Approximate solution of the Bloch equation in the frequency domain

Written by
M.W.F.M. Bannenberg
Utrecht University
2015

Contents

1 Preface

Traditional magnetic resonance (MR) image reconstruction is based on the (inverse) Fourier transform operation. This way of MR imaging, by filling the k-space, is very time consuming and has to take into account effects caused by the relaxation and such. Recently new approaches,[2][3], to MR imaging has offered a possibility to achieve fast quantitative mapping. This technique is depending heavily on numerical approximations and scientific computations. One equation that has to be solved many times is the Bloch equation.[2]

In 2008 S. Balac and L. Chupin published an article[1] about a fast new way to solve the Bloch equation given some radio-frequency(RF) field inhomogeneities. The algorithm in this publication is much faster than any other widely used algorithm for solving the Bloch equation by finite differences schemes. However, in their algorithm they make use of a very basic RF-field with minor perturbations. But what if we want to compute the magnetisation vector for a more time-varying RF-field? Is it possible to do so by altering the given algorithm, and will the computation time still be significantly lower?

This thesis was written during a ten weeks long internship at UMCU, Image division, 7-Tesla group. Here I worked under the supervision of Alessandro Sbrizzi. It describes my steps towards creating an extended version of the Balac/Chupin algorithm: from analysing the published algorithm and implementing it in MATLAB to implementing the final algorithm into the work environment given by A. Sbrizzi. Note that until chapter four all formulas and computations are derived from the publication by Balac and Chupin[1].

In the results section I will compare my new algorithm with an old one and finally I will discuss where there is room for more improvement, if any.

2 Introduction

We want to solve the Bloch equation given a RF pulse $(B_x + iB_y)(t)$, a gradient $G(t)$, T_1, T_2, η_0. Suppose the pulse (and gradient) functions can be approximated by four Fourier terms. Then we can alter our Bloch equation in a way that it becomes an infinite set of differential equations. By truncating this infitinte set we are goint to approach the solution to the Bloch equation.

3 Analysing the publication

In this section we will walk through the publication: Fast approximate solution of Bloch equation for simulation of RF artifacts in Magnetic Resonance Imaging, by Balac and Chupin[1].

We start with the Bloch equation:

$$\frac{d\mathbf{M}}{dt} = \gamma(\mathbf{M} \times \mathbf{B}) - \frac{M_x\mathbf{x} + M_y\mathbf{y}}{T_2} - \frac{M_z - \eta_0}{T_1}\mathbf{z}$$

This equation calculates the nuclear magnetisation $\mathbf{M}$ as a function depending on time given relaxation times T_1, T_2. In this equation $\mathbf{M}$ is the nuclear magnetisation, $\mathbf{B}$ is a magnetic field with $\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_1$ where $\mathbf{B}_0$ is the main magnetic field in the scanner and $\mathbf{B}_1$ is the magnetic field induced by the radio-frequency(RF) pulse. The gyro-magnetic constant is given by γ and is 42.58Mhz per tesla, T_1 and T_2 are the relaxation times in seconds and η_0 is the value of the equilibrium magnetisation. We rewrite this equation in matrix form:

$$\frac{d}{dt}\mathbf{M}(t) = A(t)\mathbf{M}(t) + \mathbf{b}.$$

Where

$$A(t) = \begin{pmatrix} -\tau_2 & \gamma B_0 & -\gamma B_y(t) \\ -\gamma B_0 & -\tau_2 & \gamma B_x(t) \\ \gamma B_y(t) & -\gamma B_x(t) & -\tau_1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ \tau_1\eta_0 \end{pmatrix}, \ \tau_i = 1/T_i$$

Because it is more convenient to solve this differential system in the rotating frame we apply a change of unknown. We substitute $\mathbf{M}(t) = R(t)\mathbf{m}(t)$ with

$$R(t) = \begin{pmatrix} \cos(\omega_0 t) & \sin(\omega_0 t) & 0 \\ -\sin(\omega_0 t) & \cos(\omega_0 t) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

This substitution result in the following equation:

$$\frac{d}{dt}\mathbf{M}(t) = \tilde{A}(t)\mathbf{M}(t) + \mathbf{b}. \tag{1}$$

Where

$$\tilde{A}(t) = \begin{pmatrix} -\tau_2 & 0 & -\omega_a(t) \\ 0 & -\tau_2 & \omega_b(t) \\ \omega_a(t) & -\omega_b x(t) & -\tau_1 \end{pmatrix} \ , \ \mathbf{b} \ = \begin{pmatrix} 0 \\ 0 \\ \tau_1\eta_0 \end{pmatrix}.$$

with

$$\omega_a(t) = \gamma B_y(t)\cos(\omega_0 t) + \gamma B_x(t)\sin(\omega_0 t)$$
$$\omega_b(t) = \gamma B_x(t)\cos(\omega_0 t) - \gamma B_y(t)\sin(\omega_0 t)$$

Note that we assume that matrix components $(1,3)$ and $(2,3)$ are the real and imaginary components respectively of the RF-pulse. in the rotational frame. This differential system can be solved with numerical approximation. Now we are going to solve this differential system with a new algorithm. We start by noting that $\tilde{A}(t)$ is continuous and has periodic coefficients with period ω_0, this can be seen by the elements ω_a and ω_b. Using techniques inspired by Floquet theory we look for a formal solution of the following type:

$$\mathbf{m}(t) = \sum_{k\in\mathbb{Z}} \mathbf{m}_k(t)e^{2ik\omega_0 t}.$$

We rewrite matrix $\tilde{A}(t)$ into it's Fourier decomposition and thus $\tilde{A}(t) = \sum_{k=-1}^{1} A_{2k}e^{2ik\omega_0 t}$. In this sum we have:

$$A_0 = \begin{pmatrix} -\tau_2 & 0 & -\omega_a^{(0)} \\ 0 & -\tau_2 & \omega_b^{(0)} \\ \omega_a^{(0)} & -\omega_b^{(0)} & -\tau_1 \end{pmatrix} \text{ and } A_2 = \begin{pmatrix} 0 & 0 & -\omega_a^{(2)} \\ 0 & 0 & \omega_b^{(2)} \\ \omega_a^{(2)} & -\omega_b^{(2)} & 0 \end{pmatrix}$$

$$\omega_a^{(0)} = \frac{1}{2}\gamma(u_2+u_1),\ \omega_a^{(2)} = -\frac{1}{4}\gamma(v_1 - u_2 + i(u_1+v_2)),$$
$$\omega_b^{(0)} = \frac{1}{2}\gamma(2B_1 + u_1 - v_2),\ \omega_b^{(2)} = \frac{1}{4}\gamma(u_1 + v_2 + i(u_2 - v_1)),$$

with u_1, u_2, v_1, v_2 as the permutation factors, B_1 the strength of $\mathbf{B}_1$ and $A_{-2} = \bar{A}_2$ where $\bar{A}_2$ is the complex conjugate. Combining these two ideas into the matrix differential system (1) gives us: $\sum_{k\in\mathbb{Z}} \mathbf{r}_k(t)e^{2ik\omega_0 t} = 0$ where

$$\mathbf{r}_k(t) = \frac{d}{dt}\mathbf{m}_k(t) - \sum_{j=-1}^{1} A_{2j}\mathbf{m}_{k-j}(t) - 2ik\omega_0\mathbf{m}_k(t) - \delta_k\mathbf{b}$$

With the sequence $(\delta_k)_{k\in\mathbb{Z}}$, $\delta_0 = 1$ and $\delta_k = 0$. Due to the fact that $\mathbf{r}_k(t)$ is not a standard Fourier expansion it cannot be deduced that $\mathbf{r}_k = 0$ for all k in $\mathbb{Z}$. However, when we can solve this for all k under appropriate condition so that the series $\sum_{k\in\mathbb{Z}} \mathbf{m}_k(t)e^{2ik\omega_0 t}$ converges then the formal solution $\mathbf{m}(t)$ will be the solution to the differential system. If $\sum_{k\in\mathbb{Z}} \mathbf{m}(0) = \mathbf{M}_0$ then $\mathbf{m}(t)$ will be the solution to the differential system given that $\mathbf{m}(0) = \mathbf{M}_0$. We now have the following infinite system of differential equations:

$$k \in \mathbb{Z} = \begin{cases} \frac{d}{dt}\mathbf{m}_k(t) = \sum_{j=-1}^{1} A_{2j}\mathbf{m}_{k-j}(t) + 2ik\omega_0\mathbf{m}_k(t) + \delta_k\mathbf{b} \\ \mathbf{m}_k(0) = \delta_k\mathbf{M}_0 \end{cases}$$

An approximate solution of the infinite differential system can be obtained by solving a truncation of the infinite differential system, making it a finite differential system. We represent this system by:

$$\mathcal{M}^{[N]}(t) = \mathcal{A}^{[N]}\mathcal{M}^{[N]}(t) + \mathcal{B}^{[N]}$$

Here in this equation we have:

$$\mathcal{A}^{[N]} = \begin{pmatrix} A_0(N) & A_{-2} & & & & \\ A_2 & A_0(N-1) & A_{-2} & & & \\ & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & A_{-2} \\ & & & & A_2 & A_0(-N) \end{pmatrix}, \ \mathcal{B}^{[N]} = (\mathbf{b}_k)_{k \in -N,\dots,N}.$$

Where $A_0(k) = A_0 + 2ik\omega_0 Id$. To this system of differential equations we can apply Duhamel's formula stating that the solution is given by:

$$\mathcal{M}^{[N]}(t) = e^{t\mathcal{A}^{[N]}}\mathcal{M}^{[N]}(0) + (\int_0^t e^{(t-s)\mathcal{A}^{[N]}}ds)\mathcal{B}^{[N]}$$

If the matrix $\mathcal{A}^{[N]}$ is diagonalisable with $\mathcal{A}^{[N]} = \mathcal{P}\mathcal{D}\mathcal{P}^{-1}$ then we have:

$$\mathcal{M}^{[N]}(t) = \mathcal{P}e^{t\mathcal{D}}\mathcal{P}^{-1}\mathcal{M}^{[N]}(0) + \mathcal{P}\mathcal{S}(t)\mathcal{P}^{-1}\mathcal{B}^{[N]}.$$

4 Implementation of the algorithm

In this chapter we are going to go through the algorithm step-by-step and implement it with MATLAB. The algorithm start with the option to choose the truncation order N and then we give the variables given by our RF-pulse. These are T_{RF} , the B_x coefficients and the B_y coefficients. In the MATLAB file , appendix A, seen as the variables `N` and `T_rf`. The next step is to compute the eigenvalues and eigenvectors of $\mathcal{A}^{[N]}$, which is denoted with `E`. To do this we must first create this matrix. We start with `E = zeros(3*(2*N+1),3*(2*N+1));`. This gives us an empty matrix with the correct dimensions. With a loop we are going to fill this matrix with the correct values.

```
for n = 0: 2*N
E(n*3+1:n*3+3,n*3+1:n*3+3) =ANUL_INIT+2*1i*(N-n)*omega_0*eye(3,3);
    if n < 2*N
        E(n*3+1:n*3+3,(n+1)*3+1:(n+1)*3+3) = conj(A_TWO);
        E((n+1)*3+1:(n+1)*3+3,n*3+1:n*3+3) = A_TWO;
    end
end
```

Here we use the following matrices:

```
ANUL_INIT = [-tau_2          0          gamma*-omega_a_0;
```

```
                    0          -tau_2              gamma*omega_b_0;
                    gamma*omega_a_0 gamma*-omega_b_0  -tau_1];



A_TWO       = [0                0                  gamma*-omega_a_2;
               0                0                  gamma*omega_b_2;
               gamma*omega_a_2 gamma*-omega_b_2   0];
```

Now we use `[P,D] = eig(E);` to compute the eigenvalues and the eigenvectors of $\mathcal{A}^{[N]}$. With `P` and `D` now known we create the diagonal matrix $\mathcal{S}$ consisting of s_i's through the following loop:

```
s = zeros(1,3*(2*N+1));
for n = 1:3*(2*N+1)
    if D(n,n) == 0
        s(n) = T_rf;
    else
        s(n) = (exp(T_rf*D(n,n)) - 1)/D(n,n);
    end
end
S = diag(s);
```

With all the ingredients known there are only two computations left, note that the e^{tD} is replaced by `Dexp`. Because by definition D is a diagonal matrix we can simply create e^{tD} by `diag(exp(T_rf*diag(D)));`.

```
Dexp = diag(exp(T_rf*diag(D)));
M_NEAT = (P*Dexp*(P\M_0')) + (P*S*(P\B_neat'));
m_small = zeros(3,1);

for k = -N : N
    m_small = m_small +
    M_NEAT(3*(N+k)+1:3*(N+k)+3)*exp(2*1i*k*omega_0*T_rf);
end
```

Note thate here we use `(P\B_neat')` instead of `inv(P)`. This option is chosen because it is faster and more accurate. We can observe that in the calculation of $\mathcal{M}^{[N]}(T_{RF})$ is just applying the given formula. For $\mathbf{m}(T_{RF})$ we use a small loop to sum. Because we are intrested in the rotating frame solution the last step of the algorithm is superfluous and we have already have the desired solution. In Figure 1 we see our plotted solution. We plot the solution throughout the interval. This is just for visual effect. The algorithm can solve the Bloch equation for a given time without using the time interval starting with zero.

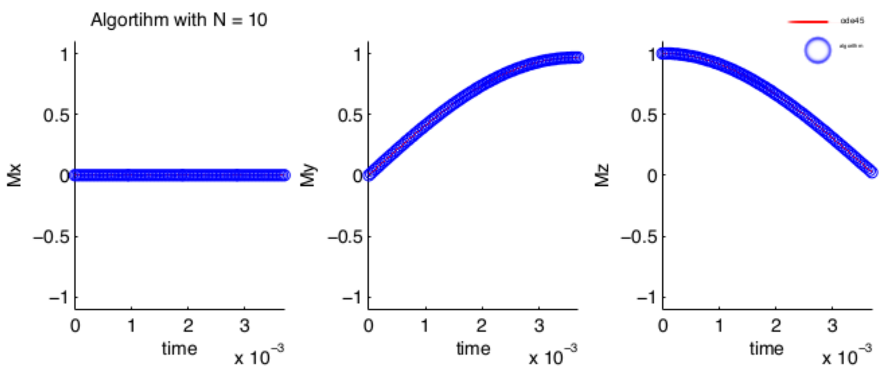

Figure 1: The solution of the algorithm
$\mathbf{M}(T_r f) = (0, 0.9999996, 2.505088\text{e}{-7})$ on T_{RF}
with $\mathbf{M}_0^T = (0,0,1)$

5 Expansion of the algorithm

In this chapter we are goin to expand the algorithm so that the algorithm is able to accept more time-varying RF-pulses. To do so we first assume that

$$\tilde{B}_y = \omega_a$$

$$\tilde{B}_x = \omega_b$$

Where $\tilde{B}_x$ and $\tilde{B}_y$ are the real and imaginary part of the RF-pulse, $RF = \gamma\tilde{B}_x + i\gamma\tilde{B}_y$, in the rotation frame respectively. With $\tilde{B}_x$ and $\tilde{B}_y$ in $\mathbb{R}$. For our expansion we start with the following equation:

$$R(t)\frac{d}{dt}\mathbf{m}(t) = \tilde{A}\mathbf{m}(t) + \mathbf{b}$$

where

$$\tilde{A}(t) = \begin{pmatrix} -\tau_2 & 0 & -\gamma B_y \\ 0 & -\tau_2 & \gamma B_x \\ \gamma B_y & -\gamma B_x & -\tau_1 \end{pmatrix}$$

Where B_y and B_x are our RF-pulse components. We can rewrite $\tilde{A}$ into a sum: $\tilde{A}(t) = \sum_{i=-1}^{1} A_{2k}e^{2ik\omega_0 t} = A_{-2}e^{-2ik\omega_0 t} + A_0 + A_2 e^{2ik\omega_0 t}$ with

$$A_0 = \begin{pmatrix} -\tau_2 & 0 & -\omega_a^{(0)} \\ 0 & -\tau_2 & \omega_b^{(0)} \\ \omega_a^{(0)} & -\omega_b^{(0)} & -\tau_1 \end{pmatrix} \text{ en } A_2 = \begin{pmatrix} 0 & 0 & -\omega_a^{(2)} \\ 0 & 0 & \omega_b^{(2)} \\ \omega_a^{(2)} & -\omega_b^{(2)} & 0 \end{pmatrix}$$

and $A_{-2} = \bar{A}_2$ is the complex conjugate matrix of A_2. For the expansion of the algorithm we would like to add more terms of $e^{2ik\omega_0 t}$. This can be done by adding more terms in the sum and thus more matrices. If we want to add two matrices A_{-4} and A_4 we expand the sum with two terms: $\tilde{A}^*(t) = \sum_{i=-2}^{2} A_{2k} e^{2ik\omega_0 t}$. This gives us:

$$\tilde{A}^* = \tilde{A} + \begin{pmatrix} 0 & 0 & -\gamma\bar{m_a} \\ 0 & 0 & \gamma\bar{m_b} \\ \gamma\bar{m_a} & -\gamma\bar{m_b} & 0 \end{pmatrix} e^{-4i\omega_0 t} + \begin{pmatrix} 0 & 0 & -\gamma m_a \\ 0 & 0 & \gamma m_b \\ \gamma m_a & -\gamma m_b & 0 \end{pmatrix} e^{4i\omega_0 t}.$$

Here we have $m_a = m_{a_1} + m_{a_2} i$ en $m_b = m_{b_1} + m_{b_2} i$ as components of the matrices A_{-4} and A_4. If we simplify this equation we can do this in general by the following equality:

$$-\bar{m_a} e^{-4i\omega_0 t} - m_a e^{4i\omega_0 t} = 2m_{a_2}\sin(4\omega_0 t) - 2m_{a_1}\cos(4\omega_0 t)$$

Applying this results in the following matrix:

$$\tilde{A}^* = \begin{pmatrix} -\tau_2 & 0 & a \\ 0 & -\tau_2 & b \\ c & d & -\tau_1 \end{pmatrix}$$

$$a = -\gamma B_y + 2m_{a_2}\sin(4\omega_0 t) - 2m_{a_1}\cos(4\omega_0 t)$$
$$b = \gamma B_x + 2m_{b_1}\cos(4\omega_0 t) - 2m_{b_2}\sin(4\omega_0 t)$$
$$c = \gamma B_y + 2m_{a_1}\cos(4\omega_0 t) - 2m_{a_2}\sin(4\omega_0 t)$$
$$d = -\gamma B_x + 2m_{b_2}\sin(4\omega_0 t) - 2m_{b_1}\cos(4\omega_0 t)$$

We see that we now know how to add terms of $\sin(4\omega_0 t)$ and $\cos(4\omega_0 t)$ to our RF-pulse. We can apply the same technique to the coefficients of A_{-2} and A_2. This gives us the complete freedom to represent RF-pulses which is in the form of:

$$\gamma B_y = -c_1 + 2(m_{2_{a_2}}\sin(2\omega_0 t) - m_{2_{a_1}}\cos(2\omega_0 t)) + 2(m_{4_{a_2}}\sin(4\omega_0 t) - m_{4_{a_1}}\cos(4\omega_0 t))$$

$$\gamma B_x = c_2 + 2(m_{2_{b_1}}\cos(2\omega_0 t) - m_{2_{b_2}}\sin(2\omega_0 t)) + 2(m_{4_{b_1}}\cos(4\omega_0 t) - m_{4_{b_2}}\sin(4\omega_0 t))$$

Although this is the result for only five terms in the sum, we can apply this technique to create even more terms. In MATLAB code this means we add another matrix `AFOUR`:

```
omega_a_0 = B_y(1);
omega_a_2 = B_y(2);
omega_a_4 = B_y(4);

omega_b_0 = B_x(1);
omega_b_2 = B_x(2);
omega_b_4 = B_x(4);
```

```
ANUL_INIT = [-tau_2            gamma*G*x          gamma*-omega_a_0;
             -gamma*G*x         -tau_2             gamma*omega_b_0;
             gamma*omega_a_0  gamma*-omega_b_0   -tau_1];
A_TWO     = [0                0                  gamma*-omega_a_2;
             0                0                  gamma*omega_b_2;
             gamma*omega_a_2  gamma*-omega_b_2   0];
A_FOUR    = [0                0                  gamma*-omega_a_4;
             0                0                  gamma*omega_b_4;
             gamma*omega_a_4  gamma*-omega_b_4   0];
```

Note that in matrix `ANUL_INIT` there are new terms: $\pm$`gamma*G*x`. These terms represent the gradient field present in the domain. This gradient is spatial dependent and not time dependent. This means that it is constant for each voxel and thus the corresponding matrix elements remain constant.

6 Results

With the extended algorithm implemented we would like to see some results. First we check if the algorithm is functioning correctly and we test the algorithm versus the ode45 algorithm. Next we will implement the algorithm so that it can be spatially dependent. To do so we will loop over the field of view(FOV). The field of view is a given length in centimeters. The MATLAB of the new expanded algorithm is given in appendix A.

To check whether the extended algorithm is an accurate approximation of the solution we use the standard ordinary differential equation given by MATLAB: the ode45. The MATLAB code we use to compare the two methods is given in appendix B. We let the ode45 solve the differential equation given by (9) in [1]. The MATLAB code for the function called by ode45 is given in appendix C.

The RF-pulse seen in Figure 2 is constructed by:

```
B_x(1) = 1;
B_x(2) = 1/2;
B_x(4) = 1/2;
B_x(6) = 1;
B_x(8) = 1;
```

In Figure 3 we see the changes of the magnetisation vector during the RF-pulse. In this figure it is the ode45 solution is the line and the new algorithm solutions are the circles. The computation time of the ode45 = 0.364 seconds.The computation time of the new algorithm = 0.0123 seconds, the difference is measured between the two magnetisation solutions at time T_{RF}. This is almost thirty times faster, and this is a comparison of a highly optimised ode45 algorithm versus a new, not yet optimised, for instance we could use a faster eigenvalue calculation, algorithm. The maximum absolute difference between the two solutions equals: $1.272 * 10^{-4}$. If we leave all parameters the same but increase

N to 30. The difference between the two solutions equals: $6.67 * 10^{-10}$ and the computation times for ode45 and the algorithm are 0.47 and 0.041 respectively. This means that for nine digits accuracy we are almost ten times faster.
We now create a spatial plot of the magnetisation and let the gradient be spatial dependent with $G(x) = Gx$. For this experiment we used the same RF-pulse as the in the previous experiment, Figure 2.The MATLAB code for creating this is given in appendix D. In Figure 5 we see the magnetisation with a field of view of 10 centimeters. In Table 1 we compare the computation times of the algorithm with different truncation orders. In the third column we take the maximum difference between the spatial solution computed by the algorithm with N terms given by the first column and the reference solution computed by the algorithm with N = 100.

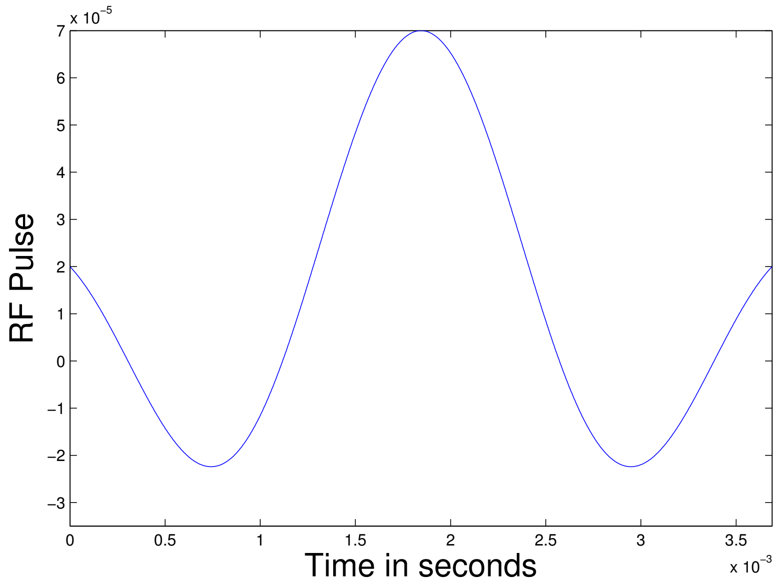

Figure 2: RF pulse

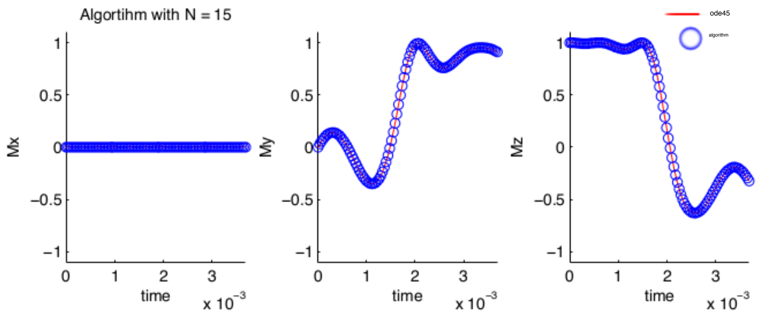

Figure 3: magnetisation

N	Difference with N = 100 algorithm	Computiation time of the algorithm in seconds	Computation time of ode45 in seconds
10	0.0185	0.699	3.377
15	0.00125	1.376	4.284
20	4.262e-05	2.407	6.727
25	4.014e-07	3.848	15.188
30	1.265e-08	5.693	23.398
35	1.280e-10	9.544	56.390
40	2.898e-12	15.626	139.387
45	9.5034e-14	22.622	356.507
50	1.0603e-13	29.563	567.469

Table 1: Computation time and difference, in Figure 7 we see a plot of this table.

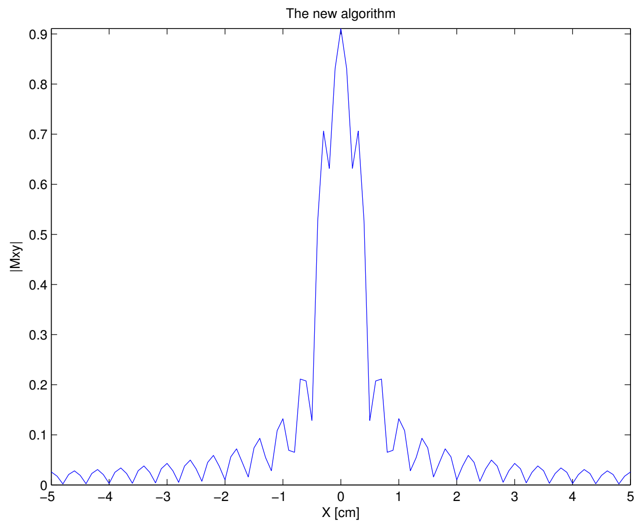

Figure 4: Magnetisation at $T_r f = 0.0037$ over the FOV, 10cm wide, computed with the new algorithm

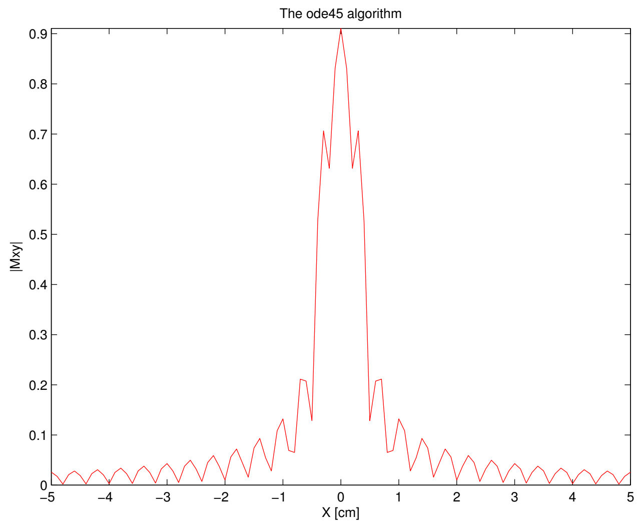

Figure 5: Magnetisation at $T_r f = 0.0037$ over the FOV, 10cm wide, computed with the ode45 algorithm

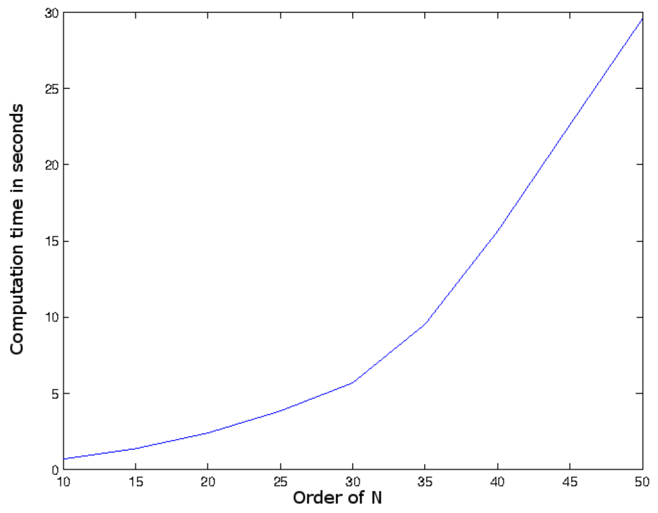

Figure 6: The left axis is the computation time in seconds and the horizontal axis is the truncation order N

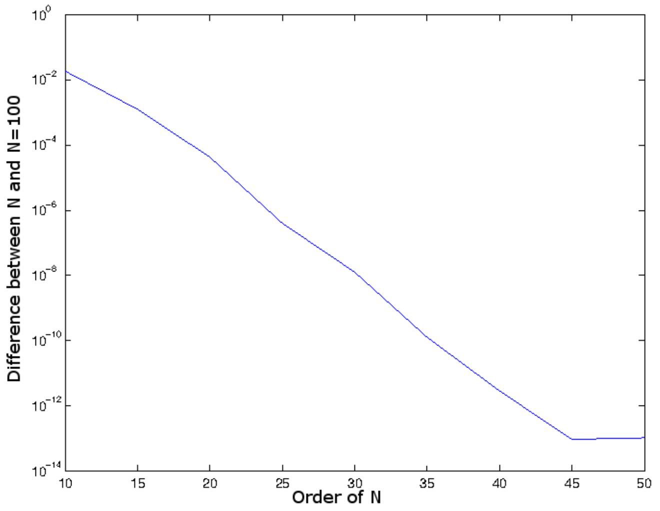

Figure 7: Difference between N and N=100, the left axis is the difference and the horizontal axis is the truncation order N

7 Discussion and Conclusion

The test results as we have seen in the previous chapter are quite positive. With the new algorithm we are able to approach a solution faster than the ode45, and with realistic precision. For instance if the required precision is only three digits we see from Table 1 that the algorithm is only three times faster. However if we should need a precision of say eleven digits we see that the new algorithm can be almost ten times faster. We've seen that the new approach to the Bloch equation, explained by Balac and Chupin[1], is indeed a successful approach. And by expanding their algorithm we've created a new algorithm to handle time varying RF pulses.

There is room for even more expansion. For instance we could imagine adding a time dependent gradient in the same way we've added the time dependent RF-pulse, which we could integrate in the terms of A_0, A_2, and so on.

The bottleneck for our algorithm remains the same as in the original algorithm. The time it takes to calculate the eigenvalues of the matrix $\mathcal{A}^{[N]}$. Further examination and optimisation of this part of the algorithm could mean a vast step forward in gaining an even faster version of the algorithm. However the main skeleton we've created in this thesis would remain the same.

8 Appendix

8.1 Appendix A

```
function sol = solver(x,T_rf,T_1,T_2,omega_0,gamma,G,B_y,B_x,N
                       ,offsetcoeff,offsetparam)
format long

tau_1 = (T_1)^(-1);
tau_2 = (T_2)^(-1);

eta_0 = 1;
B_neat = zeros(1,3*(2*N+1));
B_neat(3*N+1:3*N+3) = [0 0 tau_1*eta_0];


M_0 = zeros(1,3*(2*N+1));
M_0(3*N+1)     = 0;
M_0(3*N+2)     = 0;
M_0(3*N+3)     = 1;

omega_a_0 = B_y(1);
omega_a_2 = B_y(2);
omega_a_4 = B_y(4);
omega_a_6 = B_y(6);
omega_a_8 = B_y(8);

omega_b_0 = B_x(1);
omega_b_2 = B_x(2);
omega_b_4 = B_x(4);
omega_b_6 = B_x(6);
omega_b_8 = B_x(8);

ANUL_INIT = [-tau_2          gamma*G*x         gamma*-omega_a_0;
             -gamma*G*x       -tau_2            gamma*omega_b_0;
             gamma*omega_a_0 gamma*-omega_b_0  -tau_1];
A_TWO     = [0               0                 gamma*-omega_a_2;
             0               0                 gamma*omega_b_2;
             gamma*omega_a_2 gamma*-omega_b_2  0];
A_FOUR    = [0               0                 gamma*-omega_a_4;
             0               0                 gamma*omega_b_4;
             gamma*omega_a_4 gamma*-omega_b_4  0];
A_SIX     = [0               0                 gamma*-omega_a_6;
             0               0                 gamma*omega_b_6;
             gamma*omega_a_6 gamma*-omega_b_6  0];
A_EIGHT   = [0               0                 gamma*-omega_a_8;
             0               0                 gamma*omega_b_8;
             gamma*omega_a_8 gamma*-omega_b_8  0];

E = zeros(3*(2*N+1),3*(2*N+1));

for n = 0: 2*N
    E(n*3+1:n*3+3,n*3+1:n*3+3) =  ANUL_INIT+2*1i*(N-n)*omega_0*eye(3,3);
    if n < 2*N
        E(n*3+1:n*3+3,(n+1)*3+1:(n+1)*3+3) = conj(A_TWO);
        E((n+1)*3+1:(n+1)*3+3,n*3+1:n*3+3) = A_TWO;
    end
    if n < 2*N-1
        E(n*3+1:n*3+3,(n+2)*3+1:(n+2)*3+3) = conj(A_FOUR);
        E((n+2)*3+1:(n+2)*3+3,n*3+1:n*3+3) = A_FOUR;
    end
    if n < 2*N-2
        E(n*3+1:n*3+3,(n+3)*3+1:(n+3)*3+3) = conj(A_SIX);
        E((n+3)*3+1:(n+3)*3+3,n*3+1:n*3+3) = A_SIX;
    end
    if n < 2*N-3
        E(n*3+1:n*3+3,(n+4)*3+1:(n+4)*3+3) = conj(A_EIGHT);
        E((n+4)*3+1:(n+4)*3+3,n*3+1:n*3+3) = A_EIGHT;
    end
end

[P,D] = eig(E);
s = zeros(1,3*(2*N+1));

for n = 1:3*(2*N+1)
    if D(n,n) == 0
        s(n) = T_rf;

    else
        s(n) = (exp(T_rf*D(n,n)) - 1)/D(n,n);
    end
end

S = diag(s);
Dexp = diag(exp(T_rf*diag(D)));
M_NEAT = (P*Dexp*(P\M_0')) + (P*S*(P\B_neat'));
m_small = zeros(3,1);

for k = -N : N
 m_small = m_small + M_NEAT(3*(N+k)+1:3*(N+k)+3)*exp(2*1i*k*omega_0*
 T_rf + 2*1i*k*offsetcoeff*pi/offsetparam);
end
sol = real(m_small)';
```

8.2 Appendix B

```
clear all
profile on

B_1 = 10^-5;
G = 3;
x = 0;
offsetcoeff = -1; offsetparam = 4; %offset in omega_0
                            %offsetcoeff*pi/offsetparam

T_1 = 0.750;
T_2 = 0.050;

gamma = 42.58*10^6;
omega_0 = 10^-5*gamma;
omega_1 = gamma*B_1;
```

```
N = 15;
B_1 = 10^-5;
B_0 = 1;

B_y(1) = 0; B_x(1) = 1;
B_y(2) = 0; B_x(2) = 1/2;
B_y(4) = 0; B_x(4) = 1/2;
B_y(6) = 0; B_x(6) = 1;
B_y(8) = 0; B_x(8) = 1;
B_x = B_x*B_1; %scaling to B_1

T_rf = (pi)/(2*gamma*B_1);
eind = T_rf;
options = odeset('RelTol',1e-5,'AbsTol',[1e-13 1e-13 1e-13]);
[T,Y] = ode45(@rigid,
              [0 eind],
              [0 0 1],
              options,
              T_1,T_2,
              omega_0,
              gamma,G,
              B_y,B_x,
              x,offsetcoeff,
              offsetparam);
Tlong = 0: eind/100: 1*eind;
Z = zeros(numel(Tlong),3);

for g = 1:101 %numel(Tlong)
    Z(g,:) = solver(x,Tlong(g),
                    T_1,T_2,
                    omega_0,gamma,
                    G,B_y,B_x,
                    N,offsetcoeff,
                    offsetparam);
end

figure(3)
plot3(Y(:,1),Y(:,2),Y(:,3),'r')
hold on
plot3(Z(:,1),Z(:,2),Z(:,3),'b')
grid on
axis([-1 1 -1 1 -1 1])

t = 0:eind/1000:eind ;
w = omega_0*t + offsetcoeff*pi/offsetparam;
rfwave = B_x(1,1) + B_x(1,2)*1i
            + 2*B_x(2)*cos(2*w)
            + 2*B_x(4)*cos(4*w)
            + 2*B_x(6)*cos(6*w)
            + 2*B_x(8)*cos(8*w);



profile off
p = profile('info');
time(1) = p.FunctionTable(12,1).TotalTime
```

```
            / p.FunctionTable(12,1).NumCalls;
time(2) = p.FunctionTable(4,1).TotalTime;

str = sprintf('Algortihm with N = %i',N);

%reference = [0   0.910663572044444  −0.322602599458463];
%created with N=100
%difference = max(abs(Z(101,:)−reference));

figure(1)
subplot(2,3,1)
hold on
plot(T,Y(:,1),'r');xlabel('time');ylabel('Mx');title('ode45');
    axis([T(1) eind −1.1 1.1]);
plot(0:eind/100:eind,Z(:,1),'o');xlabel('time');ylabel('Mx');
    title(str);axis([0 eind −1.1 1.1]);
hold off
subplot(2,3,2)
hold on
plot(T,Y(:,2),'r');xlabel('time');ylabel('My');
    axis([T(1) eind −1.1 1.1]);
plot(0:eind/100:eind,Z(:,2),'o');xlabel('time');ylabel('My');
    axis([0 eind −1.1 1.1]);
hold off
subplot(2,3,3)
hold on
plot(T,Y(:,3),'r');xlabel('time');ylabel('Mz');
    axis([T(1) eind −1.1 1.1]);
plot(0:eind/100:eind,Z(:,3),'o');xlabel('time');ylabel('Mz');
    axis([0 eind −1.1 1.1]);
hold off
figure(2)
plot(t,rfwave);axis([0 eind −2*B_1 max(abs(rfwave))]);ylabel('RF Pulse');
```

8.3 Appendix C

```
function dy = rigid(t,y,B_0,B_1,T_1,T_2,omega_0,gamma,G,B_y,B_x,x)
eta_0 = 1;
tau_1 = (T_1)^(−1);
tau_2 = (T_2)^(−1);
dy = zeros(3,1);

w = omega_0*t;

omega_tilda13 = −gamma*(B_y(1) +
    2*B_y(2)*cos(2*w) + 2*B_y(4)*cos(4*w));

omega_tilda23 = gamma*(B_x(1) +
    2*B_x(2)*cos(2*w) + 2*B_x(4)*cos(4*w));

omega_tilda31 = −omega_tilda13;

omega_tilda32 = −omega_tilda23;
```

```
dy(1) = -tau_2*y(1) + gamma*x*G*y(2)+ omega_tilda13*y(3);
dy(2) = -gamma*x*G*y(1) -tau_2*y(2) + omega_tilda23*y(3);
dy(3) = omega_tilda31*y(1) + omega_tilda32*y(2) - tau_1*y(3)
    + B_0*tau_1*eta_0;
```

8.4 Appendix D

```
clear all
%load z.mat %put reference solution here

totaltime = 0;
N = 15;
B_1 = 10^-5;
B_0 = 1;

B_y(1) = 0; B_x(1) = 1;
B_y(2) = 0; B_x(2) = 1/2;
B_y(4) = 0; B_x(4) = 1/2;
B_y(6) = 0; B_x(6) = 1;
B_y(8) = 0; B_x(8) = 1;
B_x = B_x*B_1; %scaling to B_1

offsetcoeff = -1; offsetparam = 4; %offset in omega_0
                          %offsetcoeff*pi/offsetparam
T_1 = 1.750; %in seconds
T_2 = 0.050; %in seconds
gamma = 42.58*10^6;

eind1 = (pi)/(2*gamma*B_1);
dt = eind1/512;
t = 0:dt:eind1;
omega_0 = B_1*gamma; w = omega_0*t + offsetcoeff*pi/offsetparam;

RF1 = B_x(1,1) + B_x(1,2)*1i
        + 2*B_x(2)*cos(2*w)
        + 2*B_x(4)*cos(4*w)
        + 2*B_x(6)*cos(6*w)
        + 2*B_x(8)*cos(8*w)

RF2 = sinc((-256:256)/128)/2*B_1*10;
                %REFERENCE SINC FUNCTION PLOTTED IN BLUE

G = 3*B_1*10; % gradient value scaled to the force of B_1

FOV = 10; % spatial domain range
x = -FOV/2:FOV/100:FOV/2;

options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-4]);

for i = 1:numel(x)
    Q = odesolve(x(i),T_1,T_2,
                omega_0,gamma,
                eind1,B_y,B_x,
                G,N,offsetcoeff,
                offsetparam);
    ode(i,:) = Q(1,:);
    zn(i,:)  = Q(2,:);
    totaltime = totaltime + Q(3,1);
end

odeplot = abs(ode(:,1)+1i*ode(:,2));
algplot = abs(zn(:,1)+1i*zn(:,2));
alg80plot = abs(z(:,1)+1i*z(:,2));


%diff(n) = max(max(abs(z(:,:)-zn(:,:))));
diff = max(max(abs(zn(:,:)-z(:,:))));
        %where z is the reference solution
T = totaltime;



figure(1);
%subplot(2,2,1)
%plot(x,odeplot,'r');xlabel('X [cm]');ylabel('|Mxy|');
    %title('The ode45 algorithm');
    %axis([-FOV/2 FOV/2 0 max(odeplot)]);
%subplot(2,2,2)
plot(x,algplot,'b');xlabel('X [cm]');ylabel('|Mxy|');
    title('The new algorithm');
    axis([-FOV/2 FOV/2 0 max(algplot)]);
%figure(2);%subplot(2,2,3:4)
%hold on
%plot(t,RF1','r');ylabel('RF Pulse');
    %axis([0 4*10^-3 -10^-5 10^-5]);xlabel('time')
%plot(t,RF2');ylabel('RF Pulse');%axis('auto');xlabel('time')



%end

%iterations = 10:5:q*5 + 5;    %used for the last figure, axis
%figure(1)                     %axis were set manually
%hold on;
%plotyy(iterations,diff,iterations,T)
%semilogy(n,diff,n,T)
```

References

[1] Fast approximate solution of Bloch equation for simulation of RF artifacts in Magnetic Resonance Imaging
Stephane Balac, Laurent Chupin,
Institut Camille Jordan (UMR CNRS 5208), INSA de Lyon, 69621 Villeurbanne, France
Mathematical and Computer Modelling 48 (2008) 1901–1913,
http://www.sciencedirect.com/science/article/pii/S0895717708000708

[2] Spin TomogrAphy in Time domain: the MR-STAT projectSpin Tomography in Time Domain: the MR-STAT project,
by Alessandro Sbrizzi, Annette van der Toorn, Hans Hoogduin, Peter R

Luijten, and Cornelis A van den Berg.
In Proceedings of the 23rd ISMRM, Toronto, 2015. Pag. 3712

[3] Magnetic resonance fingerprinting
Dan Ma, Vikas Gulani, Nicole Seiberlich,
Kecheng Liu, Jeffrey L. Sunshine,
Jeffrey L. Duerk and Mark A. Griswold

[4] Rowland, Todd and Weisstein, Eric W.
"Matrix Exponential."
From MathWorld–A Wolfram Web Resource
http://mathworld.wolfram.com/MatrixExponential.html