Pulseq MR course

Pulseq in a Browser and Basic Sequences

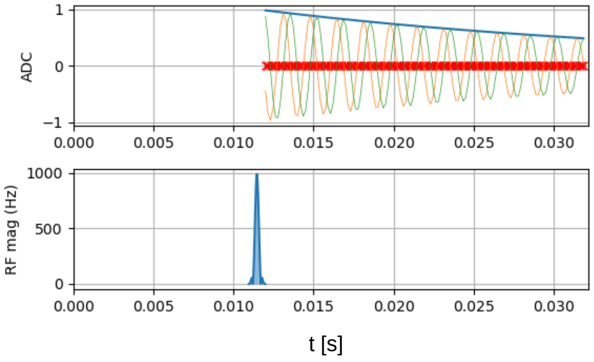
Prof. Dr. Moritz Zaiss
Department for Artificial Intelligence in Biomedical Engineering
University Clinic Erlangen, Friedrich-Alexander University (FAU) Erlangen-Nürnberg

moritz.zaiss@fau.de

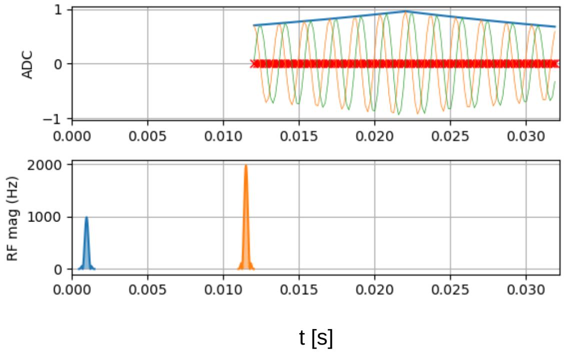
www.mr-physik.med.fau.de/ag-zaiss

Basic Sequences

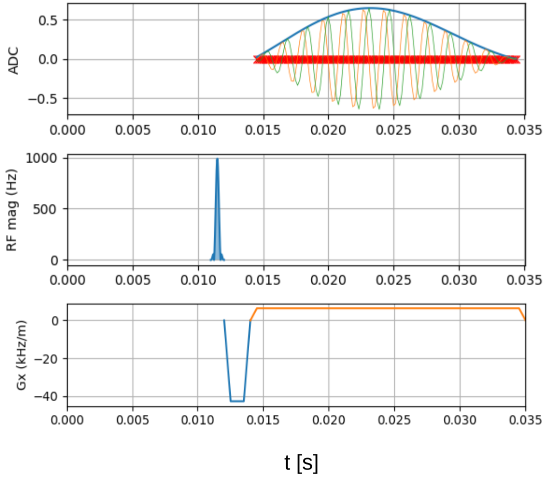
FID



spin echo

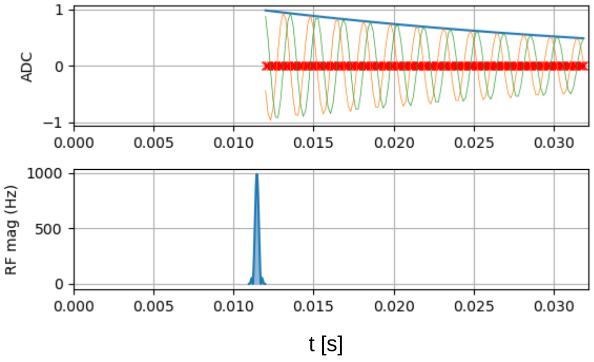


gradient echo

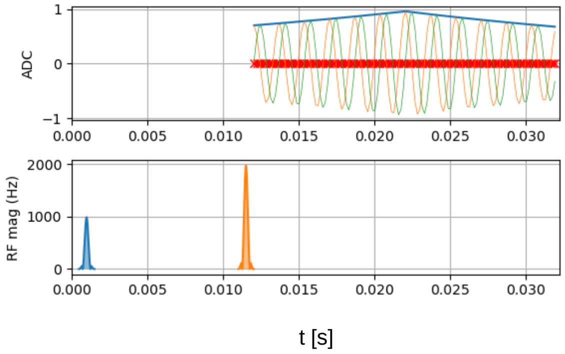


Basic Sequences

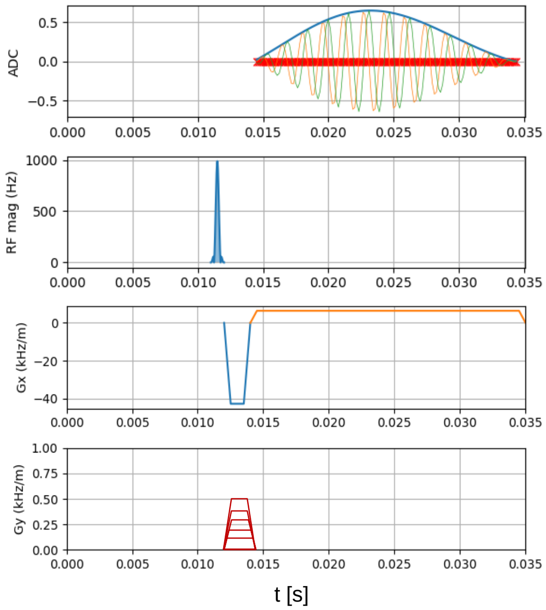
FID



spin echo

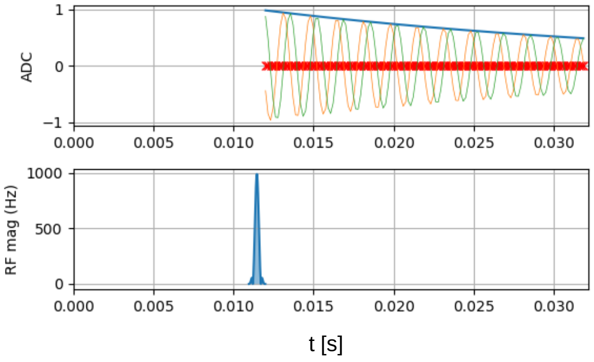


2D gradient echo

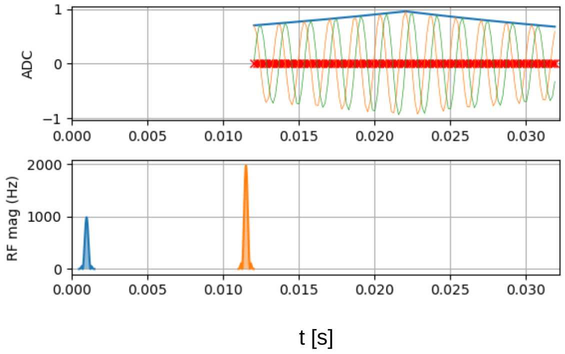


Basic Sequences

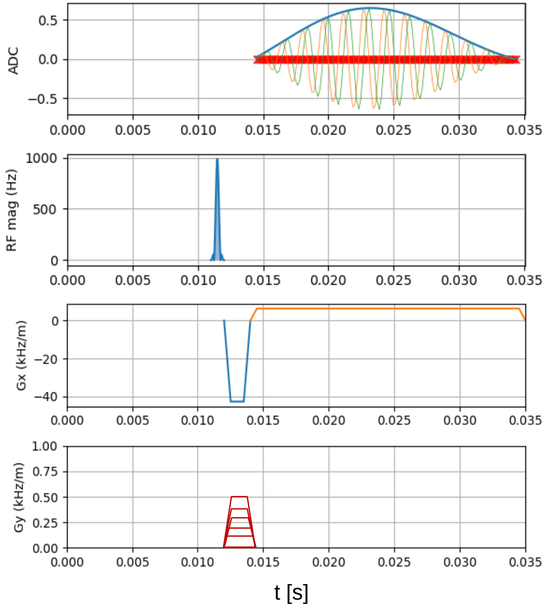
FID



spin echo

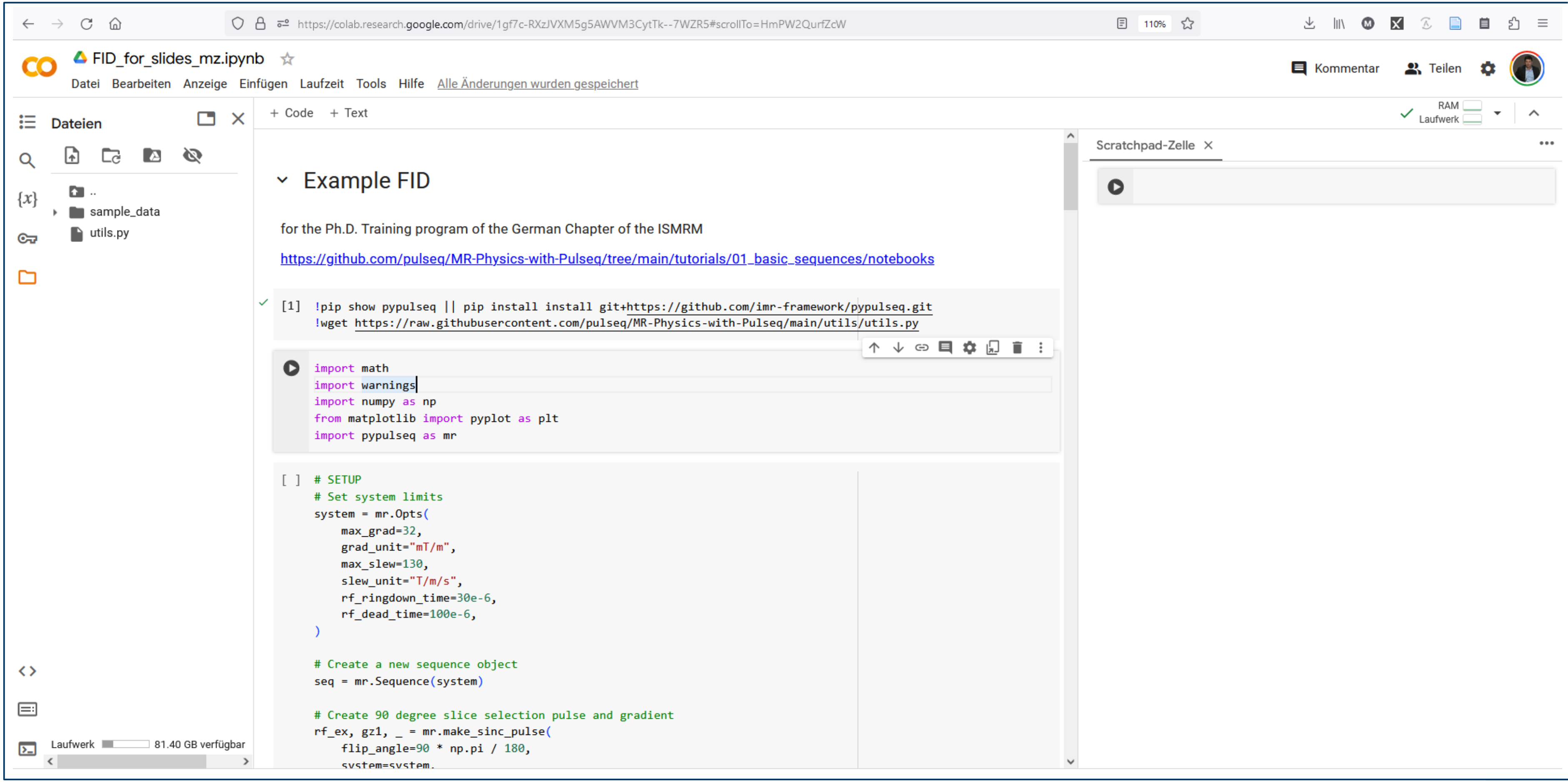


2D gradient echo

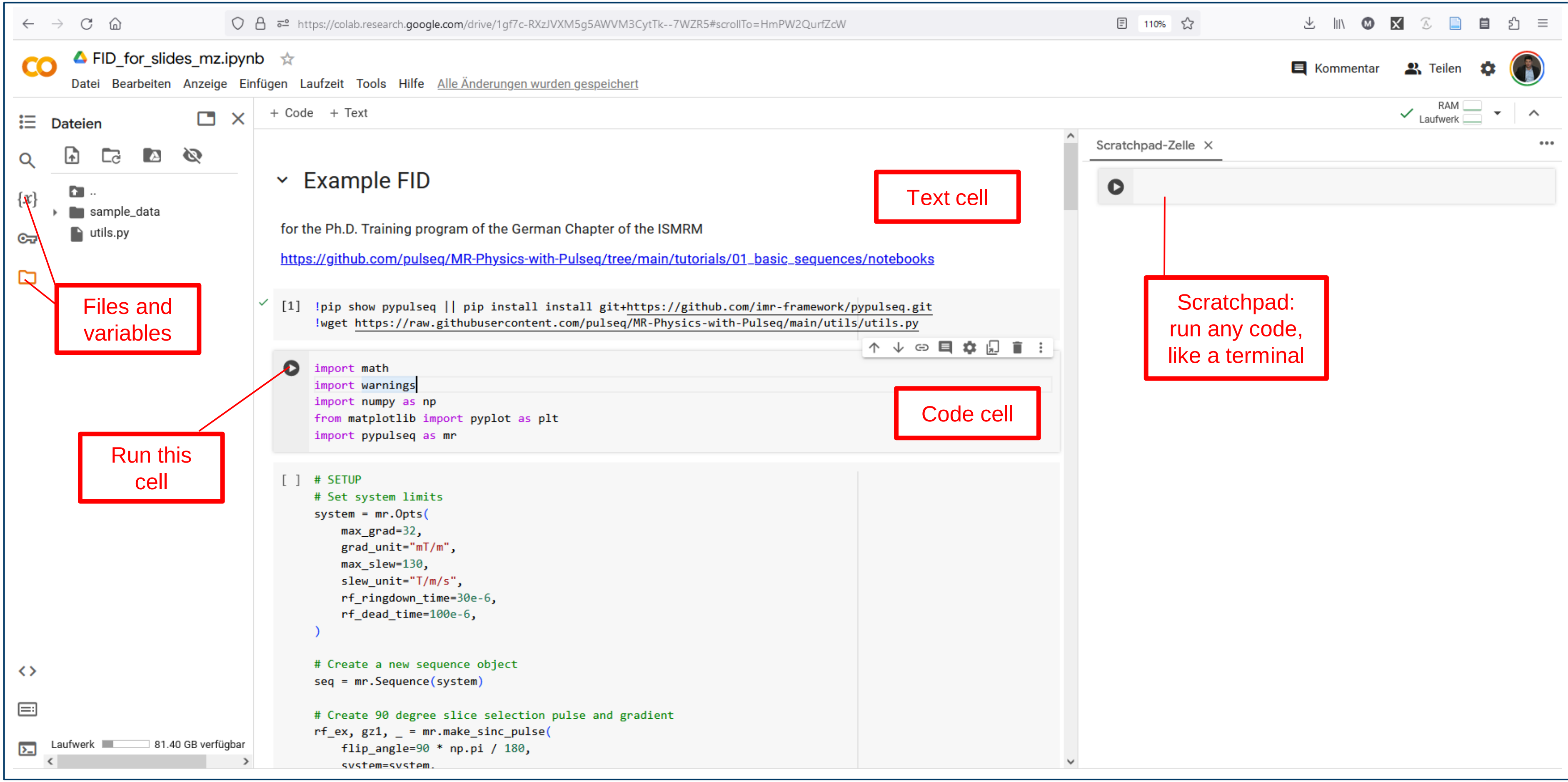


1. We want to create these sequences in Pulseq
2. We want to do it in python using PyPulseq
3. We want to do it in a Browser using
   Google Colab Notebook / Jupyter Notebooks

Google Colab – Jupyter Notebooks



Google Colab – Jupyter Notebooks



Pulseq workflow

```
system = mr.opts('MaxGrad',30,'GradUnit','mT/m',...
    'MaxSlew',170,'SlewUnit','T/m/s');
seq=mr.Sequence(system);

fov = 220e-3; Nx=64; Ny=64; TE = 10e-3; TR = 20e-3;

[rf, gz] = mr.makeSincPulse(15*pi/180,system,'Duration',4e-3,...
    'SliceThickness',5e-3,'apodization',0.5,'timeBwProduct',4);

gx = mr.makeTrapezoid('x',system,'FlatArea',Nx/fov,'FlatTime',6.4e-3);
adc = mr.makeAdc(Nx,'Duration',gx.flatTime,'Delay',gx.riseTime);
gxPre = mr.makeTrapezoid('x',system,'Area',-gx.area/2,'Duration',2e-3);
gzReph = mr.makeTrapezoid('z',system,'Area',-gz.area/2,'Duration',2e-3);
phaseAreas = ((0:Ny-1)-Ny/2)*1/fov;

delayTE = TE - mr.calcDuration(gxPre) - mr.calcDuration(rf)/2 ...
    - mr.calcDuration(gx)/2;
delayTR = TR - mr.calcDuration(gxPre) - mr.calcDuration(rf) ...
    - mr.calcDuration(gx) - delayTE;
delay1 = mr.makeDelay(delayTE);
delay2 = mr.makeDelay(delayTR);

for i=1:Ny
    seq.addBlock(rf,gz);
    gyPre = mr.makeTrapezoid('y',system,'Area',phaseAreas(i),...
                             'Duration',2e-3);
    seq.addBlock(gxPre,gyPre,gzReph);
    seq.addBlock(delay1);
    seq.addBlock(gx,adc);
    seq.addBlock(delay2)
end

seq.write('gre.seq')
```

a runnable gradient echo sequence code (similar to Siemens' example miniFlash)

- Define the system properties
- Define high-level parameters (convenience)
- Define pulses and ADC objects used in the sequence
- Calculate the delays and reordering tables
- Loop and define sequence blocks
- Duration of each block is defined by the duration of the longest event

- *Copy '*.seq' to the scanner and run it!*

Colab - Setup and Define System Properties

```
[1] !pip show pypulseq || pip install install git+https://github.com/imr-framework/pypulseq.git
    !wget https://raw.githubusercontent.com/pulseq/MR-Physics-with-Pulseq/main/utils/utils.py
```

- Install the latest version of pypulseq

```
[ ] import math
    import warnings
    import numpy as np
    from matplotlib import pyplot as plt
    import pypulseq as mr
```

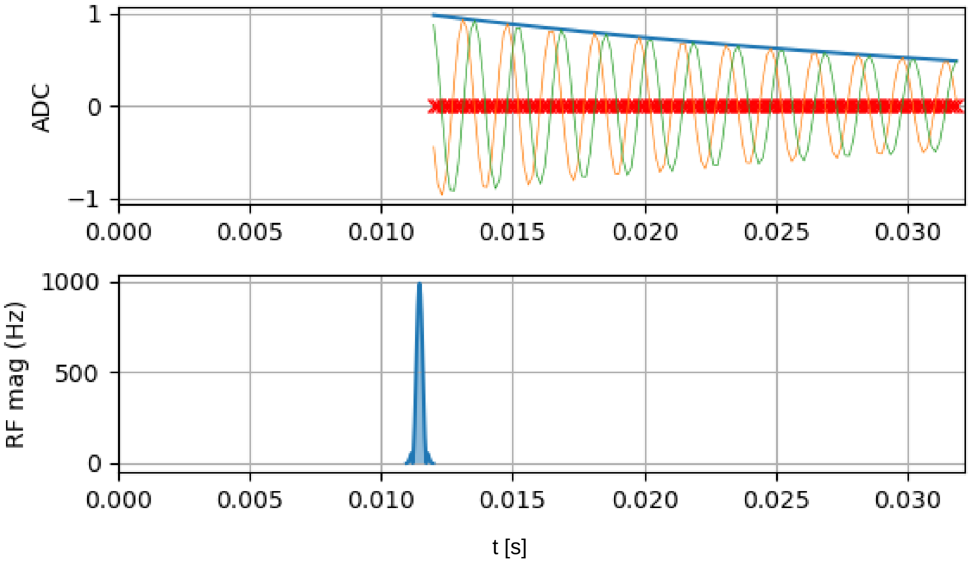
- Import required packages
- pypulseq as mr

```
[ ] # SETUP
    # Set system limits
    system = mr.Opts(
        max_grad=32,
        grad_unit="mT/m",
        max_slew=130,
        slew_unit="T/m/s",
        rf_ringdown_time=30e-6,
        rf_dead_time=100e-6,
    )
```

- Setup and MR system with scanner limits (matching your scanner)

Basic Sequences

FID



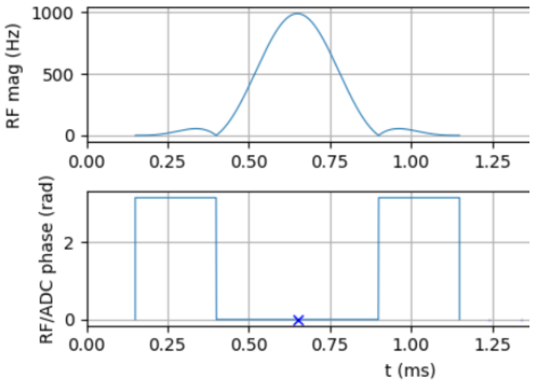
1. RF pulse
2. ADC event

Colab – Define Pulses and ADC objects

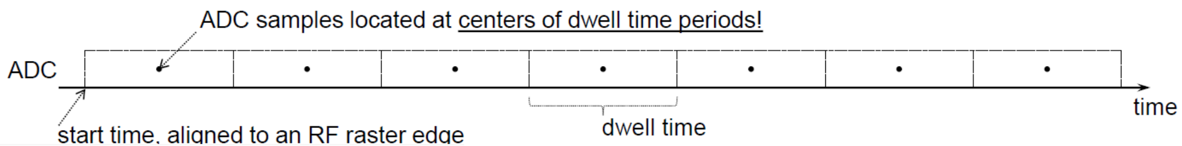
```python
# Create 90 degree slice selection pulse and gradient
rf_ex, gz1, _ = mr.make_sinc_pulse(
    flip_angle=90 * np.pi / 180,
    system=system,
    duration=1e-3,
    slice_thickness=5e-3,
    apodization=0.5,
    time_bw_product=4,
    phase_offset=0* np.pi / 180,
    return_gz=True,
)

dwell=10e-5
Nread=256
Nphase=1
print(f"""The bandwidth is {1/dwell:.2f} Hz
and {1/(dwell*Nread):.2f} Hz/px for the {Nread*dwell*1000:.2f} ms ADC.""")

adc = mr.make_adc(
    num_samples=Nread,
    duration=Nread*dwell,
    phase_offset=0* np.pi / 180,
    delay=1e-5,
    system=system,)
```

```
The bandwidth is 10000.00 Hz
and 39.06 Hz/px for the 25.60 ms ADC.
```

- Make_sinc_pulse



- Make_adc

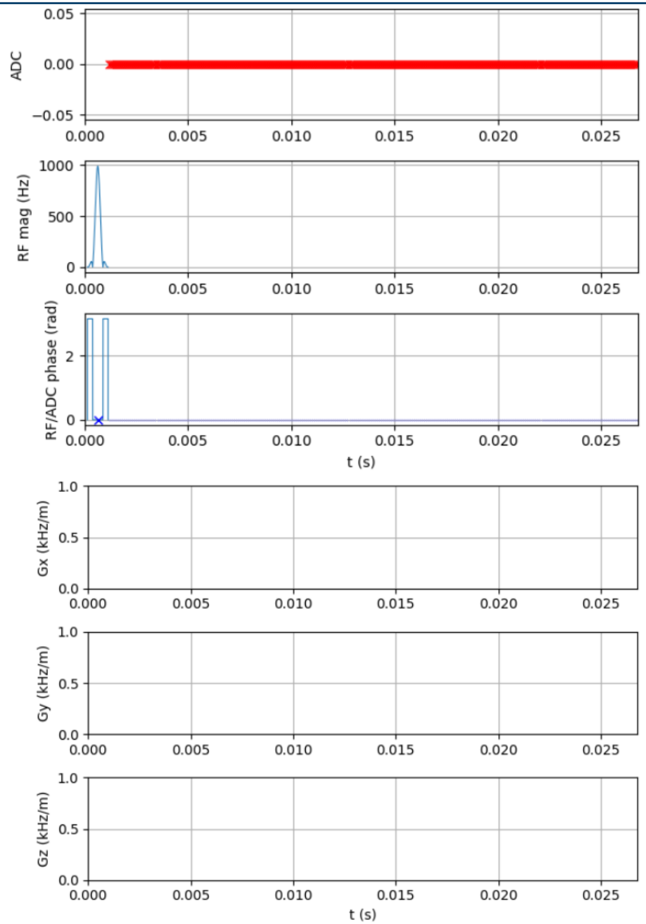


Colab – Construct FID Sequence

```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
seq.add_block(rf_ex)
seq.add_block(adc)
```

plot sequence

```
#@title plot sequence
# plot the entire sequence
seq.plot()
```



Colab – check_timing   if passed: seq.write

0s

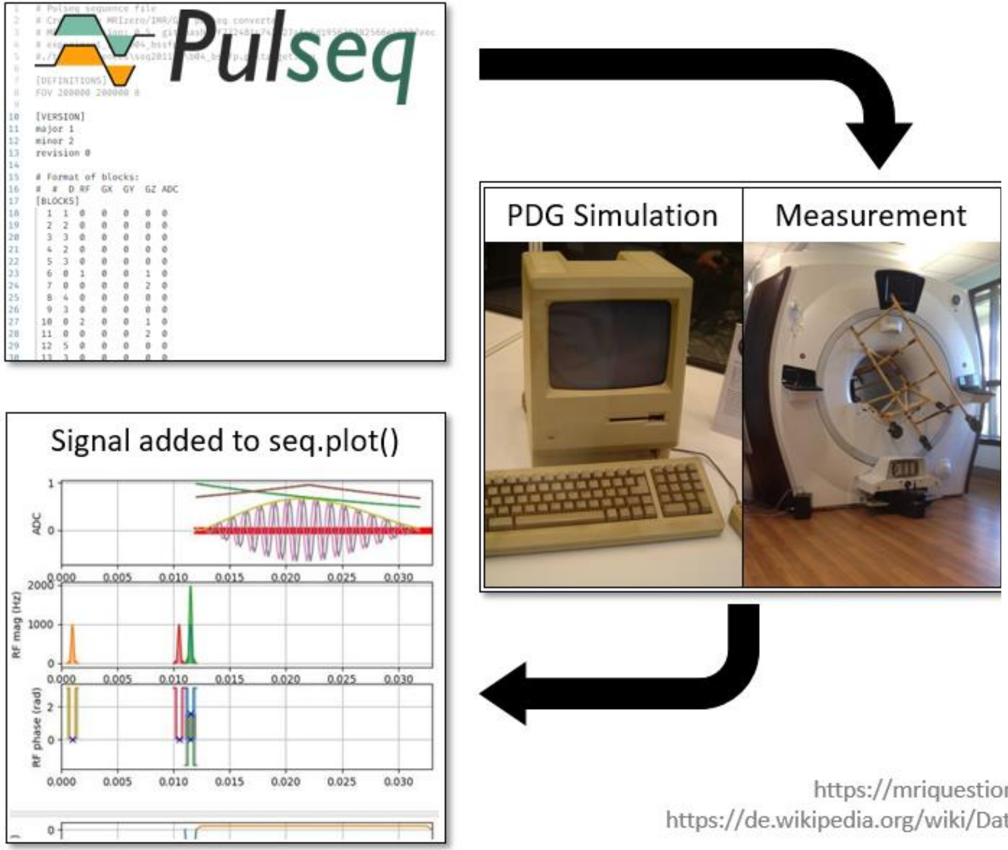
```
# check sequence timing
(ok, error_report) = seq.check_timing()  # Check whether the timing
if ok:
    print("Timing check passed successfully")
else:
    print("Timing check failed. Error listing follows:")
    [print(e) for e in error_report]
```

```
Timing check passed successfully
```

write seq file and export

0s

```
#@title write seq file and export
seq_filename='FID.seq'
seq.write(seq_filename)
from google.colab import files
files.download(seq_filename)  # Download locally
```



https://mriquestions.com/projectiles.html
https://de.wikipedia.org/wiki/Datei:Old_computer_2.jpg

Colab – MR-zero simulation (simple)

simulation setup

```
[2] #@title simulation setup
!pip install MRzeroCore &> /dev/null
!wget https://github.com/MRsources/MRzero-Core/raw/main/documentation/playground_mr0/numerical_brain_cropped.mat
import MRzeroCore as mr0
import utils # several helper functions for simulation and recon
```

simulation (simple)

```
[ ] #@title simulation (simple)
# kscape_adc.shape is [N_coils, N_meas, N_adc]
kspace_adc=utils.simulate_2d(seq, noise_level=0, n_coils=1, dB0=+100, B0_scale=1, B0_polynomial=None)
#sp_adc, t_adc = util.pulseq_plot(seq,signal=kspace_adc) # for pypulseq below dev branch.
seq.plot(plot_now=False)
mr0.util.insert_signal_plot(seq, kspace_adc)
plt.show()
```

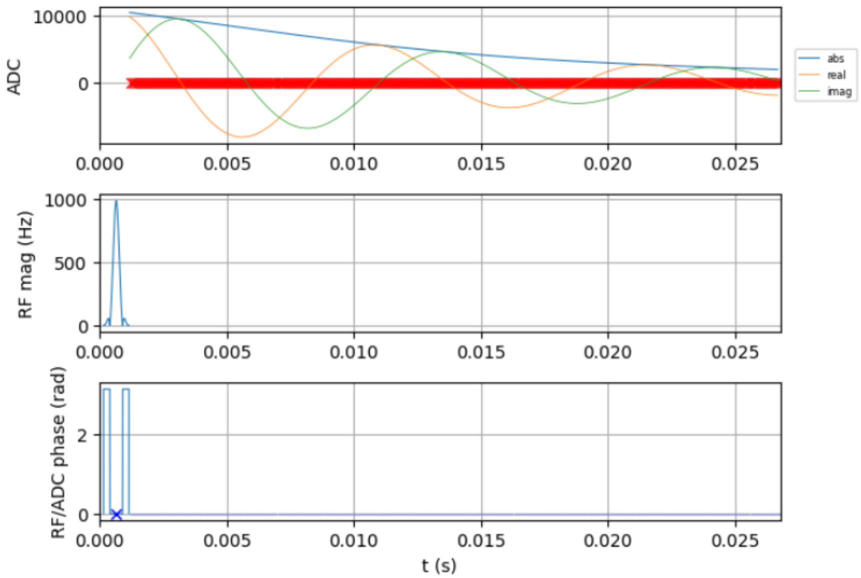
Colab – MR-zero simulation (simple)

simulation setup

```
#@title simulation setup
!pip install MRzeroCore &> /dev/null
!wget https://github.com/MRsources/MRzero-Core/raw/main/documentation/playground_mr0/numerical_brain_cropped.mat
import MRzeroCore as mr0
import utils # several helper functions for simulation and recon
```

simulation (simple)

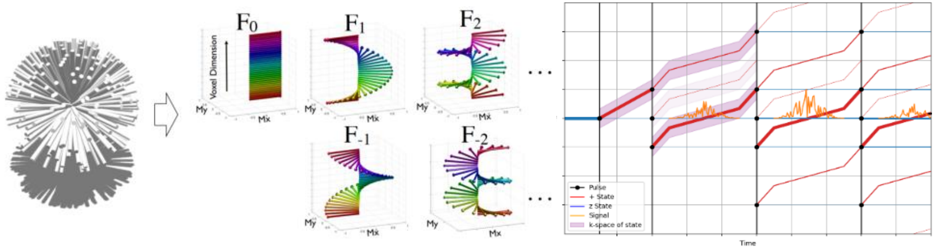
```
[ ] #@title simulation (simple)
# kscape_adc.shape is [N_coils, N_meas, N_adc]
kspace_adc=utils.simulate_2d(seq, noise_level=0, n_coils=1
#sp_adc, t_adc = util.pulseq_plot(seq,signal=kspace_adc) #
seq.plot(plot_now=False)
mr0.util.insert_signal_plot(seq, kspace_adc)
plt.show()
```



Features and Limitations of our simulation

Phase Distribution Graph simulation

- **EPG-based simulation**
  - full echo shape
  - full encoding
  - arbitrary timing



Jonathan Endres

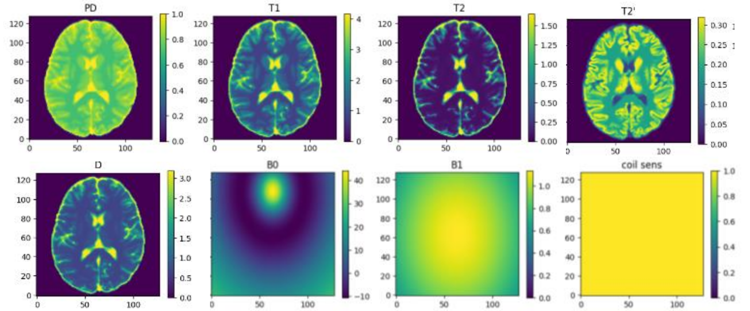
- Instantaneous pulses (center pulse assumed)
  - no slice profile, no rf off-resonance, no SMS ->mr.simRf.m ->Tony Stöcker

- Pulseq standard >1.2 required, tested until 1.4

Endres J, Dang N, Glang F, Loktyushin A, Weinmüller S, Zaiss M.
Phase distribution graphs for differentiable and efficient simulations of arbitrary MRI sequences. In: Proc. ISMRM 30, 2022

Features and Limitations

- PD, T1, T2, T2’, D(isotropic), B0, B1 (all static)
  - compartments possible
- 1D/2D/3D possible
- Differentiable



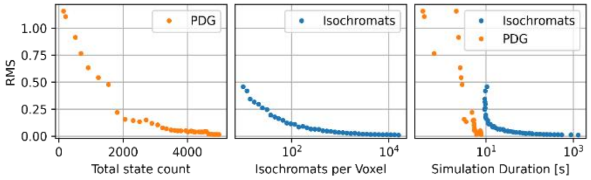
Features and Limitations

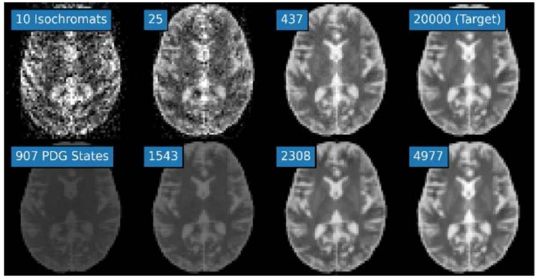
- PD, T1, T2, T2’, D(isotropic), B0, B1 (all static)
    - compartments possible
- 1D/2D/3D possible, mimicking MRS is possible
- Differentiable

- Faster than isochromat solutions

- EPG state analysis possible

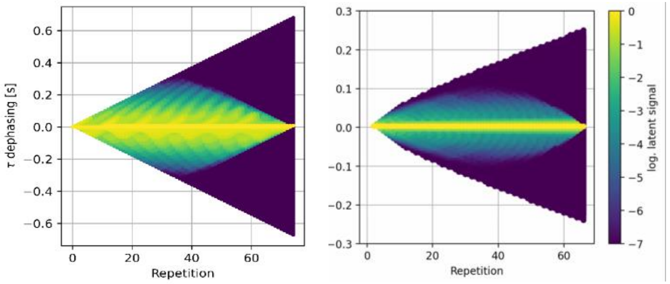
- Recon: Adjoint, FFT, soon: GRAPPA/SENSE





phase graphs: graph.plot()

bSSFP RARE



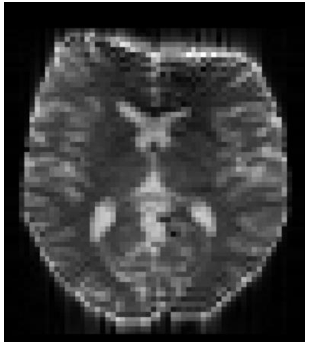
Simulation enables…

… fast feedback to find bugs / new techniques.

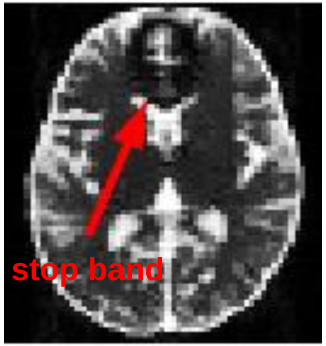
… to solve more mistakes before a real scan.

… to understand artifacts (encoded/non-encoded/recon-related)

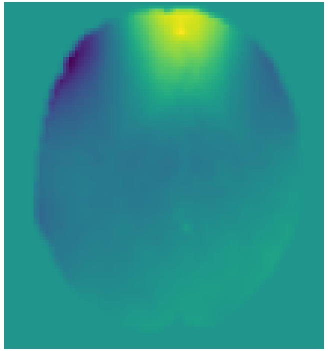
SE-EPI



bSSFP



phantom B0



Colab – MR-zero simulation (advanced)

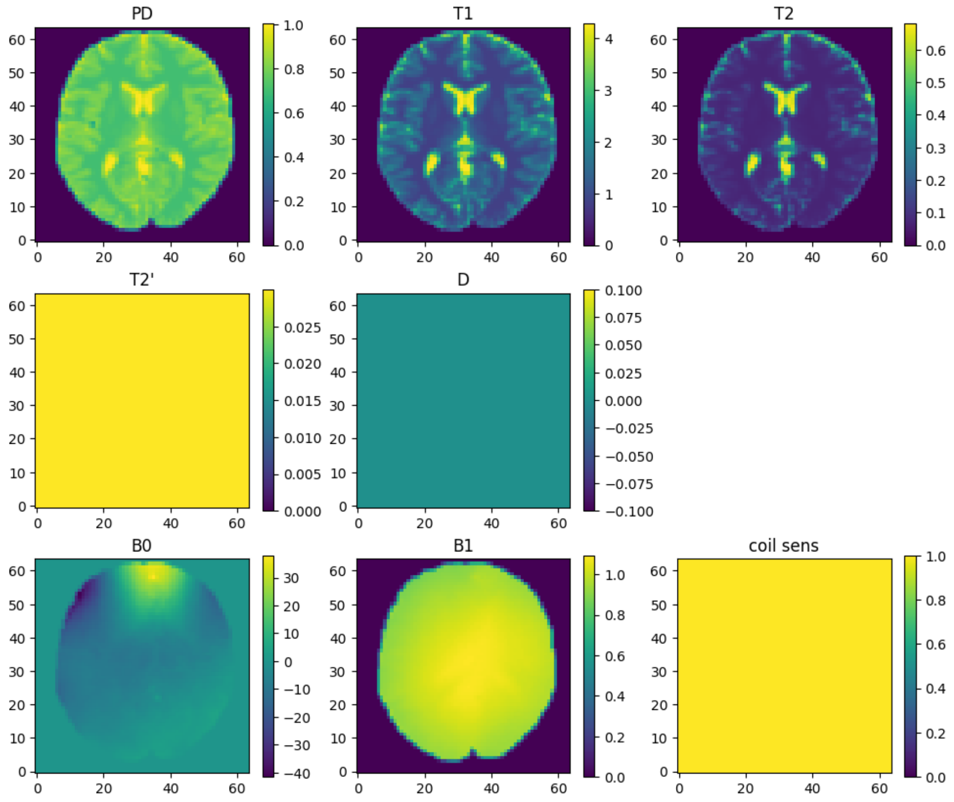
Define object parameters obj_p using brain phantom

simulation (advanced)

```
#@title simulation (advanced)
# %% S4: SETUP SPIN SYSTEM/object on which we can run the seq seq_filename
sz = [64, 64]

# (i) load a phantom object from file
obj_p = mr0.VoxelGridPhantom.load_mat('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 for comparison
PD = obj_p.PD
B0 = obj_p.B0

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

Colab – MR-zero simulation (advanced)

Define object parameters obj_p



Colab – MR-zero simulation (advanced)

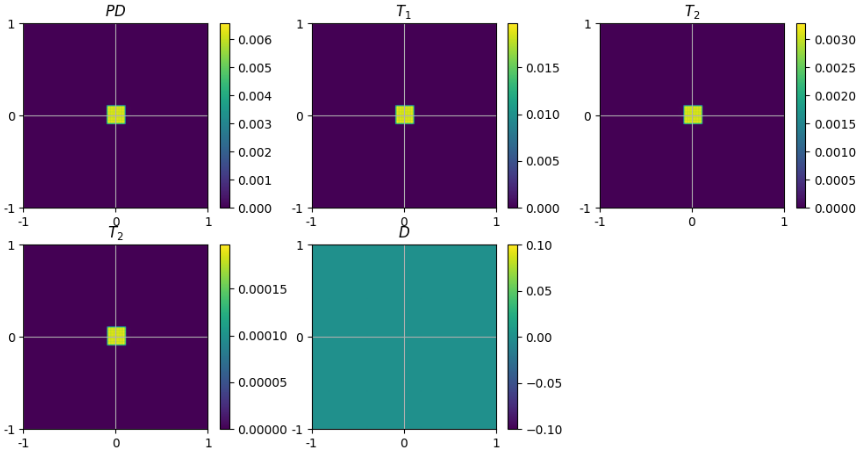
Define object parameters obj_p using a pixel phantom

FID in a pixel phantom - simulation

```
#@title FID in a pixel phantom - simulation
# %% S4: SETUP SPIN SYSTEM/object on which we can run the MR sequenc

#@markdown  The B0 inhomogeneity brings you from the rotating frame
#@markdown Try dB0=0 and dB0=500 for a test.
dB0 = 0 #@param {title:'dB0',type:"slider", min:0, max:500, step:10}

# set phantom  manually to a pixel phantom. Coordinate system is [-0
obj_p = mr0.CustomVoxelPhantom(
        pos=[[0., 0., 0]],
        PD=[1.0],
        T1=[3.0],
        T2=[0.5],
        T2dash=[30e-3],
        D=[0.0],
        B0=0,
        voxel_size=0.1,
        voxel_shape="box"
    )
# Manipulate loaded data
obj_p.B0+=dB0
obj_p.D*=0
obj_p.plot()
# Convert Phantom into simulation data
obj_p=obj_p.build()
```

The B0 inhomogeneity brings you from the rotating frame FID at dB0=0, closer to the lab frame FID at dB0=B0. Try dB0=0 and dB0=500 for a test.

dB0: 0



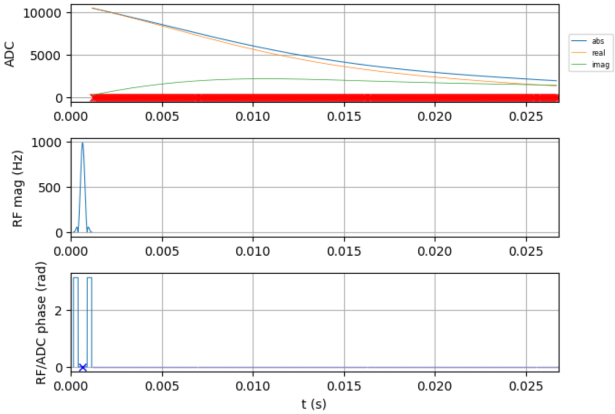
Colab – MR-zero simulation (advanced)

Compute and execute phase distribution graph

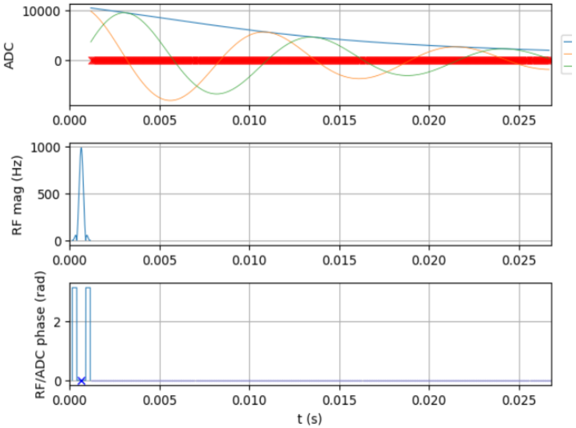
```
# %% S5:. SIMULATE  the external.seq file and add acquired signal to ADC plot

# Read in the sequence
seq0 = mr0.Sequence.import_file(seq_filename)
seq0.plot_kspace_trajectory()
# Simulate the sequence
graph = mr0.compute_graph(seq0, obj_p, max_state_count=1000, min_state_mag=1e-5)
signal = mr0.execute_graph(graph, seq0, obj_p)

# PLOT sequence with signal in the ADC subplot
#sp_adc, t_adc = util.pulseq_plot(seq,signal=kspace_adc) # for pypulseq below dev branch.
seq.plot(plot_now=False)
mr0.util.insert_signal_plot(seq, signal)
plt.show()
```

Play with the simulation!

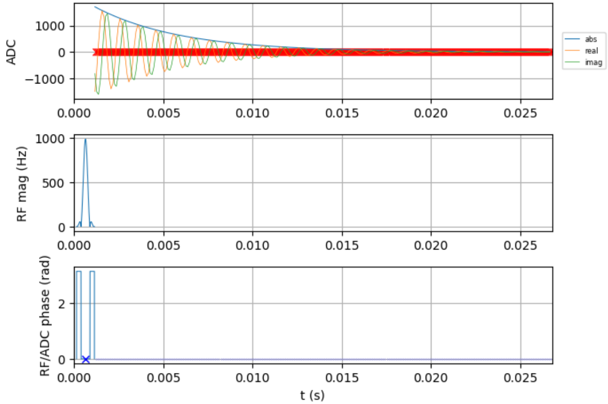
dB0 10 Hz



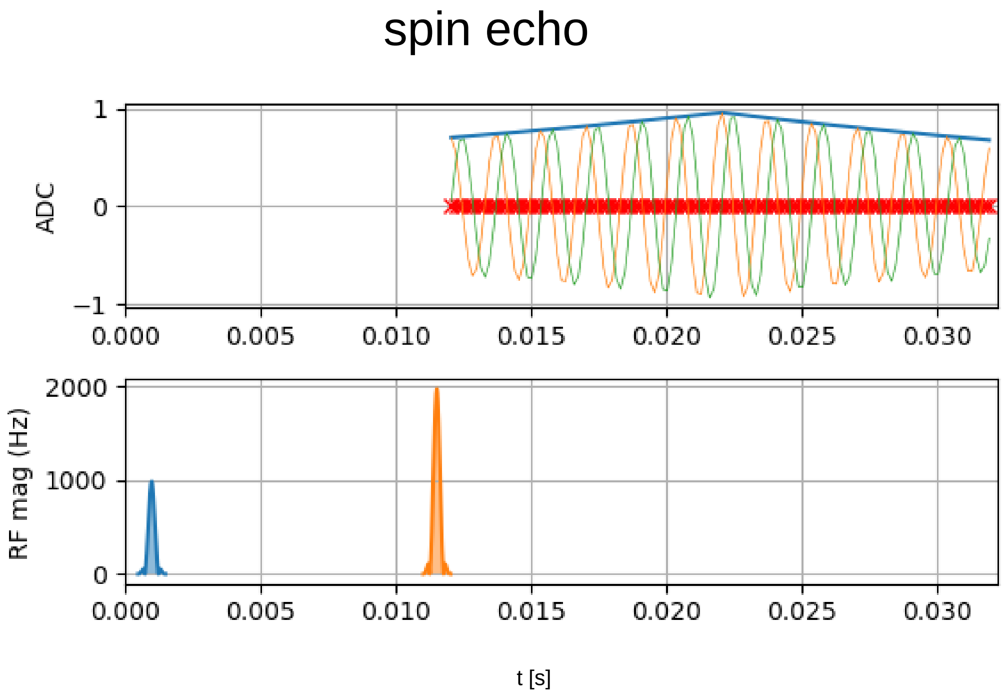
dB0 100 Hz



dB0 1000 Hz, short T2*



Basic Sequences – spin echo



1. Two RF pulses
2. ADC event
3. Accurate timing!

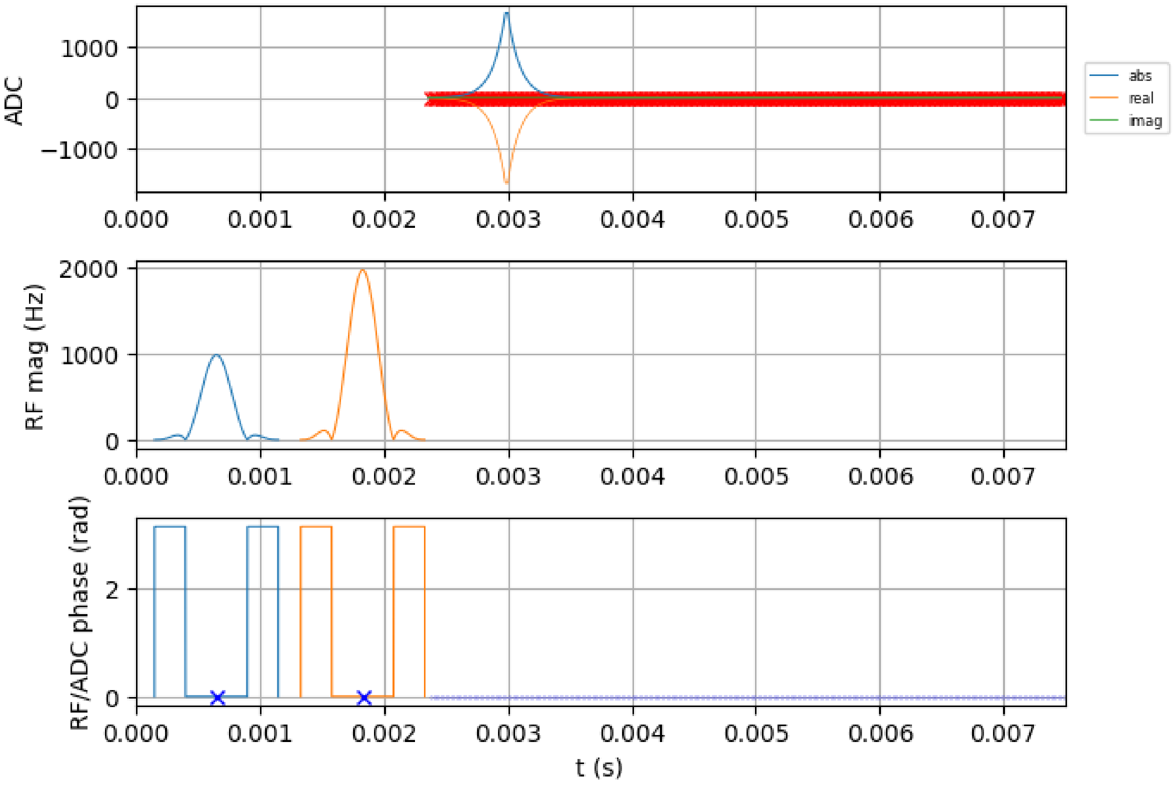
Basic Sequences – spin echo

```
# Create 180 degree refocusing pulse
rf_ref, _, _ = mr.make_sinc_pulse(
    flip_angle=180 * np.pi / 180,
    system=system,
    duration=1e-3,
    slice_thickness=5e-3,
    apodization=0.5,
    time_bw_product=4,
    phase_offset=0* np.pi / 180,
    return_gz=True,
)
```

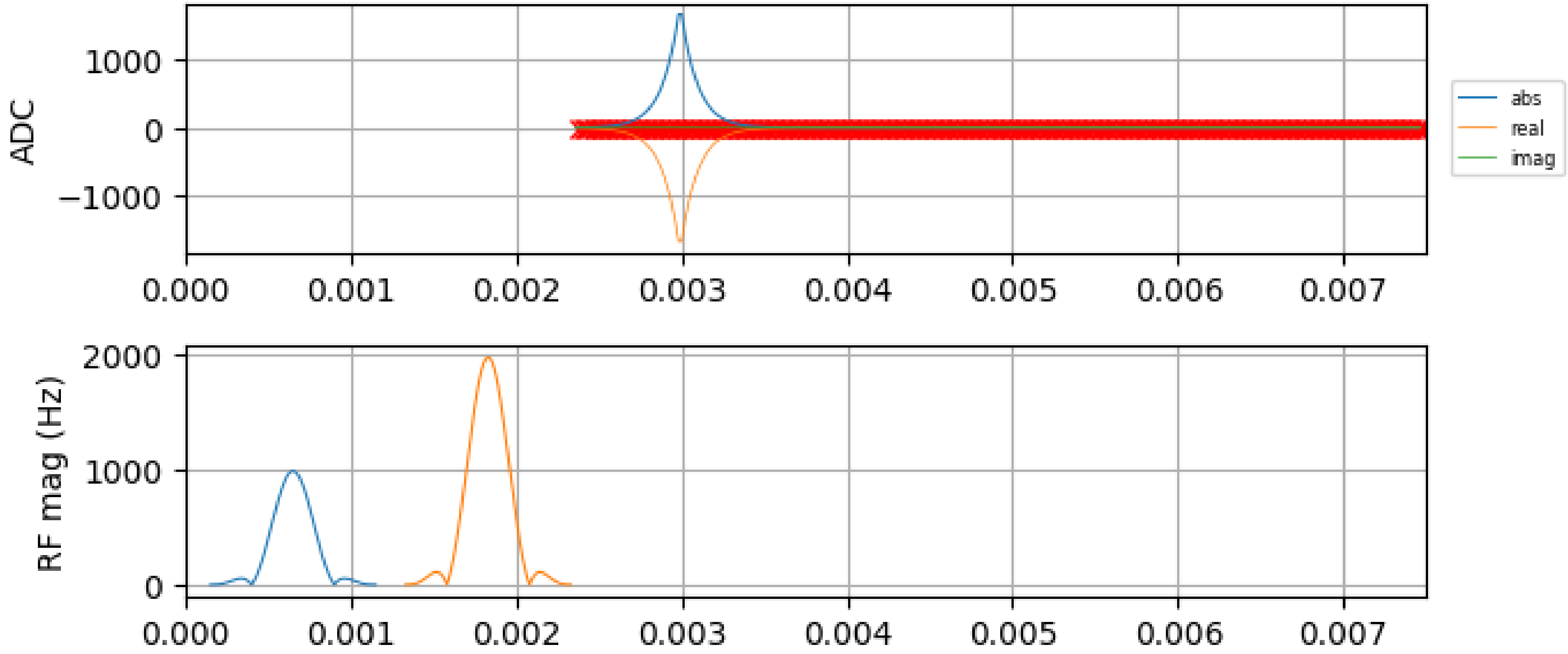
Basic Sequences – spin echo

0 s

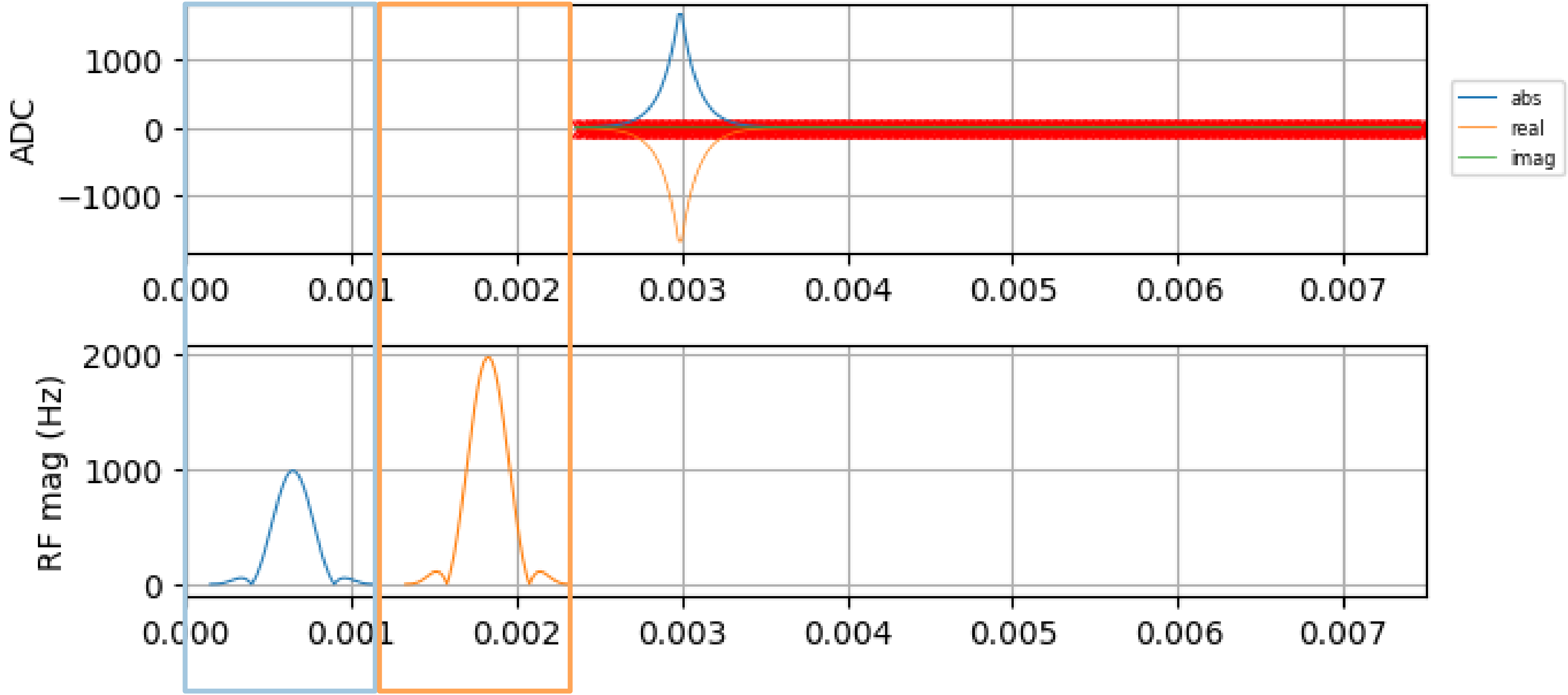
```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
seq.add_block(rf_ex)
seq.add_block(rf_ref)
seq.add_block(adc)
```



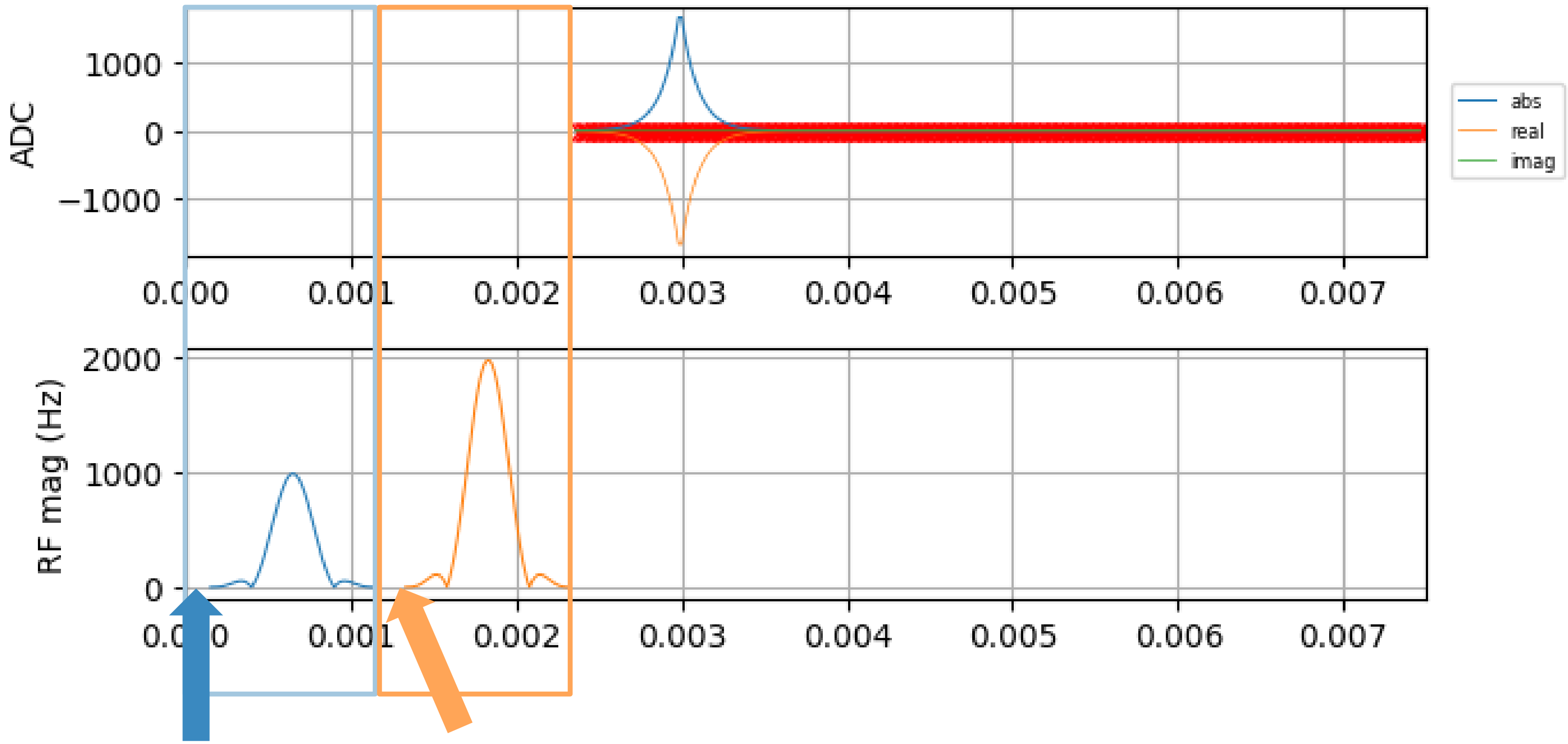
Basic Sequences – spin echo



Basic Sequences – spin echo

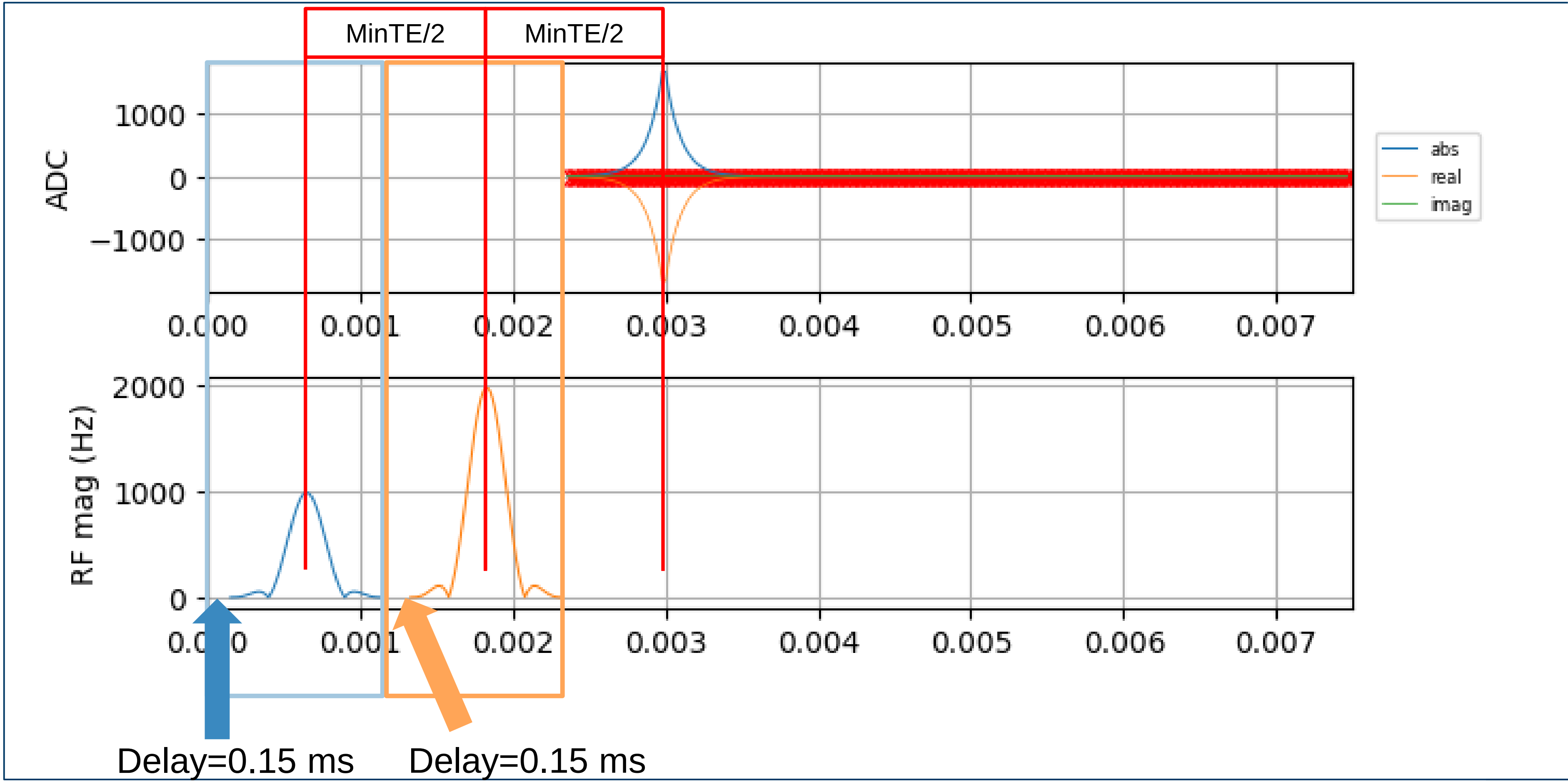


Basic Sequences – spin echo

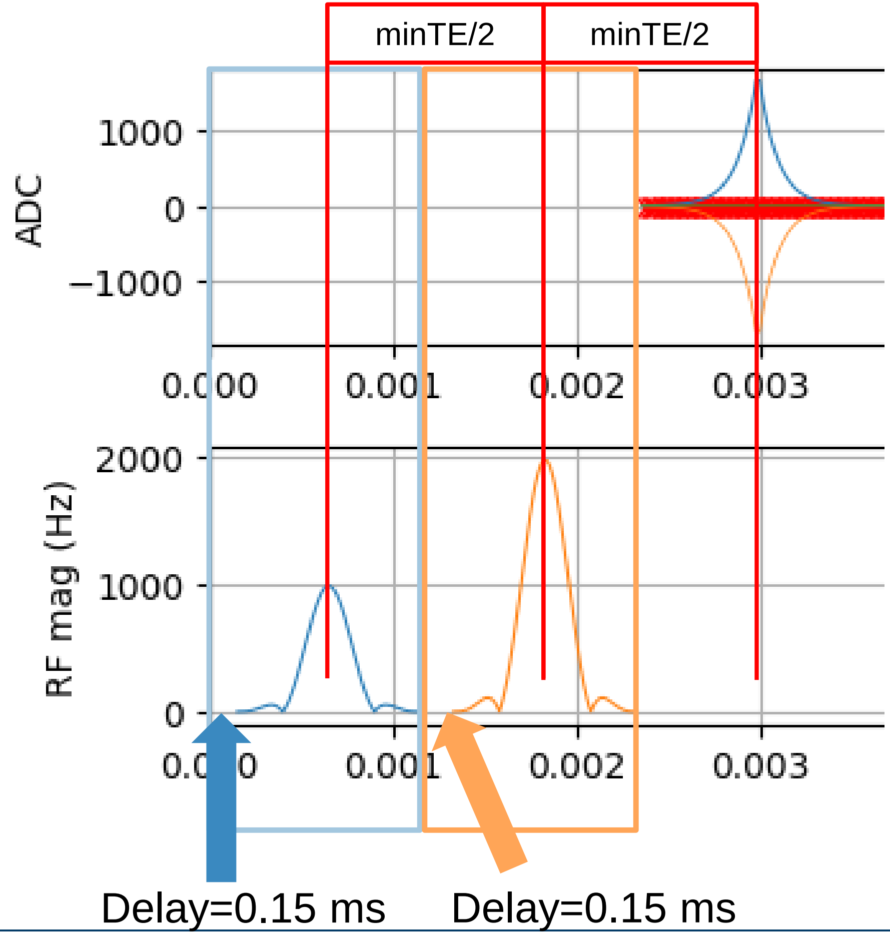


Delay=0.15 ms Delay=0.15 ms

Basic Sequences – spin echo



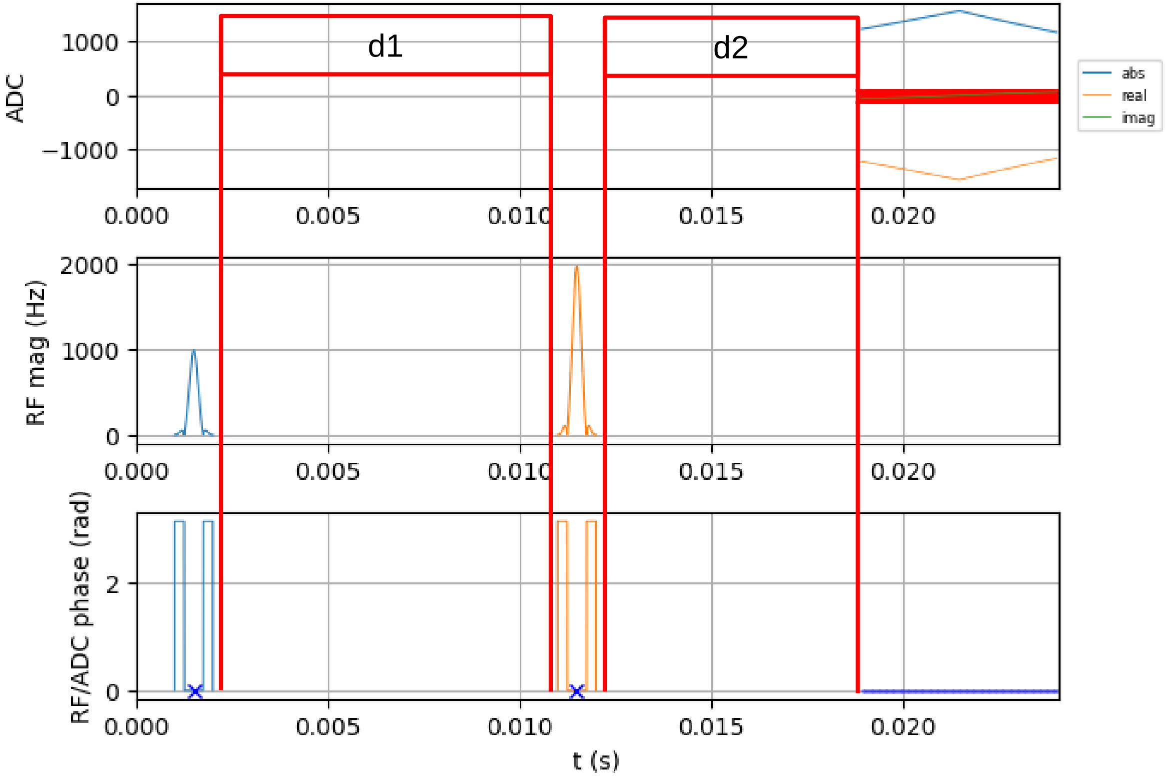
Basic Sequences – spin echo




```
ct_ref=mr.calc_rf_center(rf_ref)   # rfex center time
ct_ref=mr.calc_rf_center(rf_ref)   # rfref center time


minTE2 = ct_ex[0]+ ct_ref[0] + rf_ref.delay
           + rf_ex.ringdown_time

       = 0.5 ms  + 0.5 ms + 0.15 ms
           + 30µs

       = 1.18 ms
```

```
# Create 180 degree refocusing pulse
rf_ref, _, _ = mr.make_sinc_pulse(
    flip_angle=180 * np.pi / 180,
    system=system,
    duration=1e-3,
```

Basic Sequences – spin echo

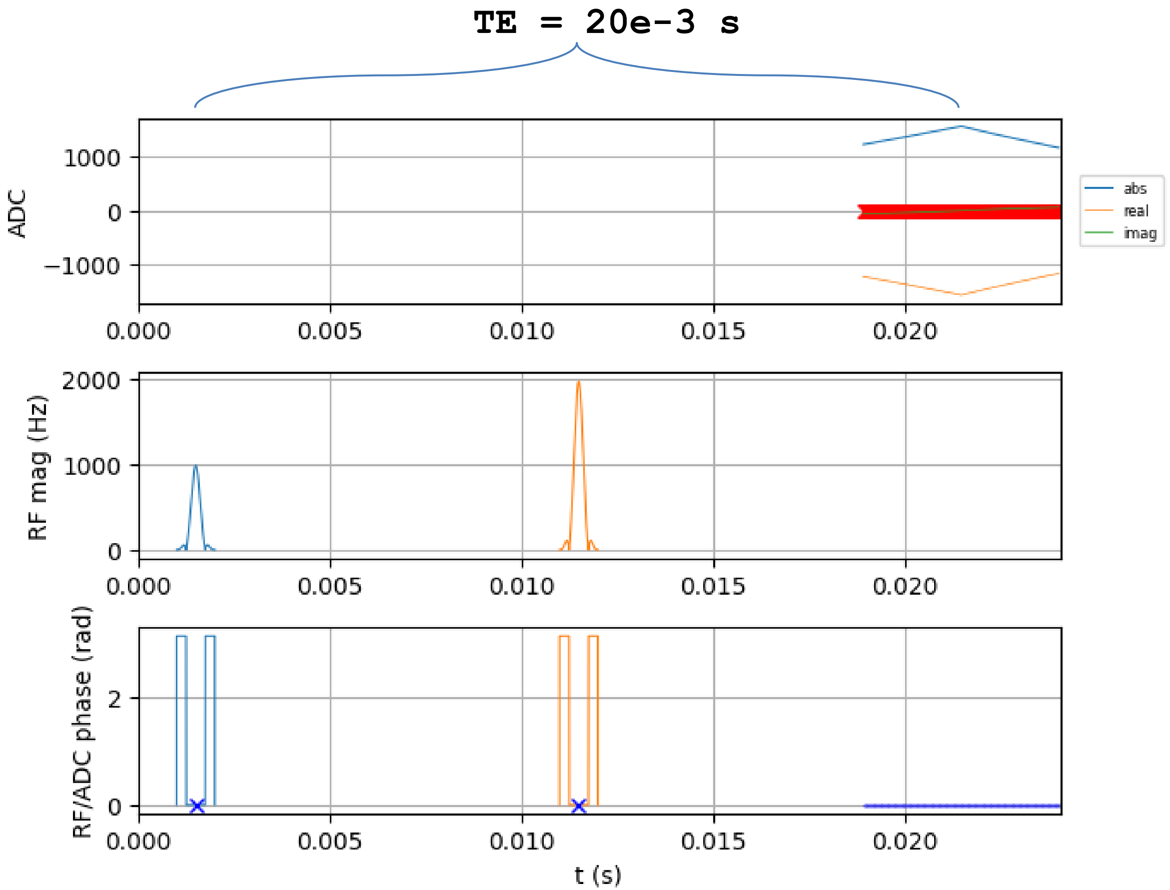



```
d2 = minTE2+d1  #TE/2
-ct_ref[0] -rf_ex.ringdown_time
-adc.delay -Nread*dwell/2
```

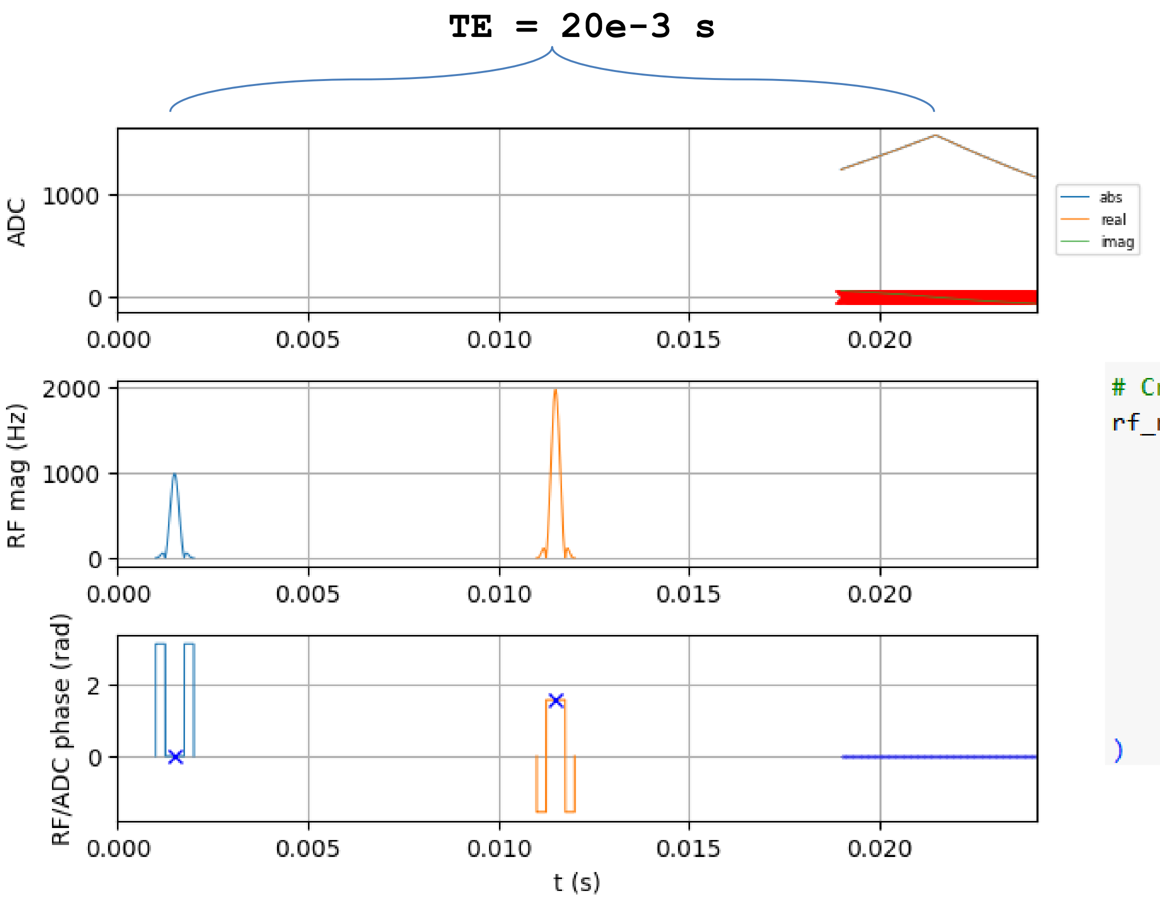
```
TE = 20e-3
```

```
d1 = TE/2-minTE2
```

Basic Sequences – spin echo



Basic Sequences – spin echo




```python
# Create 180 degree refocusing pulse
rf_ref, _, _ = mr.make_sinc_pulse(
    flip_angle=180 * np.pi / 180,
    system=system,
    duration=1e-3,
    slice_thickness=5e-3,
    apodization=0.5,
    time_bw_product=4,
    phase_offset=90* np.pi / 180,
    return_gz=True,
)
```

Basic Sequences – spin echo

Calculate delays

```
#@title Calculate delays
ct_ex=mr.calc_rf_center(rf_ex)    # rf center time returns time and index of the center of the pulse
print(ct_ex[0])
ct_ref=mr.calc_rf_center(rf_ex)    # rf center time returns time and index of the center of the pulse
print(ct_ref[0])
print(mr.calc_duration(rf_ex))
print(rf_ex.delay)
print(rf_ex.ringdown_time)

TE=20e-3  # wanted echo time
minTE2 = ct_ex[0]+rf_ex.ringdown_time + rf_ref.delay+ ct_ref[0]   # echo top to echo top

d1=TE/2-minTE2
      # TE/2   - half ref pulse                 - half adc      -adc.delay
d2 = minTE2+d1 - ct_ref[0]+rf_ex.ringdown_time  - Nread*dwell/2 -adc.delay
```

```
0.0005
0.0005
0.00118
0.00015000000000000001
3e-05
```

Basic Sequences – spin echo

Naïve spin echo

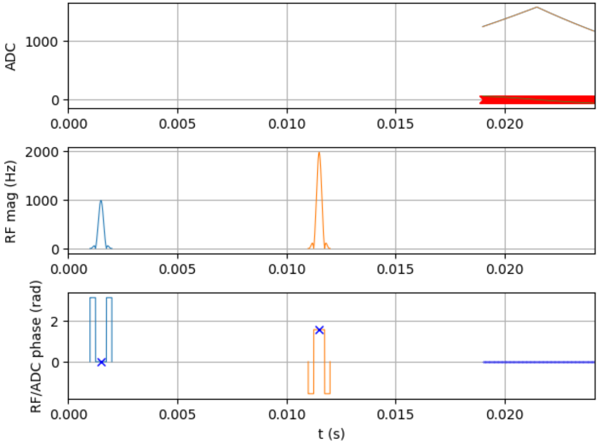
```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
# sequence programming
seq.add_block(rf_ex)
seq.add_block(rf_ref)
seq.add_block(adc)
```

Correct spin echo

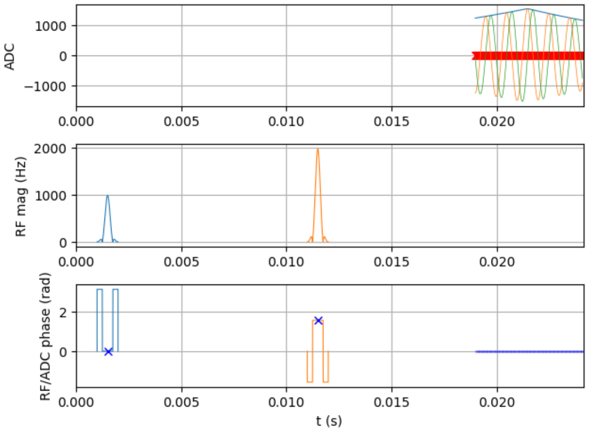
```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
# sequence programming
seq.add_block(rf_ex)
seq.add_block(mr.make_delay(d1))
seq.add_block(rf_ref)
seq.add_block(mr.make_delay(d2))
seq.add_block(adc)
```

Basic Sequences – spin echo - let‘s play

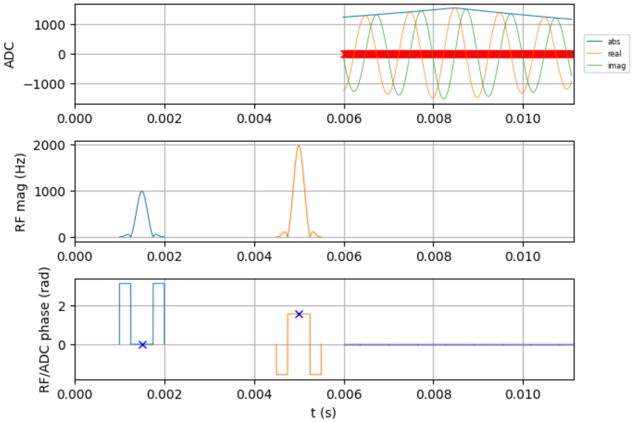
dB0=10 Hz



dB0=1000 Hz

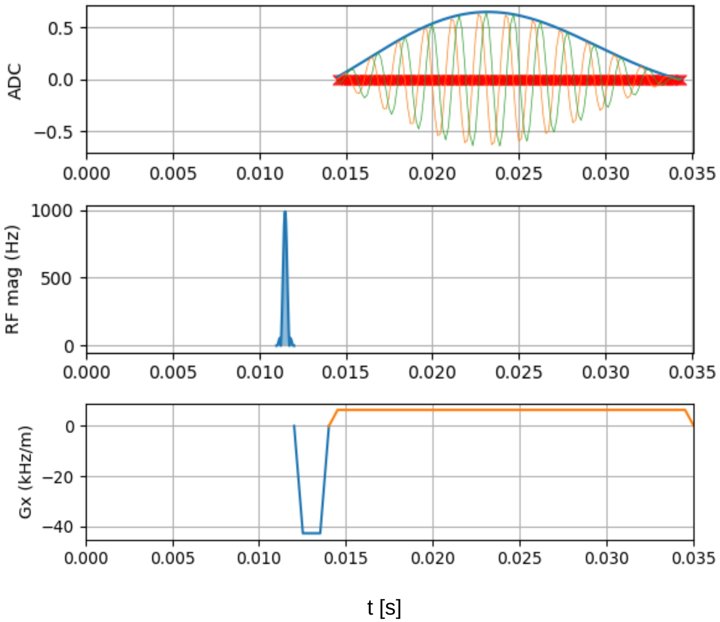


minTE + ADC/2



Basic Sequences - GRE

gradient echo



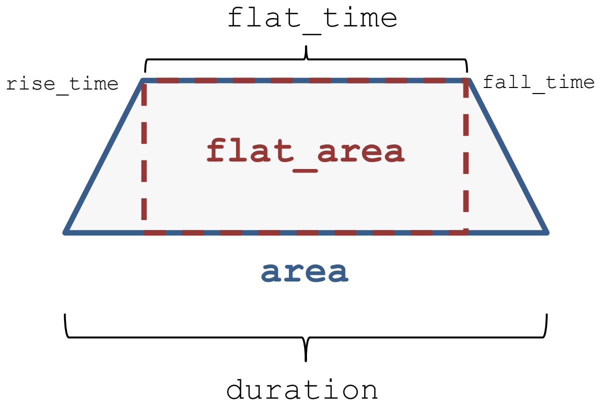
1. Rf pulse
2. ADC
3. Readout gradient
4. Rewinder gradient

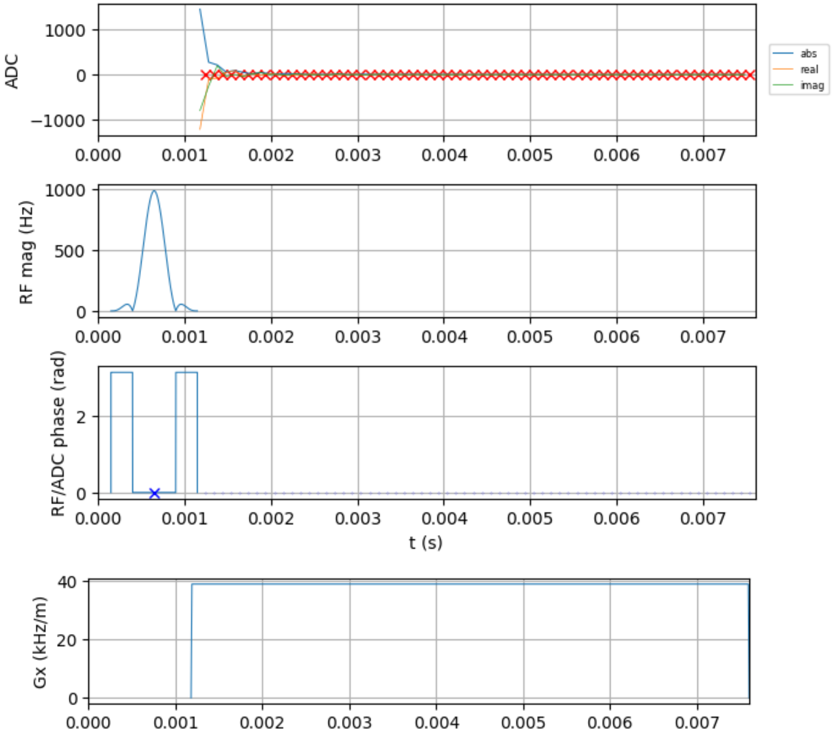
Basic Sequences - GRE

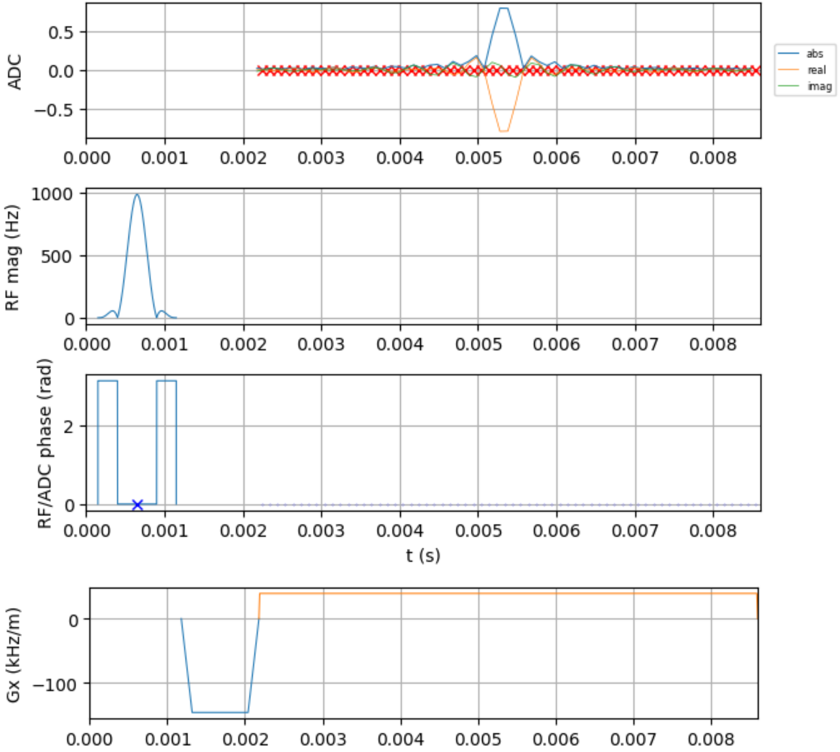
```
# define high level parameters

fov=256e-3
dwell=10e-5
Nread=64
Nphase=1
```

```
gx =mr.make_trapezoid(channel='x',flat_area=Nread/fov,flat_time=Nread*dwell)
```

```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
seq.add_block(rf_ex)
adc.delay=gx.rise_time
seq.add_block(adc,gx)
```




ADC
abs
real
imag
RF mag (Hz)
RF/ADC phase (rad)
t (s)
Gx (kHz/m)

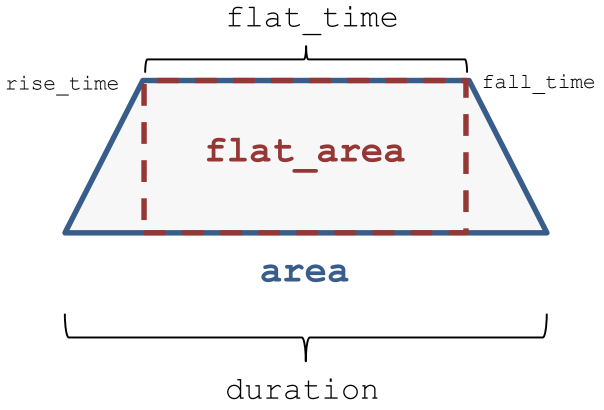

ADC
abs
real
imag
RF mag (Hz)
RF/ADC phase (rad)
t (s)
Gx (kHz/m)

Basic Sequences - GRE

```
# define high level parameters

fov=256e-3
dwell=10e-5
Nread=64
Nphase=1
```

```
gx =mr.make trapezoid(channel='x',flat area=Nread/fov,flat time=Nread*dwell)
gx_pre =mr.make_trapezoid(channel='x', area=-gx.area/2,duration=1e-3)
```

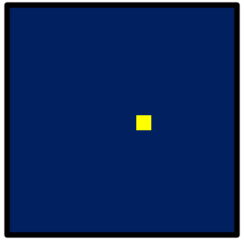
```
# CONSTRUCT SEQUENCE
# Create a new sequence object
seq = mr.Sequence(system)
seq.add_block(gx_pre)
seq.add_block(rf_ex)
adc.delay=gx.rise_time
seq.add_block(adc,gx)
```

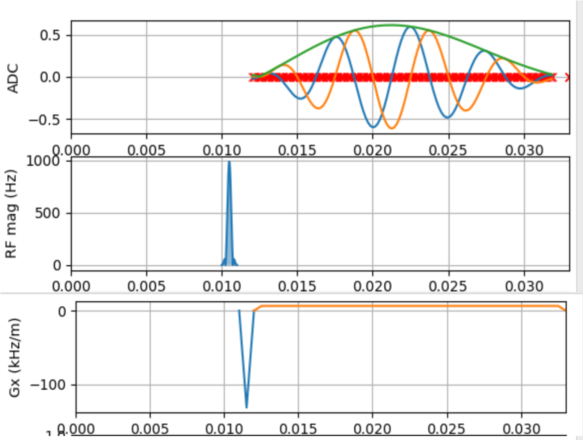


Basic Sequences – 1D GRE

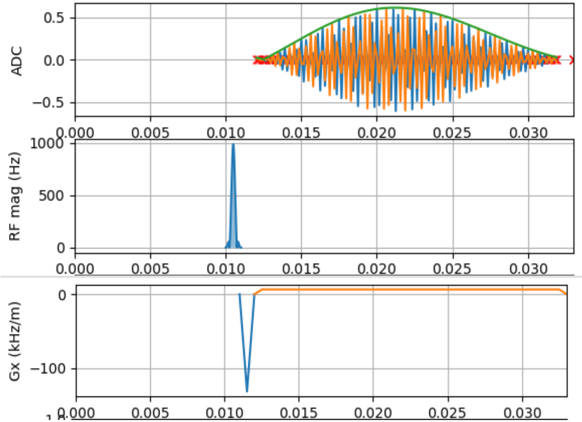
The gradient echo yields spatial encoding:

„Pixels at the edge generate fast oscillations“

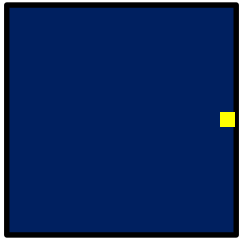




t [s]



t [s]



Coffee Break!