MAGNETIC RESONANCE

PULSE SEQUENCE PROGRAMMING

USING THE FRAMEWORKS MRTWIN AND PULSEQ

DESIGNED BY: M. ZAISS '19
UPDATED 02/2024

SIMULATION BY J.ENDRES

MORITZ.ZAISS@FAU.DE

IF YOU COME ACROSS ANY PROBLEMS, CONTACT ME: MORITZ.ZAISS@FAU.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 guided tour to discover this landscape on your own. The tools used here are research tools. Once you reach the end of the described paths, you can boldly go where no one has gone before.

About This File

MRTwin is a framework created by the research group of M.Zaiss based on Pulseq and a fast MRI simulation. Pulseq is an open file format created by Maxim Zaitsev (https://github.com/pulseq/pulseq), used herein via the python version PyPulseq (https://github.com/imr-framework/pypulseq). The currently used simulation is described here https://github.com/MRsources/MRzero-Core. The Latex file was created using Overleaf, based on the free template of Armin Dubert (armindubert2019@gmail.com).

SECTION 1

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/
- https://github.com/markus-nilsson/mri-simulator

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

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

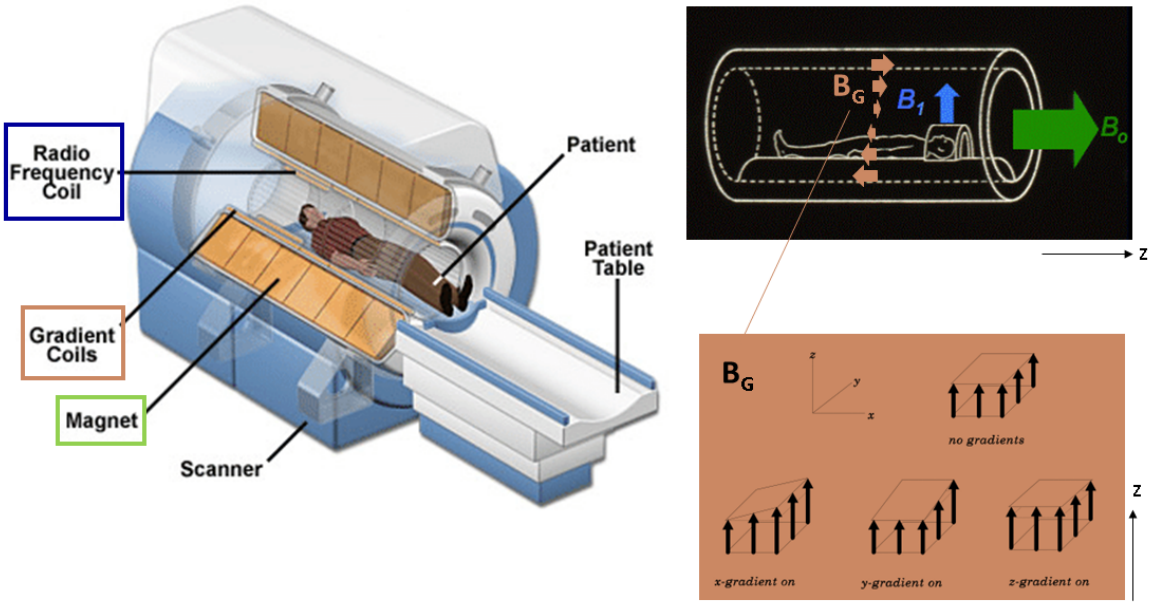

Figure 1: Typical MRI scanner with a super-conduction coil for generating the static B_0 field, a radio-frequency coil to generate a circular polarized B_1 field in the transverse plane (xy), and several gradient coils, to generate a linear alteration of the z-component of the magnetic field depending 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

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

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

$$\dot{\vec{M}} = \gamma\cdot\vec{M}\times\vec{B_0} \tag{3}$$

To solve this equation, create a second derivative in time

$$\ddot{\vec{M}} = \gamma\dot{\vec{M}}\times\vec{B_0} = \gamma\vec{M}\times\gamma\vec{B_0}\times\vec{B_0} \tag{4}$$

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

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

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

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

$$\vec{B}(\vec{r},t) = \vec{B_G}(\vec{x}',t) + \vec{B_1} = \begin{pmatrix} 0 \\ 0 \\ \vec{G}(t)\cdot\vec{r} \end{pmatrix} + \begin{pmatrix} B_1 \cdot \cos(\omega_{rf}\cdot t) \\ B_1 \cdot \sin(\omega_{rf}\cdot t) \\ 0 \end{pmatrix} \tag{7}$$

.

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

$$\alpha = \omega_1 \cdot t_p. \tag{9}$$

For varying pulses the product becomes an integral

$$\alpha = \int \omega_{\mathbf{1}}(\mathbf{t})\mathbf{dt}. \tag{10}$$

The rf phase ϕ_{rf} is defined by the angle of $\vec{\omega_1}$ in the x-y plane; in equations (7) and (8) above the vector $\vec{\omega_1}$ is in x-direction. In general $\vec{\omega_1}$ can point in any direction in the x-y-plane:

$$\vec{\omega_1} = \begin{pmatrix} \omega_1 \cdot \cos(\phi_{rf}) \\ \omega_1 \cdot \sin(\phi_{rf}) \\ 0 \end{pmatrix} \tag{11}$$

An RF event can therefore be defined by the tuple

- **rf event**: (rf flip angle, rf phase, rf frequency, rf usage)= $(\alpha, \phi_{rf}, \Delta\omega_{rf}, \text{usage})$

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. The fourth component is not a real property of the rf event, but is often used to tell the reconstruction if this pulse is meant as excitation pulse (usage=1), or as refocusing pulse (usage=2).

For only gradients, you can convince yourself, that depending on the spin position within the gradient field, also just a rotation with the frequency

$$\omega_G(\vec{r}) = \gamma \vec{G}(t) \cdot \vec{r} \tag{12}$$

around the z-axis occurs. The actual rotation angle depends here on the spatial position. Still, not the gradient, but the gradients area, or the gradient moment

$$\vec{\mathbf{g}} = \int \vec{\mathbf{G}}(\mathbf{t})\mathbf{dt} \tag{13}$$

of a finite gradient event defines this rotation angle by $\gamma \vec{g} \cdot \vec{r}$. Thus a gradient event has 4 degrees of freedom, the gradient amplitdes and the time, or the gradient areas and the time:

- **gradient event**: (g_x, g_y, g_z, t) or (G_x, G_y, G_z, t)

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 $\omega_0 \cdot \Delta t$. 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:

- **time delays**: Δt

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

These events will be generated in the MR sequence definitions or sequence schemes in section 1.4.

1.2.3 Pulse sequence events in code

In the python codes the mentioned events will be realized by Pulseq event blocks, as described in detail below.

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

$$U_{ind} = -\frac{d}{dt}\Phi = -\dot{\Phi} \tag{14}$$

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 6.1, this voltage can be measured and originates dominantly from the transverse magnetization:

$$signal \sim \frac{d}{dt}\int d^3r\,[M_x(\vec{r},t) + M_y(\vec{r},t)] \tag{15}$$

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

$$s(t) = \int_{\vec{r}} M_{xy}(\vec{r}) \cdot \exp\left(-i\gamma \cdot \vec{r} \cdot \int \vec{G}(t)\,dt\right) d^3r \tag{16}$$

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

$$s_x(t) = \int_{\vec{r}} M_x(\vec{r}) \cdot \cos(\gamma \cdot \vec{r} \cdot \vec{g})\,d^3r\, s_y(t) = \int_{\vec{r}} M_y(\vec{r}) \cdot \sin(\gamma \cdot \vec{r} \cdot \vec{g})\,d^3r \tag{17}$$

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. (16), if we now use a new variable

$$\vec{k}(t) = \vec{g}(t)\frac{\gamma}{2\pi} \tag{18}$$

we obtain something that looks exactly like a Fourier transformation from the real space to the spatial frequency space, or k-space.

$$s(t) = s(\vec{k}(t)) = \int_{\vec{r}} M_{xy}(\vec{r}) \cdot \exp(-i2\pi\vec{r} \cdot \vec{k})\,d^3r \tag{19}$$

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.

$$M_{xy}(\vec{r}) = \int_{\vec{k}} s(\vec{k}) \cdot \exp(-i2\pi\vec{r} \cdot \vec{k})\,d^3k \tag{20}$$

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 (16).

With equations (6), (19), and (20) 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.

A last remark is the emphasis on the k-space variable $\vec{k}(t)$. As the signal we acquire is actually the signal at a certain k-space coordinate $\vec{k}(t)$, it is helpful to keep in mind that this is really the integral of all gradients played out since the last excitation, or the sum of all previous gradient moments.

$$\vec{k}(t) = \frac{\gamma}{2\pi} \int \vec{G}(t)\, dt = \frac{\gamma}{2\pi} \sum_i \vec{g}_i \tag{21}$$

Thus, gradients let us move in the k-space. Moreover, 180-degree rf pulses invert all magnetization, thus all rotations become inverted an we get mirrored in k-space, i.e. $\vec{k} = -\vec{k}$. Excitation pulses take new magnetization from the z-axis without any rotation past, this after excitation we are at $\vec{k} = 0$ again.

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.

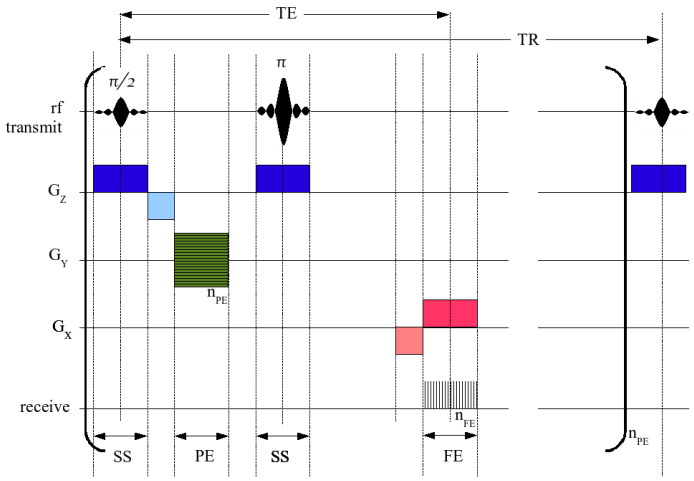

Figure 2: An MR sequence (spin echo). Taken under creative commons license from https:\\commons.wikimedia.org/wiki/File:MRI_2DFT_SE_PulseSequence.png

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 and is called the k-space trajectory of a sequence. 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.

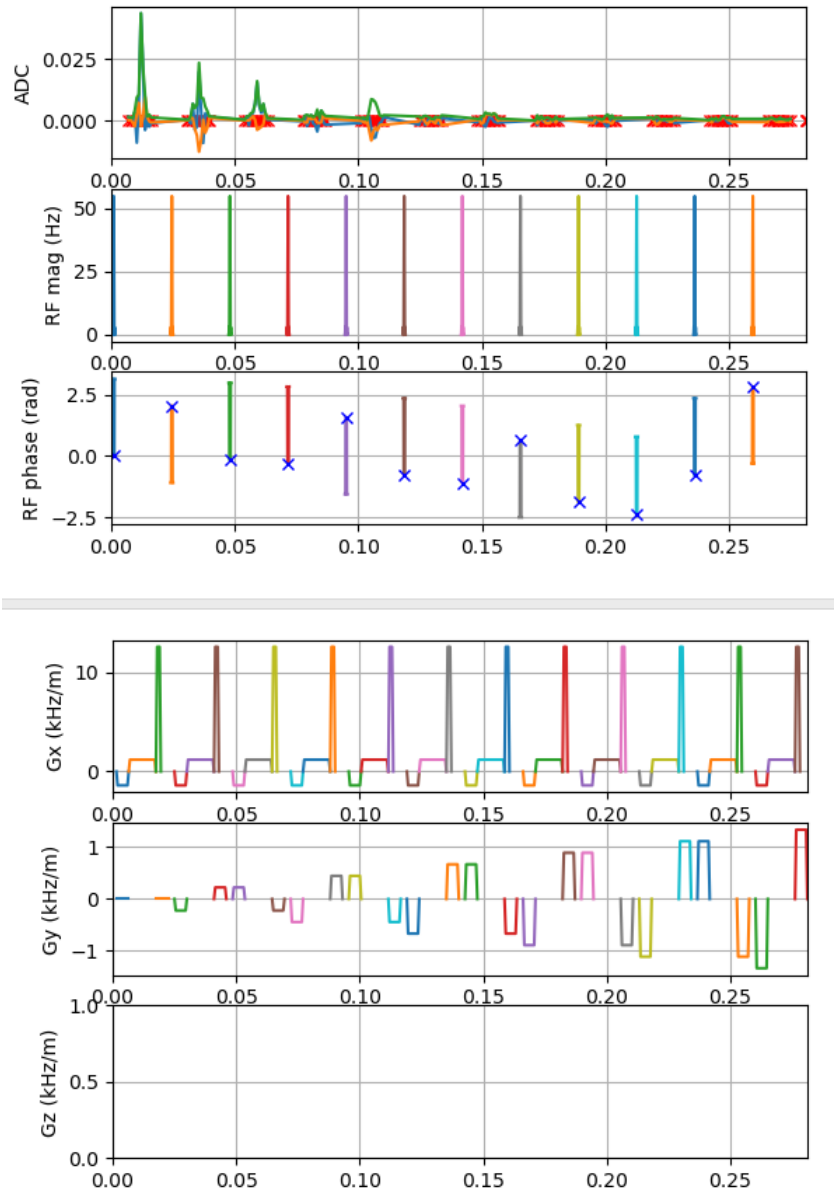

Figure 3: An MR sequence scheme (gradient echo, 12x12).

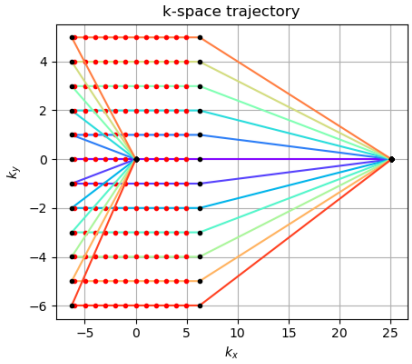

Figure 4: An MR sequence scheme as picture (gradient echo,12x12). This is the same sequence as in Figure 3. In addition to the events, also the k-space trajectory is given (bottom-right). This reflects the cumulative sum of played out gradient moments and the coverage of the acquired k-space. Full-coverage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

2

SECTION

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 commands as given in the readme and follow the readme instructions carefully: https://github.com/mzaiss/MRTwin_pulseq In the folder ./ex you find all exercises. To check you python skills and also test the versions of the installed packages you can run the file **./ex/ex_help01_python.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. Then follow the README_HOME.md I strongly recommend sypder as editor, as all test were done in this environment. Start spyder. To have plots as separate window, which is important as we add lines subsequently to some plots and also because it allows zooming, 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/ex/* 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

Goto https://github.com/mzaiss/MRTwin_pulseq follow the README_CIP.md
To start spyder run the following commands in a terminal:

1. module load python3/anaconda-2022.05
2. spyder

I strongly recommend sypder as editor, as all tests were done in this environment. Start spyder. To have plots as separate window, which is important as we add lines subsequently to some plots, 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/ex/* 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 useful 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 **ex_help01_python.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 **ex** in the main folder. The main exercise files (exA0x ...) are in this folder **MRTwin_pulseq/ex** or **./ex**, **which will always be the work directory in the following**
We follow some exercises of the package for this tutorial.

2.6 exB05 - 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 **exB05_GRE_2D_fully_relaxed.py**. This is a gradient echo imaging sequence, where everything is set up except some excitation flip angles are zero. When you run this script as it is you should see the plot as in Figure 5

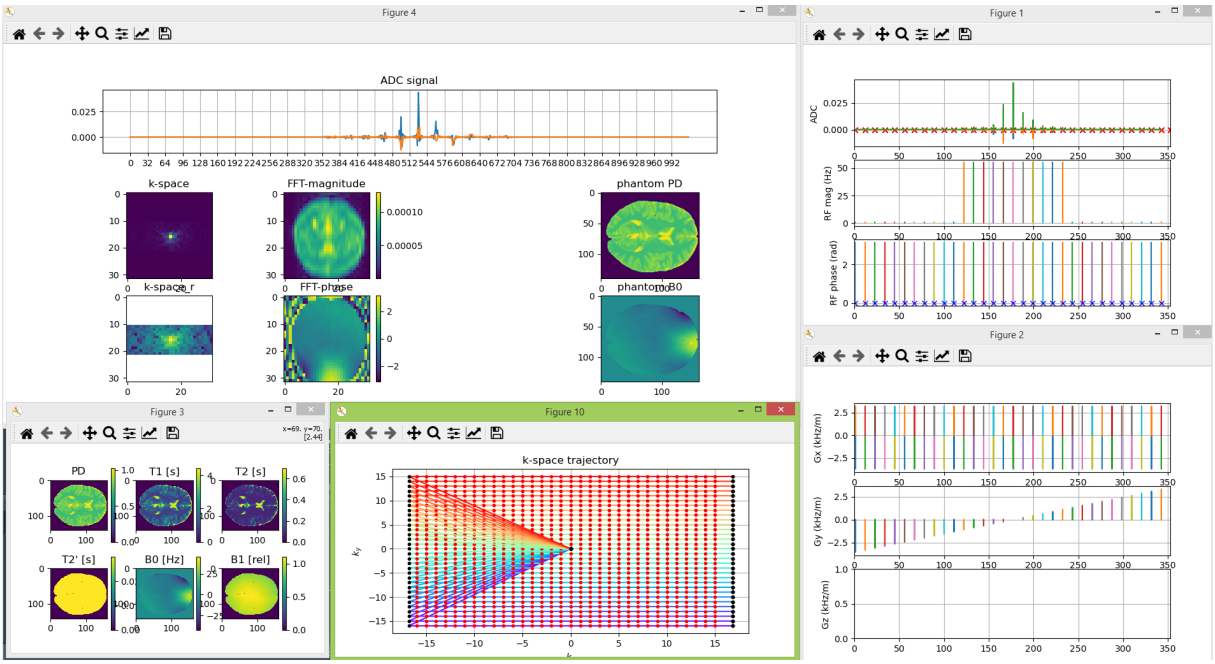

Figure 5: Output from exB05. On the right hand side you see the MR sequence schemes (SubFigure 1: ADC, RF amplitue, and RF phase, and SubFigure 2: gradient events). SubFigure 3 shows the Input for scanner -proton density (PD), relaxation times and field inhomogeneity. Subfigure 4 shows the acquired MR signal, the acquired k-space, the MR images - magnitude image and phase image. The images look quite blurry, we will fix this now.

Then search in this file for the term 'tutorial1'. You will find the following code, which defines as gradient echo MRI sequence

```
# ======
# CONSTRUCT SEQUENCE
# ======
for ii in range(-Nphase//2, Nphase//2):  # e.g. -64:63
    seq.add_block(pp.make_delay(1))
    if np.abs(ii)>1:    #tutorial1:    # (2)    if np.mod(ii,2)==1:
```

```
        seq.add_block(rf0)
    else:
        seq.add_block(rf1)
    gp= pp.make_trapezoid(channel='y', area=ii, duration=5e-3, system=system)
    seq.add_block(gx_pre,gp)
    seq.add_block(adc,gx)
    if ii<Nphase-1:
        seq.add_block(pp.make_delay(10))
```

The line `np.abs(ii)>1` excites only magnetization at the k-space center (lines -1,0 and +1), which leads to a blurry image. Change this to `np.abs(ii)>5` or `np.abs(ii)==30` to excite different or more lines and observe the effect on the images.
With more and more lines, you should see the image formation as depicted in Figure 6.

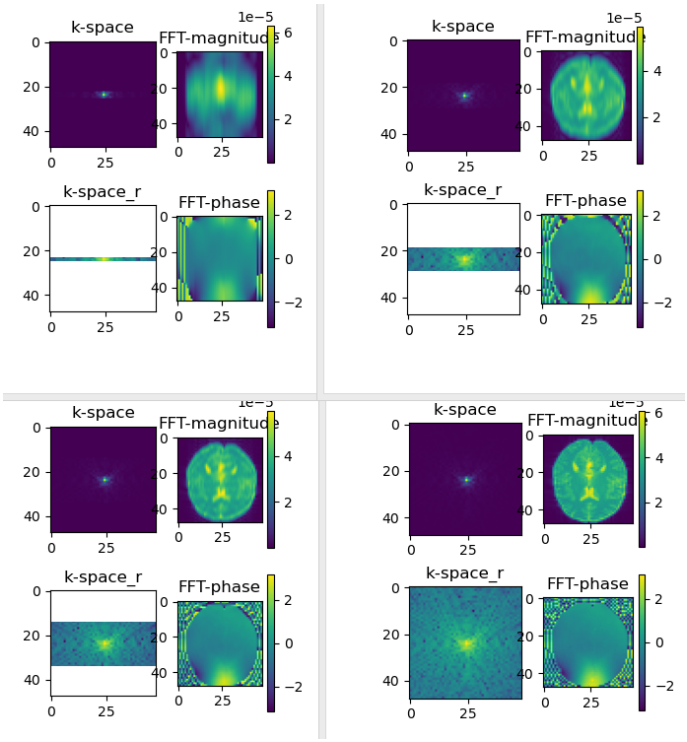

Figure 6: Output from exA08, when more and more lines are excited.

2.7 Overview over MR Twin simulation file

To get an overview of the whole framework, the file exA08_GRE_2D_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.

2.7.1 Section S0

The first section S0 sets up the environment with required paths and python libraries that are loaded. As well as some classes of MRTwin for for simulation and of course the python Pulseq package pypulseq for generating pulseq files.
NOTE: if you want to go to definitions quickly (CTRL-click), on some systems you have to add the included paths ('./codes, ./codes/GradOpt_python, ...) again in your spyder path manually in the menu (Tools/PYTHONPATH):

```
# %% S0. SETUP env
import MRzeroCore as mr0
import pypulseq as pp
import numpy as np
import torch
from matplotlib import pyplot as plt
import util

# makes the ex folder your working directory
import os
os.chdir(os.path.abspath(os.path.dirname(__file__)))

experiment_id = 'exB05_GRE_2D_fully_relaxed'
```

2.7.2 Section S1

In Section S1 the scanner system for Pulseq is defined. Here we have several physical limits of our real MR scanner, such as maximum gradient amplitude, maximum slew rate of the gradients. This is necessary to calculate gradient and rf events that can be played out by real hardware. We will learn more about these definitions in ex_help02.

```
# %% S1. SETUP sys

# choose the scanner limits
system = pp.Opts(
    max_grad=28, grad_unit='mT/m', max_slew=150, slew_unit='T/m/s',
    rf_ringdown_time=20e-6, rf_dead_time=100e-6,
    adc_dead_time=20e-6, grad_raster_time=50 * 10e-6
)
```

2.7.3 Section S2

In Section S2 the Pulseq sequence is defined. After some variable definitions you find the basic 4 building blocks as mentioned above:

- RF events - realized by the functions *make_sinc_pulse, make_block_pulse*

- gradient events - realized by the function *make_trapezoid*
- delays - realized by the function *make_delay*
- ADC events - realized by the function *make_adc*

They are added to the sequence e.g. by the code `seq.add_block(gx_pre,gp)`

```
# %% S2. DEFINE the sequence
seq = pp.Sequence()

# Define FOV and resolution
fov = 1000e-3
slice_thickness = 8e-3
sz = (48, 48)   # spin system size / resolution
Nread = 48    # frequency encoding steps/samples
Nphase = 48    # phase encoding steps/samples

# Define rf events
rf1, _, _ = pp.make_sinc_pulse(
    flip_angle=90 * np.pi / 180, duration=1e-3,
    slice_thickness=slice_thickness, apodization=0.5, time_bw_product=4,
    system=system, return_gz=True
)
rf0, _, _ = pp.make_sinc_pulse(
    flip_angle=0.001 * np.pi / 180, duration=1e-3,
    slice_thickness=slice_thickness, apodization=0.5, time_bw_product=4,
    system=system, return_gz=True
)
# rf1 = pp.make_block_pulse(flip_angle=90 * np.pi / 180, duration=1e-3, system=system)

# Define other gradients and ADC events
gx = pp.make_trapezoid(channel='x', flat_area=Nread, flat_time=10e-3, system=system)
adc = pp.make_adc(num_samples=Nread, duration=10e-3, phase_offset=0 * np.pi / 180, delay=gx.rise_time, system=system)
gx_pre = pp.make_trapezoid(channel='x', area=-gx.area / 2, duration=5e-3, system=system)

# ======
# CONSTRUCT SEQUENCE
# ======
for ii in range(-Nphase // 2, Nphase // 2):  # e.g. -64:63
    seq.add_block(pp.make_delay(1))

    if np.abs(ii) > 5:  # tutorial1:   # (2)   if np.mod(ii,2)==1:    # (3) np.abs(ii)==30:   with high FA
        seq.add_block(rf0)  # add rf0 with zero flip_angle
    else:
        seq.add_block(rf1)  # add rf1 with 90° flip_angle

    gp = pp.make_trapezoid(channel='y', area=ii, duration=5e-3, system=system)
    seq.add_block(gx_pre, gp)
    seq.add_block(adc, gx)
    if ii < Nphase - 1:
        seq.add_block(pp.make_delay(10))
```

2.7.4 Section S3

Here the sequence is checked, the sequence schemes are plotted and the sequence is exported as .seq-file called 'external.seq'

```
# %% S3. CHECK, PLOT and WRITE the sequence  as .seq
# Check whether the timing of the sequence is correct
ok, error_report = seq.check_timing()
if ok:
    print('Timing check passed successfully')
else:
    print('Timing check failed. Error listing follows:')
    [print(e) for e in error_report]

# PLOT sequence
sp_adc, t_adc = mr0.util.pulseq_plot(seq, clear=False)

# Prepare the sequence output for the scanner
seq.set_definition('FOV', [fov, fov, slice_thickness])
seq.set_definition('Name', 'gre')
seq.write('out/external.seq')
seq.write('out/' + experiment_id + '.seq')
```

2.7.5 Section S4

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*.
As a second option, a self defined phantom can be defined pixel-wise. Finally, the parameters can still be manipulated and are plotted.

```
# %% S4: SETUP SPIN SYSTEM/object on which we can run the MR sequence external.seq from above
sz = [64, 64]

if 1:
    # (i) load a phantom object from file
    # obj_p = mr0.VoxelGridPhantom.load_mat('../data/phantom2D.mat')
    obj_p = mr0.VoxelGridPhantom.load_mat('../data/numerical_brain_cropped.mat')
    obj_p = obj_p.interpolate(sz[0], sz[1], 1)
    # Manipulate loaded data
    obj_p.T2dash[:] = 30e-3
    obj_p.D *= 0
    obj_p.B0 *= 1    # alter the B0 inhomogeneity
    # Store PD and B0 for comparison
    PD = obj_p.PD
    B0 = obj_p.B0
else:
    # or (ii) set phantom  manually to a pixel phantom. Coordinate system is [-0.5, 0.5]^3
    obj_p = mr0.CustomVoxelPhantom(
        pos=[[-0.4, -0.4, 0], [-0.4, -0.2, 0],
             [-0.3, -0.2, 0], [-0.2, -0.2, 0], [-0.1, -0.2, 0]],
        PD=[1.0, 1.0, 0.5, 0.5, 0.5],
        T1=1.0,
        T2=0.1,
        T2dash=0.1,
        D=0.0,
        B0=0,
        voxel_size=0.1,
        voxel_shape="box"
    )
    # Store PD for comparison
    PD = obj_p.generate_PD_map()
    B0 = torch.zeros_like(PD)

obj_p.plot()
```

```
# Convert Phantom into simulation data
obj_p = obj_p.build()
```

2.7.6 S5: Run the simulation

The next section runs the simulation using the seq-file ex/out/external.seq and the just defined phantom object obj_p and plots the signal into the Pulseq sequence schemes in the ADC plot.

```
# %% S5:. SIMULATE  the external.seq file and add acquired signal to ADC plot

# Read in the sequence
seq0 = mr0.Sequenceimport_file("out/external.seq")
seq0.plot_kspace_trajectory()
# Simulate the sequence
graph = mr0.compute_graph(seq0, obj_p, 200, 1e-3)
signal = mr0.execute_graph(graph, seq0, obj_p)

# PLOT sequence with signal in the ADC subplot
sp_adc, t_adc = util.pulseq_plot(seq, clear=True, signal=signal.numpy())
```

2.7.7 S5b: Case of real MR scan

The external.seq file can also be directly executed at a real MR scanner, giving back the same complex signal. Then the signal is obtained via the following code:

```
signal = util.get_signal_from_real_system('out/' + experiment_id + '.seq.dat', Nphase, Nread)
```

This code expects that the seq file was send to an MRI scanner, played out, and that the data file was received back in the
out folder.

2.7.8 S6: MR image reconstruction

Here the signal is rearranged and 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.

```
# %% S6: MR IMAGE RECON of signal ::: #####################################
fig = plt.figure()  # fig.clf()
plt.subplot(411)
plt.title('ADC signal')
spectrum = torch.reshape((signal), (Nphase, Nread)).clone().t()
kspace = spectrum
plt.plot(torch.real(signal), label='real')
plt.plot(torch.imag(signal), label='imag')


# this adds ticks at the correct position szread
major_ticks = np.arange(0, Nphase * Nread, Nread)
ax = plt.gca()
ax.set_xticks(major_ticks)
ax.grid()

space = torch.zeros_like(spectrum)
```

```
# fftshift
spectrum = torch.fft.fftshift(spectrum, 0)
spectrum = torch.fft.fftshift(spectrum, 1)
# FFT
space = torch.fft.ifft2(spectrum)
# fftshift
space = torch.fft.ifftshift(space, 0)
space = torch.fft.ifftshift(space, 1)
```

For final depiction of images we use the util function util.MR_imshow(data, *args, **kwargs). This is because plt.imshow shows the matrix (x,y) as (col,rows), but we typically expect that the x-direction is from left to right so as rows. Thus, util.MR_imshow is used and transposes the data to (rows,col) before plotting.

```
plt.subplot(345)
plt.title('k-space')
util.MR_imshow(np.abs(kspace.numpy()))
plt.subplot(349)
plt.title('k-space_r')
util.MR_imshow(np.log(np.abs(kspace.numpy())))

plt.subplot(346)
plt.title('FFT-magnitude')
util.MR_imshow(np.abs(space.numpy()))
plt.colorbar()
plt.subplot(3, 4, 10)
plt.title('FFT-phase')
util.MR_imshow(np.angle(space.numpy()), vmin=-np.pi, vmax=np.pi)
plt.colorbar()

# % compare with original phantom obj_p.PD
plt.subplot(348)
plt.title('phantom PD')
util.MR_imshow(PD)
plt.subplot(3, 4, 12)
plt.title('phantom B0')
util.MR_imshow(B0)
```

SECTION 3

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

Overview of basic sequences

4.1 FID

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

$$s(t) = s_0 \cdot e^{-R_2^* \cdot t} \tag{22}$$

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

$$R_2^* = R_2' + R_2 \tag{23}$$

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 (22) 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:

$$s(t) = s_0 \cdot e^{-i \cdot w_0 \cdot t} \cdot e^{-R_2^* \cdot t} \tag{24}$$

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 6.1), thus the name free induction decay.
If the excitation flip angle is not 90°, the rotating frame equation (22) changes to

$$s(t) = s_0 \cdot \sin(\alpha) \cdot e^{-R_2^* \cdot t} \tag{25}$$

4.1.1 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 distributed 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.
Note: This is not the current implementation of the simulation, as we use an Extended-Phase-Graph approach (https://doi.org/10.1002/jmri.24619). However, the same dephasing effect is modeled there as well.

4.1.2 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

$$m_z(t) = (m_i - m_0) \cdot e^{-R_1 \cdot t} + m_0 \tag{26}$$

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:

$$s(t) = [(\cos(\alpha) - m_0) \cdot e^{-R_1 \cdot t} + m_0] \cdot \sin(\alpha) \cdot e^{-R_2^* \cdot t} \tag{27}$$

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.

4.2 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 7 shows the runners analogy of the spin echo and an animation of this phenomenon can be found here.
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.

$$\hat{s}(t) = s_0 \cdot e^{-R_2 \cdot t} \tag{28}$$

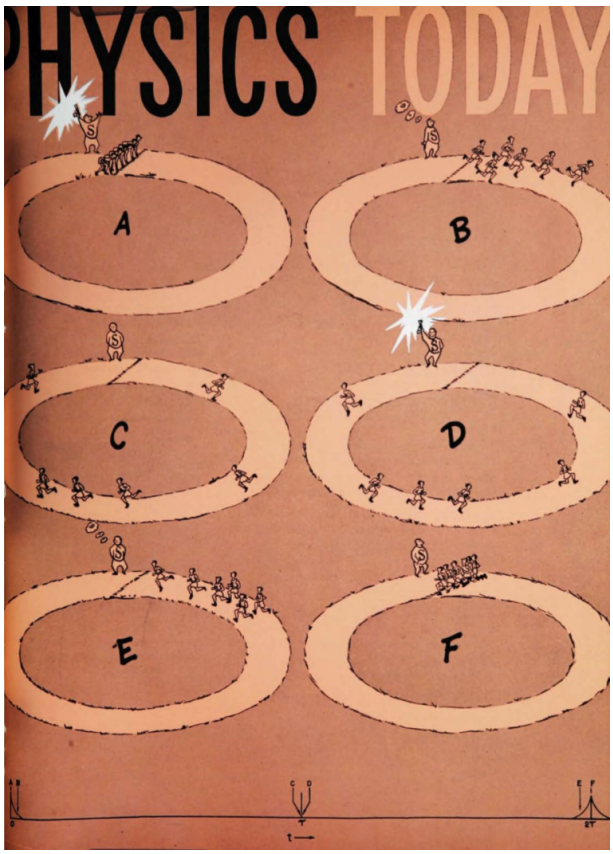

Figure 7: Erwin Hahn's famous spin echo analogy on the cover of Physics Today. The first bang reflects the 90° excitation, the second bang reflects the 180° refocusing pulse.

The 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 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 $t_{dephase} = t_{rephase}$.

4.3.1 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. (19) and (20)). 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.

4.4 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

4.5 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.

4.6 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.

4.7 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.

4.8 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

4.9 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.

4.10 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.

4.10.1 bSSFP links

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

5

SECTION

Exercises

The exercises are structured by sequence types:

- A : basic echoes
- B : GRE encoding
- C : spin echo and RARE
- D : balanced SSFP
- E : export to real system
- F : undersampling and reconstruction
- help : some helper files for coding

5.1 ex_help01_python

This file gives a quick overview over python, numpy, torch and plotting commands that are often used. At the end of the file some exercises with wrong code or code that doesn't run is given for you to fix. It is also helpful file to check weather all required packages are installed.

5.2 ex_help02_pypylseq

This file gives a quick overview over pypulseq and how to load and plot .seq-files, and how to generate basic sequence objects with pypulseq and write a seq file.
Hint: Blocks in Pulseq will always be played out after each other. If you want to play events out simultaneously you have to put them into the same block, separated by a comma.
You cannot edit blocks, to alter the events, you have to either define new events before adding them in a block. Or you alter old events after a first block, to then add the altered event to a new block.

1. Try to generate the sequence as displayed in Figure 8.

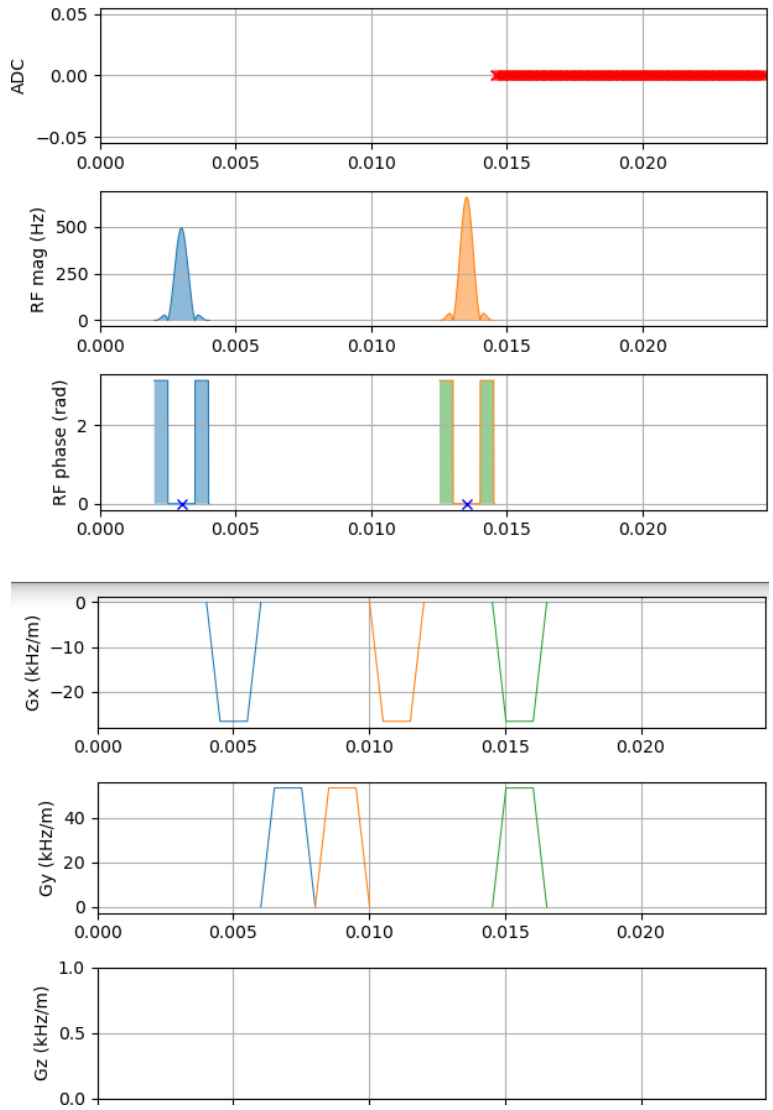

Figure 8: An arbitrary MR sequence. Try to generate it with pypulseq.

5.2.1 Exact pulse timing

When generating an rf pulse and inspecting it carefully, you will realize, that it has additional properties. Just put the enter the rf name in the console:

```
namespace(type='rf',
          signal=array([-2.17000952e-07, -1.73933907e-06, -5.88120996e-06, ...,
                        0.00000000e+00,  0.00000000e+00,  0.00000000e+00]),
          t=array([1.00000e-06, 2.00000e-06, 3.00000e-06, ..., 2.00998e-01,
                   2.00999e-01, 2.01000e-01]),
          freq_offset=0,
          phase_offset=0,
          dead_time=0.0001,
          ringdown_time=2e-05,
          delay=0.0005)
```

and you get the delay, the dead time and the ringdown time. The delay is a delay before the pulse is played out, even if you requested a delay of 0 ms in the make_sinc function, this delay is overwritten by either the dead time, or the rise time of the gradient, if it is a slice selective pulse (if it has a thickness). The dead time is the time it takes the RF coil to switch between transmit

and receive modes and is a security time where no events can be played out. Teh gradient rise time, depends on your gradient system, but typically your rf pulses are 0.5 ms delayed.
The ringdown time refers to the time period during which the RF coil continues to oscillate after the RF excitation pulse has been turned off. This ringing is due to the resonant nature of the RF coil, and can result in unwanted signals in the acquired data, leading to image artifacts.It counts as part of your pulse duration.
To calculate some important durations there exists some function:

```
rf.delay                    # delay before the start of the pulse due to dead time or gradient rise time
ct=pp.calc_rf_center(rf)    # rf center time returns time and index of the center of the pulse
ct[0]                       # this is the rf center time
rf.ringdown_time            # this is the rf ringdown time after the rf pulse
pp.calc_duration(rf)        # this is the rf duration including delay and ringdown time
```

For a pulse with requested duration 1 ms the total duration
is **0.00152 s** consisting of

- **0.00100 s** requested duration
- **0.00050 s** rf delay
- **0.00002 s** ringdown_time

Further, calc_rf_center(rf1) tells us that 0.00050 s after the delay, we have the center of the pulse, this at 0.00100 s after the start of the block.
With this information you can now do the next exercises

2. Create a sequence with a single rf pulse centered at exactly 0.1 s.
3. Create a second rf pulse exactly 0.2 s.

5.3 exA00_phantom

To be able to simulate a MR signal, we need to define at least a minimal **MR phantom object** describing our spin system. Get familiar with the pixelwise definition of MR phantom objects, and load more complex MR phantoms

5.4 ex01_FID

This is the simplest MR sequence that just consists of an ADC event and an rf excitation.

1. Add an rf and an adc event and observe the signal in the ADC plot. Why do we see only a decay and not a damped oscillation? Try to have the ADC at 12 ms, use calc_duration to get the duration of the RF pulse.
2. alter the flip angle (FA) and find the flip angle for max. signal S, guess the function S(FA).
3. Search for the B0 inhomogeneity if the object (obj_p.B0). In the on-resonant case this is 0. By changing the value (e.g. obj_p.B0 +=100) you can mimic the laboratory frame dynamic.

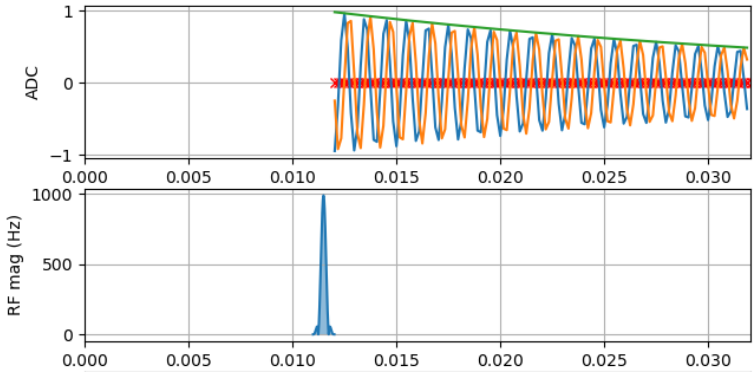

Figure 9: An FID in the on-resonant rotating frame (green) and in an off-resonant frame, eg. the lab fram (orange/blue).

4. Is the decay of the FID given by T1,T2,or T2*? Add more ADC blocks. When is the signal approximately 0? Estimate the numerical value of the decay rate R (or relaxation time T=1/R) of the FID signal.
5. Alter the rf phase offset and adc phase offset. What do you observe? What is the most general condition to get a real positive signal?
6. Add different delays between rf and adc, try different adc durations. What is the best timing for highest FID signal?
7. (Bonus): use scipy.optimize.curve_fit to fit the FID signal and find the exact value of the decay rate. Alter the relaxation times of the pixel phantom to find out what T2* or R2* is.
8. Save the current file with the suffix _single_echo
9. Repeat the FID sequence multiple times, add a T1 recovery time delay (Trec) of 1 s after each ADC. Why is the first signal higher than all the others?
10. alter the recovery time delay Trec, when do you achieve equal signal in all ADCs / the highest signal?
11. Cover a range of different Trec in one measurement from 0.1 to 3 s. What is the recovery rate R or recovery time T? T1,T2,or T2*?
12. Save the current file with the suffix _multi_echo

5.5 exA02_spinecho

1. This ex starts from the single FID. Add a delay of a few ms between the rf and the adc. Add a further rf event to create an echo in the ADC (starting at 12 ms). Which flip angle generates the highest echo? (if you still want to see the FID, you can also add a second FID.)

2. try to find a way so that the echo forms in the middle of the ADC. What is the general rule for the echo position in time?

3. Try to adjust rf phase offset / adc phase offset to get (i) a real and negative echo, (ii) a real and positive echo.

4. Save the current file with the suffix _single_echo

5. Add more ADCs and refocusing rf pulses to retrieve the spin echo in every ADC. What is the numerical vakue of the decay rate of the echo top? IS this T1, T2 or T2*?

6. (Bonus): use scipy.optimize.curve_fit to fit the maximum SE signal in every ADC and find the exact value of the decay rate. Alter the relaxation times of the pixel phantom to find out what the decay rate is.

7. Save the current file with the suffix _multi_echo

5.6 exA03_gradient_echo

1. This starts again from the FID example. Now add a gradient to the ADC block with the same duration. What do you observe when altering the gradient area from zero to higher values?

2. Use a gradient area that equals the number of sampling points. Add a second ADC and try now to recover the decayed signal without using an additional RF event, but an additional gradient.

3. What is the condition to get the gradient echo in the center of the second acquisition phase?

4. Add a short gradient before the first ADC, what changes?

5. Remove the last ADC and its gradient. Can you generate a gradient echo which is centered directly in the single ADC block (starting at 12 ms)?

6. Save the current file with the suffix _single_echo

7. generate a whole train of gradient echoes after one excitation

8. (Bonus): use scipy.optimize.curve_fit to fit the maximum GRE signal in every ADC and find the exact value of the decay rate. Alter the relaxation times of the pixel phantom to find out what the decay rate is.

9. Save the current file with the suffix _multi_echo

5.7 Bonus: FID - SE - GRE

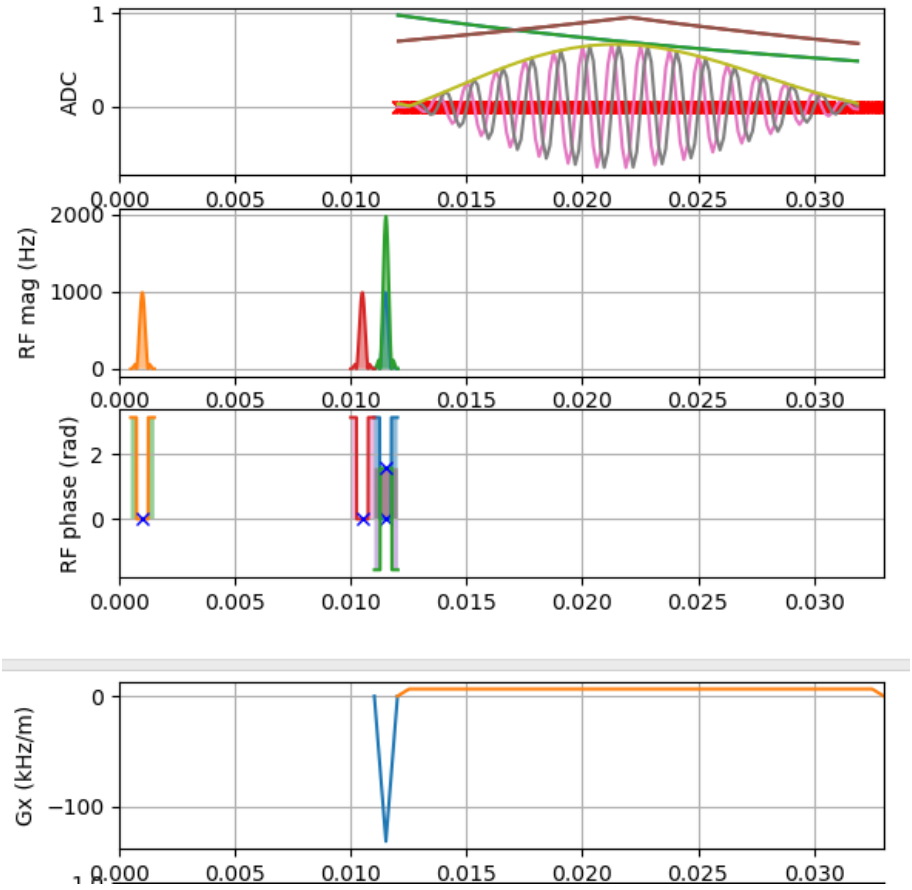

Figure 10: The first 3 exercises (single_echo) with the right timing are shown here in the same plot. Recreate this figure (ADC starts at 12 ms). Identify the FID, SE and GRE in the plot. What can you conclude form their relative sizes?

5.8 exA04_STE

stimulated echo. This is a sequence with three rf-pulses. Run the sequence and investigate the signal. There is the FID signal and 4 additional echoes. Can you explain all of them? Use the alteration described in the following to track down the origin.

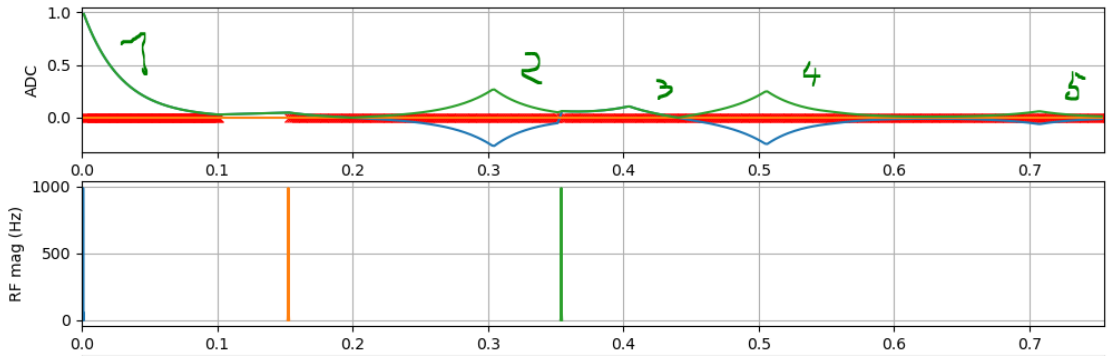

Figure 11: A sequence with three 90 degree pulses generates 5 echoes.

1. Try altering the flipangle to a 90, 90 90 scheme, and a 90, 180, 180 scheme, or 90, 120, 120 scheme. is this a spin echo? what is its TE, where was this excited?
2. Try adding delays to study the origin of the echoes. (prolong the second block by 1.5s)
3. Try adding gradients in the first ADC, does it affect the echoes in the last ADC?.
4. read http://mriquestions.com/stimulated-echoes.html and identify A, B, C, D, E
5. Try to define the different echo times in your code, TE_SE, TE_STE Move the STE after the SE, or before the SE. Move them to overlap perfectly
6. Add one gradient event to kill the spin echo in the third repetition
7. Add two gradient events to only have the STE left in the third rep.

5.9 exA05_incoherent_echo_train

This is a sequence with one 60 and many 120 degree rf-pulses. The distance between the first two pulses is different than between all others. Also the ADc is not centered between the pulses. The spin system here was manipulated for short T2* and long T2 for better visibility of the echoes.

1. Convince yourself that this leads to many echoes that add up almost chaotically, and that the ADC center gets always a different intensity.
2. Add a small random time deviation e.g.
   `seq.add_block(make_delay(np.random.randint(5)*1e-3))`
   which makes the sequence even more incoherent. Observe the echo train and convince yourself that after few pulses, this gets chaotic.

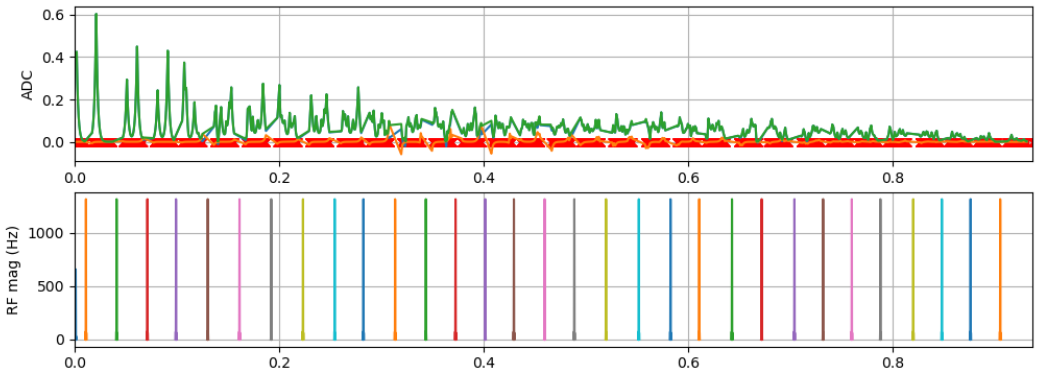

Figure 12: Incoherent echo train as described by exA05.2

If you want to understand such echo trains you should read about *phase graphs*:
https://doi.org/10.1002/cmr.1820030302
https://doi.org/10.1002/jmri.24619

5.10 exB01_gradientecho_pixel

This file starts from A03.

1. Alter the position of the pixel in the phantom (obj_p). Here you can now make the most important observation for encoding. What do you observe in the signal when you move the pixel from the center to the edge?
2. Why is only the movement of the pixel in one dimension affecting the signal?
3. Instead of x gradient use a y gradient!
4. Create a random pixel position using
   obj_p[np.random.randint(64),np.random.randint(64),:]=torch.tensor([1, 1, 0.1, 30e-3, 0, 1]).
   Can you estimate where the pixel is solely by looking at the signal with x and y encoding?

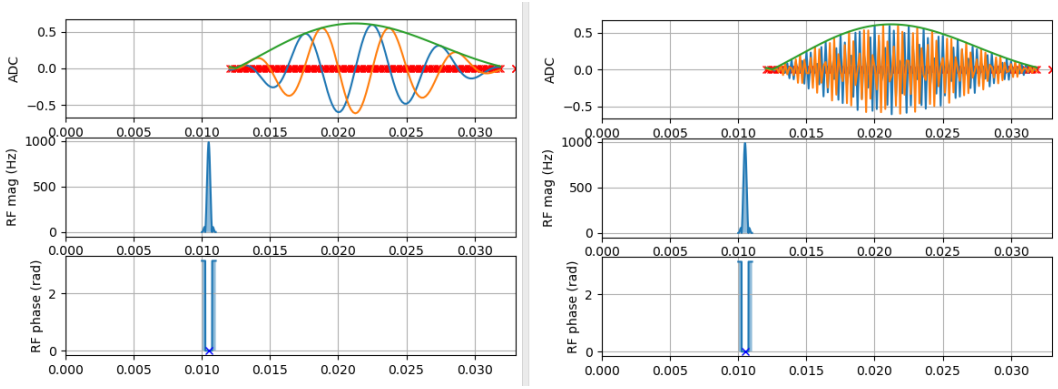

Figure 13: GREs of a pixel phantom. Left: x-gradient-echo, right: y-gradient-echo. Where is the pixel?

5. Make a second pixel. What do you observe now?

5.11 exB02_gradient_echo_freq_enc

This file starts from B01. We want now to create the 1D - Fourier transform of the signal to get a 1D image. This is also called frequency encoding.

1. This file has now a reconstruction section S6, go though this section and understand what plots are created. To separate different frequencies, perform a fourier transform of the signal. Learn from ex_help03_fft.py
2. Compare your result to the upsampled phantom projection given at the end of the file.
3. What happens if you change y to x gradient encoding?
4. What happens if you use both x and y gradients simultaneously?
5. What happens if you alter the number of Nread samples?
6. (Bonus) Repeat the whole sequence multiple times. Run a for loop to FFT every repetition in the same manner.

5.12 exB03_gradient_echo_phase_enc

This file starts from B01. We want now to create the 1D - fourier transform of the signal in the phase direction to get a 1D image. This is also called phase encoding.
For this we need multiple phase encoding steps, thus we need multiple repetitions of our gradient echo sequence. The number of repetition steps is equal to the number of phase encoding steps, which is why we call this number Nphase.

1. Use the same rf and gradient blocks in every repetition, add a recovery time at the end of each repetition. What is the recovery time needed to have same echo amplitudes? is there a general rule for this?

2. Comment out the frequency encoding gradients, if not already done. You should get an approximately constant signal in every repetition.

3. Now add phase encoding gradients in y-direction; they are not played out during the ADC, but before, like a re-winder. Give each repetition a different phase encoding gradient moment. Now you should see an echo along the repetition dimension or phase encoding dimension. Try to center the echo again.

4. As every repetition has Nread samples, create a loop over Nread to generate the Fourier transform along Nphase, or every sampling point in read direction. Compare your result to the upsampled phantom:

5. Increase Nphase, what do you observe? (You can reduce Nread to accelerate the simulation)

6. What happens if you change y to x gradient encoding

7. what happens if you use both x and y gradients?

5.13 exB04_gradient_echo_freqphase_2D

This file starts from B03 and combines B02 and B03. We want now to have both frequency and phase encoding simultaneously, to be able to create the 2D - Fourier transform of the signal, and to get a 2D image with full encoding.

1. Outgoing from the phase encoding exercise, we can again activate the frequency encoding gradients (in x) in addition to the phase encoding gradients (in y). Instead of the projection of the phantom you should now see additional gradient echoes in each phase encoding step.

2. To have full encoding you now have to Fourier transform a second time, once a loop over Nread and FFT along Nphase, and once a loop over Nphase and an FFT over Nread. If you got this right, your plot will show the profiles along x of the phantom at each phase encoding position in y.

3. Plot the result as an image,
   plt.imshow() or
   plt.imshow(np.abs(space), interpolation='none',aspect = sz[0]/Nread)

4. The two Fourier transforms can be concatenated using np.fft.ifft2.

5. Set the read and phase encoding steps to 64. Change the phantom to 2D phantom, or brain

6. Use plt.imshow() to show the magnitude image (np.abs), and the phase image (np.angle); if the phase has wraps or chekckerboard, check your fftshift. As we reshaped the signal into (Nphase,Nread), but phase encoding is typically in y, we might need a transpose to get the correct image orientation as the phantom. Compare with the phantom plots!

Now you have your first fully encoded MR image!

5.14 exB05_GRE_2D_fully_relaxed

This file is a cleaned up version of the solution to exB04.

1. Plot the k-space as an image. Use abs(kspace) and log(abs(kspace). Plot the k-space trajectory.

2. Now, excite only certain k-space lines with 90 degree, set other rf flip angle close to 0.

3. What happens if you only excite every second line?

4. With different settings, try different phantoms. What is the meaning of the angle or phase image?

5. Select again all k-space lines. At the end of section S5, noise is added to the simulated signal. Lower the flip angles to see what the lowest reasonable flip angle is.

6. Prolong the echo time, what do you observe in magnitude and phase images?

5.15 exB06_GRE_to_FLASH_2D

The current sequence has a very long scan time.

1. Check the total scan time of the sequence in the plot. Where do we loose most time?

2. Lower the recovery time after each repetition to 0.1 s. What do you observe? What is the shortest time to still have a good image?

3. Lower the excitation flip angle to 5 degree. What do you observe? What is now the shortest time to still have a good image? (This is also called "Long TR spoiling") Read more here: http://mriquestions.com/spoiling—what-and-how.html.

4. Turn off all phase gradients and look at the signals. Do you see additional non-centered echoes when you reduce TR? Where are they originating from?

5. Find a way to get rid of transverse magnetization from the previous rep using a gradient. (gradient spoiling, spoiler or crusher gradient) can you now go even shorter with the event times? How short?

6. Another artifact can arise from the fact that we always flip the magnetization in the same direction in each repetition. Implement an altered rf phase in each repetition (RF spoiling / RF cycling)

7. Include a phase backblip to balance all phase gradients, so the endpoint of all k-space trajectory is the same.

8. How fast can you now go without artifacts? Calculate the speedup compare to the fully relaxed solution.

With combined gradient and RF spoiling, you have generated the famous FLASH sequence.

5.16 exB07_FLASH_2D_reordering

Sometimes it is useful to acquire the center k-space first. Play with ex_help04_permute.py to realize centric reordered phase encoding. To have the FFT working the same permutation has to be done with the raw k-space

5.17 exB08_FLASH_2D_MP

The current sequence is the FLASH sequence.

1. Add an inversion pulse before the sequence. Add a certain recovery time after this pulse. If it's not an 180 degree pulse you might need a spoiler gradient.

2. Play with these two properties flipangle and recovery time.

3. Can you achieve a dark CSF? That's a FLAIR

4. Can you achieve a dark CSF and dark WM? That is a DIR: http://mriquestions.com/double-ir.html

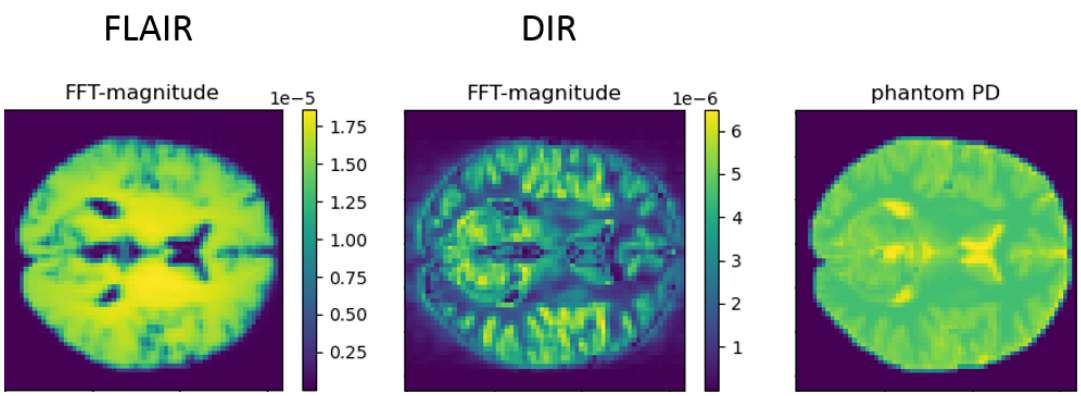

Figure 14: A centric FLASH with FLAIR and DIR magentization preparation.

5.18 exB09_GRE_EPI

This is currently a FLASH sequence like B08.

1. remove all RF events except for the very first one, make this 90°, remove all phase encoding gradients, remove all rf phases Now, there should be only one echo in the very first repetition.

2. Think of a way to get back again some magnetization in the second repetition without using an additional rf event, but a gradient.

3. If the last task was successful, do the same trick for all repetitions. You are fighting against T2* decay, thus, your sequence must be fast. Speed-up the sequence until you see an echo in every repetition.

4. Find a strategy to cover the full k-space again, by adding phase encoding-gradients. Large gradients require more time, you need to find a strategy with small gradients.

5. If your image looks weird, you might have forgotten something. We saw such ghosts before, when every second line was missing. Analyse carefully the k-space trajectory plot. What is written in your ADC signal? Check ex_help05_reverse to find a solution.

6. The image can still show a ghost. This is a so called N/2 ghost. See mriquestions.com N/2 ghost for more info. However, here in the simulation it should be possible to remove it completely. Carefully analyse you ADC trajectory.

7. Now you have an echo-planar imaging sequence, EPI, one of the fastest MRI sequences. While its speed is amazing, it also has some drawbacks: Try increasing the B0 inhomogeneity (obj_p.B0*=2;) Try to prolong the event times. What do you observe? Can you explain this?

8. (Bonus) You could save even more time with Extended Trapezoidal gradients. Try out make_extended_trapezoid.

5.19 exC01_spinecho_2D

1. This is starts from A02. What is TE and TE/2? What is TR? Add code to calculate both.

2. We want to generate a gradient echo at the same time point as the spin echo. So add the frequency encoding gradients accordingly.

3. Add a recovery event add the end, and repeat the SE for Nphase times. Add frequency encoding and check that all echoes have the same intensity.

4. Investigate the k-space trajectory plot carefully: What is the effect of the 180 degree pulse on the k-space location? Considering this, add the phase encoding gradients correctly to generate an image.

5. Alter the echo time TE, you should see the T2w of the SE MRI. For long enough TE you should see only brain liquor.

6. Decrease the repetition time TR, what is the shortest total acquisition time (TA) you can go with still good quality?

5.20 exC02_spinecho_to_RARE_2D

This is starts from C01, which is relatively slow. We want to generate a so called turbo spin echo (TSE), or Rapid Aquisition relaxation enhanced (RARE).
The idea is similar to the GRE EPI, we can also reuse the transverse magnetization after the first acquisition.

1. Try to do so, by using now RF events instead of gradients only. First try to get another echo in the second repetition, without a fresh 90° pulse. Remove gradients for testings.

2. Once you get an echo in the second repetition, repeat to get echoes in all repetitions. As you fight against T2-decay you must be again fast. Do all echoes have the same sign?

3. Add back the frequency encoding gradients. What is needed to have the same gradient echo in all repetitions? Do you need the rewinder gradient? If all echos look proper, add back the phase encoding to get RARE MR imaging.

4. Most probably you still have imaging artefacts. As for the RARE the timing is very important, thus check again that the echo time TE/2 from center of the 90 to the center of the first 180 pulse, and the TE between the center of the 180 pulses match exactly, and that you have the ADC centered around the TE. The function calculate_duration can help. Some events have have a rise-time or delay times that need to e considered.

5. Try reordering of the sequence, e.g. centric reordering, how does this affect the contrast, the image?

6. What is the effective echo time for centric or linear reordering?

7. Instead of 90-180 try a 30-90 scheme. Check the signals without phase encoding again. Do you see additional echoes?

8. To remove the higher echoes, move further out in k-space with the rewinder and back before the readout. Your k-space trajectory must look like this: < = >

5.21 exD01_bSSFP_2D

This starts from B05 which was the fully relaxed GRE sequence. flip angle was set to 5 degree

1. As before let us decrease the recovery time. This time make it very short.You should observe an image with artifacts when going from 5 to 0.002 s. Last time we tried to get rid of higher echoes. This time we want to understand them better.

2. We saw in exA04 (stimulated echo) that gradients act on certain echoes, and older echoes contain their complete gradient history. Thus, echoes can be at the same time point, but can have a different encoding as their transverse magnetization saw different gradients.

This can actually be observed in the k-space plot: do you see the additional intensity at the edge of the k-space.

3. In the second repetition, the FID or GRE signal starts at k=0, then the rewinder and readout is applied. At which k-space location does the spin echo start?How can you realize that also the spin echo starts at k=0 at the beginning of the second repetition?

4. You might still see some artifacts, especially in the phase. This is because the spin echo and the FID will have a different phase. To correct this you must alter the rf phase in every cycle.

5. Now you have a balance ssfp sequence! If you switch of the gradients again, you will see that it oscillates in the beginning: This can be solved using a prep-pulse, a so called alpha/2 pulse with the correct timing.

6. try different phantoms, manually increase the B0 inhomogeneity until you see the stopping bands.

7. Is there any stimulated echo?

8. If all timing and prep is correct, you can also try centric reordering.

5.22 exD02_bSSFP_2D_rotatedFOV

This starts from solD01.
We want to rotate the FOV for this sequence now. Remember: a 90 degree rotation we could achieve by switching the x and the y gradients. A general rotation needs for every gradient always a mixture of x and y. The transformation you can get by recapitulate the
https://en.wikipedia.org/w/index.php?title=Polar_coordinate_system

5.23 exE01_FLASH_2D

Instead of simulating, we will now send the sequence file to a real MRI scanner.
To do so, we first have to make our sequence scanner ready.
The first problem is that we always assumed a 2D object, at a real scanner the object is 3D and we need to do slice selection, thus we need slice selection gradients with our rf pulses.
https://mriquestions.com/slice-selective-excitation.html
We get them automatically from pypulseq, and need to add them properly to our sequence:

```
rf1, gz, gzr = pp.make_sinc_pulse(
    flip_angle=90 * np.pi / 180, phase_offset=90 * np.pi / 180, duration=1e-3,
    slice_thickness=slice_thickness, apodization=0.5, time_bw_product=4,
    system=system, return_gz=True
)
...
...
seq.add_block(rf1, gz)
seq.add_block(gzr)
```

Be careful with altered timing!
The second problem is, that we always worked in unit coordinates in k-space 1/m. We just matched the object size to 1 m. Remove the line

```
obj_p.size=torch.tensor([fov, fov, slice_thickness])
```

Now you see the actual size of the phantom in a 1 m FOV. Now you need to adjust the FOV or zoom!
The gradient scaling to get the right FOV is of course very important for a real scan.
To send the sequence to the real scanner, you can add in section S5 the following lines:

```
use_simulation = False

if use_simulation:
    seq_file = mr0.PulseqFile("out/external.seq")
    seq0 = mr0.Sequence.import_file("out/external.seq")
    #seq0.plot_kspace_trajectory()
    graph = mr0.compute_graph(seq0, obj_p, 200, 1e-3)
    signal = mr0.execute_graph(graph, seq0, obj_p)
    spectrum = torch.reshape((signal), (Nphase, Nread)).clone().transpose(1, 0)
    kspace = spectrum
    # PLOT sequence with signal in the ADC subplot
    plt.close(11);plt.close(12)
    sp_adc, t_adc = mr0.util.pulseq_plot(seq, clear=False, signal=signal.numpy())

else:
    signal = mr0.util.get_signal_from_real_system('out/' + experiment_id + '.seq.dat', Nphase, Nread)
    spectrum = torch.reshape((signal), (Nphase, Nread, 20)).clone().transpose(1, 0)
```

You see that there is a difference between the signals.
The simulated signal has (Nphase, Nread) elements, the real signal is acquired woth a 20-channel head coil, thus more signal elements are available (Nphase, Nread,20). We have to either FFT all coil data, and sum it. Or we just look at one individual coil. Either before FFT select one coil:

```
spectrum =spectrum[:,:,0]
```

or after FFT do:

```
if use_simulation==False:
    space = torch.sum(space.abs(), 2)
```

The second option will have more signal (sum of squares of all coils), but we loose the phase information. The first option has less signal, but we can reconstruct the phase.
To make sure that your sequence is not affected by previous sequences, you can add a 5 s delay before your sequence, so you get the full signal.
Finally, alter the experiment id by adding your name as a suffix:

```
experiment_id = 'exE02_RARE_2D_Meier_A'
```

Now you have to manually copy the seq file (exE02_RARE_2D_Meier_A.seq) to a FAUbox folder. Then after the scan there will be a .seq.dat file with the same filename. If you copy this back, the code will read the real MR data and reconstruct it.

5.24 exF01_bSSFP_2D_radial_nufft

This starts from solD02.
Instead of Cartesian sampling, we want to realize a radial sampling scheme, which means we acquire the spokes of a wheel, and every acquired line runs through the center of the k-space, just with a different angle, until the full k-space is covered.

1. Try to sample the k-space not Cartesian, but with radial spokes
2. Reconstruct this by using the non-uniform FFT

5.25 exF02_bSSFP_2D_radial_CS

Try to use the CS approaches for the radial undersampled data.

1. Reduce the number of spokes by Nphase,
2. provide the regridded kspace and the undersampling pattern of the radial trajectory to to the CS algorithm of the previous exercise.

5.26 exF03_simple_undersampled_CS

Try to use the CS approaches for the Cartesian undersampled phantom data

1. For this just use the code as is and play with the undersampling pattern and the iterative reconstruction parameters denoising_strength and number_of_iterations in the CS reconstruction part S6.
2. How many points can be excluded?
3. You can also try wavelet domain (uncomment code line)

SECTION 6

Appendix

6.1 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

$$U_{ind} = -\frac{d}{dt}\Phi = -\dot{\Phi} \tag{29}$$

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)

$$signal \sim \frac{d}{dt}\int d^3r\,[B_x^-(\vec{r})M_x(\vec{r},t) + B_y^-(\vec{r})M_y(\vec{r},t) + B_z^-(\vec{r})M_z(\vec{r},t)] \tag{30}$$

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.

$$signal \sim \frac{d}{dt}\int d^3r\,[M_x(\vec{r},t) + M_y(\vec{r},t)] \tag{31}$$

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):

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t}M_+(\vec{r},0) \tag{32}$$

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

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\omega_G t}M_+(\vec{r},0) \tag{33}$$

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})}e^{-i\omega_0 t - i\gamma\cdot(x\cdot G_x(t) + y\cdot G_y(t))t}M_+(\vec{r},0) \tag{34}$$

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:

$$signal \sim \frac{d}{dt} \int d^3r [e^{-t/T_2(\vec{r})} e^{-i\omega_0 t - i\omega_G t} M_+(\vec{r},0)] \tag{35}$$

$$signal \sim -i\omega_0 \int d^3r [e^{-t/T_2(\vec{r})} e^{-i\omega_0 t - i\omega_G t} M_+(\vec{r},0)] \tag{36}$$

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

$$signal \sim -i\omega_0 \int d^3r [e^{-t/T_2(\vec{r})} e^{-i\omega_0 t - i\omega_G t} M_+(\vec{r},0)] \tag{37}$$

$$signal \sim \int d^3r [e^{-t/T_2(\vec{r})} e^{-i\omega_G t} M_+(\vec{r},0)] \tag{38}$$

This corresponds to the rotating frame form of the Bloch equations

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})} e^{-i\gamma\cdot(x\cdot G_x(t) + y\cdot G_y(t))t} M_+(\vec{r},0) \tag{39}$$

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

$$M_+(\vec{r},t) = e^{-t/T_2(\vec{r})} e^{-i2\pi\cdot(k_x\cdot x + k_y\cdot y)} M_+(\vec{r},0) \tag{40}$$

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

$$signal \sim \int d^3r M_+(\vec{r},t) \tag{41}$$

$$signal \sim \int d^3r [e^{-t/T_2(\vec{r})} e^{-i2\pi\cdot(k_x\cdot x + k_y\cdot y)} M_+(\vec{r},t_0)] \tag{42}$$

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.