Computational Physics - Exercise 9: Nuclear Magnetic Resonance

Torben Steegmann

Introduction

The Bloch equations describe the dynamics of nuclear magnetization in a magnetic field and are fundamental in the fields of nuclear magnetic resonance. Named after Felix Bloch, these equations are a set of differential equations that model the behavior of the magnetization vector of nuclear spins in an external magnetic field.

In this exercise we will discuss the theoretical background of the Bloch equations and describe as well as implement a numerical solution.

Simulation model and method

Background

Nuclear Magnetic Resonance (NMR) studies the behavior of atomic nuclei within a magnetic field using the Bloch equations. These equations describe the dynamics of nuclear magnetization, which precesses around the magnetic field and undergoes relaxation processes. In the absence of an external magnetic field, the magnetic moments of spin-1/2 nuclei exhibit thermal motion, thus while not randomly oriented we can effectively assume them to be.

When placed in a static magnetic field, these nuclei align either parallel (spin-up) or anti-parallel (spin-down) to the field direction. Under this influence, their magnetic moments undergo Larmor precession at a frequency known as the Larmor frequency.

NMR experiments involve applying a secondary oscillating magnetic field (radiofrequency pulse) at the Larmor frequency. This field tilts the magnetization vector of the nuclei from the z-axis (aligned with the main magnetic field) into the xy-plane. The extent of this rotation depends on the strength and duration of the applied magnetic field.

NMR has widespread applications in medical science, particularly in Magnetic Resonance Imaging (MRI). MRI uses NMR principles to generate detailed images of tissues based on their nuclear magnetic properties.

Analytical Solution

The Bloch equations describe the time evolution of the magnetization vector $\overrightarrow{M} = (M^x, M^y, M^z)$ and are given by:

$$\frac{d\overrightarrow{M}}{dt} = \gamma \overrightarrow{M} \times \vec{B}(t), \tag{1}$$

thus:

$$\frac{dM^x}{dt} = \gamma(M^y B^0 - M^z B^y) - \frac{M^x}{T_2}, \tag{2}$$

$$\frac{dM^y}{dt} = \gamma(M^y B^x - M^0 B^y) - \frac{M^x}{T_2}, \tag{3}$$

$$\frac{dM^z}{dt} = \gamma(M^x B^y - M^y B^x) - \frac{M^z - M^0}{T_2}, \tag{4}$$

where $\vec{B} = (B^x, B^y, B^0)$ is the magnetic field vector, M_0 is the magnetization for $t \to \infty$, γ is the gyromagnetic ratio, which is a constant that relates the magnetic moment of a nucleus to its angular momentum. T_1 is the longitudinal relaxation time, which describes the time it takes for M^z to return to its equilibrium value along the z-axis and T_2 is the transverse decoherence time, which describes the time it takes for M_x and M_y to decay to zero in the x-y-plane.

During this exercise we will make a few mathematical assumptions to make the solution easier. First of all we are going to assume $M_0 = 0$, thus reducing Equation 4 to

$$\frac{dM^z}{dt} = \gamma(M^x B^y - M^y B^x) - \frac{M^z}{T_2}. \tag{5}$$

Furthermore, we assume $B^x(t) = h\cos(\omega t)$ and $B^y(t) = -h\sin(\omega t)$, thus

$$\vec{B} = (h\cos(\omega t), -h\sin(\omega t), B^0). \tag{6}$$

Numerical Solution

Since solving the analytical equations involves integration, a numerical solution is used to find good approximations. Using the product formula approach we say the numerical solution for $\overrightarrow{M}(t)$ is:

$$\overrightarrow{M}(t) = e^{\tau\gamma B(t-\frac{\tau}{2})} e^{\tau\gamma B(t-\frac{3\tau}{2})} \ldots e^{\tau\gamma B(\frac{\tau}{2})} M(0). \tag{7}$$

This is the result of reformulating Equation 1 into a matrix formulation:

$$\frac{dM}{dt} = \gamma BM = \gamma \begin{pmatrix} 0 & B^z & -B^y \\ -B^z & 0 & B^x \\ B^y & -B^x & 0 \end{pmatrix} \begin{pmatrix} M^x \\ M^y \\ M^z \end{pmatrix} \tag{8}$$

For each $e^{\tau\gamma B(\ldots)}$ we divide B into $B + C$ so that $e^{\tau\gamma B}$ is $e^{\tau\gamma\frac{C}{2}} e^{\tau\gamma B} e^{\tau\gamma\frac{C}{2}}$ where

$$C = -\begin{pmatrix} \frac{1}{\gamma T_2} & 0 & 0 \\ 0 & \frac{1}{\gamma T_2} & 0 \\ 0 & 0 & \frac{1}{\gamma T_1} \end{pmatrix}, \tag{9}$$

thus,

$$e^{\tau\gamma\frac{C}{2}} = \begin{pmatrix} e^{-\frac{\tau}{2T_2}} & 0 & 0 \\ 0 & e^{-\frac{\tau}{2T_2}} & 0 \\ 0 & 0 & e^{-\frac{\tau}{2T_1}} \end{pmatrix}. \tag{10}$$

What remains is solving the matrix for $e^{\tau\gamma B}$. For this we assume B remains constant during each time-step. After calculation the matrix for $e^{\tau\gamma B}$ where $\gamma = 1$ can is formulated as Equation 11:

$$
\begin{aligned}
e^{\tau B_{\{11\}}} &= \frac{(B^x)^2 + \left[(B^y)^2 + (B^z)^2\right]\cos(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{12\}}} &= \frac{B^x B^y[1-\cos(\Omega t)] + \Omega B^z \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{13\}}} &= \frac{B^x B^z[1-\cos(\Omega t)] - \Omega B^y \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{21\}}} &= \frac{B^x B^y[1-\cos(\Omega t)] - \Omega B^z \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{22\}}} &= \frac{(B^y)^2 + \left[(B^x)^2 + (B^z)^2\right]\cos(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{23\}}} &= \frac{B^y B^z[1-\cos(\Omega t)] + \Omega B^x \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{31\}}} &= \frac{B^x B^z[1-\cos(\Omega t)] + \Omega B^y \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{32\}}} &= \frac{B^y B^z[1-\cos(\Omega t)] - \Omega B^x \sin(\Omega t)}{\Omega^2} \\
e^{\tau B_{\{33\}}} &= \frac{(B^z)^2 + \left[(B^x)^2 + (B^y)^2\right]\cos(\Omega t)}{\Omega^2},
\end{aligned}
\tag{11}
$$

where $\Omega^2 = (B^x)^2 + (B^y)^2 + (B^z)^2$.

Implementation

We begin by defining preliminary values given by the exercise. The ones that do not vary throughout configurations are defined as follows:

$$\vec{B}(t) = \left(h\cos\left(\omega_0\left(t - \frac{\tau}{2}\right) + \varphi\right), -h\sin\left(\omega\left(t - \frac{\tau}{2}\right) + \varphi\right), B^0\right) \tag{12}$$

$$B^0 = 2\pi f_0, f_0 = 4 \tag{13}$$

$$h = 2\pi f_1, f_1 = \frac{1}{4} \tag{14}$$

$$\gamma = 1 \to \omega_0 = B^0 \tag{15}$$

$$\tau = 0.01, \overrightarrow{M}(1) = (0, 0, -1). \tag{16}$$

The rest of the preliminary values vary throughout the exercise as different configurations are picked. These can be controlled via the macro CONFIGURATION.

We implement the product formula approach as function NMR(). Here we iterate over all time-steps and apply the product formula to all three elements of the vector for M (M_t). During each time-step we calculate the vector for B at the current time-step, and then calculate its exponential matrix. Lastly, we store the resulting vector of M in three matrices, where each stores the value of one axis, for easy plotting.

Simulation results

The results of the most interesting configurations can be seen in Figure 1 to Figure 4.

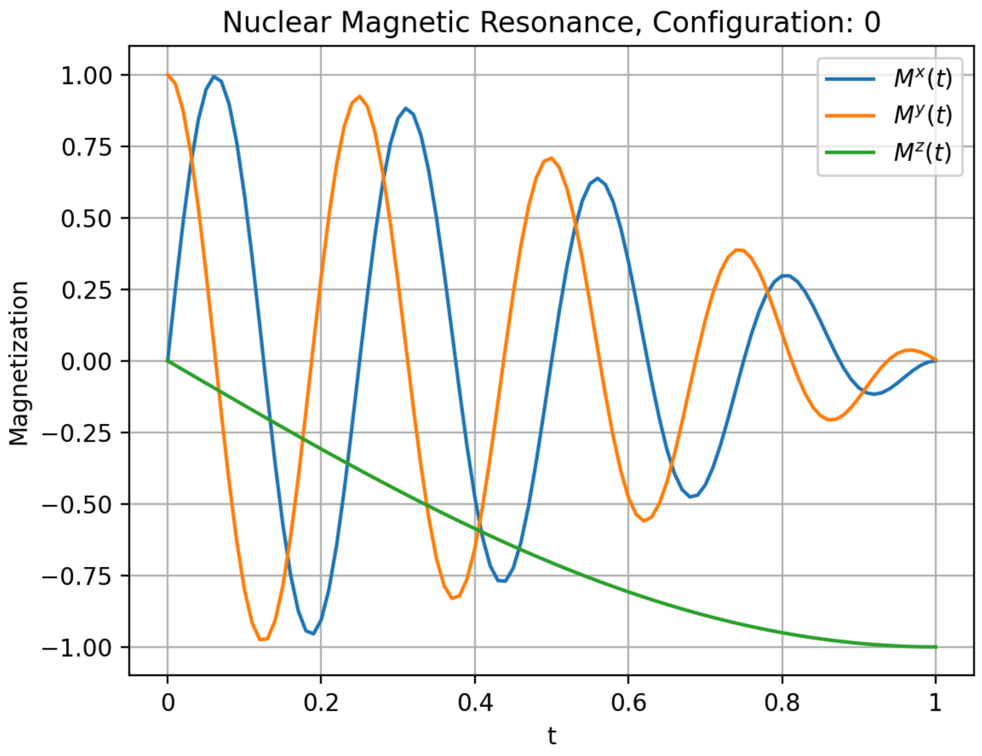


Figure 1: NMR simulation using Configuration 0. Here we restrict the time interval to one second. The configuration sets the initial values to $\overrightarrow{M}(0) = (0, 1, 0)$, $\varphi = 0$, $\frac{1}{T_1} = \frac{1}{T_2} = 0$. The z-axis of M transitions from 0 to −1, the x- and y-axes oscillate between 1 and -1 initially, while dampening towards 0 over time.

We start with an initial configuration in Figure 1. Here we set $\overrightarrow{M}(0) = (0, 1, 0)$, $\varphi = 0$, $\frac{1}{T_1} = \frac{1}{T_2} = 0$. These inverses of relaxation times indicate infinitely large relaxation times, which effectively means there is no relaxation. As predicted in the exercise, the resulting vector after 1 second of the simulation is $\overrightarrow{M}(1) = (0, 0, -1)$.

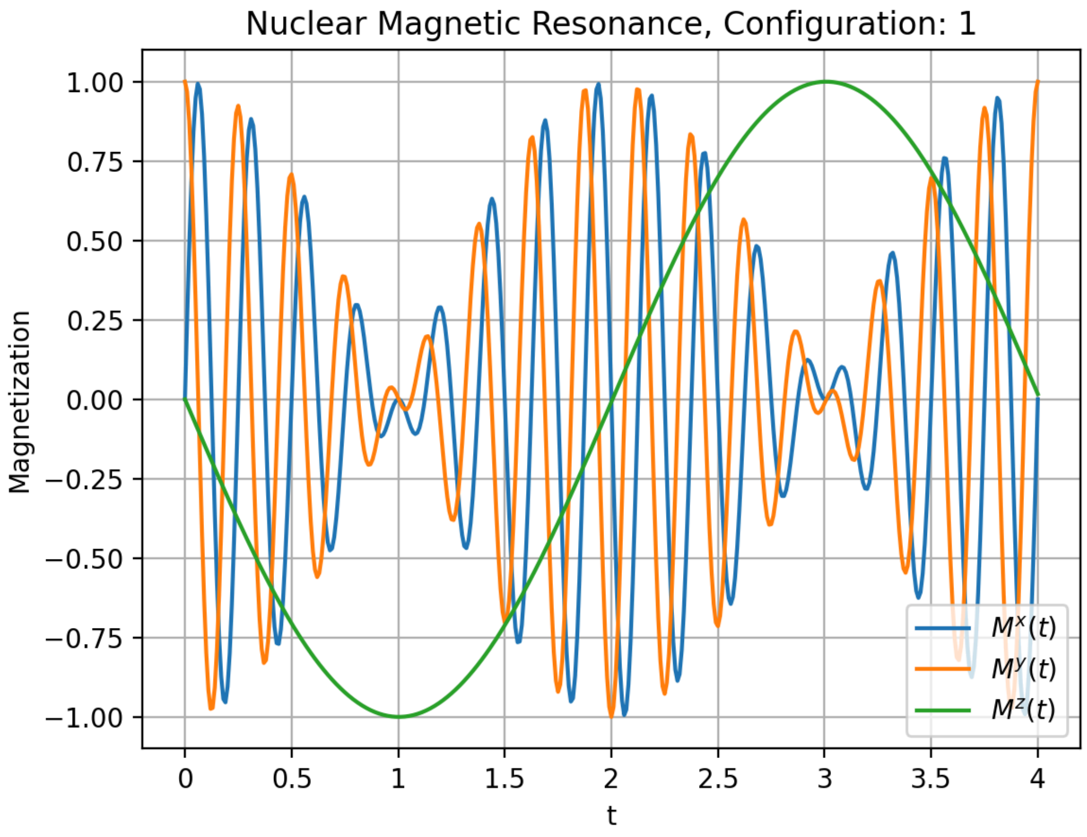


Figure 2: NMR simulation using Configuration 1. The configuration is the same as Figure 1, but with the time interval extended to 4 seconds. After the first second, the z-axis of M transitions from −1 to 1 and then back to 0. The x- and y-axes oscillate whenever the z-axis is not at 1 or -1; they oscillate strongest when the z-axis is 0.

Figure 2 expands the time frame to 4 seconds, showing that oscillation is not done after 1 second but continues as M^z transitions from −1 to 1. This indicates a spin transition from negative parallel alignment to the magnetic field to positive parallel alignment. We then see the z-axis go back to 0 where the pattern starts again, confirming that relaxation does not happen.

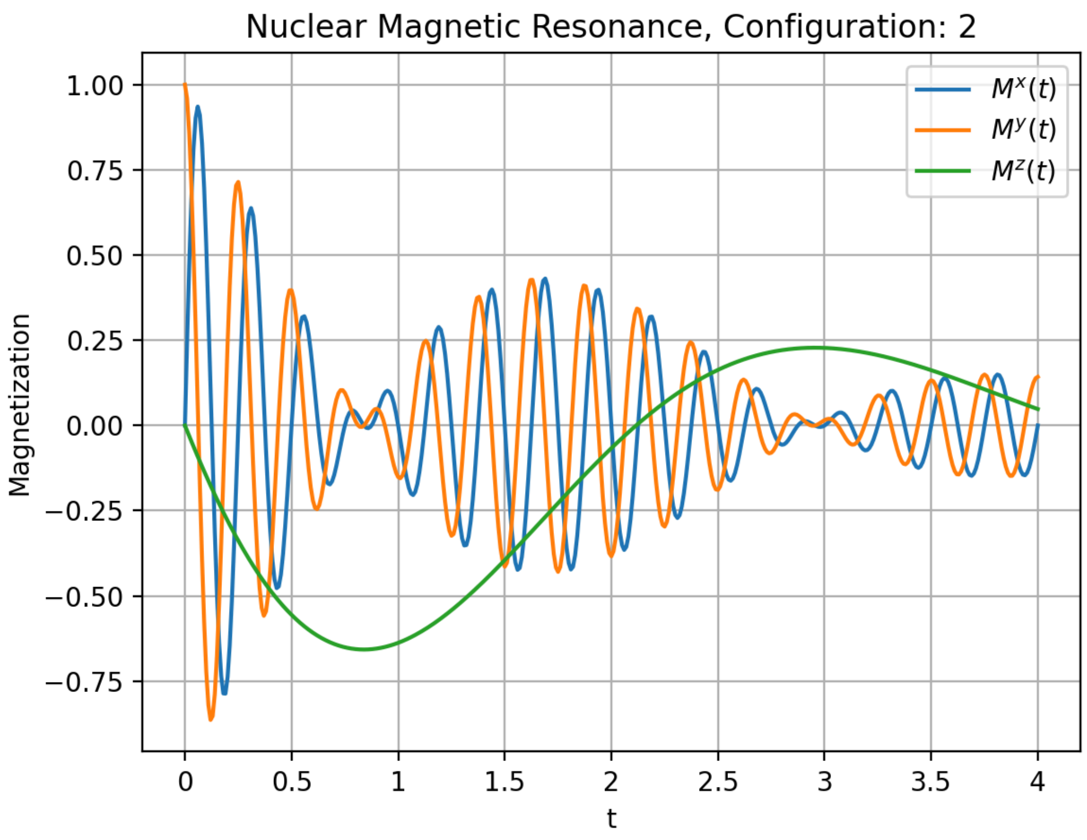


Figure 3: NMR simulation using Configuration 2. The configuration sets the initial values to $\overrightarrow{M}(0) = (0, 1, 0)$, $\varphi = 0$, $\frac{1}{T_1} = 0$, $\frac{1}{T_2} = 1$. The z-axis of M transitions from 0 to −0.6, the x- and y-axes oscillate between 1 and -0.8 initially, while rapidly dampening towards 0 over time.

In Figure 3 the $\frac{1}{T_2}$ relaxation time has been set to 1 second. This results in overall relaxation. As the longitudinal relaxation time T_1 is still set to ∞ ($\frac{1}{T_1} = 0$), M^z now oscillates strongest when M^x and M^y approximate 0.

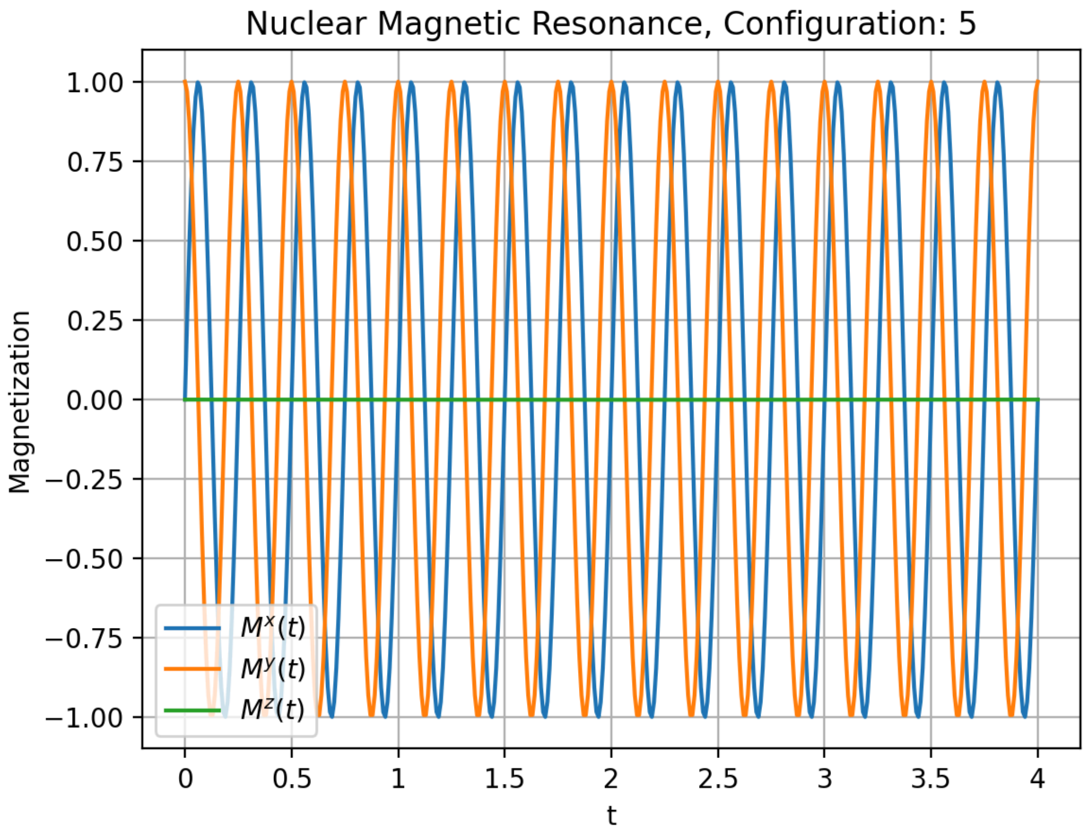
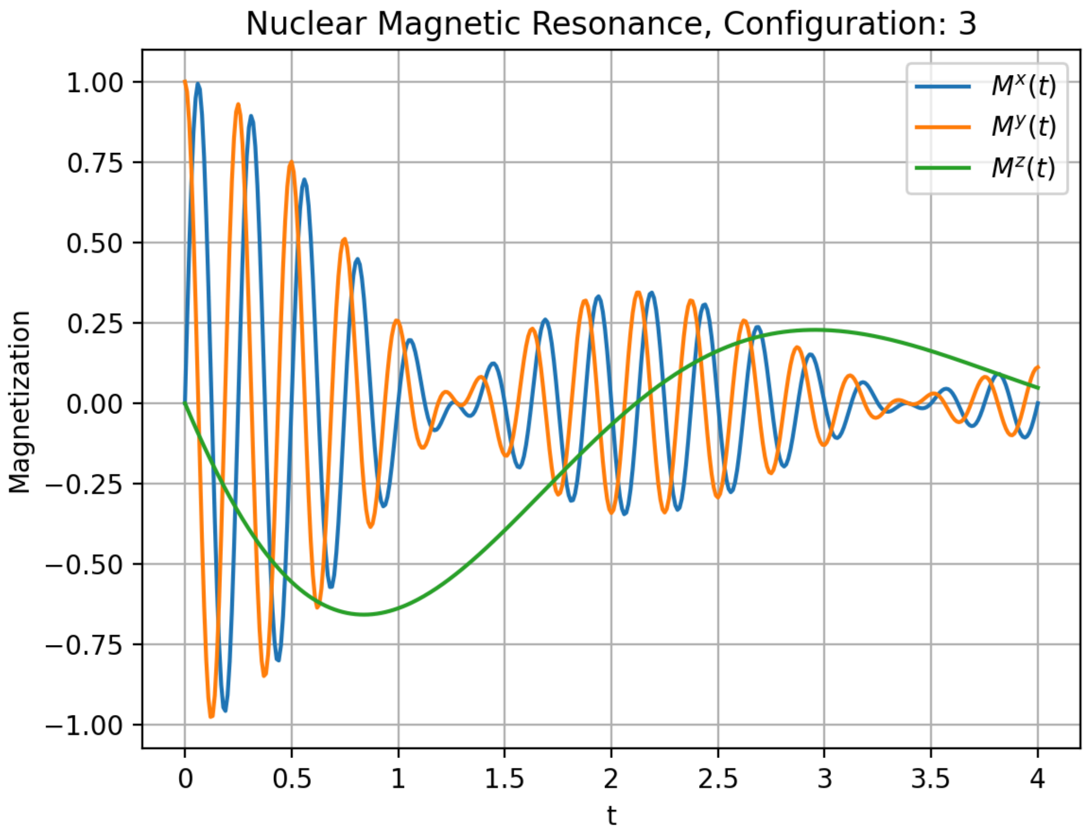


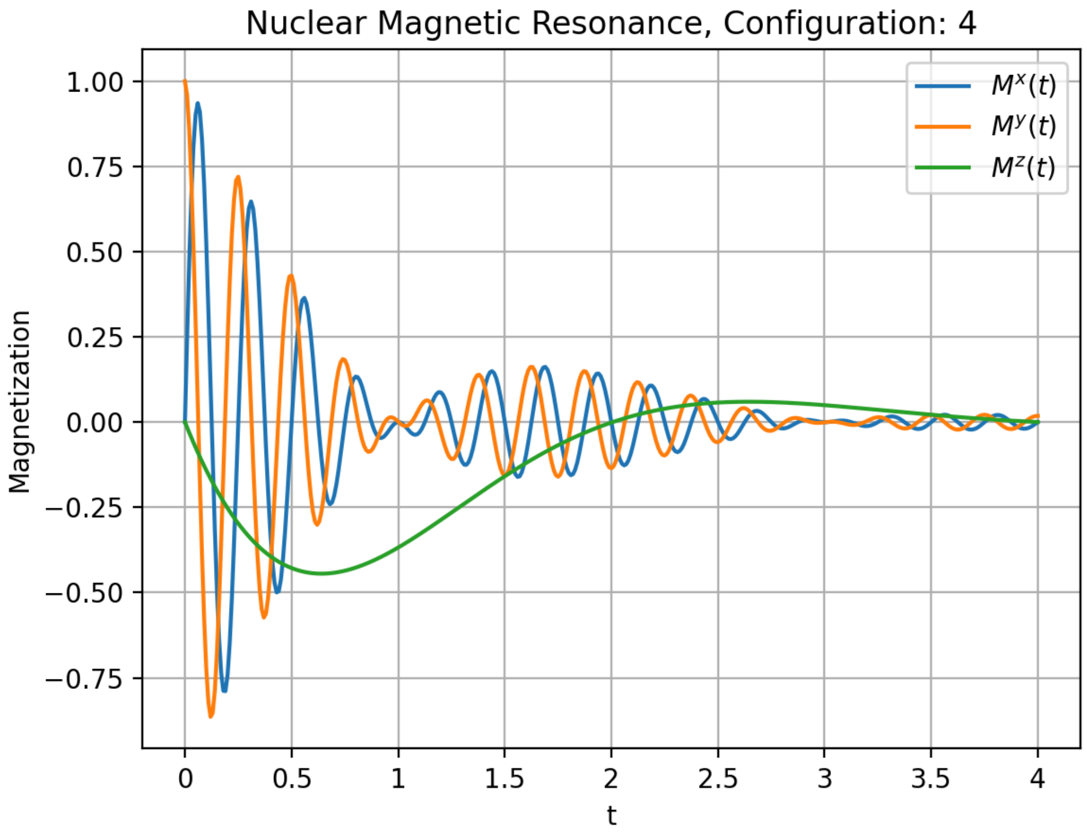
Figure 4: NMR simulation using Configuration 1. The configuration sets the initial values to $\overrightarrow{M}(0) = (0, 1, 0)$, $\varphi = 0.5 \cdot \pi$, $\frac{1}{T_1} = \frac{1}{T_2} = 0$. The z-axis stays constant at 0. The x-and y-axis oscillate between 1 and −1 without damping.

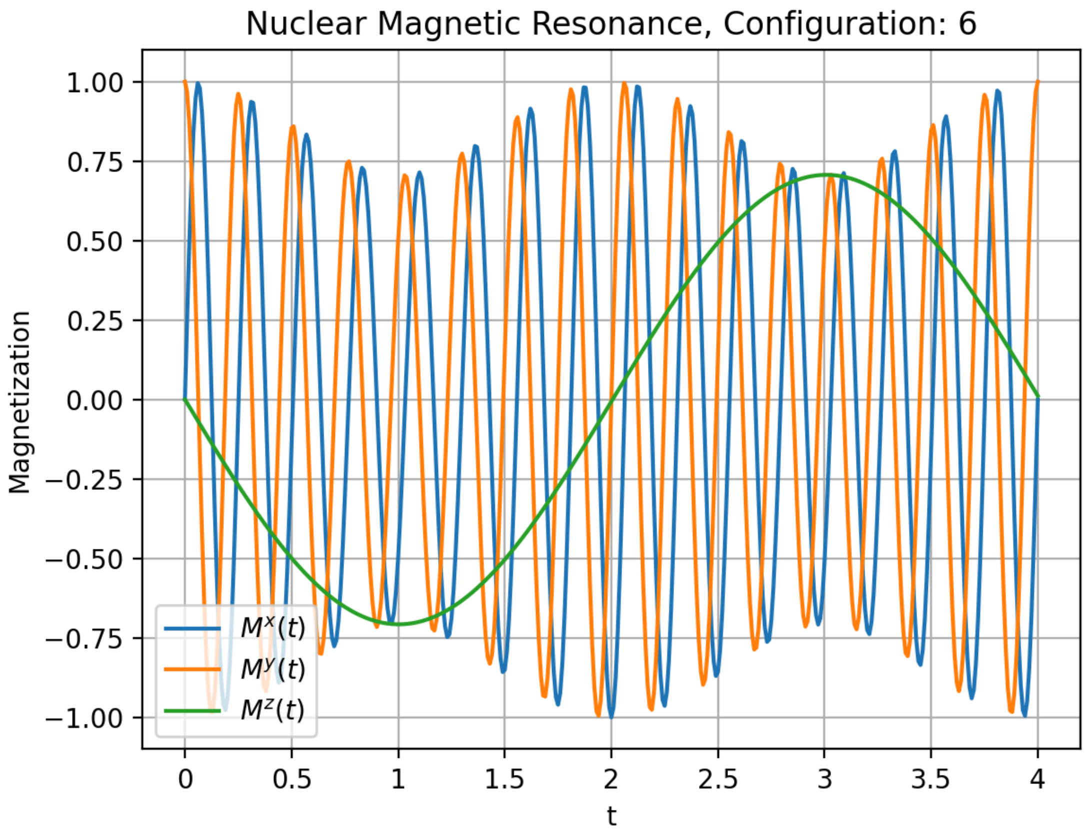
In Figure 4 we have introduced a phase shift in the magnetic field of $\varphi = \frac{\pi}{2}$. As we simulate a rotation by $\frac{\pi}{2}$, and relaxation is disabled, we can see that the z-axis remains constant, while the x-and y-axis are oscillating harmonically between 1 and −1 without damping.

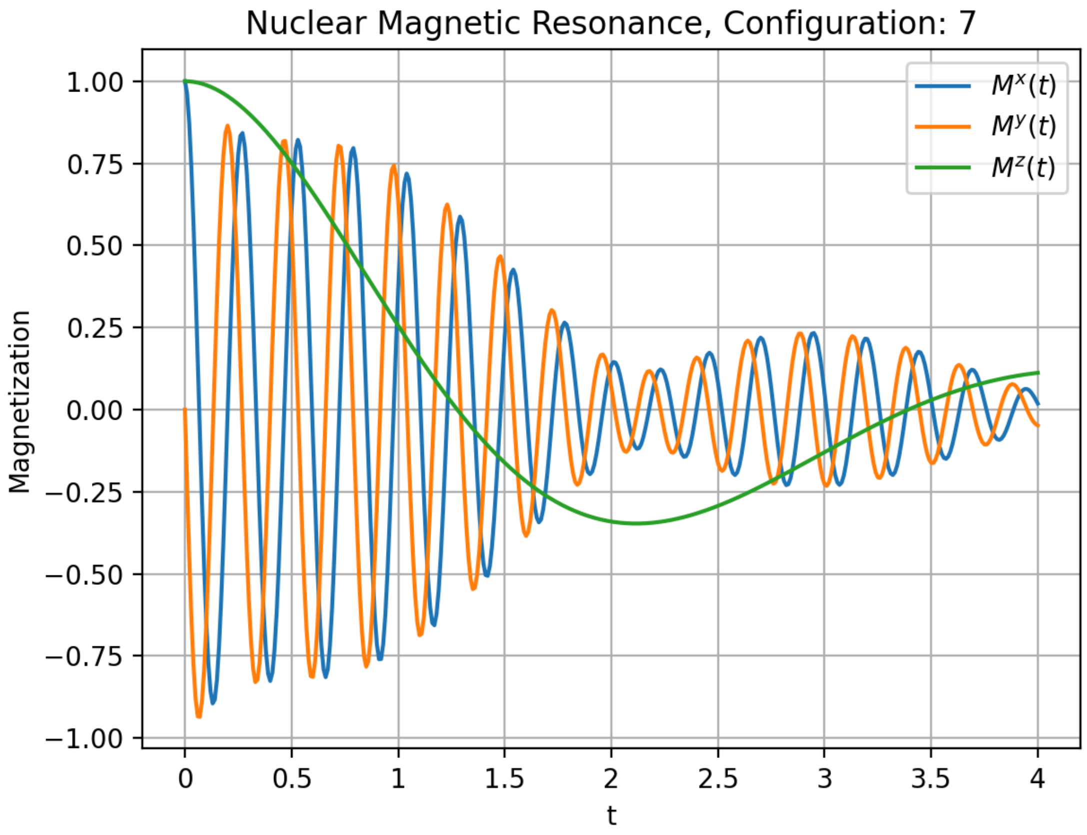
Additional Configuration

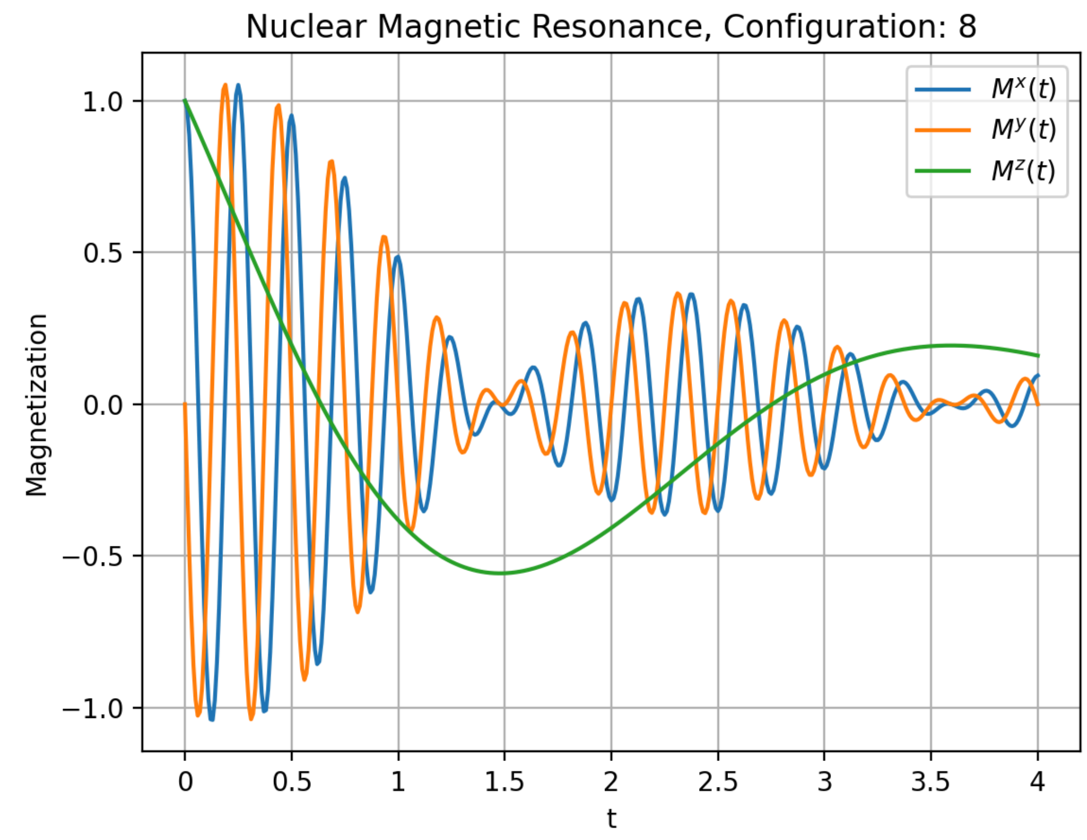
Further configurations do not yield further insights, we include the results here for completion. They all differ only in the initial values for φ, $\overrightarrow{M}(0)$, $\frac{1}{T_1}$, $\frac{1}{T_2}$. The interested reader can find the exact values in the configurations in the appendix.

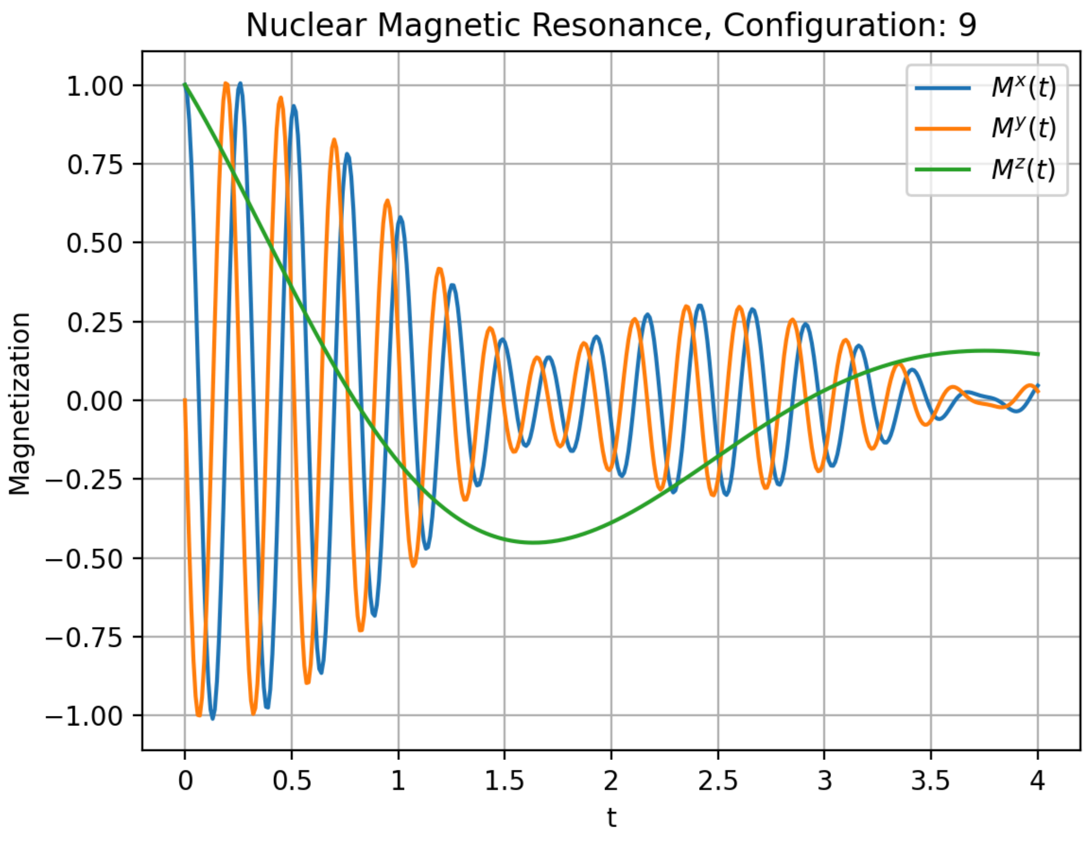

Nuclear Magnetic Resonance, Configuration: 3
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
0.5
1
1.5
2
2.5
3
3.5
4

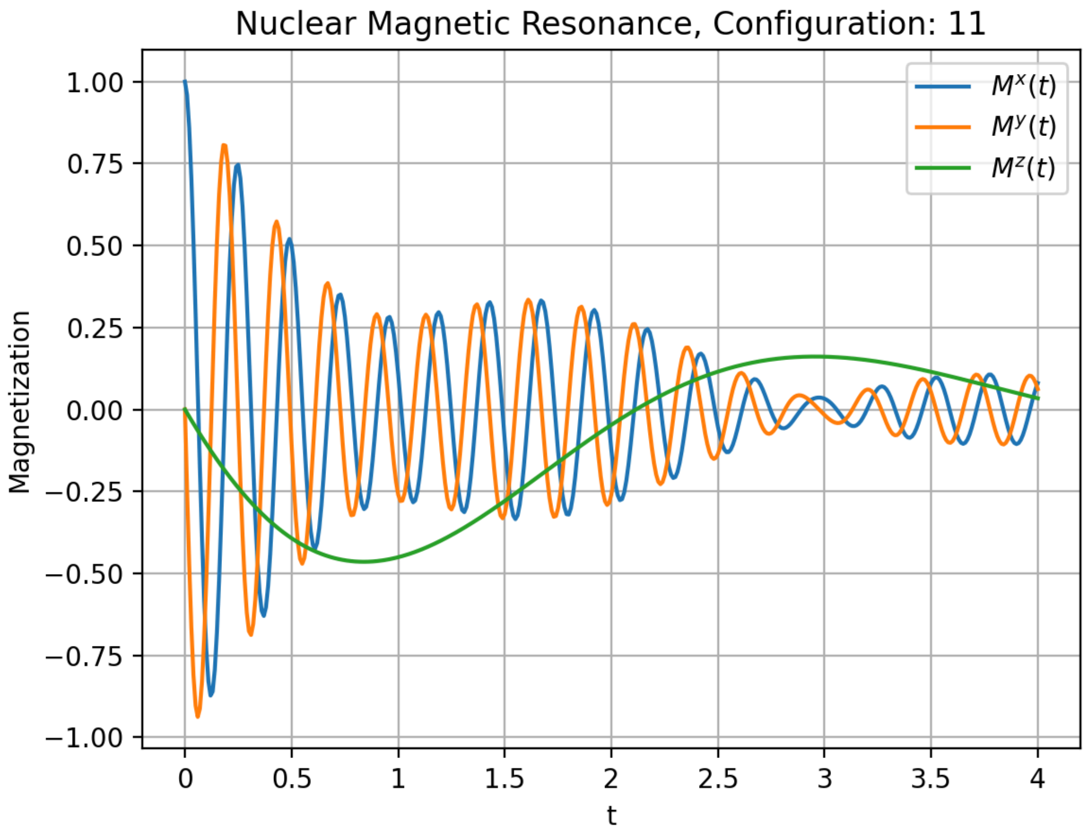

Nuclear Magnetic Resonance, Configuration: 4
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
0.5
1
1.5
2
2.5
3
3.5
4

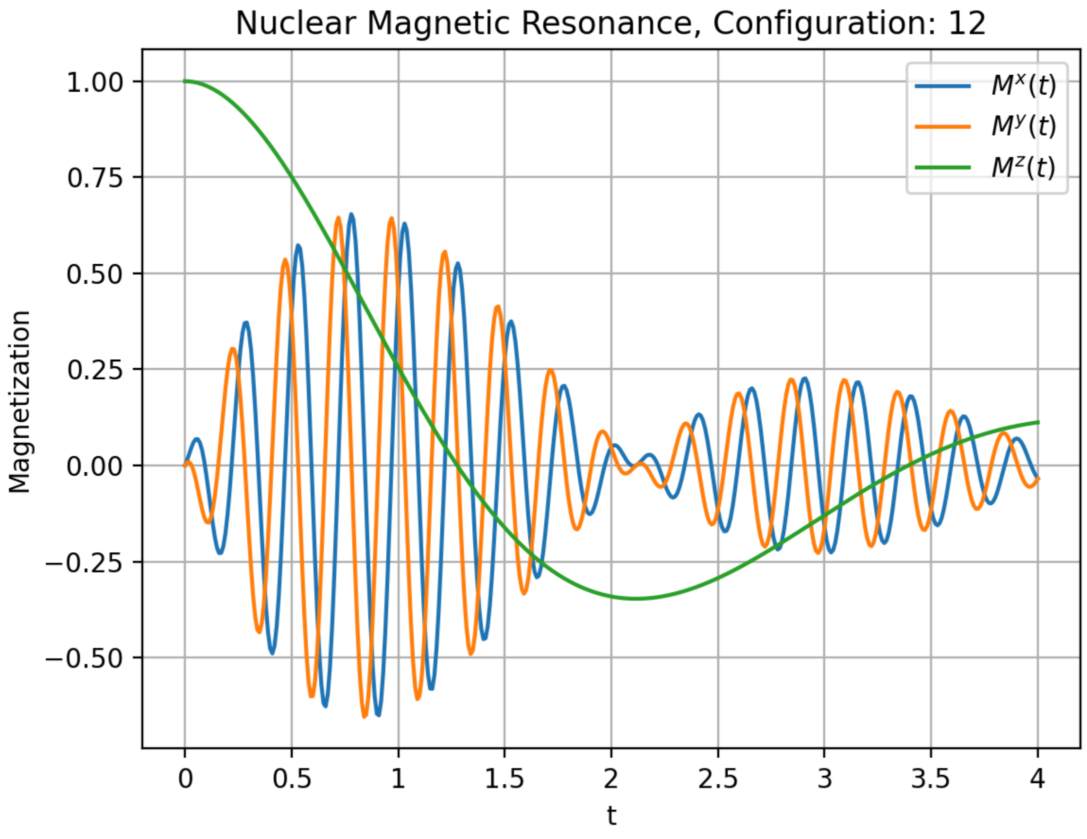

Nuclear Magnetic Resonance, Configuration: 6
Magnetization
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
0.5
1
1.5
2
2.5
3
3.5
4
t
$M^x(t)$
$M^y(t)$
$M^z(t)$

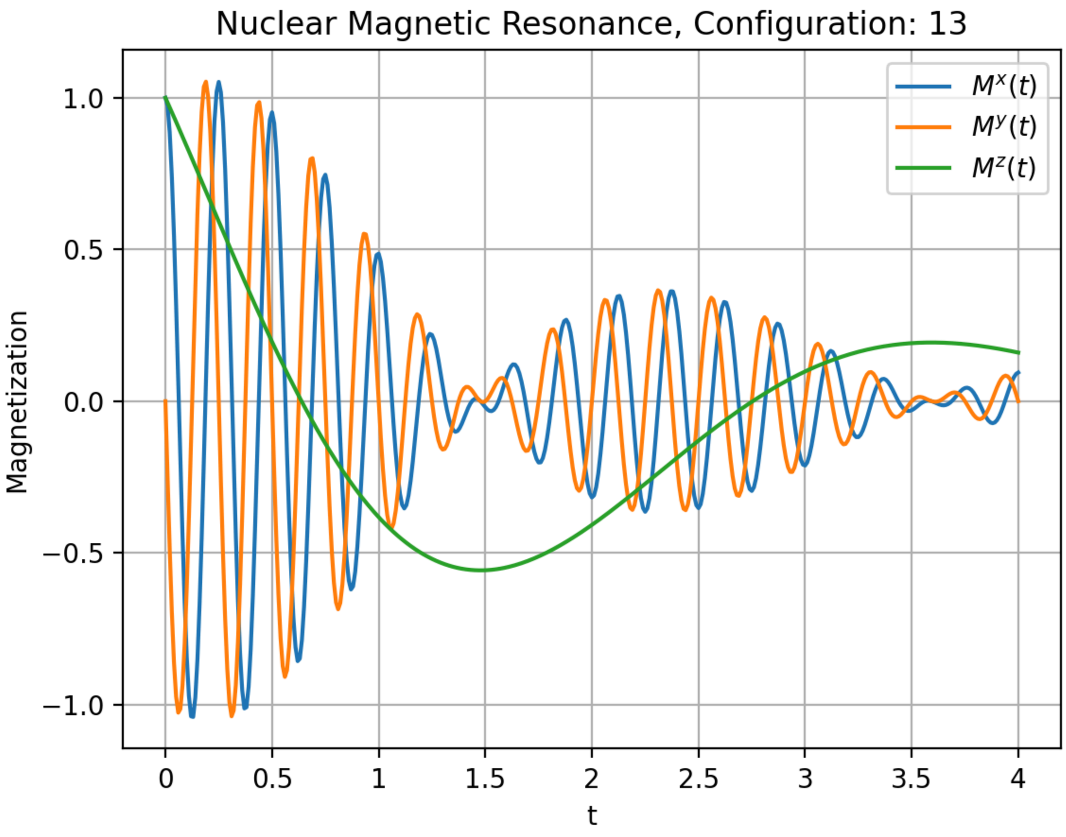

Nuclear Magnetic Resonance, Configuration: 7
Magnetization
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
0.5
1
1.5
2
2.5
3
3.5
4
t
$M^x(t)$
$M^y(t)$
$M^z(t)$

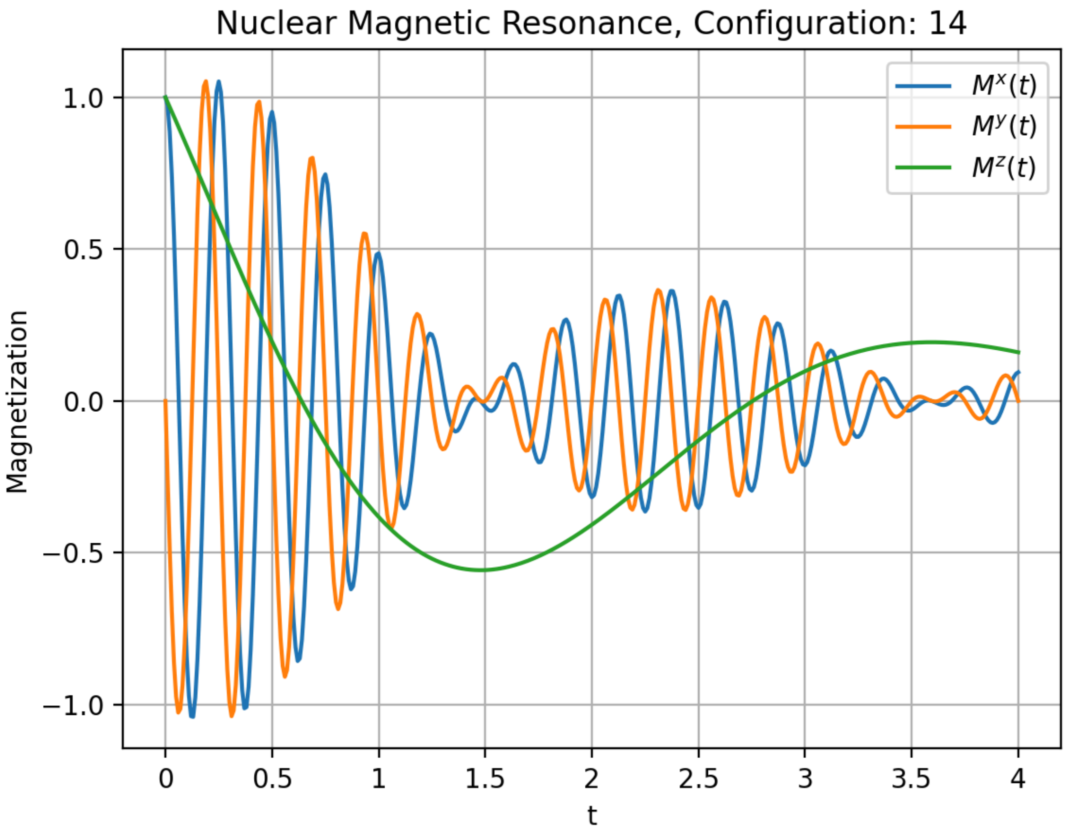

Nuclear Magnetic Resonance, Configuration: 8
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.0
0.5
0.0
−0.5
−1.0
0
0.5
1
1.5
2
2.5
3
3.5
4


Nuclear Magnetic Resonance, Configuration: 9
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
0.5
1
1.5
2
2.5
3
3.5
4


Nuclear Magnetic Resonance, Configuration: 11
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
0.5
1
1.5
2
2.5
3
3.5
4


Nuclear Magnetic Resonance, Configuration: 12
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0
0.5
1
1.5
2
2.5
3
3.5
4


Nuclear Magnetic Resonance, Configuration: 13
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.0
0.5
0.0
−0.5
−1.0
0
0.5
1
1.5
2
2.5
3
3.5
4


Nuclear Magnetic Resonance, Configuration: 14
$M^x(t)$
$M^y(t)$
$M^z(t)$
Magnetization
t
1.0
0.5
0.0
−0.5
−1.0
0
0.5
1
1.5
2
2.5
3
3.5
4

Discussion

In this exercise, we have explored the origins and applications of the Bloch equations, which describe the dynamics of nuclear magnetization in the presence of an external magnetic field. We examined both analytical and numerical solutions to these equations, focusing on, and implementing, the product formula approach for numerical simulations.

The Bloch equations are fundamental in NMR and describe how the components of the magnetization vector, $\vec{M}$, evolve over time under the influence of relaxation processes and external magnetic fields. By simulating different configurations, we observed how initial conditions and relaxation times affect the magnetization components M^x, M^y, and M^z.

In conclusion, the Bloch equations provide a robust framework for understanding the dynamics of nuclear magnetization in NMR. By simulating different configurations and initial conditions, we gained insights into the behavior of the magnetization vector and the effects of relaxation processes. These simulations underscore the importance of initial parameters and relaxation times in determining the coherence and equilibrium state of the system. This understanding is crucial for interpreting NMR signals and for the practical application of NMR techniques in various fields, including medical imaging.

Appendix

```cpp
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>

#define WITHOUT_NUMPY
#include "matplotlibcpp.h"

using namespace std;

// controll configuration usage
#define CONFIGURATION 0

namespace e1
{
// preliminaries
double constexpr pi = M_PI;
double constexpr f_0 = 4;
double constexpr f_1 = 0.25;
double constexpr B_0 = 2 * pi * f_0;
double constexpr h = 2 * pi * f_1;
double constexpr w_0 = B_0;
double constexpr y = 1;
std::vector<double> M_1 = {0, 0, -1};
double constexpr tau = 0.01;
double constexpr t_0 = 0.;
```

```
#if CONFIGURATION == 0
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 1.;
double constexpr T_1 = 0.;
double constexpr T_2 = 0.;
double phi = 0.;
#endif

#if CONFIGURATION == 1
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 0.;
double phi = 0.;
#endif
#if CONFIGURATION == 2
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.;
#endif
#if CONFIGURATION == 3
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 1.;
double constexpr T_2 = 0.;
double phi = 0.;
#endif
#if CONFIGURATION == 4
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 1.;
double constexpr T_2 = 1.;
double phi = 0.;
#endif
#if CONFIGURATION == 5
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 0.;
double phi = 0.5 * pi;
#endif
#if CONFIGURATION == 6
std::vector<double> M_0 = {0, 1, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 0.;
double phi = 0.25 * pi;
#endif
#if CONFIGURATION == 7 // example from exercise
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.;
#endif
#if CONFIGURATION == 8
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.5 * pi;
#endif
#if CONFIGURATION == 9
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.25 * pi;
#endif
#if CONFIGURATION == 10
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.5 * pi;
#endif
#if CONFIGURATION == 11
std::vector<double> M_0 = {1, 0, 0};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.25 * pi;
#endif
#if CONFIGURATION == 12
std::vector<double> M_0 = {0, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.25 * pi;
#endif
#if CONFIGURATION == 13
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.5 * pi;
#endif
#if CONFIGURATION == 14
std::vector<double> M_0 = {1, 0, 1};
double constexpr t_1 = 4.;
double constexpr T_1 = 0.;
double constexpr T_2 = 1.;
double phi = 0.5 * pi;
#endif

std::vector<double> B(double t) { return std::vector{h * std::cos(w_0 * (t - 0.5 * tau) + phi),
-h * std::sin(w_0 * (t - 0.5 * tau) + phi), B_0}; };

void NMR()
{
    double t = t_0;
    std::vector<double> M_t = M_0;
    std::vector<double> time_x((t_1 / tau) + 1);
    std::vector<double> time_y((t_1 / tau) + 1);
    std::vector<double> time_z((t_1 / tau) + 1);

    time_x[0] = M_0[0];
    time_y[0] = M_0[1];
    time_z[0] = M_0[2];
    // iterate over all time-steps
    for (int i = 1; i < (t_1 / tau) + 1; ++i)
    {
        t += tau;

        // calculation of e^(tau C / 2) * M
        M_t[0] *= std::exp(-tau * 0.5 * T_2);
        M_t[1] *= std::exp(-tau * 0.5 * T_2);
        M_t[2] *= std::exp(-tau * 0.5 * T_1);

        // calc current B
        std::vector<double> B_t = B(t);
        std::vector<std::vector<double>> e_B(3, std::vector<double>(3));
        // calc current omega vals
        double Omega_squared = (B_t[0] * B_t[0]) + (B_t[1] * B_t[1]) + (B_t[2] * B_t[2]);
        double Omega = std::sqrt(Omega_squared);

        // calculation of e^(tau B)
       e_B[0][0] = ((B_t[0] * B_t[0]) + (((B_t[1] * B_t[1]) + (B_t[2] * B_t[2])) * std::cos(Omega
* tau * y))) / (Omega_squared);
        e_B[0][1] = ((B_t[0] * B_t[1] * (1 - std::cos(Omega * tau * y))) + (Omega * B_t[2] *
std::sin(Omega * tau * y))) / (Omega_squared);
        e_B[0][2] = ((B_t[0] * B_t[2] * (1 - std::cos(Omega * tau * y))) - (Omega * B_t[1] *
std::sin(Omega * tau * y))) / (Omega_squared);

        e_B[1][0] = ((B_t[0] * B_t[1] * (1 - std::cos(Omega * tau * y))) - (Omega * B_t[2] *
std::sin(Omega * tau * y))) / (Omega_squared);
       e_B[1][1] = ((B_t[1] * B_t[1]) + (((B_t[0] * B_t[0]) + (B_t[2] * B_t[2])) * std::cos(Omega
* tau * y))) / (Omega_squared);
        e_B[1][2] = ((B_t[1] * B_t[2] * (1 - std::cos(Omega * tau * y))) + (Omega * B_t[0] *
std::sin(Omega * tau * y))) / (Omega_squared);

        e_B[2][0] = ((B_t[0] * B_t[2] * (1 - std::cos(Omega * tau * y))) + (Omega * B_t[1] *
std::sin(Omega * tau * y))) / (Omega_squared);
        e_B[2][1] = ((B_t[1] * B_t[2] * (1 - std::cos(Omega * tau * y))) - (Omega * B_t[0] *
std::sin(Omega * tau * y))) / (Omega_squared);
       e_B[2][2] = ((B_t[2] * B_t[2]) + (((B_t[1] * B_t[1]) + (B_t[0] * B_t[0])) * std::cos(Omega
* tau * y))) / (Omega_squared);

        // calculation of e^(tau B) * M
        auto M_old = M_t;
        M_t[0] = e_B[0][0] * M_old[0] + e_B[0][1] * M_old[1] + e_B[0][2] * M_old[2];
        M_t[1] = e_B[1][0] * M_old[0] + e_B[1][1] * M_old[1] + e_B[1][2] * M_old[2];
        M_t[2] = e_B[2][0] * M_old[0] + e_B[2][1] * M_old[1] + e_B[2][2] * M_old[2];

        // calculation of e^(tau C / 2) * M
        M_t[0] *= std::exp(-tau * 0.5 * T_2);
        M_t[1] *= std::exp(-tau * 0.5 * T_2);
        M_t[2] *= std::exp(-tau * 0.5 * T_1);

        // store for plotting
        time_x[i] = M_t[0];
        time_y[i] = M_t[1];
        time_z[i] = M_t[2];
    }

    // plotting
    namespace plt = matplotlibcpp;

    plt::plot(time_x, {{"label", "$M^x(t)$"}});
    plt::plot(time_y, {{"label", "$M^y(t)$"}});
    plt::plot(time_z, {{"label", "$M^z(t)$"}});

    std::string title = "Nuclear Magnetic Resonance, Configuration: ";
    title = title.append(std::to_string(CONFIGURATION));
    plt::title(title);


#if CONFIGURATION != 0
    std::vector<double> ticks = {0, 50, 100, 150, 200, 250, 300, 350, 400};
    std::vector<std::string> labels = {"0", "0.5", "1", "1.5", "2", "2.5", "3", "3.5", "4"};
    plt::xticks(ticks, labels);
#endif
#if CONFIGURATION == 0
    std::vector<double> ticks = {0, 20, 40, 60, 80, 100};
    std::vector<std::string> labels = {"0", "0.2", "0.4", "0.6", "0.8", "1"};
    plt::xticks(ticks, labels);
#endif

    plt::xlabel("t");
    plt::ylabel("Magnetization");
    plt::grid(true);
    plt::legend();
    // plt::show();
    std::string filename = "../figures/Configuration_";
    filename = filename.append(std::to_string(CONFIGURATION));
    plt::save(filename, 1080);
}
} // namespace e1

int main()
{
    e1::NMR();
    return 0;
}
```