Computational Physics - Exercise 9
Bloch equations

Christian Gorjaew (354259)
christian.gorjaew@rwth-aachen.de

Julius Meyer-Ohlendorf (355733)
julius.meyer-ohlendorf@rwth-aachen.de

July 7, 2020

1. Introduction

In the following we solve the Bloch equations using a `Product formula algorithm`. Different initial configurations of the magnetization, different relaxation times and different magnetic fields are considered. The resulting time evolution of the magnetization is simulated and investigated.

2. Simulation model and method

Model:
The Bloch equations describe the time evolution of the macroscopic magnetization $\vec{M}(t) = (M_x(t), M_y(t), M_z(t))$ under the influence of an external magnetic field $\vec{B} = (B_x, B_y, B_z)$:

$$\begin{aligned} \frac{dM_x}{dt} &= \gamma \left(B_z M_y - B_y M_z\right) - \frac{M_x}{T_2} \\ \frac{dM_y}{dt} &= \gamma \left(B_x M_z - B_z M_x\right) - \frac{M_y}{T_2} \\ \frac{dM_z}{dt} &= \gamma \left(B_y M_x - B_x M_y\right) - \frac{M_z - M_0}{T_1}, \end{aligned} \tag{1}$$

with the magnetization M_0 for $t \to \infty$, the gyromagnetic ratio γ, the dissipation time T_1 and dephasing time T_2.
We now show analytical results of the Bloch equations, which are important for the later interpretation of our numerical results.
If one assumes a static magnetic field in the z-direction $\vec{B} = (0, 0, B_0)$ and sets $\frac{1}{T_1} = 0$ and $M_0 = 0$, the solution of the Bloch equations is given by:

$$\vec{M}(t) = \begin{pmatrix} e^{-t/T_2} \left(M_x(t=0)\cos(\omega_0 t) + M_y(t=0)\sin(\omega_0 t)\right) \\ e^{-t/T_2} \left(-M_x(t=0)\sin(\omega_0 t) + M_y(t=0)\cos(\omega_0 t)\right) \\ const \end{pmatrix}, \tag{2}$$

with the larmor precession frequency $\omega_0 = \gamma B_0$. It describes the precession with exponential damping of the magnetization around the static magnetic field in the z-direction.
In a different setup one assumes a general magnetic field with a static component in the z-direction and time dependent components in the x- and y-direction $\vec{B} = (h\cos(\omega_0 t), h\sin(\omega_0 t), B_0)$ and sets $\frac{1}{T_1} = \frac{1}{T_2} = 0$. The solution can be derived by transforming to the frame of reference rotating with $\omega = \omega_0$ around the z-axis. The z-component of the magnetization is given by:

$$M_z(t) = M_z(t=0)\cos(\gamma h t) - M_y(t=0)\sin(\gamma h t) \tag{3}$$

This corresponds to a spiral like rotation around the z-axis. The direction of the magnetization is periodically moving from the positive and negative z-direction to the xy plane starting from the initial direction at $t = 0$.
We solve the Bloch equations assuming a strong magnetic field in the z-direction and time dependent magnetic fields in the x and y-direction:

$$\vec{B}(t) = \begin{pmatrix} h\cos(\omega_0 t + \phi) \\ -h\sin(\omega_0 t + \phi) \\ B_0 \end{pmatrix} \tag{4}$$

and set $B_0 = 2\pi f_0$, $f_0 = 4$, $h = 2\pi f_1$, $f_1 = 1/4$, $\omega_0 = \gamma B_0$, $\gamma = 1$ and $M_0 = 0$. For the initial magnetization configurations, the relaxation times and the phase shift of the magnetic field ϕ

we consider 4 different cases.
Case 1:
The initial magnetization is set to $\vec{M}(t=0) = (0,1,0)$ and the phase shift of the magnetic field to $\phi = 0$. This setup is modeled with different combinations of relaxation times $\frac{1}{T_1} = 0,0,1,1$ and $\frac{1}{T_2} = 0,1,0,1$.
Case 2:
In this setup we consider $\vec{M}(t=0) = (1,0,0)$ and the phase shift is set to $\phi = \pi/2$. The combinations of relaxation times are the same as in the first case.
Case 3:
All parameters as in case 2 except for $\vec{M}(t=0) = (0,0,1)$.
Case 4:
All parameters as in case 2 except for $\vec{M}(t=0) = (1,0,1)$.

Method:
In order to solve the Bloch equations numerically, they are considered in matrix form:

$$\frac{d}{dt}\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \gamma \underbrace{\begin{pmatrix} 0 & B_z & -B_y \\ -B_z & 0 & B_x \\ B_y & -B_x & 0 \end{pmatrix}}_{\boldsymbol{B}} \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} \tag{5}$$

If one assumes that the magnetic field $\vec{B}$ is constant over time, the formal solution with a time step τ is given by:

$$\vec{M}(t+\tau) = e^{\tau\gamma\boldsymbol{B}}\vec{M}(t) \tag{6}$$

The components of the matrix exponential are easily calculated (c.f. lecture 20, slide 58/59). In the case of a time dependent magnetic field $\vec{B}(t)$, one assumes the magnetic field to be constant within the time step τ. The solution can then be derived as:

$$\vec{M}(t+\tau) = e^{\tau\gamma\boldsymbol{B}(t+\tau/2)}\vec{M}(t) \tag{7}$$

Hence, the magnetization at time t is given by repeatedly applying the matrix exponential and assuming the magnetic field to be constant at each step:

$$\vec{M}(t) = e^{\tau\gamma\boldsymbol{B}(t-\tau/2)}e^{\tau\gamma\boldsymbol{B}(t-3\tau/2)}...e^{\tau\gamma\boldsymbol{B}(\tau/2)}\vec{M}(t) \tag{8}$$

This form defines the `Product formula algorithm`. In order to include relaxation times, one needs to replace $e^{\tau\gamma\boldsymbol{B}(t)}$ by $e^{\tau\gamma\boldsymbol{C}/2}e^{\tau\gamma\boldsymbol{B}(t)}e^{\tau\gamma\boldsymbol{C}/2}$, where

$$e^{\tau\gamma\boldsymbol{C}/2} = \begin{pmatrix} e^{-\tau/2T_2} & 0 & 0 \\ 0 & e^{-\tau/2T_2} & 0 \\ 0 & 0 & e^{-\tau/2T_1} \end{pmatrix} \tag{9}$$

In our simulations we use a time step of $\tau = 0.0025$ and perform $m = 2400$ product algorithm steps, corresponding to a simulation time of $t_{\text{sim}} = 6$.

3. Simulation results

The `python` code used for the simulation can be found in the Appendix A.

Case 1:
Our simulated magnetization as a function of time for case 1 can be seen figure 1.

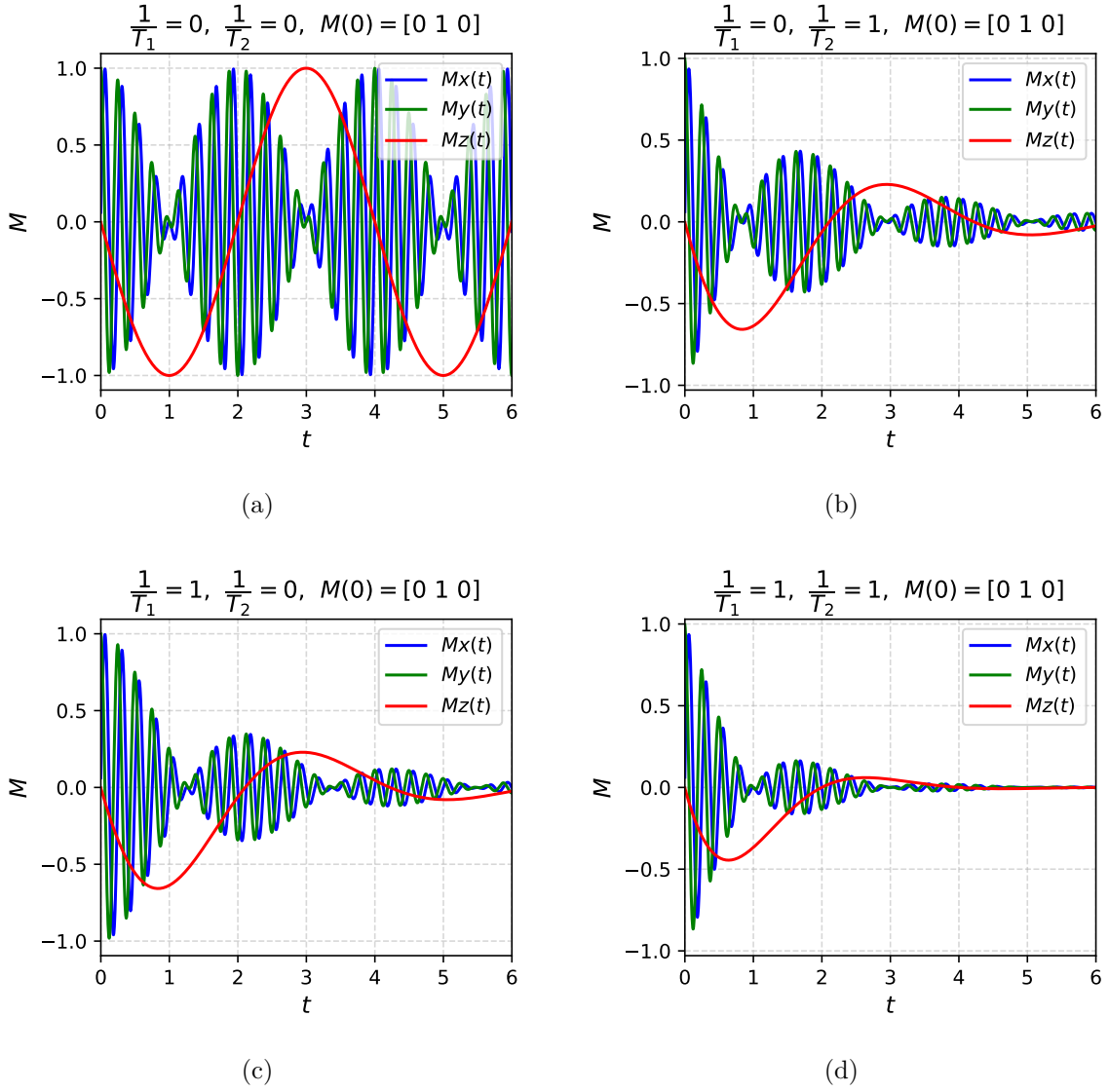
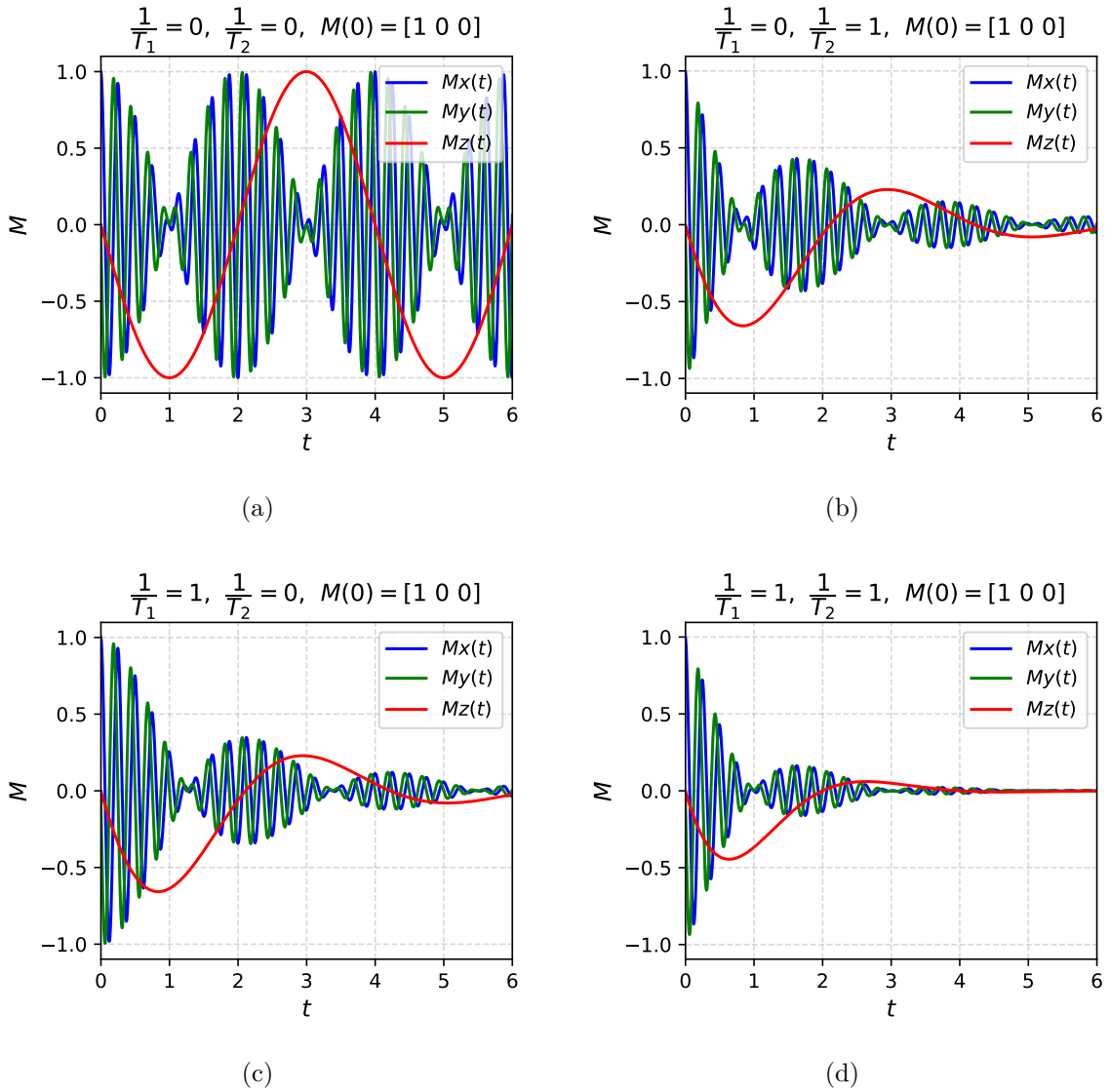


Figure 1: Simulated magnetization as a function of time for $\vec{M}(t=0) = (0,1,0)$ and different combinations of relaxation times. Simulation parameters: time step $\tau = 0.0025$, steps $m = 2400$, phase shift $\phi = 0$

The simulation setup in figure 1(a) with $\frac{1}{T_1} = \frac{1}{T_2} = 0$ corresponds to the setup for which we derived the analytical result of $M_z(t)$ (c.f. equation (3)). Our simulated result follows the analytical behaviour. At time $t = 0$ the magnetization is pointing in the y-direction. The system then starts to evolve and the magnetization vector precesses around the z-direction with the Larmor frequency ω_0 in a spiral like rotation. Therefore, the envelope of the x and y oscillations decreases to 0 at time $t = 1$, where the magnetization is pointing in the negative z-direction with $\vec{M}(t=0) = (0,0,-1)$. Hence, the magnetization is rotated by $\pi/2$. At time $t = 2$ the magnetization evolves to $\vec{M}(t=2) = (0,-1,0)$ and reaches $\vec{M}(t=3) = (0,0,1)$ at time $t = 3$.

Finally at $t = 4$ the magnetization returns to the initial direction of $\vec{M}(t = 0) = (0, 1, 0)$. This overall behaviour then periodically repeats over time. When relaxation times are included in the simulation, the components of the magnetization are exponentially damped (Figure 1(b)-1(d)). In the cases where $\frac{1}{T_1} = 1$ and $\frac{1}{T_2} = 0$ (c.f. Figure 1(c)) or $\frac{1}{T_1} = 1$ and $\frac{1}{T_2} = 1$ (c.f. Figure 1(d)) one observes, that the minima and maxima of the magnetization in the z-direction do not occur at the same time as the zeros of the x- and y-component's envelope. As one would expect the effect of the damping is strongest when both relaxation times are included (c.f. Figure 1(d)).

Case 2:



(a) (b)

(c) (d)

Figure 2: Simulated magnetization as a function of time for $\vec{M}(t = 0) = (1, 0, 0)$ and different combinations of relaxation times. Simulation parameters: time step $\tau = 0.0025$, steps $m = 2400$, phase shift $\phi = \pi/2$.

The magnetization behaves qualitatively as in case 1 except the phases of the x- and y-components inside the envelope are phase-shifted by $\pi/2$.

Case 3:

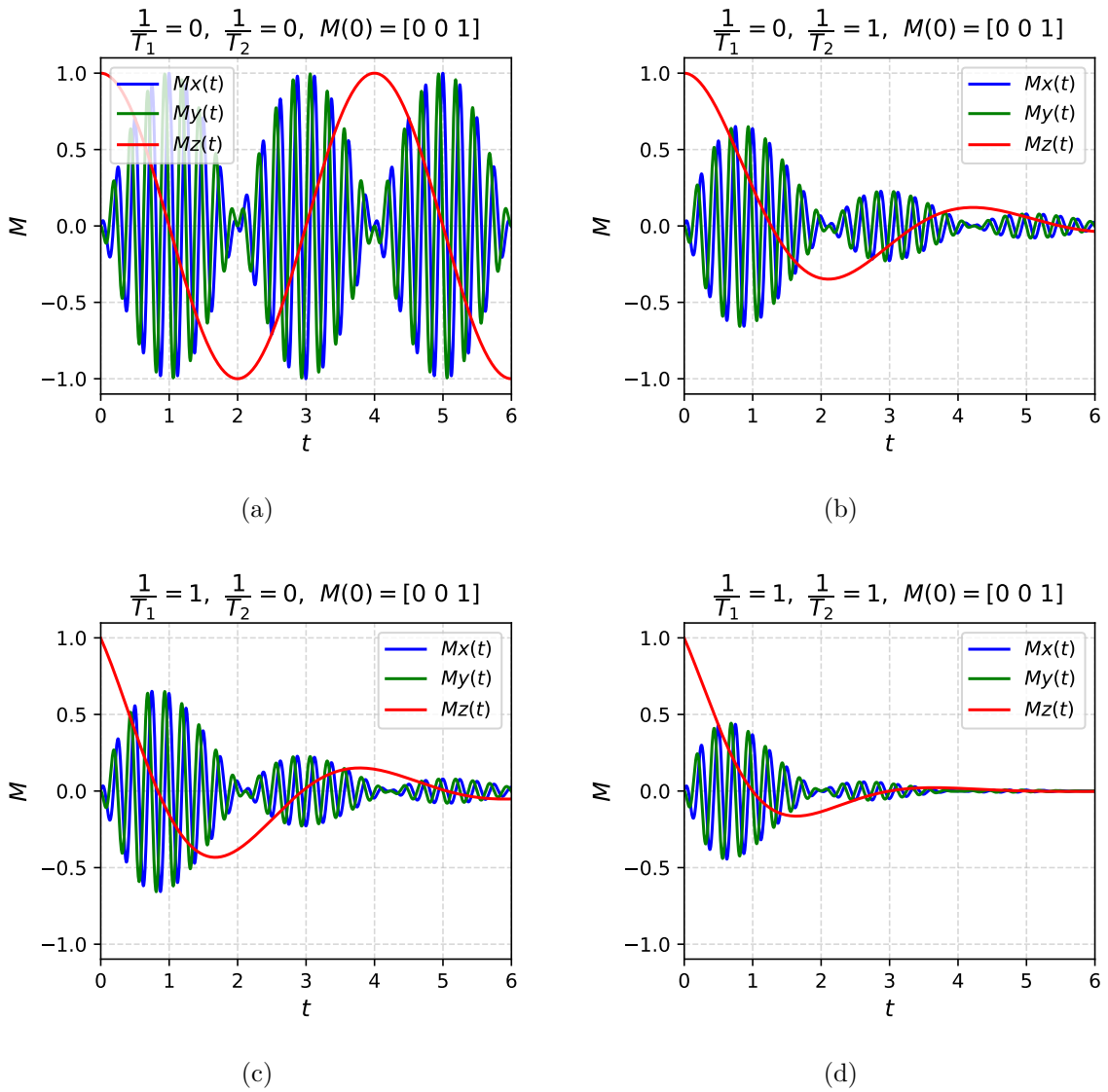


Figure 3: Simulated magnetization as a function of time for $\vec{M}(t=0) = (0,0,1)$ and different combinations of relaxation times. Simulation parameters: time step $\tau = 0.0025$, steps $m = 2400$, phase shift $\phi = \pi/2$.

In this case, the magnetization is initially oriented fully in the z-direction, while no magnetization is present in the x-y-plane. Effectively, this results phase shift of all components by $\pi/2$.

Case 4:

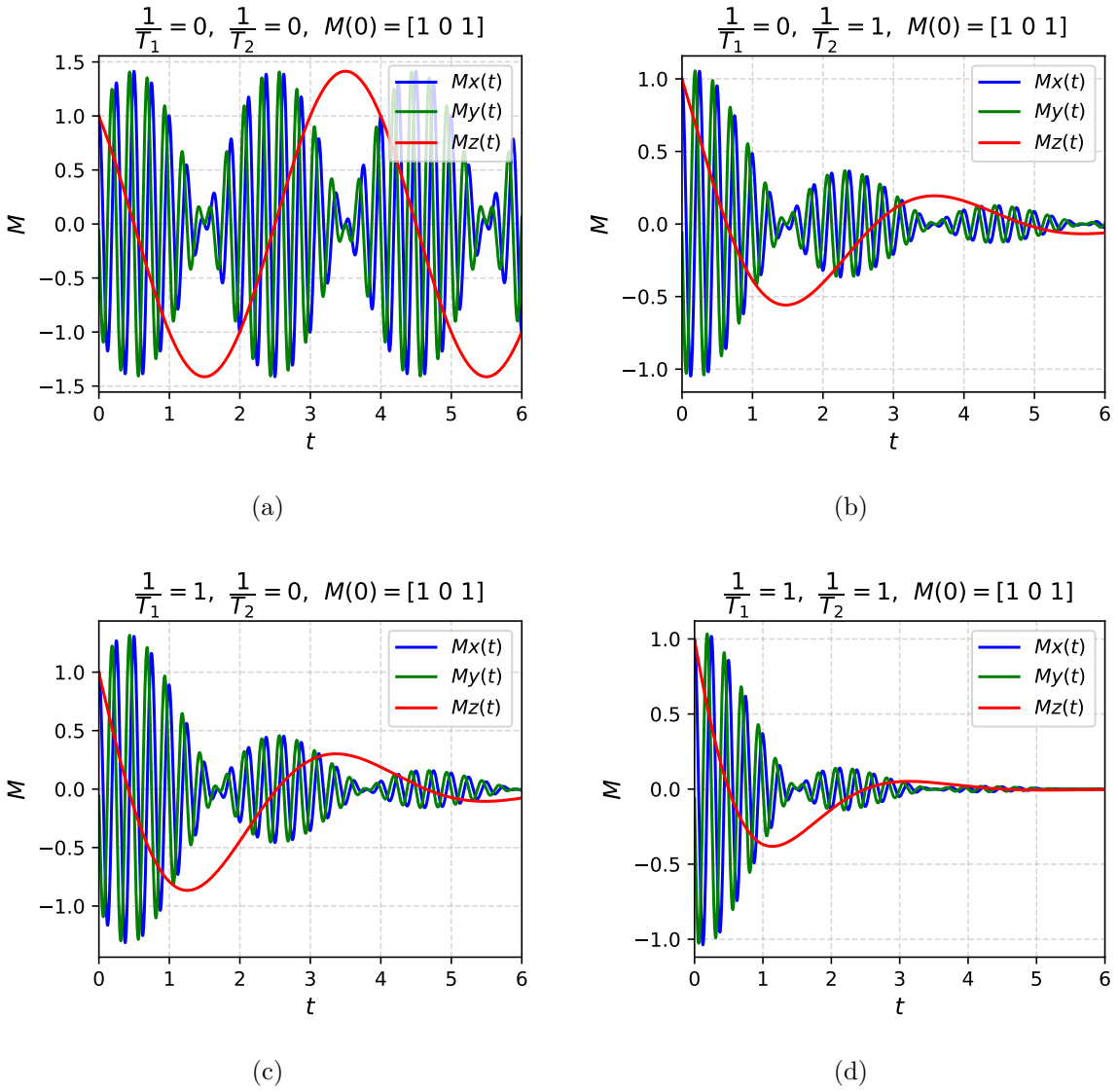


Figure 4: Simulated magnetization as a function of time for $\vec{M}(t=0) = (1, 0, 1)$ and different combinations of relaxation times. Simulation parameters: time step $\tau = 0.0025$, steps $m = 2400$, phase shift $\phi = \pi/2$

Here, the initial magnetization points equally into y- and z-directions. As before, the results for the parameters in this case leads to a phase shift of all components, in this case by $\pi/4$ when compared to case 1.

4. Discussion

The NMR simulations computed with the `product formula algorithm` show a behaviour that is expected from the theoretical calculations following from the Bloch equations. We could observe the Lamor precession of the x- and y-components of the magnetization that occur due to the presence of a constant magnetic field in the z-direction. Furthermore, the oscillation of the magnetization's z-component could be seen when a oscillating magnetic field in the x-y-plane, resonant with the Lamor frequency, is present. It could be seen, that the oscillation frequency corresponds to the field strength of the latter field. Presence of T_1 decay and T_2 dephasing lead to an exponential decay of the undisturbed curves describing the magnetization dynamics. Besides, it could be seen that different initial configurations of the magnetization effectively result in phase shifts.

A. Code used for simulation

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
import h5py as h5


def calc_B(t, h, omega_0, B_0, phi):
    """
    Calculating the magnetic field at time t
    """

    Bx = h * np.cos(omega_0 * t + phi)
    By = -h * np.sin(omega_0 * t + phi)
    Bz = B_0

    return(Bx, By, Bz)

def product_step(M, t, inv_T2, inv_T1, tau, gamma, phi, h, omega_0, B_0):
    """
    Performing one product algorithm step on given M
    """

    # calculating matrix exponential of B
    e_tB = np.zeros((3,3))
    Bx, By, Bz = calc_B(t + tau / 2, h, omega_0, B_0, phi)

    Omega_2 = Bx**2 + By**2 + Bz**2
    Omega = np.sqrt(Omega_2)
    cos_Ot = np.cos(Omega * tau * gamma)
    sin_Ot = np.sin(Omega * tau * gamma)

    e_tB[0,0] = (Bx**2 + (By**2 + Bz**2) * cos_Ot) / Omega_2
    e_tB[0,1] = (Bx * By * (1 - cos_Ot) + Omega * Bz * sin_Ot) / Omega_2
    e_tB[0,2] = (Bx * Bz * (1 - cos_Ot) - Omega * By * sin_Ot) / Omega_2
    e_tB[1,0] = (Bx * By * (1 - cos_Ot) - Omega * Bz * sin_Ot) / Omega_2
    e_tB[1,1] = (By**2 + (Bx**2 + Bz**2) * cos_Ot) / Omega_2
    e_tB[1,2] = (By * Bz * (1 - cos_Ot) + Omega * Bx * sin_Ot) / Omega_2
    e_tB[2,0] = (Bx * Bz * (1 - cos_Ot) + Omega * By * sin_Ot) / Omega_2
    e_tB[2,1] = (By * Bz * (1 - cos_Ot) - Omega * Bx * sin_Ot) / Omega_2
    e_tB[2,2] = ( Bz**2 + (Bx**2 + By**2) * cos_Ot) / Omega_2

    # calculating matrix exponential of C
    e_tC2_diag = np.array([np.exp(-tau / 2 * inv_T2),
                           np.exp(-tau / 2 * inv_T2),
                           np.exp(-tau / 2 * inv_T1)])
    e_tC2 = np.diag(e_tC2_diag, 0)

    # final matrix exponential including C
    e_tBfinal = e_tC2 @ e_tB @ e_tC2

    M_new = e_tBfinal @ M

    return(M_new)

# simulation parameters
f_0 = 4
B_0 = 2 * np.pi * f_0
f_1 = 1 / 4
h = 2 * np.pi * f_1
gamma = 1
omega_0 = gamma * B_0
```

1

```
###### Change parameters here ####
time_max = 6
tau = 0.0025
m = int(time_max / tau) # number of product steps
time_arr = tau * np.arange(0, m+1)

simulating = True # whether new simulation should be performed
plotting = True # whether results of simulation should be plotted

# which case to simulate
case1 = False
case2 = False
case3 = True
case4 = False

inv_T1_arr = np.array([0, 0, 1, 1])
inv_T2_arr = np.array([0, 1, 0, 1])

if case1:
    M_initial = np.array([0, 1, 0])
    phi = 0
    extra_string = 'case1'

if case2:
    M_initial = np.array([1, 0, 0])
    phi = np.pi / 2
    extra_string = 'case2'

if case3:
    M_initial = np.array([0, 0, 1])
    phi = np.pi / 2
    extra_string = 'case3'

if case4:
    M_initial = np.array([1, 0, 1])
    phi = np.pi / 2
    extra_string = 'case4'

M = M_initial

# excecuting the actual simulation
###################################

def simulation(M, inv_T1, inv_T2, path):
    """
    Performing simulaton loop and saving generated results
    """

    # setting up storage frame
    storage_steps = m + 1

    with h5.File(path, "a") as file:
        # setting up the storage file of type 'hdf5'
        file.create_dataset('t', shape=(storage_steps,), maxshape=(None,),
                            chunks=True, dtype=float)
        file.create_dataset('Mx', shape=(storage_steps,), maxshape=(None,),
                            chunks=True, dtype=float)
        file.create_dataset('My', shape=(storage_steps,), maxshape=(None,),
                            chunks=True, dtype=float)
        file.create_dataset('Mz', shape=(storage_steps,), maxshape=(None,),
```

```
                                chunks=True, dtype=float)

        # actual simulation loop
        for step in range(0, m+1):

            # perform a product step
            M = product_step(M, time_arr[step], inv_T2, inv_T1,
                             tau, gamma, phi, h, omega_0, B_0)

            file['t'][step] = time_arr[step]
            file['Mx'][step] = M[0]
            file['My'][step] = M[1]
            file['Mz'][step] = M[2]



if simulating:
    for i in range(len(inv_T1_arr)):
        print('Simulating loop:', str(i))
        # storage path
        path = './{0}_invT1_{1}_invT2_{2}.hdf5'.format(extra_string,
               inv_T1_arr[i], inv_T2_arr[i])
        simulation(M, inv_T1_arr[i], inv_T2_arr[i], path)


# creating plots of simulation
###################################

def plot_func(inv_T1, inv_T2, path):
    """
    Generates plot of Mx, My and Mz as function of time
    """

    data = h5.File(path, "r")

    t_arr = data['t'][:]
    Mx = data['Mx'][:]
    My = data['My'][:]
    Mz = data['Mz'][:]
    data.close()

    colors = ['b', 'g', 'r']

    fig, ax = plt.subplots(figsize=(4,3.5))
    title1 = '$\dfrac{{1}}{{T_1}}=${0},  $\dfrac{{1}}{{T_2}}=${1},  '.format(
              inv_T1, inv_T2)
    title2 = '$M(0)=${0}'.format(M_initial)
    title = title1 + title2
    ax.set_title(title, fontsize=12)
    ax.plot(t_arr, Mx, color=colors[0], label='$Mx(t)$')
    ax.plot(t_arr, My, color=colors[1], label='$My(t)$')
    ax.plot(t_arr, Mz, color=colors[2], label='$Mz(t)$')
    ax.set_xlim(0,t_arr[-1])
    if case3:
        ax.set_ylim(-1.1 * np.max(Mz), 1.1 * np.max(Mz))
    else:
        ax.set_ylim(-1.1 * np.max(Mx), 1.1 * np.max(Mx))
    ax.set_xlabel('$t$', fontsize=12)
    ax.set_ylabel('$M$', fontsize=12)
    ax.grid(linestyle='--', alpha=0.5)
    ax.legend(fontsize=10)
```

```
    fig.tight_layout()
    fig.savefig('{0}_invT1_{1}_invT2_{2}.pdf'.format(extra_string,
                inv_T1, inv_T2))



if plotting:
    for i in range(len(inv_T1_arr)):
        # storage path
        path = './{0}_invT1_{1}_invT2_{2}.hdf5'.format(extra_string,
               inv_T1_arr[i], inv_T2_arr[i])
        plot_func(inv_T1_arr[i], inv_T2_arr[i], path)
```