Magnetic resonance pulse sequence programming using the frameworks MRTwin and pulseq

Produced by: M. Zaiss '19

moritz.zaiss@uk-erlangen.de

If you come across any problems, contact me at
moritz.zaiss@uk-erlangen.de

Table of Contents

Foreword

This practical course is a hands-on course. Thus, the theory is more in the background, instead, you have the opportunity to make your experiences with the spins and the scanner yourself. When a lecture is like being told a story of a beautiful landscape, this practical course is a guide tour to discover this landscape on your own. The tools used here are research tools. Once you reached the end of the described path, you can boldly go where no one has gone before.

About This File

MRTwin is a framework created by Alexander Loktyushin and Moritz Zaiss. Pulseq is an open file format created by Maxim Zaitsev (https://github.com/pulseq/pulseq). The Latex file was created using Overleaf, based on the free template of Armin Dubert (armindubert2019@gmail.com).

1 SECTION

A brief review of MR physics.

To be able to code MR sequences, knowledge in MR physics is required. We recommend to at least have attended the *MR physics I* course at FAU or have obtained similar knowledge. Still, as a brief reminder of MR physics needed for sequence coding, we want to recapitulate some basics.

1.1 Links to useful websites and apps

Used in this script and in the course

- http://mriquestions.com/
- www.drcmr.dk/BlochSimulator
- **Fun and useful app:** https://play.google.com/store/apps/developer?id=Lars+G.+Hanson

General MR and Bloch sites

- http://www.mritoolbox.com/ParameterDatabase.html
- https://www.imaios.com/en/e-Courses/e-MRI/
- http://mrsrl.stanford.edu/ brian/bloch/

1.2 MR signal generation

1.2.1 Dynamics of magnetization

The fundamental equation describing the dynamic of a magnetization vector, and with that describing the reaction to all external fields we can alter on purpose, is given by the Bloch equation without relaxation

$$\dot{\vec{M}} = \gamma \cdot \vec{B} \times \vec{M} \tag{1}$$

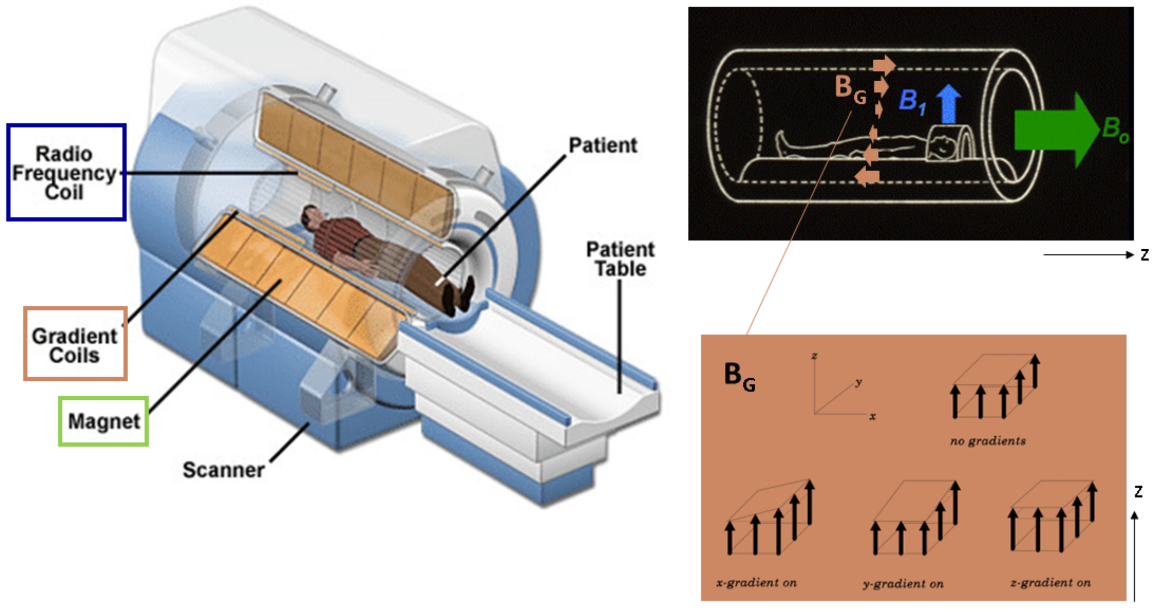

Figure 1: Typical MRI scanner with a super-conduction coil for generating the static B_0 field, a radio-frequency coil to generate a circular polarized B_1 field in the transverse plane (xy), and several gradient coils, to generate a linear alteration of the z-component of the magnetic field depending on the x,y, and z-position. Images from https://nationalmaglab.org/education/magnet-academy/learn-the-basics/stories/mri-a-guided-tour, www.mri-q.com, and https://www.researchgate.net/figure/a-MRI-Scanner-Cutaway-b-MRI-Scanner-Gradient-Magnets-MRI-A-Guided-Tour-2015_fig2_299512554

We can thus change the magnetization vector $\vec{M}$ by changing the external magnetic fields $\vec{B}$. The current hardware of clinical MR scanners is depicted in Figure 1. Thus, $\vec{B}$ consists of three different parts, (i) a static magnetic field B_0 that is oriented in the z-direction. (ii) A dynamic magnetic linear field gradient in all three directions leading to a position dependent field $\vec{B_G}(\vec{r},t) = (0,0,\vec{G}(t)\cdot\vec{r})$. (The (gradient) is in all directions, but the magnetic fields is always in z-direction). And (iii) the dynamic magnetic field $\vec{B_1}(t)$ of an electro-magnetic radio frequency irradiation. $\vec{B_1}(t)$ is perpendicular to $\vec{B}_0$ thus it has only a x- and y- component. At a given point in space and time $(\vec{x},t)$ the magnetic field is given by

$$B(\vec{x},t) = \vec{B}_0 + B_G(\vec{x},t) + \vec{B}_1 = \begin{pmatrix} 0 & + & B_1\cdot\cos(\omega_{rf}\cdot t+\phi) \\ 0 & + & B_1\cdot\sin(\omega_{rf}\cdot t+\phi) \\ B_0 & + & \vec{G}\cdot\vec{r} \end{pmatrix} \tag{2}$$

Lets first solve the simplest equation where $\vec{G}$ and B_1 are zero. Then equation (1) simplifies to

$$\dot{\vec{M}} = \gamma\cdot\vec{B}_0\times\vec{M} \tag{3}$$

To solve this equation, create a second derivative in time

$$\ddot{\vec{M}} = \gamma\vec{B}_0\times\dot{\vec{M}} = \gamma\vec{B}_0\times\gamma\vec{B}_0\times\vec{M} \tag{4}$$

Please convince yourself that the R.H.S. simplifies to

$$\gamma^2 \vec{B}_0 \times \vec{B}_0 \times \vec{M} = \gamma^2 \begin{pmatrix} 0 \\ 0 \\ B_0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ B_0 \end{pmatrix} \times \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} =$$

$$\gamma^2 \begin{pmatrix} 0 \\ 0 \\ B_0 \end{pmatrix} \times \begin{pmatrix} -B_0 \cdot M_y \\ B_0 \cdot M_y \\ 0 \end{pmatrix} = \begin{pmatrix} -\gamma^2 B_0^2 \cdot M_x \\ -\gamma^2 B_0^2 \cdot M_y \\ 0 \end{pmatrix},$$

with that equation (1) simplifies to

$$\ddot{\vec{M}} = \begin{pmatrix} -\gamma^2 B_0^2 \cdot M_x \\ -\gamma^2 B_0^2 \cdot M_y \\ 0 \end{pmatrix} \Leftrightarrow \begin{pmatrix} \ddot{M}_x = -\gamma^2 B_0^2 \cdot M_x \\ \ddot{M}_y = -\gamma^2 B_0^2 \cdot M_y \\ \ddot{M}_z = 0 \end{pmatrix}. \tag{5}$$

This forms exactly the differential equation of the $\sin(\omega_0 \cdot t)$ and $\cos(\omega_0 \cdot t)$ functions, with the important Larmor frequency $\omega_0 = \gamma B_0$. The solution of this equation is given by

$$\begin{pmatrix} M_x(t) = \cos(\omega_0 t + \phi) M_i \\ M_y(t) = \sin(\omega_0 t + \phi) M_i \\ M_z(t) = const. \end{pmatrix} \tag{6}$$

Please convince yourself that there must be a 90 degree phase shift between x and y component, this originates from equation (1), which must also hold for the sin, cos solution approach.
Equation (6) is just a constant rotation around the z-axis, that's the Larmor precession in the static B_0 field, and this precession is very fast as γB_0 is in the order of MHz for protons.

1.2.2 Transformation to rotating frame of reference

This is the conclusion of the previous section: in a constant magnetic field B_0 along z axis the transverse magnetization M_{xy} rotates around this axis in clockwise direction with angular frequency ω_0. If the observer were rotating around the same axis in clockwise direction with angular frequency Ω, M_{xy} it would appear to him rotating with angular frequency ω_0 -Ω. Specifically, if the observer were rotating around the same axis in clockwise direction with angular frequency ω_0, the transverse magnetization M_{xy} would appear to him stationary.
This can be expressed mathematically in the following way:

- Let (x, y, z) the Cartesian coordinate system of the "laboratory" (or "stationary") frame of reference, and

- (x', y', z') =(x', y', z) be a Cartesian coordinate system that is rotating around the z axis of the laboratory frame of reference with angular frequency Ω. This is called the "rotating frame of reference". Physical variables in this frame of reference will be denoted by a prime.

Coming back to equation (2) this leads now to an important simplification when transforming to the rotating frame. First of all the effect of B_0 can be removed, as it is already considered. Secondly, we can now bring the radiofrequency field B1 in resonance with the Larmor precession, namley $\omega_{rf} = \omega_0$, which means that also the rf field is static in this rotating frame. This simplifies the effective magnetic fields in the rotating frame to

$$\vec{B}(\vec{r}, t) = \vec{B_G}(\vec{x}', t) + \vec{B_1} = \begin{pmatrix} B_1 \cdot \cos(\phi_{rf}) \\ B_1 \cdot \sin(\phi_{rf}) \\ \vec{G}(t) \cdot \vec{r} \end{pmatrix} \tag{7}$$

.

Including also transverse relaxation (which is only decay of transverse magnetization) and longitudinal relaxation (which includes a recovery to the thermal magnetization M0) we obtain the full Bloch equation system:

$$\frac{d}{dt}\begin{pmatrix} M'_x \\ M'_y \\ M'_z \end{pmatrix} = \begin{pmatrix} -\frac{1}{T_2} & \Delta\omega & 0 \\ -\Delta\omega & -\frac{1}{T_2} & \omega_1 \\ 0 & -\omega_1 & -\frac{1}{T_1} \end{pmatrix}\begin{pmatrix} M'_x \\ M'_y \\ M'_z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{M_0}{T_1} \end{pmatrix} \tag{8}$$

,where $\Delta\omega = \omega_0 - \omega_{rf} + \vec{G}(t) \cdot \vec{r}$, and $\omega_1 = \gamma B_1$. Thus by controlling the external fields $\vec{G}(t)$ and $\vec{B_1}(t)$ we can, according to equation (1), control the dynamics of the magnetization in each point $(\vec{x}, t)$. In the rotating frame, the driven dynamics are always rotations, the free dynamics are exponential decays.
Convince yourself by manipulating a magnetization vector on www.drcmr.dk/BlochSimulator
For example, for B_1 only, you can convince yourself, following the B_0 case, that this yields a precession of $\vec{M}$ around the vector $\vec{B_1}$ with the frequency $\omega_1 = \gamma B_1$ for the time the rf pulse is switched on t_p. For this reason it is simpler to use the angle of this rotation, for constant B_1 this would be $\alpha = \omega_1 \cdot t_p$. For varying pulses $\alpha = \int \omega_1(t) dt$. An RF event can therefore be defined by the tuple

- **rf event**: (rf flip angle, rf phase)= (α, ϕ_{rf})

The RF field is generated by an radiofrequency transmit coil and only has x and y components. Thus, it has two degrees of freedom: the amplitude ω_1,, defining the rotation angle α and the phase ϕ_{rf} defining the angle between x' and y' component and the rotation axis. (It actually has a third component, the off-resonance frequency $\Delta\omega$, which is considered to be 0 here.)
For only gradients, you can convince yourself, that depending on the spin position within the gradient field, also just a rotation with the frequency $\omega_G(\vec{r}) = \gamma\vec{G}(t) \cdot \vec{r}$ around the z-axis occurs. The actual rotation angle depends here on the position. Still the gradient moment $\vec{g} = \int \vec{G(t)} \cdot \vec{r} dt$ of a finite gradient event defines this rotation by $\gamma\vec{g} \cdot \vec{r}$. Thus a gradient moment event has three degrees of freedom:

- **gradient moment event**: (g_x, g_y, g_z)

The linear gradients, can be altered by changing the current in the gradient coils. Gradients are especially important during encoded signal acquisition, as they link position and frequency.
We normally have no control of the B_0 field, as it is the static magnetic field generated by the superconducting coil. Still we sometimes can make use of B_0 and the Larmor preseccion by certain event time delays to evolve by $\omega_0 \cdot t$. In addition, the event time has influence on the relaxation of the magnetization and can by that affect the actual image contrast. You will see soon that correct timings are extremely important for MRI sequence to work:

- **event time**:t

So now we have a way to alter the magnetization. The reason we want to do this, is because, as soon as we have magnetization in the transverse plane, the Larmor precession of this magnetization will induce a voltage in our receive coils. To acquire this signal we need a last event type. This actually just a flag if we want to acquire signal that is induced in the receive coils, ie. if the analogue-to-digital converter (ADC) is switched on or off. As the acquired signal is is also a two component signal or complex signal, an ADC event also requires a phase angle, that is typically matched to the excitation angle ϕ_{rf}

- **ADC event:** (boolean ADC on/off, ADC rotation angle)

In summary we have 3 different events we can play out to interact with the magnetization, and one event to enable and adjust the signal acquisition

1. **radio frequency events**: flip angle and phase angle of rf pulse
2. **gradient moment event**: gradient moment in x,y,z
3. **time event**: delay time for dephasing, rephssing and relaxation
4. **ADC event**: flag if the ADC is acquiring signal, phase of ADC

Exactly these events will be displayed in the MR sequence definitions or sequence schemes in section 1.4.

1.3 MR induction and signal reconstruction

The origin of the signal is a voltage that is induced in the receive coils, because of a change in the magnetic flux

$$U_{ind} = -\frac{d}{dt}\Phi = -\dot{\Phi} \tag{9}$$

The magnetic flux Φ is changing in time due to the magnetic moments oscillating, this our magnetization vector. As shown in more detail in the Appendix 5.1, this voltage can be measured and originates dominantly from the transverse magnetization:

$$signal \sim \frac{d}{dt}\int d^3r[M_x(\vec{r},t) + M_y(\vec{r},t)] \tag{10}$$

We can now use equation (1) to calculate the actual signal. With this, we already have understood where the signal comes from, however, this signal is homogeneous over the field-of-view; we have no spatial encoding yet. The basic recipe for a spatial encoding is that we have transverse relaxation $M_{xy}(\vec{r})$, and an ADC event running, *and in addition a linear gradient.* Then from all excited spin over the whole field-of-view we will receive a summed up complex signal s(t) in the receive coils

$$s(t) = \int_{\vec{r}} M_{xy}(\vec{r}) \cdot \exp\left(-i\gamma \cdot \vec{r} \cdot \int \vec{G}(t)dt\right) d^3r \tag{11}$$

There is not really a complex voltage measured, this is just a mathematical trick. Of course each coils receives their own real signal:

$$s_x(t) = \int_{\vec{r}} M_x(\vec{r}) \cdot \cos(\gamma \cdot \vec{r} \cdot \vec{g}) d^3r s_y(t) = \int_{\vec{r}} M_y(\vec{r}) \cdot \sin(\gamma \cdot \vec{r} \cdot \vec{g}) d^3r \tag{12}$$

But because these are always 90 degree phase shifted they fulfill the relation of a combined complex number $M_{xy} = M_x + i \cdot M_y$. Thus, we do not measure the spin density, we actually measure the sum of the transverse magnetization in each voxel. If using the exp depiction of this complex number, we will measure a magnitude and a phase of this magnetization state.
So back to eq. (11), if we now use a new variable $\vec{k}(t) = \vec{g}(t)\frac{\gamma}{2\pi}$we obtain something that looks exactly like a Fourier transformation from the real space to the spatial frequency space, or k-space.

$$s(t) = s(\vec{k}(t)) = \int_{\vec{r}} M_{xy}(\vec{r}) \cdot \exp(-i2\pi\vec{r}\cdot\vec{k})d^3r \tag{13}$$

Thus, the signal for a given gradient moment g, or k, is actually just exactly the Fourier transform of the excited transverse magnetization. As the Fourier transform is invert-able, the inverse Fourier transform of the signal acquired at a full k-space grid will generate an image of the transverse magnetization.

$$M_{xy}(\vec{r}) = \int_{\vec{k}} s(\vec{k}) \cdot \exp(-i2\pi\vec{r}\cdot\vec{k})d^3k \tag{14}$$

To look what happens during encoding run the www.drcmr.dk/BlochSimulator using the mode PLANE, GX=2, GY=3, 90-hard; on the right you will see the sum of all M_{xy} which is the received signal of (11).
With equations (6), (13), and (14) we have everything to generate an MR image: (i) we know how to create transverse magnetization by rf events, (ii) we know how to spatially encode it using gradient events, and (iii) we know how to interpret it to generate an image.

1.4 MR sequence definition

An MRI sequence is a particular time scheme of radio frequency pulses and pulsed field gradients, resulting in a particular image appearance. It is a time table of exactly the defined building blocks, RF events, gradient events, and receive or ADC events. One typical depiction of MR sequence is the sequence scheme, as depicted for a spin echo sequence in Figure 2. This is a compact form of depiction as it shows one block that gets repeated after TR (repetition time), however not everything is exactly the same in each repetition as shown by the green block which shows various phase encoding steps for each repetition. A different type of scheme is shown in Figure 3, which shows a whole gradient-echo sequence with every TR/repetition one after the other. Here every event, including the event time, is plotted as a function of an event index. One could also plot all events as function of the event time. This depiction is typically used also by vendors, as every single event played out con be checked and debugged.

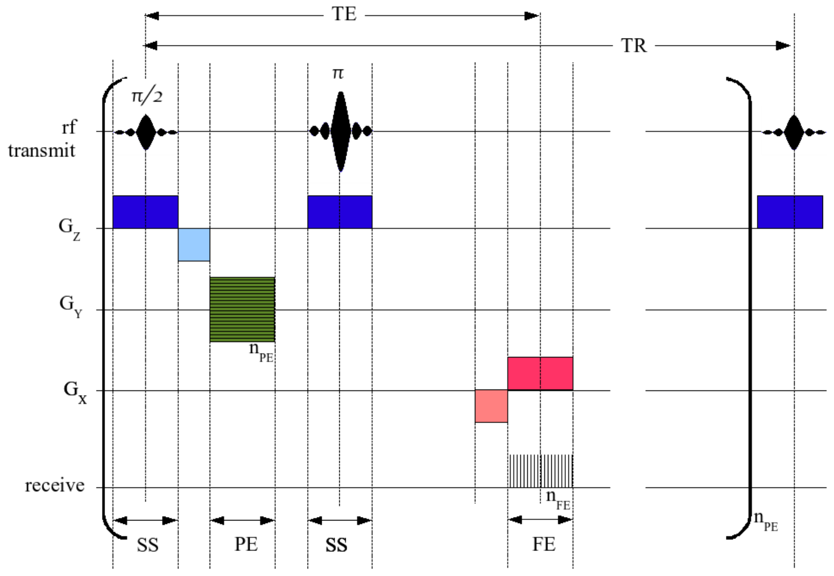

Figure 2: An MR sequence (spin echo). Taken under creative commons license from https:\\commons.wikimedia.org/wiki/File:MRI_2DFT_SE_PulseSequence.png

A different depiction of MR sequences that gives a nice overview for small resolutions is given in Figure 4. All events of each repetition are written in a matrix and the intensity/color in this picture gives eg. the gradient strength, or the RF flip angle. This view can be helpful in debugging sequences, and find asymmetries, etc. In addition to the gradient moments, also the k-space locations that are visited during the sequence are plotted. This is simply given by the cumulative sum of the gradient moments in x and y after each excitation. In the case of Figure 4, the color code of the k-space line acquisition show that this is a centric-reordered sequence, which first acquires center k-space lines and then outer k-space lines. This is also visible in the alternating y-gradient moment patterns. For every repetition, k=(0,0) after the excitation pulse, then the read-rewinder and the phase encoding gradient moments are played out and e.g. k=(-6,-6).Then the read gradient is played out and k increases from e.g. k=(-6,-6) to k(5,-6). After the readout, a

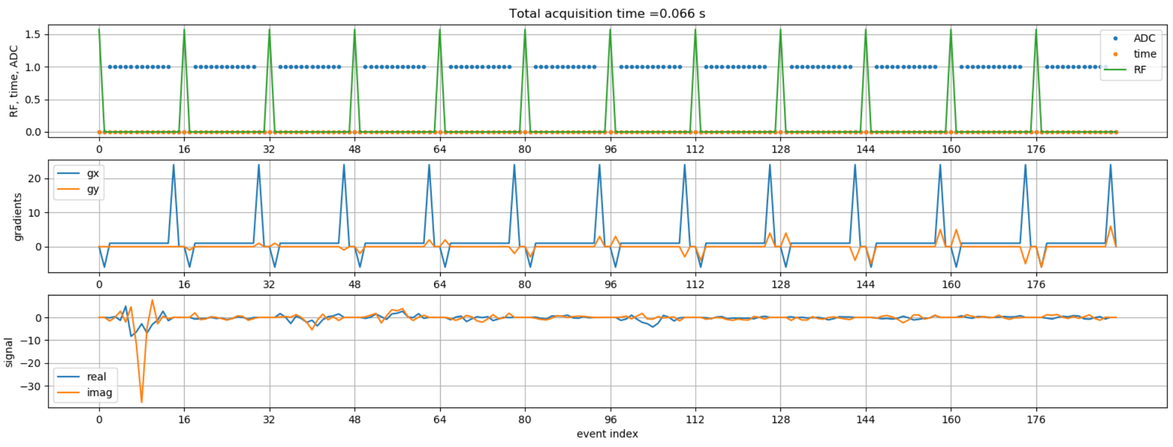

Figure 3: An MR sequence scheme (gradient echo, 12x12).

spoiler gradient is played out, while the phase encoding is reset, leading to the final state of each repetition k=(32,0).

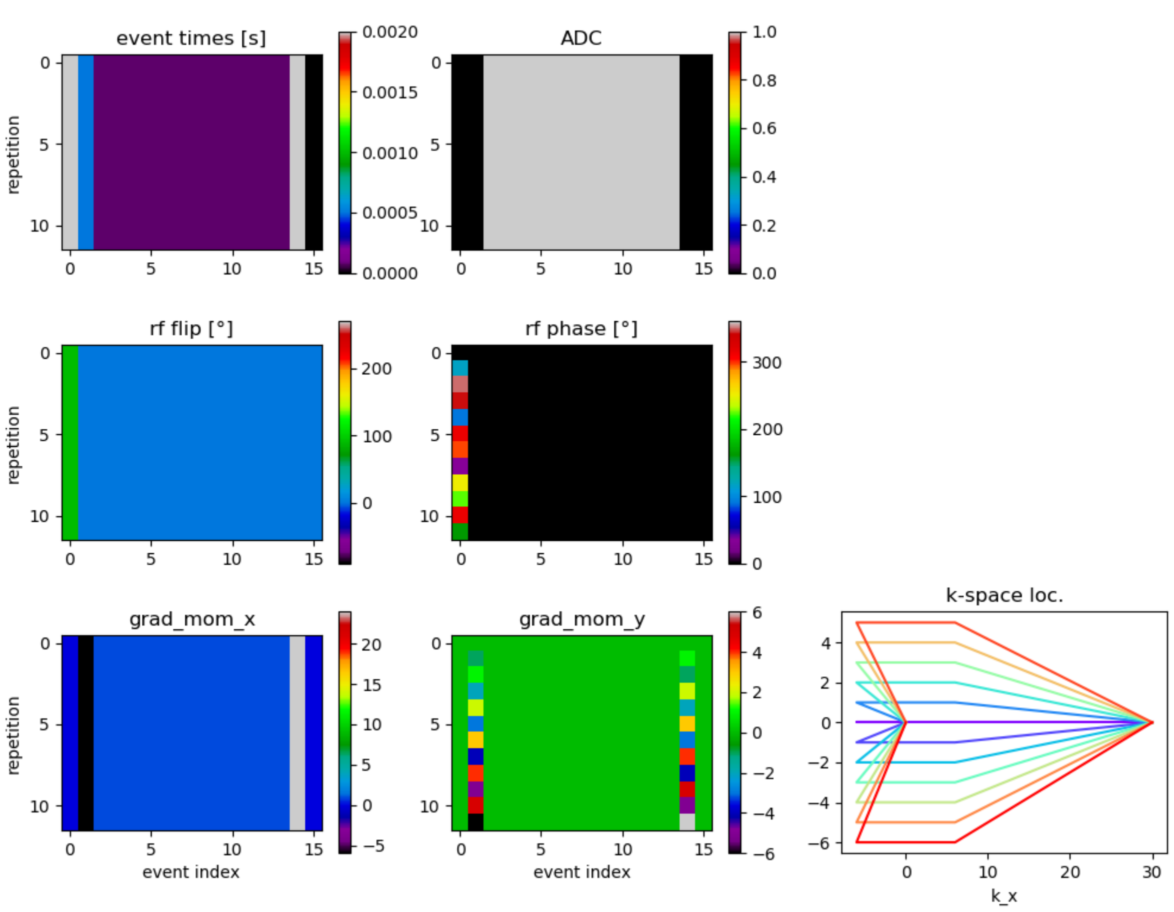

Figure 4: An MR sequence scheme as picture (gradient echo,12x12). This is the same sequence as in Figure 3. In addition to the events, also the k-space trajectory is given (bottom-right). This reflects the cumulative sum of played out gradient moments and the coverage of the acquired k-space. Full-coverage is necessary for FFT.

SECTION 2

MRTwin sequence programming and simulation framework

MRTwin is part of a larger project for automatic sequence programming. Still all sequences can also be coded manually and simulated using the included Bloch simulation. The next sections give a short overview over the coding and simulation environment MRTwin.

2.1 Downloading Python, Pytorch and Spyder

(This part is not necessary in the CIP pool at FAU, as it is already installed, skip to section 2.3). Python is free to download and is available on all types of operating systems. We recommend to install Anaconda. For Linux see https://docs.anaconda.com/anaconda/install/linux/. For windows see https://www.anaconda.com/distribution/windows . In addition to python some extension packages are required like Pytorch. install them by using the following commands

- pip install opencv-python
- pip install termcolor
- for pytorch find your correct command here: https://pytorch.org/ e.g. without gpu: conda install pytorch torchvision cpuonly -c pytorch with gpu conda install pytorch torchvision cudatoolkit=10.1 -c pytorch

Versions that were tested

- `torch.__version__`: '1.3.0'
- `np.__version__`: '1.18.1'
- `scipy.__version__`:'1.4.1'
- `matplotlib.__version__`:'3.1.1'

You can also run file exP01.py to test versions.

2.2 Usage on your own PC

Download all the code files from github https://github.com/mzaiss/MRTwin_pulseq and copy them to your project folder. Start spyder. To have plots as separate window, go to Tools->Preferences. Then on the rider *IPythonKonsole* go to *Graphics* and choose for the graphics backend: Automatic.

Then you have to close and restart spyder.

I recommend to switch the layout to matlab layout. Go to View->Layouts -> Matlab Layout or Rstudio Layout.

Once this is set up, make the project code folder *MRTwin_pulseq/code/MRtwin* your current folder and work directory in spyder. **Version control** To compare your changes to the original files or to a solution we need a difference tool. On windows to you can use Winmerge or Tortoise for Unix Systems see the next section for the FAU-CIP pool.

2.3 Usage in the FAU-CIP pool

All project files will be in the folder
/Proj/ciptmp/zaissmz
Copy the folder *MRTwin_pulseq* to your home folder.
To start spyder run the following commands in a terminal:

1. module load python
2. spyder

To have plots as separate window, go to Tools->Preferences. Then on the rider *IPythonKonsole* go to *Graphics* and choose for the graphics backend: Automatic.

Then you have to close and restart spyder.

I recommend to switch the layout to matlab layout. Go to View->Layouts -> Matlab Layout or Rstudio Layout.

Once this is set up, make the project code folder *MRTwin_pulseq/code/MRtwin* your current folder and work directory in spyder.

2.3.1 Version control

If you want compare changed files or sample solutions you can use the tools *diff*, or more visually advanced *Kompare* or *Meld*. This can be very helpful for debugging. To find out what you altered in the whole project you can use git:

- git diff –name-only (shows only the file names of files you changed)
- git diff (shows completely report what you changed in each file of the project, press q to leave)
- git difftool –tool kompare (calls kompare for each changed file one after the other)

Find more ueful commands in Readme_CIP.md (On windows machines this can be done using the tool TortoiseGit.)

2.4 A brief introduction in python and torch.

We use python in here as it is open source and easy to debug and extend. Also with pytorch python provides a rather simple possibility of parallelization of code, auto-differentiation.
If you are not familiar with python, please make sure to understand the file **exP01.py** from the MRTwin code, as it covers most of the used functions in the whole code and course.

2.5 MRTwin code

After obtaining the MRTwin package, you should see the folder structure as depicted in Figure 5. The main exercise files (exA0x ...) are in **MRTwin_pulseq/code/MRtwin** or **./code/MRTwin**, **which will always be the work directory in the following**

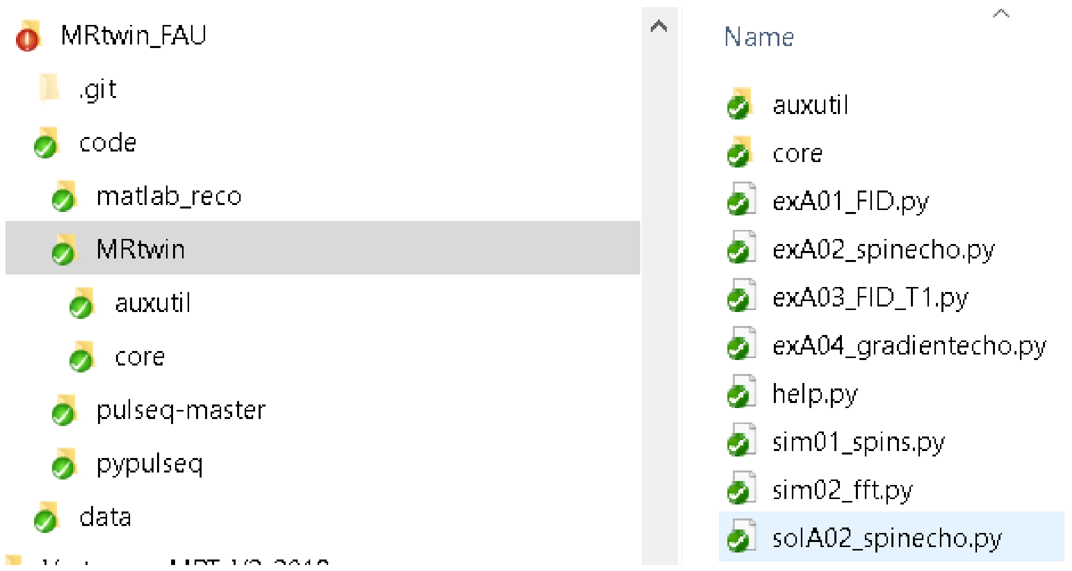

Figure 5: *MRTwin* folder structure

.

We follow some exercises of the package for this tutorial.

2.6 exA09 - a first taste of MR imaging

Before going through the code section by section, let us quickly generate a first image. Open the source file exA09_GRE_fully_relaxed.py. This is a gradient echo imaging sequence, where everything is set up except all excitation flip angles are zero. When you run this script as it is you should see the plot as in Figure 6
Then go directly to line 168. You will find the following code.

```
# RF events: flips and phases
rf_event = torch.zeros((T,NRep,2), dtype=torch.float32)
rf_event[3,16,0] = 0*np.pi/180  # 0 deg excitation
rf_event = setdevice(rf_event)
scanner.init_flip_tensor_holder()
scanner.set_flip_tensor_withB1plus(flips)
```

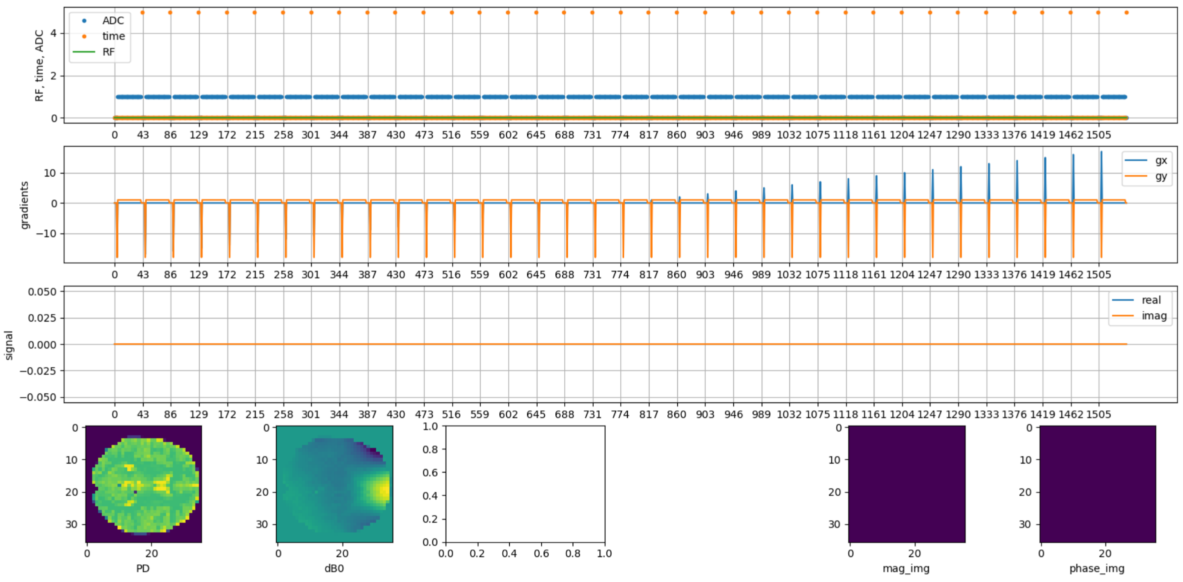

Figure 6: Output from exA09. Top row: ADC, time and RF events. Second row: gradient moments. Third row: MR signal. Bottom row left: Input for scanner -proton density (PD) and field inhomogeneity dB0. Right: MR images - magnitude image and phase image. Both are zero as no transverse magnetization was excited.

As you can see the array containing all rf flip events is initialized with zeros. The dimension of flips is given by all events (T) within one reception, and all repetitions (NRep), the last dimension of flips is for the flip angle and phase flips[T,Nrep,flip/phase]. Also the additional line flips[3,16,0], which is the flip at the third action of repetition 16 (center of k-space), is set to zero. No change here to flips[3,16,0]=90*np.pi/180 and simulate. Then add more and more repetitions

```
# RF events: flips and phases
rf_event[3,16,0] = 90*np.pi/180  # 0 deg excitation
rf_event[3,15,0] = 90*np.pi/180  # 0 deg excitation
rf_event[3,17,0] = 90*np.pi/180  # 0 deg excitation
```

and simulated. Even better, use broadcasting

```
# RF events: flips and phases
rf_event[3,10:20,0] = 0*np.pi/180  # 0 deg excitation
```

And simulate until you have excitation in every repetition Even better, use broadcasting

```
# RF events: flips and phases
rf_event[3,:,0] = 90*np.pi/180  # 0 deg excitation
```

With more and more lines, you should see the image formation as depicted in Figure 7.

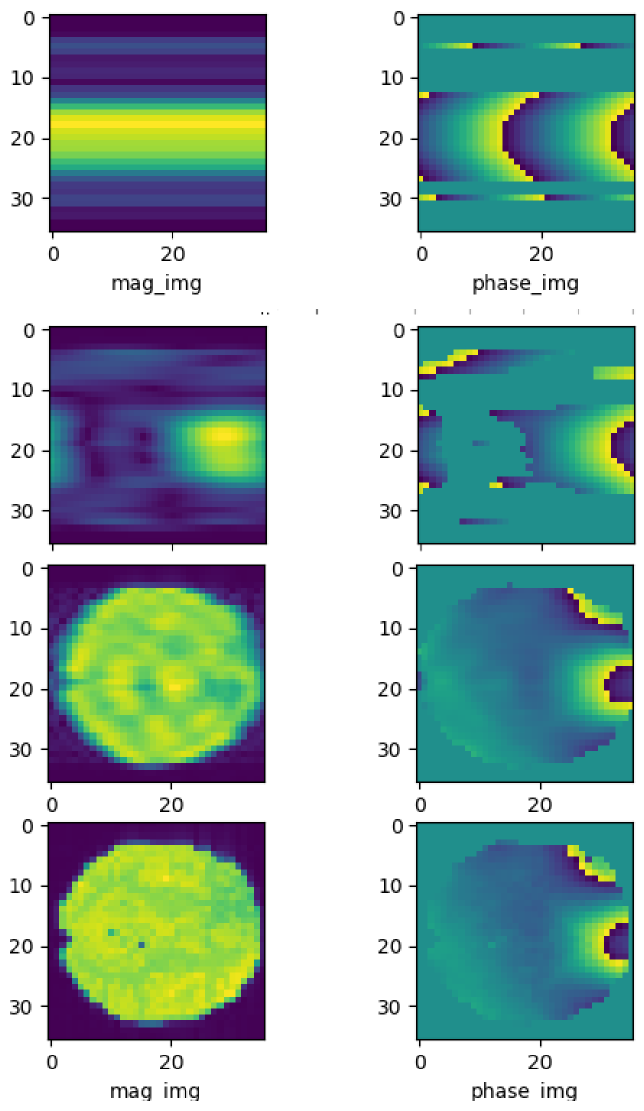

Figure 7: Output from exA09, when more and more lines are excited.

2.7 Overview over MR Twin simulation file

To get an overview of the whole framework, the file exA09_GRE_fully_relaxed.py is explained here section by section. In the very beginning the *experiment_id* is defined which should always be a copy of the current filename, to both filename and experiment id, add your last name after copying the exercise file. The *sequence_class* is specific to a certain sequence. In *experiment_description* is a course description of the current file. In *exexercise* are the tasks written that you sould perform one after the other.

```
"""
Created on Tue Jan 29 14:38:26 2019
@author: mzaiss

"""
experiment_id = 'exA09_GRE_fully_relaxed'
sequence_class = "gre_dream"
experiment_description = """
2 D imaging
"""
```

```
excercise = """
A09.1. plot the k-space as an image
A09.2. excite only certain k space lines with 90 degree, other rf_event to 0
A09.3. try different flip angles.
A09.4. try different phantoms.
A09.5. try to prolong the echo time, what do you observe?
"""
```

The next block are required python libraries that are loaded. As well as some classes of MRTwin such as the *scanner* class and the *spins* class.

```
import os, sys
import numpy as np
import scipy
import scipy.io
from  scipy import ndimage
import torch
import cv2
import matplotlib.pyplot as plt
from torch import optim
import core.spins
import core.scanner
import core.target_seq_holder
```

Then some global variables are defined concerning the real scanning, and computational performance and devices. If you want to run larger images and many spins, make sure that you have a GPU and that the flag is switched on.

```
do_scanner_query = False

double_precision = False
use_gpu = 1
gpu_dev = 0

if sys.platform != 'linux':
    use_gpu = 0
    gpu_dev = 0
print(experiment_id)
print('use_gpu = ' +str(use_gpu))
```

In the next section several useful functions are defined. The function *set_device* is necessary to call for all torch variables to make sure it runs on GPU/CPU.

```
# NRMSE error function
def e(gt,x):
    return 100*np.linalg.norm((gt-x).ravel())/np.linalg.norm(gt.ravel())

# torch to numpy
def tonumpy(x):
    return x.detach().cpu().numpy()

# get magnitude image
def magimg(x):
    return np.sqrt(np.sum(np.abs(x)**2,2))

def phaseimg(x):
    return np.angle(1j*x[:,:,1]+x[:,:,0])

def magimg_torch(x):
  return torch.sqrt(torch.sum(torch.abs(x)**2,1))
```

```
def tomag_torch(x):
    return torch.sqrt(torch.sum(torch.abs(x)**2,-1))

# device setter
def setdevice(x):
    if double_precision:
        x = x.double()
    else:
        x = x.float()
    if use_gpu:
        x = x.cuda(gpu_dev)
    return x
```

2.7.1 Section S0

Now the first MRTwin related section appears, Section S0. Here the most important variables are set, like the image size *sz*, the number of sequence events *NEvnt* within one repetition and the number of repetitions *NRep*. The total number of subsequent pulse events is given by *NEvnt·NEvnt*. The number of spins per voxel is given by *NSpin*. It must be a square of an even number for symmetric reasons. Some simulations will run with 4^2 others require much more. Tested sequence normally run well with 26^2 spins.

```
#############################################################################
## S0: define image and simulation settings::: #####################################
sz = np.array([36,36])                      # image size
extraMeas = 1                               # number of measurments/ separate scans
szread=sz[1]
NEvnt = szread + 5 + 2                               # number of events F/R/P
NRep = extraMeas*sz[1]                      # number of total repetitions
NSpins = 16**2                               # number of spin sims in each voxel
NCoils = 1                                  # number of receive coil elements
noise_std = 0*1e-3                          # additive Gaussian noise std
kill_transverse = False                     #
import time; today_datestr = time.strftime('%y%m%d')
NVox = sz[0]*szread
```

2.7.2 S1: Init spin system and phantom

In section S1 the spin system is initialized. Two different predefined virtual objects can be loaded, an 2D model solution phantom *phantom2d.mat* and a brain phantom. This will be resized then to the resolution given by *sz*.
In the second part of S1 the frequency distribution of the individual spins is set up, this is a *tan* distribution, which results in an exponential decay of the net transverse magnetization by the rate *R2star*. Sometimes it makes sens to set R2star to 0 or a high value. Please leave the rest of the code as it is.

```
#########################################################
## S1: Init spin system and phantom::: #########################
# initialize scanned object
spins = core.spins.SpinSystem(sz,NVox,NSpins,use_gpu+gpu_dev,double_precision=double_precision)

cutoff = 1e-12
#real_phantom = scipy.io.loadmat('../../data/phantom2D.mat')['phantom_2D']
real_phantom = scipy.io.loadmat('../../data/numerical_brain_cropped.mat')['cropped_brain']
```

```
real_phantom_resized = np.zeros((sz[0],sz[1],5), dtype=np.float32)
for i in range(5):
    t = cv2.resize(real_phantom[:,:,i], dsize=(sz[0],sz[1]), interpolation=cv2.INTER_CUBIC)
    if i == 0:
        t[t < 0] = 0
    elif i == 1 or i == 2:
        t[t < cutoff] = cutoff
    real_phantom_resized[:,:,i] = t

#begin nspins with R2* = 1/T2*
R2star = 30.0
omega = np.linspace(0,1,NSpins) - 0.5   # cutoff might bee needed for opt.
omega = np.expand_dims(omega[:],1).repeat(NVox, axis=1)
omega*=0.99 # cutoff large freqs
omega = R2star * np.tan ( np.pi  * omega)
spins.omega = torch.from_numpy(omega.reshape([NSpins,NVox])).float()
spins.omega = setdevice(spins.omega)
## end of S1: Init spin system and phantom ::: #####################################
```

2.7.3 S2: Init scanner system

The next section sets up the *scanner* object. Here also the B_1^+ field is initialized, however for this course its set uniformly to 1.

```
#######################################################################
## S2: Init scanner system ::: #####################################
scanner = core.scanner.Scanner_fast(sz,NVox,NSpins,NRep,NEvnt,NCoils,noise_std,use_gpu+gpu_dev,double_precision=double_pr

B1plus = torch.zeros((scanner.NCoils,1,scanner.NVox,1,1), dtype=torch.float32)
B1plus[:,0,:,0,0] = torch.from_numpy(real_phantom_resized[:,:,4].reshape([scanner.NCoils, scanner.NVox]))
B1plus[B1plus == 0] = 1    # set b1+ to one, where we dont have phantom measurements
B1plus[:] = 1
scanner.B1plus = setdevice(B1plus)
```

2.7.4 S3: MR sequence definition

S3 is the most important section for this course, as here all the sequence definitions are coded. You will find the event types as defined in the physics section: *adc_mask*, *rf_event*, *gradm_event* and the *event_time*. All of them have the dimensions [NEvnt,NRep], rf_event has in addition to the flip angle the rf phase. And gradm_events have x and y gradient in the third dimension. The adc phase is not explicitly, given but set with the function call set_ADC_rot_tensor, as this tensor is very often related to the rf phase. The signs and correction terms here are necessary to match the simulation signal to the real system.

```
############################################################################
## S3: MR sequence definition ::: #####################################
# begin sequence definition
# allow for extra events (pulses, relaxation and spoiling) in the first five and last two events (after last readout event
adc_mask = torch.from_numpy(np.ones((NEvnt,1))).float()
adc_mask[:5]  = 0
adc_mask[-2:] = 0
scanner.set_adc_mask(adc_mask=setdevice(adc_mask))

# RF events: rf_event and phases
rf_event = torch.zeros((NEvnt,NRep,2), dtype=torch.float32)
rf_event[3,16,0] = 0*np.pi/180  # 90deg excitation
```

```
rf_event = setdevice(rf_event)
scanner.init_flip_tensor_holder()
scanner.set_flip_tensor_withB1plus(rf_event)
# rotate ADC according to excitation phase
rfsign = ((rf_event[3,:,0]) < 0).float()
scanner.set_ADC_rot_tensor(-rf_event[3,:,1] + np.pi/2 + np.pi*rfsign) #GRE/FID specific

# event timing vector
event_time = torch.from_numpy(0.08*1e-3*np.ones((NEvnt,NRep))).float()
event_time[3,:] =  2e-3 + 3e-3
event_time[-1,:] =  5
event_time = setdevice(event_time)

# gradient-driver precession
# Cartesian encoding
gradm_event = torch.zeros((NEvnt,NRep,2), dtype=torch.float32)
gradm_event[4,:,1] = -0.5*szread
gradm_event[5:-2,:,1] = 1
gradm_event[4,:,0] = torch.arange(0,NRep,1)-NRep/2
gradm_event = setdevice(gradm_event)

scanner.init_gradient_tensor_holder()
scanner.set_gradient_precession_tensor(grad_event,sequence_class)  # refocusing=False for GRE/FID, adjust for higher echo
## end S3: MR sequence definition ::: #####################################
```

Please note!

As seen in section S0, the event dimension has in addition to the image dimension (given by szread) 7 additional events.

```
NEvnt = szread + 5 + 2
```

Our convention is that 5 events come before the readout adc events. And 2 events come after the readout. Moreover we reserve:

- events 0, 1, 2, : extra events for preparation pulses etc.
- event 3: excitation pulse event
- event 4: rewinder/ prewinder gradient event and delay for TE
- event 5:-2 : adc events, only grad and adc here
- event -2 : spoiler gradient event
- event -1 : delay only, no gradients are played out (also not in sim!)

We suggest to have only one type of event per event index. However, for the simulation it is possible to have all types within the same event index. Then for every index all types of events are performed in the following order:

1. 1. ADC
2. 2. RF
3. 3. gradient moment
4. 4. delay.

With this the sequence definition is finished an we can simulate the sequence.

2.7.5 S4: MR simulation forward process

Now the scanner is ready. The signal must be initialized and then the forward simulation can be called to play out the defined sequence for the defined spin system (phantom). There are two types of forward simulations *forward* and *forward_fast*.The latter one is faster as it calculates the adc events in one shot. However, if you want realistic simulation during the ADC, like echo formation etc., you need to run the slower *forward* method. For most imaging applications, *forward_fast* will do fine.

In the second part a helper object is created, namely *targetSeq*. It is a helper class that has two useful plotting functions *print_seq_pic*, which plots the sequence as picture (see Figure 4). And *print_seq*, which plots the sequence scheme as in Figure 3 together with the signal. In the Figure 4, you can again see the 5 extra events before ADC, and the 2 extra events after ADC.

```
############################################################################
## S4: MR simulation forward process ::: ####################################
scanner.init_signal()
scanner.forward_fast(spins, event_time)
#scanner.forward(spins, event_time)

targetSeq = core.target_seq_holder.TargetSequenceHolder(rf_event,event_time,gradm_event,scanner,spins,scanner.signal)
targetSeq.print_seq_pic(True,plotsize=[12,9])
targetSeq.print_seq(plotsize=[12,9])
```

The print_seq commands are very important for us to see and debug our sequences. Try also

```
targetSeq.print_seq(plotsize=[12,9],time_axis=1)
```

to plot the sequence and signal as function of event times instead of event index.
S5: MR reconstruction of signal
In the last section, the MR signal, given by *scanner.signal* is processed. Correctly ordered and shifted to form the complex k-space, the signal can be directly Fourier transformed using a FFT to image domain. The complex image is plotted as magnitude and phase image.

```
#%% ############################################################################
## S5: MR reconstruction of signal ::: #####################################

spectrum = tonumpy(scanner.signal[0,adc_mask.flatten()!=0,:,:2,0].clone())
spectrum = spectrum[:,:,0]+spectrum[:,:,1]*1j # get all ADC signals as complex numpy array

spectrum = np.roll(spectrum,szread//2,axis=0)
spectrum = np.roll(spectrum,NRep//2,axis=1)
space = np.zeros_like(spectrum)
space = np.fft.ifft2(spectrum)
space = np.roll(space,szread//2-1,axis=0)
space = np.roll(space,NRep//2-1,axis=1)
space = np.flip(space,(0,1))

plt.subplot(4,6,19)
plt.imshow(real_phantom_resized[:,:,0].transpose(), interpolation='none'); plt.xlabel('PD')
plt.subplot(4,6,20)
plt.imshow(real_phantom_resized[:,:,3].transpose(), interpolation='none'); plt.xlabel('dB0')
plt.subplot(4,6,21) # plot the kspace as an image, lowest ferquency should be in the center


plt.subplot(4,6,23)
plt.imshow(np.abs(space).transpose(), interpolation='none',aspect = sz[0]/szread); plt.xlabel('mag_img')
plt.subplot(4,6,24)
mask=(np.abs(space)>0.2*np.max(np.abs(space))).transpose()
plt.imshow(np.angle(space).transpose()*mask, interpolation='none',aspect = sz[0]/szread); plt.xlabel('phase_img')
plt.show()
```

2.7.6 S4: Case of real MR scan

The mentioned helper object, *targetSeq*, also manages to send the sequence to the real system.

```
#############################################################################
## S4: MR simulation forward process ::: ##################################
scanner.init_signal()
#scanner.forward_fast(spins, event_time)
#scanner.forward(spins, event_time)

targetSeq = core.target_seq_holder.TargetSequenceHolder(rf_event,event_time,gradm_event,scanner,spins,scanner.signal)
targetSeq.print_seq_pic(True,plotsize=[12,9])
targetSeq.print_seq(plotsize=[12,9])

# this exports the sequence as *.seq file in the folder ./out/
targetSeq.export_to_pulseq(experiment_id,today_datestr,sequence_class,plot_seq=True,single_folder=True)

# this imports the signal from the real scanner (*.seq.dat file in the folder ./out/ )
scanner.get_signal_from_real_system(experiment_id,today_datestr,single_folder=True)
```

As this will overwrite the scanner.signal array, we commented the simulation to save time. The exact same reconstruction (S5) as for the simulated scanner.signal can now be used.

3

SECTION

Application at real MRI scanner - Introduction of pulseq

Pulseq is an open source framework for the development and execution of magnetic resonance (MR) pulse sequences for imaging and spectroscopy. In summary, MRI sequence can be programmed directly in MATLAB or Python and executed on real hardware. A central contribution of the pulseq project is an open file format to compactly describe MR sequences suitable for execution on an MRI scanner. The detailed file specification can be obtained on http://pulseq.github.io/. For this practical course we use a python implementation of the *pulseq* file generator, as well as the high level notation*MRTwin* also implemented in python.

3.1 Pulseq interpreter at real scanner (IDEA Siemens)

If you received the sequence files, copy both .so and .dll files into the sequence folder on the MR scanner or in your IDEA environment (C:\Medcom\MriCustomer\seq). The .seq files have to be placed in the subfolder "seq/pulseq". To access the sequence from the MR scanner, you need the pulseq interpreter running. Navigate to Dot Cockpit -> Program Editor -> Browse -> Default -> Sequence Region -> Customer Sequences -> Default. Now insert the pulseq sequence to any protocol. Run this protocol and it will automatically read the pulseq file "external.seq" in "seq/pulseq".

3.2 Python and pulseq Files

For generation of pulseq files that are readable by an MR scanner we run python scripts (extension .py). From python sometimes data files are called to load virtual MR objects as matrices (extension .npy or.mat). When the python files run sucessfully, pulseq files are created (extension .seq). For example, an acceptable file name might be myfile.seq. The .seq files have to be placed in the subfolder (C:\Medcom\MriCustomer\seq\pulseq). The pulseq interpreter always searches for the seq file external.seq. Thus to run one specific file you have to rename it to this very name, external.seq.

3.3 Pulseq standard

Gradient and RF events are exactly the events you will find in the BLOCKS of the pulseq file format. Every block consists of a delay block (D) , an RF event block (RF), and a gradient event block for each direction (GX, GY, GZ). In addition to the gradient and RF events, an analog to digital converter event (ADC) can be played out. An ADC event will not alter the magnetization, this is just an event to steer when we actually want to acquire signals of the moving magnetization vectors in our receive coils. One of the simplest MR experiments thus would be the following, an RF event for exiting some magnetization, and a subsequent ADC event to acquire the excited signal:

```
# Format of blocks:
##   D RF   GX   GY   GZ ADC
[BLOCKS]
1   0  1    0    0    0  0
2   0  0    0    0    0  1
```

In the simplest case the RF event is a block pulse of constant amplitude, with no phase and no off-resonance. In the pulseq format the Rf and ADC events would look like this:

```
# Format of RF events:
# id amplitude magid phaseid delay freq phase
# ..         Hz     ....      ....    us    Hz    rad
[RF]
1           2500  1  2  0  0  0

# Format of ADC events:
# id num dwell delay freq phase
# ..   ..     ns     us    Hz    rad
[ADC]
1   64   50000    0  0  0
```

The ADC event has here 64 samples.
Some events can be played out simultaneously within the same block, e.g. a gradient event for encoding and the ADC event, or a RF and gradient event for spatial selective excitation. Others are forbidden, like ADC event during an RF event. This would then lead to signal distortions as the acquired signal is governed by the played out Rf field and not by the MR signal. However, at most MR systems this is just not possible to run.
In summary, an MR sequence is a sequence of gradient and RF events played out subsequently. With these events and some delays in-between we can completely control the dynamic of the magnetization vectors in each voxel and then acquire an encoded signal using an ADC event. Collecting enough ADCs with different encoding then allows for subsequent image reconstruction.

4
SECTION

Exercises

4.1 Overview

The exercises are structured by sequence types:

- A : basics and GRE
- B : spin echo and RARE
- C : stimulated echo
- D : balanced SSFP
- E : export to real system
- F : undersampling and reconstruction

4.2 FID - A1

The free-induction decay (FID) sequence consists of a 90° excitation pulse and a subsequent readout. The observed signal decay, given by

$$s(t) = s_0 \cdot e^{-R_2^* \cdot t} \tag{15}$$

, is dominated by intravoxel inhomogeneity dephasing R_2' with additional contribution of spin-spin relaxation rate R_2.

$$R_2^* = R_2' + R_2 \tag{16}$$

The signal s is a complex number and the complex phase of s_0 depends on two phase angles:

1. the RF excitation pulse phase ϕ
2. the ADC phase factor ϕ_{adc}

Depending on these phases the signal will be real or imaginary or complex. To get a real signal one should match excitation and ADC rotation angle $\phi_{adc} = \phi + const$. The constant can depend on the coil and also on the standard rf excitation phase, eg. x' or y'.
The FID as given in equation (15) shows the case in the rotating frame of reference, which is the standard case also in the simulation. By adding a strong B_0 inhomogeneity we can emulate being in the lab frame or in an off-resonant frame. Then the general form of the FID can be observed, which is a damped oscillation:

$$s(t) = s_0 \cdot e^{-i \cdot w_0 \cdot t} \cdot e^{-R_2^* \cdot t} \tag{17}$$

The oscillation frequency is given by the B_0-inhomogeneity for our emulated case, or the Larmor frequency for the lab frame of reference. This oscillation is necessary to generate the alternating magnetic field and by that induce a voltage in the receive coils (see section 5.1), thus the name free induction decay.
If the excitation flip angle is not 90°, the rotating frame equation (15) changes to

$$s(t) = s_0 \cdot \sin(\alpha) \cdot e^{-R_2^* \cdot t} \tag{18}$$

4.2.1 Simulation details

The R_2' dynamic cannot be simply simulated by adding a factor $\exp(-R_2' \cdot t)$ otherwise all information about the intra-voxel dephasing state would be lost. Thus, a sum of the signal of many spins or iso-chromats is used to mimick the intra-voxel behavior. To get the observed mono-exponential decay these isochromats have slightly different off-resonance frequencies following the distribution $R_2' = \tan(\pi \cdot X)$, where X is uniformly distrbuted between -0.5 and 0.5. In the code this is realized by

```
R2dash = 30.0
omega = np.linspace(0,1,NSpins) - 0.5    # cutoff might bee needed for opt.
omega = np.expand_dims(omega[:],1).repeat(NVox, axis=1)
omega*=0.99 # cutoff large freqs
omega = R2dash * np.tan ( np.pi  * omega)
```

In vivo this distribution is similar, yet many spins per voxel are available. The simulation requires enough iso-chromats for each voxel to reach similar convergence. However for numerical stability some low populated high-frequency states were cut-off.

4.2.2 FID with T1 recovery

After a 90° pulse the z-component of the magnetization m_z is zero. After that it recovers from this initial state m_i with the spin-lattice recovery rate R1 towards the thermal equilibrium m_0

$$m_z(t) = (m_i - m_0) \cdot e^{-R_1 \cdot t} + m_0 \tag{19}$$

If 90 ° excitations are played out repeatedly, for subsequent excitations only the z-magnetization after this recovery is available. This leads to the signal equation for the second FID:

$$s(t) = [(\cos(\alpha) - m_0) \cdot e^{-R_1 \cdot t} + m_0] \cdot \sin(\alpha) \cdot e^{-R_2^* \cdot t} \tag{20}$$

This leads to the fact that (i) this timing and the flip angle can be optimized to have the most signal acquisition per unit time (google Ernst Angle), and (ii) that the R1 relaxation rate can be measured by such an experiment.

4.3 Spin Echo

The spin echo (SE) sequence consist of a 90° excitation and a 180° refocusing pulse. The 180° pulse inverts all the dephased magnetization and thus reverses the R_2' decay until a so-called echo is formed. Figure 8 shows the runners analogy of the spin echo and an animation of this phenomenon can be found here.

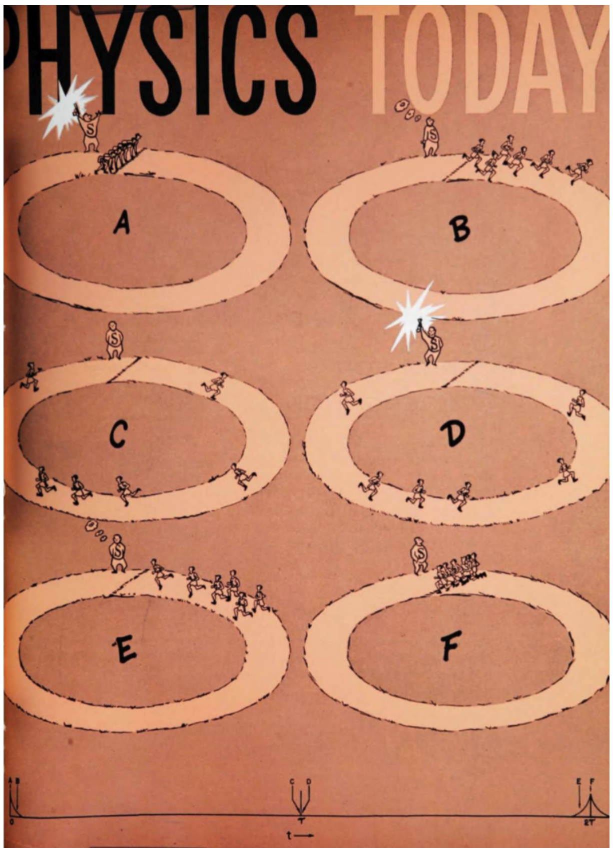

Figure 8: Erwin Hahn's famous spin echo analogy on the cover of Physics Today. The first bang reflects the 90° excitation, the second bang reflects the 180° refocusing pulse.

The observed signal is a combination of the FID behavior and the formation of an echo. The hull of this signal is given by the residual R2-decay that cannot be reversed.

$$\hat{s}(t) = s_0 \cdot e^{-R_2 \cdot t} \tag{21}$$

The faster dynamic is FID-and reversed-FID-like. The echo will form exactly when the time after the 180° pulse is the same as the time between the 90° and the 180° pulse. With the definition of the echo time TE as the time from the excitation til the echo forms. The 180° pulse must occur after $\frac{TE}{2}$.
The RF phase of both the excitation and the refocusing pulse have a direct effect on the phase of the echo. Using a 90° phase shift of the excitation pulse yields same phase of FID and echoes and also reflects the very stable CPMG implementation.

4.3.1 Simulation details

In the simulation we work on an event grid, thus for the exact timings, the duration of all previous events must be taken into account or set to zero. For the spin echo example B01 it can be calculated like this:

```
idx_echo= int(5 + (NEvnt-7)/2)
TE2_2 = event_time[3:idx_echo,0].sum()

#event_time[2,:] = TE2_2 - 0.08*1e-3*2
TE2_1 = event_time[1:3,0].sum()
event_time = setdevice(event_time)

TA = tonumpy(torch.sum(event_time))
TR = tonumpy(torch.sum(event_time[:,0]))
TE = TE2_1 + TE2_2
```

4.4 GRE

The gradient echo (GRE) sequence consists of an excitation pulse followed by a dephasing gradient and subsequent rephasing gradient during the readout. A gradient means that we have an additional linear frequency distribution within each and between different voxels. Similar to the previously described R_2' decay this results in an additional dephasing and a signal decay even faster than R_2^* during the first magnetic field gradient. By an inversed gradient this decay can be inverted an a gradient echo occurs. This time the hull of this echo is given by the FID decay, thus the size of the echo is $\propto e^{-R_2^* \cdot t}$. The gradient echo is now independent of the timing and occurs when the sum of all gradient moments is zero. TE at $\sum G \cdot t = 0$; for constant gradient:s TE at $t_{dephase} = t_{rephase}$.

4.4.1 Gradient encoding and k-space

A closer look to a two pixel gradient echo reveals that we can separate two pixels by their frequencies they send during the readout rephasing gradient. And these frequency contributions can be separated from the signal by using an FFT. This brought us to the general concept of FFT reconstruction and that in MRI we measure directly points in the k-space and the k-space coordinates are determined by the gradient experienced by the transverse magnetization (compare eqns. (13) and (14)). The gradient moment used as input in the simulation is directly linked to the k-space coordinate.

- after a 90° excitation k=0
- k=cumsum(grad_moments), this allows free movement in k-space for rewinder or phase encoding or spoiling.
- a 180° pulse inverts all rotations made: k=-k

By these operations we can play out gradients in such a way that the k-space is equidistantly and fully covered ready for a two-dimensional FFT. If gradients are not played out in linear order (centric reordering or EPI zig-zag) another reordering of the k-space must be performed before FFT. Undersampled k-spaces can also be reconstructed as discussed in the radial imaging part below.

4.5 Slice selection

The simulation is completely 2D thus we don't need slice selection here. However in a real system we need to encode also the third dimension. One way to do this is directly be the selection of the excited volume using slice selection gradients. This means that an RF pulse of since shape with a certain bandwidth is played out during a slice selection gradient, thus only the magnetization in a certain frequency window, or slice, is excited. Thus after such a pulse the problem is again two dimensional and equivalent to our simulation. For more details see here: http://mriquestions.com/slice-selective-excitation.html

4.6 GRE imaging

With the encoded signal and the FFT reconstruction we can now generate a full GRE MRI sequence. The fully relaxed gradient echo sequence consists several repetitions of the following building block:

- 90° excitation
- rewinder and phase encoding gradient
- frequency encoding gradient
- relaxation delay

This is repeated for all different phase encoding until the k-space is filled and the sequence schemes were already shown in the beginning in Figures 3 and 4 for centric reordered GRE.

4.7 FLASH - fast low angle shot

As seen above, reducing the relaxation delay will reduce the signal of the next excitation, thus the GRE sequence with 90° excitation cannot be played out too fast without loosing signal. One way to "save" z-magnetization for later excitations is using lower flip angles - instead of 90° only 5°-20°. This allows to reduce the TR further, but it also leads to interference with "old" transverse magnetization of a previous excitations. This old transverse magnetization experienced different gradients and thus is differently encoded. There is two ways to resolve this, either take care of the encoding of this magnetization, which we will do in a later exercise, or to get rid of residual transverse magnetization by so called **spoiling**.

- **Long relaxation spoiling.** When TR»T2*, the transverse magnetization will naturally decay to zero by the end of the cycle. Thus any gradient echo sequence using TR values of several hundred milliseconds or longer will be "naturally" spoiled.

- **Gradient spoiling**. In this method spoiling is performed by applying the slice-select (and sometimes readout) gradients with variable amplitudes at the end of each cycle just before the next RF pulse. The strength of the spoiler gradient is varied linearly or semi-randomly from view to view.
- **RF-spoiling.** Here the RF phase is changed according to a predefined formula from cycle to cycle. Using a completely randomized pattern of phase changes is not ideal in that unintended spin clustering may occur and the degree of spoiling may change from one interval to the next. A superior method is to increment the phase quadratically using a recursive formula.

In both theory and practice, RF-spoiling is superior to gradient spoiling because it does not generate eddy currents and is spatially invariant. Both are used to overcome the time-consuming relaxation spoiling. This list was adjusted from http://mriquestions.com/spoiling—what-and-how.html where more details can be found.
Thus to generate the fast FLASH sequence the GRE scheme is slightly changed. The FLASH sequence consists of several repetitions of the following building block:

- 5° excitation (or another low angle close to the Ernst Angle) with quadratically increasing phase
- rewinder and phase encoding gradient
- frequency encoding gradient
- spoiler gradient with minimal relaxation delay

The FLASH sequence allows for TR below 5 ms and thus fast 2D and 3D imaging of decent resolution.

4.8 GRE EPI

Echo planar imaging is just a repeated usage of the gradient echo principle for the same transverse magnetization. The GRE-EPI sequence consists of only one 90° excitation, prewinder, and then subsequent gradient echo readouts of alternating polarity and small phase encoding blips in between. Thus, excited transverse magnetization is re-used again and again to sample to full k-space. As the signal decay is fast, the whole sequence must be fast, but this is also now possible as all other RF events are not needed. EPI is one of the fastest MRI readouts with 2D imaging acquisition in the 10-100 ms range. A SE-EPI is also possible, but is a homework exercise.

4.9 Spin echo imaging

As shown above, the GRE signal decays with R_2^* . Quite a lot of signal can be retrieved by employing a refocusing pulse so that a spin echo is measured which only decays with R_2. However for spatial encoding we still need a gradient echo, a spin echo (SE) sequence combines both echoes to occur at the same time. The basic SE sequence consists of a 90° excitation pulse, followed by a gradient prewinder. It is called prewinder here as it has the same polarity as the readout gradient. The "rewinding" is not done by the subsequent 180° refocusing pulse which induces the spin echo, and at the same time brings us from k to -k. To have the SE and the GRE at the same time (or in other words the SE in the k-space center), both echo time conditions as described above must be full-filled.

- 90° excitation with 90° phase
- prewinder k/2
- 180° refocusing pulse
- phase encoding gradient
- frequency encoding gradient
- long relaxation delay

4.10 Turbo spin echo imaging

Similar as the EPI also the SE can be accelerated by reusing transverse magnetization. Outgoing from the SE sequence, this is realized by additional 180° pulses in each repetition that refocus the same spin echo again and again. This first seems simple as the 180° generate the rewinding directly. However, repeated coherent RF pulses lead to higher echoes when RF excitation is imperfect, which is almost always the case in real scanners due to B1 field inhomogeneities. This leads to partial excitation of 180° refocusing pulse and vice versa, as well as stimulated echoes. To remove them gradients are used to move these echoes further out in k-space. The turbo spin echo (TSE) looks then like this:

1. 90° excitation with 90° phase
2. prewinder k/2 + additional read gradient moment k
3. 180° refocusing pulse
4. phase encoding gradient + additional gradient moment k
5. frequency encoding gradient + ADC
6. phase encoding gradient inverted + additional read gradient moment k
7. back to 3

Steps 3-7 are repeated for all phase encoding steps, leading to the sequence scheme in Figure X and Y.

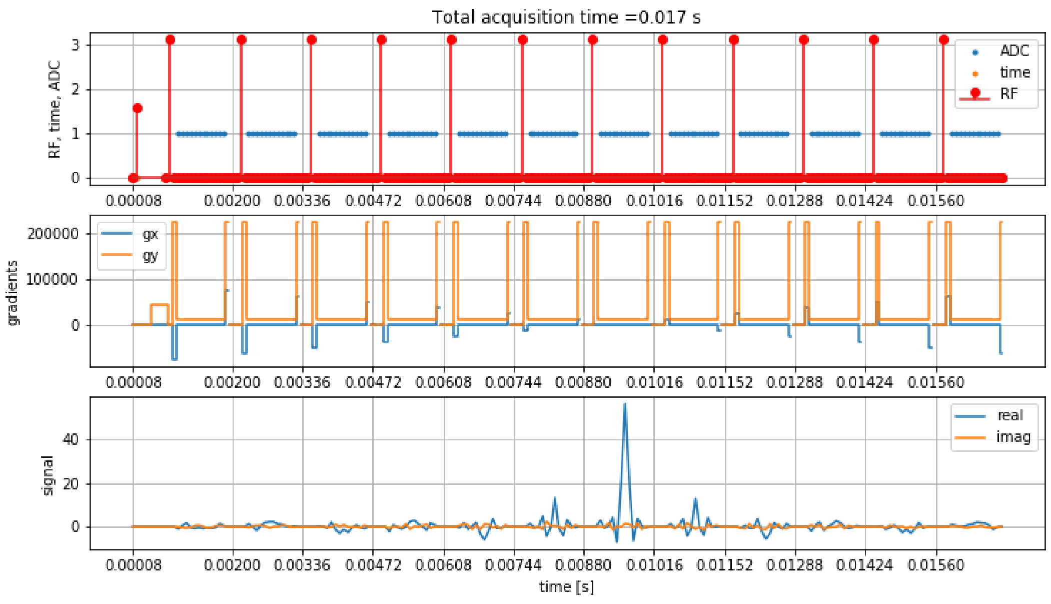

Figure 9: TSE sequence scheme (12x12), readout in y.

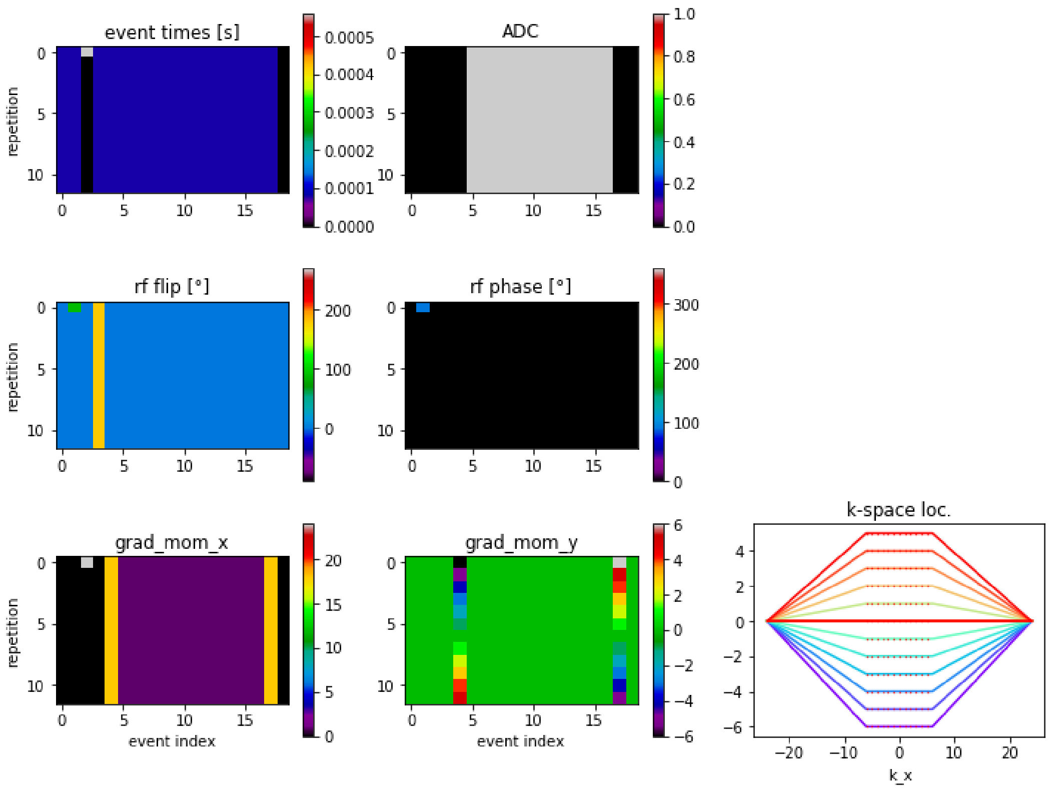

Figure 10: TSE sequence scheme picture (12x12), readout in x.

4.11 balanced SSFP imaging

In both fast imaging techniques FLASH and TSE we had additional spoiler gradients to remove higher echoes from past excitation. However, this is all magnetization that yields signal for the MR image. The driven equilibrium, TRUFI or bSSFP sequence uses all the magnetization to generate signal. The problem of different encodings is solved by balancing the gradients, that means that all gradient moments add up to zero at each TR and we are back in the k-space center after each TR. The timing must be such that the spin echoes and gradient echoes match and the RF phases of the excitation pulses are alternating so that the spin echo and gradient echo have the same polarity. Furthermore, an $\alpha/2$ pulse is used to catalyze the steady state that builds up during the balanced steady-state free precession (bSSFP) sequence.

1. $\alpha/2 = 10°$ pre-pulse and TR/2 delay
2. α pulse with 180° phase increment
3. prewinder gradient (read and phase)
4. readout gradient + ADC
5. rewinder gradient (read and phase) to k=0
6. back to 2.

2.-7. is repeated until all phase encoding steps are covered.

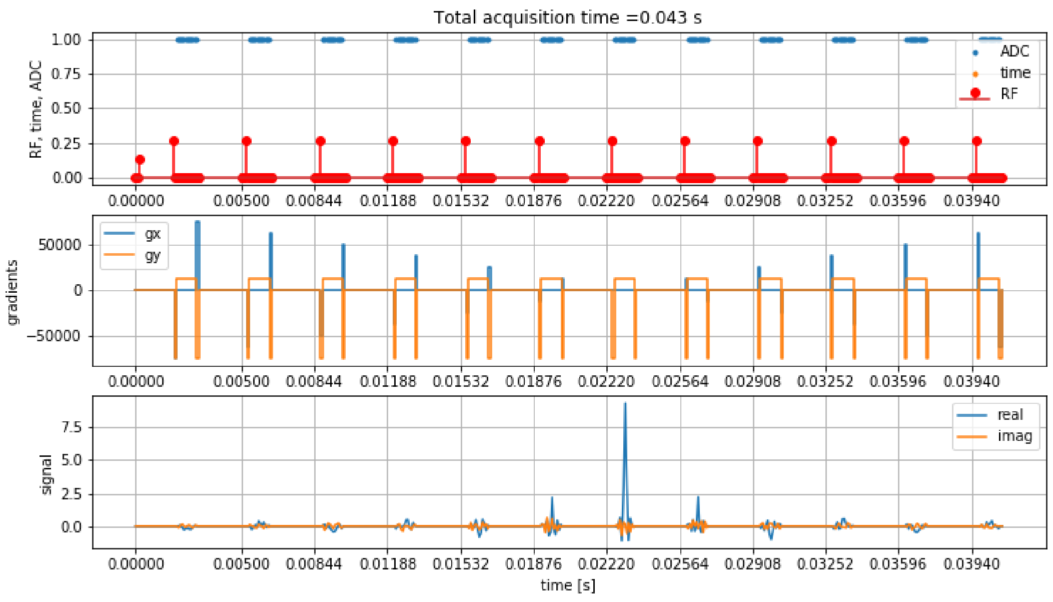

Figure 11: bSSFP sequence scheme (12x12), readout in y.

4.11.1 bSSFP links

- forelesninger pdf
- mriquestions.com SSFP
- http://mriquestions.com/4-or-more-rf-pulses.html

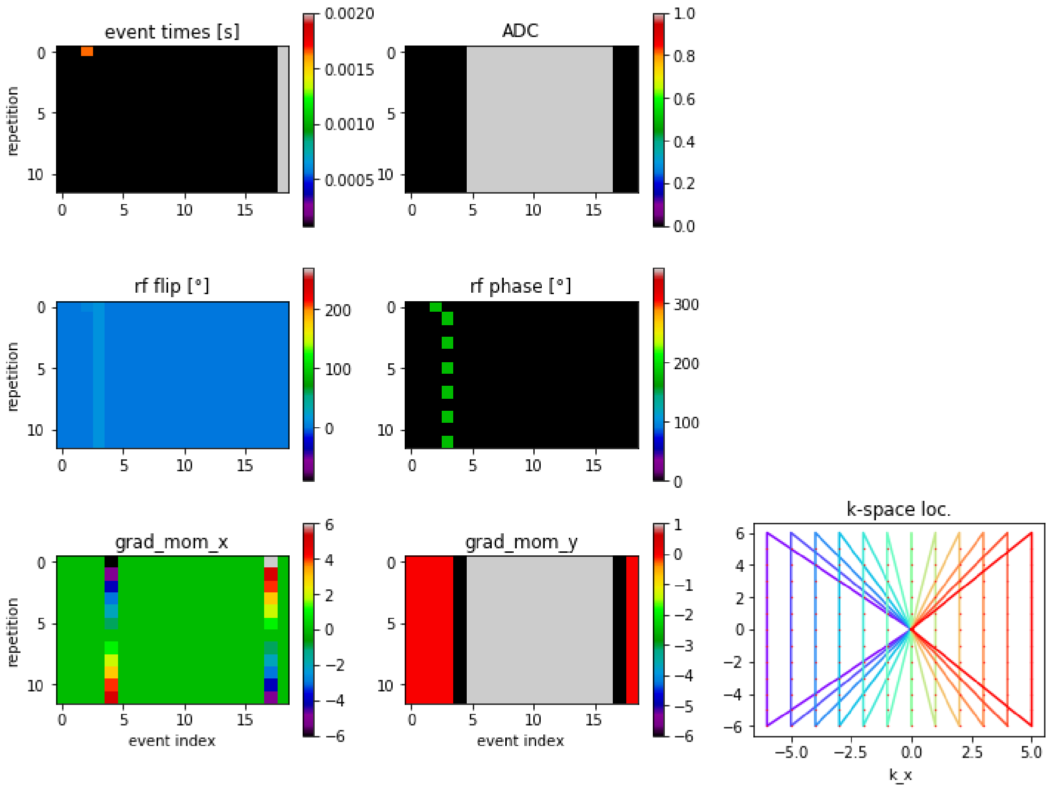

Figure 12: bSSFP sequence scheme picture (12x12), readout in x.

SECTION 5

Appendix

5.1 From induction to the demodulated signal

The origin of the signal is a voltage that is induced in the receive coils, because of a change in the magnetic flux

$$U_{ind} = -\frac{d}{dt}\Phi = -\dot{\Phi} \tag{22}$$

The magnetic flux Φ is changing in time due to the magnetic moments oscillating, this our magnetization vector.

We start from the signal equation in Haacke et al. (eq. 7.14)

$$signal \sim \frac{d}{dt}\int d^3r[B_x^-(\vec{r})M_x(\vec{r},t) + B_y^-(\vec{r})M_y(\vec{r},t) + B_z^-(\vec{r})M_z(\vec{r},t)] \tag{23}$$

where B^- is the receive field sensitivity.

As Haacke et al we neglect the much slower dynamic of M_z compared to M_{xy}. We further assume B^- to be $= 1$ homogeneously, so a flat receive sensitivity.

$$signal \sim \frac{d}{dt}\int d^3r[M_x(\vec{r},t) + M_y(\vec{r},t)] \tag{24}$$

The dynamic of $\vec{M}(\vec{r},t)$ is given by the solution of the Bloch differential equations given by Haacke et al. (eq. 4.25-4.27, and in complex form 4.32-4.34):

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t}M_+(\vec{r},0) \tag{25}$$

The T_2 relaxation should not be neglected, yet it can be assumed constant for the time derivative as $1/T_2 << \omega_0$.

In the presence of an additional linear magnetic gradient field $B_{Gx} = x \cdot G_x$ with $\omega_G = \gamma \cdot x \cdot G_x$ and analogous for y, this extends to

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\omega_G t}M_+(\vec{r},0) \tag{26}$$

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\gamma\cdot(x\cdot G_x(t) + y\cdot G_y(t))t}M_+(\vec{r},0) \tag{27}$$

As the gradient field is also much smaller compared to ω_0 their time derivative can as well be neglected, so we can already remove the time derivative from the signal equation:

$$signal \sim \frac{d}{dt}\int d^3r[e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\omega_G t}M_+(\vec{r},0)] \tag{28}$$

$$signal \sim -i\omega_0 \int d^3r[e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\omega_G t}M_+(\vec{r},0)] \tag{29}$$

After demodulation of the signal (Haacke et al eq. 7.27 and 7.28) we can remove all $\omega_0 terms$

$$signal \sim -i\omega_0 \int d^3r[e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\omega_G t}M_+(\vec{r},0)] \tag{30}$$

$$signal \sim \int d^3r[e^{-t/T_2(\vec{r})}e^{-i\omega_G t}M_+(\vec{r},0)] \tag{31}$$

This corresponds to the rotating frame form of the Bloch equations

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\gamma\cdot(x\cdot G_x(t)+y\cdot G_y(t))t}M_+(\vec{r},0) \tag{32}$$

this can be rewritten by the spatial frequencies $k(t) = \frac{\gamma}{2\pi}G\cdot t$ or more general for arbitrary gradients $k(t) = \frac{\gamma}{2\pi}\int_0^t G(t')dt'$

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i2\pi\cdot(k_x\cdot x+k_y\cdot y)}M_+(\vec{r},0) \tag{33}$$

Thus the demodulated signal for the time step t_0 to $t_0 + t$ is given by

$$signal \sim \int d^3r M_+(\vec{r},t) \tag{34}$$

$$signal \sim \int d^3r[e^{-t/T_2(\vec{r})}e^{-i2\pi\cdot(k_x\cdot x+k_y\cdot y)}M_+(\vec{r},t_0)] \tag{35}$$

with $withk = \frac{\gamma}{2\pi}\int_{t_0}^t G(t')dt'$ This is the continuous model for continuous space r and time t.