TexTOM - Manual

Moritz Frewein, Moritz Stammer, Marc Allain, Tilman Grünewald

July 22, 2026

Contents

1 Introduction **4**
1.1 Texture Tomography . . . 4
1.2 Installation . . . 4
1.2.1 GPU drivers . . . 4
1.2.2 Start . . . 5

2 Configuration **6**

3 Handling of the TexTOM software **7**

4 Workflow **8**
4.1 Data acquisition . . . 8
4.2 Data integration . . . 9
4.3 Alignment . . . 10
4.4 Model . . . 12
4.5 Data Pre-processing . . . 14
4.6 Optimization . . . 15
4.7 Analysis . . . 15
4.8 Visualization . . . 16

5 Other Advices **16**
5.1 Running TexTOM via a SSH connection . . . 16

6 Troubleshooting **16**
6.1 I cannot write into the command line . . . 16

7 Functions **17**
set_path . . . 17
check_state . . . 17
integration_test . . . 17
integrate . . . 17
generate_sample . . . 18
list_projections . . . 18

mask_detector_azimuthal . . . 18
check_geometry . . . 19
show_projection . . . 19
align_data . . . 19
check_alignment_consistency . . . 20
check_alignment_projection . . . 21
reconstruct_1d_full . . . 21
check_powder_pattern . . . 21
make_model . . . 22
show_sample_outline . . . 22
mask_peak_regions . . . 22
mask_baseline_regions . . . 23
check_baselines . . . 23
preprocess_data . . . 23
make_fit . . . 24
toggle_peak_fit . . . 24
optimize . . . 25
list_opt . . . 25
load_opt . . . 26
check_lossfunction . . . 26
check_fit_average . . . 26
check_fit_per_index . . . 26
check_fit_random . . . 27
check_residuals . . . 27
check_projections_average . . . 27
check_projections_residuals . . . 27
check_projections_orientations . . . 28
calculate_orientation_statistics . . . 28
calculate_order_parameters . . . 28
calculate_segments . . . 28
show_volume . . . 29
show_slice_ipf . . . 29
show_slice_directions . . . 30
show_volume_ipf . . . 30
show_histogram . . . 30
show_correlations . . . 31
show_voxel_odf . . . 31
show_voxel_polefigure . . . 32
show_subvolume_average_odf . . . 32
show_subvolume_average_polefigure . . . 33
save_results . . . 34
export_paraview . . . 34
list_results . . . 34
load_results . . . 34
list_results_loaded . . . 35
save_images . . . 35

help 35

1 Introduction

1.1 Texture Tomography

Texture tomography is a way of inverting tomographic X-ray diffraction data into local orientation distribution functions (ODF) of diffracting crystallites. It relies on a priori-knowledge of the crystal structure and from there models diffraction patterns for comparison with the . For parameter optimization it refines the coefficients of harmonic basis functions constructing the ODF. This approach is particularly suited for polycrystalline materials with relatively wide orientation distributions, such as biomineralized tissue.

For a detailed description of mathematical model and the experimental procedure refer to Frewein, M. P. K., Mason, J., Maier, B., Colfen, H., Medjahed, A., Burghammer, M., Allain, M. & Grünewald, T. A. (2024). IUCrJ, 11, 809-820. https://doi.org/10.1107/S2052252524006547 and references therein.

1.2 Installation

TexTOM was written and tested in Python 3.11 and in principle requires only a python installation (3.9 to 3.12) and a terminal. It is conceived to be used in iPython through a terminal, but can be imported into scripts or jupyter notebooks.

The TexTOM core for reconstructions currently depends on external packages such as Scipy, Numba, H5py, Orix, pyFAI and Mumott. We experienced issues with parallelisation due to multiple BLAS installations by Numpy and Scipy. We therefore recommend properly setting up a single openblas installation as indicated below.

We recommend creating conda environment and installing the package via pip. Install Anaconda or Miniconda (https://docs.anaconda.com/miniconda/install/)

```
conda create --name textom python=3.11
conda activate textom
conda install numpy scipy openblas
```

then

```
pip install textom
```

We encountered crashes during visualization using matplotlib using the tkinter backend, therefore we recommend using the qt backend if possible. It is available by installing e.g.

```
pip install pyqt5
```

1.2.1 GPU drivers

Two of the packages (pyFAI and Mumott) provide GPU support for their functionalities. These require additional drivers such as Cudatoolkit for Nvidia graphics cards, which can be installed via

```
conda install cudatoolkit
```

for alignment via Mumott and

```
pip install pyopencl
```

for integration via pyFAI. Please refer to the documentations of the respective packages and your hardware to find out what drivers are required. In case no drivers are found, the software will fall back to computation via CPU.

1.2.2 Start

To start TexTOM in iPython mode, make sure your environment is active and type `textom`. All TexTOM core functions (sec 7) will be available in the namespace.

You can also import them into a script or jupyter notebook:

```
from textom.textom import *
```

Note that textom currently does not support graphical input via jupyter notebooks!

TexTOM Source code is available on: `https://gitlab.fresnel.fr/textom/textom/`.

2 Configuration

After installing or updating TexTOM, we recommend opening the configuration file primarily to set how many CPUs your machine has for data processing. Type `textom_config` in your terminal and it will open the config file in your standard text editor. A standard config file will look like the following:

```
import numpy as np # don't delete this line
################################################

# Define how many cores you want to use
# If this number is higher than the available cores on your system,
# it will fall back to the maximum number
n_threads = 128

# Choose if you want to use a GPU for alignment
use_gpu = False
# needs cudatoolkit: pip install cudatoolkit

# Choose your precision
# recommended np.float64 for double or np.float32 for single precision
# this mainly concerns data handling, critical parts of the code always use double precision
data_type = np.float32

# turn on wise phrases at the start of TexTOM
fun_mode = False
```

If `n_threads` is larger than the available number, it will fall back to the maximum number of threads/cores available. After making your changes, you can save the file and close it.

3 Handling of the TexTOM software

TexTOM is conceived as a command line software in iPython. Its high-level library (section 7) is aimed to be usable without advanced knowledge in python programming. Part of its user-interface consist of files created in the sample directory. In this directory TexTOM organises intermediate results automatically. Any part of the analysis can therefore be revisited and retraced. Upon startup, TexTOM assumes that the sample directory is the one where the program is started, so the recommended way is to type

```
cd /path/to/my/sample/directory/
```

prior to starting TexTOM via the command line. Alternatively, you can set the sample directory globally via the command `set_path('path')` after starting or importing TexTOM.

The following chart shows the structure of the sample directory and its subdirectories (red). It is recommended to start the analysis in an empty directory, the subdirectories will be created automatically.

Blue files are .h5 data containers, created during the workflow. For compatibility it is not recommended to create or modify these other than through the TexTOM pipeline.

White files are human-legible text files, that can be created or modified using a text editor or a custom script. They will be created through user input during the execution of the function in the main line in the graphic. If a .py or .txt file is present in the directory prior to calling the corresponding function, the present file will be used instead of asking for user input. This is handy for analysing a series of samples that share experimental parameters.

Green files are images for direct usage or export into other software for further analysis or visualization, such as Paraview, Avizo, Dragonfly or standard image viewers.

The functions on the right hand side are printed in the order of a suggested workflow, as the arrows indicate. There is some freedom in the order of doing these steps, as long as the requirements as shown by the red arrows are respected. The state of the analysis can be checked either by manually inspecting the directory or through the function `check_state()`.

A help-menu is available directly in the command line. Typing `help('function_name')` will produce the docstring with instructions on the usage of the respective function.

Relevant functions: set_path('path'), check_state(), help('function_name')

ToC

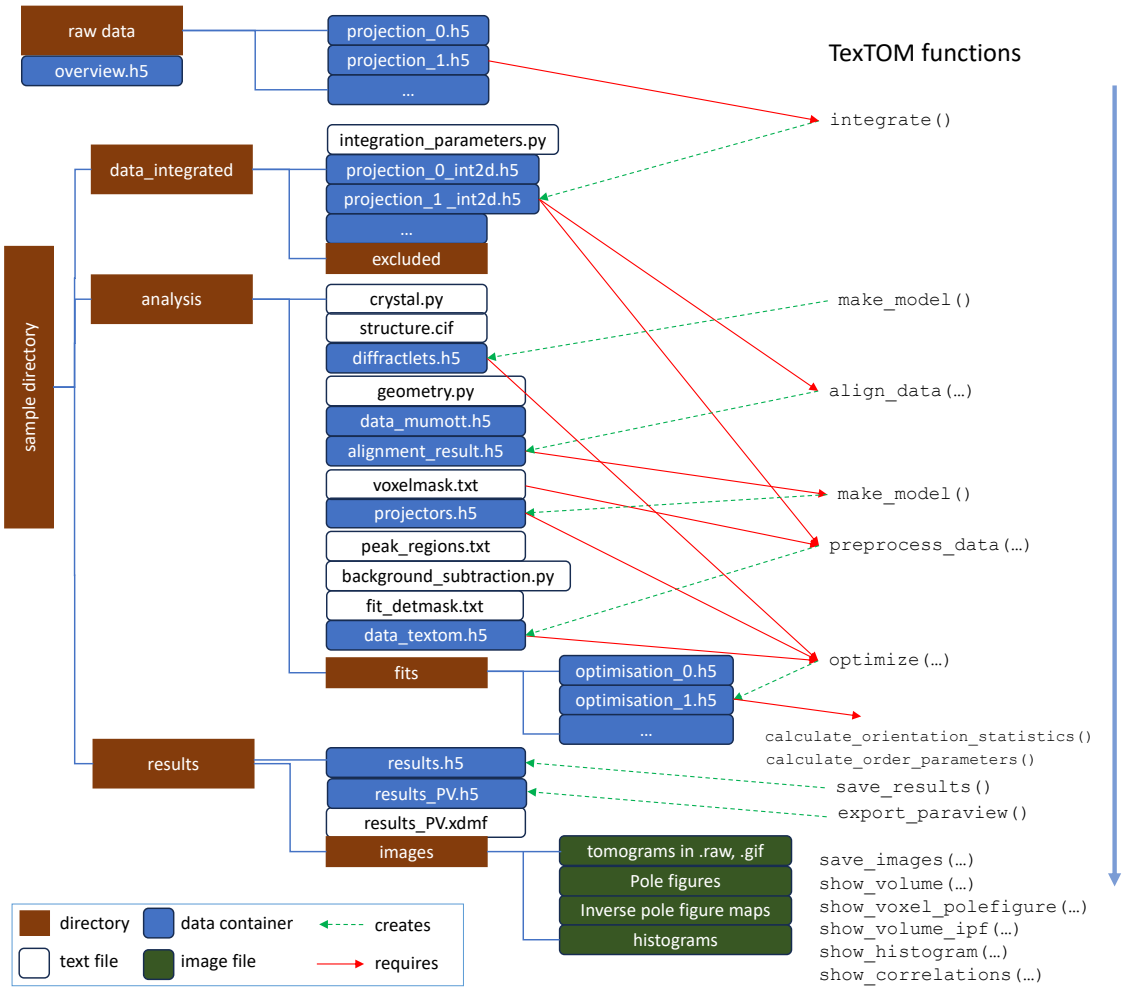


Figure 1: Structure of the sample directory and relevant functions.

4 Workflow

4.1 Data acquisition

Recording data for texture tomography is a great challenge and can only be done at appropriate synchrotron beamlines. This package contains a few scripts for the experiments but we recommend contacting a beamline scientist experienced in tensor/texture tomography or 3D-XRD in order to create acquisition scripts suitable for the beamline.

Make sure to collect all necessary metadata for the analysis and store them together with the data in container files such as `.h5`.

4.2 Data integration

The first step in data processing is integration, i.e. azimuthal rebinning ("caking") of the 2D-cartesian detector images. Here we rely on the pyFAI package (`https://pyfai.readthedocs.io`) but in principle other packages can be used as well, if a similarly structured output h5 file is created. This part already requires good knowledge of your data, as you do not want to miss any diffraction peaks when choosing the integration range. We recommend to do a test-integration during the experiment, to set up the correct .poni-file which is needed for the integration. This file defines the geometry of the experiment and can be created using the command pyFAI-calib2. Make sure to also collect the correct detector mask and optionally files for flatfield and darkcurrent correction if your detector requires them.

To start the integration, in your terminal navigate to a directory which will further contain all TexTOM analysis data (further labelled `sample_dir`).

```
cd /path/to/textom/sample_dir
```

Then start TexTOM by typing textom in your terminal. You can start the integration using the command `integrate()`, upon which a file containing all necessary parameters will open:

```
# Data path and names
path_in = 'path/to/your/experiment/overview_file.h5' # .h5 file with links to the data
h5_proj_pattern = 'mysample*.1' # projection names, * is a placeholder
# .h5 internal paths:
h5_data_path = 'measurement/eiger'
h5_tilt_angle_path = 'instrument/positioners/tilt' # tilt angle
h5_rot_angle_path = 'instrument/positioners/rot' # rotation angle
h5_ty_path = 'measurement/dty' # horizontal position
h5_tz_path = 'measurement/dtz' # vertical position
h5_nfast_path = 'technique/dim0' # fast axis number of points
h5_nslow_path = 'technique/dim1' # slow axis number of points
h5_ion_path = 'measurement/ion' # photon counter if present else None

# Parameters for pyFAI azimuthal integration
rad_range = [0.01, 37] # radial range
rad_unit = 'q_nm^-1' # radial parameter and unit ('q_nm^-1', ''2th_deg', etc)
azi_range = [-180, 180] # azimuthal range in degree
npt_rad = 100 # number of points radial direction
npt_azi = 120 # number of points azimuthal direction
npt_rad_1D = 2000 # number of points radial direction
int_method=('bbox','csr','cython') # pyFAI integration methods
    # for GPU change 'cython' to 'opencl'
poni_path = 'path/to/your/poni_file.poni'
mask_path = 'path/to/your/mask.edf'
polarisation_factor= 0.95 # polarisation factor, usually 0.95 or 0.99
solidangle_correction = True
flatfield_correction = None #or /path/to/file
```

```
darkcurrent_correction = None #or /path/to/file

# Integration mode
mode = 2 # 1: 1D, 2: 2D, 3: both

# Parallelisation
n_tasks = 8 # number of integrations performed in parallel
cores_per_task = 16 # size of the cluster that performs a single integration
# set both values to 1 if GPU is used
```

The first part contains information about your data. We assume that these are stored in `.h5` files as common practice at the ESRF. The first line is the overview file that contains links to all datasets. In the second line you can specify which files should be integrated using a pattern with a * serving as a placeholder for other characters. In the following there are the `.h5` internal paths to the necessary metadata for TexTOM, which will be carried into the integrated files. `h5_nfast_path` and `h5_nslow_path` are only relevant if the experiment was performed in scanning mode, upon which all data of one projection will be in the same data array with the horizontal and vertical position not specified. If the experiment was performed in continuous rotation (controt) mode, these parameters should be set to None. The last parameter is optional for the measurement of an ionisation chamber or diode, which records the incoming photon flux during the respective measurement.

Then choose the integration mode, 2D is required for TexTOM, 1D can be done additionally e.g. for diffraction tomography.

In the next block declare on how many CPUs you want to work parallely, the `n_tasks` specifies how many files will be integrated at the same time, `cores_per_task` means how many CPUs work on each task.

The last block are parameters for pyFAI, of particular importance are the radial range, which should cover your peaks and the number of points (npt_rad), which should be enough to resolve the individual peaks (although the code will also handle overlapping peaks or peaks which are in a single bin to the cost of some information loss due to their averaging). The required angular resolution depends on the sharpness of the features in the data in azimuthal direction, keep in mind that it is recommended to use a similar angular resolution for the construction of orientation distribution functions and diffractlets, where the computation time will scale with the power of 3 of the number of angular sampling points `npt_azi`. Furthermore, point to the data files you have recorded during your beam time and specify angular resolution etc. File paths should be complete paths and don't need to be in the sample directory, nor need to be accessible during the following steps.

Relevant functions: integrate()

ToC

4.3 Alignment

The first step of the alignment is the sorting of the data.

Go to the `data_integrated/` or `data_integrated_1d/` directory created by the integration script and make sure that all .h5 files are valid datasets, which you want to use for the reconstruction (other file extensions will be ignored). Move files that you don't want to use to a subfolder (e.g. named excluded). The program uses all data in the `sub_data directory` with pattern in

the filename. By default it uses data in `data_integrated/`, you can use others by typing e.g. `align_data(sub_data='data_integrated_1d')`

Next, choose the q-range you want to use for alignment. You can use array indices to select a range using the `q_index_range parameter` or give a `q-range` directly in the units specified in the `radial_units` field in the data (this parameter has priority if specified). TexTOM will average over all data in this range and treat them as scalar tomographic data for alignment. We recommend using either the SAXS region of the sample or an intense peak with little azimuthal variation.

TexTOM uses the alignment code from the Mumott tensor tomography package, which contains 2 pipelines. By default we use the optical flow alignment, but you can choose phase matching alignment in the parameters. If you want to crop the projections, set the `crop_image` parameter to the desired borders (e.g. ((0,-1),(10,-10)) for the full image in x-direction, while cropping 10 points at the top and bottom) Take note that cropping only works with the phase matching alignment, which will be chosen automatically if crop_image is defined.

The TexTOM alignment pipeline will downsample the data until arriving at the sampling defined by `regroup_max`, by default 16, corresponding to a downsampling to blocks of 16x16 pixels. Then the alignment will start at the lowest sampling, take the found values and proceed to the next highest until it reaches the original sampling. This approach has proven efficient even for large samples, but can be omitted by setting `regroup_max=1`. For the remaining parameters see the description below.

When you start the alignment using align_data(...), it will open a file labelled geometry.py, which contains information about the experimental setup. Most parameters are equivalent to the Mumott notation (`https://mumott.org/tutorials/inspect_data.html#Geometry`), which defines the arrangement of sample, detector, rotation and tilt angles.

```
### base coordinate system: ###
beam_direction = (1,0,0) # p in mumott
transverse_horizontal = (0,1,0) # j in mumott
transverse_vertical = (0,0,1) # k in mumott
################################ don't change

# detector geometry:
flip_detector_ud=False
flip_detector_lr=False
detector_direction_origin = (0,-1,0) # this conforms to standard pyFAI output
detector_direction_positive_90 = (0,0,-1) # this conforms to standard pyFAI output

# sample movements:
inner_axis = (0,0,1) # inner rotation axis
outer_axis = (0,1,0) # outer rotation axis
scan_mode = 'line' #'column' # 'line_snake' # 'column_snake'
flip_fov=False # flip fast and slow axis argument (for the case it was defined the reverse way i

# For calculating projectors:
Dbeam = 0.3 # beam size in um (FWHM)
Dstep = 0.5 # scanning step size in um
```

When you close and save the file, it will be automatically stored in `sample_dir/analysis/`

geometry.py and in the following, this file will be used. You can also create a geometry file in sample_dir/analysis/ prior to starting the alignment, then this file will directly be used (e.g. when you have several samples from the same beamtime, copy the geometry file after defining it for the first sample.). The default values are given for the configuration published in Frewein et al. IUCRJ (2024), an experiment carried out at the ESRF, ID13 EH3 nanobeam instrument.

After aligning, the function will create the file analysis/alignment_result.h5 in the sample directory, which contains the shifts found in the process. Refer to this file for checking sinograms and tomograms after alignment. You can also use the function check_alignment_consistency() to check if there are projections which deviate from the model. Inspect them and their agreement with the data using check_alignment_projection(g), where g is an integer number corresponding to the projection number. This number g is assigned after sorting the data files alphabetically. The x-axis label in the plot shown by check_alignment_consistency() uses the same labelling.

If you choose to add, remove or change data or changing the q-range after doing an alignment, redo the alignment with the setting redo_import=True. Else it will use the changes you made. If you just want to change the number of integrations or the regrouping, this is not necessary.

Relevant functions: align_data(...), check_alignment_consistency(), check_alignment_projection(g)

ToC

4.4 Model

Next you have to calculate the model, which consists of 2 parts: Diffractlets and Projectors.

Diffractlets are calculated from the crystal structure given by a .cif file, you have to provide. When you start the model calculation using make_model(), you will receive another file to edit (crystal.py), containing information about the location of your .cif file, X-ray energy, q-range and desired angular resolution. Save the file and it will be copied to sample_dir/analysis/crystal.py. The function will create the file diffractlets_hsh.h5, containing the diffractlets. If sample_dir/analysis/ contains already a diffractlets_hsh.h5 file, it will use this without asking.

If you start calculating diffractlets, a window will appear with a simulated powder pattern and the integrated intensity over one projection as comparison. Check if the powder pattern corresponds to your data and adapt the cif file accordingly if it is not the case. In particular, it might be necessary to adapt lattice parameters in case of a heavily strained sample. You can adapt some parameters in the crystal.py file for better visibility and to adapt the powder pattern: q_range lets you cut away the SAXS region, which might give you a too high y-limit in the figure. Increasing cutoff_structure_factor allows you to exclude low intensity peaks. You might need to adapt max_hkl if you expect peaks with higher Miller indices than the given number.

Note that all visible peaks in this window will be calculated as diffractlets. Later, you will be able to choose which ones will actually be used for optimization.

Once you adapted the parameters, you can close the figure and state that you're not happy and it will replot the window.

```
import numpy as np
## Define diffraction-related parameters:
# x-ray energy in keV
E_keV = 15.2
# q range for fitting: (lower,upper) boundary in nm^-1
```

```
q_range = (10,35)
# path to crystal cif file
cifPath = 'BaCO3.cif'
# parameters for diffractlet calculation
cutoff_structure_factor=1e-4
max_hkl=4

odf_mode = 'hsh' # 'grid' #
grid_resolution = 15 # degree # ignored if hsh
hsh_max_order = 12 # ignored if gridbased
```

The projectors contain information on which voxels contribute to which pixel in the data and are thus depend on a finished alignment. Once you finished the alignment you can start calculating the projectors, which requires some more user input for masking the sample. The program will open a histogram of voxels based on the tomogram resulting from alignment. Choose the lower cutoff to mask out voxels with low or zero density of crystallites, upon which the resulting object will automatically be smoothed and holes inside the structure will be filed. You will be shown a 3D outline of the sample.

After processing, this will create a file `analysis/projectors.h5`, which is used in further processing of this specific sample.

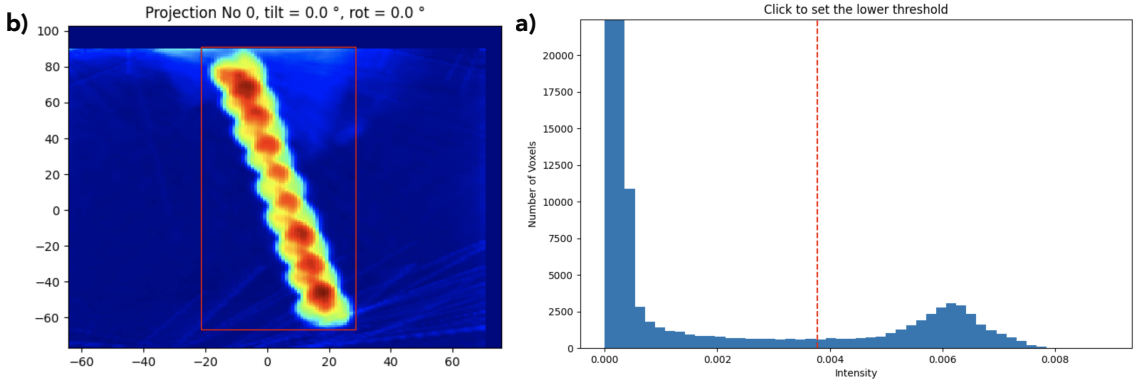


Figure 2: Textom input for masking during calculation of projectors. a) choose the smallest region that surrounds your sample. b) choose the threshold in the tomogram below which you only expect background.

Note that choosing a threshold and thereby masking voxels can lead to artifacts, especially on the surface and in regions with low crystalline material. It can be advantageous to choose a low threshold a the cost of longer processing times.

Relevant functions: make_model() check_powder_pattern() show_sample_outline() list_sample_rotations()

ToC

4.5 Data Pre-processing

When the model is ready, the data has to pass through a pre-processing step preprocess_data(...), where it is rebinned for each individual peak and outliers are removed. You will be asked to choose the q-ranges around the peaks you would like to use for optimization, and to define the detector mask. Text files will be created, these can be re-used for other samples and will be automatically chosen if present in the `analysis/` directory.

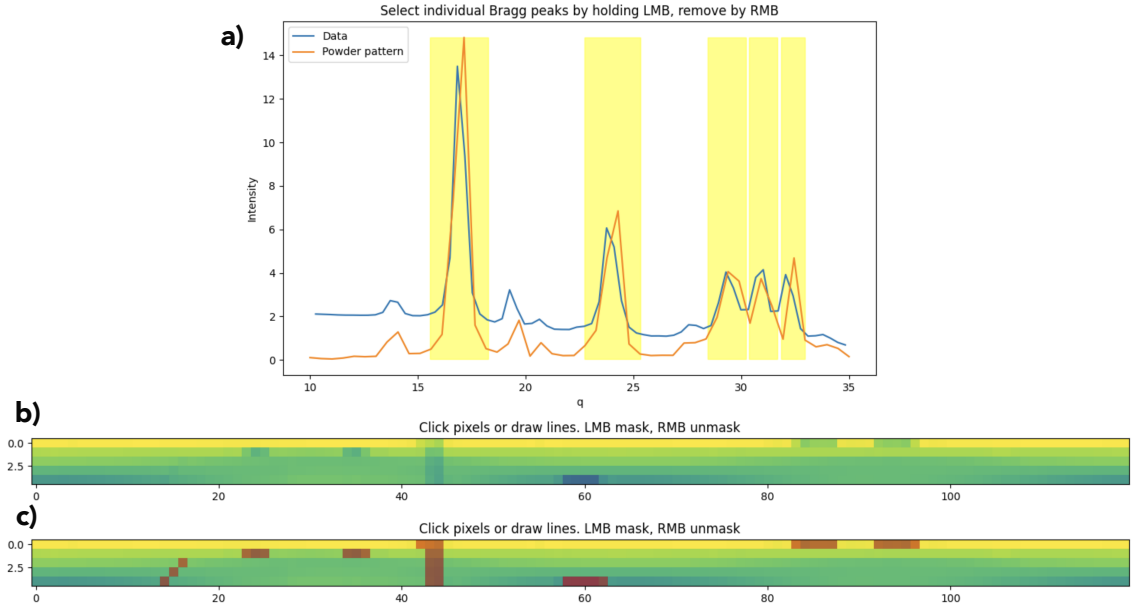


Figure 3: Textom input for choosing q-regions and masking pixels in the integrated images. a) Averaged data and calculated powder pattern with 5 peak regions marked for processing. b) Averaged data, azimuthally resoled. c) Data as before with pixels masked (red)

Also background subtraction is crucial. We provide several baseline subtraction schemes:

`'linear'` draws a straight line below each of the chosen peaks, from the last unmasked point to the next one. In case of overlapping peaks, it will draw the line for all overlapping ones together, if all are chosen for the refinement.

`'chebyshev'` draws a Chebyshev polynomial baseline, fitting all data that are not masked as a peak. This mode fails in some cases if not all peaks are chosen. In this case, one can use `'chebyshev_auto'`, which automatically masks peaks and draws the baseline. For these modes, one might have to adapt the parameters in the `background_subtraction.py` file to get the best results.

If no background subtraction is required, `'none'` is also an option.

To choose the appropriate background model, the program provides a selection of fits in a window before processing the data. Upon inspecting the baselines, you can adapt the parameters in `background_subtraction.py` and state that you are not happy, to replot with the new parameters.

Relevant functions: preprocess_data() check_baselines() mask_peak_regions() mask_detector_pixels()

ToC

4.6 Optimization

If all previous steps have been performed, you can start an optimization. The basic function that starts a TexTOM optimization is simply called optimize() and performs a gradient-based optimization of the ODF parameters in each voxel. It will save a `.h5` file with the found parameters and metadata on the optimization in the directory `analysis/fits/`. Already performed optimizations can be loaded via load_opt(...). An optimization can be stopped via `ctrl+c` anytime and will save the last values.

Relevant functions: optimize(),
list_opt(), load_opt(...),
check_lossfunction(), check_fit_average(), check_fit_random(...), check_residuals(),
check_projection_average(), check_projection_residuals(), check_projection_orientations()

ToC

4.7 Analysis

Upon obtaining a fit, you can calculate_orientation_statistics(), which will fill the preferred orientation (`g_pref`, the orientation in axis-angle parameters; `a_pref/b_pref/c_pref` are the corresponding unit cell directional vectors) and standard deviation (`std`) per voxel into a global `results` dictionary. It will also contain the (`scaling`) parameter, which corresponds to the amount of crystalline material in the voxel. You can check its current content via list_results_loaded(). There is also a simple segmentation algorithm calculate_segments(...), which calculates the misorientation between neighboring voxels and segments on this base. The misorientation (`mori`) and indices of the segments will be saved into results.

If your sample obeys a fibre texture or similar, with one of the axes aligned along a certain direction, you can calculate nematic order parameters characterize them using `calculate_order_parameters(axis=(0,0,1))` Nematic directors and order parameters will be added to results.

Using the function save_results() is necessary to save them to the hard drive. They can later be inspected list_results() and reloaded load_results(...) for visualization.

Note that arrays are stored in row-major (C-order) memory layout, so that axes have to be permuted when importing e.g. into Matlab.

There is also the possibility to export a file that is compatible with the fortran-style geometry used in paraview, using the command `export_paraview()`. The resulting `.xdmf`-file can be opened with paraview using the "XDMF Reader" (just drag and drop the file into paraview and choose this one).

Relevant functions: calculate_orientation_statistics(), calculate_order_parameters(axis=(0,0,1)), calculate_segments(...),
save_results(), list_results(), list_results_loaded(), load_results(...) export_paraview()

ToC

4.8 Visualization

The TexTOM package also contains some basic tool to visualize texture, in particular one can show tomograms of all scalar quantities using show_volume('scalar',...). This function gives the possibility inspect local ODFs upon clicking on a voxel.

Preferred orientations can be analogously visualized via inverse polefigures show_volume_ipf(...)

To show pole figures show_voxel_polefigure(x,y,z,(h,k,l)), it is necessary to know the indices of the desired voxels, to be found out via the former functions. You also have to provide the Miller indices as an argument.

Refer to the documentation of the individual functions for saving and further processing.

Relevant functions: show_volume('scalar',...), show_volume_ipf(...), show_slice_ipf(...), show_voxel_odf(...), show_voxel_polefigure(x,y,z,(h,k,l)), show_histogram(...), show_correlations(...), save_images(...)

5 Other Advices

5.1 Running TexTOM via a SSH connection

You can use TexTOM through an SSH connection either with a script or via the iPython mode. Input will be requested via text files and matplotlib figures.

To enable editing text files, a terminal based editor (e.g. Vim) will be used. Familiarize yourself with its handling.

Make sure you enable graphical output by running

```
ssh -Y profile@server
```

6 Troubleshooting

6.1 I cannot write into the command line

Sometimes a figure blocks the command line and will request more input after it is closed. Try closing all figures to continue.

Never close any figures if a calculation is running, as it might lead to a crash of the program!

ToC

7 Functions

set_path(path)

```
Set the path where integrated data and analysis is stored

Parameters
----------
path : str
    full path to the directory, must contain a folder '/data_integrated'
```

ToC

---

check_state()

```
Prints in terminal which parts of the reconstruction are ready
```

ToC

---

integration_test(dset_no=0)

```
Set up integration parameters, perform a 2D and 1D integration and plot
    them

Parameters
----------
dset_no : int, optional
    allows to choose a different file from the raw data folder
    (they will be assigned indices in alphabetic order), by default 0
```

ToC

---

integrate(parallel=True, wait=5.0, confirm=True, ignore=[], parallel_gpu=False)

```
Integrates raw 2D diffraction data via pyFAI

All necessary input will be handled via the file integration_parameters.py

Parameters
```

```
----------
mode : str
    'online' does one by one, updates filelist after each integration
    'parallel' loads filelist when started, can be parallelized over CPUs
wait : float
    waits this number of seconds after integrating the last file before
    stopping, by default 5.
confirm : bool
    if True, will open integration_parameters.py and ask for confirmation
    else, takes current file and starts
ignore : list
    here you can provide a list of datasets you want to skip (if they crash
    they will automatically be skipped)
```

ToC

---

generate_sample()

```
Allows the creation of custom samples for testing the pipeline.
Input of parameters occurs via a custom generation.py file, which opens
    automatically,
in addition to usual textom input files.
```

ToC

---

list_projections(datafile_pattern='.h5')

```
Gives a list in the terminal of all projections present in the model, with
    rotation and tilt angle.
Useful to choose a certain projection for other functions using index.
```

ToC

---

mask_detector_azimuthal(projection=0, datafile_pattern='.h5')

```
Opens a window that allows excluding detector pixels by mouse clicks
Gives the average of a whole projection to find regions to mask.

Parameters
----------
projection : int, optional
```

```
    projection index, by default 0
datafile_pattern : str, optional
    allows use a certain file that contains the pattern, by default '.h5'
```

ToC

---

check_geometry(testfile_path, testfile_h5path, dset_no=0, vmin=1, vmax=10, logscale=False)

```
No docstring available.
```

ToC

---

show_projection(projection=0, datafile_pattern='.h5', q_range=None)

```
Shows a whole projection as average where one can click on a pixel to show
    the whole diffraction pattern

Parameters
----------
projection : int, optional
    projection index, by default 0
```

ToC

---

align_data(pattern='.h5', sub_data='data_integrated', align_by_transmission=False, q_index_range=(0, 5), q_range=False, crop_image=False, mode='optical_flow', redo_import=False, regroup_max=16, align_horizontal=True, align_vertical=True, pre_rec_it=5, pre_max_it=5, last_rec_it=40, last_max_it=5, do_align=True)

```
Align data using the Mumott package

Requires that data has been integrated and that sample_dir contains
a subfolder with the data

Parameters
----------
```

```
pattern : str, optional
    substring contained in all files you want to use, by default '.h5'
sub_data : str, optional
    subfolder containing the data, by default 'data_integrated'
q_index_range : tuple, optional
    determines which q-values are used for alignment (sums over them), by
        default (0,5)
q_range : tuple, optional
    give the q-range in nm instead of indices e.g. (15.8,18.1), by default
        False
mode : str, optional
    choose alignment mode, 'optical_flow' or 'phase_matching', by default '
        optical_flow'
crop_image : bool or tuple of int, optional
    give the range you want to use in x and y, e.g. ((0,-1),(10,-10))
    only available with the 'phase_matching'-option, by default False
redo_import : bool, optional
    set True if you want to recalculate data_mumott.h5, by default False
regroup_max : int, optional
    maximum size of groups when downsampling for faster processing, by
        default 16
align_horizontal : bool, optional
    align your data horizontally, by default True
align_vertical : bool, optional
    align your data vertically, by default True
pre_rec_it : int, optional
    reconstruciton iterations for downsampled data, by default 5
pre_max_it : int, optional
    alignment iterations for downsampled data, by default 5
last_rec_it : int, optional
    reconstruciton iterations for full data, by default 40
last_max_it : int, optional
    alignment iterations for full data, by default 5
```

ToC

---

check_alignment_consistency()

```
Plots the squared residuals between data and the projected tomograms,
with data shifted according to the alignment.
```

ToC

---

check_alignment_projection(g=0)

```
Plots the data and the projected tomogram of projection g

Parameters
----------
g : int, optional
    projection running index, by default 0
```

ToC

---

reconstruct_1d_full(q_index_range=None, redo_import=False, only_mumottize=False, batch_size=10)

```
Reconstructs scalar tomographic data such as azimutally averaged
diffraction data. Uses the same alignment as textom

Parameters
----------
q_index_range : list or ndarray or None, optional
    [starting_q_index, end_q_index], takes the whole range if None, by
        default None
redo_import : bool, optional
    set true if you want to redo the preprocessing, by default False
only_mumottize : bool, optional
    only preprocesses a file analysis/rec1d/data_rec1d.h5, by default False
batch_size : int, optional
    number of q-values to load at the same time. Needs to be an integer
        fraction
    of the total number of q-values, else it will crash at the last batch.
        Higher
    numbers will decrease i/o time, but require more memory, by default 10
```

ToC

---

check_powder_pattern(projection=0, datafile_pattern='.h5', check=False)

```
Calculates the theoretical powder pattern according to the input in crystal
    .py
and plots it together with the average over a given projection.

Parameters
```

```
----------
projection : int, optional
    Index of the projection from the data_integrated directory,
    (sorted by rotation angles or alphabetical, as stated in the terminal)
       by default 0
datafile_pattern : str, optional
    allows use a certain file that contains the pattern, by default '.h5'
```

ToC

---

make_model(light_mode=False)

```
Calculates the TexTOM model for reconstructions

Is automatically performed by the functions that require it
```

ToC

---

show_sample_outline()

```
Plots the sample outline after masking.
```

ToC

---

mask_peak_regions(projection=0, datafile_pattern='.h5')

```
lets the user choose the regions that contain the peaks for fitting

Parameters
----------
projection : int, optional
    projection index, by default 0
datafile_pattern : str, optional
    allows use a certain file that contains the pattern, by default '.h5'
```

ToC

---

mask_baseline_regions(projection=0, datafile_pattern='.h5')

```
lets the user choose the regions that contain the peaks for fitting

Parameters
----------
projection : int, optional
    projection index, by default 0
datafile_pattern : str, optional
    allows use a certain file that contains the pattern, by default '.h5'
```

ToC

---

check_baselines(projection=0, datafile_pattern='.h5', min_prominence=0.8)

```
Picks some data containing from the given projection and calculates
    baselines with all available methods,
using the input parameters from the background_subtraction.py file.

Data will be picked based on the prominence of the first chosen peak, so
    that datasets will be shown
that actually contain diffraction peaks. The min_prominence parameter
    decides which images will be
accepted for display.

Parameters
----------
projection : int, optional
    projection index, by default 0
datafile_pattern : str, optional
    allows use a certain file that contains the pattern, by default '.h5'
min_prominence : float, optional
    threshold for prominence of the first peak for a pattern to be
        displayed, default 0.8
```

ToC

---

preprocess_data(proj_test=0, pattern='.h5', use_ion=True)

```
Loads integrated data and pre-processes them for TexTOM

Parameters
```

```
----------
proj_test : int, optional
    projection index (only for displaying), by default 0
pattern : str, optional
    substring contained in all files you want to use,
    (this actually filters the files you process), by default '.h5'
use_ion : bool, optional
    choose if you want to normalize data by the field 'ion' in the
    data files, by default True
```

ToC

---

make_fit(redo=True, sf_from_data=False, set_negative_data_zero=False)

```
Initializes a TexTOM fit object for reconstructions

Is automatically performed by the functions that require it

Parameters
----------
redo : bool, optional
    set True for re-importiing data and model, by default True
sf_from_data : bool, optional
    set True to calculate structure factors from the data instead of the
        atomic model,
    by default False
set_negative_data_zero : bool, optional
    set True to set all negative data zero, by default False
```

ToC

---

toggle_peak_fit(peak_indices)

```
Toggle on/off if chosen peak(s) are used for optimization or not

Parameters
----------
peak_indices : int | list | 1d array
    Indices as provided in the terminal after loading data for fitting
```

ToC

---

optimize(mode=4, proj='full', zero_peak=None, redo_fit=False, step_size_0=1.0, tol=0.001, minstep=1e-09, max_iter=500, alg='simple', save_h5=True)

```
Performs a single TexTOM parameter optimization

Parameters
----------
mode : int, optional
    set 0 for only optimizing order 0, 1 for highest order, 2 for all,
    3 for all but the lowest order, 4 for all with enhanced stepsize
        calibration
    by default 4
proj : str, optional
    choose projections to be optimized: 'full', 'half', 'third', 'notilt',
    by default 'full'
zero_peak : int or None
    index of the peak you want to use for 0-order fitting (should be as
    isotropic as possible), if None uses the whole dataset, default None
redo_fit : bool, optional
    recalculate the fit object, by default False
step_size_0 : float, optional
    starting value for automatic stepsize tuning, by default 1.
tol : float, optional
    tolerance for precision break criterion, by default 1e-3
minstep : float, optional
    minimum stepsize in line search (another break criterion), by default 1
        e-9
max_iter : int, optional
    maximum number of iterations, by default 500
alg : str, optional
    choose algorithm between 'backtracking', 'simple', 'quadratic',
    by default 'simple'
save_h5 : bool, optional
    choose if you want to save the result to the directory analysis/fits,
    by default True
```

ToC

---

list_opt(info=True)

```
Shows all stored optimizations in the terminal
```

ToC

---

load_opt(h5path='last', redo_fit=False)

```
Loads a previous Textom optimization into memory
useful: load_opt(results['optimization'])

Parameters
----------
h5path : str, optional
    filepath, just filename or full path
    if 'last', uses the youngest file is used in analysis/fits/,
    by default 'last'
```

ToC

---

check_lossfunction(opt_file=None)

```
Plots the lossfunction of an optimization

Parameters
----------
opt_file : str or None, optional
    name of the file in the fit directory, if None takes currently loaded
        one, by default None
```

ToC

---

check_fit_average()

```
Plots measured and reconstructed average intensity for each diffraction
    pattern
```

ToC

---

check_fit_per_index(index, mode='line')

```
Plots measured and reconstructed intensity for a chosen diffraction pattern
```

ToC

---

check_fit_random(N=10, mode='line')

```
Generates TexTOM reconstructions and plots them with data for random points

Parameters
----------
N : int, optional
    Number of images created, by default 10
mode : str, optional
    plotting mode, 'line' or 'color', by default line
```

ToC

---

check_residuals()

```
Plots the squared residuals summed over each projection
```

ToC

---

check_projections_average(G=None)

```
Plots the reconstructed average intensity for chosen projections with data

Parameters
----------
G : int or ndarray or None, optional
    projection indices, if None takes 10 equidistant ones, by default None
```

ToC

---

check_projections_residuals(g=0)

```
Plots the residuals per pixel for chosen projections with data

Parameters
----------
g : int e, optional
    projection index, by default 0
```

ToC

---

check_projections_orientations(G=None)

```
Plots the reconstructed average orientations for chosen projections with
    data

Parameters
----------
G : int or ndarray or None, optional
    projection indices, if None takes 10 equidistant ones, by default None
```

ToC

---

calculate_orientation_statistics()

```
Calculates preferred orientations and stds and saves them to results dict
```

ToC

---

calculate_order_parameters(axis=(0, 0, 1))

```
Calculates nematic order parameters and directors per voxel and adds them
    to results

Parameters
----------
axis : tuple, optional
    choose the axis direction along which particles are aligned, by default
        (0,0,1)
```

ToC

---

calculate_segments(thresh=10, min_segment_size=30, max_segments_number=31)

```
Segments the sample based on misorientation borders

Parameters
----------
thresh : float, optional
    misorientation angle threshold inside segment in degree, by default 10
```

```
min_segment_size : int, optional
    minimum number of voxels in segment, by default 30
max_segments_number : int, optional
    maximum number of segments (ordered by size), by default 32
```

ToC

---

show_volume(data='scaling', plane='x', colormap='inferno', cut=1, save=False, show=True)

```
Visualizes the whole sample by slices, colored by a value of your choice

Parameters
----------
data : str or list, optional
    name of one entry in the results dict or list of entries,
    by default 'scaling'
plane : str, optional
    sliceplane 'x'/'y'/'z', by default 'z'
colormap : str, optional
    identifier of matplotlib colormap, default 'inferno'
    https://matplotlib.org/stable/users/explain/colors/colormaps.html
cut : int, optional
    cut colorscale at upper and lower percentile, by default 0.1
save : bool, optional
    saves tomogram as .gif to results/images/, by default False
show : bool, optional
    open the figure upon calling the function, by default True
```

ToC

---

show_slice_ipf(h, plane='z')

```
Plots an inverse pole figure of a sample slice

Parameters
----------
h : int
    height of the slice
plane : str, optional
    slice direction: x/y/z, by default 'z'
```

ToC

---

show_slice_directions(h, plane='z', direction='c')

```
Plots an inverse pole figure of a sample slice

Parameters
----------
h : int
    height of the slice
plane : str, optional
    slice direction: x/y/z, by default 'z'
```

ToC

---

show_volume_ipf(plane='z', save=False, show=True)

```
Plots inverse pole figures as a tomogram with a slider to scroll through
    the sample

Parameters
----------
plane : str, optional
    slice direction: x/y/z, by default 'z'
save : bool, optional
    saves tomogram as .gif to results/images/, by default False
show : bool, optional
    open the figure upon calling the function, by default True
```

ToC

---

show_histogram(x, nbins=50, cut=0.1, segments=None, save=False)

```
plots a histogram of a result parameter

Parameters
----------
x : str,
    name of a scalar from results
bins : int, optional
    number of bins, by default 50
cut : int, optional
    cut upper and lower percentile, by default 0.1
segments : list of int, optional
    list of segments or None for all data, by default None
save : bool/str, optional
```

```
    saves image with specified file extension, e.g. 'png', 'pdf'
    if True uses png, by default False
```

ToC

---

show_correlations(x, y, nbins=50, cut=(0.1, 0.1), segments=None, save=False)

```
Plots a 2D histogram between 2 result parameters

Parameters
----------
x : str,
    name of a scalar from results
y : str,
    name of a scalar from results
bins : int, optional
    number of bins, by default 50
cut : tuple, optional
    cut upper and lower percentile of both parameters, by default (0.1,0.1)
segments : list, optional
    list of segments or None for all data, by default None
save : bool/str, optional
    saves image with specified file extension, e.g. 'png', 'pdf'
    if True uses png, by default False
```

ToC

---

show_voxel_odf(x, y, z, recenter=False, num_samples=1000.0, representation='quaternion')

```
Show a 3D plot of the ODF in the chosen voxel

Parameters
----------
x : int
    voxel x-coordinate
y : int
    voxel y-coordinate
z : int
    voxel z-coordinate
recenter : bool, optional
    if True, searches for the maximum of the distribution and rotates the
        ODF so that
```

```
    the maximum is at the center of the fundamental zone
num_samples : int/float, optional
    number of samples for plot generation, by default 1e3
representation : 'quaternion'|'euler'|'rodrigues'
    different styles ('euler' and 'rodrigues' still experimental)
```

ToC

---

show_voxel_polefigure(x, y, z, hkl=(0, 0, 1), mode='density', alpha=0.1, num_samples=10000.0, recenter=False)

```
Show a polefigure plot for the chosen voxel and hkl

Parameters
----------
x : int
    voxel x-coordinate
y : int
    voxel y-coordinate
z : int
    voxel z-coordinate
hkl : tuple, optional
    Miller indices, by default (1,0,0)
mode : str, optional
    plotting style 'scatter' or 'density', by default 'density'
alpha : float, optional
    opacity of points, only for scatter, by default 0.1
num_samples : int/float, optional
    number of samples for plot generation, by default 1e4
recenter : bool, optional
    if True, searches for the maximum of the distribution and rotates the
        ODF so that
    the maximum is at the center of the fundamental zone
```

ToC

---

show_subvolume_average_odf(x_range=None, y_range=None, z_range=None, custom_mask=None, recenter=False, num_samples=1000.0)

```
Lets you visualize the texture of a chosen part of the sample, specified by
     ranges or a custom mask.

Parameters
```

```
----------
x_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
y_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
z_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
custom_mask : 3darray(int), optional
    Input a custom voxel-mask via e.g.
    custom_mask = np.where(results['order_parameters'].flatten() > 0.5)[0],
        by default None
recenter : bool, optional
    if True, searches for the maximum of the distribution and rotates the
       ODF so that
    the maximum is at the center of the fundamental zone
num_samples : int or float, optional
    number of samples for plot generation, by default 1e3
```

ToC

---

show_subvolume_average_polefigure(x_range=None, y_range=None, z_range=None, custom_mask=None, recenter=False, hkl=(0, 0, 1), mode='density', alpha=0.1, num_samples=10000.0)

```
Lets you visualize the texture of a chosen part of the sample, specified by
    ranges or a custom mask.

Parameters
----------
x_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
y_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
z_range : tuple or None, optional
    range of voxel coordinates, e.g. (0,10), by default None
custom_mask : 3darray(int), optional
    Input a custom voxel-mask via e.g.
    custom_mask = np.where(results['order_parameters'].flatten() > 0.5)[0],
        by default None
recenter : bool, optional
    if True, searches for the maximum of the distribution and rotates the
       ODF so that
    the maximum is at the center of the fundamental zone
hkl : tuple, optional
    Miller indices, by default (1,0,0)
mode : str, optional
    plotting style 'scatter' or 'density', by default 'density'
```

```
alpha : float, optional
    opacity of points, only for scatter, by default 0.1
num_samples : int|float, optional
    number of samples for plot generation, by default 1e4
```

ToC

---

save_results()

```
Saves the results dictionary to a h5 file in the results/ directory
```

ToC

---

export_paraview()

```
Saves the results dictionary to a h5 file in the results/ directory
Here, the coordinate system is converted to fortran order (z,y,x)
```

ToC

---

list_results()

```
Shows all results .h5 files in results directory
```

ToC

---

load_results(h5path='last', make_bg_nan=False)

```
Loads the results from a h5 file do the results dictionary

Parameters
----------
h5path : str, optional
    filepath, just filename or full path
    if 'last', uses the youngest file is used in results/,
    by default 'last'
make_bg_nan : bool, optional
    if true, replaces all excluded voxels by NaN
```

ToC

---

list_results_loaded()

```
Shows all results currently in memory
```

ToC

---

save_images(x, ext='raw')

```
Export results as .raw or .tiff files for dragonfly

Parameters
----------
x : str,
    name of a scalar from results, e.g. 'scaling'
ext : str,
    desired file type by extension, can do 'raw' or 'tiff', default: 'raw'
```

ToC

---

help(method=None, module=None, filter='')

```
Prints information about functions in this library

Parameters
----------
method : str or None, optional
    get more information about a function or None for overview over all
        functions, by default None
module : str or None, optional
    choose python module or None for the base TexTOM library, by default
        None
filter : str, optional
    filter the displayed functions by a substring, by default ''
```

ToC
