AFM Python Analysis Tutorial

This tutorial demonstrates the basics of the AFM python tools for data analysis. Currently the scripts are available as an offline file but will be uploaded as a python package in due course. Following the steps below will show you how to carry out simple analysis and plotting of AFM data produced with Nanoscope and explain how to carryout your own more specific analysis. For each snippet of code - try running it to see what it outputs.

Prerequisites

To use the tools you will need a few python packages (numpy, plotly, scipy and easygui). To install these type the following commands into a terminal window:

In [ ]:

```
pip install numpy
```

In [ ]:

```
pip install plotly
```

In [ ]:

```
pip install scipy
```

In [ ]:

```
pip install easygui
```

Getting started

Start off by creating a folder containing the afm.py file and your script. For beginers it may be best to use a jupyter notebook here, as has been used for this tutorial. Open your script file and type (note this is case sensetive):

In [ ]:

```
from afm import AFM
```

To carry out basic analysis you can use the inbuilt 'run' command.

(Check the window hasn't opened behind python if this is running slowly.)

In [ ]:

```
expt = AFM()
expt.run()
```

This will open a window, prompting you to select the AFM data you wish to analyse. Navigate to the folder and select the file with the '.000' extension. The tools will automatically align the data with thebaseline and set the surface height to zero. A plot will open with the processed force curves, use the slider to move through each experiment. The code will also aclculate the adhesion and modules. The mean and standard deviation will appera in the output.

It is likley you will wish to carry out more detailled analysis than provided by this function. The following sections will guide you through all the individual functions available.

Naming variables

If you have carried out multiple experiments and would like to compare the results, you should assign an instance of the AFM class for each. Name a variable for each experiment and call the class:

In [ ]:

```
expt_1 = AFM()
expt_2 = AFM()
expt_3 = AFM()
```

Importing data

To import a dataset use the 'input_files' command for each experiment:

Note - the GUI for selecting a file will only run on a local version of python (not in the Colab tutorial). This will prompt an error in Colab - use manual file_name entry below.

In [ ]:

```
expt_1.input_files()
expt_2.input_files()
expt_3.input_files()
```

This will open a window prompting you to select the '.000' file for each experiment. The code will then automatcally import all files with the same name.

If you would prefer to manually set the file name (useful if you are running the script multiple time), you can name the 'file_name' variable and set the 'gui_on' flag in 'import_files' to 'False':

In [ ]:

```
expt_1.file_name = '/insert/your/file/path/here.000'
expt_1.input_files(gui_on = False)
```

Naming experiments

It may be useful to name each experiment, for plotting etc. To do so, use the following command:

In [ ]:

```
expt_1.set_run_name('Sample 1')
```

Finding nanoscope parameters

The next step is to import the relevant parameters from the file (such as deflection sensetivity and spring constant). To do so, due the 'nanoscope_params' function as follows:

In [ ]:

```
expt_1.nanoscope_params()
expt_2.nanoscope_params()
expt_3.nanoscope_params()
```

If you would like to manually set the deflaction sensetivity, use the following function:

In [ ]:

```
expt_1.set_def_sens(100)
```

If you would like to manually set the spring constant, use the following function:

In [ ]:

```
expt_2.set_spr_const(0.32)
```

Reading force curve data

Now we have the parameters, we need the actual data. To import this, use the 'nanoscope_read' command:

In [ ]:

```
expt_1.nanoscope_read()
expt_2.nanoscope_read()
expt_3.nanoscope_read()
```

Plot raw data

To view the raw force curve data before any processing use the command below. The number in the brackets specifies which curve to plot. Remember, when using python, the numbers start at 0 (i.e. for 100 force curves the numbers are 0 to 99).

In [ ]:

```
expt_1.plot_raw(1)
```

Baseline

To adjust the baseline for each curve use the 'baseline' function. This works by taking the mean value over a specified area and setting this to zero. There are default values included to get you started:

In [ ]:

```
expt_1.baseline()
expt_2.baseline()
expt_3.baseline()
```

If you need to change the region used for baselining (for example, if there is noise on part of the approach) use the 'start_pos' and 'end_pos' inputs. Note these should be a fraction of the distance from the contact section of the curve towards the baseline. The default values are 'start_pos = 0.45' and 'end_pos = 0.8'. Change them using as follows:

In [ ]:

```
expt_1.baseline(start_pos = 0.5, end_pos = 0.7)
```

You can also choose to exculed curves with a noisy baseline. The code calculates teh standard deviation over the specified region. If this is above a specified value the curve will be deleted. You can do this speparatley for the approach and retract curves by setting 'max_approach_noise' and 'max_retract_noise'. The default value for both is 1 and this can be changed as follows:

In [ ]:

```
expt_1.baseline(max_approach_noise = 2, max_retract_noise = 4)
```

To change the baseline region and maximum noise together:

In [ ]:

```
expt_1.baseline(
    start_pos = 0.5,
    end_pos = 0.7,
    max_approach_noise = 2,
    max_retract_noise = 4
)
```

Contact point

To set surface height to zero, use the 'contact' function. Note this currently interpolates between the first point below zero and the previous point, so may have trouble for noisy data:

In [ ]:

```
expt_1.contact()
expt_2.contact()
expt_3.contact()
```

Plot adjusted curves

To view individual curves once they have been aligned, use the 'plot_adjusted' command, typing the number of the curve to plot in the brackets:

In [ ]:

```
expt_1.plot_adjusted(10)
```

You can also plot all of the curves and use a slider to cycle through them, using 'plot_curves':

In [ ]:

```
expt_3.plot_curves()
```

Delete bad curves

If a curve is noisy and needs to be removed, use the 'delete_curve' function. Put the numbers for all the curves to delete in the brackets:

In [ ]:

```
expt_1.delete_curve(8,59,87)
expt_2.delete_curve(17)
```

Calcluate adhesion

To calculate the adhesion for each curve, use 'calc_adhesion'. The mean and standard deviation for the adhesion calculated for each experiment will be printed in the output.

In [ ]:

```
expt_1.calc_adhesion()
expt_2.calc_adhesion()
expt_3.calc_adhesion()
```

It is possible to plot a histogram of the calculated adhesion values by setting 'plot_hist' to 'True':

In [ ]:

```
expt_2.calc_adhesion(plot_hist = True)
```

Calculate modulus

To caclculate the elastic modulus, use 'calc_modulus':

In [ ]:

```
expt_1.calc_modulus()
expt_2.calc_modulus()
expt_3.calc_modulus()
```

The method use currently follows the equations give here:
https://nanite.readthedocs.io/en/stable/sec_examples.html
(https://nanite.readthedocs.io/en/stable/sec_examples.html)

The default method uses as poisson ratio of 0.5 and indenter radius of 10 nm (10E-09). To set these manually use the 'poisson_ratio' and 'indenter_radius' inputs:

In [ ]:

```
expt_1.calc_modulus(poisson_ratio = 0.4, indenter_radius = 20 * 10e-9)
```

As with adhesion, it is possible to plot a histogram of the results:

In [ ]:

```
expt_2.calc_modulus(plot_hist = True)
```

Plot adhesion vs modulus

To view a scatter plot of adhesion vs modulus use the following command:

In [ ]:

```
expt_1.plot_adhesion_modulus()
```

Saving and reloading results

To save the calculate results, including force curves and parameters, use 'save_data'. This will open a window prompting you to select a folder and file name for saving. Leave the file extension blank.

In [ ]:

```
expt_1.save_data()
```

You can also specify the filename and folder in the brackets:

In [ ]:

```
expt_1.save_data('filename')
expt_2.save_data(r'\path\to\file')
```

To load saved data, create a new variable with the AFM class and call the 'load_data' function:

In [ ]:

```
from afm import AFM
loaded_data = AFM()
loaded_data.load_data()
```

Built in comparison plots

There are a number of functions built in for plotting comparisons of the adhesion and modulus data for separate experiments. These have been left as basic functions. For more specific plots, folow the custom plotting section below.

To use the comparison plots, call the functions listed below directly from the AFM class with input variables for each experiment as an input.

Histogram comparison of adhesion

In [ ]:

```
AFM.overlay_adhesion_hist(expt_1,expt_2,expt_3)
```

Histogram comparison of modulus

In [ ]:

```
AFM.overlay_modulus_hist(expt_1,expt_2,expt_3)
```

Box plot comparison of adhesion

In [ ]:

```
AFM.overlay_adhesion_box(expt_1,expt_2,expt_3)
```

Box plot comparison of modulus

In [ ]:

```
AFM.overlay_modulus_box(expt_1,expt_2,expt_3)
```

Bar chart (with error bars) comparison of adhesion

In [ ]:

```
AFM.overlay_adhesion_bar(expt_1,expt_2,expt_3)
```

Bar chart (with error bars) comparison of modulus

In [ ]:

```
AFM.overlay_modulus_bar(expt_1,expt_2,expt_3)
```

Custom plotting and analysis

The calculated variables are stored within each 'experiment' variable. These can be called when writing custom plotting or analysis scripts specific to your application. The variables available for each experiment are as follows:

In [ ]:

```
expt_1.approach         # All approach curve data (nN)
expt_1.retract          # All retract curve data (nN)
expt_1.z_position       # All tip position data (nm)

expt_1.approach[:,i]    # Adhesion curve for experiment i
expt_1.retract[:,i]     # Adhesion curve for experiment i
expt_1.z_position[:,i]  # Adhesion curve for experiment i

expt_1.def_sens         # Deflection sensetivity (nm/V)
expt_1.spr_const        # Spring constant (N/m)

expt_1.adheison         # Adhesion data (nN)
expt_1.modulus          # Modulus data (MPa)

expt_1.run_name         # Experiment name
```

Below is an example custom plotting script to overlay retract curves for multips experiments. This example uses plotly. There are a number of plotting tools in this package, and more detail (including example scripts) is available from https://plotly.com/python/ (https://plotly.com/python/)

In [ ]:

```
import  plotly.graph_objects  as      go
from    afm                   import  AFM

expt_1 = AFM()
expt_2 = AFM()

expt_1.run()
expt_2.run()

fig = go.Figure()

curve_number = 12

fig.add_trace(go.Scatter(y = expt_1.retract[:,curve_number],
                         x = expt_1.z_position[:,curve_number],
                         mode='lines',
                         name = 'Sample 1'))

fig.add_trace(go.Scatter(y = expt_2.retract[:,curve_number],
                         x = expt_2.z_position[:,curve_number],
                         mode='lines',
                         name = 'Sample 2'))

fig.update_layout(
    yaxis_title_text='Force (nN)',
    xaxis_title_text='Tip separation (nm)')

fig.show()
```

Example analysis script

Finally, below is an example script to carry out the above analysis for 3 experiments:

In [ ]:

```python
from afm  import  AFM

expt_1 = AFM()
expt_2 = AFM()
expt_3 = AFM()

expt_1.set_run_name('Sample 1')
expt_2.set_run_name('Sample 2')
expt_3.set_run_name('Sample 3')

expt_1.input_files()
expt_2.input_files()
expt_3.input_files()

expt_1.nanoscope_params()
expt_2.nanoscope_params()
expt_3.nanoscope_params()

expt_1.set_def_sens(100)
expt_2.set_def_sens(100)
expt_3.set_def_sens(100)

expt_1.set_spr_const(0.32)
expt_2.set_spr_const(0.32)
expt_3.set_spr_const(0.32)

expt_1.nanoscope_read()
expt_2.nanoscope_read()
expt_3.nanoscope_read()

expt_1.baseline()
expt_2.baseline()
expt_3.baseline()

expt_1.contact()
expt_2.contact()
expt_3.contact()

expt_1.plot_curves()
expt_2.plot_curves()
expt_3.plot_curves()

expt_1.calc_adhesion()
expt_2.calc_adhesion()
expt_3.calc_adhesion()

expt_1.calc_modulus()
expt_2.calc_modulus()
expt_3.calc_modulus()

expt_1.save_data()
expt_2.save_data()
expt_3.save_data()

AFM.overlay_adhesion_hist(expt_1,expt_2,expt_3)
AFM.overlay_modulus_bar(expt_1,expt_2,expt_3)
```