

From 50mT to 7T: *Pulseq* Live

Across Hardware Platforms and Field Strengths

Maxim Zaitsev¹, Thomas O'Reilly², Benjamin Menkuec³, Marcus Prier⁴, David Schote⁴,
Jason Stockmann^{5,6}, Lincoln Craven-Brightman^{5,6}, Melissa Haskell^{7,8}, Jon-Fredrik Nielsen⁸

1. *High Field Magnetic Resonance Center, Center for Medical Physics and Biomedical Engineering, Medical University of Vienna, Vienna, Austria*
2. *Department of Radiology, Leiden University Medical Center, Leiden, Netherlands*
3. *University of Applied Sciences and Arts Dortmund, Dortmund, Germany*
4. *Otto von Guericke Universität Magdeburg, Magdeburg, Germany*
5. *Department of Radiology, Massachusetts General Hospital, Charlestown, MA, USA*
6. *Harvard Medical School, Boston, MA, USA*
7. *Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA*
8. *fMRI laboratory and Biomedical Engineering, University of Michigan, Ann Arbor, MI, USA.*



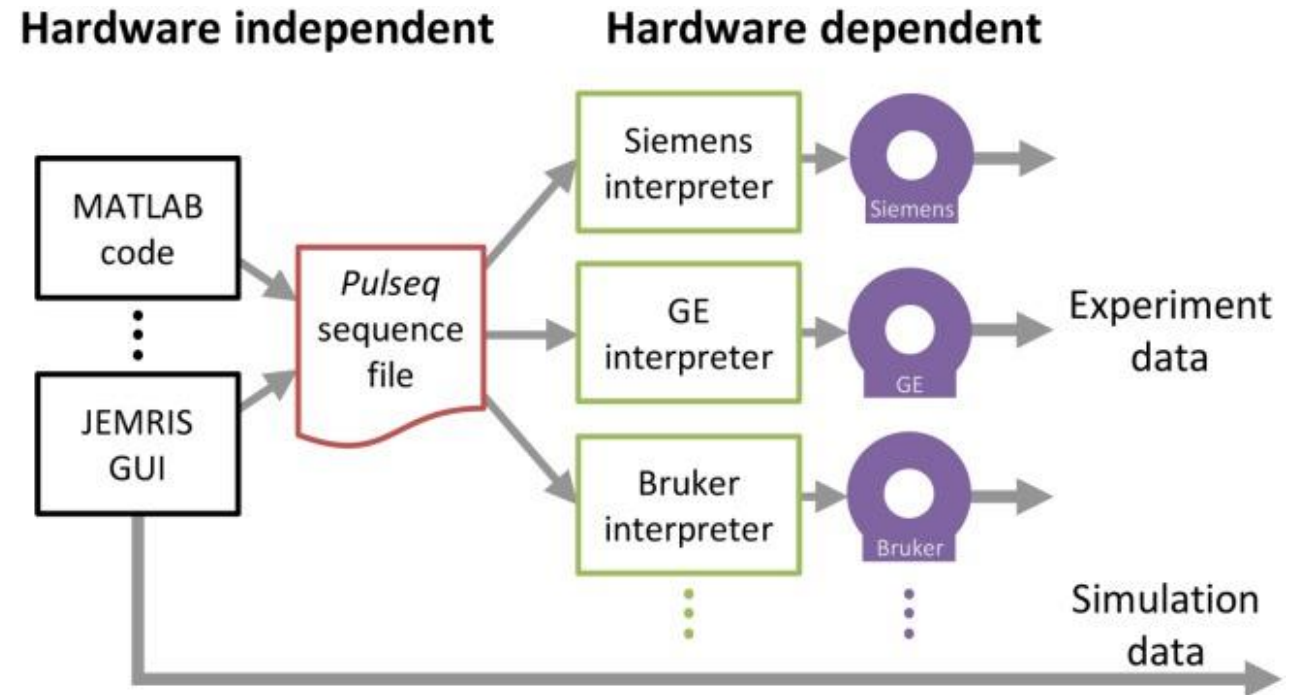
<http://pulseq.github.io>

Outline

- **Pulseseq** sequence definition and programming language overview
- Using **Pulseseq** on different types of scanners:
 - Siemens, GE, and tabletop scanners using OCRA
- Live demo of **Pulseseq** across sites
 1. Basic pulse-acquire scans: Free Induction Decay (FID), Spin-Echo (SE)
 2. Cartesian spin-echo imaging
 3. Radial imaging (spin-echo and gradient echo)
 4. Advanced topics (gradient delays and correction, etc...)

Pulseseq sequence programming

- Cross-platform pulse programming framework for MRI
- Low level: *Pulseseq* file
- High level: MATLAB¹ or Python² toolboxes

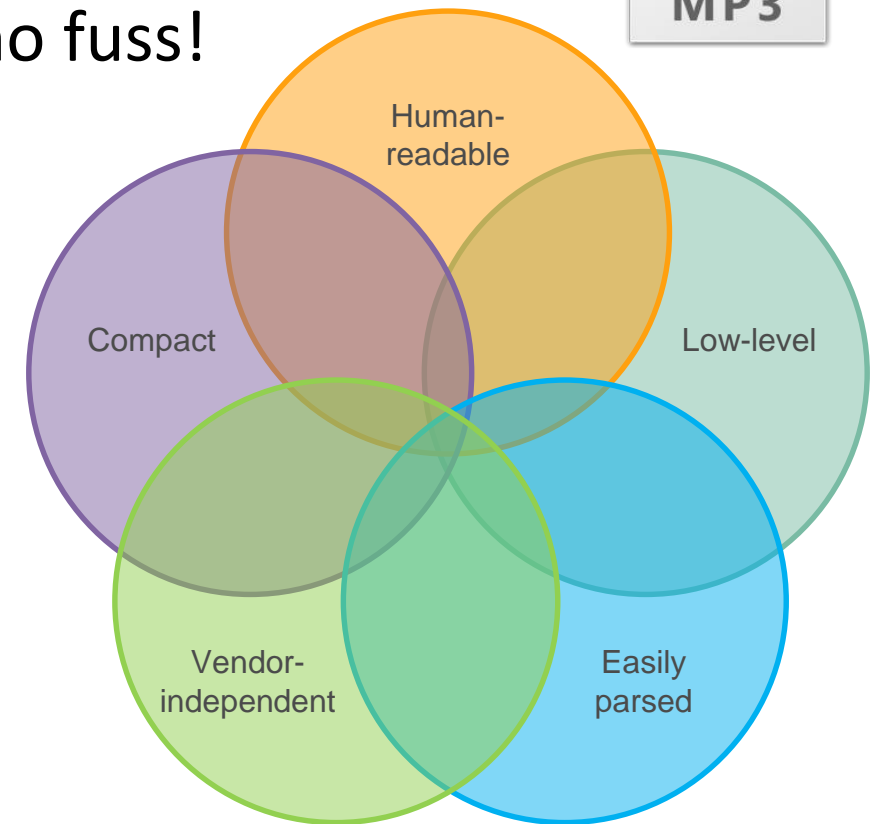


Source: Layton, et al., “Pulseseq: A rapid and hardware-independent pulse sequence prototyping framework”, MRM 2017

1. <http://pulseseq.github.io>
2. <http://github.com/imr-framework/pypulseseq>

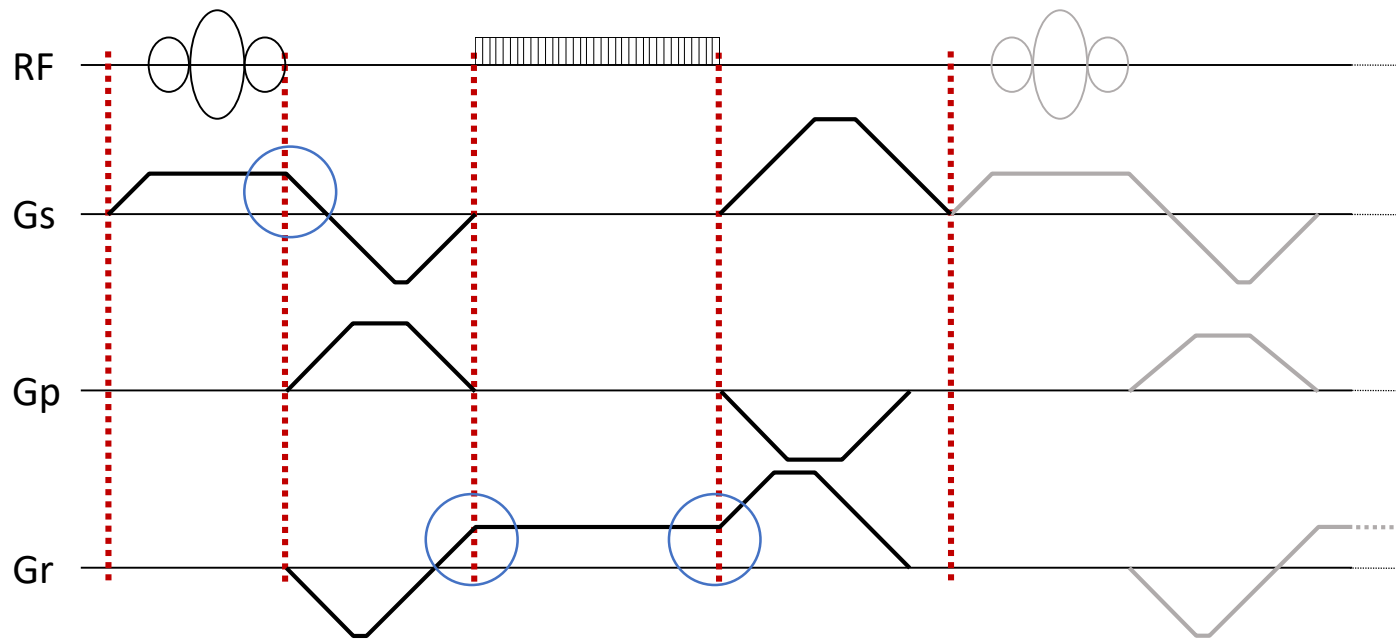
Pulseq file

- Explicit (low level) specification of the pulse sequence
 - Think of an MP3 file (or more precisely lossless FLAC)
- No loops, no parameters, no dependencies, no fuss!
- Text file (human-readable)
 - Simple hierarchy (RF pulses, gradients, shapes)
 - Event table keeps it together
 - See <http://pulseq.github.io/specification.pdf> for more details



Pulse sequence definition

Concatenation of non-overlapping blocks



- Block 1:
gradient and RF
- Block 2:
only gradients
- Block 3:
gradient and ADC
- Block 4:
only gradients
- Block 5:
gradient and RF ...

Gradients do not have to start or end at 0 at the block boundaries
(advanced feature available on some hardware platforms)

High-level programming environments

- Matlab *Pulseq* toolbox
- Python *pypulseq* toolbox



- Further options
 - Pulseq-GPI – a graphical sequence programming environment
 - TOPPE is primarily targeted at GE but can import and export *pulseq* files
 - GammaStar can export *pulseq* files
 - JEMRIS Bloch simulator can export *pulseq* files
 - CoreMRI Bloch simulator can export *pulseq* files
 - ...



Matlab Pulseseq workflow

- Define the system properties
- Define high-level parameters (convenience)
- Define pulses 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 'gre.seq' to the scanner and run it!
- *Screenshot shows an entire runnable gradient echo sequence code (similar to Siemens' example miniFlash)*

```
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')
```

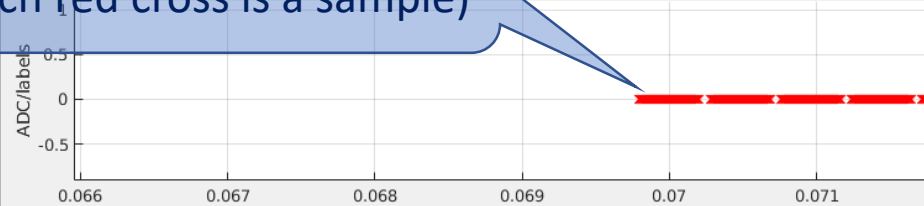
Development and debugging

- Visualize sequences with `seq.plot()`
- Verify raster alignment and system delays with `seq.checkTiming()`
- Check echo conditions and visualize k-spaces with `seq.calculateKspace()` and `calculateKspacePP()`
- Check details of gradient waveforms with `seq.gradient_waveforms()`
- Evaluate further details (TE,TR, slew rate, etc...) with `seq.testReport()`
- Listen to `seq.sound()`

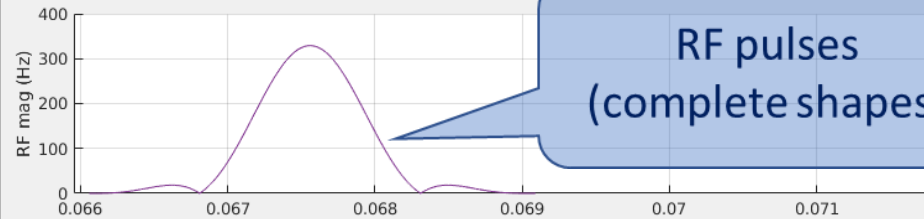


Plotting the sequence timing

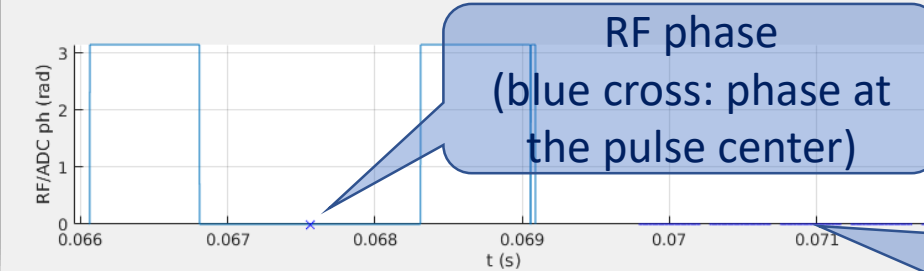
ADC samples
(each red cross is a sample)



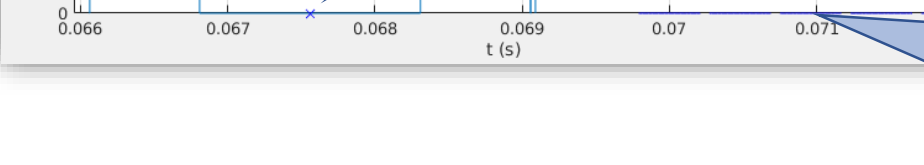
RF pulses
(complete shapes)



RF phase
(blue cross: phase at the pulse center)



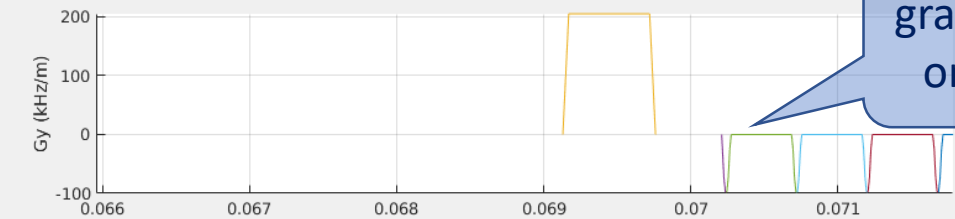
ADC phase
(blue dots: phase at each sample)



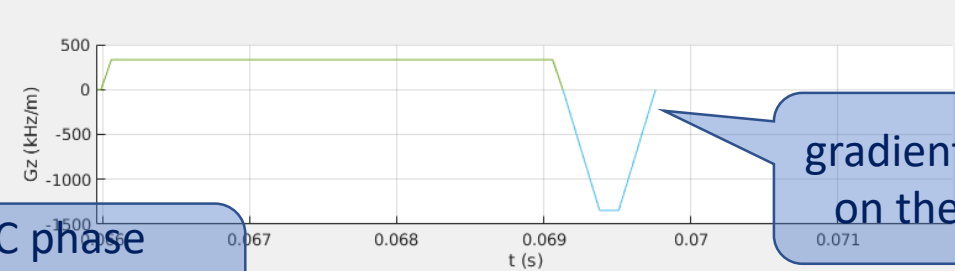
gradient events on the X axis
(each in a different color)



gradient events
on the Y axis



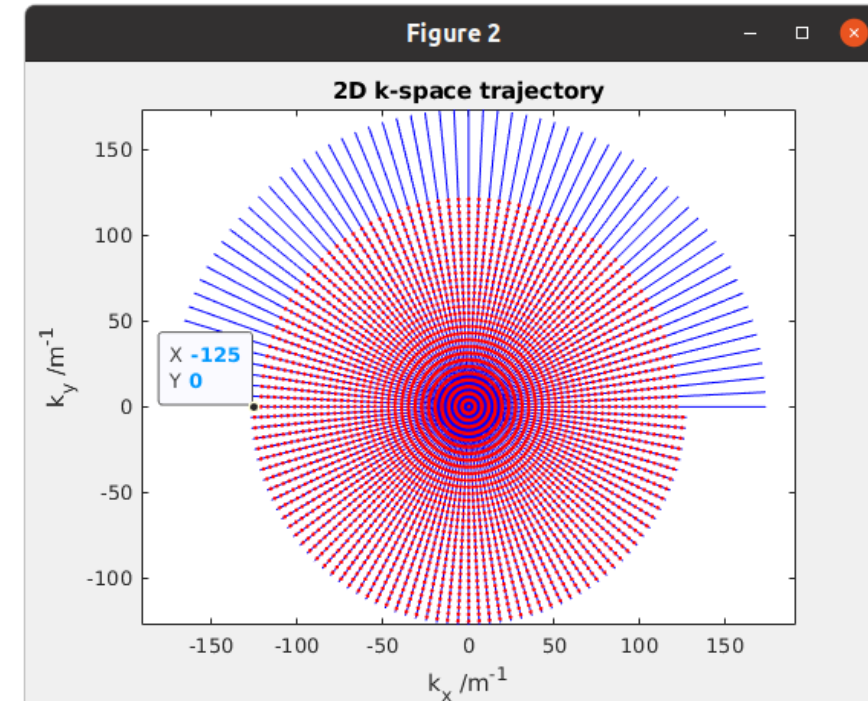
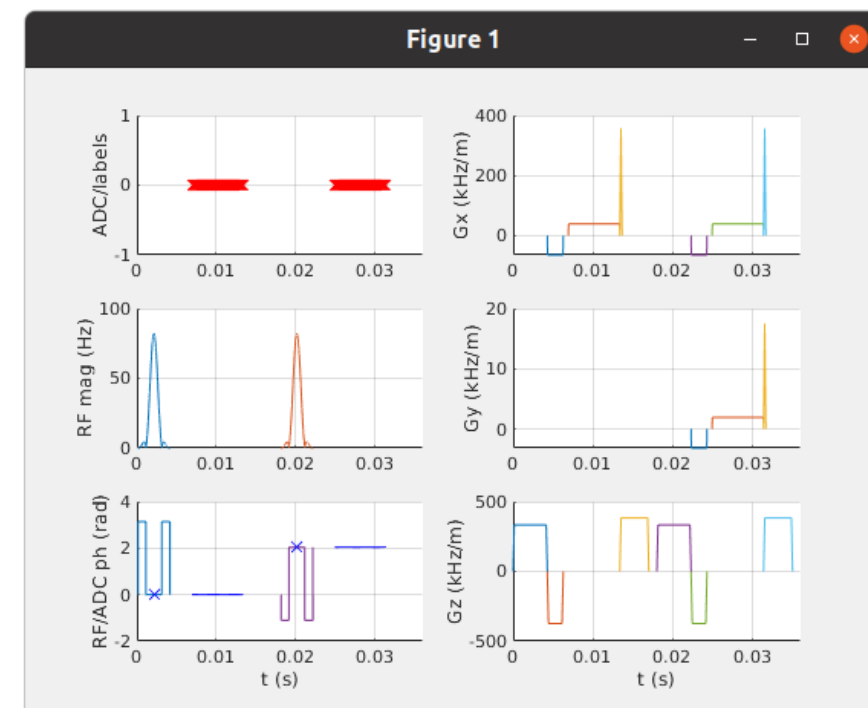
gradient events
on the Z axis



- Further plot options: 'timeRange', 'TimeDisp', 'showBlocks', 'Label'

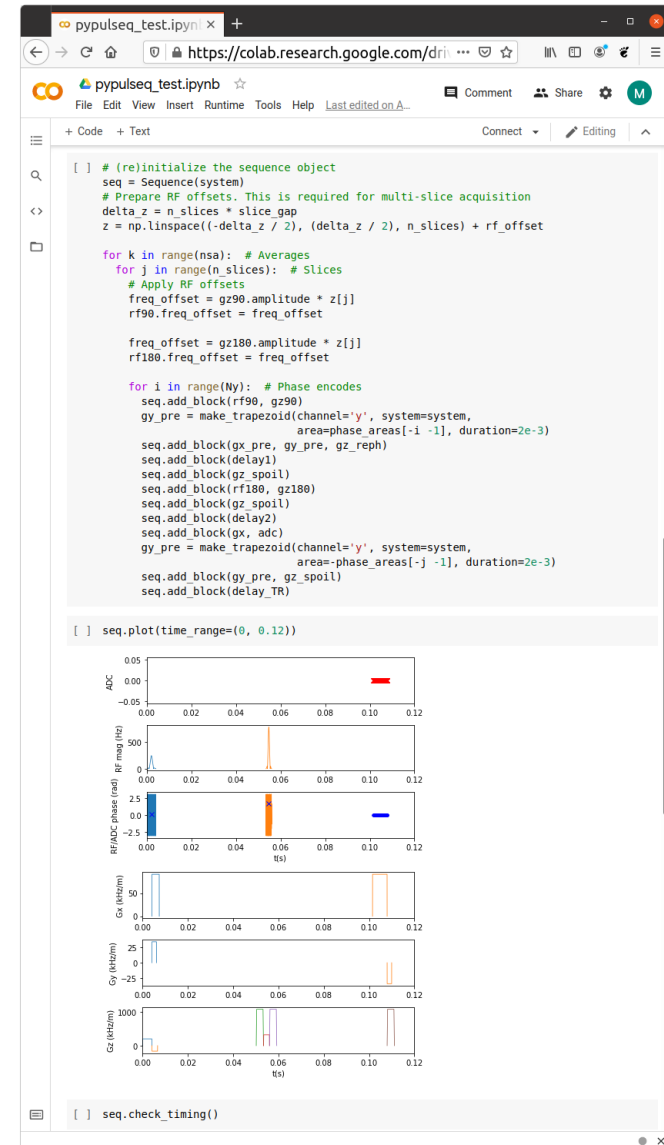
Pulseseq unit conventions

- Default to SI units where possible
- Default gradient unit is **Hz/m**
 - Other units possible
- K-space unit is **1/m**
- FOV calculations are trivial
$$k_{\max} = 1/\Delta x \quad \Delta k = 1/\text{FOV}$$
 - No γ required!
 - No 2π required!
- K-space analysis is trivial
 - Spatial resolution of the sequence in Fig. 2 is $1\text{m}/125 = 4\text{mm}$



PyPulseseq : a 100% free *Pulseseq*

- **PyPulseseq** is a close replica of the original Pulseseq toolbox that does not require a MATLAB license
- Runs in many of established Python environments, e.g. in Jupyter notebook (<http://jupyter.org/>)
- **PyPulseseq** may now run in your browser via the Google Colaboratory
- To learn more about **PyPulseseq** check out ISMRM 2021 e-poster #3760 on Thursday or visit <http://github.com/imr-framework/pypulseseq>

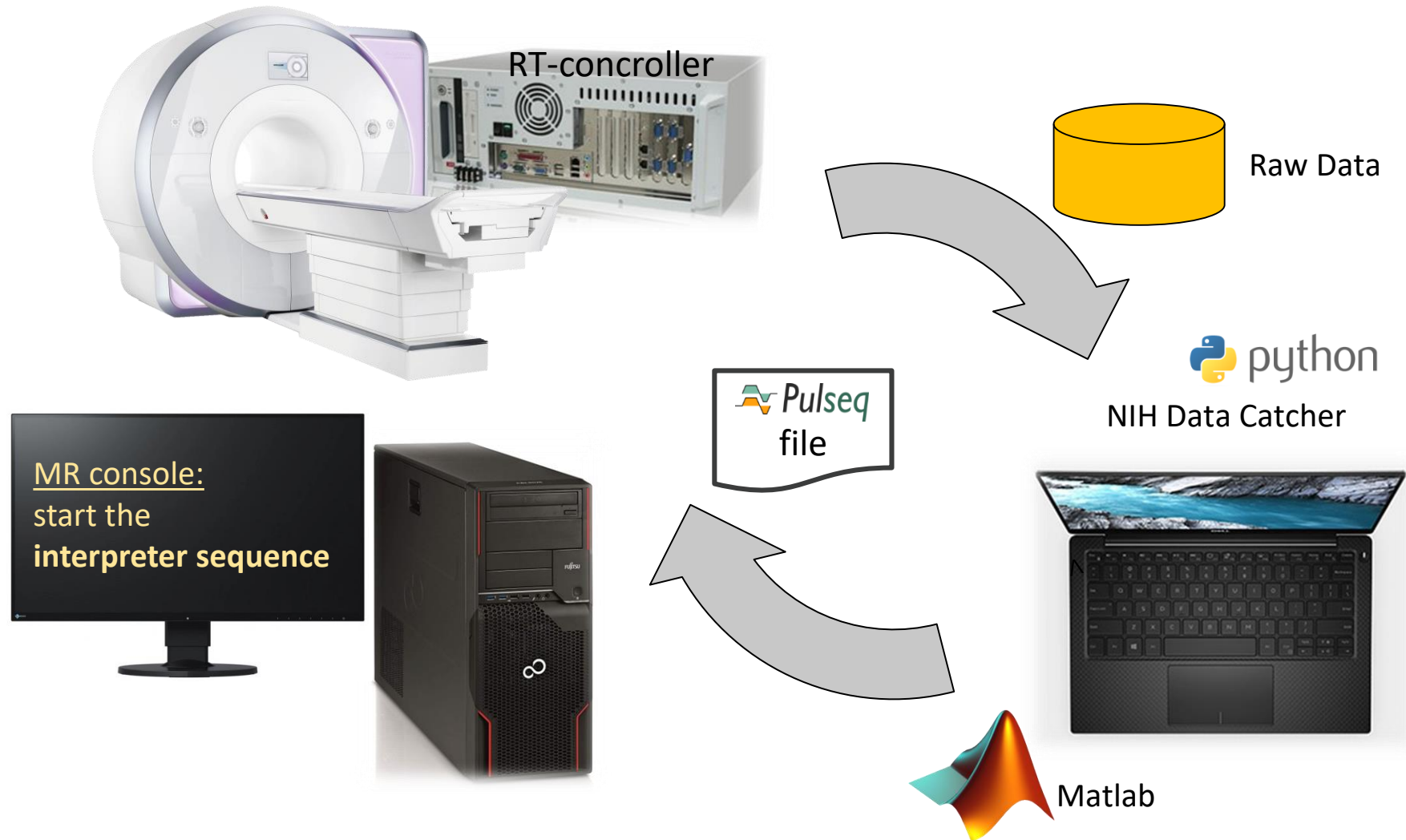


Outline

- *Pulseseq* sequence definition and programming language overview
- Using **Pulseseq** on different types of scanners:
 - Siemens, GE, and tabletop scanners using OCRA
- Live demo of *Pulseseq* across sites
 1. Basic pulse-acquire scans: Free Induction Decay (FID), Spin-Echo (SE)
 2. Cartesian spin-eco imaging
 3. Radial imaging (spin-echo and gradient echo)
 4. Advanced topics (gradient delays and correction, etc...)

Pulseq on Siemens scanners

- *Optional step:*
connect an additional
PC to the scanner
- Save the .seq file on
the scanner as
external.seq
- Run the
interpreter_sequence
on the scanner
- *Optional step:*
stream raw data to
your PC with
NIH_DataCatcher
- or export raw data
manually

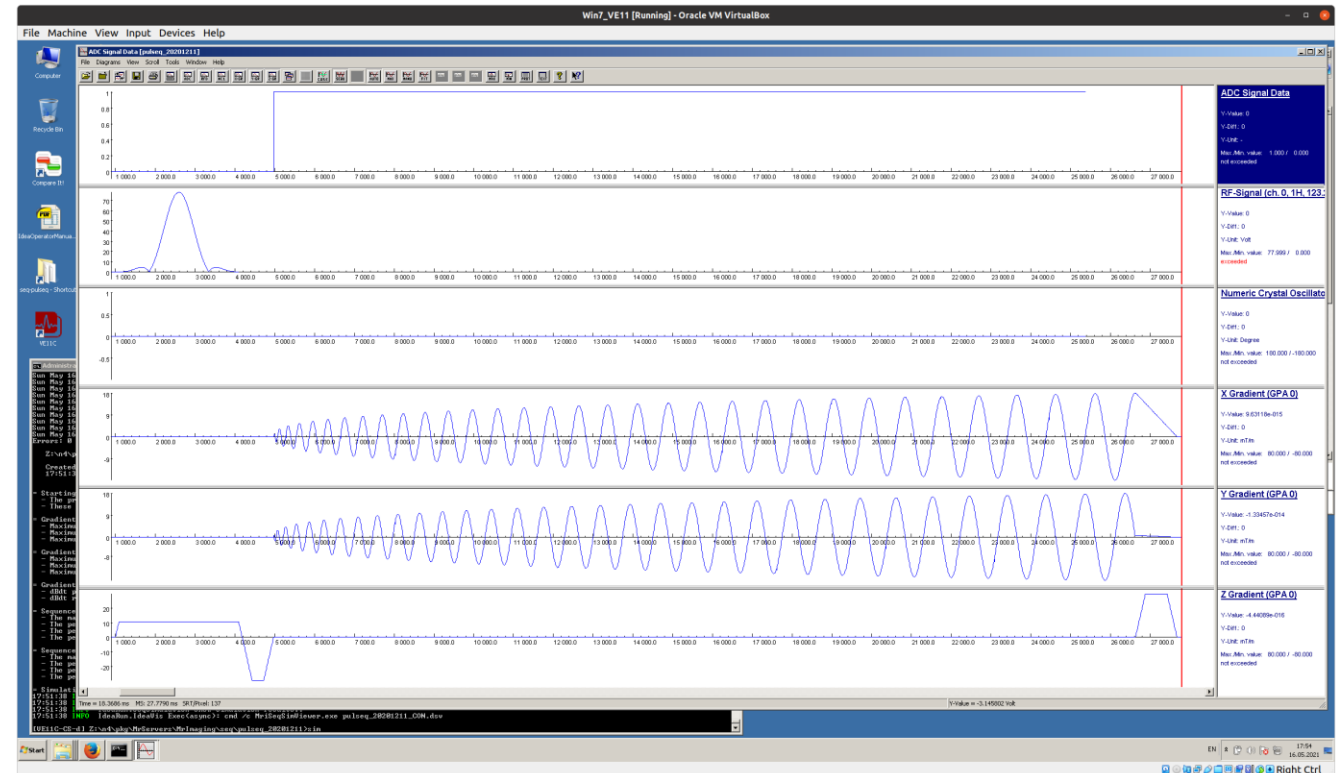
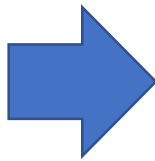
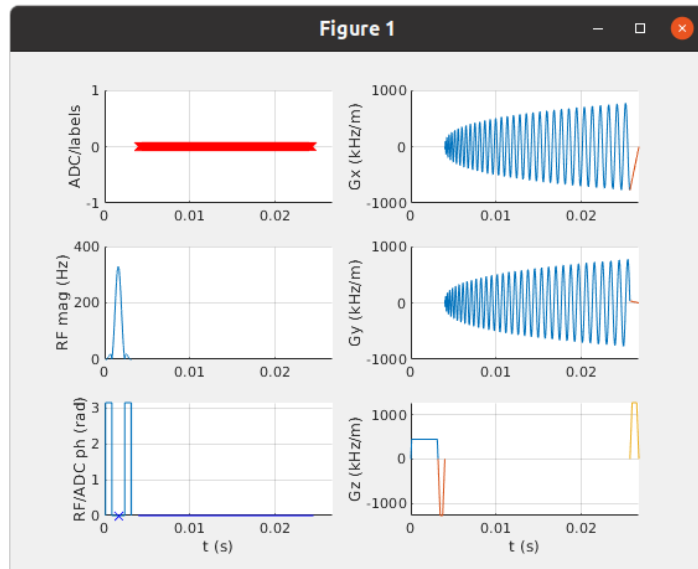




IDEA simulation with *Pulseq*

Pulseq interpreter sequence can also be used with the Siemens' IDEA

1. Save your .seq file as %CustomerSeq%/Pulseq/external.seq
2. In the IDEA command run **sim**



To acquire the Pulseq interpreter sequence contact pulseq.mr@uniklinik-freiburg.de

Pulseq on GE scanners

Step 1: Download the 'PulseqGEq' and 'TOPPE' Matlab toolboxes

```
$ git clone git@github.com:toppeMRI/PulseGEq.git  
$ git clone git@github.com:toppeMRI/toppe.git
```

Step 2: Set system hardware parameters (max gradient, slew, etc)

```
>> sys = toppe.systemspecs('maxGrad',5,'gradUnit','G/cm');
```

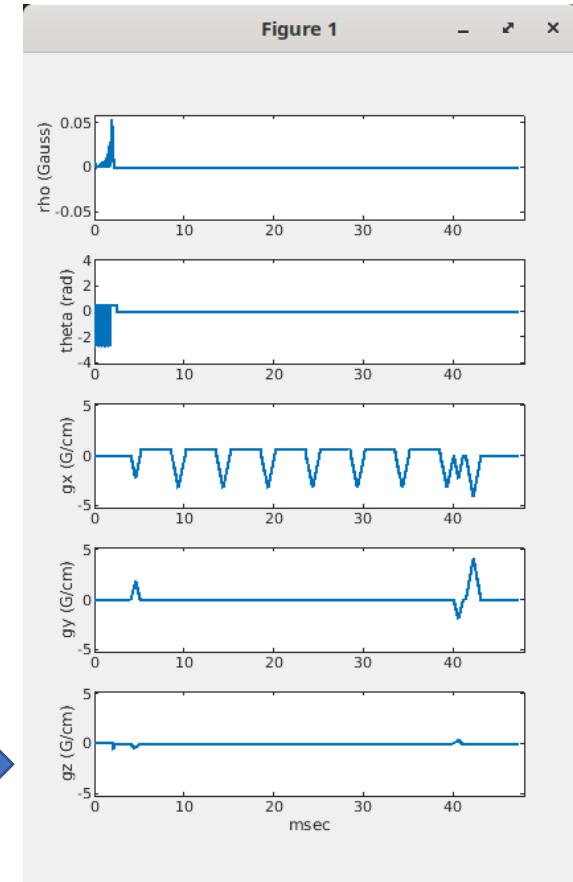
Step 3: Convert the .seq file to the TOPPE file format

```
>> pulseseq.seq2ge('se_radial.seq','system',sys);
```

Step 4: Preview the sequence in movie/loop mode

```
>> toppe.playseq(3);
```

Step 5: Copy 'toppeScanFiles.tar' to the scanner, untar, and run the TOPPE interpreter sequence

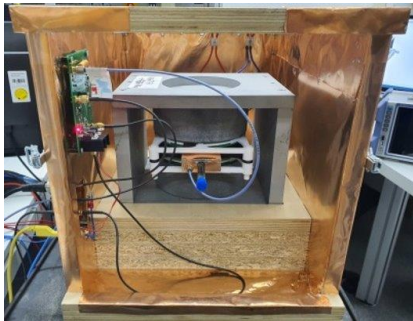




Pulseq on tabletop scanners using OCRA/FLOCRA

OCRA/FLOCRA: Open source consoles compatible with tabletop and custom scanners

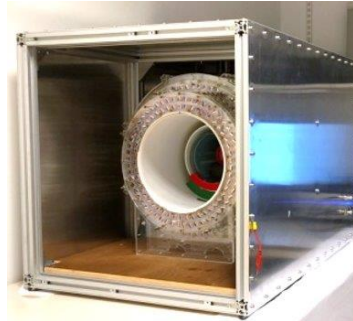
- Runs on Red Pitaya 122 (FLOCRA/OCRA) and 125 (OCRA) boards
- Open source Python code runs MR experiments using *Pulseq* .seq files



0.48T Tabletop
Dortmund, Germany



0.35T Tabletop
Boston, MA, USA



50mT Halbach
Leiden, Netherlands



Red Pitaya 122 board

FLOCRA: \$ git clone <https://github.com/lcbMGH/flocra-pulseq.git>

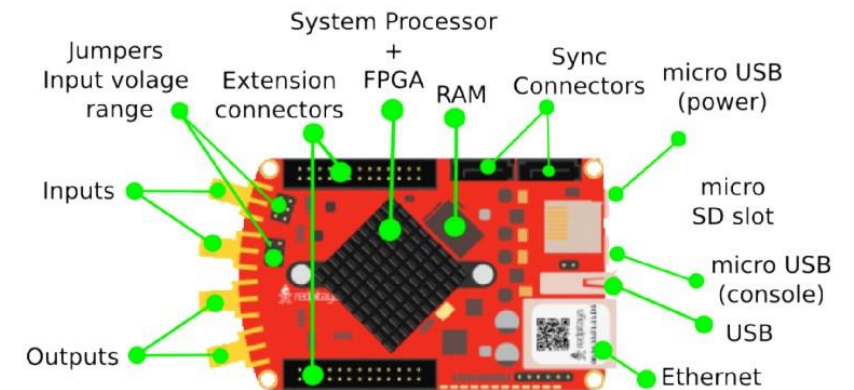
OCRA: \$ git clone --recurse-submodules <https://github.com/OpenMRI/ocra-pack.git>



Pulseq on tabletop scanners using OCRA/FLOCRA

RP-122/125 Board Hardware:

- Two RF inputs, two RF outputs, capable of 300 kHz to 100 MHz signals.
- 16 digital pins for extensions – used for gradient SPI bus and TR-switch control
- 125 (RP-125), 122.88 (RP122) MS/s sample rate
- Compatible with OCRA1 and GPA-FHDO gradient DAC boards
- Ethernet connection for control



FLOCRA: \$ git clone <https://github.com/lcbMGH/flocra-pulseq.git>

OCRA: \$ git clone --recurse-submodules <https://github.com/OpenMRI/ocra-pack.git>

Outline



- *Pulseseq* sequence definition and programming language overview
- Using *Pulseseq* on different types of scanners:
 - Siemens, GE, and tabletop scanners using OCRA
- Live demo of ***Pulseseq*** across sites
 1. Basic pulse-acquire scans: Free Induction Decay (FID), Spin-Echo (SE)
 2. Cartesian spin-eco imaging
 3. Radial imaging (spin-echo and gradient echo)
 4. *Advanced topics (gradient delays and correction, etc...) are only covered in the live portion of the tutorial*

Pulseq – seq01

- Example 1:
free induction decay
sequence
- After line 25
the sequence is ready
(in Matlab's memory)
- Further lines: display,
sanity checks, export to
disk

RF pulse

ADC

basic
parameters

create blocks

```
/home/zaitsev/pulseq_home/pulseq_public/matlab/ismrmTutorial/seq/seq01_Fid.m

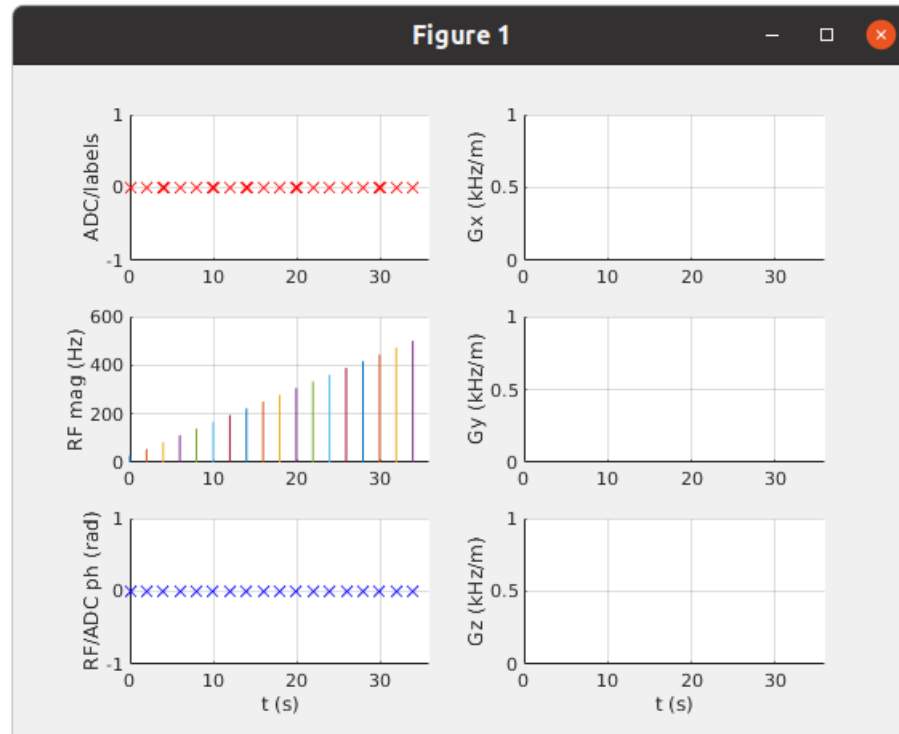
EDITOR PUBLISH VIEW
New Open Save Find Files Compare Go To EDIT Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE BREAKPOINTS RUN

1 - system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2 -     'adcDeadTime', 20e-6);
3 -
4 - seq=mr.Sequence(system); % Create a new sequence object
5 - Nx=4096;
6 - Nrep=1;
7 - adcDur=51.2e-3;
8 - rfDur=500e-6;
9 - TR=5000e-3;
10 - TE=10e-3;
11 -
12 - % Create non-selective pulse
13 - rf = mr.makeBlockPulse(pi/2,'Duration',rfDur, 'system', system);
14 -
15 - % Define delays and ADC events
16 - adc = mr.makeAdc(Nx,'Duration',adcDur, 'system', system, 'delay', TE-rfDur/2-system.rfRingdownTime);
17 -
18 - delayTR=TR-mr.calcDuration(rf);
19 - assert(delayTR>=0);
20 -
21 - % Loop over repetitions and define sequence blocks
22 - for i=1:Nrep
23 -     seq.addBlock(rf);
24 -     seq.addBlock(adc,mr.makeDelay(delayTR));
25 - end
26 -
27 - % show the "interesting part" of the sequence
28 - seq.plot('timeRange',[0 TE+rfDur+adcDur+1e-3]);
29 -
30 - % check whether the timing of the sequence is compatible with the scanner
31 - [ok, error_report]=seq.checkTiming;
32 -
33 - if (ok)
34 -     fprintf('Timing check passed successfully\n');
35 - else
36 -     fprintf('Timing check failed! Error listing follows:\n');
37 -     fprintf([error_report{:}]);
38 -     fprintf('\n');
39 - end
40 -
41 - seq.write('fid.seq') % Write to pulseq file
42 - %seq.install('siemens'); % copy to scanner

UTF-8 script Ln 3 Col 1
```

Pulseq – seq02

- Example 2:
FID sequence with
flip angle variation



```
/home/zaitsev/pulseseq_home/pulseseq_public/matlab/ismrmTutorial/seq/seq02_Fid_multipleFAs.m - [x]
EDITOR PUBLISH VIEW
New Open Save Find Files Compare Go To EDIT Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE BREAKPOINTS RUN

1 system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2         'adcDeadTime', 20e-6);
3
4 seq=mr.Sequence(system); % Create a new sequence object
5 Nx=4096;
6 Nrep=1;
7 adcDur=51.2e-3;
8 rfDur=1000e-6; % increased to 1ms
9 TR=2000e-3; % may increase to ~5s avoid T1 saturation
10 TE=10e-3;
11 flip_angles=18;
12
13 % Define delays and ADC events
14 adc = mr.makeAdc(Nx,'Duration',adcDur, 'system', system, 'delay', TE-rfDur/2-system.rfRingdownTime);
15
16 delayTR=TR-mr.calcDuration(rf);
17 assert(delayTR>=0);
18
19 % Loop over repetitions and define sequence blocks
20 for f=1:flip_angles
21     % Create non-selective pulse
22     rf = mr.makeBlockPulse(pi/flip_angles*f,'Duration',rfDur, 'system', system);
23     % Loop over repetitions and define sequence blocks
24     for i=1:Nrep
25         seq.addBlock(rf);
26         seq.addBlock(adc,mr.makeDelay(delayTR));
27     end
28 end
29
30 % show the entire sequence
31 seq.plot();
32
33 % check whether the timing of the sequence is compatible with the scanner
34 [ok, error_report]=seq.checkTiming();
35
36 if (ok)
37     fprintf('Timing check passed successfully\n');
38 else
39     fprintf('Timing check failed! Error listing follows:\n');
40     fprintf([error_report{:}]);
41     fprintf('\n');
42 end
43
44 seq.write('fid.seq') % Write to pulseseq file
45 %seq.install('siemens'); % copy to scanner

UTF-8 script Ln 3 Col 1
```

RF pulse
now here

Pulseq – seq03

- Example 3:
naive spin-echo
- The sequence is ready
after line 34
- We (ab)use k-space
calculation routines for
their ability to deliver
timestamps

two RF
pulses

check TE
calculation

```
/home/zaitsev/pulseseq_home/pulseseq_public/matlab/ismrmTutorial/seq/seq03_SE.m
EDITOR PUBLISH VIEW
New Open Save Find Files Compare Go To Insert Comment % % % % Breakpoints Run Run and Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
1 - system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2 - 'adcDeadTime', 20e-6);
3 -
4 - seq=mr.Sequence(system); % Create a new sequence object
5 - Nx=4096;
6 - Nrep=1;
7 - adcDur=51.2e-3;
8 - rfDur=1000e-6;
9 - TR=100e-3;
10 - TE=54e-3;
11 -
12 - % Create non-selective excitation and refocusing pulses
13 - rf_ex = mr.makeBlockPulse(pi/2,'Duration',rfDur, 'system', system);
14 - rf_ref = mr.makeBlockPulse(pi,'Duration',rfDur, 'system', system, 'PhaseOffset', pi/2, 'use', 'refocusing');
15 -
16 - % Define delays and ADC events
17 - delayTE1=TE/2-mr.calcDuration(rf_ex)/2-mr.calcDuration(rf_ref)/2;
18 - delayTE2=TE/2-mr.calcDuration(rf_ref)+rf_ref.delay+mr.calcRfCenter(rf_ref)-adcDur/2; % this is not perfect, but
19 -
20 - adc = mr.makeAdc(Nx,'Duration',adcDur, 'system', system, 'delay', delayTE2);
21 -
22 - delayTR=TR-mr.calcDuration(rf_ex)-delayTE1-mr.calcDuration(rf_ref);
23 -
24 - assert(delayTE1>=0);
25 - assert(delayTE2>=0);
26 - assert(delayTR>=0);
27 -
28 - % Loop over repetitions and define sequence blocks
29 - for i=1:Nrep
30 -     seq.addBlock(rf_ex);
31 -     seq.addBlock(mr.makeDelay(delayTE1));
32 -     seq.addBlock(rf_ref);
33 -     seq.addBlock(adc,mr.makeDelay(delayTR));
34 - end
35 -
36 - % show the entire sequence showing the block structure
37 - seq.plot('showBlocks',1,'timeDisp','ms');
38 -
39 - % check whether the timing of the sequence is compatible with the scanner
40 - [ok, error_report]=seq.checkTiming();
41 -
42 - if (ok)
43 -     fprintf('Timing check passed successfully\n');
44 - else
45 -     fprintf('Timing check failed! Error listing follows:\n');
46 -     fprintf([error_report{:}]);
47 -     fprintf('\n');
48 - end
49 -
50 - seq.write('se.seq') % Write to pulseseq file
51 - %seq.install('siemens'); % copy to scanner
52 -
53 - % calculate k-space but only use it to check the TE calculation
54 - [ktraj_adc, t_adc, ktraj, t_ktraj, t_excitation, t_refocusing] = seq.calculateKspacePP();
55 -
56 - assert(abs(t_refocusing-t_excitation-TE/2)<1e-6); % check that the refocusing happens at the 1/2 of TE
57 - assert(abs(t_adc(Nx/2)-t_excitation-TE)<adc.dwell); % check that the echo happens as close as possible to the
```

more delays
to calculate

Pulseq – seq04

- Example 4:
spin-echo with
spoilers around
the refocusing pulse
- Create spoiling gradient
- Use it in the sequence in
two different blocks

spoiling
area

```
/home/zaitsev/pulseq_home/pulseq_public/matlab/ismrmTutorial/seq/seq04_SE_withSpoilers.m
EDITOR PUBLISH VIEW
New Open Save Find Files Compare Go To Find Insert Comment Indent Breakpoints Run Run and Advance Run Section Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN

1 system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2 'adcDeadTime', 20e-6);
3
4 seq=mr.Sequence(system); % Create a new sequence object
5 Nx=4096;
6 Nrep=1;
7 adcDur=51.2e-3;
8 rfDur=1000e-6;
9 TR=250e-3;
10 TE=54e-3;
11 spA=1000; % spoiler area in 1/m (=Hz/m*s)
12 % todo: change spoiler area, remove one of spoilers and observe the signal
13
14 % Create non-selective excitation and refocusing pulses
15 rf_ex = mr.makeBlockPulse(pi/2, 'Duration', rfDur, 'system', system);
16 rf_ref = mr.makeBlockPulse(pi, 'Duration', rfDur, 'system', system, 'PhaseOffset', pi/2, 'use', 'refocusing');
17
18 % calculate spoiler gradient, let's put it on X axis for now
19 g_sp=mr.makeTrapezoid('x', 'Area', spA, 'system', system);
20 rf_ref.delay=max(mr.calcDuration(g_sp), rf_ref.delay);
21
22 % Define delays and ADC events
23 delayTE1=TE/2-(mr.calcDuration(rf_ex)-mr.calcRfCenter(rf_ex)-rf_ex.delay)-rf_ref.delay-mr.calcRfCenter(rf_ref);
24 delayTE2=TE/2-mr.calcDuration(rf_ref)+rf_ref.delay+mr.calcRfCenter(rf_ref)-adcDur/2; % this is not perfect, but
25 assert(delayTE2>mr.calcDuration(g_sp));
26
27 adc = mr.makeAdc(Nx, 'Duration', adcDur, 'system', system, 'delay', delayTE2);
28
29 delayTR=TR-mr.calcDuration(rf_ex)-delayTE1-mr.calcDuration(rf_ref);
30
31 assert(delayTE1>=0);
32 assert(delayTE2>=0);
33 assert(delayTR>=0);
34
35 % Loop over repetitions and define sequence blocks
36 for i=1:Nrep
37 seq.addBlock(rf_ex);
38 seq.addBlock(mr.makeDelay(delayTE1));
39 seq.addBlock(rf_ref, g_sp);
40 seq.addBlock(adc, g_sp, mr.makeDelay(delayTR));
41 end
42
43 seq.plot(); % show the entire sequence
44
45 % check whether the timing of the sequence is compatible with the scanner
46 [ok, error_report]=seq.checkTiming();
47
48 if (ok)
49 fprintf('Timing check passed successfully\n');
50 else
51 fprintf('Timing check failed! Error listing follows:\n');
52 fprintf([error_report{:}]);
53 fprintf('\n');
54 end
55
56 seq.write('se.seq') % Write to pulseq file
57 %seq.install('siemens'); % copy to scanner
58
59 % calculate k-space but only use it to check timing
60 [ktraj_adc, t_adc, ktraj, t_ktraj, t_excitation, t_refocusing] = seq.calculateKspacePP();
61
62 assert(abs(t_refocusing-t_excitation-TE/2)<1e-6); % check that the refocusing happens at the 1/2 of TE
63 assert(abs(t_adc(Nx/2)-t_excitation-TE)<adc.dwell); % check that the echo happens as close as possible to the m
```

Pulseq – seq05

- Example 5:
Cartesian spin-echo imaging sequence
- Use shorter ADC window compared to the spectroscopy sequence
- More gradients to create
- More complex delay calculation
- The sequence is ready at line 67
- *See timing checks in the live portion*

dummy
scans

calculate
delays

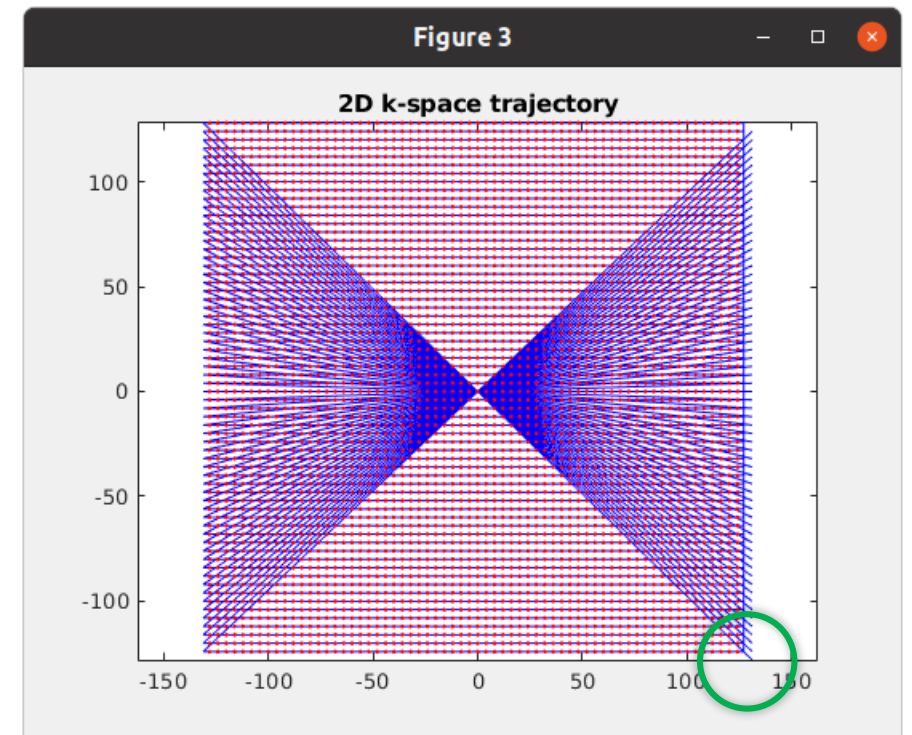
basic
parameters

PE steps

```
/home/zaitsev/pulseq_public/matlab/isrmrTutorial/seq/seq05_CartesianSE.m
EDITOR PUBLISH VIEW
New Open Save Compare Go To Find Comment % % % Breakpoints Run Run and Advance Run Section Run Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
1 system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2 'adcDeadTime', 20e-6);
3
4 seq=mr.Sequence(system); % Create a new sequence object
5 adcDur=2.5e-3;
6 rfDur1=3e-3;
7 rfDur2=1e-3;
8 TR=150e-3;
9 TE=54e-3; % 20ms still works with the chosen parameters & syssem props
10 spA=1000; % spoiler area in 1/m (=Hz/m*s) % MZ: need 5000 for my oil
11
12 sliceThickness=3e-3; % slice
13 fov=250e-3; Nx=256; % Define FOV and resolution
14 Ny=Nx; % number of radial spokes
15 Ndummy=10; % number of dummy scans
16
17 % Create 90 degree slice selection pulse and gradient and even the
18 % refocusing gradient; we will not use the latter however but will subtract
19 % its area from the first (left) spoiler
20 [rf_ex, gs, gr] = mr.makeSincPulse(pi/2,system,'Duration',3e-3,...
21 'SliceThickness',sliceThickness,'apodization',0.4,'timeBwProduct',4);
22
23 % Create non-selective refocusing pulse
24 rf_ref = mr.makeBlockPulse(pi,'Duration',rfDur2,'system',system,'PhaseOffset',pi/2,'use','refocusing'); % need
25
26 % calculate spoiler gradient
27 g_sp1=mr.makeTrapezoid('z','Area',spA+gsr.area,'system',system);
28 rf_ref.delay=max(mr.calcDuration(g_sp1),rf_ref.delay);
29
30 g_sp2=mr.makeTrapezoid('z','Area',spA,'system',system);
31
32 % Define delays and ADC events
33 deltak=1/fov;
34 gr = mr.makeTrapezoid('x',system,'FlatArea',Nx*deltak,'FlatTime',adcDur);
35 adc = mr.makeAdc(Nx,system,'Duration',adcDur,'delay',gr.riseTime);
36
37 grPredur = 5e-3; % use a fixed time to make this gradient visible on the plot
38 grPre = mr.makeTrapezoid('x',system,'Area',gr.area/2+deltak/2,'Duration',grPredur);
39 phaseAreas = ((0:Ny-1)-Ny/2)*deltak;
40
41 delayTE1=TE/2-(mr.calcDuration(gs)-mr.calcRfCenter(rf_ex)-rf_ex.delay)-rf_ref.delay-mr.calcRfCenter(rf_ref)-grPredur;
42 delayTE2=TE/2-mr.calcDuration(rf_ref)+rf_ref.delay-mr.calcRfCenter(rf_ref)-adc.delay-adcDur/2; % this is not perfect,
43 delayTR=TR-mr.calcDuration(gs)-grPredur-delayTE1-mr.calcDuration(rf_ref)-delayTE2-mr.calcDuration(gr);
44
45 assert(delayTE1>=0);
46 assert(delayTE2>mr.calcDuration(g_sp2));
47 assert(delayTR>=0);
48
49 % Loop over repetitions and define sequence blocks
50 for i=(1:Ndummy):Ny
51 seq.addBlock(rf_ex,gs);
52 if (i>0) % semi-negative index -- dummy scans
53 gy = mr.makeTrapezoid('y','Area',phaseAreas(i),'Duration',grPredur);
54 else
55 gy = mr.makeTrapezoid('y','Area',0,'Duration',grPredur);
56 end
57 seq.addBlock(grPre,gy);
58 seq.addBlock(mr.makeDelay(delayTE1));
59 seq.addBlock(rf_ref,g_sp1);
60 seq.addBlock(g_sp2, mr.makeDelay(delayTE2));
61 if (i>0) % semi-negative index -- dummy scans
62 seq.addBlock(adc,gr);
63 else
64 seq.addBlock(gr);
65 end
66 seq.addBlock(gy,mr.makeDelay(delayTR));
67 end
68
69 % show the first non-dummy TR with the block structure
70 seq.plot('showBlocks',1,'timeRange',TR*(Ndummy+[0 1]),'timeDisp','us');
```

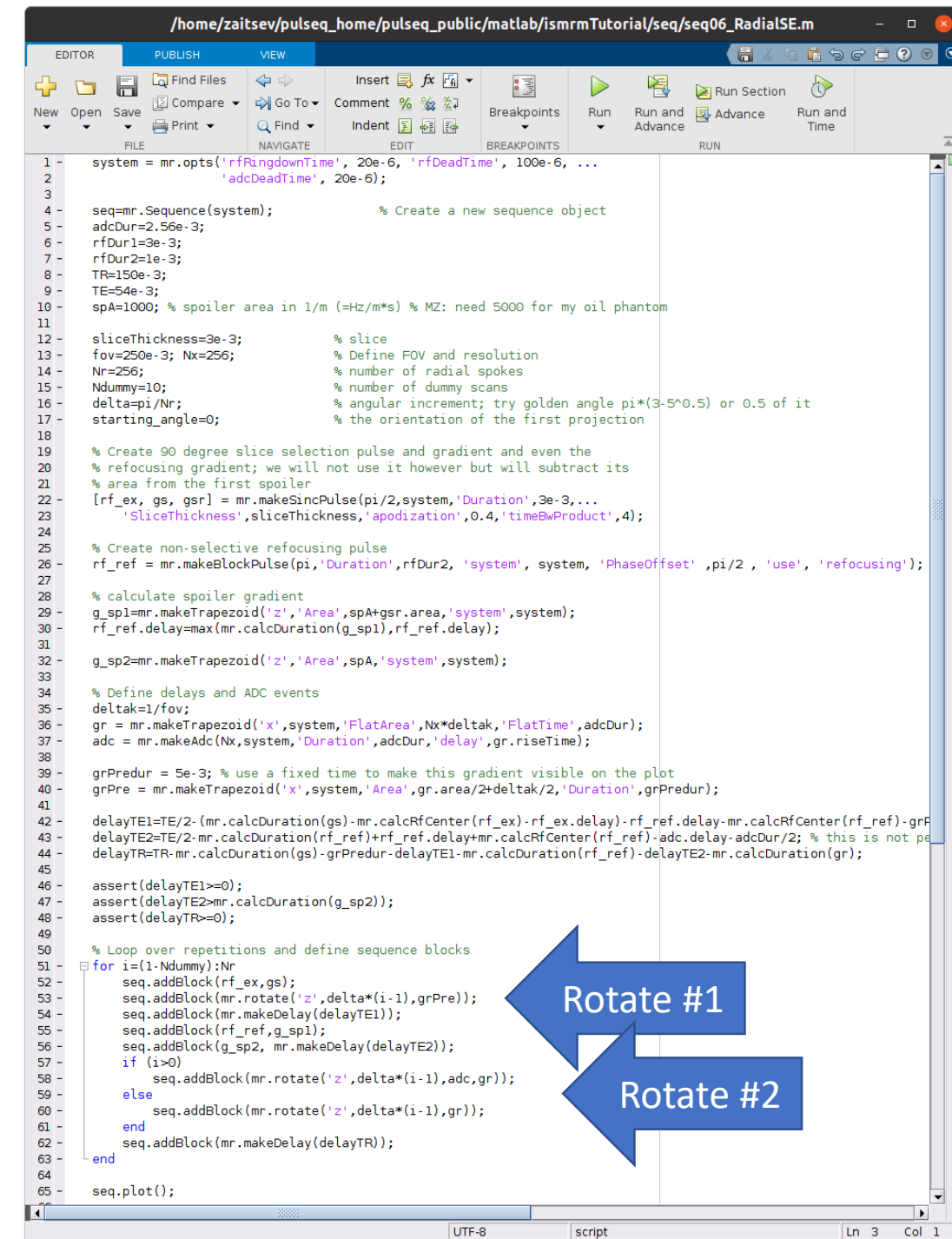

Pulseseq – seq05 contd.

- Example 5:
Cartesian spin-echo imaging sequence
- Very useful option in the Pulseseq toolbox is the k-space calculation
- See how we travel in the k-space to the bottom right in the last TR to be “teleported” to the upper left corner (*see also $k(t)$ plots in the live session*)



Pulseq – seq06

- Example 6:
2D radial spin-echo imaging sequence
- Remove phase encoding
- Rotate read prephaser and readout gradients
- The sequence is actually simpler
- The sequence is ready at line 63
- *See timing checks in the live portion*



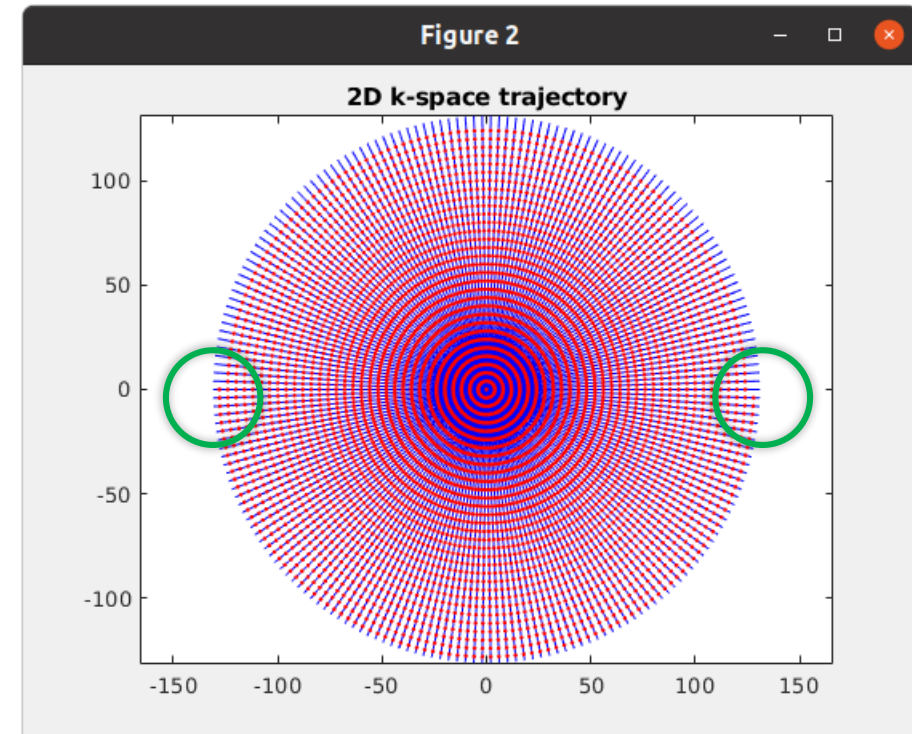
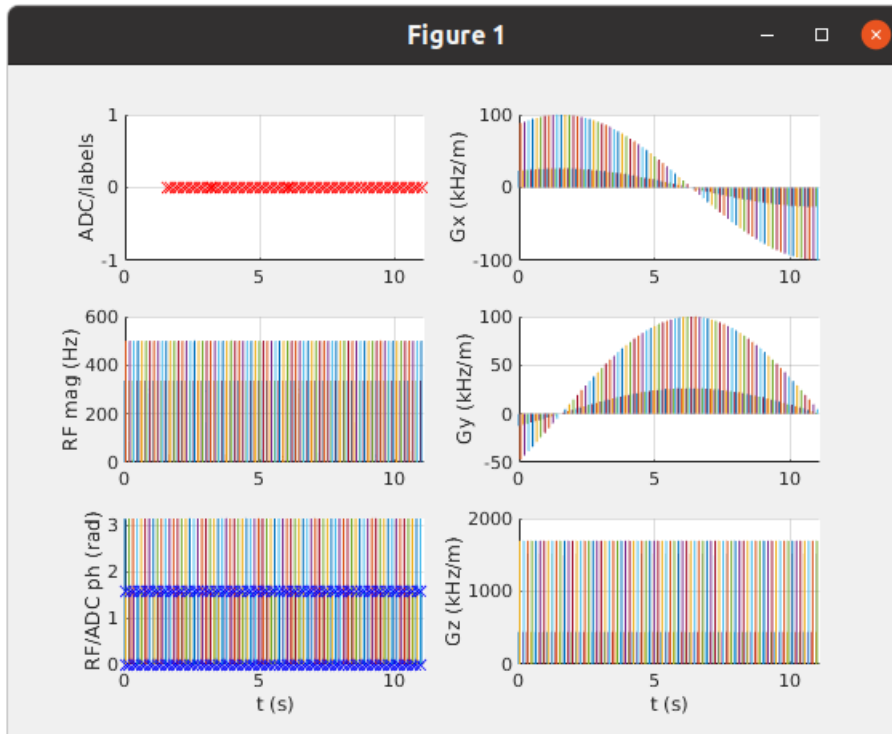
```
/home/zaitsev/pulseseq_home/pulseseq_public/matlab/ismrmTutorial/seq/seq06_RadialSE.m
EDITOR PUBLISH VIEW
New Open Save Compare Print Find Indent Breakpoints Run Run and Advance Run Section Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
1 system = mr.opts('rfRingdownTime', 20e-6, 'rfDeadTime', 100e-6, ...
2 'adcDeadTime', 20e-6);
3
4 seq=mr.Sequence(system); % Create a new sequence object
5 adcDur=2.56e-3;
6 rfDur1=3e-3;
7 rfDur2=1e-3;
8 TR=150e-3;
9 TE=54e-3;
10 spA=1000; % spoiler area in 1/m (=Hz/m*s) % MZ: need 5000 for my oil phantom
11
12 sliceThickness=3e-3; % slice
13 fov=250e-3; Nx=256; % Define FOV and resolution
14 Nr=256; % number of radial spokes
15 Ndummy=10; % number of dummy scans
16 delta=pi/Nr; % angular increment; try golden angle pi*(3-5^0.5) or 0.5 of it
17 starting_angle=0; % the orientation of the first projection
18
19 % Create 90 degree slice selection pulse and gradient and even the
20 % refocusing gradient; we will not use it however but will subtract its
21 % area from the first spoiler
22 [rf_ex, gs, gsr] = mr.makeSincPulse(pi/2, system, 'Duration', 3e-3, ...
23 'SliceThickness', sliceThickness, 'apodization', 0.4, 'timeBwProduct', 4);
24
25 % Create non-selective refocusing pulse
26 rf_ref = mr.makeBlockPulse(pi, 'Duration', rfDur2, 'system', system, 'PhaseOffset', pi/2, 'use', 'refocusing');
27
28 % calculate spoiler gradient
29 g_sp1=mr.makeTrapezoid('z', 'Area', spA+gsr.area, 'system', system);
30 rf_ref.delay=max(mr.calcDuration(g_sp1), rf_ref.delay);
31
32 g_sp2=mr.makeTrapezoid('z', 'Area', spA, 'system', system);
33
34 % Define delays and ADC events
35 deltak=1/fov;
36 gr = mr.makeTrapezoid('x', system, 'FlatArea', Nx*deltak, 'FlatTime', adcDur);
37 adc = mr.makeAdc(Nx, system, 'Duration', adcDur, 'delay', gr.riseTime);
38
39 grPredur = 5e-3; % use a fixed time to make this gradient visible on the plot
40 grPre = mr.makeTrapezoid('x', system, 'Area', gr.area/2+deltak/2, 'Duration', grPredur);
41
42 delayTE1=TE/2 - (mr.calcDuration(gs) - mr.calcRfCenter(rf_ex) - rf_ex.delay) - rf_ref.delay - mr.calcRfCenter(rf_ref) - grP
43 delayTE2=TE/2 - mr.calcDuration(rf_ref) + rf_ref.delay + mr.calcRfCenter(rf_ref) - adc.delay - adcDur/2; % this is not pe
44 delayTR=TR - mr.calcDuration(gs) - grPredur - delayTE1 - mr.calcDuration(rf_ref) - delayTE2 - mr.calcDuration(gr);
45
46 assert(delayTE1>=0);
47 assert(delayTE2>mr.calcDuration(g_sp2));
48 assert(delayTR>=0);
49
50 % Loop over repetitions and define sequence blocks
51 for i=(1:Ndummy):Nr
52 seq.addBlock(rf_ex, gs);
53 seq.addBlock(mr.rotate('z', delta*(i-1), grPre));
54 seq.addBlock(mr.makeDelay(delayTE1));
55 seq.addBlock(rf_ref, g_sp1);
56 seq.addBlock(g_sp2, mr.makeDelay(delayTE2));
57 if (i>0)
58 seq.addBlock(mr.rotate('z', delta*(i-1), adc, gr));
59 else
60 seq.addBlock(mr.rotate('z', delta*(i-1), gr));
61 end
62 seq.addBlock(mr.makeDelay(delayTR));
63 end
64
65 seq.plot();
66
67 UTF-8 script Ln 3 Col 1
```

Rotate #1

Rotate #2

Pulseseq – seq06 contd.

- *Time to explore sequence timing and the trajectory*



Pulseq – seq07

- Example 7:
2D radial gradient echo
- Add RF spoiling
 - Introduce counters
 - Update phase of the RF pulse
 - Remember to update ADC phase
- Add gradient spoiler at the end of the readout gradient
- The sequence is ready at line 63
- *See timing checks in the live portion*

gradient
spoiling

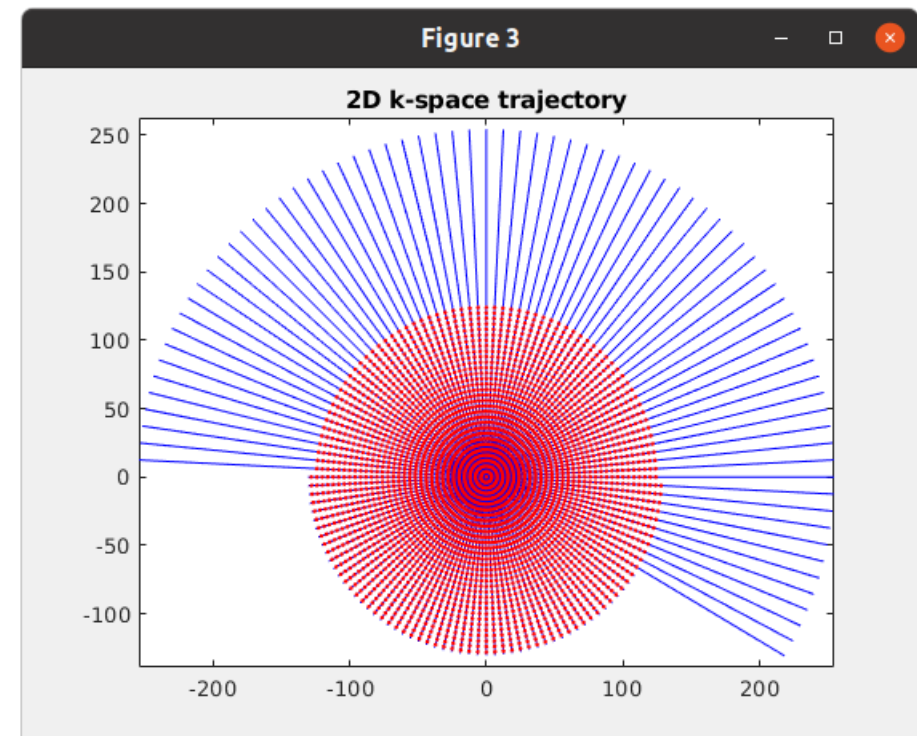
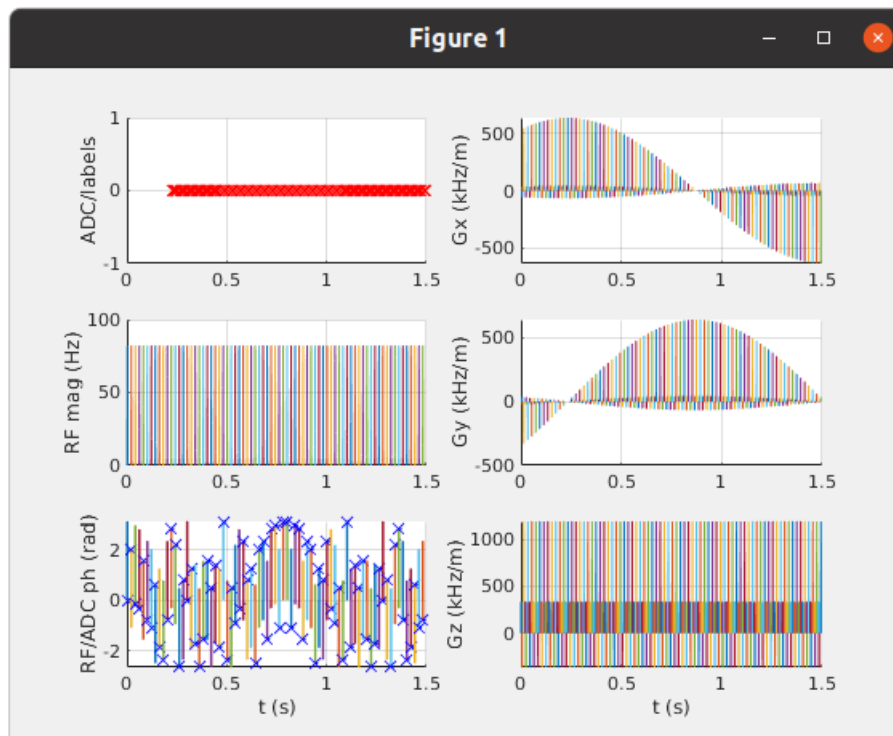
RF spoiling

RF
spoiling

```
/home/zaitsev/pulseq_home/pulseq_public/matlab/ismrmTutorial/seq/seq07_RadialGradientEcho.m
EDITOR PUBLISH VIEW
New Open Save Find Files Insert fx Breakpoints Run Run and Advance Run Section Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
1 seq=mr.Sequence(); % Create a new sequence object
2 fov=250e-3; Nx=256; % Define FOV and resolution
3 alpha=30; % flip angle
4 sliceThickness=3e-3; % slice
5 TE=8e-3; % TE; give a vector here to have multiple TEs (e.g. for field mapping)
6 TR=20e-3; % only a single value for now
7 Nr=256; % number of radial spokes
8 Ndummy=10; % number of dummy scans
9 delta=pi/Nr; % angular increment; try golden angle pi*(3-5^0.5) or 0.5 of it
10 |
11 % more in-depth parameters
12 rfSpoilingInc=117; % RF spoiling increment
13
14 % set system limits
15 sys = mr.opts('MaxGrad', 28, 'GradUnit', 'mT/m', ...
16 'MaxSlew', 80, 'SlewUnit', 'T/m/s', 'rfRingdownTime', 20e-6, ...
17 'rfDeadTime', 100e-6, 'adcDeadTime', 10e-6);
18
19 % Create alpha-degree slice selection pulse and gradient
20 [rf, gz] = mr.makeSincPulse(alpha*pi/180, 'Duration', 4e-3, ...
21 'SliceThickness', sliceThickness, 'apodization', 0.5, 'timeBwProduct', 4, 'system', sys);
22
23 % Define other gradients and ADC events
24 deltak=1/fov;
25 gx = mr.makeTrapezoid('x', 'FlatArea', Nx*deltak, 'FlatTime', 6.4e-3, 'system', sys);
26 adc = mr.makeAdc(Nx, 'Duration', gx.flatTime, 'Delay', gx.riseTime, 'system', sys);
27 gxPre = mr.makeTrapezoid('x', 'Area', -gx.area/2-deltak/2, 'Duration', 2e-3, 'system', sys); % we need this "d
28 gzReph = mr.makeTrapezoid('z', 'Area', -gz.area/2, 'Duration', 2e-3, 'system', sys);
29
30 % gradient spoiling
31 gxSpoil=mr.makeTrapezoid('x', 'Area', 0.5*Nx*deltak, 'system', sys);
32 gzSpoil=mr.makeTrapezoid('z', 'Area', 4/sliceThickness, 'system', sys);
33
34 % Calculate timing
35 delayTE=ceil((TE - mr.calcDuration(gxPre) - gz.fallTime - gz.flatTime/2 ...
36 - mr.calcDuration(gx)/2)/seq.gradRasterTime)*seq.gradRasterTime;
37 delayTR=ceil((TR - mr.calcDuration(gxPre) - mr.calcDuration(gz) ...
38 - mr.calcDuration(gx) - delayTE)/seq.gradRasterTime)*seq.gradRasterTime;
39 assert(all(delayTR>=mr.calcDuration(gxSpoil, gzSpoil)));
40
41 rf_phase=0;
42 rf_inc=0;
43
44 for i=(-Ndummy):Nr
45     for c=1:length(TE)
46         rf.phaseOffset=rf_phase/180*pi;
47         adc.phaseOffset=rf_phase/180*pi;
48         rf_inc=mod(rf_inc+rfSpoilingInc, 360.0);
49         rf_phase=mod(rf_phase+rf_inc, 360.0);
50         %
51         seq.addBlock(rf,gz);
52         phi=delta*(i-1);
53         seq.addBlock(mr.rotate('z', phi, gxPre, gzReph));
54         seq.addBlock(mr.makeDelay(delayTE(c)));
55         if (i>0)
56             seq.addBlock(mr.rotate('z', phi, gx, adc));
57         else
58             seq.addBlock(mr.rotate('z', phi, gx));
59         end
60         seq.addBlock(mr.rotate('z', phi, gxSpoil, gzSpoil, mr.makeDelay(delayTR)));
61     end
62 end
63 seq.plot();
64
UTF-8 script Ln 10 Col 1
```

Pulseseq – seq07 contd.

- *Time to explore sequence timing and the trajectory (e.g. RF phase and the gradient spoiling effects on the trajectory)*



Pulseq – recon scripts

- All recon scripts are similar
- General logic
 - We expect data to be stored in a directory of choice (line 7 in this example points to data)
 - Raw data MUST always be accompanied with a Pulseq file with the identical name
 - By default all scripts load the most recent data
 - To load the second last data set replace 'end-0' with 'end-1' (line 14)
 - Ideally we expect raw data in a 3D array with dimensions: **samples x channels x readouts**
 - All data counters, data sorting and similar parameters is calculated from the Pulseq file
- Here is the complete FID/SE example

file number
selector

data path

fix some
data sets

special data
processing
begins here

```
/home/zaitsev/pulseq_home/pulseq_public/matlab/ismrmTutorial/recon/recon01_FID.m
EDITOR PUBLISH VIEW
New Open Save Find Files Compare Go To Insert Comment % fx Breakpoints Run Run and Advance Run Section Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN

1 % very basic FID data handling example
2 %
3 % it loads Matlab .mat files with the rawdata in the format
4 %   adcLen x channels x readouts
5 % it also seeks an accompanying .seq file with the same name to interpret
6 % the data
7
8 %% Load the latest file from the specified directory
9 path='/data/ismrm2021demo'; % directory to be scanned for data files
10
11 pattern='*.mat';
12 D=dir([path filesep pattern]);
13 [~,I]=sort([D(:).datenum]);
14 data_file_path=[path D(I(end-0)).name] % use end-1 to reconstruct the second-last data set, etc...
15
16 data_unsorted = load(data_file_path);
17
18 if isstruct(data_unsorted)
19     fn=fieldnames(data_unsorted);
20     assert(length(fn)==1); % we only expect a single variable
21     data_unsorted=data_unsorted.(fn{1});
22 end
23
24 %% Load sequence from file
25 seq = mr.Sequence(); % Create a new sequence object
26 seq_file_path = [data_file_path(1:end-3) 'seq'];
27 seq.read(seq_file_path); % no because it conflicts with the FID multiple FAS
28 %seq.read(seq_file_path,'detectRFuse'); % this is a better option for the SE sequences
29
30 %% raw data preparation
31
32 if ndims(data_unsorted)<3 && size(data_unsorted,1)==1
33     % OCRA data may need some fixing
34     [~,~,eventCount]=seq.duration();
35     readouts=eventCount(6);
36     data_unsorted=reshape(data_unsorted,[length(data_unsorted)/readouts,1,readouts]);
37 elseif ndims(data_unsorted)==2 && size(data_unsorted,2)>1
38     data_unsorted=reshape(data_unsorted,[size(data_unsorted,1),1,size(data_unsorted,2)]);
39 end
40
41 [adc_len,channels,readouts]=size(data_unsorted);
42 rawdata = permute(data_unsorted, [1,3,2]); % channels last
43
44 %% we just want time stamps: t_adc, t_excitation
45 [~,t_adc,~,~,t_excitation]=seq.calculateKspacePP();
46
47 t_excitation_adc=t_excitation((end-readouts+1):end); % remove dummy/prep scans
48 t_adc=reshape(t_adc,[adc_len,readouts]);
49 t_e=t_adc-t_excitation_adc(ones(1,adc_len),:);
50
51 % plot raw data
52 figure; plot(t_e, abs(rawdata(:,:))); title('raw signal(s)'); xlabel('time /s');
53 hold on; plot(t_e(1),0); % trick to force Y axis scaling
54
55 if (readouts>1)
56     figure; plot(abs(rawdata(1,:))); title('time evolution (first FID point)');
57     dataavg=squeeze(mean(rawdata,2));
58     figure; plot(t_e, abs(dataavg)); title('averaged FIDs');
59 end
60
61 % calculate frequency axis
62 f=zeros(size(t_e));
63 adc_dur=(t_adc(end,1)-t_adc(1,1))/(adc_len-1)*adc_len;
64 f(:)=1/adc_dur;
65 f=cumsum(f,1)-0.5/adc_dur*adc_len;
66 % plot spectra
67 figure; plot(f, abs(fftshift(fft(fftshift(rawdata(:,:),1),1))))); title('spectra(um/a)'); xlabel('frequency /Hz');
68
```


Pulseq – 2D FFT

- The beginning of the script is similar to the FID/SE example
- Screen shot shows automatic counter-free data sorting part
- k-space trajectory automatically updates data reordering for (almost) any 2D Cartesian sequence
- *Following (not shown) part is a trivial 2D FFT and a sum-of-square combination for multichannel data*

k-space trajectory

remove slice dimension

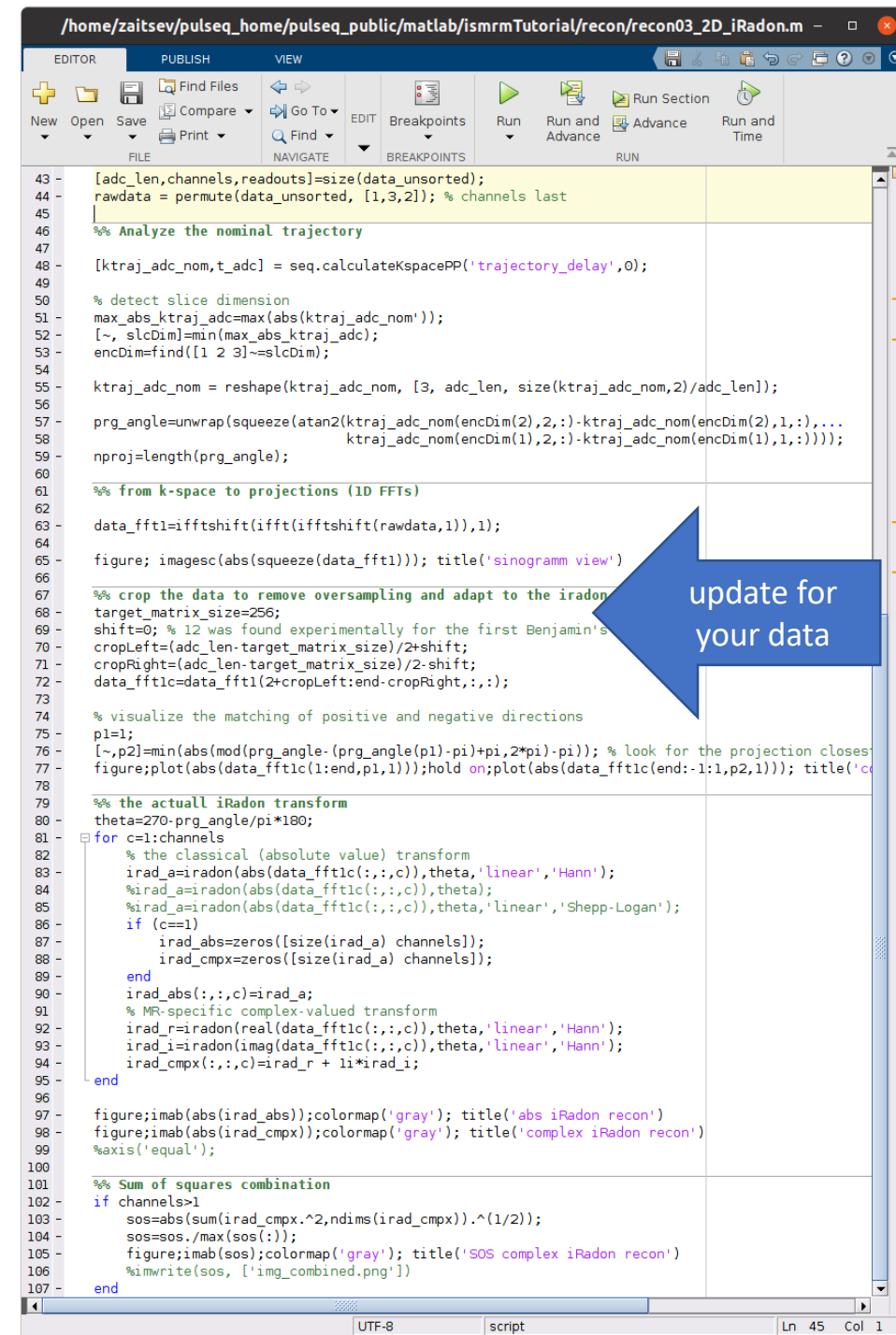
calculate reordering

sort the data

```
42 [adc_len, channels, readouts] = size(data_unsorted);
43 % the incoming data order is [kx coils acquisitions]
44 data_coils_last = permute(data_unsorted, [1, 3, 2]);
45
46 %% Plot and analyze the trajectory data (ktraj_adc)
47
48 [ktraj_adc, ktraj, t_excitation, t_refocusing, t_adc] = seq.calculateKspace();
49 figure; plot(ktraj(1,:), ktraj(2,:), 'b', ...
50             ktraj_adc(1,:), ktraj_adc(2,:), 'r'); % a 2D plot
51 axis('equal'); title('2D k-space trajectory');
52
53 % try to detect the data ordering
54 k_extent = max(abs(ktraj_adc), [], 2);
55 k_scale = max(k_extent);
56 k_threshold = k_scale/5000;
57
58 % detect unused dimensions and delete them
59 if any(k_extent < k_threshold)
60     ktraj_adc(k_extent < k_threshold, :) = []; % delete rows
61     k_extent(k_extent < k_threshold) = [];
62 end
63
64 % detect dk, k-space reordering and repetitions (or slices, etc)
65 kt_sorted = sort(ktraj_adc, 2);
66 dk_all = kt_sorted(:, 2:end) - kt_sorted(:, 1:(end-1));
67 dk_all(dk_all < k_threshold) = NaN;
68 dk_min = min(dk_all, [], 2);
69 dk_max = max(dk_all, [], 2);
70 dk_all(dk_all < dk_min, :ones(1, size(dk_all, 2))) > k_threshold = NaN;
71 dk_all_cnt = sum(isfinite(dk_all), 2);
72 dk_all(~isfinite(dk_all)) = 0;
73 dk = sum(dk_all, 2) ./ dk_all_cnt;
74 [~, k0_ind] = min(sum(ktraj_adc.^2, 1));
75 kindex = round((ktraj_adc - ktraj_adc(:, k0_ind * ones(1, size(ktraj_adc, 2)))) ./ dk(:, ones(1, size(ktraj_adc, 2)))));
76 kindex_min = min(kindex, [], 2);
77 kindex_mat = kindex - kindex_min(:, ones(1, size(ktraj_adc, 2))) + 1;
78 kindex_end = max(kindex_mat, [], 2);
79 sampler = zeros(kindex_end);
80 repeat = zeros(1, size(ktraj_adc, 2));
81 for i = 1:size(kindex_mat, 2)
82     if (size(kindex_mat, 1) == 3)
83         ind = sub2ind(kindex_end, kindex_mat(1, i), kindex_mat(2, i), kindex_mat(3, i));
84     else
85         ind = sub2ind(kindex_end, kindex_mat(1, i), kindex_mat(2, i));
86     end
87     repeat(i) = sampler(ind);
88     sampler(ind) = repeat(i) + 1;
89 end
90 if (max(repeat(:)) > 0)
91     kindex = [kindex; (repeat + 1)];
92     kindex_mat = [kindex_mat; (repeat + 1)];
93     kindex_end = max(kindex_mat, [], 2);
94 end
95 %figure; plot(kindex(1,:), kindex(2,:), 'r');
96
97 %% sort the k-space data into the data matrix
98 data_coils_last = reshape(data_coils_last, [adc_len * readouts, channels]);
99
100 data = zeros([kindex_end, channels]);
101 if (size(kindex, 1) == 3)
102     for i = 1:size(kindex, 2)
103         data(kindex_mat(1, i), kindex_mat(2, i), kindex_mat(3, i), :) = data_coils_last(i, :);
104     end
105 else
106     for i = 1:size(kindex, 2)
107         data(kindex_mat(1, i), kindex_mat(2, i), :) = data_coils_last(i, :);
108     end
109 end
110
111
```

Pulseq – iRadon

- The beginning of the script is similar to the FID/SE/2D FFT examples
- Projection directions are calculated from the Pulseq sequence file
- Matrix size and projection data centering needs to be adjusted manually
- Example also shows a comparison between absolute value and complex inverse Radon transformations
 - Check out the gradient-echo data to notice the difference

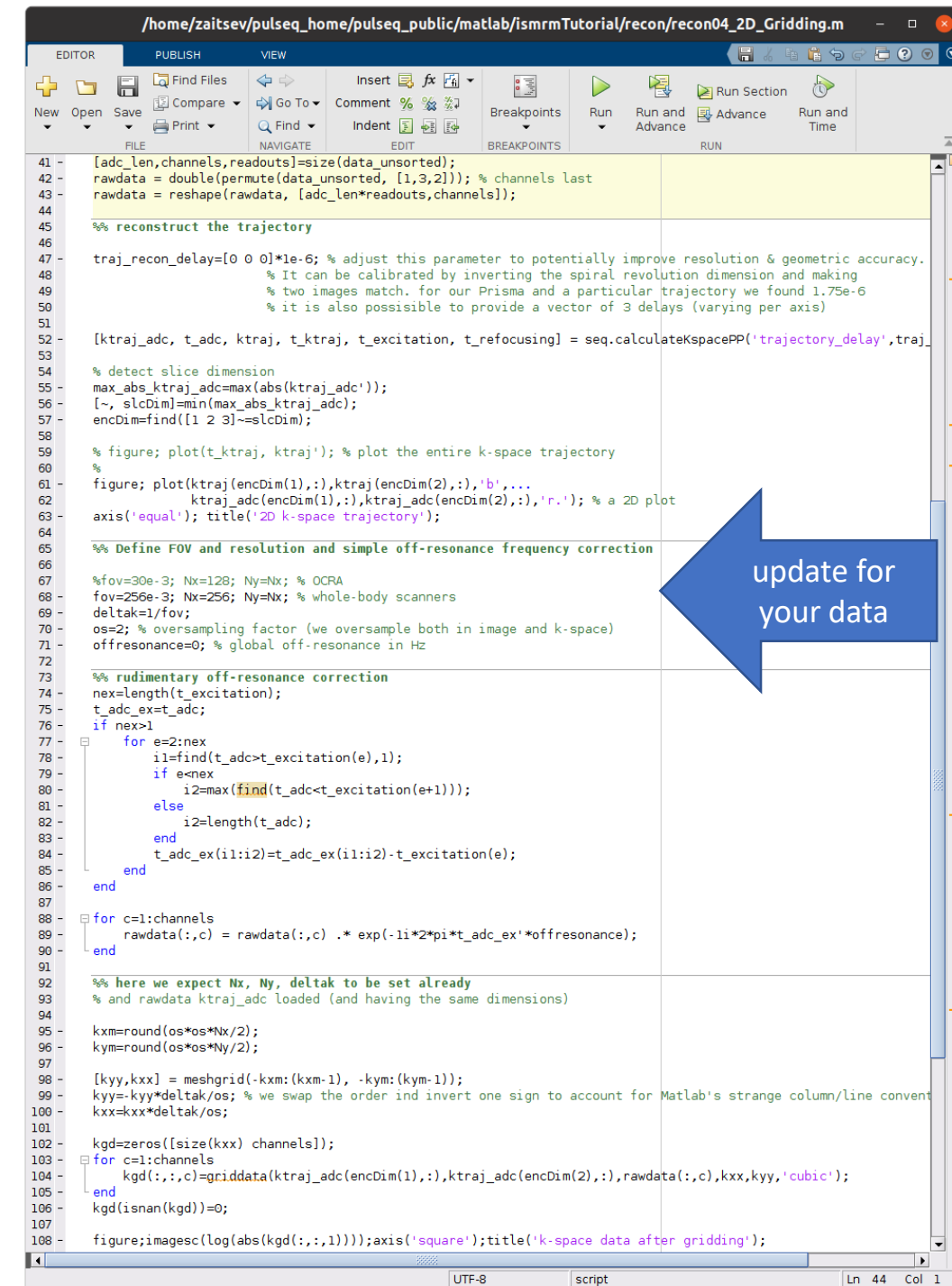


```
43 - [adc_len,channels,readouts]=size(data_unsorted);
44 - rawdata = permute(data_unsorted, [1,3,2]); % channels last
45 -
46 - %% Analyze the nominal trajectory
47 -
48 - [ktraj_adc_nom,t_adc] = seq.calculateKspacePP('trajectory_delay',0);
49 -
50 - % detect slice dimension
51 - max_abs_ktraj_adc=max(abs(ktraj_adc_nom));
52 - [~, slcDim]=min(max_abs_ktraj_adc);
53 - encDim=find([1 2 3]~=slcDim);
54 -
55 - ktraj_adc_nom = reshape(ktraj_adc_nom, [3, adc_len, size(ktraj_adc_nom,2)/adc_len]);
56 -
57 - prg_angle=unwrap(squeeze(atan2(ktraj_adc_nom(encDim(2),2,:)-ktraj_adc_nom(encDim(2),1,:),...
58 -                               ktraj_adc_nom(encDim(1),2,:)-ktraj_adc_nom(encDim(1),1,:))));
59 - nproj=length(prg_angle);
60 -
61 - %% from k-space to projections (1D FFTs)
62 -
63 - data_fftl=ifftshift(ifft(ifftshift(rawdata,1),1),1);
64 -
65 - figure; imagesc(abs(squeeze(data_fftl))); title('sinogramm view')
66 -
67 - %% crop the data to remove oversampling and adapt to the iradon
68 - target_matrix_size=256;
69 - shift=0; % 12 was found experimentally for the first Benjamin's
70 - cropLeft=(adc_len-target_matrix_size)/2+shift;
71 - cropRight=(adc_len-target_matrix_size)/2-shift;
72 - data_fftlc=data_fftl(2+cropLeft:end-cropRight,:,:);
73 -
74 - % visualize the matching of positive and negative directions
75 - p1=1;
76 - [~,p2]=min(abs(mod(prg_angle-(prg_angle(p1)-pi)+pi,2*pi)-pi)); % look for the projection closes
77 - figure;plot(abs(data_fftlc(1:end,p1,1)));hold on;plot(abs(data_fftlc(end:-1:1,p2,1))); title('cc
78 -
79 - %% the actual iRadon transform
80 - theta=270-prg_angle/pi*180;
81 - for c=1:channels
82 -     % the classical (absolute value) transform
83 -     irad_a=iradon(abs(data_fftlc(:,:,c)),theta,'linear','Hann');
84 -     %irad_a=iradon(abs(data_fftlc(:,:,c)),theta);
85 -     %irad_a=iradon(abs(data_fftlc(:,:,c)),theta,'linear','Shepp-Logan');
86 -     if (c==1)
87 -         irad_abs=zeros([size(irad_a) channels]);
88 -         irad_cmpx=zeros([size(irad_a) channels]);
89 -     end
90 -     irad_abs(:,:,c)=irad_a;
91 -     % MR-specific complex-valued transform
92 -     irad_r=iradon(real(data_fftlc(:,:,c)),theta,'linear','Hann');
93 -     irad_i=iradon(imag(data_fftlc(:,:,c)),theta,'linear','Hann');
94 -     irad_cmpx(:,:,c)=irad_r + 1i*irad_i;
95 - end
96 -
97 - figure;imab(abs(irad_abs));colormap('gray'); title('abs iRadon recon')
98 - figure;imab(abs(irad_cmpx));colormap('gray'); title('complex iRadon recon')
99 - %axis('equal');
100 -
101 - %% Sum of squares combination
102 - if channels>1
103 -     sos=abs(sum(irad_cmpx.^2,ndims(irad_cmpx)).^(1/2));
104 -     sos=sos./max(sos(:));
105 -     figure;imab(sos);colormap('gray'); title('SOS complex iRadon recon')
106 -     %imwrite(sos, ['img_combined.png'])
107 - end
```

update for your data

Pulseq – 2D Gridding

- The beginning of the script is similar to the FID/SE/2D FFT examples
- Calculate the sampling trajectory from the Pulseq sequence file
 - Any crazy trajectory can be reconstructed
 - *Also 2D Cartesian data are compatible*
- FOW and imaging matrix size may need to be adjusted manually
- The screen shot ends with showing the gridded k-space data
- *Following (not shown) part is a trivial 2D FFT and a sum-of-square combination for multichannel data*



```
41 [adc_len,channels,readouts]=size(data_unsorted);
42 rawdata = double(permute(data_unsorted, [1,3,2])); % channels last
43 rawdata = reshape(rawdata, [adc_len*readouts,channels]);
44
45 %% reconstruct the trajectory
46
47 traj_recon_delay=[0 0]*1e-6; % adjust this parameter to potentially improve resolution & geometric accuracy.
48 % It can be calibrated by inverting the spiral revolution dimension and making
49 % two images match. for our Prisma and a particular trajectory we found 1.75e-6
50 % it is also possible to provide a vector of 3 delays (varying per axis)
51
52 [ktraj_adc, t_adc, ktraj, t_ktraj, t_excitation, t_refocusing] = seq.calculateKspacePP('trajectory_delay',traj_
53
54 % detect slice dimension
55 max_abs_ktraj_adc=max(abs(ktraj_adc'));
56 [~, slcDim]=min(max_abs_ktraj_adc);
57 encDim=find([1 2 3]~=slcDim);
58
59 % figure; plot(t_ktraj, ktraj'); % plot the entire k-space trajectory
60 %
61 figure; plot(ktraj(encDim(1,:),:),ktraj(encDim(2,:),:),'b',...
62 ktraj_adc(encDim(1,:),:),ktraj_adc(encDim(2,:),:),'r'); % a 2D plot
63 axis('equal'); title('2D k-space trajectory');
64
65 %% Define FOV and resolution and simple off-resonance frequency correction
66
67 %fov=30e-3; Nx=128; Ny=Nx; % OCRA
68 fov=256e-3; Nx=256; Ny=Nx; % whole-body scanners
69 deltak=1/fov;
70 os=2; % oversampling factor (we oversample both in image and k-space)
71 offresonance=0; % global off-resonance in Hz
72
73 %% rudimentary off-resonance correction
74 nex=length(t_excitation);
75 t_adc_ex=t_adc;
76 if nex>1
77     for e=2:nex
78         i1=find(t_adc>t_excitation(e),1);
79         if e<nex
80             i2=max(find(t_adc<t_excitation(e+1)));
81         else
82             i2=length(t_adc);
83         end
84         t_adc_ex(i1:i2)=t_adc_ex(i1:i2)-t_excitation(e);
85     end
86 end
87
88 for c=1:channels
89     rawdata(:,c) = rawdata(:,c) .* exp(-1i*2*pi*t_adc_ex'*offresonance);
90 end
91
92 %% here we expect Nx, Ny, deltak to be set already
93 % and rawdata ktraj_adc loaded (and having the same dimensions)
94
95 kxm=round(os*os*Nx/2);
96 kym=round(os*os*Ny/2);
97
98 [kyy,kxx] = meshgrid(-kxm:(kxm-1), -kym:(kym-1));
99 kyy=-kyy*deltak/os; % we swap the order and invert one sign to account for Matlab's strange column/line convent
100 kxx=kxx*deltak/os;
101
102 kgd=zeros([size(kxx) channels]);
103 for c=1:channels
104     kgd(:,c)=griddata(ktraj_adc(encDim(1,:),:),ktraj_adc(encDim(2,:),:),rawdata(:,c),kxx,kyy,'cubic');
105 end
106 kgd(isnan(kgd))=0;
107
108 figure;imagesc(log(abs(kgd(:,1,1))));axis('square');title('k-space data after gridding');
```


Outline

- *Pulseseq* sequence definition and programming language overview
- Using *Pulseseq* on different types of scanners:
 - Siemens, GE, and tabletop scanners using OCRA
- Live demo of *Pulseseq* across sites
 1. Basic pulse-acquire scans: Free Induction Decay (FID), Spin-Echo (SE)
 2. Cartesian spin-eco imaging
 3. Radial imaging (spin-echo and gradient echo)
 4. *Advanced topics (gradient delays and correction, etc...)*
are only covered in the live portion of the tutorial

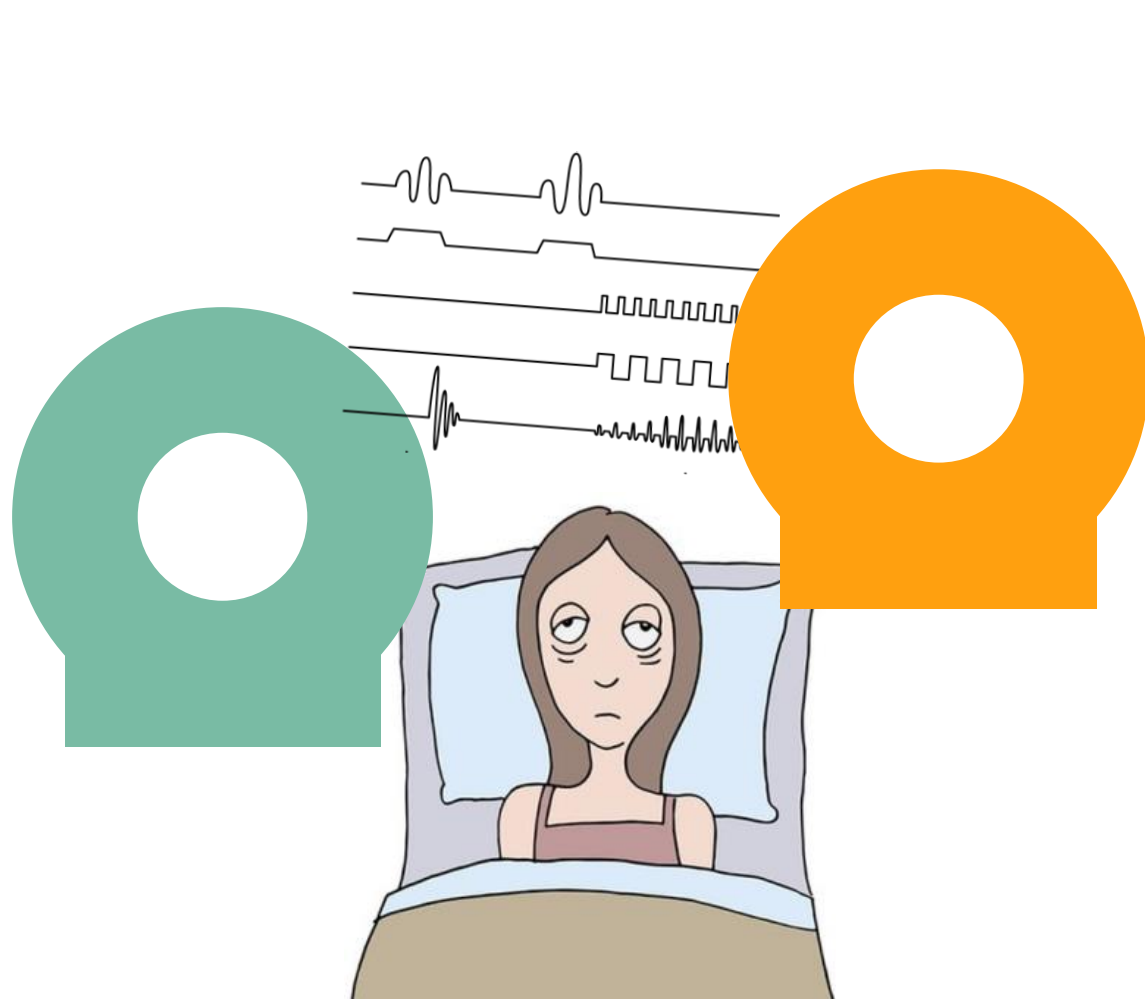
How to reach us

Site	<i>Pulseq</i> Users & Contact Info
Vienna 7 T Siemens	Maxim Zaitsev <maxim.zaitsev@meduniwien.ac.at>
Michigan 3 T GE	Jon-Fredrik Nielsen <jfnielse@umich.edu> Melissa Haskell <mhask@umich.edu>
Dortmund 0.48 T	Benjamin Menkuec <benjamin.menkuec@fh-dortmund.de>
Boston 0.35 T	Lincoln Craven-Brightman: <lcraven-brightman@mgh.harvard.edu> Jason Stockmann <jstockmann@mgh.harvard.edu>
Magdeburg 0.27 T	Marcus Prier <marcus.prier@ovgu.de> David Schote <david.schote@st.ovgu.de>
Leiden 50 mT	Tom O'Reilly <t.o_reilly@lumc.nl>

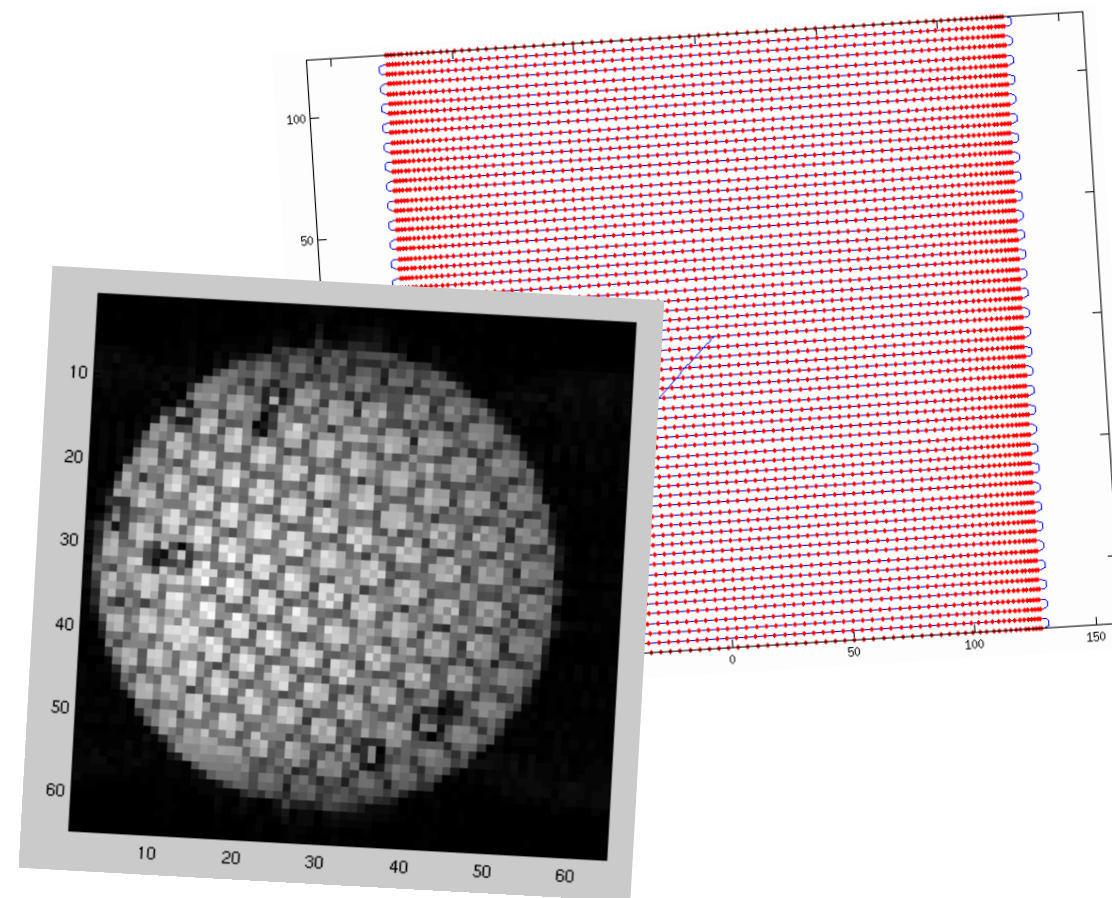


Join us on Wed May 19 at 19:00 UTC

 **Pulseseq** – that's the way you do it!



...dream of a sequence in the morning...



...check the images in the afternoon!