TexTOM - Manual

Moritz Frewein, Moritz Stammer, Marc Allain, Tilman Grünewald

April 14, 2025

Contents

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 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 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
```

then

```
pip install textom
```

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
```

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.

To start TexTOM in iPython mode, make sure your environment is active and type `textom`. All TexTOM core functions (sec 5) will be available in the namespace.

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

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

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
n_threads = 128

# Choose if you want to use a GPU for integration and alignment (True/False)
use_gpu = True

# Choose your precision
# recommended np.float64 for double or np.float32 for single precision
data_type = np.float32
```

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 5) 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.

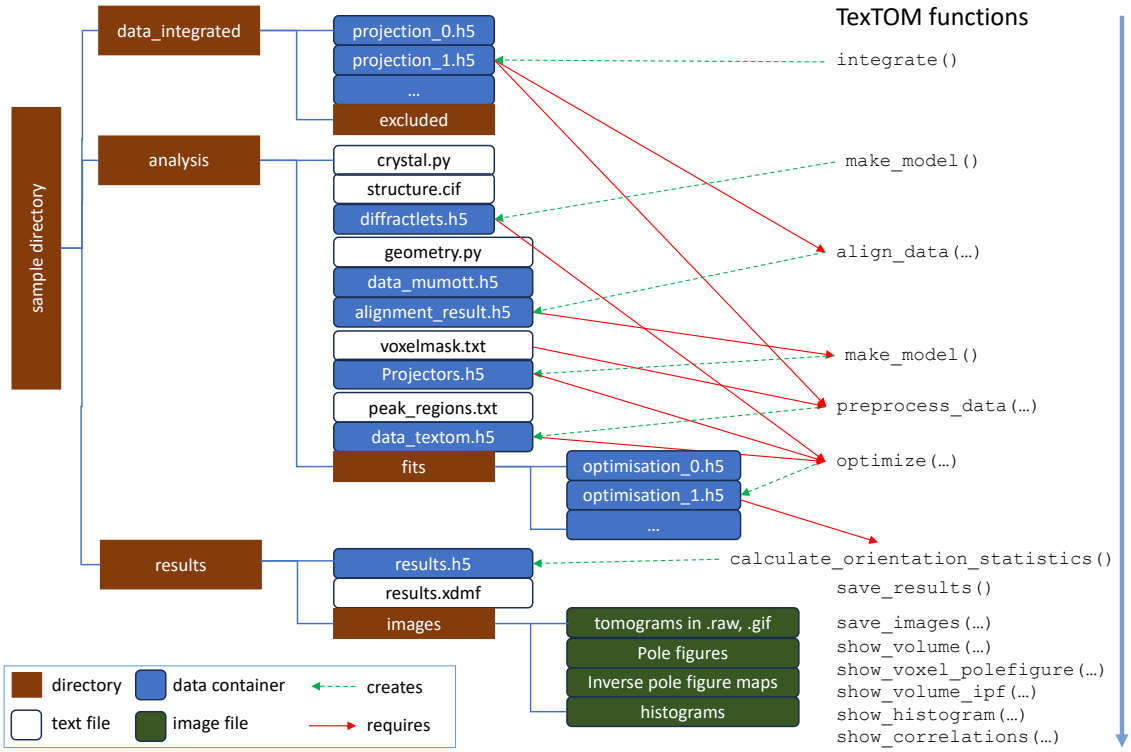

Figure 1: Structure of the sample directory and relevant functions.

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 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()`.

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

ToC

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`.

ToC

4.2 Data integration

The first step in data processing is integration, i.e. azimutal rebinning ("caking") of the 2D-carthesian 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 structure 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. In addition, you need to define beam diameter, step size and scanning mode.

```
# directional vectors
detector_direction_origin = (0,0,1)
detector_direction_positive_90 = (0,-1,0)
inner_axis = (0,0,1) # inner rotation axis
outer_axis = (0,1,0) # outer rotation axis
beam_direction = (1,0,0) # p in mumott
transverse_horizontal = (0,1,0) # j in mumott
transverse_vertical = (0,0,1) # k in mumott

# beam size in um (FWHM)
Dbeam = 0.3
```

```
# step size for scanning in um
Dstep = 0.5

# scanning mode
scan_mode = 'line' #'column' # 'line_snake' # 'column_snake'
```

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 crystal.h5, containing the diffractlets. As this calculation can be lengthy, it is advised to perform it in advance and reuse `diffractlets.h5` for other samples. Keep in mind that the simulated X-ray energy needs to be identical. If `sample_dir/analysis/` contains already a `diffractlets.h5` file, it will use this without asking.

```
import numpy as np
## Define diffraction-related parameters:
# x-ray energy in keV
Ex = 15.2
# angular resolution on detector
dchi = 2*np.pi / 120
# q range for fitting (lower )
q = np.linspace(24.5, 53, num=50)
# path to crystal cif file
```

```
cifPath = 'analysis/BaCO3.cif'
# crystal size (repeat unit cell along each axis)
crystalsize = (15,15,15)
# angular sampling
sampling = 'cubochoric' # or 'simple' (legacy)
```

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 you will be shown a 3D outline of the sample. You can remove other parts of the sample using the input in the figure. After processing, this will create a file `analysis/projectors.h5`, which is used in further processing of this specific sample.

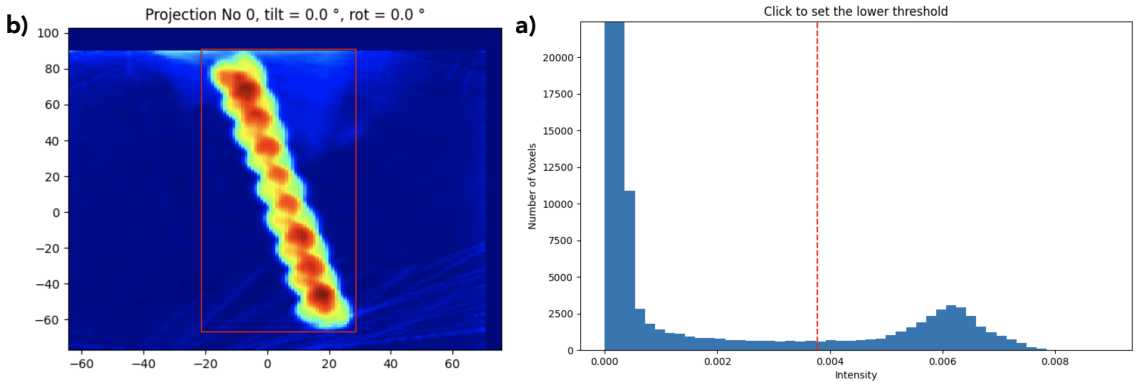

Figure 2: Textom input for masking during calculation of projectors. a) choose the smallest region that surrounds your sample. b) choose the threshold in the tomogram below which you only expect background.

Relevant functions: make_model()

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 filtered according to which data is masked, renormalized and outliers are removed. You will be also 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. There is also a simple background subtraction pipeline, which can be turned on using the argument `draw_baselines=order_polynomial`. Note that this feature is still experimental and might not work with every sample. In any case, it is advised to check the background carefully.

Relevant functions: preprocess_data(...)

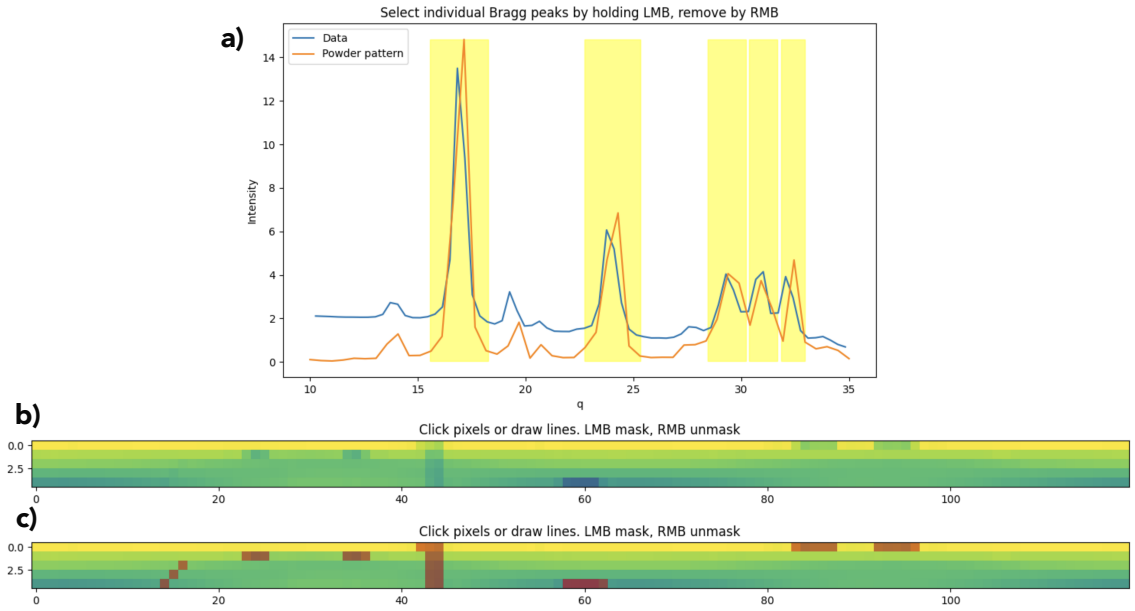

Figure 3: Textom input for choosing q-regions and masking pixels in the integrated images. a) Averaged data and calculated powder pattern with 5 peak regions marked for processing. b) Averaged data, azimuthally resoled. c) Data as before with pixels masked (red)

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(order,mode,...) 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.

The first argument of the function ist the order of the hyperspherical harmonic basis functions. Which orders are available and to be optimized can be seen via check_orders(). We advise to start with low orders and gradually add higher ones. If you increase the order, TexTOM will keep the retrieved coefficients of the lower orders, if you decrease it, the higher ones will be lost and can be reloaded from respective optimization file.

The second argument is the fitting mode of which there are 3: `mode=0` is only suited for order 0 and corresponds to the reconstruction of scalar data. With no further argument, it uses the average over all detector points for this, but it can be directed to utilize a single peak (numbered by n_p starting with the lowest q-value) using the argument `zero_peak` $= n_p$. This is advised if you have a peak of good intensity, that is fairly isotropic. `mode=1` will optimize using the full azimuthally resolved data set, but will vary only the highest order indicated, leaving all other coefficients

unchanged. `mode=2` will optimize all coefficients up to the order provided. A full optimization pipeline is contains all modes, starting with 0 and then a variation of 1 and 2. We recommend using `mode=1` upon going to a higher order, possibly followed by another optimization in the same order with `mode=2`.

There is a full pipeline optimize_auto(...), that follows this optimzation strategy, however keep in mind that the optimal procedure might vary from sample to sample and needs to be verified. Ultimately, you will always get a reconstruction and it is crucial to double and triple check its validity.

Relevant functions: check_orders(...),
optimize(order, mode,...), optimize_auto(...), adjust_data_scaling(),
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 algoritm 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.

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.

Relevant functions: calculate_orientation_statistics(), calculate_segments(...),
save_results(), list_results(), list_results_loaded(), load_results(...)

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(...)

ToC

5 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

---

integrate(mode='online', wait=5.0, confirm=True, ignore=[])

```
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

```
align_data(pattern='.h5', sub_data='data_integrated', q_index_range=(0,
5), q_range=False, crop_image=False, mode='optical_flow',
redo_import=False, flip_fov=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)
```

```
Align data using the Mumott optical flow alignment

Requires that data has been integrated and that sample_dir contains
a subfolder with 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
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)), by
        default False
mode : str, optional
    choose alignment mode, 'optical_flow' or 'phase_matching', by default '
        optical_flow'
redo_import : bool, optional
    set True if you want to recalculate data_mumott.h5, by default False
flip_fov : bool, optional
    only to be used if the fov is in the wrong order in the integrated
    data files, 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
```

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(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
----------
redo_import : bool, optional
    _description_, 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

---

make_model()

```
Calculates the TexTOM model for reconstructions

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

ToC

---

show_sample_outline()

```
No docstring available.
```

ToC

---

absorption_correction(absorption_tomogram=False, absorption_constant_voxel=False)

```
No docstring available.
```

ToC

---

preprocess_data(pattern='.h5', flip_fov=False, baselines='simple', use_ion=True)

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

Parameters
----------
pattern : str, optional
    substring contained in all files you want to use, by default '.h5'
flip_fov : bool, optional
    only to be used if the fov is in the wrong order in the integrated
    data files, by default False
baselines : str or False, optional
    'simple' to draw a straight line under each peak, (does not work if
       there is peak-overlap!)
    'polynomial' to fit the whole q-range minus the peaks with a 5th order
       polynomial (can
        handle peak-overlap, but data needs to be rather smooth where there
            is no peaks),
```

```
    'False' for no baselines (if your background is low), by default '
        simple'
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)

```
Initializes a TexTOM fit object for reconstructions

Is automatically performed by the functions that require it

Parameters
----------
redo : bool, optional
    set True for recalculating, by default True
```

ToC

---

check_orders(n_max=20, exclude_ghosts=True)

```
Lists the sHSH orders available for the present symmetry

Parameters
----------
n_max : int, optional
    maximum order displayed, by default 20
```

ToC

---

optimize(order=0, mode=0, proj='full', zero_peak=None, redo_fit=False, tol=0.001, minstep=1e-09, max_iter=500, alg='quadratic', save_h5=True)

```
Performs a single TexTOM parameter optimization

Parameters
----------
order : int, optional
    maximum sHSH order to be used, by default 0
```

```
mode : int, optional
    set 0 for only optimizing order 0, 1 for highest order, 2 for all,
    by default 0
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
tol : float, optional
    tolerance for precision break criterion, by default 1e-3
minstep : float, optional
    minimum stepsize in line search, by default 1e-9
max_iter : int, optional
    maximum number of iterations, by default 500
alg : str, optional
    choose algorithm between 'backtracking', 'simple', 'quadratic',
    by default 'quadratic'
save_h5 : bool, optional
    choose if you want to save the result to the directory analysis/fits,
    by default True
```

ToC

---

optimize_auto(max_order=8, start_order=None, zero_peak=None, tol_0=1e-07, tol_1=0.001, tol_2=0.0001, minstep_0=1e-09, minstep_1=1e-09, minstep_2=1e-09, max_iter=500, projections='full', alg='quadratic', adj_scal=True, redo_fit=False)

```
Automated TexTOM reconstruction workflow

Parameters
----------
max_order : int, optional
    maximum HSH order to be used, by default 8
start_order : int or None, optional
    lowest order to be fitted, if None continues where you are standing,
    by default None
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
tol_0 : _type_, optional
    tolerance for precision break criterion, mode 0, by default 1e-7
tol_1 : _type_, optional
```

```
    tolerance for precision break criterion, mode 1, by default 1e-3
tol_2 : _type_, optional
    tolerance for precision break criterion, mode 2, by default 1e-4
minstep_0 : _type_, optional
    minimum stepsize in line search, mode 0, by default 1e-9
minstep_1 : _type_, optional
    minimum stepsize in line search, mode 1, by default 1e-9
minstep_2 : _type_, optional
    minimum stepsize in line search, mode 2, by default 1e-9
max_iter : int, optional
    maximum number of iterations, by default 500
projections : str, optional
    choose projections to be optimized: 'full', 'half', 'third', 'notilt',
       by default 'full'
alg : str, optional
    choose algorithm between 'backtracking', 'simple', 'quadratic',
    by default 'quadratic'
adj_scal : bool, optional
    rescales data after 0-order optimization, see adjust_data_scaling(), by
        default True
redo_fit : bool, optional
    recalculate the fit object, by default False
```

ToC

---

adjust_data_scaling()

```
Reestimates the data based on the assumption that normalization constants
contain noise. To be used after fitting the 0th order
```

ToC

---

list_opt(info=True)

```
Shows all stored optimizations
```

ToC

---

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

```
Loads a previous Textom optimization into memory
seful: 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 the reconstructed average intensity for each projection with data

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

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 prefered orientations and stds and saves them to results dict
```

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='z', 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, num_samples=1000, kernel=None, info=True)

```
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
num_samples : int/float, optional
    number of samples for plot generation, by default 1000
kernel : float / 'auto' / None, optional
    choose the strength k of the kernel or 'auto' to refine k,
    if None clips the distribution instead, by default None
```

ToC

---

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

```
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
```

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
```

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
