

TFY4330 - Analysis of AFM data

Setup

Install Gwyddion and export .spm files as .gwy

Install gwyddion: <http://gwyddion.net/download.php#stable-sources>

(macOS needs an open source installer. Follow this guide:

<http://gwyddion.net/documentation/user-guide-en/installation-osx.html>)

Then open each of your .spm files in Gwyddion.

Go to File -> Save as -> Add .gwy to the filename and make sure gwy-file format is selected.

Hint: ctrl + shift + s is the shortcut for save as.

When this is done, we have no more use for Gwyddion, and may close it (and uninstall, really).

Make an environment for AFM data analysis, and install from environment.yml or requirements.txt

Everyone should have this as a bookmark:

<https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

Open your prompt, and navigate to the folder where you have the .yml and .txt files.

Now Either:

1. Create a new environment and fill it with the .yml file directly:

- `conda env create -n TFY4330_AFM -f TFY4330_AFM.yml`

You may give it another name. For pip, do the pip equivalent.

OR:

1. Create a new empty environment with conda (Again for pip, do the pip equivalent):

- `conda env create --name your-awesome-enviroment-name`

2. Activate the environment: `conda activate your-awesome-enviroment-name`

3. Install using the following:

- `conda env update --file TFY4330_AFM.yml --prune`

To check if the environment is created, type `conda env list`.

You should see a new environment in the list.

Activate this environment by typing conda activate whatever-new-name-in-the-list.

Check if the packages are installed by typing conda list

For PIP or if Murphys law:

1. Create a new environment
2. Activate the environment
3. Install python (3.8 or higher, but not too high)
4. Use pip to install gwyfile, matplotlib_scalebar, and h5py.
5. Install what remains of: matplotlib, numpy, pandas, scipy, scikit-learn, scikit-image, jupyter, ipykernel (conda or pip)
6. If you get requirement errors during the exercise, install the missing packages.

If you feel lucky:

1. Activate your previously created TFY4330 environment
2. pip install gwyfile
3. See if the code runs. If not, install the missing packages.
4. Hope there are no conflicts.

```
In [1]: #TODO: Test dependencies by running this cell.
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

import os
import sys

#NOTE: In case you do the analysis in another folder than the code, you need to provide the path to the code
wallpy_cmd = #NOTE: Path to the code
sys.path.append(os.path.join(wallpy_cmd, "Display"))
sys.path.append(os.path.join(wallpy_cmd, "IO"))
sys.path.append(os.path.join(wallpy_cmd, "Functionality"))
from figures import FigureSinglePlot, FigureSubplots
from universal_reader import GwyFile
import transformations as trans
import attributes as attr
import scipy as sp

%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

AFM analysis

Create GwyFile objects

For each .gwy file you have, create a GwyFile object.

The GwyFile object requires the *path* and *filename* of the .gwy file as input args

Hint: This repetitive task can be done in a loop, giving you a list of GwyFile objects.

Hint: One can get all files in a folder by using the os.listdir() function, and one can filter the files by checking the file extension using str().endswith('your_desired_file_extension').

In [2]:

```
#TODO: Enter the path to the folder containing the .gwy files
#TODO: Get all the correct files in the folder
#TODO: Create a list of GwyFile objects
```


GwyFile Conversion

Run the '**call()**' method, which can be done simply by calling the object. I.e.:

```
gwy = GwyFile(*args)  
gwy()
```

This will create a h5 file structure holding the AFM image as well as necessary metadata. This can also be performed in a loop.

Do not mind error messages about keywords not found. Notice that some channels may have been lost. This is because the Bruker software does not include enough unique channel information in the name. If you want to keep both Trace and Retrace, add something to the channel name in the Gwyddion data Browser, like "Trace" and "Retrace". Distinguishing the channel names in Gwyddion before conversion is also something one can do to separate height amplitude and phase from mfm amplitude and phase.

```
In [3]: #TODO: Call on the GwyFile objects.  
#TODO: When finished running this one, comment it out so that you do not spend time
```

Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.

```
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
Phase_mfm not found in keywords dictionary.
Phase_None_4
AmplitudeActual_main not found in keywords dictionary.
ampMFM_None_5
Channels found:
ZSensor not found in keywords dictionary.
ZSensor_None_0
AmplitudeError not found in keywords dictionary.
Amplitude_None_1
Phase not found in keywords dictionary.
Phase_None_2
AmplitudeActual not found in keywords dictionary.
ampMFM_None_3
```

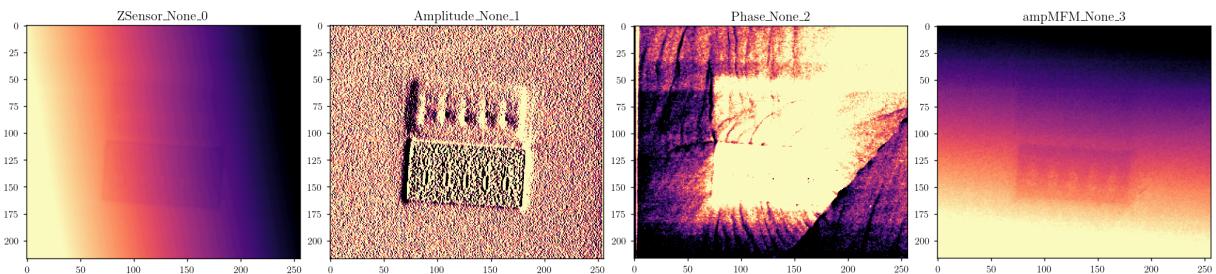
Out[3]: [None, None, None, None, None, None, None, None, None]

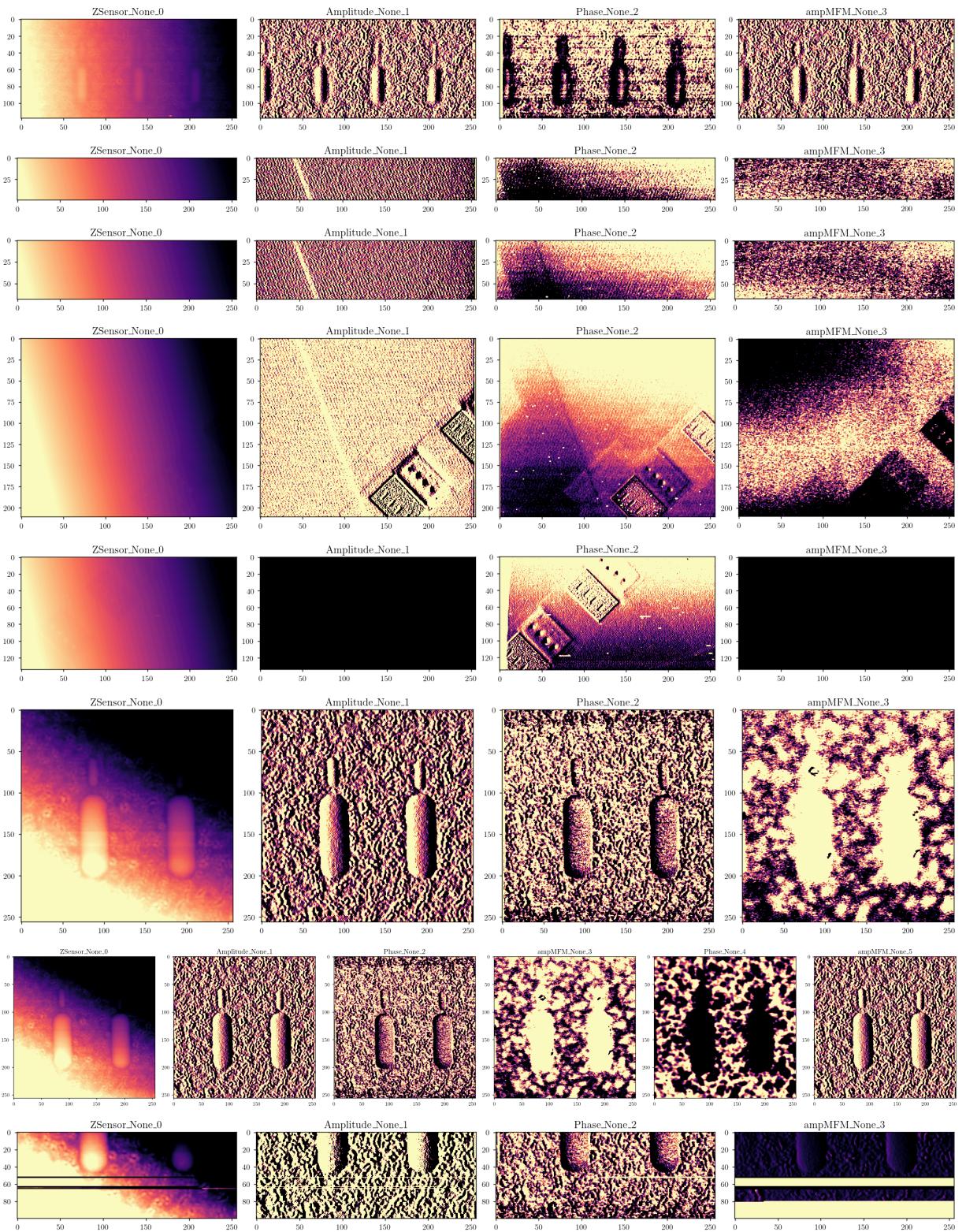
Get an overview of your data

We can plot all the channels with channel name and indices using the 'create_overview_file()' method.

If you do this in a loop, you can create overview .png files for each GwyFile object, and if you rememeber to use %matplotlib inline, you will get an overview of all the plots in the notebook.

In [4]: *#TODO: Create an overview file/plot for each of the GwyFile objects.*





Choose files and channels of interest

Chose an overview scan and a close-up high resolution scan of the features of interest.

These you may extract from the list, e.g. by indexing.

NB! Make sure you know what you extract. The instrument mixes and matches channels, and you may not get what you expect.

Generally, the MFM should not correlate perfectly with the topography, and have worse

spatial resolution.

Optional, by opening the .spm file in a text editor, you can confirm that the phase and amplitude channels indeed are MFM channels by the "Scan Line: Interleave" keyword for the given "Image Data" keyword.

For each of the interesting files, extract:

- A channel with the quantitative topography (Height, or Zsensor)
- A channel with the error signal (Amplitude)
- A channel with the MFM amplitude (ampMFM)
- A channel with the MFM phase (Phase)

Extracting channels is most easily done by indexing the desired channel, gwyfile[channel_index], where channel_index is the number at the end of the channel title in the overview plot.

For more on indexing, visit the source code.

```
In [5]: #TODO: Extract GwyFile objects of interest, as well as the corresponding channels o
```

Process the height data

As you can see from the overview, there is a gradient in the height data. This is due to the AFM and the sample not being perfectly parallel.

We can remove this gradient by fitting and subtracting a plane.

We will also see if there are any second or higher order trends in the data that need to be removed.

Finally, we can do the same subtraction by providing only a few points that we know to be flat background.

This can be helpful when the entire sample surface is not flat due to FIB-milling or lithography, for instance.

1. Make a processed_height array by using the plane_level() method from transformations.py (Imported as trans).
2. Make another height array by using the poly2D_least_squares() method from transformations.py. Provide the order n of the polynomial fit.
3. Make a third height array by using the poly2D_3point_level() method where you set order n and provide n+1 points that should be flat.

Do this for both the overview and the high resolution image.

Hint: plane_level returns a tuple for no obvious reason.

Hint: Look at what arguments the methods take.

```
In [6]: #TODO: Play with background subtraction
```

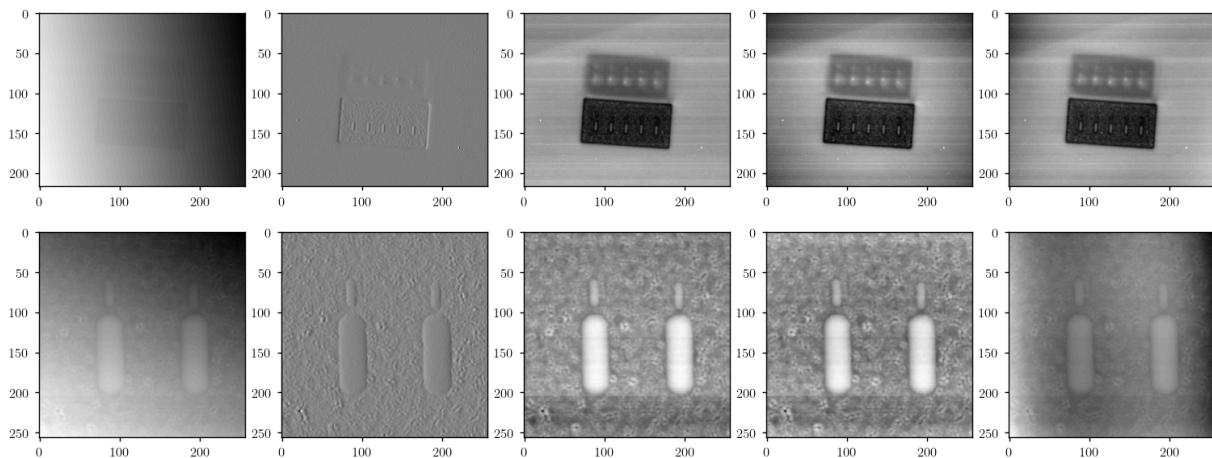
Analysing the topography

Plot the raw height data, the topography amplitude signal, and the three processed height arrays side by side using plt.subplots() and imshow().

Do this for both the overview and the high resolution image. You can decide if you want to make two figures or one figure with two rows.

In the latter case, you will get a 2D array of axes objects. You can iterate over axs.shape[1], and call axs[0][i].imshow() and axs[1][i].imshow(), respectively, for overview and hr image.

```
In [7]: #TODO: Plot the height data alongside and evaluate the background subtraction method
```



Plot Height, MFM amplitude, and MFM phase

Choose one of the processed height arrays, and plot it together with the MFM amplitude and phase.

Again, include overview and high resolution in same figure. You can decide if you want overview and hr on top of each other, or side by side.

Make it pretty by:

- cropping if necessary
- adding scalebars
- choosing colormaps for the different signals
- adjusting vmin and vmax by using np.quantile(arr, q)
- adding letters and labels.
- Decide on whether to use colorbars or not.

Hint: In attributes.py, imported as attr, you can find an implementation of a scalebar function, as well as subplot labelling function.

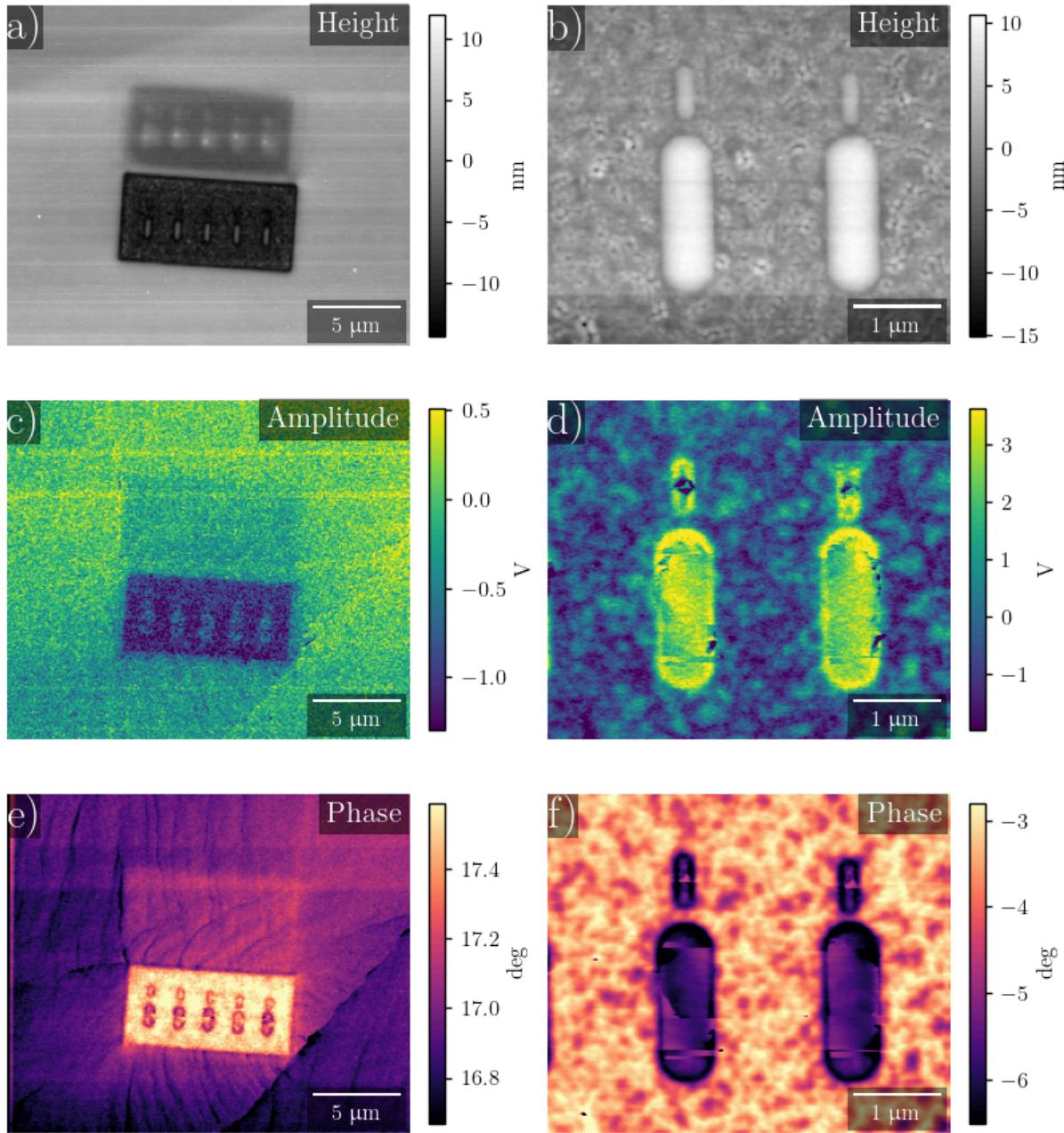
```
attr.add_scalebar(ax, res)
```

```
attr.add_letter(ax, letter), where letter can be chr(97) + n to quickly get a, b, c, d etc.
```

OR, use the functions used in previous exercises.

To get the resolution of the image, use `gwy_file_obj.index_metadata(index, "xres")`

In [22]: `#TODO: Make a pretty figure with overview and high resolution data, using height, a
#TODO: Try to reduce the amount of repetitive code by defining lists and subsequent`



Drawing crossections

1D crossections are often useful and necessary to properly analyze and present AFM data.

Make two crossections from the high-resolution data.

You can either do vertical/horizontal crossections, or utilize the functions in `crossectioning.py` to draw arbitrary crossections.

For the latter, the points may be chosen interactively by switching to `%matplotlib qt`, plotting a helping figure, calling the `retrieve_crossection_coordinates()` function, reading the

coordinates, and retrieving the crossection using the `retrieve_crossection()` function.

Alternatively, coordinates may be entered directly into the last function.

In either case, choose an appropriate line width to average over to get an acceptable signal to noise ratio. Here, slicing and `arr.mean(axis: int)` may be useful.

```
In [24]: #TODO: Draw crossections from the high resolution data
```

Plotting a figure with crossections

For this figure, you will be using gridspec one way or another.

You can either code using gridspec, or you can try `FigureSubplots` from `figures.py`.

This class is an overhead class that simplifies the use of gridspec. \

The figure should fulfill the following requirements:

- The figure should be 3x3 in terms of grid cells (`nrows=3, ncols=3` for `FigureSubplots`)
- 1 out of 3 column widths should be used for the 2D image of the channel.
- 2 out of 3 column widths should be used for the crossections.
- The position of the crossection should be indicated on the 2D image.

Hint: Read the `FigureSubplots` docstring. It is unfortunately not very readable due to many `**kwargs`.

Essentially, create the fig by initiated a class instance.

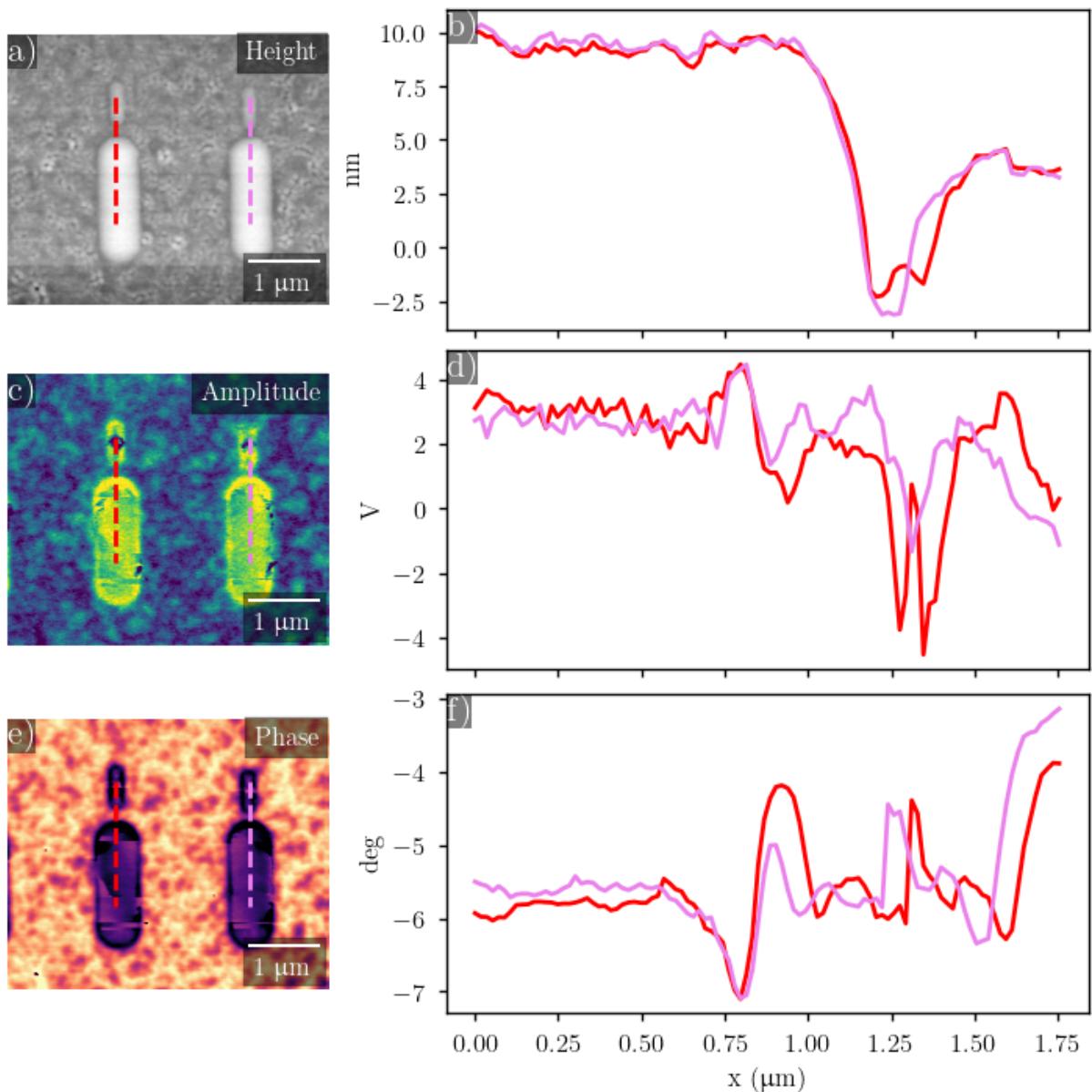
For each subplot, call `create_subplot(row:int, col:int, rowspan:int, colspan:int)` with the desired parameters. Note that the span is the end index for some reason...

This will return `Ax` objects, not `axis` objects. This is an overhead class, and you may access the actual axis object by writing `Ax.ax`.

Labelling can be done using the `label_subplot()` method.

Finally, call the figure object to save.

```
In [32]: #TODO: Create a figure with subplots using FigureSubplots or Gridspec directly.  
#TODO: Plot the 2D data and the crossections in the same figure.  
#TODO: Add all the fuzz.
```



Plotting in 3D with phase coloring

MFM gives quantitative (x,y,z)-information as well as qualitative information about the magnetic properties of the sample.

This can be visualized in a single plot by plotting the height data in 3D with surface coloring representing the MFM phase.

Often these types of plots do not convey that much information, but they are satisfying and cool, and one should therefore learn to make them.

Use the `fig = plt.figure()` and `ax = fig.add_subplot(111, projection='3d')` to make a 3D plot.

Use `ax.plot_surface()` to plot the high resolution height data, and use the `c` argument to color the surface by the MFM phase.

By using `mpl.colors.Normalize()` and `cm.ScalarMappable()` you can create a colorbar for the phase.

Finally, make everything pretty.

```
In [23]: #TODO: Create figure and ax with 3D projection
#TODO: Make the x, and y axes in by np.arange and np.meshgrid to map out all coord
#TODO: Choose a cmap from mpl.cm.'colormap_name', and normalize the color range usi
#TODO: Normalize the phase data by passing it as an argument to the norm object. Th
#TODO: Plot the surface using ax.plot_surface(). Provide the X, Y, Z and facecolors
#TODO: Do all the pretty fuzz, like labels, limits, remove grid, make transparent,
```

