Dicomifier is a set of tools to convert Bruker data to DICOM files, and DICOM files to NIfTI. It retains meta-data (e.g. MR parameters such as echo time or subject parameters such as weight or height) throughout the conversion process, and aligns the meta-data from Bruker on the DICOM dictionary for unified processing pipelines.

Getting started? Have a look at the installation instructions and basic usage. If you’re in a hurry and have Anaconda installed, conda install -c conda-forge dicomifier should get you going.

Need to know more about your options and arguments? The commands reference is for you.

Want to find out how it works? The API reference is a good place to start, followed by the source code.

Interested in how the vendors store diffusion MRI meta-data? There is a section just for you.

Using the software? No citation is mandatory, but it is always appreciated. See the entry on Zenodo for details.
1.1 Installing pre-compiled packages

If you have Anaconda installed, a pre-built package is available on conda-forge: running conda install -c conda-forge dicomifier will install the latest stable version.

Debian and Ubuntu user can also use unofficial packages: following the instructions to set up the unofficial repository, and install the *dicomifier-cli* package with the usual tools provided by your distribution. The latest stable version of Dicomifier is available for the two latest long-term support versions of Ubuntu and for the two latest versions of Debian. Old versions of Ubuntu and Debian may be supported, but may not have the latest version of Dicomifier.

1.2 Compiling from source

If you need to compile Dicomifier from scratch, you will need

- A build environment with a C++11 compiler and Python 3.5
- CMake 3.5
- The Boost libraries 1.58
- Pybind11 2.0.1
- Odil 0.12.0
- The dateutil, nibabel and numpy Python modules

The compilation process follows the usual CMake workflow: create a build directory, run cmake to configure the build environment, and run cmake --build . to build Dicomifier.

You may also refer to the continuous integration scripts to see working examples.
1.3 Basic usage

Dicomifier has a single command line executable, called dicomifier, which accepts several commands:

- List convertible data in a directory: `dicomifier list my_directory`
- Convert Bruker data to DICOM: `dicomifier to-dicom bruker_directory dicom_directory`
- Convert Bruker or DICOM data to NIfTI: `dicomifier to-nifti bruker_or_dicom_directory nifti_directory`

To list all available commands, run `dicomifier --help`; to get help about a specific command, run `dicomifier some-command --help`. 
Dicomifier has a single command line executable, called `dicomifier`, which accepts several commands:

- List convertible data in a directory: `dicomifier list my_directory`
- Convert Bruker data to DICOM: `dicomifier to-dicom bruker_directory dicom_directory`
- Convert Bruker or DICOM data to NIfTI: `dicomifier to-nifti bruker_or_dicom_directory nifti_directory`

To list all available commands, run `dicomifier --help`; to get help about a specific command, run `dicomifier some-command --help`.

## 2.1 list

List Bruker and DICOM data which can be converted (alias: `ls`).

**Usage:**

```
dicomifier list [options] source [source ...]
```

### 2.1.1 Arguments

**source**
- Bruker directory, DICOM file, directory or DICOMDIR.

### 2.1.2 Options

- `--json`
  - Display contents in JSON format
- `--verbose <level>`
  - Verbosity level (`warning`, `info`, or `debug`
2.2 search

Search for DICOM data matching the supplied search terms.

Search terms can be specified either as a tag name or tag number, and may be followed by a value. Value-less search terms match files where the tag is present, while valued search terms match elements following the DICOM rules (match any value in the element, ? and * wildcards for strings).

Usage:

dicomifier search [options] source [source ...]

2.2.1 Arguments

source
  DICOM file, directory or DICOMDIR.

2.2.2 Options

-m, --match
  Search criteria

-p, --pipe
  Format data to be sent to dicomifier to-nifti

-v <level>, --verbose <level>
  Verbosity level (warning, info, or debug)

2.3 to-dicom

Convert Bruker data to DICOM (aliases: dicom, dcm).

Usage:

dicomifier to-dicom [options] source [source ...] destination

2.3.1 Arguments

source
  Directories to scan for convertible data. This can be a top-level Bruker directory, or a sub-directory containing a single experiment or reconstruction.

destination
  Output directory

2.3.2 Options

--dicomdir, -d
  Creates a DICOMDIR in the destination directory
--layout <layout>, -l <layout>
Layout of the DICOM files on the disk, among
  • flat: all files are stored in the destination directory
  • nested: files are stored using a subject/study/series hierarchy in the destination directory (default)

--multiframe, -m
Save multiframe DICOM files (one file per series) instead of the classic single-frame DICOM files (one file per slice)

--transfer-syntax <syntax>, -t <syntax>
Transfer syntax of the output files, among
  • ImplicitVRLittleEndian (default)
  • ExplicitVRLittleEndian

-v <level>, --verbose <level>
Verbosity level (warning, info, or debug)

2.4 to-nifti

Convert Bruker or DICOM data to NIfTI (aliases: nifti, nii).

Usage:

dicomifier to-nifti [options] source [source ...] destination

2.4.1 Arguments

source
Directories to scan for convertible data. This can be a Bruker directory, DICOM files or directories or DICOMDIR, or -. In the latter case, a NUL-separated list of file names is read from the standard input.

destination
Output directory

2.4.2 Options

--dtype <dtype>, -d <dtype>
Pixel type

--zip, -z
Compress NIfTI files

-v <level>, --verbose <level>
Verbosity level (warning, info, or debug)

2.5 diffusion-scheme

Convert dMRI meta-data stored in JSON (aliases: diffusion, diff).

Usage:
2.5.1 Arguments

source
Source JSON file

format
Output format, among:
  * mrtrix
  * fsl

[destination ...]
Output files, depends on the output format
  * mrtrix: single scheme file
  * fsl: the bvecs and bvals files, in that order

2.5.2 Options

-i <image>, --image <image>
Image file for formats using image-based direction coordinates

-v <level>, --verbose <level>
Verbosity level (warning, info, or debug)
3.1 dicomifier.bruker

class dicomifier.bruker.Dataset
A Bruker dataset. This objects implements part of the mapping protocol: "foo" in d, d["foo"] and for
x in d are valid constructs.

    get_field(self: dicomifier.bruker.Dataset, name: str) → dicomifier::bruker::Field
    Return the field, throw an exception if field is missing.

    get_used_files(self: dicomifier.bruker.Dataset) → List[str]
    Return a set of files used to create the dataset (except the PixelData file)

    has_field(self: dicomifier.bruker.Dataset, name: str) → bool
    Test if dataset contains a given field.

    items(self: dicomifier.bruker.Dataset) → Iterator
    Iterate on fields

    load(self: dicomifier.bruker.Dataset, path: str) → None
    Load dataset from file, update any existing field.

class dicomifier.bruker.Field
A field (i.e. record in JCAMP-DX jargon) of a Bruker data set: container of named variant-typed items.

class Item

    get_int(self: dicomifier.bruker.Field.Item) → int
    Return an int-convertible item, throw an exception if not convertible.

    get_real(self: dicomifier.bruker.Field.Item) → float
    Return a real-convertible item, throw an exception if not convertible.

    get_string(self: dicomifier.bruker.Field.Item) → str
    Return a string item, throw an exception if not string-typed.
get\_struct \((self: \text{dicomifier.bruker.Field.Item}) \rightarrow \text{List[dicomifier.bruker.Field.Item]}\)
Return a struct item, throw an exception if not struct-typed.

is\_int \((self: \text{dicomifier.bruker.Field.Item}) \rightarrow \text{bool}\)
Test whether item is an int.

is\_real \((self: \text{dicomifier.bruker.Field.Item}) \rightarrow \text{bool}\)
Test whether item is a real.

is\_string \((self: \text{dicomifier.bruker.Field.Item}) \rightarrow \text{bool}\)
Test whether item is a string.

is\_struct \((self: \text{dicomifier.bruker.Field.Item}) \rightarrow \text{bool}\)
Test whether item is a struct.

get\_int \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{int}\)
Return a int-convertible item, throw an exception if not convertible.

get\_real \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{float}\)
Return a real-convertible item, throw an exception if not convertible.

get\_string \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{str}\)
Return a string item, throw an exception if not string-typed.

get\_struct \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{List[boost::variant<boost::detail::variant::recursive\_flag<long>, double, std::basic\_string<char, std::char\_traits<char>, std::allocator<char>>, std::vector<boost::recursive\_variant, std::allocator<boost::recursive\_variant>>]}\)
Return a struct item, throw an exception if not struct-typed.

is\_int \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{bool}\)
Test whether item is an int.

is\_real \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{bool}\)
Test whether item is a real.

is\_string \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{bool}\)
Test whether item is a string.

is\_struct \((self: \text{dicomifier.bruker.Field}, \text{index: int}) \rightarrow \text{bool}\)
Test whether item is a struct.

### 3.2 dicomifier.bruker\_to\_dicom

#### 3.2.1 dicomifier.bruker\_to\_dicom.convert

class \text{dicomifier.bruker_to_dicom.convert.ReconstructionContext} \((\text{path})\)
Add reconstruction context to logger.

filter \((\text{record})\)
Determine if the specified record is to be logged.

Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place.

dicomifier.bruker\_to\_dicom.convert\._convert\_directory \((\text{source, dicomdir, multiframe, writer})\)
Convert a Bruker directory to DICOM and write the files.

Parameters
• **source** – source directory
• **dicomdir** – whether to create a DICOMDIR
• **multiframe** – whether to create multi-frame DICOM objects
• **writer** – writer object from the io module

```python
dicomifier.bruker_to_dicom.convert.convert_element(bruker_data_set, dicom_data_set, bruker_name, dicom_name, type_, getter, frame_index, generator, vr_finder)
```

Convert a Bruker element to a DICOM element.

**Parameters**

- **bruker_data_set** – source Bruker data set
- **dicom_data_set** – destination DICOM data set
- **bruker_name** – Bruker name of the element
- **dicom_name** – DICOM name of the element
- **type** – DICOM type of the element (PS 3.5, 7.4)
- **getter** – function returning the value using the Bruker data set, the generator and the frame index or None (direct access)
- **frame_index** – index in frame group
- **generator** – FrameIndexGenerator associated with the Bruker data set
- **vr_finder** – function returning the DICOM VR from the dicom_name

```python
dicomifier.bruker_to_dicom.convert.convert_module(bruker_data_set, dicom_data_set, module, frame_index, generator, vr_finder)
```

Convert a DICOM module.

**Parameters**

- **bruker_data_set** – source Bruker data set
- **dicom_data_set** – destination DICOM data set
- **module** – sequence of 4-element tuples describing the conversions (Bruker name, DICOM name, DICOM type, getter)
- **frame_index** – index in a frame group
- **generator** – object that will manage the frame_index
- **vr_finder** – function to find the VR knowing only the dicom_name

```python
dicomifier.bruker_to_dicom.convert.convert_reconstruction(data_set, iod_converter, writer)
```

Convert and save a single reconstruction.

**Parameters**

- **iod_converter** – conversion function
- **writer** – writer object from the io module

```python
dicomifier.bruker_to_dicom.convert.to_2d(data_set)
```

Convert the Bruker data set from 3D to 2D.
3.2.2 dicomifier.bruker_to_dicom.io

dicomifier.bruker_to_dicom.io.create_dicomdir(paths, directory, patient_key, study_key, series_key, image_key)

Create a DICOMDIR from all files found in paths.

Parameters

- **paths** – list of DICOM files and directories storing DICOM files
- **directory** – destination directory of the DICOMDIR
- **patient_key** – patient-level keys to add to the DICOMDIR
- **study_key** – study-level keys to add to the DICOMDIR
- **series_key** – series-level keys to add to the DICOMDIR
- **image_key** – image-level keys to add to the DICOMDIR

dicomifier.bruker_to_dicom.io.get_bruker_info(directory)

Return basic information of a Bruker data set stored in the specified directory as a DICOM data set with the Patient, General Study, Patient Study, and General Series modules.

3.2.3 dicomifier.bruker_to_dicom.iods

dicomifier.bruker_to_dicom.iods.enhanced_mr_image_storage(bruker_data_set, transfer_syntax)

Convert bruker_data_set into dicom_data_set by using the correct transfer_syntax. This function will create one data_set per reconstruction (multiFrame format)

Parameters

- **bruker_data_set** – Bruker data set in dictionary form
- **transfer_syntax** – Wanted transfer syntax for the conversion

dicomifier.bruker_to_dicom.iods.mr_image_storage(bruker_data_set, transfer_syntax)

Function to convert specific burker images into dicom

Parameters

- **bruker_data_set** – bruker data set to convert
- **transfer_syntax** – target transfer syntax

3.3 dicomifier.dicom_to_nifti

3.3.1 dicomifier.dicom_to_nifti.convert

class dicomifier.dicom_to_nifti.convert.SeriesContext(data_set)

Add series context to logger.

filter(record)

Determine if the specified record is to be logged.

Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place.
dicomifier.dicom_to_nifti.convert.convert_paths(paths, destination, zip, dtype=None, extra_splitters=None)

Convert the DICOM files found in a collection of paths (files, directories, or DICOMDIR) and save the result in the given destination.

Parameters
- **paths** – Collection of paths to scan for DICOM files
- **destination** – Destination directory
- **zip** – whether to zip the NIfTI files
- **dtype** – if not None, force the dtype of the result image
- **extra_splitters** – additional splitters to be used when building stacks

dicomifier.dicom_to_nifti.convert.convert_series(series_files, dtype=None, finder=None, extra_splitters=None)

Return the NIfTI image and meta-data from the files containing a single series.

Parameters
- **dtype** – if not None, force the dtype of the result image
- **finder** – if not None, series finder object to overwrite the Series Instance UID
- **extra_splitters** – additional splitters to be used when building stacks

dicomifier.dicom_to_nifti.convert.convert_series_data_sets(data_sets, dtype=None, extra_splitters=None)

Convert a list of dicom data sets into NIfTI

Parameters
- **data_sets** – list of dicom data sets to convert
- **dtype** – if not None, force the dtype of the result image
- **extra_splitters** – additional splitters to be used when building stacks

dicomifier.dicom_to_nifti.convert.merge_images_and_meta_data(images_and_meta_data)

Merge the pixel and meta-data of geometrically coherent images.

### 3.3.2 dicomifier.dicom_to_nifti.image

dicomifier.dicom_to_nifti.image.find_element(data_set, frame_index, tag, sequence)

Return a data set, potentially nested inside the given data set and sequence, which contain the given tag.

dicomifier.dicom_to_nifti.image.get_geometry(stack)

Compute the geometry (origin, spacing, orientation) for the given stack.

dicomifier.dicom_to_nifti.image.get_image(stack, dtype, cache=None)

Get the NIfTI image of the given stack

Parameters
- **stack** – collection of data set and an associated frame number for multi-frame datasets
- **dtype** – if not None, force the dtype of the result image
- **cache** – optional cache of linear pixel data for multi-frame data sets

dicomifier.dicom_to_nifti.image.get_linear_pixel_data(data_set)

Return a linear numpy array containing the pixel data.
dicomifier.dicom_to_nifti.image.get_orientation(stack)
   Compute the orientation of the stack.

dicomifier.dicom_to_nifti.image.get_origin(stack)
   Compute the origin of the stack.

dicomifier.dicom_to_nifti.image.get_shaped_pixel_data(data_set, frame_index, linear_pixel_data)
   Return the pixel data located in a dataset (and possibly one of its frame) shaped according to number of rows,
columns and frames.

dicomifier.dicom_to_nifti.image.get_spacing(stack)
   Compute the spacing of the stack.

3.3.3 dicomifier.dicom_to_nifti.io

dicomifier.dicom_to_nifti.io.get_dicomdir_files(path)
   Return the list of files indexed in a DICOMDIR file.

dicomifier.dicom_to_nifti.io.get_files(paths)
   Return the DICOM files found in the paths. Each path can be a single file, a directory (scanned recursively), or
a DICOMDIR file.

dicomifier.dicom_to_nifti.io.get_series_directory(meta_data)
   Return the directory associated with the patient, study and series of the NIfTI meta-data.

dicomifier.dicom_to_nifti.io.write_nifti(nifti_data, destination, zip, series_directory=None)
   Write the NIfTI image and meta-data in the given destination.

   Parameters

   • nifti_data – Pair of NIfTI image and meta-data
   • destination – Destination directory
   • zip – whether to zip the NIfTI files
   • series_directory – if provided, override the automated series-based output directory name

3.3.4 dicomifier.dicom_to_nifti.meta_data

dicomifier.dicom_to_nifti.meta_data.convert_data_set(data_set, specific_character_set)
   Convert a DICOM data set to its NIfTI+JSON representation.

dicomifier.dicom_to_nifti.meta_data.convert_element(element, specific_character_set)
   Convert a DICOM element to its NIfTI+JSON representation: the “Value” (or “InlineBinary”) attribute of its
standard DICOM JSON representation.

dicomifier.dicom_to_nifti.meta_data.get_meta_data(stack, cache=None)
   Get the meta-data of the current stack

   will keep the priority order for repeating element with the following rules: low_priority = per_frame_seq
high_priority = top_level (data_set) (if the same element is present in both shared and per_frame, we will keep
the element of the shared seq)

   Parameters

   • stack – collection of data set and an associated frame number for multi-frame datasets
• cache – optional cache of meta-data for multi-frame data sets

dicom.dicom_to_nifti.meta_data.get_tag_name(tag)
Convert a DICOM tag to its NIfTi+JSON representation: the tag keyword if known, and the standard DICOM JSON representation otherwise.

3.3.5 dicom.dicom_to_nifti.series

dicom.dicom_to_nifti.series.split_series(files)
Split specified DICOM files in series.

3.3.6 dicom.dicom_to_nifti.siemens

dicom.dicom_to_nifti.siemens.parse_asconv(mr_phoenix_protocol)
Return the acquisition settings stored in the ASCCONV section of the MrPhoenixProtocol field stored in the CSA headers.
dicom.dicom_to_nifti.siemens.parse_csa(csa)
Return a dictionary of (tag, items) of the CSA.
dicom.dicom_to_nifti.siemens.parse_element(csa, start)
Return a pair (name, items), total_size
dicom.dicom_to_nifti.siemens.parse_item(csa, start)
Return a pair content, size
dicom.dicom_to_nifti.siemens.parse_protocol(data)
Parse (as a nested dictionary) the ASCII version of protocol data.

3.3.7 dicom.dicom_to_nifti.stacks

class dicom.dicom_to_nifti.stacks.OrientationGetter
Return the ideal orientation of a data set, i.e. allow small variations in the actual orientation.
dicom.dicom_to_nifti.stacks.frame_group_index_getter(data_set, tag)
Return bruker_to_dicom-specific frame group information.
dicom.dicom_to_nifti.stacks.ge_complex_image_component_getter(data_set, tag)
Return GE-specific Complex Image Component data.
dicom.dicom_to_nifti.stacks.ge_diffusion_getter(data_set, tag)
Return GE-specific diffusion data.
dicom.dicom_to_nifti.stacks.get_diffusion(data_set, tag)
Get b-value and gradient diffusion from the data_set.
dicom.dicom_to_nifti.stacks.get_dimension_index(data_set, tag, in_stack_position_index)
Return the dimension index pointer without InStackPosition in order to find the different volumes

Parameters in_stack_position_index – index of the In Stack Position element within the Dimension Index tuple
dicom.dicom_to_nifti.stacks.get_frame_position(data_set, frame_index)
Get the position of the specified frame.
dicomifier.dicom_to_nifti.stacks.get_in_stack_position_index(data_set)
    Return the position of In Stack Position element inside the Dimension Index.

dicomifier.dicom_to_nifti.stacks.get_stacks(data_sets, extra_splitters=None)
    Return the stacks contained in the data sets. The result is a dictionary in
    which the values are pairs of (data_set, frame_index) (in the case of single-frame
    data sets, frame_index is None), and in which the keys are tuples of
    selectors. In this context, a selector is defined as a pair of (group sequence,
    group, tag) (group sequence and group being None for single-frame data sets),
    and a value.

    Parameters
    • data_sets – list of dicom data sets
    • extra_splitters – additional splitters to be used when building stacks

dicomifier.dicom_to_nifti.stacks.siemens_coil_getter(data_set, tag)
    Return Siemens-specific coil identifier.

dicomifier.dicom_to_nifti.stacks.sort(key, frames)
    Sort the frames of a stack according to the items present in the stack key.

3.4 dicomifier.nifti

3.4.1 dicomifier.nifti.diffusion

Extract diffusion-related information from JSON meta-data and convert it to other formats. Unless otherwise specified, all b-values extracted from meta-data are expressed in \( s/m^2 \) (i.e. SI units).

dicomifier.nifti.diffusion.from_ge_private(data)
    Extract diffusion gradient direction and b-value from GE-specific elements
    (0019,xxbb, 0019,xxbc, 0019,xxbd, and 0043,xx39).


dicomifier.nifti.diffusion.from_siemens_csa(data)
    Extract diffusion gradient direction and b-value from Siemens-specific elements
    (CSA Image Header Info (0029,xx10)).

dicomifier.nifti.diffusion.from_standard(data)
    Extract diffusion gradient direction and b-value from standard DICOM elements
    (MR Diffusion Sequence).

dicomifier.nifti.diffusion.to_fsl(scheme, transform, bvecs_fd, bvals_fd)
    Save a diffusion scheme in FSL bvecs+bvals format. A reference transform is required as
    the bvecs are store in image coordinates, not in patient coordinates. This transform must
    correspond to an image-to-patient transform, e.g. what is stored in the affine member of
    nibabel images.

dicomifier.nifti.diffusion.to_mrtrix(scheme, fd)
    Save a diffusion scheme in MRtrix format to a file-like object.
For many other vendors, the diffusion-related meta data (b-values and direction of diffusion-encoding gradient) is barely specified in the official documentation, the two major problems being the unit in which the b-value is expressed and the coordinate system which defines the direction of the diffusion gradient. While the former is usually easy to determine (e.g. in neuro-imaging, a b-value of around 1000 $s/mm^2$ is customary), the latter is more complicated. An error in the frame of reference will remain unnoticed for some applications (e.g. a map of fractional anisotropy) but will yield radical changes in the output of direction-dependent metrics like tractography.

These documents attempt to describe how the diffusion-related meta data is stored in the files of various vendors.

## 4.1 Diffusion-related meta data in Bruker files

### 4.1.1 Data sets

We have two data sets which have been acquired to the intent of determining the precise semantics of the diffusion-related parameters stored in Bruker files. They both comprise acquisitions with similar field of views, but with different slice geometry, both in the slice direction (axial, coronal, sagittal or oblique) and in the direction of the readout gradient within the slice plane (e.g. left-right or antero-posterior for an axial slice). The usual abbreviations for the human anatomical directions (LR, AP, HF) will be used in this document; even though the terms “head” and “feet” might be confusing for animals and readily replaced with respectively “rostral” and “caudal”, we will stick with the Bruker nomenclature and use “head” and “feet”.

The first data set is a phantom composed of three pieces of a strand of climbing rope (courtesy of Lucas Soustelle, ICube, University of Strasbourg-CNRS), each piece angled slightly differently. Three different acquisitions were performed, one sagittal, one coronal and one axial.

<table>
<thead>
<tr>
<th>Series</th>
<th>Orientation</th>
<th>Readout</th>
<th>Phase</th>
<th>Slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Sagittal</td>
<td>HF</td>
<td>AP</td>
<td>LR</td>
</tr>
<tr>
<td>6</td>
<td>Coronal</td>
<td>HF</td>
<td>LR</td>
<td>AP</td>
</tr>
<tr>
<td>7</td>
<td>Axial</td>
<td>LR</td>
<td>AP</td>
<td>HF</td>
</tr>
</tbody>
</table>
The second data set is a rat head (courtesy of Chrystelle Po, ICube, University of Strasbourg-CNRS), with a total of seven acquisitions: two axial, two sagittal and two coronal, each set with two different readout directions, and an oblique slice prescription, close to axial.

<table>
<thead>
<tr>
<th>Series</th>
<th>Orientation</th>
<th>Readout</th>
<th>Phase</th>
<th>Slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Axial ()</td>
<td>LR</td>
<td>AP</td>
<td>HF</td>
</tr>
<tr>
<td>8</td>
<td>Axial ()</td>
<td>AP</td>
<td>LR</td>
<td>HF</td>
</tr>
<tr>
<td>9</td>
<td>Sagittal</td>
<td>HF</td>
<td>AP</td>
<td>LR</td>
</tr>
<tr>
<td>10</td>
<td>Sagittal</td>
<td>AP</td>
<td>HF</td>
<td>LR</td>
</tr>
<tr>
<td>11</td>
<td>Coronal</td>
<td>HF</td>
<td>LR</td>
<td>AP</td>
</tr>
<tr>
<td>12</td>
<td>Coronal</td>
<td>LR</td>
<td>HF</td>
<td>AP</td>
</tr>
<tr>
<td>13</td>
<td>Oblique (close to axial)</td>
<td>LR</td>
<td>AP</td>
<td>HF</td>
</tr>
</tbody>
</table>

All meta-data are loaded from the method files in each series directory.

```python
import glob
import os
import subprocess
import dicomifier
import nibabel
import numpy
import pandas
import tabulate

if not os.path.isdir("../../../tests/data/input"):
    subprocess.call(['../../../tests/download_data'])

root = "../../../tests/data"
bruker_paths = {
    "rope": os.path.join(root, "input", "20180818_175759_Rope_ChosenOne_1_2"),
    "rat": os.path.join(root, "input", "20171114_094354_Plateforme_R17_06_1_2")
}

series = {"rope": [5, 6, 7], "rat": [6, 8, 9, 10, 11, 12, 13]}

data = {}
for name in series:
    data[name] = {}
    for s in series[name]:
        data_set = dicomifier.bruker.Dataset()
        data_set.load(os.path.join(bruker_paths[name], str(s), "method"))
        data[name][s] = data_set
```

### 4.1.2 Units and coordinate systems

According to the official documentation of STB_InitDiffusionPreparation, PVM_DwDir and PVM_DwBvalEach are input parameters on which the user has control, while PVM_DwGradVec and PVM_DwEffBval are computed by Paravision.

The input parameters (PVM_DwDir and PVM_DwBvalEach) do not contain the actual diffusion scheme: the series number 5 of the rope data set has 13 volumes (including $b = 0$), but PVM_DwDir only contains 12 values, and the full value of PVM_DwBvalEach is $[1500]$ – obviously expressed in $s/mm^2$.

On the other hand, the output parameters contain the actual diffusion scheme, including the $b = 0$ acquisition. However, the $b$-value is the effective one (i.e. taking into account the diffusion brought by the imaging gradients), which
may confuse software expecting “exact” shells. Also note that the items in \texttt{PVM\_DwGradVec} are not unit length: they are gradient amplitude, relative to the maximum gradient amplitude allowed by the system. Even though the norm of \texttt{PVM\_DwDir} and \texttt{PVM\_DwGradVec} differ, they are colinear.

The parameters of series number 5 of the rope data set is summarized below.

<table>
<thead>
<tr>
<th>\text{DwEffBval}</th>
<th>\text{DwGradVec}</th>
<th>\text{Norm}</th>
<th>\text{DwDir}</th>
<th>\text{Norm}</th>
<th>Colinear?</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>[0.0, 0.0, 0.0]</td>
<td>0</td>
<td>[0.26, 0.07, 0.96]</td>
<td>nan</td>
<td>N/A</td>
</tr>
<tr>
<td>1512</td>
<td>[0.05, 0.01, 0.2]</td>
<td>0.21</td>
<td>[-0.43, 0.14, 0.89]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1507</td>
<td>[-0.09, 0.03, 0.19]</td>
<td>0.21</td>
<td>[-0.12, -0.56, 0.82]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1509</td>
<td>[-0.03, -0.12, 0.17]</td>
<td>0.21</td>
<td>[0.74, -0.27, 0.61]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1500</td>
<td>[0.08, 0.14, 0.13]</td>
<td>0.21</td>
<td>[0.4, 0.66, 0.64]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1500</td>
<td>[-0.08, 0.16, 0.11]</td>
<td>0.21</td>
<td>[-0.37, 0.75, 0.55]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1511</td>
<td>[-0.18, -0.02, 0.09]</td>
<td>0.21</td>
<td>[-0.89, -0.09, 0.45]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1500</td>
<td>[-0.13, -0.15, 0.07]</td>
<td>0.21</td>
<td>[-0.61, -0.72, 0.32]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1500</td>
<td>[0.08, -0.18, 0.08]</td>
<td>0.21</td>
<td>[0.37, -0.85, 0.37]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1500</td>
<td>[0.19, 0.07, 0.05]</td>
<td>0.21</td>
<td>[0.91, 0.32, 0.26]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1519</td>
<td>[0.03, 0.21, 0.02]</td>
<td>0.21</td>
<td>[0.13, 0.99, 0.08]</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>1511</td>
<td>[-0.18, 0.1, 0.02]</td>
<td>0.21</td>
<td>[-0.86, 0.5, 0.08]</td>
<td>1</td>
<td>True</td>
</tr>
</tbody>
</table>

Both for the rope and for the rat data set, the diffusion scheme is the same for all slice orientations. This, and the fact that \texttt{PVM\_DwGradVec} are gradient amplitudes used directly in the pulse program, indicates that the diffusion gradient directions are expressed in slice coordinates (i.e. \((1, 0, 0)\) is the readout axis, and \((0, 0, 1)\) is the slice-selection axis, or slice normal).

### 4.1.3 Conversion to subject coordinates

In order to convert this data to subject coordinates (among others, used in DICOM and MRtrix), we need the coordinates of the imaging gradients in subject coordinates. The documentation of \texttt{STB\_UpdateTraj} states that the parameter \texttt{PVM\_SPackArrGradOrient} contains those values:

\texttt{gradOrient}: Gradient orientation matrix transferring between RPS (slice) and XYZ (object) coordinate system (note: XYZ represents AP-LR-HF)

However, looking at the values of \texttt{PVM\_SPackArrGradOrient}, it seems that the XYZ coordinate system is instead the usual LR-AP-HF system. On the rope data set, the series 5, 6, and 7 are respectively sagittal, coronal and axial, and their respective gradient orientation matrices are:

- \textbf{5}: \[
\begin{pmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{pmatrix}
\]

- \textbf{6}: \[
\begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

- \textbf{7}: \[
\begin{pmatrix}
1.0 & -0.0 & -0.0 \\
0.0 & 1.0 & -0.0 \\
0.0 & 0.0 & 1.0 \\
\end{pmatrix}
\]

It also seems that the matrix is either the transpose of the aforementioned transform for column-vectors, or is designed to be used with row-vectors.

With those two remarks, it is possible to define the directions of the diffusion gradient in subject coordinates by multiplying the transpose of \texttt{PVM\_SPackArrGradOrient} by each normalized entry of\texttt{PVM\_DwGradVec}.

### 4.1. Diffusion-related meta data in Bruker files
4.1.4 Validation

We validate the transformation using a simple MRtrix-based pipeline:

1. Create a scheme file following MRtrix format
2. Estimate the diffusion tensors from the diffusion-weighted image and the scheme
3. Extract the first eigenvector from the tensor map

4.1.5 Qualitative

The following images show that the hue, indicating the principal direction of the direction tensor, is similar for the different series.
4.1.6 Quantitative

It is possible to go one step further and compute the angle between the principal eigenvector of a reference series (5 for the rope data set, 6, 9, and 11 for the rat data set) and the principal eigenvector of subsequent series (6 and 7 for the rope data set and respectively (8, 13), 10 and 12 for the rat data set). Due to some limitations in the data sets (one-dimensional object in the rope data set, high anisotropy of the voxels in the rat data set), we only focus on angles larger than 45 °; however, since the fields of view of the acquisitions are orthogonal to one another, any error would show as 90 ° angles between an image and its reference, and would appear at every voxel.

The following images show that, although there are large differences remaining, most of the object show similar principal directions, which validates the transform.
4.2 Diffusion-related meta data in GE files

As opposed to Siemens data, there seem to be no structured private DICOM field in GE data. Even though the Diffusion \textit{b-value} public field (0018,9087) sometimes appear at the top-level in DICOM files, this is not always the case and the direction of the diffusion gradient is neither at the top-level, nor in the standard \textit{MR Diffusion Sequence} field.

We can however run through all tags and identify those which are coherent within volumes and may correspond to either the b-value or the direction of the diffusion gradient. Using the diff-testing tools of Dicomifier on two arbitrary
data sets, this yields the following.

- **Diffusion b-value** (obviously)
- (0019,10bb), (0019,10bc), and (0019,10bd) within the \textit{GEMS\_ACQU\_01} private creator: possible match for the direction of the diffusion gradient
- (0043,1039), within the \textit{GEMS\_PARAM\_01} private creator: match of the \textit{Diffusion b-value} element, when present.

The b-matrix is either missing or encoded in a binary vendor-specific field.

To validate that the three fields mentioned above are the directions of the diffusion gradient, we can plot them to check that they are distributed on the surface of a sphere.

```python
import matplotlib.pyplot
from mpl_toolkits.mplot3d import Axes3D
import numpy
import odil

data_sets = [
    odil.Reader.read_file("./TEST\_DIFF\_GEDISCOVERY750w/a/z{}/".format(x)
        if x<100 else x)
    for x in range(1, 521)]
directions = set([
    tuple(d[tag][0] for tag in \"001910bb", \"001910bc", \"001910bd\")
    for d in data_sets])
origin = numpy.zeros((len(directions), 3))

figure = matplotlib.pyplot.figure(figsize=(8,8))
plot = figure.add_subplot(1, 1, 1, projection=\"3d\")
plot.set_proj_type(\"ortho\")

plot.quiver(*list(zip(*origin)), *list(zip(*directions)),
    arrow_length_ratio=0, normalize=True)
plot.scatter(*list(zip(*directions)))

    # draw sphere
    u, v = numpy.mgrid[0:2*numpy.pi:20j, 0:numpy.pi:10j]
x = numpy.cos(u)*numpy.sin(v)
y = numpy.sin(u)*numpy.sin(v)
z = numpy.cos(v)
plot.plot_wireframe(x, y, z, color=\"k\", lw=0.5, alpha=0.5)

plot.set_xlim(-1, 1)
plot.set_ylim(-1, 1)
plot.set_zlim(-1, 1)
plot.set_axis_off()
```

Fig. 1: Directions of diffusion gradient

4.2. Diffusion-related meta data in GE files
4.3 Diffusion-related meta data in Siemens files

**Warning:** This has been tested on Verio, Trio, Skyra and Prisma. Other models may or may not work.

**Warning:** The coordinate system of diffusion gradient direction has not yet been determined.

### 4.3.1 Private element

The diffusion-related meta-data is stored in a vendor-specific, binary-encoded, element, called *CSA Image Header Info*. The element tag is *(0029,xx19)*, under the Private Creator value *SIEMENS CSA HEADER*. The DICOM Conformance Statements (e.g. for the *Prisma*) only describe it as manufacturer model dependent information but its precise format can be found on various websites. A parser for this format has been implemented in Dicomifier.

### 4.3.2 Fields in the private element

Once parsed, the *CSA Image Header Info* has four fields obviously related to diffusion MRI: *B_value*, *DiffusionGradientDirection*, *DiffusionDirectionality*, and *B_matrix*, which match the terms of the MR Diffusion module. Two examples are shown below, the first one for a diffusion-weighted image, the second one for a \( b = 0 \) \( s/mm^2 \) image (other fields have been removed for readability).

```python
meta_data = [
    {
        'B_value': [0],
        'DiffusionGradientDirection': [],
        'B_matrix': [],
        'DiffusionDirectionality': [b'DIRECTIONAL\x00']
    },
    {
        'B_value': [1000],
        'DiffusionGradientDirection': [0.1977423, 0.17195807, -0.96505374],
        'B_matrix': [39.0, 34.0, -191.0, 30.0, -166.0, 932.0],
        'DiffusionDirectionality': [b'DIRECTIONAL\x00']
    },
    {
        'B_value': [1000],
        'DiffusionGradientDirection': [-0.32412839, -0.94072425, 0.09989204],
        'B_matrix': [105.0, 305.0, -32.0, 885.0, -94.0, 11.0],
        'DiffusionDirectionality': [b'DIRECTIONAL\x00']
    },
    {
        'B_value': [1000],
        'DiffusionGradientDirection': [-0.97839409, -0.17072155, 0.11661573],
        'B_matrix': [957.0, 167.0, -114.0, 29.0, -20.0, 14.0],
        'DiffusionDirectionality': [b'DIRECTIONAL\x00']
    }
]
```

The b-value is always present and expressed in \( s/mm^2 \). It appears to be the ideal b-value, i.e. not modulated by the imaging gradients. The direction of the diffusion gradient is not specified for non diffusion-weighted images, and neither is the b-matrix; when specified, the direction has norm 1.

The b-matrix has only six elements, and can thus be assumed to be the upper or the lower triangular part of the full b-matrix. However, looking at eigensystem of the b-matrix for both hypotheses, it appears that both the eigenvalues and the dot-product of the principal eigenvector with the direction of the diffusion gradient only match for the upper-triangular matrix.
import numpy

def from_upper(linear):
    return numpy.array([[linear[0], linear[1], linear[2]], [linear[1], linear[3], linear[4]], [linear[2], linear[4], linear[5]]])

def from_lower(linear):
    return numpy.array([[linear[0], linear[1], linear[3]], [linear[1], linear[2], linear[4]], [linear[3], linear[4], linear[5]]])

for item in meta_data:
    if not item['B_matrix']:
        continue

    upper_eigensystem = numpy.linalg.eigh(from_upper(item['B_matrix']))
    lower_eigensystem = numpy.linalg.eigh(from_lower(item['B_matrix']))

    direction = item['DiffusionGradientDirection']

    print("Upper triangular matrix: ", upper_eigensystem[0].astype(int),
          numpy.dot(direction, upper_eigensystem[1][:,-1]),
          "\nLower triangular matrix: ", lower_eigensystem[0].astype(int),
          numpy.dot(direction, lower_eigensystem[1][:,-1]),
          "\n")

<table>
<thead>
<tr>
<th>Upper triangular matrix:</th>
<th>Lower triangular matrix:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  0 1000] -1.0000001580676756</td>
<td>[-220 43 956] -0.9739920011521168</td>
</tr>
<tr>
<td>[ 0 1 1000] 0.999999873608694</td>
<td>[-916 31 969] 0.3232097831549061</td>
</tr>
<tr>
<td>[ 0 0 999] 1.0000000357323802</td>
<td>[-143 17 983] 0.9894968101548527</td>
</tr>
</tbody>
</table>

4.3. Diffusion-related meta data in Siemens files
CHAPTER 5

Indices and tables

• genindex
Python Module Index

d
dicomifier.bruker, 9
dicomifier.bruker_to_dicom.convert, 10
dicomifier.bruker_to_dicom.io, 12
dicomifier.dicom_to_nifti.convert, 12
dicomifier.dicom_to_nifti.image, 13
dicomifier.dicom_to_nifti.io, 14
dicomifier.dicom_to_nifti.meta_data, 14
dicomifier.dicom_to_nifti.series, 15
dicomifier.dicom_to_nifti.siemens, 15
dicomifier.dicom_to_nifti.stacks, 15
dicomifier.nifti.diffusion, 16
Symbols

-dicomdir, -d  
command line option, 6
-dtype <dtype>, -d <dtype>  
command line option, 7
-layout <layout>, -1 <layout>  
command line option, 6
-multiframe, -m  
command line option, 7
-transfer-syntax <syntax>, -t <syntax>  
command line option, 7
-zip, -z  
command line option, 7
-i <image>, -image <image>  
command line option, 8
-j, -json  
command line option, 5
-m, -match  
command line option, 6
-p, -pipe  
command line option, 6
-v <level>, -verbose <level>, 5-8  
[destination ...], 8
destination, 6, 7
format, 8
source, 5-8

convert_data_set() (in module dicomifier.dicom_to_nifti.meta_data), 14
convert_directory() (in module dicomifier.bruker_to_dicom.convert), 10
convert_element() (in module dicomifier.bruker_to_dicom.convert), 11
convert_element() (in module dicomifier.dicom_to_nifti.meta_data), 14
convert_module() (in module dicomifier.bruker_to_dicom.convert), 11
convert_paths() (in module dicomifier.dicom_to_nifti.convert), 12
convert_reconstruction() (in module dicomifier.bruker_to_dicom.convert), 11
convert_series() (in module dicomifier.dicom_to_nifti.convert), 13
convert_series_data_sets() (in module dicomifier.dicom_to_nifti.convert), 13
create_dicomdir() (in module dicomifier.bruker_to_dicom.io), 12

C

command line option
-dicomdir, -d, 6
-dtype <dtype>, -d <dtype>, 7
-layout <layout>, -l <layout>, 6
-multiframe, -m, 7
-transfer-syntax <syntax>, -t <syntax>, 7
-zip, -z, 7
-i <image>, -image <image>, 8
-j, -json, 5
-m, -match, 6
-p, -pipe, 6

D

Dataset (class in dicomifier.bruker), 9
destination
command line option, 6, 7
dicomifier.bruker (module), 9
dicomifier.bruker_to_dicom.convert (module), 10
dicomifier.bruker_to_dicom.io (module), 12
dicomifier.dicom_to_nifti.convert (module), 12
dicomifier.dicom_to_nifti.image (module), 13
dicomifier.dicom_to_nifti.io (module), 14
Dicomifier Documentation, Release 2.3.1

dicomifier.dicom_to_nifti.meta_data (module), 14
dicomifier.dicom_to_nifti.series (module), 15
dicomifier.dicom_to_nifti.siemens (module), 15
dicomifier.dicom_to_nifti.stacks (module), 15
dicomifier.nifti.diffusion (module), 16

E
enhanced_mr_image_storage() (in module dicomifier.bruker_to_dicom.iods), 12

F
Field (class in dicomifier.bruker), 9
Field.Item (class in dicomifier.bruker), 9
filter() (dicomifier.bruker_to_dicom.convert.ReconstructionContext method), 10
filter() (dicomifier.dicom_to_nifti.convert.SeriesContext method), 12
find_element() (in module dicomifier.dicom_to_nifti.image), 13
format
command line option, 8
frame_group_index_getter() (in module dicomifier.dicom_to_nifti.stacks), 15
from_ge_private() (in module dicomifier.nifti.diffusion), 16
from_siemens_csa() (in module dicomifier.nifti.diffusion), 16
from_standard() (in module dicomifier.nifti.diffusion), 16

G
ge_complex_image_component_getter() (in module dicomifier.dicom_to_nifti.stacks), 15
ge_diffusion_getter() (in module dicomifier.dicom_to_nifti.stacks), 15
get_bruker_info() (in module dicomifier.bruker_to_dicom.io), 12
get_dicomdir_files() (in module dicomifier.dicom_to_nifti.io), 14
get_diffusion() (in module dicomifier.dicom_to_nifti.stacks), 15
get_dimension_index() (in module dicomifier.dicom_to_nifti.stacks), 15
get_field() (dicomifier.bruker.Dataset method), 9
get_files() (in module dicomifier.dicom_to_nifti.io), 14
get_frame_position() (in module dicomifier.dicom_to_nifti.io), 15
get_geometry() (in module dicomifier.dicom_to_nifti.image), 13
get_image() (in module dicomifier.dicom_to_nifti.image), 13
get_in_stack_position_index() (in module dicomifier.dicom_to_nifti.stacks), 15
get_int() (dicomifier.bruker.Field method), 10
get_int() (dicomifier.bruker.Field.Item method), 9
get_linear_pixel_data() (in module dicomifier.dicom_to_nifti.image), 13
get_meta_data() (in module dicomifier.dicom_to_nifti.meta_data), 14
get_orientation() (in module dicomifier.dicom_to_nifti.image), 14
get_origin() (in module dicomifier.dicom_to_nifti.image), 14
get_real() (dicomifier.bruker.Field method), 10
get_real() (dicomifier.bruker.Field.Item method), 9
get_series_directory() (in module dicomifier.dicom_to_nifti.io), 14
get_shaped_pixel_data() (in module dicomifier.dicom_to_nifti.image), 14
get_spacing() (in module dicomifier.dicom_to_nifti.image), 14
get_stacks() (in module dicomifier.dicom_to_nifti.image), 14
get_string() (dicomifier.bruker.Field method), 10
get_string() (dicomifier.bruker.Field.Item method), 9
get_struct() (dicomifier.bruker.Dataset method), 9
get_used_files() (dicomifier.bruker.Dataset method), 9

H
has_field() (dicomifier.bruker.Dataset method), 9

I
is_int() (dicomifier.bruker.Field method), 10
is_int() (dicomifier.bruker.Field.Item method), 10
is_real() (dicomifier.bruker.Field method), 10
is_real() (dicomifier.bruker.Field.Item method), 10
is_string() (dicomifier.bruker.Dataset method), 10
is_string() (dicomifier.bruker.Field method), 10
is_string() (dicomifier.bruker.Field.Item method), 10
items() (dicomifier.bruker.Dataset method), 9

L
load() (dicomifier.bruker.Dataset method), 9
merge_images_and_meta_data() (in module dicomifier.dicom_to_nifti.convert), 13
mr_image_storage() (in module dicomifier.bruker_to_dicom.iods), 12

OrientationGetter (class in dicomifier.dicom_to_nifti.stacks), 15

parse_ascconv() (in module dicomifier.dicom_to_nifti.siemens), 15
parse_csa() (in module dicomifier.dicom_to_nifti.siemens), 15
parse_element() (in module dicomifier.dicom_to_nifti.siemens), 15
parse_item() (in module dicomifier.dicom_to_nifti.siemens), 15
parse_protocol() (in module dicomifier.dicom_to_nifti.siemens), 15

ReconstructionContext (class in dicomifier.bruker_to_dicom.convert), 10

SeriesContext (class in dicomifier.dicom_to_nifti.convert), 12
siemens_coil_getter() (in module dicomifier.dicom_to_nifti.stacks), 16
sort() (in module dicomifier.dicom_to_nifti.stacks), 16
command line option, 5–8
split_series() (in module dicomifier.dicom_to_nifti.series), 15

to_2d() (in module dicomifier.bruker_to_dicom.convert), 11
to_fsl() (in module dicomifier.nifti.diffusion), 16
to_mrtrix() (in module dicomifier.nifti.diffusion), 16

write_nifti() (in module dicomifier.dicom_to_nifti.io), 14