mr_utils Documentation

Release 0.0.0

N. McKibben

May 11, 2019

Contents:

CHAPTER 1

Installation

Say you want to use this package in one of your python scripts. You can install it using pip like so:

```
git clone https://github.com/mckib2/mr_utils
cd mr_utils
pip3 install -e ./
```

You'll need to manually install the *ismrmrd-python-tools* as it's currently not available on pypi. You can find it here: https://github.com/ismrmrd/ismrmrd-python-tools.git

CHAPTER 2

Modules

2.1 BART

2.1.1 BART Interface

Simple interface to call BART's python object from mr_utils.

This will verify that BART's TOOLBOX_PATH is found, if not, an exception will be raised. Consider using Bartholomew, it's meant to be a better interface to command-line BART.

mr_utils.bart.bart.**bart**(**args*)
Wrapper that passes arguments to BART installation.

2.1.2 Bartholomew

More friendly python interface for BART.

I also want this to be able to run BART on remote computer through ssh, to remove BART as a strict dependency for the local machine, much like we treat Gadgetron.

Examples

```
>>> import Bartholomew as B
>>> # Usage: B.[bart-func](args)
>>> traj_rad = B.traj(x=512, y=64, r=True)
>>> ksp_sim = B.phantom(k=True, s=8, t=traj_rad)
>>> igrid = B.nufft(ksp_sim, i=True, t=traj_rad)
```

Notice that input ndarrays are positional arguments (e.g., ksp_sim is the first argument for nufft instead of the last).

To get comma separated lists (e.g., -d x:x:x), use the List type:

```
>>> img = B.nufft(ksp_sim, i=True, d=[24,24,1], t=traj_rad)
```

To get space separated lists (e.g., resize [-c] dim1 size1 ... dimn), use Tuple type:

```
>>> ksp_zerop = B.resize(lowres_ksp,c=(0,308,1,308))
```

class mr_utils.bart.bartholomew.**BartholomewObject**
Bartholomew object - more simple Python interface for BART.

Examples

User is meant to import instance Bartholomew, e.g.,

```
>>> from mr_utils.bart import Bartholomew as B
```

format_args(*args*)
Take in positional function arguments and format for command-line.

format_kwargs(*kwargs*)
Take in named function arguments and format for command-line.

get_num_outputs()
Return how many values the caller is expecting

2.1.3 BART Client

Connect to BART over a network.

I do not believe this is working currently!

Uses paramiko to connect to a network machine (could be your own machine), opens an instance of BART and returns the result.

mr_utils.bart.client.**client**(*num_out*, *cmd*, *files*, *host=None*, *username=None*, *password=None*, *root_dir=None*)
BART client.

Parameters

- **num_out** (*int*) – Number of expected variables returned.
- **cmd** (*str*) – BART command to be run.
- **files** (*list*) – Any files to be provided to BART.
- **host** (*str, optional*) – IP address of machine we want to connect to.
- **username** (*str, optional*) – username to sign in with.
- **password** (*str, optional*) – password to use for sign in (will be plain-text!)
- **root_dir** (*str, optional*) – Root directory to run BART out of.

2.2 Coils

2.2.1 Coil PCA

Coil compression using principal component analysis.

mr_utils.coils.coil_combine.coil_pca.**coil_pca**(*coil_ims*, *coil_dim=-1*, *n_components=4*, *give_explained_var=False*, *real_imag=True*, *debug_level=30*)

Reduce the dimensionality of the coil dimension using PCA.

Parameters

- **coil_ims** (*array_like*) – Coil images.
- **coil_dim** (*int, optional*) – Coil axis, default is last axis.
- **n_components** (*int, optional*) – How many principal components to keep.
- **give_explained_var** (*bool, optional*) – Return explained variance for real,imag decomposition
- **real_imag** (*bool, optional*) – Perform PCA on real/imag parts separately or mag/phase.
- **debug_level** (*logging_level, optional*) – Verbosity level to set logging module.

Returns

- **coil_ims_pca** (*array_like*) – Compressed coil images representing n_components principal components.
- **expl_var** (*array_like, optional*) – complex valued 1D vector representing explained variance. Is returned if *give_explained_var=True*

mr_utils.coils.coil_combine.coil_pca.**python_pca**(*X*, *n_components=False*)

Python implementation of principal component analysis.

To verify I know what sklearn's PCA is doing.

Parameters

- **X** (*array_like*) – Matrix to perform PCA on.
- **n_components** (*int, optional*) – Number of components to keep.

Returns **P** – n_component principal components of X.

Return type array_like

2.2.2 GS Coil Combine Comparison

Functions to compare coil combination methods.

This actually might belong in the examples, and it needs to be checked to make sure it still works.

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**comparison_knee**()

Coil by coil, Walsh method, and Inati iterative method for knee data.

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**comparison_numerical_phantom**(*SNR=N*

Compare coil by coil, Walsh method, and Inati iterative method.

Parameters **SNR** (*float*) – Signal to noise ratio.

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**get_coil_sensitivity_maps**()

Simulate coil sensitivity maps.

Returns **csms** – List of coil sensitivity maps (arrays), one for each coil.

Return type list

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**get_numerical_phantom_params**(*SNR=N*

Preset parameters for a numerical cylindrical phantom.

Parameters **SNR** (*float*) – Signal to noise ratio, calculated: std = avg_signal/SNR.

Returns **params** – Parameter dictionary including *noise_std*, *dim*, *pc_vals*, and *coil_nums* fields.

Return type dictionary

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**get_true_im_numerical_phantom**()

Get reference bSSFP simulated phantom.

As the geometric solution to the elliptical signal model still has some residual banding, do it a few times at a bunch of different phase cycles to remove virually all banding. This ensures that the contrast will be comparable to the banded phantoms.

Returns **true_im** – Banding free reference image with true bSSFP contrast.

Return type array_like

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**ripple**(*im0*)

Calculate % ripple metric using local patches of line.

Parameters **im0** (*array_like*) – Image to calculate ripple of.

Returns Percent ripple calculated by using local patches along a line through the center of *im0*

Return type float

mr_utils.coils.gs_comparison.gs_coil_combine_comparison.**ripple_normal**(*im*)

Calculate % ripple metric.

Parameters **im** (*array_like*) – Image to calculate ripple of.

Returns Percent ripple.

Return type float

Notes

A horizontal line is drawn through the center of the image. The percent ripple is calculated along this line.

2.3 Config

Provide an easy way to set things like gadgetron host, port, etc.

The ProfileConfig object will create (if it's not already created) a file called 'profiles.config' in the top level of the project (same directory as setup.py). This file contains one or more profiles, one and only one of which must be set as active. A profile contains ports and hostnames and other parameters to use for the gadgetron, MATLAB, siemens_to_ismrmrd, etc. clients.

The config files use python's configparser format. See implementation for details.

Examples

A sample *profiles.config* file might look like:

```
[default]
gadgetron.host = localhost
gadgetron.port = 9002

[workcomp]
gadgetron.host = 10.8.1.12
gadgetron.port = 9002
matlab.host = 10.8.1.12
matlab.port = 9999
matlab.bufsize = 1024

[config]
active = workcomp
```

class mr_utils.config.config.**ProfileConfig**(*filename=None*)

ProfileConfig allows object oriented interaction with profiles.config.

activate_profile(*profile*)

Assign a profile to be active.

Parameters **profile** (*str*) – Profile label to make active.

Notes

All other profiles will still persist, but will not be used. Only one profile may be active at a time.

create_profile(*profile_name*, *args=None*)

Create a new profile.

Parameters

- **profile_name** (*str*) – New profile's label.
- **args** (*dict*) – key, value pairs of profile's attributes.

get_config_val(*key*)

Retrieve a config value.

Parameters **key** (*str*) – Key of the (key, value) pair of the value to be looked up.

Returns Value associated with *key*

Return type value

set_config(*args*, *profile=None*)

Update profile configuration files.

Parameters

- **profile** (*str*) – The profile to update.
- **args** (*dict*) – Dictionary of key, value updates.

Notes

```
Keys -> Values:
'gadgetron.host' -> (string) ip-address/hostname/etc
'gadgetron.port' -> (int) port number
```

update_file()
Update profiles.config by overwriting contents.

2.4 Compressed Sensing

2.4.1 UFT

Undersampled Fourier transform encoding model.

I'm calling "encoding model" how we encode the image domain signal to get to the acquisiton domain. In the case of MR, we measure k-space of the image we want, so the encoding model is simply the Fourier transform (ignoring all the other complications...). This object provides methods to go into k-space and get back out assuming we undersample according to some mask.

forward_ortho, inverse_ortho are probably the ones you want.

class mr_utils.cs.models.UFT.**UFT**(*samp*, *axes=None*, *scale=True*)
Undersampled Fourier Transform (UFT) data acquisiton model.

samp
Boolean sampling pattern.

Type array_like

axes
Axes to perform FFT over.

Type tuple

scale
Whether or not to scale ortho transforms.

Type bool

forward(*x*, *axes=None*)
Fourier encoding with binary undersampling pattern applied.

Parameters

- **x** (*array_like*) – Matrix to be transformed.
- **axes** (*tuple*) – Dimensions to Fourier transform if x is not 2d.

Returns Fourier transform (no fftshift) of *x* with sampling mask applied.

Return type array_like

Notes

This forward transform has no fftshift applied.

forward_ortho(*x*, *axes=None*)
Normalized Fourier encoding with binary undersampling.

Parameters

- **x** (*array_like*) – Matrix to be transformed.
- **axes** (*tuple, optional*) – Dimensions to perform FFT2 over.

Returns Fourier transform of *x* with sampling mask applied and normalized.

Return type array_like

Notes

This forward transform applied fftshift before FFT and after.

forward_s(*x*)
Fourier encoding with binary undersampling pattern applied.

Parameters **x** (*array_like*) – Matrix to be transformed.

Returns Fourier transform (with fftshift) of *x* with sampling mask applied.

Return type array_like

Notes

This forward transform applies fftshift before masking.

inverse(*x*, *axes=None*)
Inverse fourier encoding.

Parameters

- **x** (*array_like*) – Matrix to be transformed.
- **axes** (*tuple*) – Dimensions to Fourier transform if x is not 2d.

Returns Inverse fourier transform of *x*.

Return type array_like

inverse_ortho(*x*, *axes=None*)
Inverse Normalized Fourier encoding.

Parameters

- **x** (*array_like*) – Matrix to be transformed.
- **axes** (*tuple, optional*) – Dimensions to Fourier transform if x is not 2d.

Returns Inverse fourier transform of *x*, fftshifted, and normalized.

Return type array_like

Notes

This transform applied ifftshift before and after ifft2.

inverse_s(*x*)
Inverse fourier encoding with fftshift.

Parameters **x** (*array_like*) – Matrix to be transformed.

Returns Inverse Fourier transform (with fftshift) of *x*

Return type array_like

Notes

This inverse transform applies fftshift.

2.4.2 GD Fourier Encoded TV

Gradient descent algorithm for Fourier encoding model and TV constraint.

mr_utils.cs.convex.gd_fourier_encoded_tv.**GD_FE_TV**(*kspace*, *samp*, *alpha=0.5*, *lam=0.01*, *do_reordering=False*, *im_true=None*, *ignore_residual=False*, *disp=False*, *maxiter=200*)

Gradient descent for Fourier encoding model and TV constraint.

Parameters

- **kspace** (*array_like*) – Measured image.
- **samp** (*array_like*) – Sampling mask.
- **alpha** (*float, optional*) – Step size.
- **lam** (*float, optional*) – TV constraint weight.
- **do_reordering** (*bool, optional*) – Whether or not to reorder for sparsity constraint.
- **im_true** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to break out of loop if resid increases.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.

Returns **m_hat** – Estimate of im_true.

Return type array_like

Notes

Solves the problem:

$$\min_x ||d - \text{FT}(I \odot S)||_2^2 + \lambda \text{TV}(I)$$

where d is measured k-space, I is the image estimate, S is the undersampling mask, and TV is the total variation operator.

If *im_true=None*, then MSE will not be calculated.

2.4.3 GD TV

Gradient descent with built in TV and flexible encoding model.

mr_utils.cs.convex.gd_tv.**GD_TV**(*y*, *forward_fun*, *inverse_fun*, *alpha=0.5*, *lam=0.01*, *do_reordering=False*, *x=None*, *ignore_residual=False*, *disp=False*, *maxiter=200*)

Gradient descent for a generic encoding model and TV constraint.

Parameters

- **y** (*array_like*) – Measured data (i.e., y = Ax).
- **forward_fun** (*callable*) – A, the forward transformation function.

- **inverse_fun** (*callable*) – A^H, the inverse transformation function.
- **alpha** (*float, optional*) – Step size.
- **lam** (*float, optional*) – TV constraint weight.
- **do_reordering** (*bool, optional*) – Whether or not to reorder for sparsity constraint.
- **x** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to break out of loop if resid increases.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

Solves the problem:

$$\min_x ||y - Ax||_2^2 + \lambda \text{TV}(x)$$

If *x=None*, then MSE will not be calculated.

2.4.4 Proximal GD

Proximal Gradient Descent.

Flexible encoding model, flexible sparsity model, and flexible reordering model. This is the one I would use out of all the ones I've coded up. Might be slower than the others as there's a little more checking to do each iteration.

mr_utils.cs.convex.proximal_gd.**proximal_GD**(*y*, *forward_fun*, *inverse_fun*, *sparsify*, *unsparsify*, *reorder_fun=None*, *mode='soft'*, *alpha=0.5*, *alpha_start=0.5*, *thresh_sep=True*, *selective=None*, *x=None*, *ignore_residual=False*, *ignore_mse=True*, *ignore_ssim=True*, *disp=False*, *maxiter=200*, *strikes=0*)

Proximal gradient descent for generic encoding/sparsity model.

Parameters

- **y** (*array_like*) – Measured data (i.e., y = Ax).
- **forward_fun** (*callable*) – A, the forward transformation function.
- **inverse_fun** (*callable*) – A^H, the inverse transformation function.
- **sparsify** (*callable*) – Sparsifying transform.
- **unsparsify** (*callable*) – Inverse sparsifying transform.
- **reorder_fun** (*callable, optional*) – Reordering function.
- **unreorder_fun** (*callable, optional*) – Inverse reordering function.

- **mode** (*{'soft', 'hard', 'garotte', 'greater', 'less'}, optional*) – Thresholding mode.
- **alpha** (*float or callable, optional*) – Step size, used for thresholding.
- **alpha_start** (*float, optional*) – Initial alpha to start with if alpha is callable.
- **thresh_sep** (*bool, optional*) – Whether or not to threshold real/imag individually.
- **selective** (*bool, optional*) – Function returning indicies of update to keep at each iter.
- **x** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to break out of loop if resid increases.
- **ignore_mse** (*bool, optional*) – Whether or not to break out of loop if MSE increases.
- **ignore_ssim** (*bool, optional*) – Whether or not to break out of loop if SSIM increases.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.
- **strikes** (*int, optional*) – Number of ending conditions tolerated before giving up.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

Solves the problem:

$$\min_x ||y - Ax||_2^2 + \lambda \text{Sparsify}(x)$$

If *x=None*, then MSE will not be calculated. You probably want *mode='soft'*. For the other options, see docs for pywt.threshold. *selective=None* will not throw away any updates.

2.4.5 CoSaMP

Compressive sampling matching pursuit (CoSaMP) algorithm.

This implementation currently does not handle complex signals.

mr_utils.cs.greedy.cosamp.**cosamp**(*A*, *y*, *k*, *lstsq='exact'*, *tol=1e-08*, *maxiter=500*, *x=None*, *disp=False*)

Compressive sampling matching pursuit (CoSaMP) algorithm.

Parameters

- **A** (*array_like*) – Measurement matrix.
- **y** (*array_like*) – Measurements (i.e., y = Ax).
- **k** (*int*) – Number of expected nonzero coefficients.
- **lstsq** (*{'exact', 'lm', 'gd'}, optional*) – How to solve intermediate least squares problem.

- **tol** (*float, optional*) – Stopping criteria.
- **maxiter** (*int, optional*) – Maximum number of iterations.
- **x** (*array_like, optional*) – True signal we are trying to estimate.
- **disp** (*bool, optional*) – Whether or not to display iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

lstsq function - 'exact' solves it using numpy's linalg.lstsq method. - 'lm' uses solves with the Levenberg-Marquardt algorithm. - 'gd' uses 3 iterations of a gradient descent solver.

Implements Algorithm 8.7 from[1].

References

2.4.6 AMP

2D implementation of Approximate message passing algorithms.

See docstring of amp2d for reference implementation details. It's companion is LCAMP. What's interesting is that they circular shift in the transform domain. I'm not sure why they do that, but empirically it seems to work!

The wavelet transform is about what they are using. I'm trying to keep the implementation as simple as possible, so I used a built in transform from PyWavelets that is close, but I'm not sure why it doesn't match up completely.

mr_utils.cs.thresholding.amp.**amp2d**(*y*, *forward_fun*, *inverse_fun*, *sigmaType=2*, *randshift=False*, *tol=1e-08*, *x=None*, *ignore_residual=False*, *disp=False*, *maxiter=100*)

Approximate message passing using wavelet sparsifying transform.

Parameters

- **y** (*array_like*) – Measurements, i.e., y = Ax.
- **forward_fun** (*callable*) – A, the forward transformation function.
- **inverse_fun** (*callable*) – A^H, the inverse transformation function.
- **sigmaType** (*int*) – Method for determining threshold.
- **randshift** (*bool, optional*) – Whether or not to randomly circular shift every iteration.
- **tol** (*float, optional*) – Stop when stopping criteria meets this threshold.
- **x** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to ignore stopping criteria.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.

Returns **wn** – Estimate of x.

Return type array_like

[1] Eldar, Yonina C., and Gitta Kutyniok, eds. Compressed sensing: theory and applications. Cambridge University Press, 2012.

Notes

Solves the problem:

$$\min_x ||\Psi(x)||_1 \text{ s.t. } ||y - \text{forward}(x)||_2^2 < \epsilon^2$$

The CDF-97 wavelet is used. If *x=None*, then MSE will not be calculated.

Algorithm described in[1], based on MATLAB implementation found at[2].

References

2.4.7 IHT Fouier Encoded TV

Iterative hard thresholding using Fourier encoding model and TV constraint.

mr_utils.cs.thresholding.iht_fourier_encoded_total_variation.**IHT_FE_TV**(*kspace*, *samp*, *k*, *mu=1*, *tol=1e-08*, *do_reordering=False*, *x=None*, *ignore_residual=False*, *disp=False*, *maxiter=500*)

IHT for Fourier encoding model and TV constraint.

Parameters

- **kspace** (*array_like*) – Measured data in k-space.
- **samp** (*array_like*) – Sampling mask.
- **k** (*int*) – Sparsity measure (number of nonzero coefficients expected).
- **mu** (*float, optional*) – Step size.
- **tol** (*float, optional*) – Stop when stopping criteria meets this threshold.
- **do_reordering** (*bool, optional*) – Reorder column-stacked true image.
- **x** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to break out of loop if resid increases.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

[1] "Message Passing Algorithms for CS" Donoho et al., PNAS 2009;106:18914
[2] http://kyungs.bol.ucla.edu/Site/Software.html

Notes

Solves the problem:

$$\min_x ||d - \text{FT}(x)||_2^2 \text{ s.t. } ||\text{TV}(x)||_0 \leq k$$

If *im_true=None*, then MSE will not be calculated.

2.4.8 IHT TV

Iterative hard thresholding with variable encoding model, uses TV.

mr_utils.cs.thresholding.iht_tv.**IHT_TV**(*y*, *forward_fun*, *inverse_fun*, *k*, *mu=1*, *tol=1e-08*, *do_reordering=False*, *x=None*, *ignore_residual=False*, *disp=False*, *maxiter=500*)

IHT for generic encoding model and TV constraint.

Parameters

- **y** (*array_like*) – Measured data, i.e., y = Ax.
- **forward_fun** (*callable*) – A, the forward transformation function.
- **inverse_fun** (*callable*) – A^H, the inverse transformation function.
- **k** (*int*) – Sparsity measure (number of nonzero coefficients expected).
- **mu** (*float, optional*) – Step size.
- **tol** (*float, optional*) – Stop when stopping criteria meets this threshold.
- **do_reordering** (*bool, optional*) – Reorder column-stacked true image.
- **x** (*array_like, optional*) – The true image we are trying to reconstruct.
- **ignore_residual** (*bool, optional*) – Whether or not to break out of loop if resid increases.
- **disp** (*bool, optional*) – Whether or not to display iteration info.
- **maxiter** (*int, optional*) – Maximum number of iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

Solves the problem:

$$\min_x ||y - Ax||_2^2 \text{ s.t. } ||\text{TV}(x)||_0 \leq k$$

If *x=None*, then MSE will not be calculated.

2.4.9 Iterative Hard Thresholding

Simple iterative hard thresholding algorithm.

mr_utils.cs.thresholding.iterative_hard_thresholding.**IHT**(*A*, *y*, *k*, *mu=1*, *maxiter=500*, *tol=1e-08*, *x=None*, *disp=False*)

Iterative hard thresholding algorithm (IHT).

Parameters

- **A** (*array_like*) – Measurement matrix.
- **y** (*array_like*) – Measurements (i.e., y = Ax).
- **k** (*int*) – Number of expected nonzero coefficients.
- **mu** (*float, optional*) – Step size.
- **maxiter** (*int, optional*) – Maximum number of iterations.
- **tol** (*float, optional*) – Stopping criteria.
- **x** (*array_like, optional*) – True signal we are trying to estimate.
- **disp** (*bool, optional*) – Whether or not to display iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

Solves the problem:

$$\min_x ||y - Ax||_2^2 \text{ s.t. } ||x||_0 \leq k$$

If *disp=True*, then MSE will be calculated using provided x. *mu=1* seems to satisfy Theorem 8.4 often, but might need to be adjusted (usually < 1). See normalized IHT for adaptive step size.

Implements Algorithm 8.5 from[1].

References

2.4.10 Iterative Soft Thresholding

Iterative soft thresholding algorithm.

mr_utils.cs.thresholding.iterative_soft_thresholding.**IST**(*A*, *y*, *mu=0.8*, *theta0=None*, *k=None*, *maxiter=500*, *tol=1e-08*, *x=None*, *disp=False*)

Iterative soft thresholding algorithm (IST).

Parameters

- **A** (*array_like*) – Measurement matrix.
- **y** (*array_like*) – Measurements (i.e., y = Ax).
- **mu** (*float, optional*) – Step size (theta contraction factor, 0 < mu <= 1).
- **theta0** (*float, optional*) – Initial threshold, decreased by factor of mu each iteration.

[1] Eldar, Yonina C., and Gitta Kutyniok, eds. Compressed sensing: theory and applications. Cambridge University Press, 2012.

- **k** (*int, optional*) – Number of expected nonzero coefficients.
- **maxiter** (*int, optional*) – Maximum number of iterations.
- **tol** (*float, optional*) – Stopping criteria.
- **x** (*array_like, optional*) – True signal we are trying to estimate.
- **disp** (*bool, optional*) – Whether or not to display iterations.

Returns **x_hat** – Estimate of x.

Return type array_like

Notes

Solves the problem:

$$\min_x ||y - Ax||_2^2 \text{ s.t. } ||x||_0 \leq k$$

If *disp=True*, then MSE will be calculated using provided x. If *theta0=None*, the initial threshold of the IHT will be used as the starting theta.

Implements Equations [22-23] from[1]

References

2.4.11 Normalized IHT

Normalized iterative hard thresholding algorithm.

mr_utils.cs.thresholding.normalized_iht.**nIHT**(*A*, *y*, *k*, *c=0.1*, *kappa=None*, *x=None*, *maxiter=200*, *tol=1e-08*, *disp=False*)

Normalized iterative hard thresholding.

Parameters

- **A** (*array_like*) – Measurement matrix
- **y** (*array_like*) – Measurements (i.e., y = Ax)
- **k** (*int*) – Number of nonzero coefficients preserved after thresholding.
- **c** (*float, optional*) – Small, fixed constant. Tunable.
- **kappa** (*float, optional*) – Constant, > 1/(1 - c).
- **x** (*array_like, optional*) – True signal we want to estimate.
- **maxiter** (*int, optional*) – Maximum number of iterations (of the outer loop).
- **tol** (*float, optional*) – Stopping criteria.
- **disp** (*bool, optional*) – Whether or not to display iteration info.

Returns **x_hat** – Estimate of x.

Return type array_like

[1] Rani, Meenu, S. B. Dhok, and R. B. Deshmukh. "A systematic review of compressive sensing: Concepts, implementations and applications." IEEE Access 6 (2018): 4875-4894.

Notes

Implements Algorithm 8.6 from[1].

References

2.4.12 Ordinator

Combinatorial optimization to find permutation maximizing sparsity.

mr_utils.cs.ordinator.**get_xhat**(*locs*, *N*, *k*, *inverse*, *pdf_ref*, *pdf*, *pdf_metric*)
Compute xhat for given coefficient locs using basinhopping.

Parameters

- **locs** (*array_like*) – Coefficient location indices.
- **N** (*int*) – Length of the desired signal (also number of coefficients in total).
- **k** (*int*) – Desired sparsity level.
- **inverse** (*callable*) – Inverse sparsifying transform.
- **pdf_ref** (*array_like*) – Reference pdf of the prior to compare against.
- **pdf** (*callable*) – Function that estimates pixel intensity distribution.
- **pdf_metric** (*callable*) – Function that returns the distance between pdfs.

Returns

- **xhat** (*array_like*) – Inverse transform of coeffs.
- **locs** (*array_like*) – Indices of non-zero coefficients.
- **coeffs** (*array_like*) – Coefficients of xhat.

mr_utils.cs.ordinator.**obj**(*ck*, *N*, *locs*, *inverse*, *pdf_ref*, *pdf*, *pdf_metric*)
Objective function for basinhopping.

mr_utils.cs.ordinator.**ordinator1d**(*prior*, *k*, *forward*, *inverse*, *chunksize=10*, *pdf=None*, *pdf_metric=None*, *sparse_metric=None*, *disp=False*)
Find permutation that maximizes sparsity of 1d signal.

Parameters

- **prior** (*array_like*) – Prior signal estimate to base ordering.
- **k** (*int*) – Desired sparsity level.
- **forward** (*callable*) – Sparsifying transform.
- **inverse** (*callable*) – Inverse sparsifying transform.
- **chunksize** (*int, optional*) – Chunk size for parallel processing pool.
- **pdf** (*callable, optional*) – Function that estimates pixel intensity distribution.
- **pdf_metric** (*callable, optional*) – Function that returns the distance between pdfs.
- **sparse_metric** (*callable, optional*) – Metric to use to measure sparsity. Uses l1 norm by default.

[1] Eldar, Yonina C., and Gitta Kutyniok, eds. Compressed sensing: theory and applications. Cambridge University Press, 2012.

- **disp** (*bool, optional*) – Whether or not to display coefficient plots at the end.

Returns Reordering indices.

Return type array_like

Raises `ValueError` – If disp=True and forward function is not provided.

Notes

pdf_method=None uses histogram. pdf_metric=None uses l2 norm. If disp=True then forward transform function must be provided. Otherwise, forward is not required, only inverse.

pdf_method should assume the signal will be bounded between (-1, 1). We do this by always normalizing a signal before computing pdf or comparing.

class mr_utils.cs.ordinator.**pdf_default**(*prior*)

Picklable object for computing pdfs.

Uses histogram to estimate pdf.

N

Size of signal.

Type int

lims

Upper and lower bounds for range of histogram.

Type array_like or tuple

pdf_ref

pdf estimate of prior. Used to compare to pdf(xhat).

Type array_like

bins

bin locations used for construction of pdf_ref.

Type array_like

pdf(*x*)

Estimate the pdf of x.

Parameters **x** (*array_like*) – Signal to get pdf estimate of.

Returns Histogram of x.

Return type array_like

Notes

Will report when xhat has a value outside of range of pdf_ref.

mr_utils.cs.ordinator.**pdf_metric_default**(*x*, *y*)

Default pdf metric, l2 norm.

Parameters

- **x** (*array_like*) – First pdf.
- **y** (*array_like*) – Second pdf.

Returns l2 norm between x and y.

Return type float

mr_utils.cs.ordinator.**search_fun**(*locs*, *N*, *k*, *inverse*, *pdf_ref*, *pdf*, *pdf_metric*)
Return function for parallel loop.

Parameters

- **locs** (*array_like*) – Coefficient location indices.
- **N** (*int*) – Length of the desired signal (also number of coefficients in total).
- **k** (*int*) – Desired sparsity level.
- **inverse** (*callable*) – Inverse sparsifying transform.
- **pdf_ref** (*array_like*) – Reference pdf of the prior to compare against.
- **pdf** (*callable*) – Function that estimates pixel intensity distribution.
- **pdf_metric** (*callable*) – Function that returns the distance between pdfs.

Returns

- **locs** (*array_like*) – Indices of non-zero coefficients.
- **vals** (*array_like*) – Values of coefficients at locations given by locs.
- *float* – Measure of difference between pdf_ref and pdf(xhat).

2.5 Gadgetron

2.5.1 Client

Gadgetron client for running on network machines.

Adapted from https://github.com/gadgetron/gadgetron-python-ismrmrd-client.git Keeps same command line interface, but allows for import into scripts.

mr_utils.gadgetron.client.**client**(*data*, *address=None*, *port=None*, *outfile=None*, *in_group='/dataset'*, *out_group=None*, *config='default.xml'*, *config_local=None*, *script=None*, *existing_modules=['numpy', 'scipy', 'h5py']*, *script_dir=None*, *verbose=False*)
Send acquisitions to Gadgetron.

This client allows you to connect to a Gadgetron server and process data.

Parameters

- **data** (*str or array_like*) – Input file with file extension or numpy array.
- **address** (*str, optional*) – Hostname of Gadgetron. If not set, taken from profile config.
- **port** (*int, optional*) – Port to connect to. If not set, taken from profile config.
- **outfile** (*str, optional*) – If provided, output will be saved to file with this name.
- **in_group** (*str, optional*) – If input is hdf5, input data group name.
- **out_group** (*str, optional*) – Output group name if file is written.
- **config** (*xml_str, optional*) – Remote configuration file.
- **config_local** (*xml_str, optional*) – Local configuration file.

- **script** (*str, optional*) – File path to the Python script to be bundled and transfered.
- **existing_modules** (*list, optional*) – Python packages to exclude from bundling.
- **script_dir** (*str, optional*) – Directory to send script on remote machine.
- **verbose** (*bool, optional*) – Verbose mode.

Returns

- **data** (*array_like*) – Image from Gadgetron
- **header** (*xml*) – Header from Gadgetron

Raises

- `NotImplementedError` – *script* bundling is not currently implemented.
- `Exception` – *data* is not provided in the correct format.

Notes

out_group=None will use the current date as the group name.

2.5.2 Configurations

Default Config

Example Gadgetron config generation.

`mr_utils.gadgetron.configs.default.`**`default`**`()`
Default config file, default.xml.

Generates[1].

References

Distributed Config

Example generation of distributed gadget configs.

`mr_utils.gadgetron.configs.distributed.`**`distributed_default`**`()`
Generates distributed_default.xml.

Generates[1].

References

`mr_utils.gadgetron.configs.distributed.`**`distributed_image_default`**`()`
Generates distributed_image_default.xml.

Generates[2].

[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/mri_core/config/default.xml
[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/distributed/config/distributed_default.xml
[2] https://github.com/gadgetron/gadgetron/blob/master/gadgets/distributed/config/distributed_image_default.xml

References

EPI Config

Example EPI configurations.

mr_utils.gadgetron.configs.epi.**epi**()
Generates epi.xml.

Generates[1].

References

mr_utils.gadgetron.configs.epi.**epi_gtplus_grappa**()
GT Plus configuration file for general 2D epi reconstruction.

Generates[2].

References

Generic Config

Generic Gadgetron configuration files.

mr_utils.gadgetron.configs.generic.**generic_cartesian_grappa**()
Generic_Cartesian_Grappa.xml.

Generates[1].

References

GRAPPA Config

Gadgetron configs for GRAPPA gadgets.

mr_utils.gadgetron.configs.grappa.**grappa_cpu**()
Generates grappa_cpu.xml.

Generates[1].

References

mr_utils.gadgetron.configs.grappa.**grappa_float_cpu**()
Generates grappa_float_cpu.xml.

Generates[2].

[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/epi/epi.xml
[2] https://github.com/gadgetron/gadgetron/blob/master/gadgets/epi/epi_gtplus_grappa.xml
[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/mri_core/config/Generic_Cartesian_Grappa.xml
[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/grappa/config/grappa_cpu.xml
[2] https://github.com/gadgetron/gadgetron/blob/master/gadgets/grappa/config/grappa_float_cpu.xml

References

mr_utils.gadgetron.configs.grappa.**grappa_unoptimized_cpu**()
Generates grappa_unoptimized_cpu.xml.

Generates[3].

References

mr_utils.gadgetron.configs.grappa.**grappa_unoptimized_float_cpu**()
Generates grappa_unoptimized_float_cpu.xml.

Generates[4].

References

Python Config

Configs including python Gadgets.

mr_utils.gadgetron.configs.python.**python**()
python.xml

Generates[1].

References

mr_utils.gadgetron.configs.python.**python_short**()
python_short.xml

Generates[2].

References

2.5.3 Gadgets

GS Gadget

Gadgetron gadget, config file to do GS recon on each coil of GRAPPA recon.

RMS Coil Combine

Example Python Gadget.

Based on[1].

[3] https://github.com/gadgetron/gadgetron/blob/master/gadgets/grappa/config/grappa_unoptimized.xml
[4] https://github.com/gadgetron/gadgetron/blob/master/gadgets/grappa/config/grappa_unoptimized_float.xml
[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/python/config/python.xml
[2] https://github.com/gadgetron/gadgetron/blob/master/gadgets/python/config/python_short.xml
[1] https://github.com/gadgetron/gadgetron/blob/master/gadgets/python/gadgets/rms_coil_combine.py

References

class mr_utils.gadgetron.gadgets.rms_coil_combine.**RMSCoilCombine**
Gadget that using RMS method to combine coils.

process(*h*, *im*)
Process the data, combine coils.

process_config(*_cfg*)
Process XML configuration file.

2.6 Gridding

2.6.1 SC-GROG

get_gx_gy

Self calibrating GROG GRAPPA kernels.

Based on the MATLAB implementation found here: https://github.com/edibella/Reconstruction/blob/master/%2BGROG/get_Gx_Gy.m

mr_utils.gridding.scgrog.get_gx_gy.**get_gx_gy**(*kspace*, *traj=None*, *kxs=None*, *kys=None*, *cartdims=None*)
Compute Self Calibrating GRAPPA Gx and Gy operators.

Parameters

- **kspace** (*array_like*) – kspace samples.
- **traj** (*array_like*) – k-space trajectory.
- **kxs** (*array_like*) – kx coordinates.
- **kys** (*array_like*) – ky coordinates.
- **cartdims** (*tuple*) – Expected dimensions of cartesian grid.

Returns

- **Gx** (*array_like*) – GRAPPA kernel in x
- **Gy** (*array_like*) – GRAPPA kernel in y

scgrog

Self calibrating GROG implementation.

Based on the MATLAB GROG implementation found here: https://github.com/edibella/Reconstruction

mr_utils.gridding.scgrog.scgrog.**fracpowers**(*idx*, *Gx*, *Gy*, *dkxs*, *dkys*)
Wrapper function to use during parallelization.

Parameters

- **idx** (*array_like*) – Indices of current fractioinal power GRAPPA kernels
- **Gx** (*array_like*) – GRAPPA kernel in x
- **Gy** (*array_like*) – GRAPPA kernel in y

- **dkxs** (*array_like*) – Differential x k-space coordinates
- **dkys** (*array_like*) – Differential y k-space coordinates

Returns

- **ii** (*array_like*) – row indices
- **jj** (*array_like*) – col indices
- **Gxf** (*array_like*) – Fractional power of GRAPPA kernel in x
- **Gyf** (*array_like*) – Fractional power of GRAPPA kernel in y

mr_utils.gridding.scgrog.scgrog.**grog_interp**(*kspace*, *Gx*, *Gy*, *traj*, *cartdims*)

Moves radial k-space points onto a cartesian grid via the GROG method.

Parameters

- **kspace** (*array_like*) – A 3D (sx, sor, soc) slice of k-space
- **Gx** (*array_like*) – The unit horizontal cartesian GRAPPA kernel
- **Gy** (*array_like*) – Unit vertical cartesian GRAPPA kernel
- **traj** (*array_like*) – k-space trajectory
- **cartdims** (*tuple*) – (nrows, ncols), size of Cartesian grid

Returns Interpolated cartesian kspace.

Return type array_like

mr_utils.gridding.scgrog.scgrog.**scgrog**(*kspace*, *traj*, *Gx*, *Gy*, *cartdims=None*)

Self calibrating GROG interpolation.

Parameters

- **kspace** (*array_like*) – A 4D (sx, sor, nof, soc) matrix of complex k-space data
- **traj** (*array_like*) – k-space trajectory
- **Gx** (*array_like*) – The unit horizontal cartesian GRAPPA kernel
- **Gy** (*array_like*) – Unit vertical cartesian GRAPPA kernel
- **cartdims** (*tuple*) – Size of Cartesian grid.

Returns

- **kspace_cart** (*array_like*) – Cartesian gridded k-space.
- **mask** (*array_like*) – Boolean mask where kspace is nonzero.

Notes

If cartdims=None, we'll guess the Cartesian dimensions are (kspace.shape[0], kspace.shape[0], kspace.shape[2], kspace.shape[3]).

2.7 Data Loading

2.7.1 load_mat

Load data from MATLAB file type.

Uses scipy.io.loadmat to load recent versions of .MAT files. Version 7.3 is supported. It'll try to make some intelligent guesses if it runs into trouble, meaning, 'it will die trying!'. If you don't like that philosophy, go ahead and use scipy.io.loadmat directly.

mr_utils.load_data.mat.**deal_with_7_3**(*data*)
Clean up data structures for MATLAB 7.3.

Parameters **data** (*array_like*) – Data from .mat file.

Notes

Version 7.3 has a structured datatype that needs to be translated as a complex number.

mr_utils.load_data.mat.**load_mat**(*filename*, *key=None*)
Load data from .MAT file.

Parameters

- **filename** (*str*) – path to .mat file.
- **key** (*str, optional*) – Specific key to extract.

Returns Contents of mat file.

Return type array_like

Notes

If key=None, all keys will be extracted. If there is only one key, then its value will be provided directly, no dictionary will be returned.

2.7.2 load_raw

Handle loading in and converting Siemens raw data format.

mr_utils.load_data.raw.**load_raw**(*filename*, *use='bart'*, *bart_args='-A'*, *s2i_ROS=True*, *as_ismrmrd=False*)
Load Siemens raw data into numpy array.

Parameters

- **filename** (*str*) – File path and filename of raw data file.
- **use** (*{'bart', 's2i', 'rdi'}, optional*) – Method to use to read in raw data.
- **bart_args** (*dict, optional*) – Arguments to pass to BART.
- **s2i_ROS** (*bool, optional*) – Remove oversampling in readout when using use='s2i'.
- **as_ismrmrd** (*bool, optional*) – Leave as ismrmrd data type.

Returns

- **data** (*array_like*) – Data from raw data file.
- **ismrmrd_dataset** (*optional*) – Returned if use='s2i' and as_ismrmrd=True.

Raises

- SystemError – If BART is requested but not installed or if siemens_to_ismrmrd is requested but not installed

- `Exception` – If BART encounters an error while running or if siemens_to_ismrmrd is encounters an error while running, or a valid use option is not specified.

Notes

use: - bart – BART twix raw data reader - s2i – siemens_to_ismrmrd - rdi – rawdatarinator

2.7.3 pyport

Python port of siemens_to_ismrmrd.

Notes

The XProtocol parser (xprot_get_val) is a string-search based implementation, not an actual parser, so it's really slow, but does get the job done very well. Next steps would be to figure out how to speed this up or rewrite the parser to work with everything. I was working on a parser but was stuck on how to handle some of Siemens' very strange syntax.

There are several different XML libraries being used. xml.etree was my preference, so that's what I started with. I needed to use xmltodict to convert between dictionaries and xml, because it's quicker/easier to have a dictionary hold the config information as we move along. It turns out that schema verification is not supported by xml.etree, so that's when I pulled in lxml.etree – so there's some weirdness trying to get xml.etree and lxml.etree to play together nicely. The last one is pybx – a bizarrely complicated library that the ismrmrd python library uses. I hate the thing and think it's overly complicated for what we need to use it for.

One of the ideas I had was to pull down the schema/parammaps from the interwebs so it would always be current. While this is a neat feature that probably no one will use, it would speed up the raw data conversion to use a local copy instead, even if that means pulling it down the first time and keeping it.

The script to read in an ismrmrd dset provided in ismrmrd-python-tools is great at illustrating how to do it, but is incredibly slow, especiailly if you want to remove oversampling in readout direction. Next steps are to figure out how to quickly read in and process these datasets. I'm kind of put off from using this data format because of how unweildy it is, but I suppose it's better to be an open standards player...

The only datasets I have are cartesian VB17. So there's currently little support for anything else.

Command-line interface has not been looked at in a long time, might not be working still.

`mr_utils.load_data.pyport.`**`pyport`**(*version=False*, *list_embed=False*, *extract=None*, *user_stylesheet=None*, *file=None*, *pMapStyle=None*, *measNum=1*, *pMap=None*, *user_map=None*, *debug=False*, *header_only=False*, *output='output.h5'*, *flash_pat_ref_scan=False*, *append_buffers=False*, *study_date_user_supplied=''*)

Run the program with arguments.

Parameters

- **`version`** (*`bool, optional`*) – Prints converter version and ISMRMRD version
- **`list_embed`** (*`bool, optional`*) – Print list embedded files
- **`extract`** (*`bool, optional`*) – Extract embedded file
- **`user_stylesheet`** (*`str, optional`*) – Provide a parameter stylesheet XSL file
- **`file`** (*`str, optional`*) – SIEMENS dat file
- **`pMapStyle`** (*`str, optional`*) – Parameter stylesheet XSL

- **measNum** (*int, optional*) – Measurement number
- **pMap** (*str, optional*) – Parameter map XML
- **user_map** (*str, optional*) – Provide a parameter map XML file
- **debug** (*bool, optional*) – Debug XML flag
- **header_only** (*bool, optional*) – HEADER ONLY flag (create xml header only)
- **output** (*str, optional*) – ISMRMRD output file
- **flash_pat_ref_scan** (*bool, optional*) – FLASH PAT REF flag
- **append_buffers** (*bool, optional*) – Append protocol buffers
- **study_date_user_supplied** (*str, optional*) – User can supply study date, in the format of yyyy-mm-dd

Returns **ismrmrd_dataset** – Converted raw data

Return type ismrmrd.Dataset

Raises

- `ValueError` – When .dat file is not provided, when user-supplied parameter map XSL stylesheet AND embedded stylesheet are provided at the same time
- `IOError` – When .dat file does not exist, when provided user_stylesheet does not exist, when pMapStyle does not exist
- `NotImplementedError` – When VD raw data file is provided

Notes

Not all Raises are listed currently.

2.7.4 siemens_to_ismrmrd_client

siemens_to_ismrmrd client.

class mr_utils.load_data.siemens_to_ismrmrd_client.**FastTransport**(*sock*)

class mr_utils.load_data.siemens_to_ismrmrd_client.**TqdmWrap**(*iterable=None*, *desc=None*, *total=None*, *leave=True*, *file=None*, *ncols=None*, *mininterval=0.1*, *maxinterval=10.0*, *miniters=None*, *ascii=None*, *disable=False*, *unit='it'*, *unit_scale=False*, *dynamic_ncols=False*, *smoothing=0.3*, *bar_format=None*, *initial=0*, *position=None*, *postfix=None*, *unit_divisor=1000*, *write_bytes=None*, *gui=False*, ***kwargs*)

viewBar(*a*, *b*)
Monitor progress of sftp transfers

mr_utils.load_data.siemens_to_ismrmrd_client.**s2i_client**(*filename*, *put_file=True*, *get_file=True*, *cleanup_raw=True*, *cleanup_processed=True*, *remote_dir='/tmp'*, *host=None*, *port=22*, *username=None*, *ssh_key=None*, *password=None*, *debug_level=20*)

Runs siemens_to_ismrmrd on a remote computer.

Main idea: allow users to use siemens_to_ismrmrd even if they don't have it installed locally. They will, however, require SSH access to computer that does have it installed.

Client puts file on server using SFTP, runs siemens_to_ismrmrd over SSH, and gets the file back using SFTP. Username, password, hostname, and port is retrieved from the active profile in profiles.config. Default port is 22. If no password is found, the RSA SSH key will be used from either the specified directory in profiles.config or, if empty, use '~/.ssh/id_rsa'.

Parameters

- **filename** (*str*) – Raw data (.dat) file on the local machine (if put_file is True) or on the remote machine (if put_file is False).
- **put_file** (*bool, optional*) – Whether or not to copy the raw data file from local to remote.
- **get_file** (*bool, optional*) – Whether or not to copy the processed file from machine to local.

- **cleanup_raw** (*bool, optional*) – Whether or not to delete raw data on remote.
- **cleanup_processed** (*bool, optional*) – Whether or not to delete processed data on remote.
- **remote_dir** (*str, optional*) – Working directory on remote (default in /tmp).
- **host** (*str, optional*) – hostname of remote machine.
- **port** (*int, optional*) – Port of remote machine to connect to.
- **username** (*str, optional*) – Username to use for SSH/SFTP connections.
- **ssh_key** (*str, optional*) – RSA private key file to use for SSH/SFTP connections.
- **password** (*str, optional*) – Password to use fr SSH/SFTP connections (stored in plaintext).
- **debug_level** (*logging_level, optional*) – Level of verbosity; see python logging module.

Returns **dset** – Result of siemens_to_ismrmrd

Return type ismrmrd.Dataset

2.8 MATLAB

2.8.1 client

Connect to network machine running MATLAB to run scripts.

A way to run MATLAB scripts inside python scripts. Meant to run things until I have time to port them to Python. It's meant to match the gadgetron client.

mr_utils.matlab.client.**client_get**(*varnames*, *host=None*, *port=None*, *bufsize=None*)

Get variables from remote MATLAB workspace into python.

Parameters

- **varnames** (*list*) – List of names of variables in MATLAB workspace to get.
- **host** (*str, optional*) – host/ip-address of server running MATLAB.
- **port** (*int, optional*) – port of host to connect to.
- **bufsize** (*int, optional*) – Number of bytes to transmit/recieve at a time.

Returns **vals** – Contents of MATLAB workspace.

Return type dict

Raises ValueError – When transfered matlab workspace file cannot be read. When *varnames* is not a list type.

Notes

Notice that varnames should be a list of strings.

mr_utils.matlab.client.**client_put**(*varnames*, *host=None*, *port=None*, *bufsize=None*)

Put variables from python into MATLAB workspace.

Parameters

- **varnames** (*dict*) – Python variables to be injected into MATLAB workspace.
- **host** (*str, optional*) – host/ip-address of server running MATLAB.
- **port** (*int, optional*) – port of host to connect to.
- **bufsize** (*int, optional*) – Number of bytes to transmit/recieve at a time.

Returns

Return type None

Raises ValueError – When *varnames* is not a dictionary object.

Notes

Notice that varnames should be a dictionary: keys are the desired names of the variables in the MATLAB workspace and values are the python variables.

mr_utils.matlab.client.**client_run**(*cmd*, *host=None*, *port=None*, *bufsize=None*)
Run command on MATLAB server.

Parameters

- **cmd** (*str*) – MATLAB command.
- **host** (*str, optional*) – host/ip-address of server running MATLAB.
- **port** (*int, optional*) – port of host to connect to.
- **bufsize** (*int, optional*) – Number of bytes to transmit/recieve at a time.

Returns

Return type None

Notes

If values are not provided (i.e., None) the values for host, port, bufsize will be taken from the active profile in profiles.config.

mr_utils.matlab.client.**get_socket**(*host*, *port*, *bufsize*)
Open a socket to the machine running MATLAB.

Parameters

- **host** (*str*) – IP address of machine running MATLAB.
- **port** (*int*) – port to connect to.
- **bufsize** (*int*) – Buffer size to use for communication.

Returns

- **sock** (*socket.socket*) – TCP socket for communication
- **host** (*str*) – host ip address
- **port** (*int*) – Port we're communicating over
- **bufsize** (*int*) – Buffer size to use during communication

Notes

If values are not provided (i.e., None) the values for host, port, bufsize will be taken from the active profile in profiles.config.

2.8.2 server

Server to be running on network machine.

Must be running for client to be able to connect. Obviously, alongside this server, MATLAB should also be running.

class mr_utils.matlab.server.**MATLAB**

Object on server allowing server to communicate with MATLAB instance.

done_token

Sequence of symbols to let us know MATLAB is done communicating.

Type contract.done_token

process

Process that runs MATLAB and stays open.

Type subprocess

catch_output(*log_func=None*)

Grab the output of MATLAB on the server.

Parameters **log_func** (*callable, optional*) – Function to use to log output of MATLAB console.

exit()

Send exit command to MATLAB.

get(*varnames*)

Get variables from MATLAB workspace into python as numpy arrays.

Parameters **varnames** (*list*) – List of names of variables in MATLAB workspace to get.

Returns **tmp_filename** – Name of temporary file where MATLAB workspace contents are stored.

Return type str

Raises ValueError – When *varnames* is not a list type object.

Notes

Notice that varnames should be a list of strings.

put(*tmp_filename*)

Put variables from python into MATLAB workspace.

Parameters **tmp_filename** (*str*) – MAT file holding variables to inject into workspace.

run(*cmd*, *log_func=None*)

Run MATLAB command in subprocess.

Parameters

- **cmd** (*str*) – Command to send to MATLAB console.

- **log_func** (*callable, optional*) – Function to use to log output of MATLAB console.

class mr_utils.matlab.server.**MyTCPHandler**(*request*, *client_address*, *server*)
Create the server, binding to localhost on port.

what
The action we wish to perform.

Type {contract.RUN, contract.GET, contract.PUT}

cmd
The command to be run in the MATLAB console.

Type str

rfile
Stream from TCP socket that we are reading from.

Type stream

mr_utils.matlab.server.**start_server**()
Start the server so the client can connect.

Notes

This server must be running on the remote before client can be used to connect to it.

Examples

To run this server, simply run:

```
python3 mr_utils/matlab/start_server.py
```

2.8.3 contract

Define communication tokens for communication with MATLAB server.

2.9 Optimization

2.9.1 Gradient Descent

General implementation of gradient descent algorithm.

More of a learning exercise for myself.

mr_utils.optimization.gd.**gd**(*f*, *grad*, *x0*, *alpha=None*, *maxiter=1000000.0*, *tol=1e-08*)
Gradient descent algorithm.

Parameters

- **f** (*callable*) – Function to be optimized.
- **grad** (*callable*) – Function that computes the gradient of f.
- **x0** (*array_like*) – Initial point to start to start descent.

- **alpha** (*callable or float, optional*) – Either a fixed step size or a function that returns step size.
- **maxiter** (*int, optional*) – Do not exceed this number of iterations.
- **tol** (*float, optional*) – Run until change in norm of gradient is within this number.

Returns

- **cur_x** (*array_like*) – Estimate of optimal choice of x.
- *int* – Number of iterations.

2.9.2 gradient

Numerical derivative implementations.

mr_utils.optimization.gradient.**cd_gen_complex_step**(*f*, *x0*, *h=None*, *v=None*)

Compute generalized central difference complex step derivative of f.

Parameters

- **f** (*callable*) – Function to compute derivative of at x0.
- **x0** (*array_like*) – Point to compute derivative of f on.
- **h** (*float, optional*) – Real part of forward and backward derivatives.
- **v** (*float, optional*) – Imaginary part of forward and backwards derivatives.

Returns **g** – Gradient of f at x0

Return type array_like

Notes

If you choose h, v such that 3*h**2 =/= v**2, there will be an additional error term proportional to 3rd order derivative (not implemented). So it's in your best interest to choose h, v so this error is minimized. If h=None and v=None these values will be computed for you to satisfy this condition.

Implements Equation 5 from[2].

References

mr_utils.optimization.gradient.**complex_step_6th_order**(*f*, *x0*, *h=None*, *v=None*)

6th order accurate complex step difference method.

Parameters

- **f** (*callable*) – Function to compute derivative of at x0.
- **x0** (*array_like*) – Point to compute derivative of f on.
- **h** (*float, optional*) – Real part of forward and backward derivatives.
- **v** (*float, optional*) – Imaginary part of forward and backwards derivatives.

Returns **g** – Gradient of f at x0

Return type array_like

[2] Abreu, Rafael, et al. "On the accuracy of the Complex-Step-Finite-Difference method." Journal of Computational and Applied Mathematics 340 (2018): 390-403.

mr_utils.optimization.gradient.**fd_complex_step**(*f*, *x0*, *h=2.220446049250313e-16*)
Compute forward difference complex step of function f.

Parameters

- **f** (*callable*) – Function to compute derivative of.
- **x0** (*array_like*) – Point at which to compute derivate of f.
- **h** (*float, optional*) – Perturbation size.

Returns **g** – Gradient of f at x0

Return type array_like

mr_utils.optimization.gradient.**fd_gen_complex_step**(*f*, *x0*, *h=0*, *v=2.220446049250313e-16*)
Compute generalized forward difference complex step derivative of f.

Parameters

- **f** (*callable*) – Function to compute derivative of at x0.
- **x0** (*array_like*) – Point to compute derivative of f on.
- **h** (*float, optional*) – Real part of forward perturbation.
- **v** (*float, optional*) – Imaginary part of forward perturbation.

Returns **g** – Gradient of f at x0

Return type array_like

Notes

Implements Equation 4 from[1].

References

2.9.3 linesearch

Linesearch functions.

Once we have a direction to step, for example, the negative gradient direction in a gradient descent algorithm, then we need to know how big of a step to take. If we take too large or small a step, we may not find the minumum of the object function along the line we are stepping. A linesearch attempts to find the optimal step size in a given direction with minimal gradient and objective evaluations.

mr_utils.optimization.linesearch.**linesearch**(*obj*, *x0*, *a0*, *s*)
More sophisticated linesearch.

Parameters

- **obj** (*callable*) – Objective function.
- **x0** (*array_like*) – Current location.
- **a0** (*float*) – Current guess at stepsize.
- **s** (*array_like*) – Search direction.

[1] Abreu, Rafael, et al. "On the accuracy of the Complex-Step-Finite-Difference method." Journal of Computational and Applied Mathematics 340 (2018): 390-403.

Returns **a** – Stepsize for the current step.

Return type float

mr_utils.optimization.linesearch.**linesearch_quad**(*f*, *x*, *a*, *s*)
Simple quadratic linesearch.

Parameters

- **f** (*callable*) – Objective function.
- **x** (*array_like*) – Current location.
- **a** (*float*) – Guess for stepsize.
- **s** (*array_like*) – Search direction.

Returns **a** – Stepsize for the current step.

Return type float

2.10 Recon

2.10.1 Field Mapping

dual_echo_gre

Compute field map from dual echo GRE acquisitions.

mr_utils.recon.field_map.dual_echo_gre.**dual_echo_gre**(*m1*, *m2*, *TE1*, *TE2*)
Compute wrapped field map from two GRE images at different TEs.

Parameters

- **m1** (*array_like*) – GRE image taken with TE = TE1.
- **m2** (*array_like*) – GRE image taken with TE = TE2.
- **TE1** (*float*) – echo time corresponding to m1.
- **TE2** (*float*) – echo time corresponding to m2.

Returns **fm** – Field map in herz

Return type array_like

gs_field_map

Use the geometric solution to the elliptical signal model for field map.

mr_utils.recon.field_map.gs_field_map.**gs_field_map**(*I0*, *I1*, *I2*, *I3*, *TR*, *gs_recon_opts=None*)
Use the elliptical signal model to estimate the field map.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

- **TR** (*float*) – Repetition time of acquisitons in ms.
- **gs_recon_opts** (*dict, optional*) – Options to pass to gs_recon.

Returns **gsfm** – Wrapped field map in hertz.

Return type array_like

Notes

I0, I2 and I1, I3 must be phase-cycle pairs, meaning I0, I2 are separated by 180 degrees and I1, I3 are separated by 180 degrees. It does not matter what the actual phase-cycles are.

Implements field map estimation given in[1].

References

2.10.2 GRAPPA

grappa

Simple GRAPPA implementation for learning purposes.

Please use Gadgetron's implementation if you need to use GRAPPA for real.

mr_utils.recon.grappa.grappa.**grappa2d**(*coil_ims*, *sens*, *acs*, *Rx*, *Ry*, *kernel_size=(3, 3)*)

Simple 2D GRAPPA implementation.

Parameters

- **coil_ims** (*array_like*) – Coil images with sensity weightings corresponding to *sens*.
- **sens** (*array_like*) – Coil sensitivy maps.
- **acs** (*array_like*) – Autocalibration signal/region measurements.
- **Rx** (*int*) – Undersampling factor in x dimension.
- **Ry** (*int*) – Undersampling factor in y dimension.
- **kernel_size** (*tuple*) – Size of kernel, (x, y).

Returns **recon** – Reconstructed image from coil images and sensitivies.

Return type array_like

2.10.3 Partial Fourier

partial_fourier_pocs

Python port of Gadgetron's 2D partial_fourier_POCS.

Apologies for docstrings - not a lot in them because I don't fully understand what's happening yet.

mr_utils.recon.partial_fourier.partial_fourier_pocs.**apply_kspace_filter_ROE1**(*data*, *FRO*, *FE1*)

Apply kspace filter in readout direction 1.

[1] Taylor, Meredith, et al. "MRI Field Mapping using bSSFP Elliptical Signal model." Proceedings of the ISMRM Annual Conference (2017).

Parameters

- **data** (*array_like*) – kspace data
- **FRO** (*array_like*) –
- **FE1** (*array_like*) –

Returns **dataFiltered** – Filtered kspace data

Return type array_like

mr_utils.recon.partial_fourier.partial_fourier_pocs.**compute_2d_filter**(*fx*, *fy*)

Create a 2d filter from fx, fy.

Parameters

- **fx** (*array_like*) –
- **fy** (*array_like*) –

Returns **fxy**

Return type array_like

mr_utils.recon.partial_fourier.partial_fourier_pocs.**generate_symmetric_filter**(*length*, *filterType*, *sigma=1.5*, *width=15*)

Make a symmetric fileter.

Parameters

- **length** (*int*) – Length of the filter.
- **filterType** (*ISMRMRD.filter_type*) – The type of filter to make.
- **sigma** (*float*) –
- **width** (*int*) – Width of the filter.

Returns **filter** – The computed symmetric filter.

Return type array_like

Raises Exception – When filterType is unrecognized.

mr_utils.recon.partial_fourier.partial_fourier_pocs.**generate_symmetric_filter_ref**(*length*, *start*, *end*)

Generate symmetric filter.

Parameters

- **length** (*int*) – Length of filter.
- **start** (*int*) – Start of filter.
- **end** (*int*) – End of filter.

Returns **filter** – Symmetric filter.

Return type array_like

Raises Exception – When length, start, and end are not compatible.

mr_utils.recon.partial_fourier.partial_fourier_pocs.**partial_fourier_pocs**(*kspace*, *startRO*, *endRO*, *startE1*, *endE1*, *transit_band_RO=0*, *transit_band_E1=0*, *niter=10*, *thres=0.01*)

2D Partial Fourier POCS reconstruction.

Parameters

- **kspace** (*array_like*) – Input kspace (R0 E1 E2 ...).
- **startRO** (*int*) – Start of acquired kspace range in readout.
- **endRO** (*int*) – End of acquired kspace range in readout.
- **startE1** (*int*) – Start of acquired kspace range in E1.
- **endE1** (*int*) – End of acquired kspace range in E1.
- **transit_band_R0** (*int, optional*) – Transition band width in pixel for R0
- **transit_band_E1** (*int, optional*) – Transition band width in pixel for E1
- **niter** (*int, optional*) – Number of maximal iterations for POCS.
- **thres** (*float, optional*) – Iteration threshold.

Returns **res** – POCS reconstruction.

Return type array_like

Raises Exception – When transition bands are not 0 (partial_fourier_transition_band not implemented).

mr_utils.recon.partial_fourier.partial_fourier_pocs.**partial_fourier_reset_kspace**(*src*, *dst*, *startRO*, *endRO*, *startE1*, *endE1*)

Reset kspace for POCS reconstruction.

Parameters

- **src** (*array_like*) – Source kspace data.
- **dst** (*array_like*) – Destination for data.
- **startRO** (*int*) – Start of acquired kspace range in readout.
- **endRO** (*int*) – End of acquired kspace range in readout.
- **startE1** (*int*) – Start of acquired kspace range in E1.
- **endE1** (*int*) – End of acquired kspace range in E1.

Returns **dst** – Reset kspace data.

Return type array_like

2.10.4 SSFP

dixon

Collection of Dixon fat/water separation methods.

Implementations of methods described in Berstein (see function docstrings for references).

mr_utils.recon.ssfp.dixon.**dixon_2pt**(*IP*, *OP*)

Naive two-point Dixon method of fat/water separation.

Parameters

- **IP** (*array_like*) – In-phase image (corresponding to 0).
- **OP** (*array_like*) – Out-of-phase image (corresponding to pi).

Returns

- **W** (*array_like*) – water image
- **F** (*array_like*) – fat image

Notes

"[This implementation] ignores additional image weighting from T2* relaxation, diffusion, and flow and from other phase shifts that could arise from hardware group delays, eddy currents, and B1 receive-field nonuniformity. We have also ignored the water-fat chemical shift separation in both the slice and readout directions"[1].

Implements method described in[1]. Also equations [17.52] in[2].

References

mr_utils.recon.ssfp.dixon.**dixon_2pt_mag**(*IP*, *OP*)

Solution to two-point Dixon method using magnitude of images.

Parameters

- **IP** (*array_like*) – In-phase image (corresponding to 0).
- **OP** (*array_like*) – Out-of-phase image (corresponding to pi).

Returns

- **abs(W)** (*array_like*) – water image
- **abs(F)** (*array_like*) – fat image

Notes

Implements equations [17.53-54] in[4].

[1] Dixon, W. T. (1984). Simple proton spectroscopic imaging. Radiology,
[2] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.
[4] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

References

mr_utils.recon.ssfp.dixon.**dixon_3pt**(*IP*, *OP1*, *OP2*, *use_2pi=True*, *method='glover'*)

Three point Dixon method of fat/water separation.

Parameters

- **IP** (*array_like*) – In-phase image (corresponding to 0).
- **OP1** (*array_like*) – Out-of-phase image (corresponding to pi).
- **OP2** (*array_like*) – Out-of-phase image (corresponding to -pi or 2*pi).
- **use_2pi** (*bool, optional*) – Use 2*pi for OP2 instead of -pi.
- **method** (*{'vanilla', 'glover', 'chen'}, optional*) – Method to use to determine pc, see dixon_pc().

Returns

- **W** (*array_like*) – water image
- **F** (*array_like*) – fat image
- **B0** (*array_like*) – B0 inhomogeneity image.

Notes

"The phase difference between the two opposed-phase images is due to B0 inhomogeneity, and they are used to compute phi. The phi map is used to remove the B0 inhomogeneity phase shift from one of the opposed-phase images and thereby determine the dominant species for each pixel (i.e., whether W > F, or vice versa)."

Implements method described[6]. Also implements equations [17.71] in[7].

References

mr_utils.recon.ssfp.dixon.**dixon_3pt_dpe**(*I0*, *I1*, *I2*, *theta*)

Three point Dixon using direct phase encoding (DPE).

Parameters

- **I0** (*array_like*) – In-phase image (corresponding to phi_0 phase).
- **I1** (*array_like*) – Out-of-phase image (corresponding to phi_0 + phi).
- **I2** (*array_like*) – Out-of-phase image (corresponding to phi_0 + 2*phi).
- **theta** (*float*) – Phase term.

Returns

- **W** (*array_like*) – Water image
- **F** (*array_like*) – Fat image.

[6] Glover, G. H., & Schneider, E. (1991). Three-point Dixon technique for true water/fat decomposition with B0 inhomogeneity correction. Magnetic resonance in medicine, 18(2), 371-383.

[7] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

Notes

Note that theta_0 + theta should not be a multiple of pi!

Implements equations [17.83-84] from[9].

References

mr_utils.recon.ssfp.dixon.**dixon_3pt_eam**(*I0*, *I1*, *I2*, *method='glover'*)

Three point Dixon including echo amplitude modulation (EAM).

Parameters

- **I0** (*array_like*) – In-phase image (corresponding to phi_0 phase).
- **I1** (*array_like*) – Out-of-phase image (corresponding to phi_0 + phi).
- **I2** (*array_like*) – Out-of-phase image (corresponding to phi_0 + 2*phi).
- **method** (*{'vanilla', 'glover', 'chen'}, optional*) – Method to use to determine pc, see dixon_pc().

Returns

- **W** (*array_like*) – water image
- **F** (*array_like*) – fat image
- **A** (*array_like*) – The susceptibility dephasing map.

Notes

"...under our assumptions, ignoring amplitude effects simply results in a multiplicative error in both water and fat components. This error is usually not serious and can be ignored...there is a SNR penalty for the amplitude correction, and it is best avoided unless there is a specific need to compute A for the application of interest"[8].

Implements equations [17.78] from[8].

References

mr_utils.recon.ssfp.dixon.**dixon_extended_2pt**(*IP*, *OP*, *method='glover'*)

Extended two-point Dixon method for fat/water separation.

Parameters

- **IP** (*array_like*) – In-phase image (corresponding to 0).
- **OP** (*array_like*) – Out-of-phase image (corresponding to pi).
- **method** (*{'vanilla', 'glover', 'chen'}, optional*) – Method to use to determine pc, see dixon_pc().

Returns

- **abs(W)** (*array_like*) – water image
- **abs(F)** (*array_like*) – fat image

[9] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.
[8] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

Notes

Extended 2PD attempts to address the B0 homogeneity problem by using a generalized pc.

Implements equations [17.63] in[5].

References

mr_utils.recon.ssfp.dixon.**dixon_pc**(*IP*, *OP*, *method='vanilla'*)

Methods to determine pc, fat/water fraction within a voxel.

Parameters

- **IP** (*array_like*) – In-phase image (corresponding to 0).
- **OP** (*array_like*) – Out-of-phase image (corresponding to pi).
- **method** (*{'vanilla', 'glover', 'chen'}, optional*) – Method to determine pc. See notes.

Returns **pc** – Fat/water fraction.

Return type array_like

Raises NotImplementedError – When incorrect value for *method* used.

Notes

method: - 'vanilla': sign of W - F. - 'glover': maintain continuous image appearance by using cont. p value. - 'chen': alternative that performs 'glover' and then discretizes.

'glover' is implementation of eq [17.62], 'chen' is implementation of eq [17.64-65], 'vanilla' is eq [17.54] from[3].

References

gs_recon

Geometric solution to the elliptical signal model.

mr_utils.recon.ssfp.gs_recon.**complex_sum**(*I1*, *I2*, *I3*, *I4*)

Complex sum image combination method.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

Returns **CS** – Complex sum image.

Return type array_like

[5] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.
[3] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

mr_utils.recon.ssfp.gs_recon.**compute_Iw**(*I0*, *I1*, *Id*, *patch_size=None*, *mode='constant'*, *isophase=3.141592653589793*)

Computes weighted sum of image pair (I0,I1).

Parameters

- **I0** (*array_like*) – 1st of pair of diagonal images (relative phase cycle of 0).
- **I1** (*array_like*) – 2nd of pair of diagonal images (relative phase cycle of 180 deg).
- **Id** (*array_like*) – result of regularized direct solution.
- **patch_size** (*tuple, optional*) – size of patches in pixels (x, y). Defaults to (5, 5).
- **mode** (*{'contant', 'edge'}, optional*) – mode of numpy.pad. Probably choose 'constant' or 'edge'.
- **isophase** (*float*) – Only neighbours with isophase max phase difference contribute.

Returns **Iw** – The weighted sum of image pair (I0,I1), equation [14]

Return type array_like

Notes

Image pair (I0,I1) are phase cycled bSSFP images that are different by 180 degrees. Id is the image given by the direct method (Equation [13]) after regularization by the complex sum. This function solves for the weights by regional differential energy minimization. The 'regional' part means that the image is split into patches of size patch_size with edge boundary conditions (pads with the edge values given by mode option). The weighted sum of the image pair is returned.

The isophase does not appear in the paper, but appears in Hoff's MATLAB code. It appears that we only want to consider pixels in the patch that have similar tissue properties - in other words, have similar phase. The default isophase is pi as in Hoff's implementation.

This function implements Equations [14,18], or steps 4–5 from Fig. 2 in[1].

mr_utils.recon.ssfp.gs_recon.**get_max_magnitudes**(*I1*, *I2*, *I3*, *I4*)

Find max magnitudes for each pixel over all four input images.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

Returns **I_mag** – Maximum magnitude image at each pixel.

Return type array_like

mr_utils.recon.ssfp.gs_recon.**get_max_magnitudes_for_loop**(*I1*, *I2*, *I3*, *I4*)

Find max magnitudes for each pixel over all four input images.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).

[1] Xiang, Qing-San, and Michael N. Hoff. "Banding artifact removal for bSSFP imaging with an elliptical signal model." Magnetic resonance in medicine 71.3 (2014): 927-933.

- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

Returns **I_mag** – Maximum magnitude image at each pixel.

Return type array_like

Notes

This one loops over each pixel as verification for get_max_magnitudes().

mr_utils.recon.ssfp.gs_recon.**gs_recon**(*Is*, *pc_axis=0*, *isophase=3.141592653589793*, *second_pass=True*, *patch_size=None*)

Full 2D Geometric Solution following Xiang, Hoff's 2014 paper.

Parameters

- **Is** (*array_like*) – 4 phase-cycled images: [I0, I1, I2, I3]. (I0, I2) and (I1, I3) are phase-cycle pairs.
- **pc_axis** (*int, optional*) – Phase-cycle dimension, default is the first dimension.
- **isophase** (*float*) – Only neighbours with isophase max phase difference contribute.
- **second_pass** (*bool, optional*) – Compute the second pass solution, increasing SNR by sqrt(2).
- **patch_size** (*tuple, optional*) – Size of patches in pixels (x, y).

Returns

- **I** (*array_like*) – Second pass GS solution to elliptical signal model.
- **Id** (*array_like, optional*) – If second_pass=False, GS solution to the elliptical signal model (without linear weighted solution).

Notes

Is is an array of 4 2D images: I0, I1, I2, and I3. I0 and I2 make up the first phase-cycle pair, that is they are 180 degrees phase-cycled relative to each other. I1 and I3 are also phase-cycle pairs and must be different phase-cycles than either I0 or I2. Relative phase-cycles are assumed as follows:

- I0: 0 deg
- I1: 90 deg
- I2: 180 deg
- I3: 270 deg

Implements algorithm shown in Fig 2 of[1].

References

mr_utils.recon.ssfp.gs_recon.**gs_recon3d**(*I1*, *I2*, *I3*, *I4*, *slice_axis=-1*, *isophase=3.141592653589793*)

Full 3D Geometric Solution following Xiang, Hoff's 2014 paper.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).

- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).
- **slice_axis** (*int, optional*) – Slice dimension, default is the last dimension.
- **isophase** (*float*) – Only neighbours with isophase max phase difference contribute.

Returns **recon** – GS solution to elliptical signal model.

Return type array_like

Raises `AssertionError` – When phase-cycle images have different numbers of slices.

Notes

For more info, see mr_utils.recon.ssfp.gs_recon.

`mr_utils.recon.ssfp.gs_recon.`**gs_recon_for_loop**(*I1*, *I2*, *I3*, *I4*)
GS recon implemented using a straightfoward loop.

Parameters

- **I0** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I1** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I2** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I3** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

Returns **I** – GS solution to elliptical signal model.

Return type array_like

Notes

For verification, probably don't use this, it's slow.

`mr_utils.recon.ssfp.gs_recon.`**mask_isophase**(*numerator_patches*, *patch_size*, *isophase*)
Generate mask that chooses patch pixels that satisfy isophase.

Parameters

- **numerator_patches** (*array_like*) – Numerator patches from second pass solution.
- **patch_size** (*tuple*) – size of patches in pixels (x,y).
- **isophase** (*float*) – Only neighbours with isophase max phase difference contribute.

Returns **mask** – same size as numerator_patches, to be applied to numerator_patches and den_patches before summation.

Return type array_like

PLANET

PLANET: an ellipse fitting approach for simultaneous T1 and T2...

...mapping Using Phase-Cycled Balanced Steady-State Free Precession.

mr_utils.recon.ssfp.planet.**PLANET**(*I*, *alpha*, *TR*, *T1_guess*, *fit_ellipse=None*, *pcs=None*, *compute_df=False*, *disp=False*)

Simultaneous T1, T2 mapping using phase-cycled bSSFP.

Parameters

- **I** (*array_like*) – Complex voxels from phase-cycled bSSFP images.
- **alpha** (*float*) – Flip angle (in rad).
- **TR** (*float*) – Repetition time (in sec).
- **T1_guess** (*float*) – Estimate of expected T1 value (in sec).
- **fit_ellipse** (*callable, optional*) – Function used to fit data points to ellipse.
- **pcs** (*array_like, optional*) – Phase-cycles that generate phase-cycle images of I (required if computing df) (in rad).
- **compute_df** (*bool, optional*) – Whether or not estimate local off-resonance, df.
- **disp** (*bool, optional*) – Show debug plots.

Returns

- **Meff** (*array_like*) – Effective magnetization amplitude (arbitrary units).
- **T1** (*array_like*) – Estimate of T1 values (in sec).
- **T2** (*array_like*) – Estimate of T2 values (in sec).
- **df** (*array_like, optional*) – Estimate of off-resonance values (in Hz).

Raises

- AssertionError – If fit_ellipse returns something that is not an ellipse
- AssertionError – If the rotation fails and xc < 0 or yc =/= 0.
- AssertionError – If a, b, or Meff are outside of interval (0, 1).
- ValueError – If ellipse callapses to a line.
- ValueError – If the sign of b cannot be determined.

Notes

Requires at least 6 phase cycles to fit the ellipse. The ellipse fitting method they use (and which is implemented here) may not be the best method, but it is quick. Could add more options for fitting in the future.

fit_ellipse(x, y) should take two arguments and return a vector containing the coefficients of the implicit ellipse equation. If fit_ellipse=None then the mr_utils.utils.fit_ellipse_halir() function will be used.

pcs should be a list of phase-cycles in radians. If pcs=None, it will be determined as I.size equally spaced phasce-cycles on the interval [0, 2pi).

Implements algorithm described in[1].

[1] Shcherbakova, Yulia, et al. "PLANET: an ellipse fitting approach for simultaneous T1 and T2 mapping using phase-cycled balanced steady-state free precession." Magnetic resonance in medicine 79.2 (2018): 711-722.

References

taylor_method

Python port of Merry's bSSFP parameter mapping code.

This is an alternative to PLANET.

mr_utils.recon.ssfp.merry_param_mapping.taylor_method.**optim_wrapper**(*idx*, *Is*, *TR*, *dphis*, *offres_est*, *alpha*)

Wrapper for parallelization.

Parameters

- **idx** (*array_like*) – Indices of current pixels, must be provided as parallelization is non-sequential
- **Is** (*array_like*) – Array of phase-cycled images, (dphi, x, y).
- **TR** (*float*) – Repetition time (in sec).
- **dphis** (*array_like*) – Phase-cycles (in radians).
- **offres_est** (*array_like*) – Off-resonance map estimation (in Hz).
- **alpha** (*array_like*) – Flip angle map (in rad).

Returns

- **ii** (*int*) – Row index
- **jj** (*int*) – Column index
- **xopt** (*array_like*) – Optimized parameters: [T1, T2, offres, M0]

mr_utils.recon.ssfp.merry_param_mapping.taylor_method.**taylor_method**(*Is*, *dphis*, *alpha*, *TR*, *mask=None*, *chunksize=10*, *unwrap_fun=None*, *disp=False*)

Parameter mapping for multiple phase-cycled bSSFP.

Parameters

- **Is** (*array_like*) – Array of phase-cycled images, (dphi, x, y).
- **dphis** (*array_like*) – Phase-cycles (in radians).
- **alpha** (*array_like*) – Flip angle map (in rad).
- **TR** (*float*) – Repetition time (in sec).
- **mask** (*array_like, optional*) – Locations to compute map estimates.
- **chunksize** (*int, optional*) – Chunk size to use for parallelized loop.

- **unwrap_fun** (*callable*) – Function to do 2d phase unwrapping. If None, will use skimage.restoration.unwrap_phase().
- **disp** (*bool, optional*) – Show debugging plots.

Returns

- **t1map** (*array_like*) – T1 map estimation (in sec)
- **t2map** (*array_like*) – T2 map estimation (in sec)
- **offresmap** (*array_like*) – Off-resonance map estimation (in Hz)
- **m0map** (*array_like*) – Proton density map estimation

Raises `AssertionError` – If number of phase-cycles is not divisible by 4.

Notes

mask=None computes maps for all points. Note that *Is* must be given as a list.

Objective function wrapper used with MATLAB, unnecessary.

mr_utils.recon.ssfp.merry_param_mapping.optimize.**optimize**(*I*, *TR*, *phasecycles*, *offres*, *M0*, *alpha*, *T1*, *T2*)

Optimization driver to find T1, T2, offres, and M0 estimates.

Parameters

- **I** (*list*) – List of phase-cycled images.
- **TR** (*float*) – Repetition time (in sec).
- **phasecycles** (*array_like*) – Phase-cycles (in radians).
- **offres** (*array_like*) – Off-resonance map estimation (in Hz).
- **M0** (*float*) – Initial guess for M0.
- **alpha** (*float*) – Flip angle (in rad).
- **T1** (*float*) – Inital guess for T1 (in sec).
- **T2** (*float*) – Initial guess for T2 (in sec).

Returns

- *array_like* – Optimized values for [T1 (sec), T2 (sec), offres (Hz), M0].
- *float* – Final objective function value.

Calculates residuals for least_squares fit.

mr_utils.recon.ssfp.merry_param_mapping.elliptical_fit.**ellipticalfit**(*Ireal*, *TR*, *dphis*, *offres*, *M0*, *alpha*, *T1*, *T2*)

ELLIPTICALFIT

Parameters

- **Ireal** (*array_like*) – Hermtian transposed phase-cycle values for single pixel.

- **TR** (*float*) – Repetition time (in sec).
- **M0** – Estmated proton density from the band reduction algorithm.
- **phasecycles** – Phase-cycles (in rad).
- **offres** – Off-resonance estimation (in Hz).

Returns **J** – Real part of difference concatenated with imaginary part of difference

Return type array_like

Plot an ellipse based on MR parameters.

mr_utils.recon.ssfp.merry_param_mapping.plot_ellipse.**plotEllipse**(*T1*, *T2*, *TR*, *alpha*, *offres*, *M0*, *dphi*)

Compute (x, y) coordinates for ellipse described by MR parameters.

Parameters

- **T1** (*float*) – Longitudinal relaxation constant (in sec).
- **T2** (*float*) – Transverse relaxation constant (in sec).
- **TR** (*float*) – Repetition time (in sec).
- **alpha** (*float*) – Flip angle (in rad).
- **offres** (*float*) – Off-resonance (in Hz).
- **M0** (*float*) – Proton density.
- **dphi** (*float or bool*) – Phase-cycle value (in rad). dphi=True means use fixed linspace for dphi set, else, use the list of dphis provided.

Returns

- **x** (*array_like*) – x coordinate values of ellipse.
- **y** (*array_like*) – y coordinate values of ellipse.

2.10.5 TV Denoising

tv_denoising

Port of TVL1denoise - TV-L1 image denoising with the primal-dual algorithm.

Implementation of MATLAB script found at[1] which includes the following copyright notice:

> Copyright (c) 2016 Manolis Lourakis (lourakis **at** ics forth gr) Institute of Computer Science, Foundation for Research & Technology - Hellas Heraklion, Crete, Greece. http://www.ics.forth.gr/
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> The Software is provided "as is", without warranty of any kind.

[1] https://www.mathworks.com/matlabcentral/fileexchange/57604-tv-l1-image-denoising-algorithm

References

mr_utils.recon.tv_denoising.tv_denoising.**tv_l1_denoise**(*im*, *lam*, *disp=False*, *niter=100*)

TV-L1 image denoising with the primal-dual algorithm.

Parameters

- **im** (*array_like*) – image to be processed
- **lam** (*float*) – regularization parameter controlling the amount of denoising; smaller values imply more aggressive denoising which tends to produce more smoothed results
- **disp** (*bool, optional*) – print energy being minimized each iteration
- **niter** (*int, optional*) – number of iterations

Returns **newim** – l1 denoised image.

Return type array_like

Raises AssertionError – When dimension of im is not 2.

2.11 Simulations

2.11.1 bloch

Numerical bloch simulations using finite difference method.

mr_utils.sim.bloch.bloch.**gre**(*T1*, *T2*, *M0*, *Nt*, *h*, *alpha*, *beta*, *gamma*, *TR*, *TE*, *Bx=0*, *By=0*, *Bz=3*)

Finite difference Bloch simulation of spoiled GRE pulse sequence.

T1 [array_like] longitudinal relaxation constant.

T2 [array_like] transverse relaxation constant.

M0 [array_like] value at thermal equilibrium.

Nt [int] number of time points for finite difference solution.

h [float] step size for finite difference solutions.

alpha [float] RF pulse tip angle about x.

beta [float] RF pulse tip angle about y.

gamma [float] RF pulse tip angle about z.

TR [float] repetition time.

TE [float] echo time.

Bx [float, optional] x component of magnetic field.

By [float, optional] y component of magnetic field.

Bz [float, optional] z component of magnetic field.

Returns **spins** – Simulated spin vectors.

Return type array_like

Notes

T1, T2, M0 can be arrays (must be same size) to simulate phantoms.

mr_utils.sim.bloch.bloch.**rotation**(*alpha*, *beta*, *gamma*)
Create 3D rotation matrix from alpha,beta,gamma.

Parameters

- **alpha** (*float*) – Rotation angle about x in rad.
- **beta** (*float*) – Rotation angle about y in rad.
- **gamma** (*float*) – Rotation angle about z in rad.

Returns **rot** – Rotation matrix.

Return type array_like

mr_utils.sim.bloch.bloch.**sim**(*T1*, *T2*, *M0*, *Nt*, *h*, *alpha*, *beta*, *gamma*, *Bx=0*, *By=0*, *Bz=3*)
Finite difference solution to Bloch equations.

T1 [array_like] longitudinal relaxation constant.

T2 [array_like] transverse relaxation constant.

M0 [array_like] value at thermal equilibrium.

Nt [int] number of time points for finite difference solution.

h [float] step size for finite difference solutions.

alpha [float] RF pulse tip angle about x.

beta [float] RF pulse tip angle about y.

gamma [float] RF pulse tip angle about z.

Bx [float, optional] x component of magnetic field.

By [float, optional] y component of magnetic field.

Bz [float, optional] z component of magnetic field.

Returns **spins** – Simulated spin vectors.

Return type array_like

Notes

T1, T2, M0 can be arrays (must be same size) to simulate phantoms.

See[1] for matrix form of Bloch equations.

References

mr_utils.sim.bloch.bloch.**sim_loop**(*T1*, *T2*, *M0*, *Nt*, *h*, *alpha*, *beta*, *gamma*, *Bx=0*, *By=0*, *Bz=3*)
Loop implementation to verify matrix implementation.

T1 [array_like] longitudinal relaxation constant.

T2 [array_like] transverse relaxation constant.

[1] https://en.wikipedia.org/wiki/Bloch_equations#Matrix_form_of_Bloch_equations

M0 [array_like] value at thermal equilibrium.

Nt [int] number of time points for finite difference solution.

h [float] step size for finite difference solutions.

alpha [float] RF pulse tip angle about x.

beta [float] RF pulse tip angle about y.

gamma [float] RF pulse tip angle about z.

Bx [float, optional] x component of magnetic field.

By [float, optional] y component of magnetic field.

Bz [float, optional] z component of magnetic field.

Returns **spins** – Simulated spin vectors.

Return type array_like

2.11.2 gre

GRE simulations.

mr_utils.sim.gre.gre.**ernst**(*TR*, *T1*)

Computes the Ernst angle.

Parameters

- **TR** (*float*) – repetition time.
- **T1** (*array_like*) – longitudinal exponential decay time constant.

Returns **alpha** – Ernst angle in rad.

Return type array_like

Notes

Implements equation [14.9] from[1].

References

mr_utils.sim.gre.gre.**fzss**(*T1*, *TR*, *alpha=None*)

Dimensionless measure of steady-state longitudinal magnetization.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **TR** (*float*) – repetition time.
- **alpha** (*array_like, optional*) – flip angle (in rad).

Returns Dimensionless measure of steady-state longitudinal magnetization.

Return type array_like

[1] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

Notes

alpha=None will use the Ernst angle.

Implements equation [14.7] from[2].

References

mr_utils.sim.gre.gre.**gre_sim**(*T1*, *T2*, *TR=0.012*, *TE=0.006*, *alpha=1.0471975511965976*, *field_map=None*, *phi=0*, *dphi=0*, *M0=1*, *tol=1e-05*, *maxiter=None*, *spoil=True*)

Simulate GRE pulse sequence.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **T2** (*array_like*) – Transverse exponential decay time constant.
- **TR** (*float, optional*) – repetition time.
- **TE** (*float, optional*) – echo time.
- **alpha** (*float, optional*) – flip angle.
- **field_map** (*array_like, optional*) – offresonance field map (in hertz).
- **phi** (*float, optional*) – Reference starting phase.
- **dphi** (*float, optional*) – phase cycling of RF pulses.
- **M0** (*array_like, optional*) – proton density.
- **tol** (*float, optional*) – Maximum difference between voxel intensity iter to iter until stop.
- **maxiter** (*int, optional*) – number of excitations till steady state.

Returns Complex transverse magnetization (Mx + 1j*My)

Return type array_like

Notes

maxiter=None will run until difference between all voxel intensities iteration to iteration is within given tolerance, tol (default=1e-5).

mr_utils.sim.gre.gre.**gre_sim_loop**(*T1*, *T2*, *TR=0.012*, *TE=0.006*, *alpha=1.0471975511965976*, *field_map=None*, *dphi=0*, *M0=1*, *maxiter=200*)

Simulate GRE pulse sequence.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **T2** (*array_like*) – Transverse exponential decay time constant.
- **TR** (*float, optional*) – repetition time.
- **TE** (*float, optional*) – echo time.
- **alpha** (*float, optional*) – flip angle.

[2] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

- **field_map** (*array_like, optional*) – offresonance field map (in hertz).
- **dphi** (*float, optional*) – phase-cycling of RF pulses.
- **M0** (*array_like, optional*) – proton density.
- **maxiter** (*int, optional*) – number of excitations till steady state.

Returns **Mxy** – Complex transverse magnetization.

Return type array_like

mr_utils.sim.gre.gre.**spoiled_gre**(*T1*, *T2star*, *TR*, *TE*, *alpha=None*, *M0=1*)

Spoiled, steady state GRE contrast simulation.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **T2star** (*array_like*) – Effective transverse exponential decay time constant.
- **TR** (*float*) – repetition time.
- **TE** (*float*) – echo time.
- **alpha** (*array_like, optional*) – flip angle.
- **M0** (*array_like, optional*) – proton density.

Returns **S** – Simulated magnitude spoiled GRE image.

Return type array_like

Notes

alpha=None will use the Ernst angle.

Assuming a longitudinal steady-state and perfect spoiling. Note that dependence is on T2* rather than T2 because SE/STE formation is suppressed by spoiling and the signal is generated by gradient refocusing of an FID.

Implements equation [14.8] from[4].

References

mr_utils.sim.gre.gre.**spoiled_gre_k**(*T1*, *T2star*, *TR*, *TE*, *alpha=None*, *M0=1*, *k=1*)

Spoiled GRE contrast simulation for k excitation pulses.

See spoiled_gre().

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **T2star** (*array_like*) – Effective transverse exponential decay time constant.
- **TR** (*float*) – repetition time.
- **TE** (*float*) – echo time.
- **alpha** (*array_like, optional*) – flip angle.
- **M0** (*array_like, optional*) – proton density.

[4] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

- **k** (*int, optional*) – Number of excitation pulses the magnetization experiences.

Returns **Sk** – Simulated magnitude GRE image.

Return type array_like

Raises `AssertionError` – If k is less than 1.

Notes

alpha=None will use the Ernst angle.

Implements equations [14.10-11] from[3].

References

2.11.3 ir

Inversion recovery pulse sequence simulation.

`mr_utils.sim.ir.ir.`**`ir90`**(*T1*, *TR*, *TI*, *M0=1*)

Inversion recovery simulation with 90 deg flig angle.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant.
- **TR** (*float*) – repetition time.
- **TI** (*float*) – inversion time.
- **M0** (*array_like, optional*) – proton density.

Returns **S** – Simulated magnitude image.

Return type array_like

2.11.4 Noise Simulation

rayleigh

Noise simulations.

`mr_utils.sim.noise.rayleigh.`**`rayleigh`**(*M*, *sigma*)

Generates Rayleigh distribution of pixel intensity M.

Generates the noise distribution of magnitude MR image areas where only noise is present. This distribution governs the noise in image regions with no NMR signal.

Parameters

- **M** (*array_like*) – measured image pixel intensity
- **sigma** (*float*) – standard deviation of the Gaussian noise in the real and the imaginary images (which we assume to be equal)

Returns **pM** – Computed probability distribution of M

Return type array_like

[3] Notes from Bernstein, M. A., King, K. F., & Zhou, X. J. (2004). Handbook of MRI pulse sequences. Elsevier.

Notes

Computes Equation [2] from[1].

References

mr_utils.sim.noise.rayleigh.**rayleigh_mean**(*sigma*)
Mean of the Rayleigh distribution with standard deviation sigma.

Parameters **sigma** (*float*) – Standard deviation of Rayleigh distribution.

Returns **M_bar** – Mean of Rayleigh distribution.

Return type float

Notes

Computes Equation [3] from[2].

References

mr_utils.sim.noise.rayleigh.**rayleigh_variance**(*sigma*)
Variance of the Rayleigh distribution with standard deviation sigma.

Parameters **sigma** (*float*) – Standard deviation of Rayleigh distribution.

Returns Variance of Rayleigh distribution.

Return type float

Notes

Computes Equation [4] from[3].

References

rician

Noise simulations.

mr_utils.sim.noise.rician.**rician**(*M*, *A*, *sigma*)
Generates rician distribution of pixel intensity M.

Generates the noise distribution of a magnitude MR image.

Parameters

- **M** (*array_like*) – measured image pixel intensity
- **A** (*array_like*) – image pixel intensity in the absence of noise
- **sigma** (*float*) – standard deviation of the Gaussian noise in the real and the imaginary images (which we assume to be equal)

[1] Gudbjartsson, Hákon, and Samuel Patz. "The Rician distribution of noisy MRI data." Magnetic resonance in medicine 34.6 (1995): 910-914.
[2] Gudbjartsson, Hákon, and Samuel Patz. "The Rician distribution of noisy MRI data." Magnetic resonance in medicine 34.6 (1995): 910-914.
[3] Gudbjartsson, Hákon, and Samuel Patz. "The Rician distribution of noisy MRI data." Magnetic resonance in medicine 34.6 (1995): 910-914.

Returns **pM** – computed probability distribution of M

Return type array_like

Notes

Computes Equation [1] from[1].

References

2.11.5 se

Spin echo simulation.

mr_utils.sim.se.se.**se90**(*T1*, *T2*, *TR*, *TE*, *M0=1*)

Spin echo simulation assuming 90 deg flip angle

Parameters

- **T1** (*array_like*) – longitudinal relaxation time.
- **T2** (*array_like*) – transverse relaxation time.
- **TR** (*float*) – repetition time.
- **TE** (*float*) – echo time
- **M0** (*array_like, optional*) – Proton density.

Returns **S** – Simulated magnitude spin echo image.

Return type array_like

2.11.6 SSFP Simulations

ssfp

SSFP constrast simulation functions.

mr_utils.sim.ssfp.ssfp.**elliptical_params**(*T1*, *T2*, *TR*, *alpha*, *M0=1*)

Return ellipse parameters M, a, b.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant (in sec).
- **T2** (*array_like*) – transverse exponential decay time constant (in sec).
- **TR** (*float*) – repetition time (in sec).
- **alpha** (*float*) – flip angle (in rad).
- **M0** (*array_like, optional*) – Proton density.

Returns

- **M** (*array_like*) – Cross point.
- **a** (*array_like*) – Theta-independent ellipse parameter.

[1] Gudbjartsson, Hákon, and Samuel Patz. "The Rician distribution of noisy MRI data." Magnetic resonance in medicine 34.6 (1995): 910-914.

- **b** (*array_like*) – Theta-independent ellipse parameter.

Notes

Outputs are the parameters of ellipse an ellipse, (M, a, b). These parameters do not depend on theta.

Implementation of equations [3-5] in[1].

mr_utils.sim.ssfp.ssfp.**get_bssfp_phase**(*T2*, *TR*, *field_map*, *delta_cs=0*, *phi_rf=0*, *phi_edd=0*, *phi_drift=0*)

Additional bSSFP phase factors.

Parameters

- **T2** (*array_like*) – Longitudinal relaxation constant (in sec).
- **TR** (*float*) – Repetition time (in sec).
- **field_map** (*array_like*) – off-resonance map (Hz).
- **delta_cs** (*float, optional*) – chemical shift of species w.r.t. the water peak (Hz).
- **phi_rf** (*float, optional*) – RF phase offset, related to the combin. of Tx/Rx phases (rad).
- **phi_edd** (*float, optional*) – phase errors due to eddy current effects (rad).
- **phi_drift** (*float, optional*) – phase errors due to B0 drift (rad).

Returns **phase** – Additional phase term to simulate readout at time TE = TR/2. Assumes balanced (TE = TR/2).

Return type array_like

Notes

This is exp(-i phi) from end of p. 930 in[1].

We use a positive exponent, exp(i phi), as in Hoff and Taylor MATLAB implementations. This phase factor is also positive in equaiton [5] of[3].

In Hoff's paper the equation is not explicitly given for phi, so we implement equation [5] that gives more detailed terms, found in[2].

References

mr_utils.sim.ssfp.ssfp.**get_cart_elliptical_params**(*M*, *a*, *b*)

Get parameters needed for cartesian representation of ellipse.

Parameters

- **M** (*array_like*) – Cross point.
- **a** (*array_like*) – Theta-independent ellipse parameter.

[1] Xiang, Qing-San, and Michael N. Hoff. "Banding artifact removal for bSSFP imaging with an elliptical signal model." Magnetic resonance in medicine 71.3 (2014): 927-933.

[3] Scheffler, Klaus, and Jürgen Hennig. "Is TrueFISP a gradient-echo or a spin-echo sequence?." Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 49.2 (2003): 395-397.

[2] Shcherbakova, Yulia, et al. "PLANET: An ellipse fitting approach for simultaneous T1 and T2 mapping using phase-cycled balanced steady-state free precession." Magnetic resonance in medicine 79.2 (2018): 711-722.

- **b** (*array_like*) – Theta-independent ellipse parameter.

Returns

- **xc** (*array_like*) – x coordinates of geometric center of ellipse.
- **yc** (*array_like*) – y coordinates of geometric center of ellipse.
- **A** (*array_like*) – Semi-major axis.
- **B** (*array_like*) – Semi-minor axis.

mr_utils.sim.ssfp.ssfp.**get_center_of_mass**(*M*, *a*, *b*)
Give center of mass a function of ellipse parameters.

Parameters

- **M** (*array_like*) – Cross point.
- **a** (*array_like*) – Theta-independent ellipse parameter.
- **b** (*array_like*) – Theta-independent ellipse parameter.

Returns **cm** – Center of mass.

Return type array_like

mr_utils.sim.ssfp.ssfp.**get_center_of_mass_nmr**(*T1*, *T2*, *TR*, *alpha*, *M0=1*)
Give center of mass as a function of NMR parameters.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant (in sec).
- **T2** (*array_like*) – transverse exponential decay time constant (in sec).
- **TR** (*float*) – Repetition time (in sec).
- **alpha** (*float*) – Flip angle (in rad).
- **M0** (*array_like, optional*) – Proton density.

Returns **cm** – Center of mass.

Return type array_like

mr_utils.sim.ssfp.ssfp.**get_complex_cross_point**(*Is*)
Find intersection of two lines connecting diagonal pairs.

Parameters **Is** (*array_like*) – 4 phase-cycled images: [I0, I1, I2, I3].

Returns **M** – Complex cross point.

Return type array_like

Notes

We assume that Is has the phase-cycle dimenension along the first axis.

(xi, yi) are the real and imaginary parts of complex valued pixels in four bSSFP images denoted Ii and acquired with phase cycling dtheta = (i-1)*pi/2 with 0 < i < 4.

This is Equation [13] from[1].

mr_utils.sim.ssfp.ssfp.**get_cross_point**(*I1*, *I2*, *I3*, *I4*)
Find intersection of two lines connecting diagonal pairs.

Parameters

- **I1** (*array_like*) – First of the first phase-cycle pair (0 degrees).
- **I2** (*array_like*) – First of the second phase-cycle pair (90 degrees).
- **I3** (*array_like*) – Second of the first phase-cycle pair (180 degrees).
- **I4** (*array_like*) – Second of the second phase-cycle pair (270 degrees).

Returns

- **x0** (*array_like*) – x coordinate of cross point.
- **y0** (*array_like*) – y coordinate of cross point.

Notes

(xi,yi) are the real and imaginary parts of complex valued pixels in four bSSFP images denoted Ii and acquired with phase cycling dtheta = (i-1)*pi/2 with 0 < i < 4.

This are Equations [11-12] from[1]. There is a typo in the paper for equation [12] fixed in this implementation. The first term of the numerator should have (y2 - y4) instead of (x2 - y4) as written.

mr_utils.sim.ssfp.ssfp.**get_geo_center**(*M*, *a*, *b*)

Get geometric center of ellipse.

Parameters

- **M** (*array_like*) – Cross point.
- **a** (*array_like*) – Theta-independent ellipse parameter.
- **b** (*array_like*) – Theta-independent ellipse parameter.

Returns

- **xc** (*array_like*) – x coordinates of geometric center of ellipse.
- **yc** (*array_like*) – y coordinates of geometric center of ellipse.

mr_utils.sim.ssfp.ssfp.**get_theta**(*TR*, *field_map*, *phase_cyc=0*, *delta_cs=0*)

Get theta, spin phase per repetition time, given off-resonance.

Parameters

- **TR** (*float*) – repetition time (in sec).
- **field_map** (*array_like*) – Off-resonance map (in Hz).
- **phase_cyc** (*array_like, optional*) – Phase-cycling (in rad).
- **delta_cs** (*float, optional, optional*) – Chemical shift of species w.r.t. the water peak (Hz).

Returns **theta** – Spin phase per repetition time, given off-resonance.

Return type array_like

Notes

Equation for theta=2*pi*df*TR is in Appendix A of[6]. The additional chemical shift term can be found, e.g., in[2].

[6] Hargreaves, Brian A., et al. "Characterization and reduction of the transient response in steady-state MR imaging." Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 46.1 (2001): 149-158.

References

mr_utils.sim.ssfp.ssfp.**make_cart_ellipse**(*xc*, *yc*, *A*, *B*, *num_t=100*)
Make a cartesian ellipse, return x,y coordinates for plotting.

Parameters

- **xc** (*array_like*) – x coordinates of geometric center of ellipse.
- **yc** (*array_like*) – y coordinates of geometric center of ellipse.
- **A** (*array_like*) – Semi-major axis.
- **B** (*array_like*) – Semi-minor axis.

Returns

- **x** (*array_like*) – Cartesian x coordinates.
- **y** (*array_like*) – Cartesian y coordinates.

mr_utils.sim.ssfp.ssfp.**spectrum**(*T1*, *T2*, *TR*, *alpha*)
Generate an entire period of the bSSFP signal profile.

Parameters

- **T1** (*array_like*) – longitudinal exponential decay time constant (in sec).
- **T2** (*array_like*) – transverse exponential decay time constant (in sec).
- **TR** (*float*) – Repetition time (in sec).
- **alpha** (*float*) – Flip angle (in rad).

Returns **sig** – Full, complex SSFP spectrum.

Return type array_like

mr_utils.sim.ssfp.ssfp.**ssfp**(*T1*, *T2*, *TR*, *alpha*, *field_map*, *phase_cyc=0*, *M0=1*, *delta_cs=0*, *phi_rf=0*, *phi_edd=0*, *phi_drift=0*)
SSFP transverse signal at time TE after excitation.

Parameters

- **T1** (*float or array_like*) – longitudinal exponential decay time constant (in seconds).
- **T2** (*float or array_like*) – transverse exponential decay time constant (in seconds).
- **TR** (*float*) – repetition time (in seconds).
- **alpha** (*float or array_like*) – flip angle (in rad).
- **field_map** (*float or array_like*) – B0 field map (in Hz).
- **phase_cyc** (*float or array_like, optional*) – Linear phase-cycle increment (in rad).
- **M0** (*float or array_like, optional*) – proton density.
- **delta_cs** (*float, optional*) – chemical shift of species w.r.t. the water peak (in Hz).
- **phi_rf** (*float, optional*) – RF phase offset, related to the combin. of Tx/Rx phases (in rad).
- **phi_edd** (*float, optional*) – phase errors due to eddy current effects (in rad).
- **phi_drift** (*float, optional*) – phase errors due to B0 drift (in rad).

Returns **Mxy** – Transverse complex magnetization.

Return type numpy.array

Notes

T1, *T2*, *alpha*, *field_map*, and *M0* can all be either a scalar or an MxN array. *phase_cyc* can be a scalar or length L vector.

Implementation of equations [1–2] in[1]. These equations are based on the Ernst-Anderson derivation[4] where off-resonance is assumed to be subtracted as opposed to added (as in the Freeman-Hill derivation[5]). Hoff actually gets Mx and My flipped in the paper, so we fix that here. We also assume that the field map will be provided given the Freeman-Hill convention.

We will additionally assume that linear phase increments (phase_cyc) will be given in the form:

$$\theta = 2\pi(\delta_{cs} + \Delta f_0)\text{TR} + \Delta\theta.$$

Notice that this is opposite of the convention used in PLANET, where phase_cyc is subtracted (see equation [12] in[2]).

Also see equations [2.7] and [2.10a–b] from[4] and equations [3] and [6–12] from[5].

References

mr_utils.sim.ssfp.ssfp.**ssfp_from_ellipse**(*M*, *a*, *b*, *TR*, *field_map*, *phase_cyc=0*)
Simulate banding given elliptical signal params and field map.

Parameters

- **M** (*array_like*) – Cross point.
- **a** (*array_like*) – Theta-independent ellipse parameter.
- **b** (*array_like*) – Theta-independent ellipse parameter.
- **TR** (*float*) – Repetition time (in sec).
- **field_map** (*array_like*) – Off-resonance map (in Hz).
- **phase_cyc** (*float*) – Phase-cycling increment (in rad).

Returns **I** – SSFP simulation result.

Return type array_like

mr_utils.sim.ssfp.ssfp.**ssfp_old**(*T1*, *T2*, *TR*, *alpha*, *field_map*, *phase_cyc=0*, *M0=1*)
Legacy SSFP sim code. Try using current SSFP function.

ssfp_dictionary

Dictionary lookup of NMR parameters given bSSFP signal.

mr_utils.sim.ssfp.ssfp_dictionary.**find_atom**(*sig*, *D*, *keys*)
Find params of dictionary atom closest to observed signal profile.

Parameters

[4] Ernst, Richard R., and Weston A. Anderson. "Application of Fourier transform spectroscopy to magnetic resonance." Review of Scientific Instruments 37.1 (1966): 93-102.

[5] Freeman R, Hill H. Phase and intensity anomalies in fourier transform NMR. J Magn Reson 1971;4:366–383.

- **sig** (*array_like*) – Signal that should match an atom of the dicionary.
- **D** (*array_like*) – Dictionary of signals with keys being the MR parameters.
- **keys** (*array_like*) – Keys of dictionary D, all (T1, T2, alpha) combinations.

Returns **param_est** – T1, T2, alpha estimation based on closest dictionary atom.

Return type tuple

mr_utils.sim.ssfp.ssfp_dictionary.**get_keys**(*T1s*, *T2s*, *alphas*)

Generate matrix of params [T1, T2, alpha] to generate a dictionary.

Parameters

- **T1s** (*array_like*) – longitudinal relaxation values.
- **T2s** (*array_like*) – transverse relaxation values.
- **alphas** (*array_like*) – Flip angle values (in rad).

Returns **keys** – Valid tuples of (T1 ,T2, alpha) to simulate and lookup.

Return type array_like

Notes

T1, T2 are chosen to be feasible, i.e., T1 >= T2.

mr_utils.sim.ssfp.ssfp_dictionary.**ssfp_dictionary**(*T1s*, *T2s*, *TR*, *alphas*, *df*)

Generate a dicionary of bSSFP profiles given parameters.

Parameters

- **T1s** (*array_like*) – (1D) all T1 decay constant values to simulate.
- **T2s** (*array_like*) – (1D) all T2 decay constant values to simulate.
- **TR** (*float*) – repetition time for bSSFP simulation.
- **alphas** (*array_like*) – (1D) all flip angle values to simulate.
- **df** (*array_like*) – (1D) off-resonance frequencies over which to simulate.

Returns

- **D** (*array_like*) – Dictionary of simulated values
- **keys** (*array_like*) – Keys of dictionary D, all (T1, T2, alpha) combinations.

Notes

T1s,T2s,alphas should all be 1D arrays. All feasible combinations will be simulated (i.e., where T1 >= T2). The dictionary and keys are returned. Each dictionary column is the simulation over frequencies df. The keys are a list of tuples: (T1,T2,alpha).

mr_utils.sim.ssfp.ssfp_dictionary.**ssfp_dictionary_for_loop**(*T1s*, *T2s*, *TR*, *alphas*, *df*)

Verification for ssfp_dictionary generation.

Parameters

- **T1s** (*array_like*) – (1D) all T1 decay constant values to simulate.
- **T2s** (*array_like*) – (1D) all T2 decay constant values to simulate.

- **TR** (*float*) – repetition time for bSSFP simulation.
- **alphas** (*array_like*) – (1D) all flip angle values to simulate.
- **df** (*array_like*) – (1D) off-resonance frequencies over which to simulate.

Returns

- **D** (*array_like*) – Dictionary of simulated values
- **keys** (*array_like*) – Keys of dictionary D, all (T1, T2, alpha) combinations.

quantitative_field_mapping

Quantitative field mapping for bSSFP.

Collect quantitative MR maps (T1, T2, flip angle), then, assuming that these won't change during the duration of the scan, we can use these to take a single bSSFP scan each time point and solve for the off-resonance. Thus we get a field map at time point.

mr_utils.sim.ssfp.quantitative_field_mapping.**get_df_responses**(*T1*, *T2*, *PD*, *TR*, *alpha*, *phase_cyc*, *dfs*)

Simulate bSSFP response across all possible off-resonances.

Parameters

- **T1** (*float*) – scalar T1 longitudinal recovery value in seconds.
- **T2** (*float*) – scalar T2 transverse decay value in seconds.
- **PD** (*float*) – scalar proton density value scaled the same as acquisiton.
- **TR** (*float*) – Repetition time in seconds.
- **alpha** (*float*) – Flip angle in radians.
- **phase_cyc** (*float*) – RF phase cycling in radians.
- **dfs** (*float*) – Off-resonance values to simulate over.

Returns **resp** – Frequency response of SSFP signal across entire spectrum.

Return type array_like

mr_utils.sim.ssfp.quantitative_field_mapping.**quantitative_fm**(*Mxys*, *dfs*, *T1s*, *T2s*, *PDs*, *TR*, *alpha*, *phase_cyc*, *mask=None*)

Find field map given quantitative maps.

Parameters

- **Mxys** (*array_like*) – Complex transverse signal we measure.
- **dfs** (*array_like*) – Off-resonance values to simulate over.
- **T1s** (*array_like*) – scalar T1 longitudinal recovery value in seconds.
- **T2s** (*array_like*) – scalar T2 transverse decay value in seconds.
- **PDs** (*array_like*) – scalar proton density value scaled the same as acquisiton.
- **TR** (*float*) – Repetition time in seconds.
- **alpha** (*float*) – Flip angle in radians.

- **phase_cyc** (*float*) – RF phase cycling in radians.
- **mask** (*array_like*) – Boolean mask to tell which pixels we should compute df for.

Returns **fm** – Field map.

Return type array_like

mr_utils.sim.ssfp.quantitative_field_mapping.**quantitative_fm_scalar**(*Mxy*, *dfs*, *T1*, *T2*, *PD*, *TR*, *alpha*, *phase_cyc*)

For scalar T1, T2, PD.

Parameters

- **Mxy** (*float*) – Complex transverse signal we measure.
- **dfs** (*array_like*) – Off-resonance values to simulate over.
- **T1** (*float*) – scalar T1 longitudinal recovery value in seconds.
- **T2** (*float*) – scalar T2 transverse decay value in seconds.
- **PD** (*float*) – scalar proton density value scaled the same as acquisiton.
- **TR** (*float*) – Repetition time in seconds.
- **alpha** (*float*) – Flip angle in radians.
- **phase_cyc** (*float*) – RF phase cycling in radians.

Returns Off-resonace value that most closely matches Mxy prior.

Return type float

2.11.7 Trajectory Simulations

cartesian

Create sampling patterns for Cartesian k-space trajectories.

mr_utils.sim.traj.cartesian.**cartesian_gaussian**(*shape*, *undersample=(0.5, 0.5)*, *reflines=20*)

Undersample in Gaussian pattern.

Parameters

- **shape** (*tuple*) – Shape of the image to be sampled.
- **undersample** (*tuple, optional*) – Undersampling factor in x and y (0 < ux, uy <= 1).
- **reflines** (*int, optional*) – Number of lines in the center to collect regardless.

Returns **mask** – Boolean mask of sample locations on Cartesian grid.

Return type array_like

Raises AssertionError – If undersample factors are outside of interval (0, 1].

mr_utils.sim.traj.cartesian.**cartesian_pe**(*shape*, *undersample=0.5*, *reflines=20*)

Randomly collect Cartesian phase encodes (lines).

Parameters

- **shape** (*tuple*) – Shape of the image to be sampled.
- **undersample** (*float, optional*) – Undersampling factor (0 < undersample <= 1).
- **reflines** (*int, optional*) – Number of lines in the center to collect regardless.

Returns **mask** – Boolean mask of sample locations on Cartesian grid.

Return type array_like

Raises `AssertionError` – If undersample factor is outside of interval (0, 1].

radial

Generate radial sampling masks.

`mr_utils.sim.traj.radial.`**radial**(*shape*, *num_spokes*, *theta=None*, *offset=0*, *theta0=0*, *skinny=True*, *extend=False*)

Create 2d binary radial sampling pattern.

Parameters

- **shape** (*tuple*) – x,y dimensions of sampling pattern.
- **num_spokes** (*int*) – Number of spokes to simulate.
- **theta** (*float, optional*) – Angle between spokes (rad).
- **offset** (*int, optional*) – Number of angles to skip.
- **theta0** (*float, optional*) – Starting angle (rad).
- **skinny** (*bool, optional*) – Garuantee 1px spoke width.
- **extend** (*bool, optional*) – Extend spokes to the edge of array.

Returns **idx** – Boolean mask where samples in k-space should be taken.

Return type array_like

Notes

If theta=None, use golden angle. If skinny=True, edges of spokes with large slope may be curved. If extend=False, spokes confined in a circle.

`mr_utils.sim.traj.radial.`**radial_golden_ratio_meshgrid**(*X*, *Y*, *num_spokes*)

Create 2d binary golden angle radial sampling pattern.

Parameters

- **X** (*array_like*) – X component of meshgrid.
- **Y** (*array_like*) – T component of meshgrid.
- **num_spokes** (*int*) – Number of spokes to simulate.

Returns **samp** – Mask of where samples in k-space are taken.

Return type array_like

Notes

Issues: For steep slopes, the spokes don't make it all the way to the edge of the image and they curve (from the skeletonize)...

2.12 Test Data

2.12.1 test_data

Provide an interface to load test data for unit tests.

mr_utils.test_data.test_data.**load_test_data**(*path*, *files*, *do_return=True*)
Load test data, download if necessary.

Parameters

- **path** (*str*) – Location of directory where the test files live.
- **files** (*list*) – Specific files to return.
- **do_return** (*bool*) – Whether or not to return loaded files as a list.

Returns

- **returnVals** (*list*) – List of files loaded using np.load.
- *None, optional* – Files are downloaded to disk, but not loaded.

Notes

files should be a list. If no extension is given, .npy will be assumed. do_return=True assumes .npy file will be loaded (uses numpy.load).

2.12.2 Phantom

binary_smiley

Simple numerical phantom shaped like a smiley face. Value either 1 or 0.

mr_utils.test_data.phantom.binary_smiley.**binary_smiley**(*N*, *radius=0.75*)
Binary smiley face numerical phantom.

Parameters

- **N** (*int*) – Height and width in pixels.
- **radius** (*float, optional*) – Radius of circle used for head.

Returns **smiley** – Binary image of a smiley face.

Return type array_like

cylinder_2d

Simple cylindrical phantoms generated with different contrasts.

mr_utils.test_data.phantom.cylinder_2d.**bssfp_2d_cylinder**(*TR=0.006*, *alpha=1.0471975511965976*, *dims=(64, 64)*, *FOV=((-1, 1), (-1, 1))*, *radius=0.5*, *field_map=None*, *phase_cyc=0*, *kspace=False*)

Simulates axial bSSFP scan of cylindrical phantom.

Parameters

- **TR** (*float, optional*) – Repetition time.
- **alpha** (*float, optional*) – Flip angle (in rad).
- **dims** (*tuple of ints, optional*) – Matrix size, (dim_x,dim_y)
- **FOV** (*tuple of tuples, optional*) – Field of view in arbitrary units: ((x_min,x_max), (y_min,y_max))
- **radius** (*float, optional*) – Radius of cylinder in arbitrary units.
- **field_map** (*array_like, optional*) – (dim_x,dim_y) field map. If None, linear gradient in x used.
- **phase_cyc** (*float, optional*) – Phase cycling used in simulated bSSFP acquisition (in rad).
- **kspace** (*bool, optional*) – Whether or not to return data in kspace or imspace.

Returns **im** – Complex simulated image with bSSFP contrast.

Return type array_like

mr_utils.test_data.phantom.cylinder_2d.**cylinder_2d**(*dims=(64, 64)*, *FOV=((-1, 1), (-1, 1))*, *radius=0.5*, *params=None*)

Base 2d cylinder maps to feed to contrast simulations.

Parameters

- **dims** (*tuple of ints, optional*) – Size of 2d cylinder.
- **FOV** (*tuple of tuples, optional*) – Field of view as ((x_lo, x_hi), (y_lo, y_hi)).
- **radius** (*float, optional*) – Radius of cylinder with respect to FOV.
- **params** (*dict, optional*) – Dictionary of tissue properties, as in cylinder_2d_params.

Returns

- **PD** (*array_like*) – Proton density map.
- **T1s** (*array_like*) – T1 map.
- **T2s** (*array_like*) – T2 map.

mr_utils.test_data.phantom.cylinder_2d.**cylinder_2d_params**()

Returns properties of numerical phantom used in cylinder_2d.

Returns **params** – Dictionary of example T1, T2, and M0 values.

Return type dict

mr_utils.test_data.phantom.cylinder_2d.**spgr_2d_cylinder**(*TR=0.3*, *TE=0.003*, *alpha=1.0471975511965976*, *dims=(64, 64)*, *FOV=((-1, 1), (-1, 1))*, *radius=0.5*, *field_map=None*, *kspace=False*)

Simulates axial spoiled GRE scan of cylindrical phantom.

Parameters

- **TR** (*float, optional*) – Repetition time.
- **TE** (*float, optional*) – Echo time.
- **alpha** (*float, optional*) – Flip angle (in rad).
- **dims** (*tuple of ints, optional*) – Matrix size, (dim_x,dim_y)
- **FOV** (*tuple of tuples, optional*) – Field of view in arbitrary units, ((x_min,x_max), (y_min,y_max))
- **radius** (*float, optional*) – Radius of cylinder in arbitrary units.
- **field_map** (*array_like, optional*) – (dim_x,dim_y) field map. If None, linear gradient in x used.
- **kspace** (*bool, optional*) – Whether or not to return data in kspace or imspace.

Returns **im** – Complex simulated image with spoiled GRE contrast.

Return type array_like

phantom

Simulate shepp logan phantoms.

To Generate phantoms. You can call the following functions with the desired phantom shape as input:

- modified_shepp_logan
- shepp_logan
- yu_ye_wang

You can generate a custom phantom by specifying a list of ellipsoid parameters by calling the phantom function. Ellipsoid parameters are as follows:

- A : value inside the ellipsoid
- a, b, c : axis length of the ellipsoid (in % of the cube shape)
- x0, y0, z0 : position of the center (in % of the cube shape)
- phi, theta, psi : Euler angles defining the orientation (in degrees)

Alternatively, you can generate only one ellipsoid by calling the ellipsoid function.

Examples

To generate a phantom cube of size 32 * 32 * 32:

```
>>> from siddon.phantom import *
>>> my_phantom = shepp_logan((32, 32, 32))
>>> assert my_phantom[16, 16, 16] == -0.8
```

Notes

You can take a look at those links for explanations: http://en.wikipedia.org/wiki/Imaging_phantom http://en.wikipedia.org/wiki/Ellipsoid http://en.wikipedia.org/wiki/Euler_angles This module is largely inspired by : http://www.mathworks.com/matlabcentral/fileexchange/9416-3d-shepp-logan-phantom

Original Author: Nicolas Barbey

mr_utils.test_data.phantom.phantom.**phantom**(*shape*, *parameters_list*, *dtype=<class 'numpy.float64'>*)

Generate a cube of given shape using a list of ellipsoid parameters.

Parameters

- **shape** (*tuple of ints*) – Shape of the output cube.
- **parameters_list** (*list of dictionaries*) – List of dictionaries with the parameters defining the ellipsoids to include in the cube.
- **dtype** (*data-type, optional*) – Data type of the output ndarray.

Returns **cube** – A 3-dimensional ndarray filled with the specified ellipsoids.

Return type ndarray

See also:

shepp_logan() Generates the Shepp Logan phantom in any shape.

modified_shepp_logan() Modified Shepp Logan phantom in any shape.

yu_ye_wang() The Yu Ye Wang phantom in any shape.

ellipsoid() Generates a cube filled with an ellipsoid of any shape.

Notes

http://en.wikipedia.org/wiki/Imaging_phantom

mr_utils.test_data.phantom.phantom.**shepp_logan**(*shape*, ***kargs*)

Generates a Shepp-Logan phantom with a given shape and dtype

Parameters

- **shape** (*3-tuple of ints*) – Shape of the 3d output cube.
- **dtype** (*data-type*) – Data type of the output cube.

Returns **cube** – 3-dimensional phantom.

Return type ndarray

mr_utils.test_data.phantom.phantom.**modified_shepp_logan**(*shape*, ***kargs*)

Generates a Modified Shepp-Logan phantom with a given shape and dtype

Parameters

- **shape** (*3-tuple of ints*) – Shape of the 3d output cube.

- **dtype** (*data-type*) – Data type of the output cube.

Returns **cube** – 3-dimensional phantom.

Return type ndarray

mr_utils.test_data.phantom.phantom.**yu_ye_wang**(*shape*, ***kargs*)
Generates a Yu Ye Wang phantom with a given shape and dtype

Parameters

- **shape** (*3-tuple of ints*) – Shape of the 3d output cube.
- **dtype** (*data-type*) – Data type of the output cube.

Returns **cube** – 3-dimensional phantom.

Return type ndarray

mr_utils.test_data.phantom.phantom.**ellipsoid**(*parameters*, *shape=None*, *out=None*, *coordinates=None*)
Generates a cube filled with an ellipsoid of any shape.

Parameters

- **parameters** (*list of dictionaries*) – List of dictionaries with the parameters defining the ellipsoids to include in the cube.
- **shape** (*tuple of ints, optional*) – Shape of the output cube.
- **out** (*array_like*) – If None, initialized to all zeros.
- **coordinates** (*list*) – List of coordinates

Returns **out** – The ellipsoid.

Return type array_like

Notes

If out is given, fills the given cube instead of creating a new one.

2.12.3 coils

Extremely simple coil sensitivity maps.

Probably better to use the generate csm map functions included in ismrmrd-python-tools.

mr_utils.test_data.coils.csm.**simple_csm**(*N*, *dims=(64, 64)*)
Generate coil channel sensitivities as linear gradients in N directions.

N – number of coil sensitivities to generate. dims – tuple of dimensions.

N linear gradient gradients of size dims will be generated. These are simple because all we're doing is generating linear gradients at evenly spaced angles so the resulting maps are square.

TODO: sensitivity maps also need phases, as in: ismrmrdtools.simulation.generate_birdcage_sensitivities

Returns (N x dims[0] x dims[1]) array.

2.12.4 optimization_functions

Test functions for optimization.

See: https://en.wikipedia.org/wiki/Test_functions_for_optimization

mr_utils.test_data.optimization_functions.functions.**ackley**(*x*, *a=20*, *b=0.2*, *c=6.283185307179586*)
Ackley function.

mr_utils.test_data.optimization_functions.functions.**beale**(*x*)
Beale function.

Only for 2d x.

mr_utils.test_data.optimization_functions.functions.**bohachevsky1**(*x*)
Bohachevsky function 1. Only for 2d x.

mr_utils.test_data.optimization_functions.functions.**bohachevsky2**(*x*)
Bohachevsky function 2. Only for 2d x.

mr_utils.test_data.optimization_functions.functions.**bohachevsky3**(*x*)
Bohachevsky function 3. Only for 2d x.

mr_utils.test_data.optimization_functions.functions.**grad_ackley**(*f*, *x*, *a=20*, *b=0.2*, *c=6.283185307179586*)
Gradient of Ackley function.

mr_utils.test_data.optimization_functions.functions.**grad_bohachevsky1**(*f*, *x*)
Gradient of Bohachevsky function 1.

mr_utils.test_data.optimization_functions.functions.**grad_bohachevsky2**(*f*, *x*)
Gradient of Bohachevsky function 2.

mr_utils.test_data.optimization_functions.functions.**grad_bohachevsky3**(*f*, *x*)
Gradient of Bohachevsky function 3.

mr_utils.test_data.optimization_functions.functions.**rastrigin**(*x*, *A=10*)
Rastrigin function.

mr_utils.test_data.optimization_functions.functions.**rosenbrock**(*x*, *a=1*, *b=100*)
Rosenbrock's function.

mr_utils.test_data.optimization_functions.functions.**sphere**(*x*)
Sphere function.

2.13 Utilities

2.13.1 ellipse

General functions for working with ellipses.

mr_utils.utils.ellipse.**check_fit**(*C*, *x*, *y*)
General quadratic polynomial function.

Parameters

- **C** (*array_like*) – coefficients.
- **x** (*array_like*) – x coordinates assumed to be on ellipse.

- **y** (*array_like*) – y coordinates assumed to be on ellipse.

Returns Measure of how well the ellipse fits the points (x, y).

Return type float

Notes

We want this to equal 0 for a good ellipse fit. This polynomial is called the algebraic distance of the point (x, y) to the given conic.

This equation is referenced in[1] and[2].

References

mr_utils.utils.ellipse.**do_planet_rotation**(*I*)
Rotate complex points to fit vertical ellipse centered at (xc, 0).

Parameters **I** (*array_like*) – Complex points from SSFP experiment.

Returns

- **xr** (*array_like*) – x coordinates of rotated points.
- **yr** (*array_like*) – y coordinates of rotated points.
- **cr** (*array_like*) – Coefficients of rotated ellipse.
- **phi** (*float*) – Rotation angle in radians of effective rotation to get ellipse vertical and in the x > 0 half plane.

mr_utils.utils.ellipse.**fit_ellipse_fitzgibon**(*x*, *y*)
Python port of direct ellipse fitting algorithm by Fitzgibon et. al.

Parameters

- **x** (*array_like*) – y coordinates assumed to be on ellipse.
- **y** (*array_like*) – y coordinates assumed to be on ellipse.

Returns Ellipse coefficients.

Return type array_like

Notes

See Figure 1 from[2].

Also see previous python port: http://nicky.vanforeest.com/misc/fitEllipse/fitEllipse.html

mr_utils.utils.ellipse.**fit_ellipse_halir**(*x*, *y*)
Python port of improved ellipse fitting algorithm by Halir and Flusser.

Parameters

- **x** (*array_like*) – y coordinates assumed to be on ellipse.
- **y** (*array_like*) – y coordinates assumed to be on ellipse.

[1] Shcherbakova, Yulia, et al. "PLANET: an ellipse fitting approach for simultaneous T1 and T2 mapping using phase-cycled balanced steady-state free precession." Magnetic resonance in medicine 79.2 (2018): 711-722.

[2] Halır, Radim, and Jan Flusser. "Numerically stable direct least squares fitting of ellipses." Proc. 6th International Conference in Central Europe on Computer Graphics and Visualization. WSCG. Vol. 98. 1998.

Returns Ellipse coefficients.

Return type array_like

Notes

Note that there should be at least 6 pairs of (x,y).

From the paper's conclusion:

> "Due to its systematic bias, the proposed fitting algorithm cannot be used directly in applications where excellent accuracy of the fitting is required. But even in that applications our method can be useful as a fast and robust estimator of a good initial solution of the fitting problem..."

See figure 2 from[2].

mr_utils.utils.ellipse.**fit_ellipse_nonlin**(*x*, *y*, *polar=False*)

Fit ellipse only depending on semi-major axis and eccentricity.

Parameters

- **x** (*array_like*) – y coordinates assumed to be on ellipse.
- **y** (*array_like*) – y coordinates assumed to be on ellipse.
- **polar** (*bool, optional*) – Whether or not coordinates are provided as polar or Cartesian.

Returns

- **a** (*float*) – Semi-major axis
- **e** (*float*) – Eccentricity

Notes

Note that if polar=True, then x will be assumed to be radius and y will be assumed to be theta.

See: https://scipython.com/book/chapter-8-scipy/examples/non-linear-fitting-to-an-ellipse/

mr_utils.utils.ellipse.**get_center**(*c*)

Compute center of ellipse from implicit function coefficients.

Parameters **c** (*array_like*) – Coefficients of general quadratic polynomial function for conic funs.

Returns

- **xc** (*float*) – x coordinate of center.
- **yc** (*float*) – y coordinate of center.

mr_utils.utils.ellipse.**get_semiaxes**(*c*)

Solve for semi-axes of the cartesian form of the ellipse equation.

Parameters **c** (*array_like*) – Coefficients of general quadratic polynomial function for conic funs.

Returns

- *float* – Semi-major axis
- *float* – Semi-minor axis

Notes

https://en.wikipedia.org/wiki/Ellipse

mr_utils.utils.ellipse.**rotate_coefficients**(*c*, *phi*)
Rotate coefficients of implicit equations through angle phi.

Parameters

- **c** (*array_like*) – Coefficients of general quadratic polynomial function for conic funs.
- **phi** (*float*) – Angle in radians to rotate ellipse.

Returns Coefficients of rotated ellipse.

Return type array_like

Notes

http://www.mathamazement.com/Lessons/Pre-Calculus/09_Conic-Sections-and-Analytic-Geometry/rotation-of-axes.html

mr_utils.utils.ellipse.**rotate_points**(*x*, *y*, *phi*, *p=(0, 0)*)
Rotate points x, y through angle phi w.r.t. point p.

Parameters

- **x** (*array_like*) – x coordinates of points to be rotated.
- **y** (*array_like*) – y coordinates of points to be rotated.
- **phi** (*float*) – Angle in radians to rotate points.
- **p** (*tuple, optional*) – Point to rotate around.

Returns

- **xr** (*array_like*) – x coordinates of rotated points.
- **yr** (*array_like*) – y coordinates of rotated points.

2.13.2 find_nearest

Finding closest values in an array.

mr_utils.utils.find_nearest.**find_nearest**(*array*, *value*)
Given straws and needle, find the closest straw to the needle.

Parameters

- **array** (*array_like*) – hay stack.
- **value** (*float*) – needle.

Returns

- **idx** (*int*) – Flattened index where value is located (or closest value is located).
- *float* – The actual value at idx.

2.13.3 grad_tv

Gradient of total variation term for gradient descent update.

mr_utils.utils.grad_tv.**dTV**(*A*, *eps=1e-08*)

Compute derivative of the TV with respect to the matrix A.

Parameters

- **A** (*array_like*) – 2d matrix (can be complex).
- **eps** (*float, optional*) – small positive constant used to avoid a divide by zero.

Returns Derivative of TV w.r.t. A.

Return type array_like

Notes

Implements Equation [13] from[1].

References

2.13.4 histogram

Some functions for working with histograms.

mr_utils.utils.histogram.**dH**(*H1*, *H2*, *mode='l2'*)

Histogram metrics.

Parameters

- **H1** (*array_like*) – 1d histogram.
- **H2** (*array_like*) – 1d histogram with bins matched to H1.
- **mode** (*{'l2', 'l1', 'vcos', 'intersect', 'chi2', 'jsd', 'emd'}, optional*) – Metric to use.

Returns Distance between H1, H2.

Return type float

Notes

Similar bins means the same number and size over the same range.

Modes:

- l2 – Euclidean distance
- l1 – Manhattan distance
- vcos – Vector cosine distance
- intersect – Histogram intersection distance
- chi2 – Chi square distance

[1] Zhang, Yan, Yuanyuan Wang, and Chen Zhang. "Total variation based gradient descent algorithm for sparse-view photoacoustic image reconstruction." Ultrasonics 52.8 (2012): 1046-1055.

- jsd – Jensen-Shannan Divergence
- emd – Earth Mover's Distance

Issues:

- I'm not completely convinced that intersect is doing the right thing.

The quality of the metric will depend a lot on the qaulity of the histograms themselves. Obviously more samples and well-chosen bins will help out in the comparisons.

mr_utils.utils.histogram.**hist_match**(*source*, *template*)
Histogram matching.

Adjust the pixel values of a grayscale image such that its histogram matches that of a target image

Parameters

- **source** (*np.ndarray*) – Image to transform; the histogram is computed over the flattened array
- **template** (*np.ndarray*) – Template image; can have different dimensions to source

Returns **matched** – The transformed output image

Return type np.ndarray

Notes

https://stackoverflow.com/questions/32655686/histogram-matching-of-two-images-in-python-2-x

2.13.5 mi_ssfp

Simple maximum intensity recon for banding minimization.

mr_utils.utils.mi_ssfp.**mi_ssfp**(*images*, *pc_axis=0*, *complex=False*)
Compute maximum intensity SSFP.

Parameters

- **images** (*array_like*) – Array of phase-cycled images.
- **pc_axis** (*int, optional*) – Which dimension is the phase-cycle dimension.
- **complex** (*bool, optional*) – Whether returned data should be complex or magnitude.

Returns MI image.

Return type array_like

Notes

Implements Equation [5] from[1].

[1] Bangerter, Neal K., et al. "Analysis of multiple-acquisition SSFP." Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 51.5 (2004): 1038-1047.

References

2.13.6 orderings

Methods for orderings for signals.

Methods return flattened indices. Hopefully these orderings make the signals more sparse in some domain.

mr_utils.utils.orderings.**brute_force1d**(*x*, *T*)
Given transform matrix, T, sort 1d signal exhaustively.

Parameters

- **x** (*array_like*) – 1D signal to find ordering of.
- **T** (*array_like*) – Transform matrix.

Returns

- *array_like* – Flattened indices giving sorted order.
- *.. warning::* – This IS NOT A GOOD IDEA.

mr_utils.utils.orderings.**bulk_up**(*x*, *T*, *Ti*, *k*)
Given existing nonzero coefficients, try to make large ones larger.

Parameters

- **x** (*array_like*) – Array to find ordering of.
- **T** (*callable*) – Transform function.
- **Ti** (*callable*) – Inverse transform function.
- **k** (*float*) – Percent of coefficients to shoot for.

Returns **idx** – Flattened indices giving sorted order.

Return type array_like

Notes

Uses random_match(), should probably use min linear assignment.

mr_utils.utils.orderings.**col_stacked_order**(*x*)
Find ordering of monotonically varying flattened array, x.

Parameters **x** (*array_like*) – Array to find ordering of.

Returns **idx** – Flattened indices giving sorted order.

Return type array_like

Notes

Note that you might want to provide abs(x) if x is a complex array.

mr_utils.utils.orderings.**colwise**(*x*)
Find ordering of monotonically varying columns.

Parameters **x** (*array_like*) – Array to find ordering of.

Returns **idx** – Flattened indices giving sorted order.

Return type array_like

mr_utils.utils.orderings.**gen_sort1d**(*x*, *T*)
Given 1D transform T, sort 1d signal, x.

Parameters

- **x** (*array_like*) – 1D signal to find ordering of.
- **T** (*array_like*) – Transform matrix.

Returns Flattened indices giving sorted order.

Return type array_like

mr_utils.utils.orderings.**get_gini_sort**(*vals*)
Sort groups so that we get largest possible coefficients.

mr_utils.utils.orderings.**get_gini_sort2**(*vals*)
Sort groups to get highest possible single coefficient.

Notes

Get the largest coefficient we can and then minimize the others.

mr_utils.utils.orderings.**inverse_permutation**(*ordering*)
Given some permutation, find the inverse permutation.

Parameters **ordering** (*array_like*) – Flattened indicies, such as output of np.argsort.

Returns Inverse permutation.

Return type inverse_ordering

mr_utils.utils.orderings.**random_match**(*x*, *T*, *return_sorted=False*)
Match x to T as closely as possible pixel by pixel.

Parameters

- **x** (*array_like*) – Array to find ordering of.
- **T** (*array_like*) – Target matrix.
- **return_sorted** (*bool, optional*) – Whether or not to return the sorted matrix.

Returns

- **idx** (*array_like*) – Flattened indices giving sorted order.
- *array_like, optional* – Sorted array.

mr_utils.utils.orderings.**random_match_by_col**(*x*, *T*, *return_sorted=False*)
Given matrix T, choose reordering of x that matches it col by col.

Parameters

- **x** (*array_like*) – Array to find ordering of.
- **T** (*array_like*) – Target matrix.
- **return_sorted** (*bool, optional*) – Whether or not to return the sorted matrix.

Returns

- **idx** (*array_like*) – Flattened indices giving sorted order.
- *array_like, optional* – Sorted array.

mr_utils.utils.orderings.**random_search**(*x*, *T*, *k*, *compare='l1'*, *compare_opts=None*, *disp=False*)

Given transform T, find the best of k permutations.

Parameters

- **x** (*array_like*) – Array to find the ordering of.
- **T** (*array_like or callable*) – Transform matrix/function that we want x to be sparse under.
- **k** (*int*) – Number of permutations to try (randomly selected).
- **compare** (*{'nonzero', 'l1'} or callable, optional*) – How to compare two permutations.
- **compare_opts** (*dict, optional*) – Arguments to pass to compare function.
- **disp** (*bool, optional*) – Verbose mode.

Returns Flattened indices giving sorted order.

Return type array_like

Raises NotImplementedError – If compare is not valid option or callable.

mr_utils.utils.orderings.**rowwise**(*x*)

Find ordering of monotonically varying rows.

Parameters **x** (*array_like*) – Array to find ordering of.

Returns **idx** – Flattened indices giving sorted order.

Return type array_like

mr_utils.utils.orderings.**whittle_down**(*x*, *T*, *Ti*, *k*)

Given existing nonzero coefficients, try to remove lower ones.

Parameters

- **x** (*array_like*) – Array to find ordering of.
- **T** (*callable*) – Transform function.
- **Ti** (*callable*) – Inverse transform function.
- **k** (*float*) – Percent of coefficients to shoot for.

Returns **idx** – Flattened indices giving sorted order.

Return type array_like

Notes

Uses random_match(), should probably use min linear assignment.

2.13.7 package_script

Package a script together with all its dependencies.

For example, on a remote computer I know for a fact that numpy and scipy are available, but I cannot or cannot easily gaurantee that module x will be installed. I want to run script MyScript.py on this remote machine, but it depends on module x. package_script() will recurse through MyScript.py and prepend module x (and all of module

x's dependencies down to numpy, scipy, and default python modules, assuming I've set existing_modules=['numpy', 'scipy']).

mr_utils.utils.package_script.**get_imports**(*filename*, *existing_modules=None*)
Removes import statements and gets filenames of where imports are.

mr_utils.utils.package_script.**get_std_lib**()
Get list of all Python standard library modules.

mr_utils.utils.package_script.**package_script**(*filename*, *existing_modules=None*)
Package a script together with all dependencies.

Parameters

- **filename** (*str*) – Path to Python script we wish to package.
- **existing_modules** (*list, optional*) – List of terminating modules.

Returns Bundled python script with only existing_modules dependencies.

Return type str

Notes

"Terminating module" is a module we assume is available on the machine we want to run the packaged script on. These are python's built-in modules plus all existing_modules specified by caller.

2.13.8 percent_ripple

mr_utils.utils.percent_ripple.**percent_ripple**(*profile*)
Calculate percent ripple of the bSSFP spectral profile.

Parameters **profile** (*array_like*) – The off-resonance profile as a function of theta.

Returns Measure of ripple level of profile.

Return type float

Notes

The residual ripple can be predicted by examining the variations in the expected signal profile with free-precession angle, theta.

Implements percent ripple, Equation [11], from[1].

References

2.13.9 permutation_rank

Determining rank of a permutation and generating permutation given rank.

This implementation is due to: https://rosettacode.org/wiki/Permutations/Rank_of_a_permutation#Python

Implementation of algorithms described in[1].

[1] Bangerter, Neal K., et al. "Analysis of multiple-acquisition SSFP." Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 51.5 (2004): 1038-1047.

[1] Myrvold, Wendy, and Frank Ruskey. "Ranking and unranking permutations in linear time." Information Processing Letters 79.6 (2001): 281-284.

References

mr_utils.utils.permutation_rank.**identity_perm**(*n*)
Generate sequence 0:n-1.

mr_utils.utils.permutation_rank.**init_pi1**(*n*, *pi*)
Get the inverse permutation of pi.

mr_utils.utils.permutation_rank.**pi2rank**(*pi*, *method='rank2'*, *iterative=True*)
Return rank of permutation pi.

Parameters

- **pi** (*list*) – Permutation.
- **method** (*{'rank1', 'rank2'}, optional*) – Which ranking method to use.
- **iterative** (*bool, optional*) – Whether or not to use iterative or recursive version.

Returns **rank** – The rank of the provided permutation.

Return type int

Notes

The permutation pi should be a permutation of the list range(n) and contain n elements.

'method' should be one of {'rank1', 'rank2'} corresponding to the two schemes presented in the Myrvold and Ruskey paper. There is an iterative version available for both algorithms.

Implements algorithms from[1].

mr_utils.utils.permutation_rank.**rank2pi**(*r*, *n*, *method='rank2'*)
Given rank and permutation length produce the corresponding permutation.

Parameters

- **r** (*int*) – Rank.
- **n** (*int*) – Lenth of the permutation.
- **method** (*{'rank1', 'rank2'}*) – Which ranking method to use.

Returns **pi** – Associated permutation.

Return type list

Notes

Implements algorithms from[1].

mr_utils.utils.permutation_rank.**ranker1**(*n*, *pi*, *pi1*)
Rank1 algorithm from M&R paper.

mr_utils.utils.permutation_rank.**ranker1_iter**(*n*, *pi*, *pi1*)
Iterative version of ranker1.

mr_utils.utils.permutation_rank.**ranker2**(*n*, *pi*, *pi1*)
Ranker2 algorithm from M&R paper.

mr_utils.utils.permutation_rank.**ranker2_iter**(*n*, *pi*, *pi1*)
Iterative version of ranker2.

mr_utils.utils.permutation_rank.**unranker1**(*n*, *r*, *pi*)
Given rank produce the corresponding permutation.

Rank is given by rank1 algorithm of M&R paper.

mr_utils.utils.permutation_rank.**unranker2**(*n*, *r*, *pi*)
Given rank produce the corresponding permutation.

Rank is given by rank2 algorithm of M&R paper.

2.13.10 printtable

Tabular output.

class mr_utils.utils.printtable.**Table**(*headings*, *widths*, *formatters=None*, *pad=2*, *symbol='#'*)
Table with header and columns. Nothing fancy.

Class meant for simple column printing, e.g., printing updates for each iteration of an iterative algorithm.

headings
List of strings giving column labels.

Type list

symbol
Character to use as separator between header and table rows.

Type str

pad
Number of spaces between columns.

Type int

widths
List of widths for each column in number of characters.

Type int or list

formatters
String formatters, will use %g if None is given.

Type list

header()
Return table header.

Returns **hdr** – Table header.

Return type str

row(*vals*)
Return row of table.

Parameters **vals** (*list*) – List of numbers corresponding to display for each row.

Returns **line** – The row.

Return type str

2.13.11 rot

Rotation matrices.

mr_utils.utils.rot.**rot**(*theta*)

2D rotation matrix through angle theta (rad).

Parameters **theta** (*float*) – Angle in rad to rotate.

Returns **R** – Rotation matrix.

Return type array_like

2.13.12 sort2d

2D sorting algorithms.

mr_utils.utils.sort2d.**sort2d**(*A*)

Sorting algorithm for two-dimensional arrays.

Parameters **A** (*array_like*) – 2d array to be sorted.

Returns

- *array_like* – sorted A
- *array_like* – flattened indices giving the sort order.

Notes

Note: if A is complex, you may want to provide abs(A).

Numpy implementation of algorithm from[2].

References

mr_utils.utils.sort2d.**sort2d_loop**(*A*)

An efficient selection sorting algorithm for two-dimensional arrays.

Parameters **A** (*array_like*) – 2d array to be sorted.

Returns **B** – Monotonically sorted A (along both axes).

Return type array_like

Notes

Implementation of algorithm from[1].

[2] Zhou, M., & Wang, H. (2010, December). An efficient selection sorting algorithm for two-dimensional arrays. In Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on (pp. 853-855). IEEE.

[1] Zhou, M., & Wang, H. (2010, December). An efficient selection sorting algorithm for two-dimensional arrays. In Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on (pp. 853-855). IEEE.

References

2.13.13 sos

Simple root sum of squares image combination.

mr_utils.utils.sos.**sos**(*im*, *axes=0*)

Root sum of squares combination along given axes.

Parameters

- **im** (*array_like*) – Input image.
- **axes** (*tuple*) – Dimensions to sum across.

Returns SOS combination of image.

Return type array_like

2.13.14 wavelet

Wrappers for PyWavelets.

mr_utils.utils.wavelet.**cdf97_2d_forward**(*x*, *level*, *axes=(-2, -1)*)

Forward 2D Cohen–Daubechies–Feauveau 9/7 wavelet.

Parameters

- **x** (*array_like*) – 2D signal.
- **level** (*int*) – Decomposition level.
- **axes** (*tuple, optional*) – Axes to perform wavelet decomposition across.

Returns

- **wavelet_transform** (*array_like*) – The stitched together elements wvlt (see combine_chunks).
- **locations** (*list*) – Indices telling us how we stitched it together so we can take it back apart.

Notes

Returns transform, same shape as input, with locations. Locations is a list of indices instructing cdf97_2d_inverse where the coefficients for each block are located.

Biorthogonal 4/4 is the same as CDF 9/7 according to wikipedia[1].

References

mr_utils.utils.wavelet.**cdf97_2d_inverse**(*coeffs*, *locations*, *axes=(-2, -1)*)

Inverse 2D Cohen–Daubechies–Feauveau 9/7 wavelet.

Parameters

- **coeffs** (*array_like*) – Stitched together wavelet transform.
- **locations** (*list*) – Output of cdf97_2d_forward().

[1] https://en.wikipedia.org/wiki/ Cohen%E2%80%93Daubechies%E2%80%93Feauveau_wavelet#Numbering

- **axes** (*tuple, optional*) – Axes to perform wavelet transform across.

Returns Inverse CDF97 transform.

Return type array_like

mr_utils.utils.wavelet.**combine_chunks**(*wvlt*, *shape*, *dtype=<class 'float'>*)
Stitch together the output of PyWavelets wavedec2.

Parameters

- **wvlt** (*list of array_like*) – Output of pywt.wavedec2().
- **shape** (*tuple*) – Desired shape.
- **dtype** (*np.dtype, optional*) – Type of numpy array.

Returns

- **wavelet_transform** (*array_like*) – The stitched together elements wvlt (see notes).
- **locations** (*list*) – Indices telling us how we stitched it together so we can take it back apart.

Notes

```
We have tuples that look like this:
                             ___________________
                             |         |         |
                             | cA(LL)  | cH(LH)  |
                             |         |         |
(cA, (cH, cV, cD))  <--->    ___________________
                             |         |         |
                             | cV(HL)  | cD(HH)  |
                             |         |         |
                             ___________________
```

mr_utils.utils.wavelet.**split_chunks**(*coeffs*, *locations*)
Separate the stitched together transform into blocks again.

Parameters

- **x** (*array_like*) – Stitched together wavelet transform.
- **locations** (*list*) – Indices where the coefficients for each block are located.

Returns coeff_list – List of coefficients of wavelet decomposition (like the output of pywt.wavedec2()).

Return type list of array_like

Notes

x, locations are the output of combine_chunks().

mr_utils.utils.wavelet.**wavelet_forward**(*x*, *wavelet*, *mode='symmetric'*, *level=None*, *axes=(-2, -1)*)
Wrapper for the multilevel 2D discrete wavelet transform.

Parameters

- **x** (*array_like*) – Input data.
- **wavelet** (*str*) – Wavelet to use.

- **mode** (*str, optional*) – Signal extension mode.
- **level** (*int, optional*) – Decomposition level (must be >= 0).
- **axes** (*tuple, optional*) – Axes over which to compute the DWT.

Returns

- **wavelet_transform** (*array_like*) – The stitched together elements wvlt (see combine_chunks).
- **locations** (*list*) – Indices telling us how we stitched it together so we can take it back apart.

Notes

See PyWavelets documentation on pywt.wavedec2() for more information.

If level=None (default) then it will be calculated using the dwt_max_level function.

mr_utils.utils.wavelet.**wavelet_inverse**(*coeffs*, *locations*, *wavelet*, *mode='symmetric'*, *axes=(-2, -1)*)

Wrapper for the multilevel 2D inverse DWT.

Parameters

- **coeffs** (*array_like*) – Combined coefficients.
- **locations** (*list*) – Indices where the coefficients for each block are located.
- **wavelet** (*str*) – Wavelet to use.
- **mode** (*str, optional*) – Signal extension mode.
- **axes** (*tuple, optional*) – Axes over which to compute the IDWT.

Returns Inverse transform of wavelet transform, the original image.

Return type array_like

Notes

coeffs, locations are the output of forward().

2.14 View

A simple viewer.

The idea is for this to be really simple to use. It will do a lot of guessing if you don't provide it with details. For example, if a 3D dataset is provided as the image and you don't say which axes are in-plane, it will guess that the largest two axis are in-plane. If the 3rd dimension is small, then it will choose to view the images as a montage, if it is large it will play it as a movie. Of course there are many options if you know what you're doing (and I do, since I wrote it...).

Fourier transforms, logarithmic scale, coil combination, averaging, and converting from raw data are all supported out of the box.

mr_utils.view.view.**mat_keys**(*filename*, *ignore_dbl_underscored=True*, *no_print=False*)

Give the keys found in a .mat.

Parameters

- **filename** (*str*) – .mat filename.
- **ignore_dbl_underscored** (*bool, optional*) – Remove keys beginng with two underscores.
- **no_print** (*bool, optional*) – Don't print out they keys.

Returns **keys** – Keys present in dictionary of read in .mat file.

Return type list

mr_utils.view.view.**view**(*image*, *load_opts=None*, *is_raw=None*, *is_line=None*, *prep=None*, *fft=False*, *fft_axes=None*, *fftshift=None*, *avg_axis=None*, *coil_combine_axis=None*, *coil_combine_method='walsh'*, *coil_combine_opts=None*, *is_imspace=False*, *mag=None*, *phase=False*, *log=False*, *imshow_opts={'cmap': 'gray'}*, *montage_axis=None*, *montage_opts={'padding_width': 2}*, *movie_axis=None*, *movie_interval=50*, *movie_repeat=True*, *save_npy=False*, *debug_level=10*, *test_run=False*)

Image viewer to quickly inspect data.

Parameters

- **image** (*str or array_like*) – Name of the file including the file extension or numpy array.
- **load_opts** (*dict, optional*) – Options to pass to data loader.
- **is_raw** (*bool, optional*) – Inform if data is raw. Will attempt to guess from extension.
- **is_line** (*bool, optional*) – Whether or not this is a line plot (as opposed to image).
- **prep** (*callable, optional*) – Lambda function to process the data before it's displayed.
- **fft** (*bool, optional*) – Whether or not to perform n-dimensional FFT of data.
- **fft_axes** (*tuple, optional*) – Axis to perform FFT over, determines dimension of n-dim FFT.
- **fftshift** (*bool, optional*) – Whether or not to perform fftshift. Defaults to True if fft.
- **avg_axis** (*int, optional*) – Take average over given set of axes.
- **coil_combine_axis** (*int, optional*) – Which axis to perform coil combination over.
- **coil_combine_method** (*{'walsh', 'inati', 'pca'}, optional*) – Method to use to combine coils.
- **coil_combine_opts** (*dict, optional*) – Options to pass to the coil combine method.
- **is_imspace** (*bool, optional*) – Whether or not the data is in image space. For coil combine.
- **mag** (*bool, optional*) – View magnitude image. Defaults to True if data is complex.
- **phase** (*bool, optional*) – View phase image.
- **log** (*bool, optional*) – View log of magnitude data. Defaults to False.
- **imshow_opts** (*dict, optional*) – Options to pass to imshow. Defaults to { 'cmap'='gray' }.

- **montage_axis** (*int, optional*) – Which axis is the number of images to be shown.
- **montage_opts** (*dict, optional*) – Additional options to pass to the skimage.util.montage.
- **movie_axis** (*int, optional*) – Which axis is the number of frames of the movie.
- **movie_interval** (*int, optional*) – Interval to give to animation frames.
- **movie_repeat** (*bool, optional*) – Whether or not to put movie on endless loop.
- **save_npy** (*bool, optional*) – Whether or not to save the output as npy file.
- **debug_level** (*logging_level, optional*) – Level of verbosity. See logging module.
- **test_run** (*bool, optional*) – Doesn't show figure, returns debug object. Mostly for testing.

Returns

- **data** (*array_like*) – Image data shown in plot.
- *dict, optional* – All local variables when test_run=True.

Raises

- `Exception` – When file type is not in ['dat', 'npy', 'mat', 'h5'].
- `ValueError` – When coil combine requested, but fft_axes not set.
- `AssertionError` – When Walsh coil combine requested but len(fft_axes) =/= 2.
- `ValueError` – When there are too many dimension to display.

This repo is a collection of my implementations of algorithms and tools for MR image reconstruction, mostly in python.

CHAPTER 3

Orientation

There are few different things going on here. There are algorithms, like the geometric solution to the elliptical signal model, as well as simulations, like simulated bSSFP contrast.

There's also some python functions and objects that interact with more polished tools such as Gadgetron and BART. You can use these python interfaces to easily write in Gadgetron, MATLAB, or BART functionality into your python scripts. These functions are written with the assumption of Gadgetron, MATLAB, etc. being run on some processing server (not necessarily on your local machine). If you use these, you'll want to create a config file.

CHAPTER 4

Documentation and Tests

Documentation is almost exclusively found in the docstrings of modules, functions, and classes.

Another great way to learn how things are used is by looking in the examples. Run examples from the root directory (same directory as setup.py) like this:

```
python3 examples/cs/reordering/cartesian_pe_fd_iht.py
```

If there's not an example, there might be some tests. Individual tests can be run like this from the root directory (I recommend that you run tests from the home directory - imports will get messed up otherwise):

```
python3 -m unittest mr_utils/tests/recon/test_gs_recon.py
```

All tests can be run like so:

```
pyhon3 -m unittest discover
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Y