarts, Stela Kušar, Vignesh Shaju