NanoLib Documentation

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 5, 2016

Contents

1 Introduction

NanoLib library allows to open and analyze data saved using the Nanonis SPM Control System. The first version was developed 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. De Pietro.
The library is divided in package folders *+folder*. A function in a folder called *+folder* can be called as *folder.function*. In the following discussion, the section names with a + refers to a package folder.

2 STM_SEM

2.1 Image Structure

The functions works with a structure that holds every relevant informations. To access the scan date on a structure named *stmFile*, one should type *stmFile.header.rec_date*. The structure has the following fields:

header is a structure composed of:

- **scan_file** The name of the file
- **rec_date** The date of the scan
- **rec_time** The time of the scan
- **scan_pixels** [nx;ny], the number of pixels
- **scan_range** [rx;ry], the range [m]
- **scan_offset** [ox;oy], the offset [m]
- **scan_angle** The tilt angle of the scan
- **scan_dir** 'up' or 'down'
- **bias** The 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

2.2 +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.

2.3 +load

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

2.3.1 loadsxm

header = loadsxm(fn) loads the *.sxm* file *fn* and returns the Header. This function is called by *load.loadProcessedXXX* 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 and loads a specified channel from a *.sxm* file.

2.3.2 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.

2.3.3 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*.

2.3.4 loadProcessedPar

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

file=loadProcessedPar(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 par data are composed of a *.par* file that holds the header and of several *.tfi* files that holds int 16 binary data for each channel.
A header structure that match the .sxm header structure is extracted from the *.par* file, as well as infos about the Channels.

2.4 +mask

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

2.4.1 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.

2.4.2 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.

2.5 +op

This package contains various useful functions.

2.5.1 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*.

2.5.2 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.

2.5.3 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.

2.5.4 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.

2.5.5 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.

2.5.6 getRange

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

2.5.7 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.

2.5.8 interpHighStd

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

2.5.9 interpPeaks

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

2.6 +plot

This package contains everything needed to plot the data.

2.6.1 folder2png

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

2.6.2 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.

2.6.3 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.

2.6.4 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*.

2.6.5 plotHistogram

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

2.7 Tests

2.7.1 testDrift

This script tests the XY-offset detection.

2.7.2 testMask

This script tests the mask generation, application, and drift detection.

2.7.3 testMask2

This script tests the mask generation, application, and drift detection.

2.7.4 testSEM

Test the SEM processing and plotting

2.7.5 testSTM

Test the STM processing and plotting

2.7.6 testRadialFFT

Test *op.getRadialFFT* and plots some interesting quantities.

2.7.7 testPar

Test PAR files loading.

2.7.8 testMedian

Test different ways to load a file.

2.8 Scripts

2.8.1 createImages

Script to call *plot.folder2png* on every folder inside a folder.

2.8.2 stdVsNe

Script to study the effect of the number of electrons on the standard deviation. The normalised variance found in the hysteresis calculations is also displayed.

2.8.3 SeriesFFTSpectrum

Scripts that deduces the resolution of a series of image from the Fourier transform.

2.8.4 MultiSeriesFFTSpectrum

Scripts that deduces the resolution of a series of series of image from the Fourier transform.

2.8.5 STMFFTSpectrum

Script used to compare STM and SEM images.

2.9 Old

Some old files that are not useful.