MR SPRINT Documentation

Release 1.2

Daniel Cosmo Pizetta, Victor Hugo de Mello Pessoa

Jun 27, 2018

CONTENTS


MR SPRINT


pypi
v1.0

CHAPTER

ONE

WELCOME!

Magnetic resonance experiment simulator and visualization tool

MRSPRINT is a visual magnetic resonance simulator where you can simulate a magnetic resonance experiment - spectroscopy or imaging. Its main goal is to be an education tool that assists the student/staff to understand, interpret and explore magnetic resonance phenomena.

This tool is totally free (see license), but if you are making use of it, you need to cite us using both citations. This is very important for us.

- Article: Coming soon!
- Software: Coming soon!

If you are using this piece of software to generate images, gif's, movies, etc., or using images available on this site, please, also reference us using those citations.

1.1 What can you see?

- Precession: spins precessing in static magnetic field;
- Resonance: resonance when a RF pulse is applied;
- Contrast: T1, T2, and density of spins;
- Field inhomogeneity: isochromates can be shown with their dispersion;
- Gradient: magnetic field gradient in action, its intensity and effect over the frequency;
- Evolution: over magnetization with intensity, frequency and phase and the pulse sequence;
- FID: free induction decay, the signal;
- Echo: spin or gradient echo (rephasing/dephasing).

1.2 Future planned features

- K-space visualization;
- Multi-nuclei samples/experiments;
- T2* as sample parameter to easily setup field inhomogeneity;
- Graphical sequence editor;
- Chemical interactions;
- View on coordinate laboratory system;
- Flow (spins not fixed in positions).

1.3 Documentation

Go to documentation on ReadTheDocs! It is available on Read The Docs in HTML, EPUB, and PDF.

1.4 Download binaries - click-and-run

Binaries are for those do not wish to install any Python things. We recommend them to the ones without any programming experience. Download from links below.

- Portable Windows Binaries: coming soon!
- Portable Linux Binaries: coming soon!
- Portable Mac Binaries: coming soon!

Sou you can just download, decompress, click-and-run.

1.5 Installing from PyPI - stable, end-user

To install, do

```
$ pip install mrsprint
```

This will install all necessary dependencies then the code.

To run from terminal

```
$ mrsprint
```

1.6 Dependencies

- NumPy: Numerical mathematical library;
- SciPy: Scientific library;
- NMRGlue: NMR processing library;
- H5Py: Storing and managing data files;
- PyQtGraph: Data visualization library;
- PyQt/Pyside: Graphical framework.

Dependencies are automatically installed when using the method above.

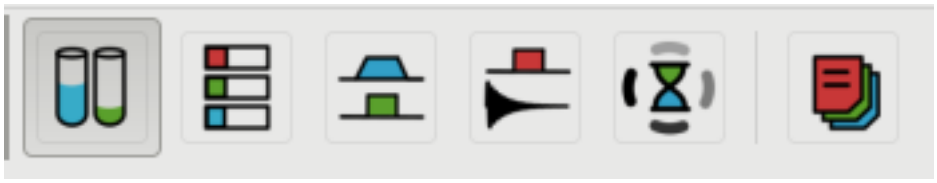
CHAPTER

TWO

OVERVIEW

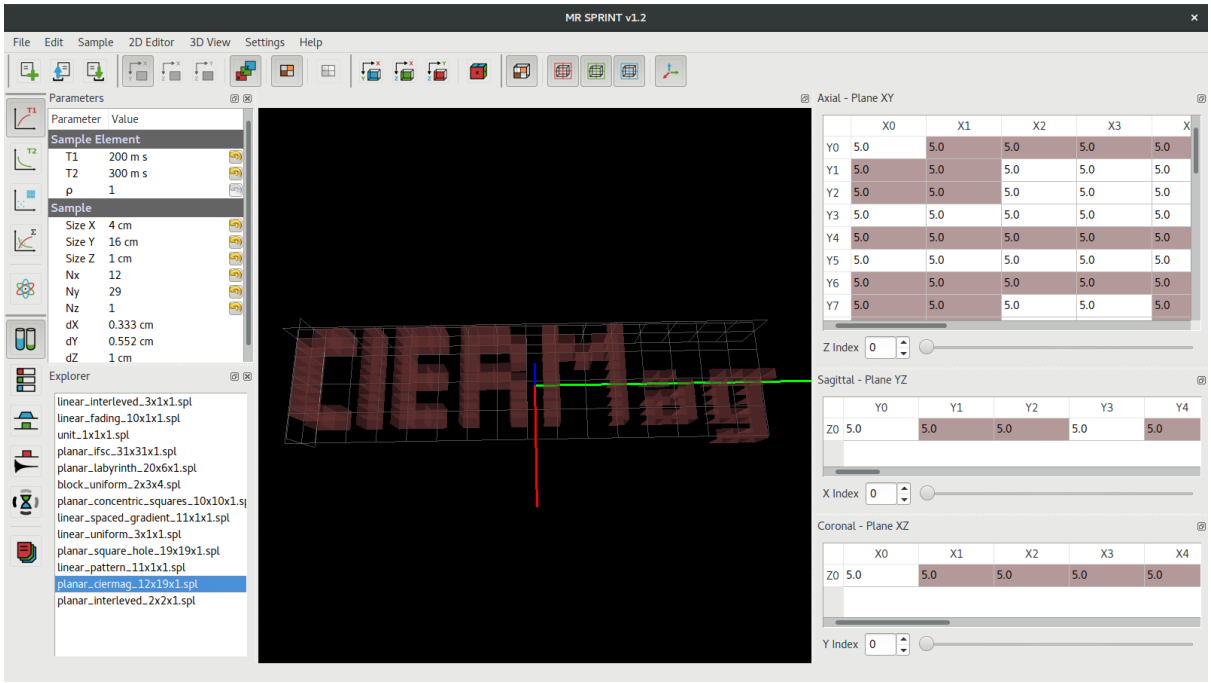
2.1 Main screens

Context editors are specialized to edit each step of experiment, see below.



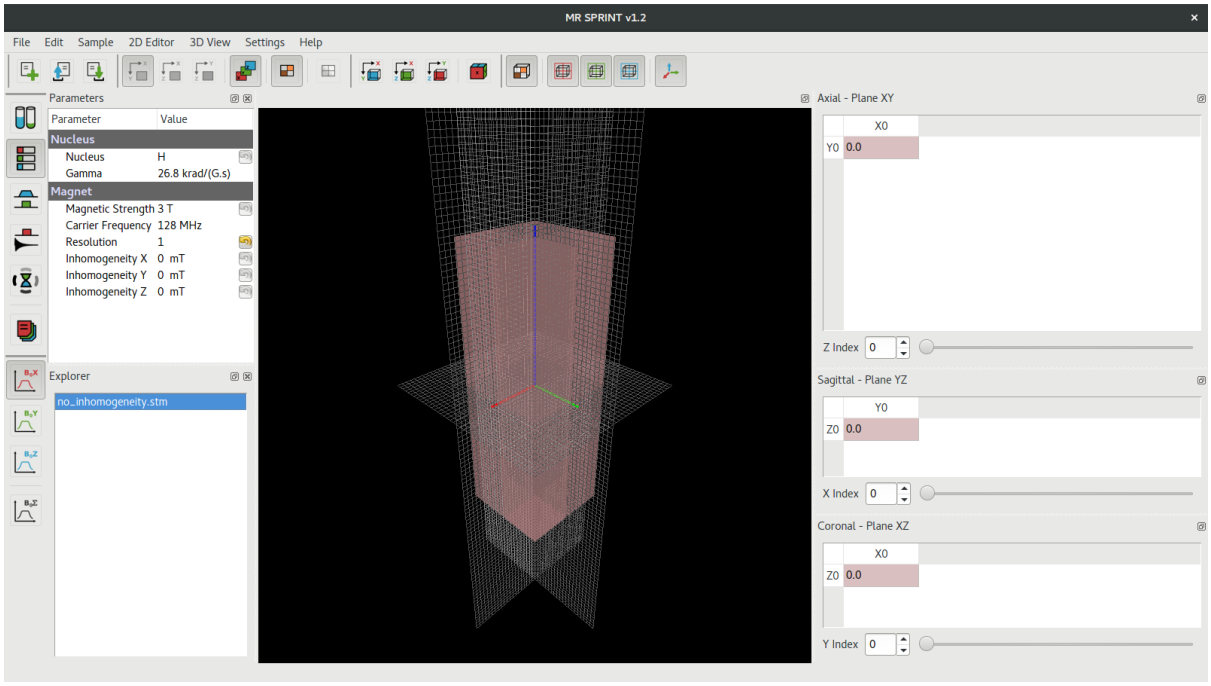
2.1.1 Sample

You can open and/or create samples to run with MR experiment. Each sample element have three characteristics: t1,t2 and density of spins.



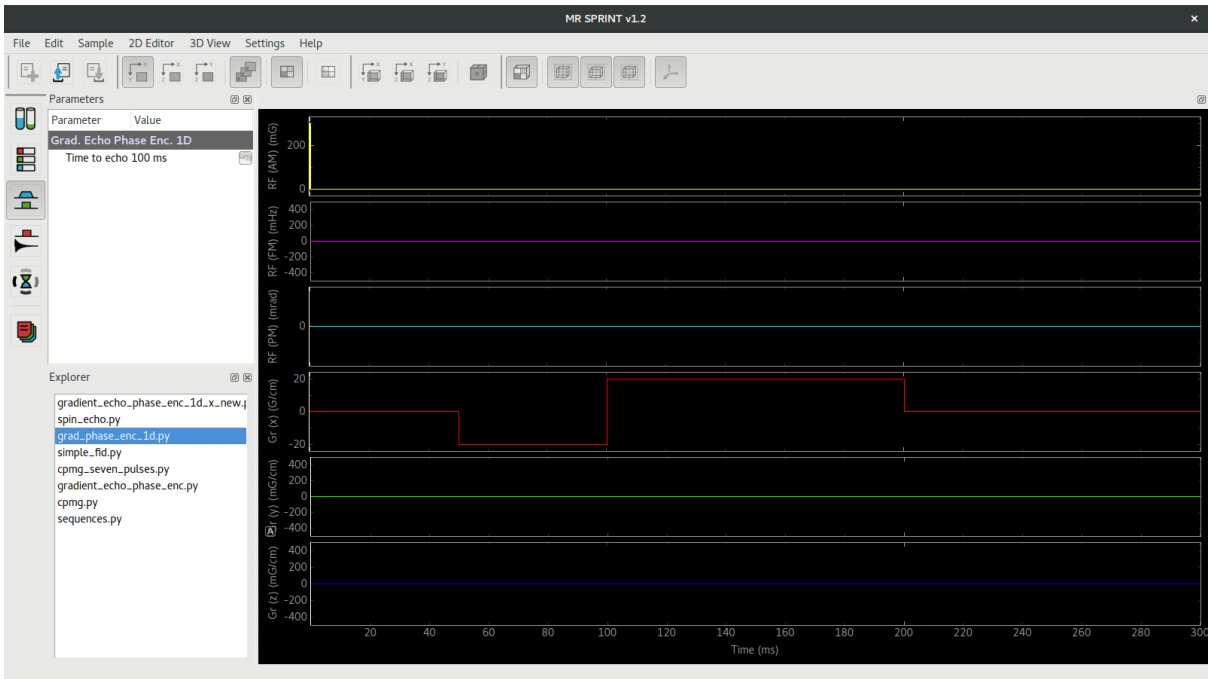
2.1.2 System

Here you can set static magnetic field and add inhomogeneity to it.



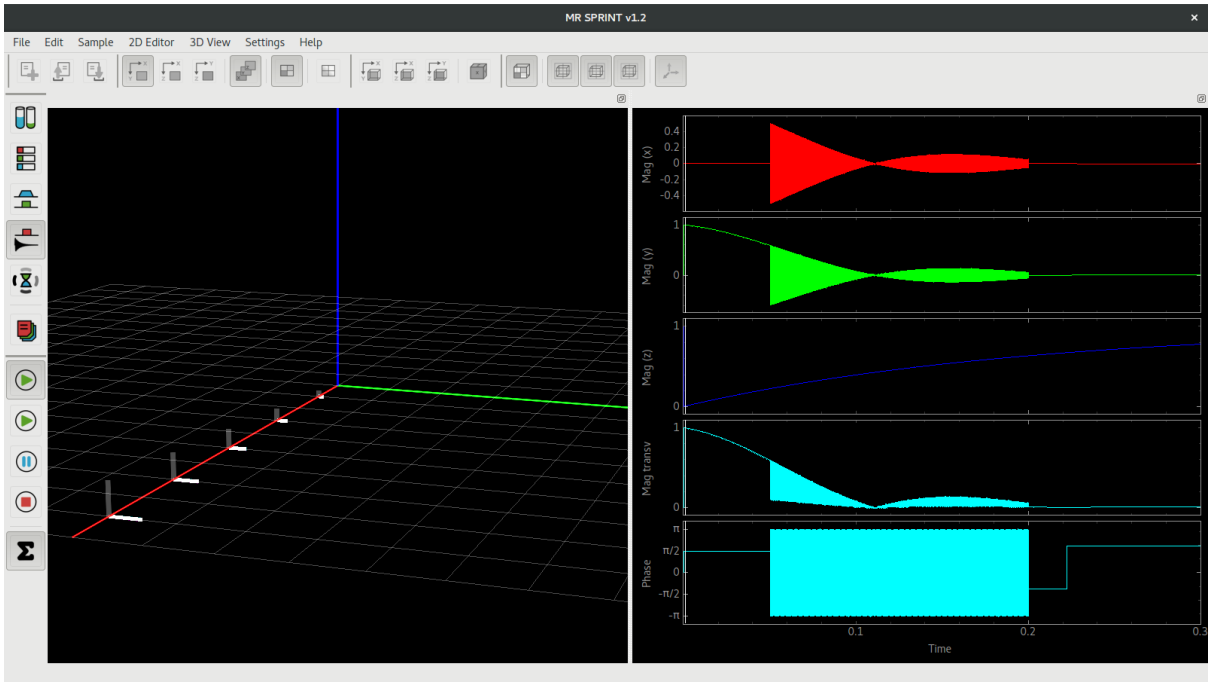
2.1.3 Sequence

You can choose a pulse sequence for your experiment, that includes the RF and gradient pulses. The sequence is programmed in Python at this moment.



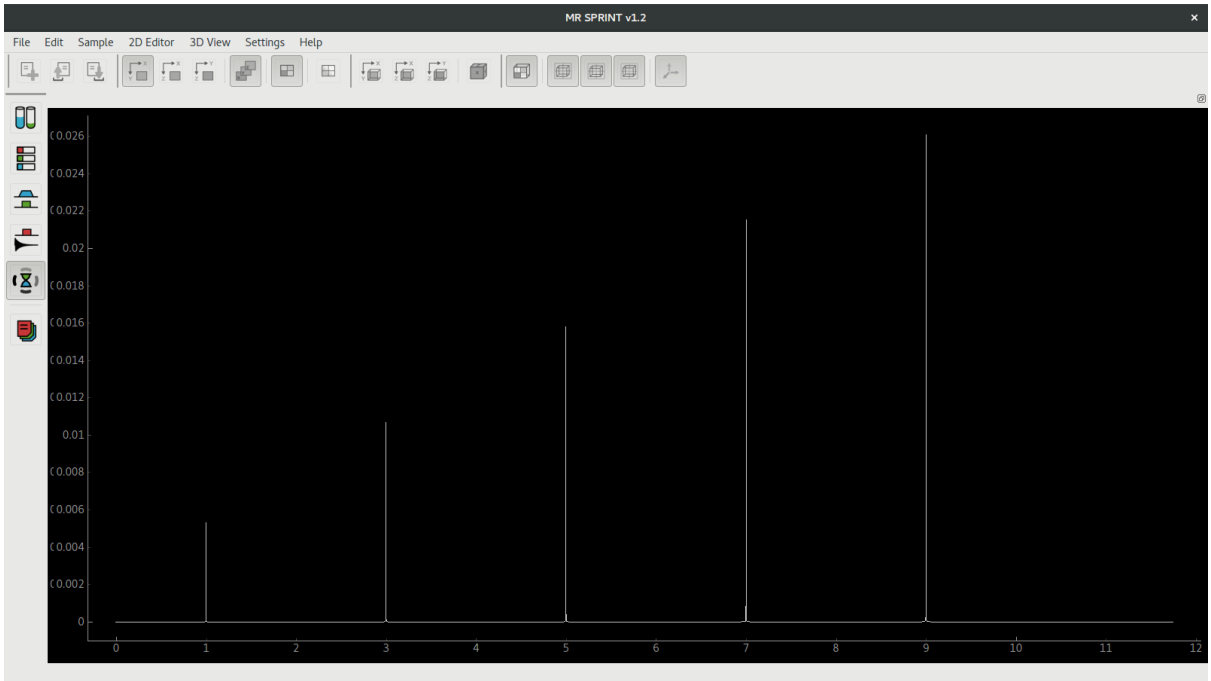
2.1.4 Simulator

At this point you can set simulation mode and other details about simulation , e.g. time resolution.



2.1.5 Processing

Finally you can process your data to plot the spectrum, imaging or other features.



2.2 2D Editor

This editor provides a table that represents the current selected slice of the 3D view. In this table you can edit the values of each element property(ies). Colors also help you to pre visualize the intensity of chosen value.

2.3 3D View

3D view is used to show contexts objects such as sample, magnet field inhomogeneity, and the evolution of magnetization. You can move the cam to adjust the perspective.

2.4 Data

Data can be saved using HDF5 files. For each context we adopted an extension to distinguish from each other. HDF5 files can be easily modified with other external tool if necessary.

CHAPTER

THREE

HISTORY

3.1 A brief history of this universe

Being short, when difficulties arises at understanding magnetic resonance phenomena, we should search tools that could help us.

At that time we have found nice explanations on Brian site.

As we are fan of Python, we discovered this implementation from Neji.

Yet, it was difficult to change parameters and create new graphics. So, this work starts. From that, many ideas have shining, and, here we are.

The main developer/maintainer is the PhD in computational physics Daniel C. Pizetta, at University of São Paulo, São Carlos, Brazil.

A lot of the code was developed by the bachelor, in physics and biomolecular sciences, Victor H.M. Pessoa in his final paper. He created many new functionalities, implemented all file interactions and connections between graphical interface and core code.

From now on, Clara Vidor is continuing the development, and we hope that will still new features to be released soon.

We also have a head professor PhD Fernando Fernandes Paiva that provides a technical support in magnetic resonance issues.

CHAPTER

FOUR

REFERENCE

4.1 examples

4.2 mrsprint

4.2.1 mrsprint package

Subpackages

mrsprint.gui package

Submodules

mrsprint.gui.mrsprint_rc module

mrsprint.gui.mrsprint_rc.**qCleanupResources**()

mrsprint.gui.mrsprint_rc.**qInitResources**()

mrsprint.gui.mw_gradient module

class mrsprint.gui.mw_gradient.**Ui_Gradient**
Bases: object

retranslateUi(*Gradient*)

setupUi(*Gradient*)

mrsprint.gui.mw_mrsprint module

class mrsprint.gui.mw_mrsprint.**Ui_MainWindow**
Bases: object

retranslateUi(*MainWindow*)

setupUi(*MainWindow*)

mrsprint.gui.mw_settings module

class mrsprint.gui.mw_settings.**Ui_Settings**
Bases: object

retranslateUi(*Settings*)

setupUi(*Settings*)

Module contents

Package for GUI related classes and objects.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/01/09

mrsprint.sequence package

Submodules

mrsprint.sequence.protocol module

mrsprint.sequence.sequence module

Module for pulse sequence related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

class mrsprint.sequence.sequence.**CPMGSequence**
Bases: mrsprint.sequence.sequence.Sequence

Generates a CPMG sequence.

class mrsprint.sequence.sequence.**GradientEchoSequence**
Bases: mrsprint.sequence.sequence.Sequence

Generate a Gradient Echo sequence.

class mrsprint.sequence.sequence.**Sequence**(*settings*, *nucleus*, ***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents a sequence for magnetic resonance systems.

getGradient()
Return the entire gradient.

getRF()
Return the entire rf.

setGradient(*grad_value*)
Properly sets the gradient.

setRF(*rf_value*)
Properly sets the rf stream signal.

Module contents

Package for sequence related classes and objects.

Authors:

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Authors: 2015/11/01

Todo: Fix these modules.

---

mrsprint.simulator package

Submodules

mrsprint.simulator.old_plot module

mrsprint.simulator.plot module

Module for plotting related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

class mrsprint.simulator.plot.**Plot**(*settings*, *mx*, *my*, *mz*, *gr*, *tp*, *freq_shift*, *position*, *max_magnetization*)

Bases: object

Class that contains the plots of simulation.

Parameters

- **settings** (Settings) – Represents the program settings.
- **mx** (np.ndarray) – Magnetization in x.
- **my** (np.ndarray) – Magnetization in y.
- **mz** (np.ndarray) – Magnetization in z.
- **gr** (np.ndarray) – Gradients of magnetic field.
- **tp** (np.ndarray) – Time.
- **freq_shift** (np.ndarray) – Frequency shift due to field inhomogeneity.
- **position** (np.ndarray) – Position of spins.
- **max_magnetization** (float) – Maximum value of magnetization.

pause()
Stop the simulation on plot.

play()
Start the simulation on plot.

plotMagnetization(*mag_win*)
Create the graphics of magnetization.

Parameters **mag_win** (pg.graphicsWindows.GraphicsWindow) – Window where plot will be made.

plotSpin(*view*)
Create the 3D exhibition of magnetization over the experiment.

Parameters **view** (pg.graphicsWindows.GraphicsWindow) – Window where the plot will be made.

run()
Run the simulation on plot, from zero.

update()
Update magnetization position vectors and timelines on graphics dynamically.

mrsprint.simulator.plot.**plot_item**(*graphics*, *x*, *y*, *line_color*, *x_axis=True*, *x_label=''*, *x_unit=''*, *y_axis=True*, *y_label=''*, *y_unit=''*)
Create a new plot item in a graph.

Parameters

- **graphics** (pl.graphicsWindow) – The GraphicsWindow where the plot will be shown.
- **x** (np.array) – List of x coordinates.
- **y** (np.array) – List of y coordinates.
- **line_color** (str) – String containing the color ("r", "g", "b", "w", for example).
- **x_axis** (bool) – Visibility of x axis.
- **x_label** (str) – Label of x axis.
- **x_unit** (str) – Unit of x axis.
- **y_axis** (bool) – Visibility of y axis.
- **y_label** (str) – Label of y axis.
- **y_unit** (str) – Unit of y axis.

Returns Plot.

Return type pg.graphicsItems.PlotItem

mrsprint.simulator.simulator module

Module for simulate the response of the sample to the magnetic signal related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

class mrsprint.simulator.simulator.**Simulator**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the simulator.

Module contents

Package for simulator related classes and objects.

Authors:

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2018/06/07

Todo: Maybe remove old_plot module.

---

mrsprint.simulator.**calculate_t2_star**(*t2*, *freq_shift*)

Returns the value for t2*, considering the range of frequencies.

Parameters

- **t2** (float [s]) – T2 value in seconds.
- **freq_shift** (float [Hz]) – Frequency shift from resonance - symetric between zero (in resonance).

Returns T2 star value.

Return type float [s]

Todo: Confirm if it is the right way to calculate or if it has a weight function for t2star Pass freq_shift as a number?

---

mrsprint.simulator.**create_positions**(*size=(1, 1, 1)*, *step=(1.0, 1.0, 1.0)*, *offset=(0.0, 0.0, 0.0)*, *dtype=<class 'numpy.float32'>*)

Creates array of positions.

Parameters

- **size** (tuple(int)) – Size in x, y and z. The minimum number is one for each axis.
- **step** (tuple(float)) – Step for each axis.
- **offset** (tuple(float)) – Offset for each axis. If zero, the final vector begins in zero and ends in size plus offset.

Returns

Position array in this format [[gr_x_plotx gr_y_plotx gr_z_plotx . . .]
[gr_x_ploty gr_y_ploty gr_z_ploty . . .][gr_x_plotz gr_y_plotz gr_z_plotz . . .]], where p is the number of position (from offset to size plus offset).

Return type np.array

Todo: Set physical unit for each args.

---

mrsprint.simulator.**frequency_shift**(*freq_shift*, *freq_step=1.0*, *offset=0.0*, *symetric=True*, *dtype=<class 'numpy.float32'>*)

Generates an array of frequency shift, from -frequency_shift to +frequency_shift, between offset.

Parameters

- **freq_shift** (float [Hz]) – Maximum frequency to shift.
- **freq_step** (float [Hz]) – Spacing between values.
- **offset** (float [Hz]) – Offset frequency.
- **symetric** (bool) – If true, generates the array between offset (-maximum shift, maximum shift), otherwise from offset to maximum frequency. Default is True.
- **dtype** (np.dtype) – Data type for the array. Default is np.float32.

Returns Array of frequency shift value.

Return type np.array [Hz]

Todo: Use the key symetric for something.

mrsprint.simulator.**reduce_magnetization_in_frequency**(*mx*, *my*, *mz*, *freq_shift*, *fsa_size*)
Reduces the magnetization vector by summing frequency components.

Parameters

- **mx** (np.array) – Array of magnetization x without reduction in frequency.
- **my** (np.array) – Array of magnetization y without reduction in frequency.
- **mz** (np.array) – Array of magnetization z without reduction in frequency.
- **freq_shift** (np.array) – Array of frequency shift.
- **fsa_size** (int) – Size of array of frequency shift.

Returns (mx, my, mz) arrays of magnetization reduced

Return type tuple

Todo: Shape size different of 2. Better solution for freq_shift array. Set physical unit for each args.

mrsprint.simulator.**reduce_magnetization_in_position**(*mx*, *my*, *mz*, *position*, *freq_shift*)
Reduces the magnetization vector by summing position components.

Parameters

- **mx** (np.array) – Array of x magnetization without reduction in frequency.
- **my** (np.array) – Array of y magnetization without reduction in frequency.
- **mz** (np.array) – Array of z magnetization without reduction in frequency.
- **position** (np.array) – Array of positions.
- **freq_shift** (np.array) – Array of frequency shift.

Returns (mx, my, mz) arrays of magnetization reduced.

Return type tuple

Todo: Test shape size different of 2. Set physical unit for each args.

mrsprint.simulator.**transform_cart_to_pol**(*x*, *y*)
Returns a transformed catesian vector into polar coordinates.

Parameters

- **x** (np.array) – X coordinate.
- **y** (np.array) – Y coordinate.

Returns (rho, phi) vectors formed by radial and angular coordinates.

Return type tuple

mrsprint.simulator.**transform_pol_to_cart**(*rho*, *phi*)
Returns a transformed polar vector into cartesian coordinates.

Parameters

- **rho** (np.array) – Radial coordinate.
- **phi** (np.array) – Angular coordinate.

Returns (x, y) vectors formed by x and y coordinates

Return type tuple

mrsprint.subject package

Submodules

mrsprint.subject.sample module

Module for sample related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

class mrsprint.subject.sample.**Nucleus**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the parameters in the nucleus.

updateGamma()
Update the value for gamma if the nucleus is changed

class mrsprint.subject.sample.**Sample**(*sample_config*, ***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the parameters of each sample.

Parameters **sample_config** (SampleConfig) – An object that represents the limits to this sample.

Todo: This class needs to be reviewed for its parameters.

---

dXUpdate()
Update the value for dX.

dYUpdate()
Update the value for dY.

dZUpdate()
Update the value for dZ.

class mrsprint.subject.sample.**SampleConfig**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that configure the limit parameters of each sample.

Todo: This class needs to be reviewed for its parameters.

---

class mrsprint.subject.sample.**SampleElement**(*sample_element_config*, ***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the parameters of each sample element.

Parameters **sample_element_config** (SampleElementConfig) – An object that represents the limits to the sample elements.

Todo: This class needs to be reviewed for its parameters.

class mrsprint.subject.sample.**SampleElementConfig**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that configure the limit parameters of each sample element.

Todo: This class needs to be reviewed for its parameters.

Module contents

Package for subject related classes and objects.

Authors:

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/10/01

mrsprint.system package

Submodules

mrsprint.system.gradient module

Module for gradient related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

Todo: Change nameOfFunctions to name_of_functions.

class mrsprint.system.gradient.**Gradient**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the gradients parameters in the sistem.

mrsprint.system.gradient.**gradient_delay**(*duration*, *dt*, *number_of_points=0*)
Generate a delay of gradient pulse.

Parameters

- **duration** (float [s]) – Delay time.
- **dt** (float [s]) – Time resolution.
- **number_of_points** (int) – Number of points.

Returns Gradient delay - a zero x, y and z gradient components

Return type np.array

mrsprint.system.gradient.**gradient_duration**(*gradient_event*, *dt*)
Return the duration of the gradient event (array).

It is based on the size and dt.

Parameters

- **gradient_event** (np.array) – An array of event.
- **dt** (float [s]) – Value of time resolution.

Returns Duration of the event.

Return type float [s]

Todo: Regard the dimensions of the array.

---

mrsprint.system.magnet module

Module for magnet related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

Todo: Maybe put all config together in magnet classes.

---

class mrsprint.system.magnet.**Magnet**(*magnet_config*, ***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the parameters in the magnet.

Parameters **magnet_config** (MagnetConfig) – An object that represents the limits to this magnet.

class mrsprint.system.magnet.**MagnetConfig**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that configure the limit parameters of the magnet.

mrsprint.system.rf module

Module for radiofrequency related classes and functions.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

class mrsprint.system.rf.**RF**(***opts*)
Bases: pyqtgraph.parametertree.parameterTypes.GroupParameter

Class that represents the RF parameters in the system.

mrsprint.system.rf.**rf_delay**(*duration*, *dt*)
Generate a delay of rf pulse.

Parameters

- **duration** (float [s]) – Delay time.
- **dt** (float [s]) – Time resolution.

Returns Rf delay - a zero am, pm, fm components in complex format.

Return type np.array(complex)

mrsprint.system.rf.**rf_duration**(*rf_event*, *dt*)

Return the duration of the rf event (array) based on the number of the points and dt.

Parameters

- **rf_event** (np.array) – An array of event.
- **dt** (float [s]) – Value of time resolution.

Returns duration of the event.

Return type float [s]

Todo: Regard the dimensions of the array.

---

mrsprint.system.rf.**square_rf_pulse**(*dt*, *gamma*, *b1_max*, *flip_angle=90*, *phase=0.0*, *degrees=True*)

Generate a hard rf pulse with a specific and constant flip angle and phase.

Parameters

- **dt** (float [s]) – Value of time resolution.
- **gamma** (float [rad/(G*s)]) – Gyromagnetic ratio of the excited nuclei.
- **b1_max** (float [G]) – Max RF amplitude.
- **flip_angle** (float [degrees, radians]) – Array of flip angle for rf pulse in degrees (if degrees = True).
- **phase** (float [degrees, radians]) – Array of phase angle for rf pulse in degrees (if degrees = True).
- **degrees** (bool) – Inform if the input is in degrees or radians.

Returns A square rf pulse in imaginary form.

Return type np.array(complex)

Module contents

Package for system related classes and objects.

Authors:

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2015/11/01

Submodules

mrsprint.globals module

Global values.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2015/06/01

mrsprint.mainwindow module

Main window of visual simulator.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/08/01

Todo: Replaning this module, divide and make it more simple.

---

class mrsprint.mainwindow.**MainWindow**(*parent=None*)

Bases: PyQt4.QtGui.QMainWindow

Main window.

About load methods: This method should exist for each type of context that could be edited in the 2D editor and 3D view. Ex. loadSystemParameters. It should be responsible for loading values from parameter tree, connect signals - to respective own actions, and set data.To keep it isolated, it should set data, and when data is changed, data triggers the GUI update. And the reverse mode.

about()

Show about message.

canClose()

Check if the user want to save the sample before closing.

Returns True if can close.

Return type bool

clearSelection2DEditor(*all_tables=False*)

Clear selection of all 2D editor tables.

Parameters **all_tables** (bool) – Informs if all tables are shown or not. Default is False.

closeEvent(*event*)

Close event.

create3DView()

Create a 3D view and the first 3D object.

It must be called when starts and when the shape or size is changed.

enableGradient2DEditor()

Enable the gradient editor for 2D editor.

fileClose()

Close file and create a new one.

fileNew()

Restart the edition from start.

fileOpen()

Open a dialog to select a HDF5 file to be opened.

fileSave()
Save the current sample in the last saving file.

If there is no current saving file, it calls fileSaveAs().

fileSaveAs()
Open a dialog to select the current saving file.

gradient2DEditor()
Run the gradient editor and set values to 2D editor tables.

Todo: Remove eval after changing itemMethodsTable2dEditor.

---

invertBackground2DEditor(*invert=False*)
Invert the background color of 2D editor.

invertBackground3DView(*invert=False*)
Invert the background color of 3D view.

Parameters **invert** (bool) – Informs if the background color should be inverted. Default is False.

itemsMethodsTable2DEditor()
Return current selected item, methods to access their indexes and the table.

Returns The current selected items, the methods attached to the selected table and the selected table

Return type list (QtGui.QTableWidgetItem()), dict (string of methods), QtGui.QTableWidget()

Todo: This could be even better if getting these values, return just a new matrix with proper indexes and values to set data.

---

loadProcessingParameters()
Load processing parameters and connect signals.

loadSampleParameters()
Load sample context and its parameters.

loadSequenceParameters()
Load sequence context and its parameters.

loadSettings()
Load all the config parameters needed by the software.

loadSimulatorParameters()
Load simulator context and its parameters.

loadSystemParameters()
Load system context and its parameters.

open(*file_path*)
Open a HDF5 file and applies the changes to the program.

Parameters **file_path** (str) – Path to the file where the data will be opened.

openSample(*file_path*)
Open a HDF5 sample file.

Parameters **file_path** (str) – Path to a file containing a sample.

openSequence(*file_path*)
Open a python sequence file.

The file must contain a class SequenceExample, that contains information about the sequence as RF pulses and the Gradient.

Parameters **file_path** (str) – Path to a file containing the sequence.

openSystem(*file_path*)
Open a HDF5 system file.

Parameters **file_path** (str) – Path to a file containing the system.

plotSequence()
Use the data from a sequence to create the RF and Gradient plots.

quit()
Close the main window.

resizeTable2DEditor()
Change the size of the tableWidgets whenever a row/column is added/removed.

sampleSettingsChanges()
Reload sample parameters.

save(*file_path*)
Create a HDF5 file to save current data.

Parameters **file_path** (str) – Path to the file where the data will be saved.

saveSample(*file_path*)
Save a HDF5 sample file.

Parameters **file_path** (str) – Path to the file where the sample will be saved.

saveSystem(*file_path*)
Save a HDF5 system file.

Parameters **file_path** (str) – Path to the file where the system will be saved.

setIndexLimits2DEditor(*nx*, *ny*, *nz*)
Set index limits for 2D editor.

Parameters

- **nx** (int) – Shape of data in the x dimension.
- **ny** (int) – Shape of data in the y dimension.
- **nz** (int) – Shape of data in the z dimension.

setSettingsPath()

showAxis3DView(*show=True*)
Show axis in 3D view.

Parameters **show** (bool) – Informs if the axis should be visible. Default is True.

simulate()
Setup a simulation.

simulatorSettingsChanges()
Reload simulator parameters.

systemSettingsChanges()
Reload system parameters.

tabify2DEditor(*value*)
Tabify 2D editor depending on value.

Parameters **value** (bool) – Informs if the tables in 2DEditor should be tabified or not.

translateGrid3DView()
Translate grids to correct position.

updateContext()
If context is changed, update interface.

updateDataFromTable2DEditor(*item_changed*)
Update data from table in 2D editor.

updateExplorer()
Update explorer files for current context.

updateObserverData()
Reread data and apply updates.

updateObserverDataSize()
Reread data and apply updates, maybe need some rebuild.

updateObserverIndex()
Reread indexes and apply updates, does not need recreated anything.

updateSelectedCells(*value*, *index*)
Update selected cells from parameter tree.

Parameters

- **value** (float) – Value to be updated in the table.
- **index** (int) – Index of selected type of data.

updateTableFromData2DEditor()
Update items of all 2D editor using 3d-array data and set color.

This method is called when the data shape is changed or when the plane index is changed. Also, when the data index is changed.

updateViewFromData3DView()
Color 3D view based on 3D data.

mrsprint.settings module

Module responsible for the configuration of settings of all other modules.

Authors:

- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>
- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>

Since: 2017/07/01

Todo: Insert log inside functions.

---

class mrsprint.settings.**Settings**(**args*, ***kwargs*)
Bases: PyQt4.QtGui.QMainWindow

Main window for settings.

Todo: Include restore default settings. Create a better check for values.

---

blockUnblockSignals(*value*)
Block or unblock signals defined by value.

Parameters **value** (bool) – True to block, false otherwise.

check()
Evaluate if parameters are between its limits before saving them.

Returns True if OK.

Return type bool

Todo: Reduce complexity and check all value limits.

open(*file_path*, *set_file=True*)
Open a HDF5 settings file.

Parameters

- **file_path** (str) – Path to a file containing the settings.
- **set_file** (bool) – set path file if true, otherwise just import.

openFile(*set_file=True*)
Open a dialog to select the config file to be opened.

save(*file_path*, *set_file=True*)
Save a HDF5 settings file.

Parameters

- **file_path** (str) – Path to a file containing the settings.
- **set_file** (bool) – set path file if true, otherwise just export.

saveFile(*set_file=True*)
Open a dialog to select the config file to be saved.

Module contents

Magnetic resonance experiment simulator and visualization tool.

Authors:

- Daniel Cosmo Pizetta <daniel.pizetta@usp.br>
- Victor Hugo de Mello Pessoa <victor.pessoa@usp.br>

Since: 2015/07/01

4.3 scripts

4.3.1 generate_qrc module

4.3.2 get_version module

4.3.3 process_icons module

4.3.4 process_ui module

CHAPTER

FIVE

DOWNLOADS

Here you will find documents and files to download.

5.1 Download binaries - click-and-run

Binaries are for those do not wish to install any Python things. We recommend them to the ones without any programming experience. Download from links below.

- Portable Windows Binaries: coming soon!
- Portable Linux Binaries: coming soon!
- Portable Mac Binaries: coming soon!

Sou you can just download, decompress, click-and-run.

5.2 Documentation

Links for latest available documentation.

CHAPTER

SIX

CHANGELOG

6.1 v1.2

- Add file explorer
- Add about
- Add docs to ReadTheDocs
- Improved and new icons, and logo
- Bug fixes

6.2 v1.1

- Fix docs, examples

6.3 v1.0

- First public API

6.4 v0.6

- New structure, revised files and screenshots
- Fix doc style
- Change nucleous to nucleus
- Fix pyqtgraph imports

6.5 v0.5

- Add new extensions for HDF5 files for contexts
- More examples of samples
- Add simulation context
- Bug fixes

6.6 v0.4

- Add sample examples
- Bug fixes
- Improvements on 2D editor and 3D plot

6.7 v0.3

- UI Improvements
- Add git-lab CI
- Add pylint
- Open and save samples in HDF5
- Bug fixes

6.8 v0.2

- Removing unnecessary things
- Improvements in code
- Docs with Sphinx

6.9 v0.1

- First version with binaries for Windows and Linux - Pyinstaller
- Add tox
- Add scripts for pyinstaller, ui, icons
- 2D editor and 3D view
- Main window and toolbars

CHAPTER

SEVEN

AUTHORS

- Daniel Cosmo Pizetta
- Victor Hugo de Mello Pessoa

CHAPTER

EIGHT

LICENSE

8.1 Code - The MIT License

8.2 Images - Creative Commons Attribution International 4.0

8.2.1 Using Creative Commons Public Licenses

Creative Commons public licenses provide a standard set of terms and conditions that creators and other rights holders may use to share original works of authorship and other material subject to copyright and certain other rights specified in the public license below. The following considerations are for informational purposes only, are not exhaustive, and do not form part of our licenses.

- **Considerations for licensors:** Our public licenses are intended for use by those authorized to give the public permission to use material in ways otherwise restricted by copyright and certain other rights. Our licenses are irrevocable. Licensors should read and understand the terms and conditions of the license they choose before applying it. Licensors should also secure all rights necessary before applying our licenses so that the public can reuse the material as expected. Licensors should clearly

mark any material not subject to the license. This includes other CC-licensed material, or material used under an exception or limitation to copyright. More considerations for licensors.

- **Considerations for the public:** By using one of our public licenses, a licensor grants the public permission to use the licensed material under specified terms and conditions. If the licensor's permission is not necessary for any reason–for example, because of any applicable exception or limitation to copyright–then that use is not regulated by the license. Our licenses grant only permissions under copyright and certain other rights that a licensor has authority to grant. Use of the licensed material may still be restricted for other reasons, including because others have copyright or other rights in the material. A licensor may make special requests, such as asking that all changes be marked or described. Although not required by our licenses, you are encouraged to respect those requests where reasonable. More considerations for the public.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

M

N

O

P

Q

R

S

T

U