NanoLib user guide

ETH Zurich
Laboratory of Solid State Physics
Microstructure Research

Danilo A. Zanin - dzanin@phys.ethz.ch

Lorenzo G. De Pietro - depietro@phys.ethz.ch

Quentin Peter - qpeter@stud.phys.ethz.ch

August 22, 2016

NanoLib library allows to open and analyze data generated by the Nanonis SPM Control System™ in MATLAB™. The first version was developed in 2015 by Quentin Peter during its master thesis *Spin Polarized Field Emission STM and Image Processing* in the Solid State Laboratory for Microstruture Research at the ETH Zurich under the supervision of Dr. U. Ramsperger and L.G. De Pietro. Some of the features of this library are still oriented to solve problems related to Quentin's thesis, e.g., *scan_type* field in the header structure (see 2.1). In next versions these feature may be changed and generalized.

NanoLib library is divided in the package folders: *+sxm*, *+dat* and *+utility*.
A function in a folder called *+folder* can be called as *folder.function*.

Contents

1 The +sxm package folder

Images generated via the scanning interface of the Nanonis SPM Control System™ have extension *file.sxm*. Once loaded, variable, from now on *sxmFile*, is a structure divided in: *i*) **header**, a structure containing all information present in the header of the *file.sxm*, and *ii*) **channels**, an array of channel structures containing data and information about every channel. Both, **header** and **channels** can be called by

sxmFile.header and *sxmfile.channels{#}*

being the number of the channel. When only one channel is loaded one refers to the channel simply by *sxmfile.channels*. More information about the substructure of header and channels is presented below.

1.1 sxmFile: header and channels structures

The functions works with a structure that holds every relevant informations. Header and channels structure have following fields:

header is a structure composed of:

- **scan_file** name of the file
- **rec_date** date of the scan
- **rec_time** time of the scan
- **scan_pixels** [nx;ny], number of pixels
- **scan_range** [rx;ry], range [m]
- **scan_offset** [ox;oy], offset [m]
- **scan_angle** tilt angle of the scan
- **scan_dir** 'up' or 'down'
- **bias** bias voltage [V]
- **scan_type** 'STM', 'SEMPA', 'NFESEM', etc.
- ··· Others informations extracted from the file

channels is an array of channel structures composed of:

- **Direction** 'forward' or 'backward'
- **Unit** 'Z' or whatever the unit is
- **Name** The name of the channel

data A $n \times m$ matrix of processed data

lineMedian A $n \times 1$ matrix of raw line median

lineMean A $n \times 1$ matrix of raw line mean

linePlane A $n \times 1$ matrix of raw line mean linear fit

lineResidualSlope a $1 \times m$ matrix of processed column mean linear fit

lineStd A $n \times 1$ matrix of processed line standard deviation

To access the scan data on a structure named *sxmFile*, one should type, e.g., *sxmFile.header.rec_date*.

1.2 +load

This folder contains everything needed to load and process *.sxm* files.

1.2.1 loadsxm

header = loadsxm(fn) loads a file named *fn.sxm* and returns the Header. This function is called by *load.loadProcessedSxM* and **should not be called directly**.

[header, data] = loadsxm(fn, i) reads the channel *i* and returns its *data*.

.sxm files are composed of an ascii header and of single precision binary data. They are separated by 0x1A 0x04 (SUB EOT). This file is provided by Nanonis SPM Control System™ and loads a specified channel from a *.sxm* file.

1.2.2 processChannel

channel = processChannel(channel, header) Process the *channel* as described below using the informations form *header*. This function is called by *load.loadProcessedXXX* and should not be called directly.

channel = processChannel(channel,header,corrType) If *corrType* is set to 'Median', the median is used instead of the mean for lines corrections. If it is set to 'PlaneLineCorrection' a linear fit is used.

The processing orientate and rotate the data so that all the images are comparable. Everything that is removed is saved in the output structure to avoid loosing informations.
The mean value of the measurement under the conditions of each pixel must be extracted from the data. As there is drift and other instabilities, the mean value

of the data is generally not a good value. The mean of each line is used instead, as the measurement conditions doesn't change too much during one line. Others possibility include the median or the mean plane. The mean plane along the line is also removed.
For STM, This offset is subtracted. For NFESEM and SEMPA, it is divided, as justified in the thesis.

1.2.3 loadProcessedSxM

file=loadProcessedSxM(fn) loads and process all the channels of *.sxm* file named *fn*. The structure *file* contains all the informations and is used in a large number of other functions.

file=loadProcessedSxM(fn, chn) only loads the channels whose numbers are in the array *chn*

file=loadProcessedSxM(fn, corrType) If *corrType* is set to 'MedianCorrection', the median is used instead of the mean for lines corrections. If it set to 'PlaneLineCorrection' a linear fit is used.

The loading is done with *load.loadsxm* and processing with *load.processChannel*.

1.3 +plot

This package contains everything needed to plot the data.

1.3.1 folder2png

folder2png(folderName) finds every *.par* and *.sxm* files in *folderName*, plot all relevant channels and saves the images in a *image* folder.

1.3.2 plotData

[h, range] = plotData(data, name, unit, header) plots the *data* using informations from the *header*. The figure title is deduced from *name* and *unit*. It returns the plot handle *h* and the chosen range *range*.

[h, range] = plotData(data, name, unit, header, xoffset, yoffset) adds an offset to the plot.

The range is 2 STD. If the data is STM, only the lines with low std are considered for the range.

1.3.3 plotChannel

[h, range] = plotChannel(channel, header) plots the *channel* using informations from the *header*. It returns the plot handle *h* and the chosen range *range*.

[h, range] = plotChannel(channel, header, xoffset, yoffset) adds an offset to the plot.

It calls *plot.plotData* on the channel data.

1.3.4 plotFile

[h, range] = plotFile(file, n) plots the n^{th} channel of *file*. It returns the plot handle *h* and the chosen range *range*.

[h, range] = plotFile(file, n, xoffset, yoffset) adds an offset to the plot.

It calls *plot.plotChannel*.

1.3.5 plotHistogram

plotHistogram(data, range) plots an histogram of *data* and draw lines on the limit of *range*. It removes the $.1\%$ most extreme values.

1.4 +op

This package contains various useful functions.

1.4.1 combineChannel

channel=combineChannel(file, name, chn, chw) combined the channels *chn* of the *file* structure with weights *chw* and return a new *channel* with name *name*.

1.4.2 filterData

[filtered, removed] = filterData(data, pixSize) filters the *data* with Fourier transform. The filtering keeps structures of approximatively *pixSize* pixels. It returns the filtered data *filtered* and the removed noise *removed*.

[filtered, removed] = filterData(data, pixSize, 'plotFFT', zoom) additionally plots the Fourier plane. The optional variable *zoom* has default value 8 and is used to zoom in the Fourier plane.

1.4.3 getOffset

[offset, XC, centerOffset] = getOffset(img1, header1, img2, header2) compares the images matrices *img1* and *img2* using informations from the two *headeri* to find the most probable *offset*. The units of *offset* are from header.scan_range. It correspond to the maximum of the cross correlation matrix *XC*. The corresponding offset relative to the centre of the two images is returned in *centerOffset*.

[offset, XC, centerOffset] = getOffset(img1, header1, img2, header2, 'mask') compares masks instead of images.

The offset is from the origin of the image, which is in a corner. The offset of the center is the centerOffset, but is less convenient to work with.

1.4.4 getRadialFFT

[wavelength, radial_spectrum] =getRadialFFT(data) Computes the *radial spectrum* of the image saved in *data* and the corresponding *wavelength*. The wavelength unit is pixel.

[wavelength, radial_spectrum] =getRadialFFT(data,pixPerUnit) Changes the wavelength unit with the number of pixels per units, *pixPerUnit*.

This function is used to study the radial spectrum of an image computed from the FFT.

1.4.5 getRadialNoise

[noise_fit, signal_start, signal_error, noise_coeff] = getRadialNoise(wavelength, radial_average) tries to fit a noise from the data of *getRadialFFT*. *noise_fit* is the detected noise. *signal_start* is the first position where the signal is detected. *signal_error* is the error caused by the discrete nature of the signal on *signal_start*. *noise_coeff* gives the power law coefficients for the first detected noise.

[noise_fit, signal_start, signal_error, noise_coeff] = getRadialNoise(wavelength, radial_average, maxNbrNoise) Limits the number of noises to *maxNbrNoise*. The default value is 10.

1.4.6 getRange

[xrange, yrange] = getRange(header) extract the ranges *xrange*, *yrange* from *header*.

1.4.7 nanHighStd

data = nanHighStd(data) is useful for STM measurements. Usually the lines with very high std don't carry informations, and thus if a line has $std > 3median$, it is set to nan.

1.4.8 nanonisMap

colorMap = nanonisMap(nPti) is a color map function that generates a Nanonis color like mapping of *nPti* number of colors. *nPti* is an optional value. If not provided *nPti* = 64 per default.

1.4.9 interpHighStd

data = interpHighStd(data) Removes the lines with high STD values and interpolates the missing values.

1.4.10 interpPeaks

data = interpPeaks(data) Removes the data witch are too far from the mean and interpolates the missing values.

1.5 +mask

Theses functions are useful to compute threshold mask and apply them.

1.5.1 applyMask

applyMask(mask) apply the boolean mask *mask* to the current figure.

applyMask(mask, color, alpha, xrange, yrange) apply the boolean mask *mask* in the range *xrange*, *yrange* with color *color* and transparency *alpha*.

The ranges are vectors containing a start point and an end point. See MATLAB's *image* documentation.

1.5.2 getMask

[maskUp, maskDown, flatData] = getMask(data, pixSize, prctUp, prctDown) flatten and filter the *data* before computing threshold masks. *flatData* is the flattened and filtered data. *maskUp* marks everithing above *prctUp* and *maskDown* below *prctDown*. The filtering is done using *op.filterData*, to which *pixSize* is passed to keep features of this approximate size.

[maskUp, maskDown, flatData] = getMask(data, pixSize, prctUp, prctDown, 'plotFFT', zoom) Additionally passes *'plotFFT',zoom* to *op.filterData* to visualize the Fourier plane. *zoom* is optional.

The flattening is done using sliding mean.

1.6 +convolve2

This in an improved version of MATLAB's conv2 matrix. It allows a better gestion of boundaries. It was downloaded from MATLAB file exchange. See the license file.

2 The +dat package folder

Besides surface imaging Nanonis SPM Control System™ allows to store data measured by the physical channels. Data from the so called experiments are stored in a *file.dat*. Once loaded, variable, from now on *datFile*, is a structure divided in: *i*) **header**, a structure containing all information present in the header of the *file.dat*, and *ii*) **channels**, an array of channel structures containing data and information about every channel. Both, **header** and **channels** can be called by

datFile.header and *datfile.channels{#}*

being the number of the channel. When only one channel is loaded one refers to the channel simply as *datfile.channels*. More information about the substructure of header and channels is presented below.

2.1 sxmFile: header and channels tructures

The functions works with a structure that holds every relevant informations. To access the scan data on a structure named *expFile*, one should type *expFile.header.rec_date*. Header and channels structure have following fields:

header is a structure composed of:

- **file** name of the file
- **path** path of the file
- **experiment** experiment name
- **sweep_signal** signal that is varied during the experiment, it can be also the time
- **rec_date** date of the scan
- **rec_time** time of the scan
- **points** number of experiment points
- **grid_points** number of experiment repetition
- **channels** list of registered channels
- **list** is a $2 \times n$ list of string, n being the number of lines in the text header. Lines in the *header.list* are of the form {'Key','data'}, e.g., {'rec_date','22.08.2016'}
- $\cdots$ Others informations extracted from the file depending on the specific experiment

channels is an array of channel structures composed of:

- **Direction** 'forward' or 'backward'
- **Unit** 'Z' or whatever the unit is
- **Name** The name of the channel
- **data** A $n \times m$ matrix of processed data

The first channel, i.e. channel(1), is reserved to the sweep_signal.

2.2 +load

This folder contains everything needed to load and process *.dat* files.

2.2.1 experiment_*

files.dat are all characterized by a unique **experiment_name**, that is saved in the first line of every *.dat* file. In the follow we refer to those *files.dat* simply as *experiments*. Different *experiments* have different headers and data characteristics. Every *experiment* have a specific function called *experiment_*∗, ∗ being the name of the experiment. *experiment_*∗ are called automatically by the *loadDat* function as listed below.

experiment_name = experiment_∗('get experiment') returns the **experiment_name**.

header = experiment_∗('get header',header,datasForKey) returns the complete *header* of the *experiment*. The input variable *header* contains only the variables *experiment* and *list*.

> **data = datasForKey(key)** is function returns the *data* according to a specific *key* as stored in the variable *header.list*.

[header,channels] = experiment_∗('process data',header,data) stores data into the *channels* structure described above. Where needed some additional processing are applied to the data. Header's information are adjusted accordingly.

Further *experiments* can be implemented by simply defining a function called *experiment_newExperiment*. New *experiment* functions **must** have the same structure described above and should be saved in the *+load* package folder.

2.2.2 getAllExperiments

experiment_list = getAllExperiments() returns a $2 \times n$ list, where n is the number of the function *experiment_*$*$. In the first column is listed the unique name of the experiment saved in the *+load* package folder. In the second column compare the correspondent function, i.e., *experiment_*$*$.

This function is used by the function *loadDat* when loading different *experiments*.

2.2.3 loadDat

By mean of the *experiment* structure described in the two previous sections, *loadDat* automatically recognize the type of *experiment* and load it.

file=loadDat() ask for a *fileName.dat* and load it.

file=loadDat(fileName) load the file named *fileName.dat*.

file=loadDat(fileName,pathName) load the file named *fileName.dat* at a given *pathName*.

2.3 +plotDat

This package contains everything needed to plot the data.

2.3.1 plotData

To be written

2.3.2 plotChannel

To be written

2.3.3 plotFile

To be written

3 +utility

The package folder *+utility* contains generic functions, which can be used when analyzing both *sxmFiles* and *datFiles*

3.0.1 getChannel

channelNumber = getChannel(channels,channelNames) returns all channel numbers where *channels.Name* matches the *channelNames*. *channelNames* can be either a single string or a list of strings.

channelNumber = getChannel(channels,channelNames,direction) returns only the channel number where *channels.Direction* matches the *direction*, too.

3.0.2 getColor

[color,colorScale] = getColor(x,xRange) computes the ratio between a value *x* and the *xRange* (2 by 1 array). It returns the *color* – and the *colorScale* – according to a predefined color map. *Jet* is the default color map.

[color,colorScale] = getColor(x,xRange,mapping) allows to provide e specified *mapping* other than the default, i.e. *jet*. *mapping* should be a color map function, e.g. *hsv*, *parula*, *hot*, *summer*m *autumn*. Note that, since *mapping* is an argument of functions *getColor*, it must be called by function handle "at" – @.

4 Examples

NanoLib library comes with few example showing the basics usage of the library. Some files are also provided in the directory *Files*. Below a list of all examples with a short explanation.

4.1 SxM_Example

example_open_SxM shows different ways to load a *file.sxm*.

example_process_option shows different ways to process a file while loading a *file.sxm*.

example_get_drift detect XY-offset between two different *sxmFiles*.

example_mask generates a mask of a *sxmFile* and apply on the original image.

example_RadialFFT applies *op.getRadialFFT* and plots some interesting quantities.

4.2 dat_Example

example_open_Dat shows different ways to load some *experiment.dat*.