# dMRI Phantom Utilities Documentation

**Tristan Kuehn**

**Feb 18, 2020**

# Contents:

dMRI phantom utilities provide a convenient framework for analyzing diffusion-weighted MR images of 3D printed axon-mimetic (3AM) phantoms. Using this package, you can automatically mask the phantoms in an image, perform a DTI or DKI fit of phantom data, and generate a ground truth image of fibre orientation or crossing angle registered to the source DWI.

CHAPTER 1

# Installation

dMRI Phantom Utilities can be installed from PyPI.:

```
pip install dmriphantomutils
```

# CHAPTER 2

# Image preparation

Having prepared a sample with a set of 3D printed phantoms for analysis, there are a few things to consider when designing a scan of those phantoms.

This package has been designed for images with anisotropic voxels, where each z-slice covers the thickness of an entire phantom. This setup using thick voxels provides a good SNR and ensures each voxel samples multiple layers of material in the phantom. The practical effect is that functions in this package that conceptually work with a single phantom expect the DWI data from that phantom to be confined to a single z-slice. It is possible to work around this expectation, dMRI Phantom Utilities will be easiest to use if the source DWIs contain one phantom per z-slice.

# CHAPTER 3

# Usage Example

To demonstrate the capabilities of dMRI Phantom Utilities, we'll perform a DTI fit of some phantom data, and use a description of the phantoms to produce ground truth images describing those phantoms.

We'll start by loading the example phantom data (available for download here):

```
import os
from dmriphantomutils import image_io
source_dir = './resources/manual_complex_bids/sub-01/ses-01/dwi/'
unmasked_dwi = image_io.load_dwi(
    os.path.join(source_dir, 'sub-01_ses-01_dwi.nii.gz'),
    os.path.join(source_dir, 'sub-01_ses-01_dwi.bval'),
    os.path.join(source_dir, 'sub-01_ses-01_dwi.bvec'))
```

To generate ground truth images of our phantoms, we'll need to use `scan_info` to describe their infill (and some supplementary information about the study):

```
import datetime
from dmriphantomutils import scan_info
bending = scan_info.Phantom(
    225, 30, 0.1, 100, scan_info.ConcentricArcPattern((0, 10)))
fanning = scan_info.Phantom(
    225, 30, 0.1, 100, scan_info.AlternatingPattern(
        scan_info.ConcentricArcPattern((-5, 0)),
        scan_info.ConcentricArcPattern((5, 0))))
kissing = scan_info.Phantom(
    225, 30, 0.1, 100, scan_info.AlternatingPattern(
        scan_info.ConcentricArcPattern((0, 6.5)),
        scan_info.ParallelLinePattern(90)))
crossing = scan_info.Phantom(
    225, 30, 0.1, 100, scan_info.AlternatingPattern(
        scan_info.ParallelLinePattern(0),
        scan_info.ParallelLinePattern(90)))
straight = scan_info.Phantom(
    225, 30, 0.1, 100, scan_info.ParallelLinePattern(0))
```

(continues on next page)

(continued from previous page)

```
water = scan_info.WaterSlice()

tube = [water, kissing, fanning, bending, crossing, straight]
scan = scan_info.SingleScan(slice(0, 6))
session = scan_info.ScanSession(datetime.date(2019, 7, 9), [scan])
study = scan_info.Study('bending', tube, [session])
```

To efficiently perform our DTI fit and automatically register our ground truths, we'll want a mask for each of our phantoms. We can use `automask` to generate these masks:

```
import numpy as np
from dmriphantomutils import automask

slice_masks = [automask.mask_phantom(
    unmasked_dwi.get_image()[..., slice_idx, 0]) for slice_idx in range(6)]
phantom_masks = []
for slice_idx in range(6):
    phantom_mask = np.zeros(unmasked_dwi.get_image().shape[0:3])
    phantom_mask[..., slice_idx] = slice_masks[slice_idx]
    phantom_masks.append(phantom_mask)
build_dir = './build/'
os.makedirs(build_dir, exist_ok=True)
for mask, idx in zip(phantom_masks, range(6)):
    image_io.save_image(mask, unmasked_dwi.img.affine,
        os.path.join(build_dir, 'mask_slice_' + str(idx) + '.nii.gz'))

# apply masks to raw nifti
phantom_dwis = [
    image_io.MaskedDiffusionWeightedImage(
        unmasked_dwi.img, unmasked_dwi.gtab, mask)
    for mask in phantom_masks]
```

Note that while `automask` generally does a good job filtering out any air bubbles in phantoms, it's a good idea to take a look at the b0 images and manually adjust the masks as necessary.

We'll now perform our DTI fit, and save a copy of the mean diffusivity maps:

```
from dmriphantomutils import dipy_fit
dtifits = [dipy_fit.fit_dti(dwi) for dwi in phantom_dwis]
for fit, dwi, idx in zip(dtifits, phantom_dwis, range(6)):
    dipy_fit.save_dti_metric_imgs(
        dwi,
        fit,
        md_path=os.path.join(build_dir, 'md_slice_' + str(idx) + '.nii.gz'))
```

With that done, we can use our MD map to find the fiducials in the phantom (the fiducials are holes in the phantom containing free water, so MD should be highest in the fiducial), and our mask to find each phantom's centroid:

```
from dmriphantomutils import transform_data
fiducials = [np.unravel_index(np.argmax(fit.md), fit.md.shape)[0:2]
    for fit in dtifits]
centroids = [transform_data.find_centroid(mask[..., slice_idx])
    for mask, slice_idx in zip(phantom_masks, range(6))]
```

Finally, we can give `transform_data` the fiducials, centroids, and phantom information to produce ground truth images in image space:

```
for mask, phantom, centroid, fiducial, dwi, slice_idx in zip(
        phantom_masks, tube, centroids, fiducials, phantom_dwis, range(6)):
    for metric, generator in (
            phantom.infill_pattern.get_geometry_generators().items()):
        metric_img = transform_data.gen_geometry_data(
            mask,
            generator,
            centroid,
            dwi.img.header['pixdim'][1],
            fiducial=(fiducial[0], fiducial[1]))
        image_io.save_image(
            metric_img, dwi.img.affine,
            os.path.join(
                build_dir,
                'slice_' + str(slice_idx) + '_metric_' + metric + '.nii.gz'))
```

We're left with masks, MD maps, and ground truth maps for each phantom in our source image. Let's make a table summarizing the MD and ground truth in one of the phantoms:

```
# load the niftis we already produced
md_img = image_io.load_derived_image(
    os.path.join(build_dir, 'md_slice_2.nii.gz'),
    mask_path=os.path.join(build_dir, 'mask_slice_2.nii.gz'))
angle_img = image_io.load_derived_image(
    os.path.join(build_dir, 'slice_2_metric_crossing_angle.nii.gz'),
    mask_path=os.path.join(build_dir, 'mask_slice_2.nii.gz'))

data_table = image_io.gen_table([angle_img, md_img])

# save our table as a tsv for further processing
np.savetxt(
    os.path.join(build_dir, 'slice_2_summary.tsv'),
    data_table,
    delimiter='\t',
    header='\t'.join(['crossing_angle', 'md']),
    comments='')
```

With this kind of table, it's straightforward to perform further analysis of your phantom data with a tool like pandas or R.

CHAPTER 4

# Module index

## 4.1 automask module

Automatically generate masks for a phantom in a scan.

automask.**main**(*nifti_path*, *slice_idx*, *mask_output*)
Load an image and mask one z-slice.

**Parameters**

- **nifti_path** (*str*) – Path to the image to be masked.
- **slice_idx** (*int*) – Index of the slice to be masked.
- **mask_output** (*str*) – Path to which the mask image should be saved.

automask.**mask_phantom**(*slice_b0*)
Generate a mask for the phantom material in a slice.

Assuming the phantom's diameter is similar to the test tube's, Otsu's method segments the phantom well. The initial results are eroded to avoid outliers near the edges of the phantom.

Big enough air bubbles may also be masked out by this function.

**Parameters** **slice_b0** (*array_like*) – 2D array of image data from a single slice (containing a single phantom).

**Returns** 2D boolean array, where True values indicate phantom voxels.

**Return type** array_like

## 4.2 b_selection module

Filter a set of gradients by direction and b-val.

b_selection.**get_acquisitions_by_bval**(*bvals*, *lower*, *upper*)
Get b-vals within a certain range.

**Parameters**

- **bvals** (*array_like*) – A 1D array of b-values to be filtered.
- **lower** (*float*) – The minimum b-value to include.
- **upper** (*float*) – The maximum b-value to include.

**Returns** Logical index array for b-values to include.

**Return type** array_like

b_selection.**get_acquisitions_by_dir**(*bvecs*, *phi*, *theta*, *tolerance*)
Get b-vectors close to a given direction.

**Parameters**

- **bvecs** (*array_like*) – 2D array of b-vectors to be filtered.
- **phi** (*float*) – Azimuthal angle of the direction, between 0 and 180.
- **theta** (*float*) – Polar angle of the point, between -180 and 180.
- **tolerance** (*float*) – b-vecs with spherical distance less than this tolerance will be included.

**Returns** Index array of the b-vectors to be included.

**Return type** array_like

b_selection.**spherical_distance**(*theta1*, *phi1*, *theta2*, *phi2*)
Get the spherical distance between two points on the unit sphere.

**Parameters**

- **theta2** (*theta1,*) – Polar angles of the two points, between 0 and pi.
- **phi1** (*phi1,*) – Azimuthal angles of the two points, between -pi and pi.

**Returns** Spherical distance between the given points.

**Return type** float

## 4.3 dipy_fit module

Wrapper that uses DIPY to fit DTI and DKI representations.

dipy_fit.**fit_dki**(*dwi*, *blur=False*)
Fit a DKI model to a DWI, applying a mask if provided.

**Parameters**

- **dwi** (DiffusionWeightedImage) – DWI data to fit to the model.
- **blur** (*bool, optional*) – True if the image should be blurred before the model fit.

**Returns** **dwifit** – A fit from which parameter maps can be generated

**Return type** DiffusionKurtosisFit

dipy_fit.**fit_dti**(*dwi*)
Fit a DTI model to a DWI, applying a mask if provided.

**Parameters**

- **dwi** (DiffusionWeightedImage) – DWI data to fit to the model.

- **blur** (*bool, optional*) – True if the image should be blurred before the model fit.

**Returns** **dwifit** – A fit from which parameter maps can be generated

**Return type** TensorFit

dipy_fit.**main**(*nifti_path*, *bval_path*, *bvec_path*, *mask_path=None*, *blur=False*, *fa_path=None*, *md_path=None*, *ad_path=None*, *rd_path=None*, *mk_path=None*, *ak_path=None*, *rk_path=None*)

Load and fit an image to a DKI model, then save its parameters.

This is meant to deal with the functionality of this module being called as a script.

**Parameters**

- **nifti_path** (*str*) – Path to the nifti DWI
- **bval_path** (*str*) – Path to the .bval file
- **bvec_path** (*str*) – Path to the .bvec file
- **mask_path** (*str, optional*) – Path to the nifti mask, if one exists
- **fa_path** (*str, optional*) – Path to which the fractional anisotropy image should be saved
- **md_path** (*str, optional*) – Path to which the mean diffusivity image should be saved
- **ad_path** (*str, optional*) – Path to which the axial diffusivity image should be saved
- **rd_path** (*str, optional*) – Path to which the radial diffusivity image should be saved
- **mk_path** (*str, optional*) – Path to which the mean kurtosis image should be saved
- **ak_path** (*str, optional*) – Path to which the axial kurtosis image should be saved
- **rk_path** (*str, optional*) – Path to which the radial kurtosis image should be saved

dipy_fit.**save_dki_metric_imgs**(*dwi*, *dkifit*, *fa_path=None*, *md_path=None*, *ad_path=None*, *rd_path=None*, *mk_path=None*, *ak_path=None*, *rk_path=None*)

Save DTI metric images as NIFTI files.

**Parameters**

- **dwi** (DiffusionWeightedImage) – The source image from which the DTI metrics are derived.
- **dkifit** (*DiffusionKurtosisFit*) – The fit data from the DKI.
- **md_path, ad_path, rd_path** (*fa_path,*) – The filepaths to which each DTI metric image should be saved.
- **ak_path, rk_path** (*mk_path,*) – The filepaths to which each DKI metric image should be saved.

dipy_fit.**save_dti_metric_imgs**(*dwi*, *dtifit*, *fa_path=None*, *md_path=None*, *ad_path=None*, *rd_path=None*)

Save DTI metric images as NIFTI files.

**Parameters**

- **dwi** (DiffusionWeightedImage) – The source image from which the DTI metrics are derived.
- **dtifit** (*TensorFit*) – The fit data from the DTI.
- **md_path, ad_path, rd_path** (*fa_path,*) – The filepath to which each metric image should be saved.

## 4.4 image_io module

Wrappers for saving and loading DWIs of 3D printed phantoms.

In this module, images are classified as DWIs or derived images. A DWI should be the raw 4D data from a diffusion MRI scan, and have associated information about the diffusion gradients. A derived image should be (usually) 3D data from analysis of a DWI.

Either of the two may have a mask associated with them.

**class** image_io.**DerivedImage**(*img*)
Bases: object

Wrapper class including an image of data derived from a DWI.

**Parameters** **img** (*SpatialImage*) – The NiBabel image of the derived data.

**get_flat_data**()
A 1D numpy array with the image data.

**get_image**()
A 3D numpy array with the image data.

**class** image_io.**DiffusionWeightedImage**(*img*, *gtab*)
Bases: object

Wrapper class including image and gradient data.

**Parameters**

- **img** (*SpatialImage*) – The NiBabel image of the DWI
- **gtab** (*GradientTable*) – The DIPY gradient table associated with the scan

**get_flat_data**()
A 1D numpy array with the image data.

**get_image**()
A 3D numpy array with the image data.

**class** image_io.**MaskedDerivedImage**(*img*, *mask*)
Bases: *image_io.DerivedImage*

Wraps an image of data derived from a DWI with a mask.

**Parameters**

- **img** (*SpatialImage*) – The NiBabel image of the derived data.
- **mask** (*array_like*) – A 3D binary numpy array, where 1s indicate voxels to be included.

**get_flat_data**()
A 1D numpy array with the masked derived data.

**get_image**()
A 3D numpy array with the derived data, ignoring the mask.

**class** image_io.**MaskedDiffusionWeightedImage**(*img*, *gtab*, *mask*)
Bases: *image_io.DiffusionWeightedImage*

Wrapper class including an image, mask, and gradient data.

**Parameters**

- **img** (*SpatialImage*) – The NiBabel image of the DWI

- **gtab** (*GradientTable*) – The DIPY gradient table associated with the scan
- **mask** (*array_like*) – A binary numpy array, where 1s indicate voxels to be included.

**get_flat_data**()
A 1D numpy array with only the masked data.

**get_image**()
A 3D numpy array with the image data, ignoring the mask.

image_io.**gen_table**(*derived_images*)
Organize the model outputs for a phantom in a table.

**Parameters** **derived_images** (*collection of MaskedDerivedImages*) – A collection of derived images, each corresponding to a single model output from the same phantom. Each of these images must have the same mask associated with them.

**Returns** A table where each row corresponds to one voxel, and each column corresponds to one derived image.

**Return type** array_like

image_io.**load_derived_image**(*image_path*, *mask_path=None*)
Load the data from a derived image.

**Parameters**

- **image_path** (*string*) – Path to the nifti derived data volume.
- **mask_path** (*string, optional*) – Path to the nifti mask, if one exists

**Returns** **img** – The derived data with a mask, if applicable.

**Return type** *DerivedImage*

image_io.**load_dwi**(*nifti_path*, *bval_path*, *bvec_path*, *mask_path=None*, *b0_threshold=250*)
Load the data needed to process a diffusion-weighted image.

**Parameters**

- **nifti_path** (*string*) – Path to the nifti DWI
- **bval_path** (*string*) – Path to the .bval file
- **bvec_path** (*string*) – Path to the .bvec file
- **mask_path** (*string, optional*) – Path to the nifti mask, if one exists
- **b0_threshold** – Threshold below which a b-value is considered zero

**Returns** **img** – The DWI data with a mask, if applicable.

**Return type** *DiffusionWeightedImage*

image_io.**save_image**(*data*, *affine*, *output_path*)
Save some data to a nifti file.

**Parameters**

- **data** (*array_like*) – The image data to be saved
- **affine** – The affine transform to be used
- **output_path** (*string*) – Path to the file to be saved

## 4.5 scan_info module

A set of classes used to describe a phantom dataset.

Broadly, the hierarchy of classes here is as follows:

- A Study is comprised of a set of ScanSessions, each examining the same sample test tube.
- A ScanSession is comprised of a set of SingleScans of (different regions of) the sample.
- A sample test tube contains a set of Phantoms.
- Each phantom has an infill Pattern.

**class** scan_info.**AlternatingPattern**(*pattern_0*, *pattern_1*)
Bases: object

An infill pattern that changes from layer to layer.

**Parameters** **pattern_1** (*pattern_0,*) – The two patterns that alternate.

**get_geometry_generators**()
A dictionary of functions that describe the infill geometry.

For alternating patterns, only the crossing angle has a consistent definition, which is based on the directions of the underlying infill patterns.

**class** scan_info.**ConcentricArcPattern**(*origin*)
Bases: object

An infill pattern composed of concentric arcs.

This pattern represents bending fibres in the brain.

**Parameters** **origin** (*tuple of float*) – The x and y coordinates of the arcs' common centre, relative to the centroid of the phantom's cross section.

**get_geometry_generators**()
A dictionary of functions that describe the infill geometry.

For concentric arc infill, there is a constant crossing angle (0), but the fibre direction (tangent to each arc) changes and there is an arc radius that changes in the phantom.

**class** scan_info.**EmptyPattern**
Bases: object

A placeholder infill pattern for empty slices.

**get_geometry_generators**()

**class** scan_info.**ParallelLinePattern**(*cura_angle*)
Bases: object

An infill pattern composed only of parallel lines.

**Parameters** **cura_angle** (*float*) – The "direction" of the lines, as specified in cura, in degrees.

**get_geometry_generators**()
A dictionary of functions that describe the infill geometry.

For parallel line infill, the fibre direction and crossing angle are both constant because there is no orientation dispersion of any kind.

**class** scan_info.**Phantom**(*hotend_temp*, *print_speed*, *layer_thickness*, *infill_density*, *infill_pattern*)
Bases: object

Description of a phantom's design and print parameters.

This class has no intrinsic functionality, but contains a full description of a phantom's design and print process.

**Parameters**

- **hotend_temp** (*float*) – The temperature, in degrees Celsius, at which the phantom was 3D printed.
- **print_speed** (*float*) – The speed, in mm/s, at which the phantom was 3D printed.
- **layer_thickness** (*float*) – The thickness, in mm, of each of the phantom's layers.
- **infill_density** (*float*) – The infill density, as a percentage, with which the phantom was printed.
- **infill_pattern** (*Pattern*) – The infill pattern with which the phantom was printed.

**class** scan_info.**ScanSession**(*date*, *scans*)
Bases: object

A description of a single day of scans covering a set of phantoms.

**Parameters**

- **date** (*datetime.date*) – The date of the scan session.
- **scans** (*list of SingleScan*) – The set of scans that were conducted.

**class** scan_info.**SingleScan**(*tube_slice*)
Bases: object

A single scan covering some subset of a set of phantoms.

There should be one DWI that corresponds to each scan.

**Parameters** **tube_slice** (*slice*) – The slice of a list of phantoms that was covered in the scan.

**class** scan_info.**Study**(*name*, *tube*, *sessions*)
Bases: object

A data class fully documenting a study using one set of phantoms.

A "study," in this context, refers to a series of scans of the same set of phantoms.

**Parameters**

- **name** (*str*) – A name for the study.
- **tube** (*list of Phantom*) – A list containing each phantom in the set that was scanned.
- **sessions** (*list of ScanSession*) – A list containing each session of phantom scans.

**class** scan_info.**WaterSlice**
Bases: object

A placeholder for a scan slice containing only water.

## 4.6 transform_data module

Enable transformation of data for comparison to ground truth.

If we know the infill pattern of a given phantom, we know a few things about the geometry of diffusion in that phantom. This module provides a way to compare scan data to that known information.

Specifically, we define a "ground truth space," where the centre of the phantom is at the origin, and a fiducial visible from the image is on the negative y-axis. Then a translation and rotation can move each voxel's coordinates from image space to ground truth space.

transform_data.**find_centroid**(*mask*)

Find the centroid of a phantom's mask.

**Parameters** **mask** (*array_like*) – A 2D binary array, where 1s indicate voxels containing a phantom.

**Returns** A 1D array containing the coordinates of the mask's centroid.

**Return type** array_like

transform_data.**gen_geometry_data**(*mask_data*, *geometry_generator*, *centroid*, *scaling*, *angle=None*, *fiducial=None*)

Generate geometric ground truth data from a mask.

**Parameters**

- **mask_data** (*array_like*) – 3D mask of points to be analyzed.
- **geometry_generator** (*function(tuple of float)*) – Function to get the quantity of interest given a point.
- **centroid** (*tuple of int*) – Centroid of the phantom in image space.
- **scaling** (*float*) – Isotropic scale factor from coords to image space.
- **angle** (*float, optional*) – The angle by which the phantom would need to be rotated to have the fiducial at the bottom in the x-y plane. Exactly one of angle or fiducial must be included as an argument.
- **fiducial** (*tuple of float, optional*) – The location of the fiducial in image space. Exactly one of fiducial or angle must be included as an argument.

**Returns** An image of the calculated geometry data

**Return type** array_like

transform_data.**transform_image_point**(*point*, *centroid*, *angle=None*, *fiducial=None*)

Perform a rigid transform of a given point.

The infill pattern definitions assume the origin is at the centroid of the phantom. This is never the case for scan data, so to compare scan data to a ground truth, we need to translate the image data to move the phantom's centroid to the origin, and rotate it to align a fiducial to the known ground truth.

**Parameters**

- **point** (*tuple of int*) – The indices of the point to be transformed in image space
- **centroid** (*tuple of float*) – The indices of the phantom's centroid in image space
- **angle** (*float, optional*) – The angle by which the phantom would need to be rotated to have the fiducial at the bottom in the x-y plane. Exactly one of angle or fiducial must be included as an argument.
- **fiducial** (*tuple of float, optional*) – The location of the fiducial in image space. Exactly one of fiducial or angle must be included as an argument.

**Returns** The corresponding indices of the original point in ground truth space.

**Return type** tuple of float

# Python Module Index

# Index

## S



## T



## W