mrsd

Julien Lamy

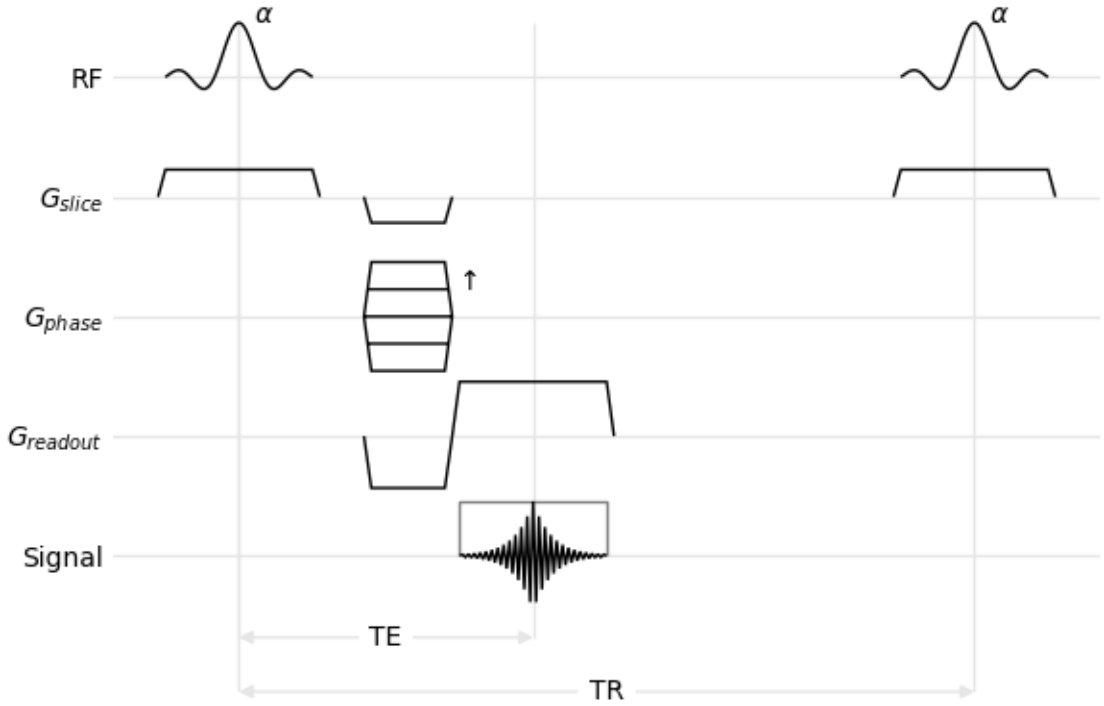
Mar 16, 2023

CONTENTS:

mrsd is a Python toolkit to generate MR sequence diagrams, available on PyPi. To install from source, the only dependencies are Matplotlib and numpy.

To get started, have a look at the tutorials (*FLASH* and *multi-echo FLASH*) and the examples. Once familiar with the basic concepts, you can find more details in the *API Documentation*.

Rather use MATLAB than Python? mrisd is the answer!



CHAPTER

ONE

FLASH TUTORIAL

This example shows how to draw the sequence diagram of the 2D FLASH sequence, as show, in FLASH imaging. Rapid NMR imaging using low flip-angle pulses, Haase et al., Journal of Magnetic Resonance, 67(2), pp. 258-266, 1986.

Start by creating two matplotlib objects, a figure and a plot, then create an empty sequence diagram which will be drawn in the matplotlib plot. The second argument is a list of *channels*: horizontal time lines for the various sequence events. Channel names may contain math expression inside dollar characters: they will be interpreted according to LaTeX rules. Channel names are arbitrary, and carry no specific meaning.

```
import matplotlib.pyplot
import mrsd

figure, plot = matplotlib.pyplot.subplots(figsize=(4,4), tight_layout=True)
diagram = mrsd.Diagram(
    plot, ["RF", "$G_{slice}$", "$G_{phase}$", "$G_{readout}$", "Signal"])
```

1.1 Slice-selective Pulse

The 2D FLASH sequence starts with a slice-selective RF pulse, i.e. an RF pulse played concurrently with a gradient on the slice axis.

Within *mrsd*, everything that happens during a sequence (RF pulses, gradients, echoes, etc.) is called an *event*. Each event has a duration and a position in time, which can be specified by the `begin`, `end` or `center` of the event. This can be used to synchronize events: the RF pulse is centered on 0, and the slice-selection gradient is centered on the RF pulse.

We start by creating the *RFPulse* with a duration of 2 (time units are arbitrary), and an amplitude of 1 (amplitudes of events are normalized between -1 and 1). We then create the *Gradient*, with a flat-top centered on the pulse, and an amplitude of 0.5. Once created, those two objects are added (*add()*) to their respective channels.

```
pulse = mrsd.RFPulse(2, 1, center=0)
slice_selection = mrsd.Gradient(pulse.duration, 0.5, center=pulse.center)

diagram.add("RF", pulse)
diagram.add("$G_{slice}$", slice_selection)
```

It is also possible to directly add object to the diagram by calling the appropriate functions of the *Diagram* class: *rf_pulse()* and *gradient()*.

Fig. 1: Empty sequence diagram

```
pulse = diagram.rf_pulse("RF", 2, 1, center=0)
slice_selection = diagram.gradient(
    "$G_{slice}$", pulse.duration, 0.5, center=pulse.center)
```

Ramp times can be added to the gradients, using either the `ramp` parameter (symmetric ramp-up and ramp-down times), or both the `ramp_up` and `ramp_down` parameters (asymmetric gradients). These parameters can be used in both forms of gradient creation.

Since selective pulses are a common pattern, a convenience function (*selective_pulse()*) exists to create both objects:

```
pulse, slice_selection = diagram.selective_pulse(
    "RF", "$G_{slice}$", 2, gradient_amplitude=0.5, ramp=0.1, center=0)
```

The envelope of RF-pulses can be tuned using the `envelope` parameter: it defaults to `sinc`, but `box` or `gaussian` can also be used. You can also use the convenience functions *sinc_pulse()*, *hard_pulse()*, and *gaussian_pulse()* of the *Diagram* objects.

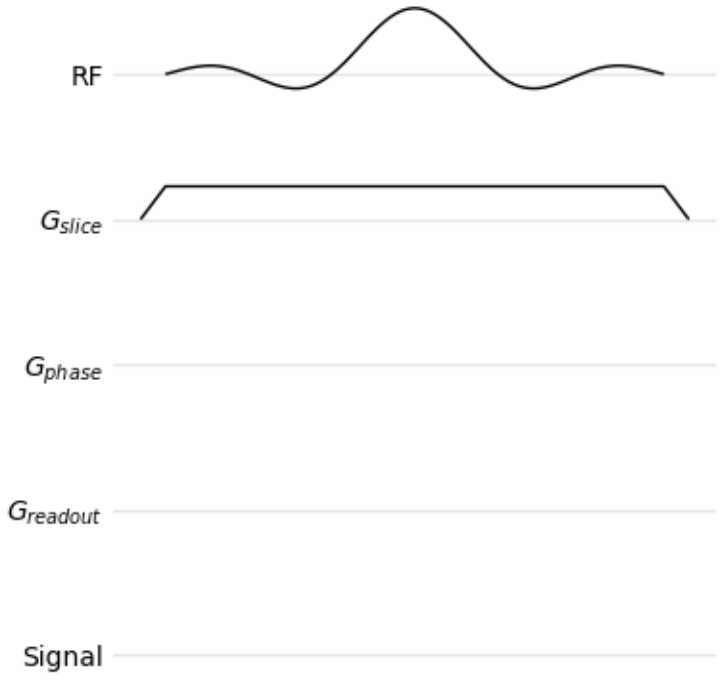


Fig. 2: Slice-selective pulse

1.2 Readout

The readout occurs at echo time, and comprises three events: the *ADC* being switched on and off, the *Echo*, and the readout gradient. As for the selective pulse, the objects can be created then added to the diagram. Note that the ADC object takes an extra parameter, `ec`: this is passed to matplotlib and can be used to change the aspect of the drawn object (color, line style, transparency, etc.).

```
TE = 4
d_readout = 2

adc = mrsd.ADC(d_readout, center=TE, ec="0.5")
echo = mrsd.Echo(d_readout, 1, center=adc.center)
readout = mrsd.Gradient(d_readout, 1, ramp=0.1, center=adc.center)

diagram.add("Signal", adc)
diagram.add("Signal", echo)
diagram.add("$G_{readout}$", readout)
```

The readout events can also be added using functions from the *Diagram* class: *adc()* and *mrsd.Diagram.echo()*.

```
adc = diagram.adc("Signal", d_readout, center=TE, ec="0.5")
echo = diagram.echo("Signal", d_readout, 1, center=adc.center)
readout = diagram.gradient(
    "$G_{readout}$", d_readout, 1, ramp=0.1, center=adc.center)
```

As for the selective pulse, the readout is a common pattern for which a convenience function (*readout()*) can create

all three events.

```
adc, echo, readout = diagram.readout(
    "Signal", "$G_{readout}$", d_readout, ramp=0.1, center=TE,
    adc_kwargs={"ec": "0.5"})
```

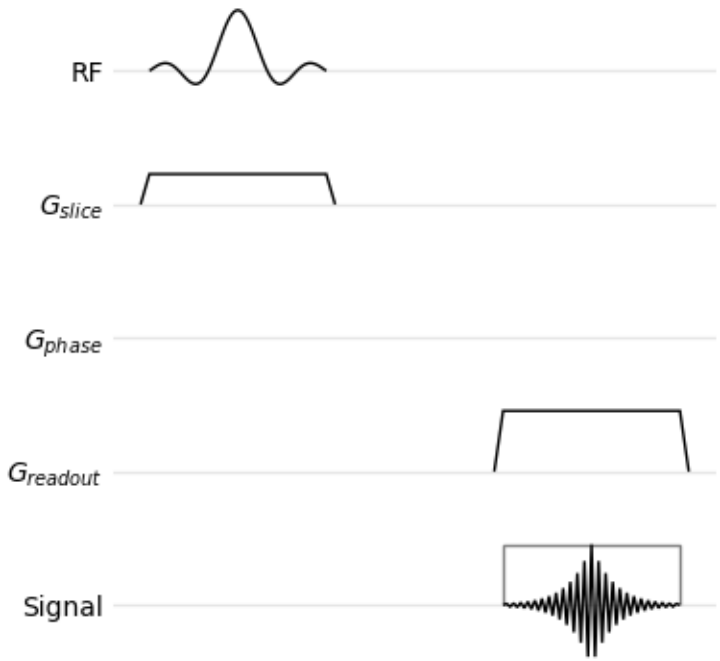


Fig. 3: Pulse and readout

1.3 Encoding Gradients

The phase-encoding gradient is displayed as overlayed gradient lobes to represent the different repetitions of the sequence: in *mrsd*, this is called a *MultiGradient*.

```
d_encoding = 1

phase_encoding = mrsd.MultiGradient(d_encoding, 1, 0.1, end=readout.begin)
diagram.add("$G_{phase}$", phase_encoding)
```

As for the other events, it can be directly added using a function from the *Diagram* class (*multi_gradient()*).

```
diagram.multi_gradient("$G_{phase}$", d_encoding, 1, 0.1, end=readout.begin)
```

The prephasing lobe of the readout gradient depends on the readout gradient itself: its area must be minus one-half that of the main readout lobe. *Gradient* events have an *adapt()* function which create a new gradient from an existing one and an area ratio.

```
readout_prephasing = readout.adapt(d_encoding, -0.5, 0.1, end=readout.begin)
diagram.add("$G_{readout}$", readout_prephasing)
```

Similarly, the slice-encoding gradient must be rewound.

```
diagram.add(
    "$G_{slice}$",
    slice_selection.adapt(d_encoding, -0.5, 0.1, end=readout.begin))
```

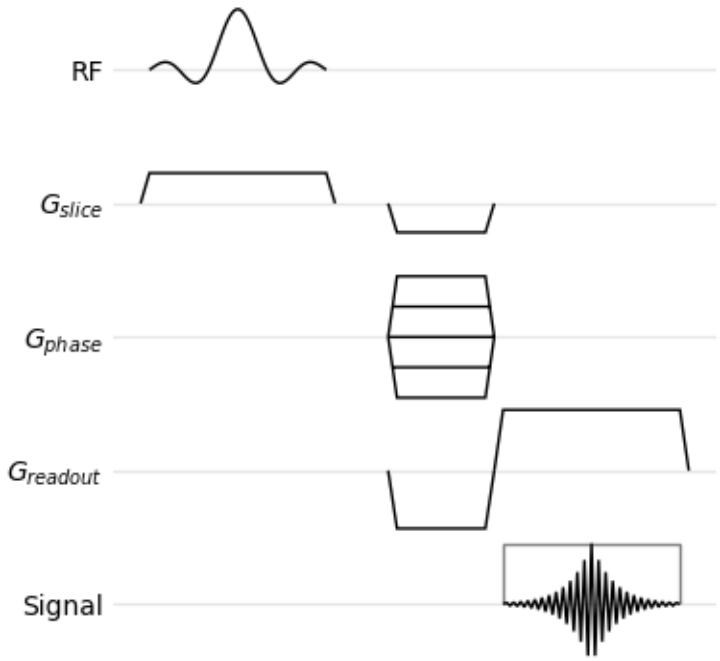


Fig. 4: Pulse, readout, and encoding gradients

1.4 Annotations and Copies

The sequence diagram can be supplemented with annotations for a better understanding of the sequence. Time intervals (*interval()*) can show the TE and TR.

```
TR = 10

diagram.interval(0, TE, -1.5, "TE")
diagram.interval(0, TR, -2.5, "TR")
```

Channel specific annotations (e.g. flip angles and direction of phase encoding) are added using *annotate()*.

```
diagram.annotate("RF", 0.2, 1, r"$\alpha$")
diagram.annotate("$G_{phase}$", phase_encoding.end, 0.5, r"$\uparrow$")
```

Finally, we show the beginning of the next repetition by creating copies of the RF pulse and the slice-selection gradient and moving them by the value of TR (*move()*).

```
diagram.add("RF", copy.copy(pulse).move(TR))
diagram.add("$G_{slice}$", copy.copy(slice_selection).move(TR))
diagram.annotate("RF", TR+0.2, 1, r"$\alpha$")
```

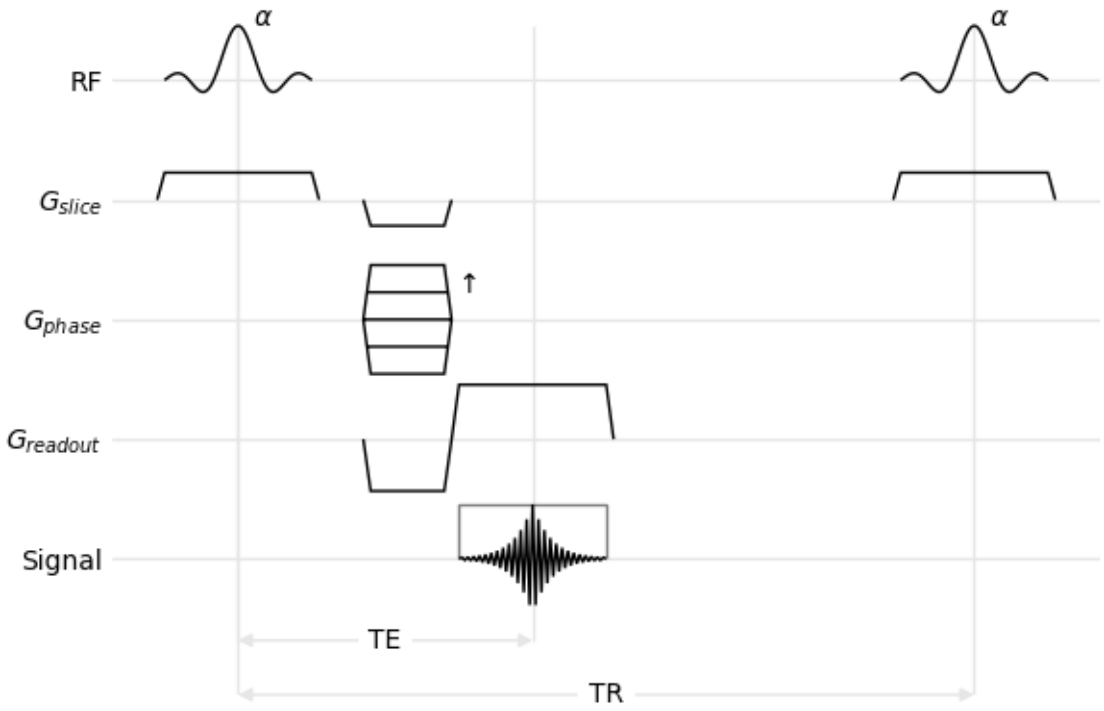


Fig. 5: Full diagram of the FLASH sequence

CHAPTER

TWO

MULTI-ECHO FLASH TUTORIAL

This tutorial extends the *FLASH Tutorial* to show a more complex readout pattern with alternating polarities and how to add other matplotlib objects to a sequence diagram.

We will start from a simple diagram with only three channels, the RF pulses, the readout gradient and the echoes.

```
import copy

import matplotlib.pyplot
import mrsd
import numpy

figure, plot = matplotlib.pyplot.subplots(tight_layout=True)
diagram = mrsd.Diagram(plot, ["RF", "$G_{RO}$", "Echoes"])
```

We will then define sequence parameters, add the RF pulse and show the TR. Note that this diagram uses a rectangular pulse instead of a sinc one.

```
T2_star = 5
TE, TR = 4, 20
d_pulse, d_readout, d_ramp = 0.5, 1, 0.1
train_length = 10

pulse = diagram.hard_pulse("RF", d_pulse, 1, center=0)
diagram.add("RF", copy.copy(pulse).move(TR))
diagram.interval(0, TR, -1.5, "TR")
```

We then add the first echo, its readout gradient and the prephasing lobe of the readout gradient. In this example, we do not show the ADC to make the diagram more readable.

```
echo = diagram.echo("Echoes", d_readout, 1, center=TE)
readout = diagram.gradient("$G_{RO}$", d_readout, 1, d_ramp, center=echo.center)
diagram.add(
    "$G_{RO}$", readout.adapt(d_readout, -0.5, d_ramp, end=readout.begin))
```

This part is very similar to the *FLASH Tutorial* and yields the following diagram.

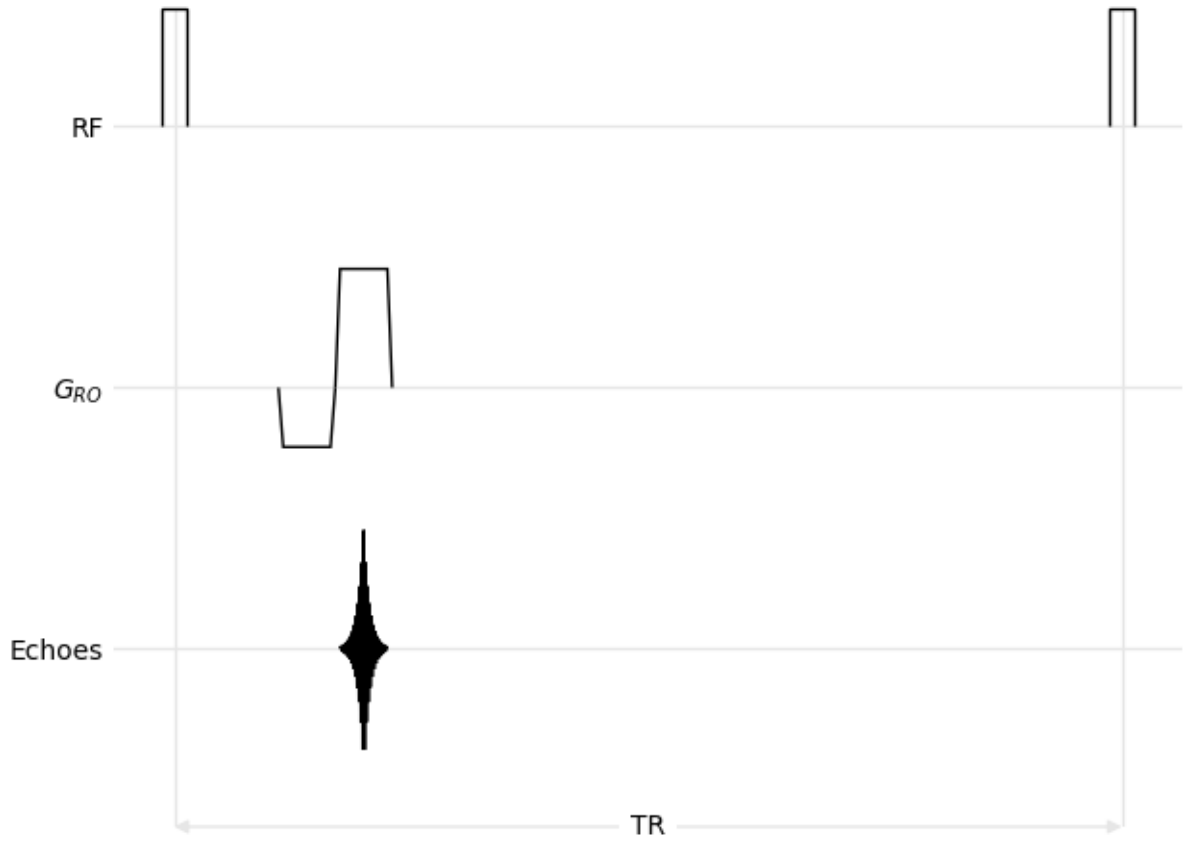


Fig. 1: Single echo

2.1 Multiple Echoes

In this tutorial, the sequence uses bipolar readout gradients and echoes as close as possible with respect to the gradient ramps. This means that the beginning of a readout gradient is the end of a previous one, and that the center of each echo is located at the center of its corresponding gradient lobe. We also modulate the echo amplitudes with a T_2^* decay.

```
for echo in range(1, train_length):
    gradient_amplitude = (-1)**echo
    readout = diagram.gradient(
        "$G_{RO}$", d_readout, gradient_amplitude, ramp=d_ramp,
        begin=readout.end)

    elapsed = readout.center - TE
    echo_amplitude = numpy.exp(-elapsed/T2_star)
    diagram.echo("Echoes", d_readout, echo_amplitude, center=readout.center)
```

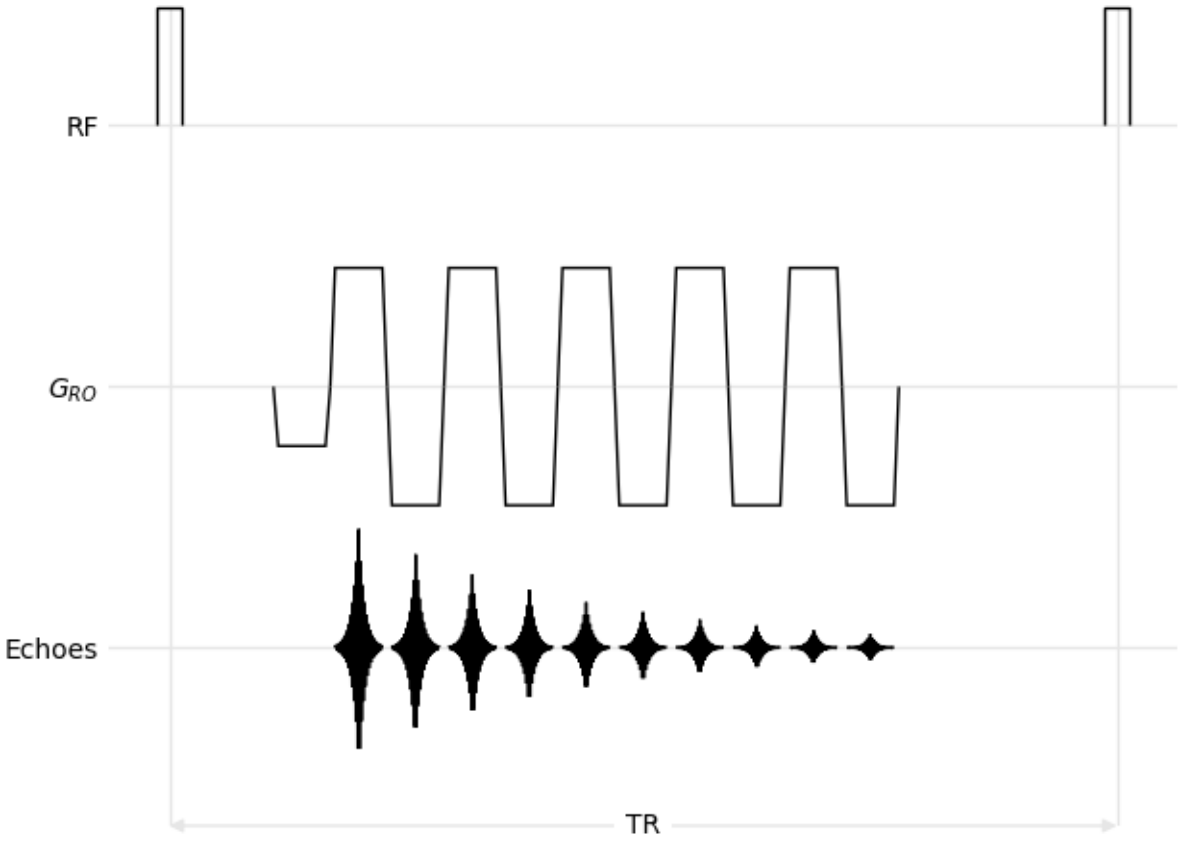


Fig. 2: Multiple echoes

2.2 T_2* Decay

Finally, we show a continuous T_2^* decay curve overlayed on top of the echoes. This is achieved by calling the usual matplotlib functions of the `plot` object, with the curve beginning at the first echo located at `TE` and ending at the last echo, using the last `readout` object. The vertical position must set to the RF channel, which can be accessed using the *`y()`* function of the *`Diagram`* class.

```
xs = numpy.linspace(0, readout.end-TE, 100)
ys = numpy.exp(-xs/T2_star)
plot.plot(xs+TE, ys+diagram.y("Echoes"), color="C0", lw=1)
plot.text(TE+xs[len(xs)//2], 0.5, "$e^{-t/T_2^*}$", color="C0")
```

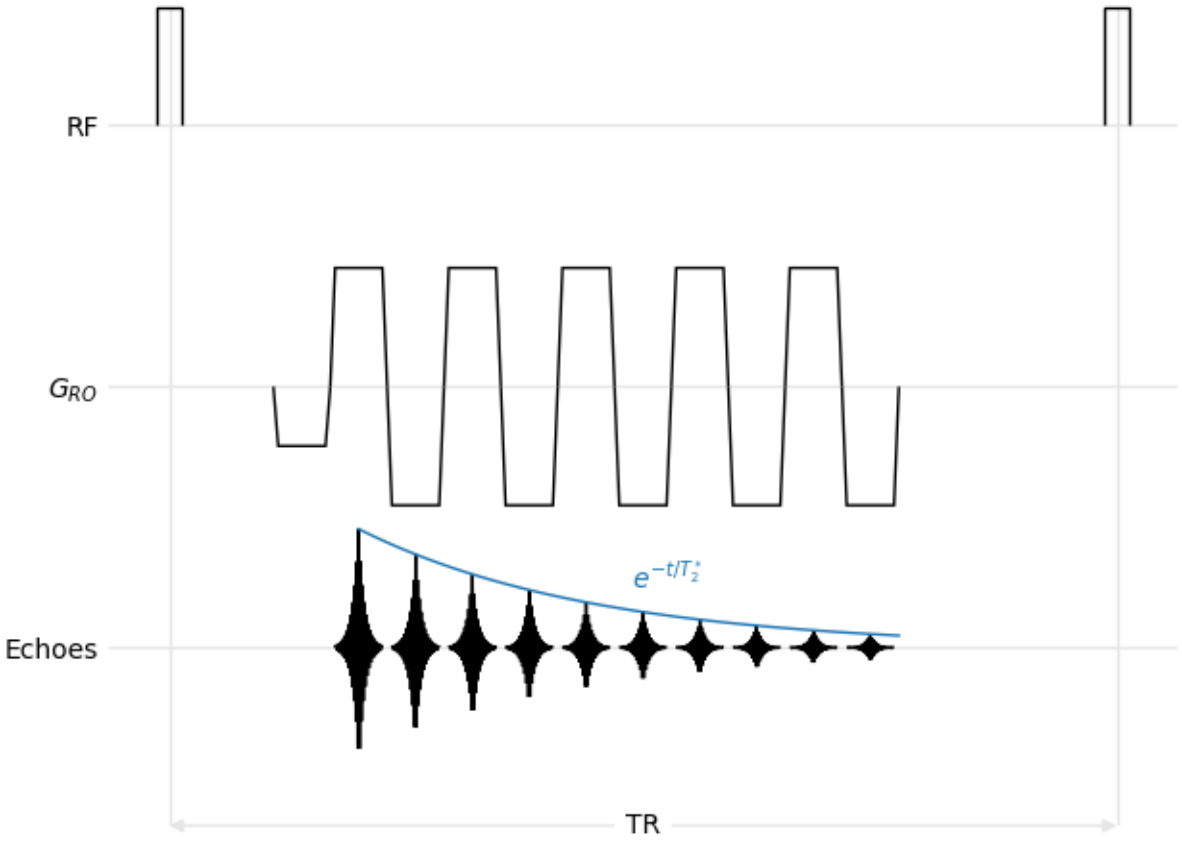


Fig. 3: Multiple echoes and overlayed T_2* decay

CHAPTER

THREE

API DOCUMENTATION

3.1 Diagram

class mrsd.**Diagram**(*plot*, *channels*)

MR sequence diagram

Parameters

- **plot** – an instance of matplotlib axes (plot, subplot, etc.)
- **channels** – sequence of channels names in the plot, from top to bottom

add(*channel*, *event*)

Add an event to the specified channel.

adc(*channel*, **args*, ***kwargs*)

Add an ADC event to the specified channel.

echo(*channel*, **args*, ***kwargs*)

Add an echo event to the specified channel.

gradient(*channel*, **args*, ***kwargs*)

Add a gradient event to the specified channel.

multi_gradient(*channel*, **args*, ***kwargs*)

Add a multi-gradient event to the specified channel.

rf_pulse(*channel*, **args*, ***kwargs*)

Add an RF pulse event to the specified channel.

gaussian_pulse(*channel*, **args*, ***kwargs*)

Add a hard RF pulse event to the specified channel.

hard_pulse(*channel*, **args*, ***kwargs*)

Add a hard RF pulse event to the specified channel.

sinc_pulse(*channel*, **args*, ***kwargs*)

Add a sinc RF pulse event to the specified channel.

readout(*adc_channel*, *gradient_channel*, *duration*, *echo_amplitude=1*, *gradient_amplitude=1*, *ramp=0*, *ramp_up=None*, *ramp_down=None*, *begin=None*, *end=None*, *center=None*, *adc_kwargs=None*, *echo_kwargs=None*, *gradient_kwargs=None*)

Add a readout block (echo, ADC and gradient)

Parameters

- **adc_channel** – channel of the echo and ADC
- **gradient_channel** – channel of the gradient
- **duration** – duration of the echo, ADC, and gradient flat-top
- **echo_amplitude** – amplitude of the echo
- **gradient_amplitude** – amplitude of the gradient flat-top
- **ramp,ramp_up,ramp_down** – ramp durations of the gradient. Use *ramp* for symmetric gradients, and both *ramp_up* and *ramp_down* for asymmetric gradients
- **begin,end,center** – time of the begin, end, or center of the echo/ADC. Only one must be specified.
- **adc_kwargs** – extra parameters for the ADC event (e.g. style)
- **echo_kwargs** – extra parameters for the echo event (e.g. style)
- **gradient_kwargs** – extra parameters for the gradient event (e.g. style)

selective_pulse(*pulse_channel*, *gradient_channel*, *duration*, *pulse_amplitude=1*, *gradient_amplitude=1*, *envelope=None*, *ramp=0*, *ramp_up=None*, *ramp_down=None*, *begin=None*, *end=None*, *center=None*, *pulse_kwargs=None*, *gradient_kwargs=None*)

Add a selective pulse block (pulse and gradient)

Parameters

- **pulse_channel** – channel of the RF pulse
- **gradient_channel** – channel of the gradient
- **duration** – duration of the pulse and gradient flat-top
- **pulse_amplitude** – amplitude of the RF pulse
- **gradient_amplitude** – amplitude of the gradient flat-top
- **envelope** – envelope of the pulse (default to sinc)
- **ramp,ramp_up,ramp_down** – ramp durations of the gradient. Use *ramp* for symmetric gradients, and both *ramp_up* and *ramp_down* for asymmetric gradients
- **begin,end,center** – time of the begin, end, or center of the echo/ADC. Only one must be specified.
- **pulse_kwargs** – extra parameters for the pulse event (e.g. style)
- **gradient_kwargs** – extra parameters for the gradient event (e.g. style)

annotate(*channel*, *x*, *y*, *text*, ***kwargs*)

Add an annotation

Parameters

- **channel** – channel to which the annotation is added
- **x** – time of the annotation
- **y** – relative position of the annotation in the channel
- **text** – text of the annotation
- **kwargs** – extra parameters passed to matplotlib.axes.Axes.text

interval(*begin*, *end*, *y*, *label*, *color='k'*)

Add a time interval annotation

Parameters

- **begin,end** – begin and end time of the interval
- **y** – vertical position of the annotation
- **label** – label of the annotation
- **color** – color of the annotation label

y(*channel*)

Return the y coordinate of the center of a channnel.

3.2 Events

class mrsd.**Event**(*duration*, *amplitude*, *begin=None*, *end=None*, *center=None*, *offset=None*, ***kwargs*)

Abstract sequence event

Parameters

- **duration** – total duration.
- **amplitude** – normalized amplitude between -1 and +1.
- **begin,end,center** – time of the begin, end, or center of the event. Only one must be specified.
- **offset** – horizontal and vertical offset to position the event.
- **kwargs** – passed to matplotlib.patches.Patch

move(*offset*)

Move on the time axis, return the object

3.2.1 ADC

class mrsd.**ADC**(*duration*, ***kwargs*)

ADC/readout event, represented by a rectangle of amplitude 1.

3.2.2 Echo

class mrsd.**Echo**(*duration*, *amplitude=1*, ***kwargs*)

Echo event, represented by an oscillating exponential.

3.2.3 Gradient

class mrsd.**Gradient**(*flat_top*, *amplitude*, *ramp=0*, ***kwargs*)

Gradient event, represented by a trapezoid.

Parameters

- **flat_top** – duration of the gradient flat-top
- **amplitude** – amplitude of the gradient flat-top
- **ramp,ramp_up,ramp_down** – ramp durations of the gradient. Use *ramp* for symmetric gradients, and both *ramp_up* and *ramp_down* for asymmetric gradients

adapt(*flat_top*, *area_factor*, *ramp=0*, *ramp_up=None*, *ramp_down=None*, ***kwargs*)

Return a gradient with an area equal to a factor of the current gradient.

3.2.4 Multi-Gradient

class mrsd.**MultiGradient**(*flat_top*, *amplitude*, *ramp=0*, *steps=5*, ***kwargs*)

Multiple gradient events (e.g. phase encoding), represented by nested trapezoids.

Parameters

- **flat_top** – duration of the gradient flat-top
- **amplitude** – maximum amplitude of the gradient flat-top
- **ramp,ramp_up,ramp_down** – ramp durations of the gradient. Use *ramp* for symmetric gradients, and both *ramp_up* and *ramp_down* for asymmetric gradients
- **steps** – number of steps drawn between the maximum and minimum amplitude

3.2.5 RF Pulse

class mrsd.**RFPulse**(*duration*, *amplitude*, *envelope=<function sinc_envelope>*, ***kwargs*)

RF pulse event, represented by a user-defined envelope

Parameters

- **duration** – duration of the pulse
- **amplitude** – maximum amplitude
- **envelope** – sinc_envelope (default), box_envelope, or gaussian_envelope
- **lobes** – number of lobes on each side of a sinc envelope (defaults to 3)
- **apodization** – apodization of a sinc envelope (None (default), “hann”, or “hamming”)
- **sd** – standard deviation of a Gaussian envelope (defaults to 0.3)
- **points** – number of points used to draw the envelope (defaults to 101)

INDEX