

**Supplementary Information:**

**Synergizing Human Expertise and AI Efficiency with Language Model for Microscopy Operation and Automated Experiment Design**

Yongtao Liu,<sup>1\*</sup> Marti Checa,<sup>1</sup> Rama K. Vasudevan<sup>1</sup>

<sup>1</sup> Center for Nanophase Materials Sciences, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA

\*Corresponding: [liuy3@ornl.gov](mailto:liuy3@ornl.gov)

**Supplementary Information includes:**

1. Supplementary Figures S1- S10
2. Supplementary Note I: Record of using ChatGPT to program AEcroscopy workflow.
3. Supplementary Note II: Record of using ChatGPT to interpret BEPFM data
4. Supplementary Note III: Record of using ChatGPT to program NanoSurf workflow
5. Supplementary Note IV: Record of using ChatGPT to program Zurich Instrument workflow

You

Please move the AFM tip to the center of the scanning region and set the setpoint to 2

ChatGPT

To move the AFM tip to the center of the scanning region and set the setpoint to 2 V using the AEcroscopy API, we can modify the `tip\_control()` function parameters accordingly. Assuming the scanning region is normalized (with coordinates ranging from -1 to 1), the center would typically be at `[0, 0]`. Here's the updated Python script:

```
python Copy code
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Acroscopy
from Acroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your BEPyAE.exe file

# Initialize BEPyAE with AR18 (set offline_development=True for offline mode)
newexp.init_BEPyAE(offline_development=True)

# Set tip parameters and move the tip
# Set the setpoint to 2 V and move the tip to the center of the scanning region
newexp.tip_control(tip_params_dict={"set_point_V_00": 2, "next_x_pos_00": 0, "next_y_pos_01": 0,
do_move_tip=True,
do_set_setpoint=True,
feedbackon=False})
```

```
do_set_setpoint=True,
feedbackon=False)
```

Key Points:

1. **Path to BEPyAE.exe:** Replace `r"C:\Path\To\BEPyAE.exe"` with the correct path to your `BEPyAE.exe` file.
2. **Setpoint and Position:**
  - \* The setpoint is now set to 2 V(`"set\_point\_V\_00": 2`).
  - \* The tip is moved to the center of the scanning region, which is assumed to be at `[0, 0]`(`"next\_x\_pos\_00": 0, "next\_y\_pos\_01": 0`).
3. **Offline Development Mode:** The `offline\_development` parameter is set to `True` for development or testing. Change it to `False` for actual measurements.

Ensure that your Python environment is properly set up with the AEcroscopy package and its dependencies, and that `BEPyAE.exe` is correctly located at the specified path. This script follows the guideline you provided and is tailored for a typical scanning setup. Adjustments may be needed depending on the specific configuration of your AFM system.

Figure S1. Converting individual commands to Python code using AE-GPT. Here the command is to set tip parameters. AE-GPT provided a comprehensive script accordingly.

 You  
How to perform a spiral scan with 5V applied via the tip

 ChatGPT

To perform a spiral scan with a voltage of 5V applied via the tip using the AEcroscopy API, you need to use the appropriate function for conducting a spiral scan and set the necessary parameters, including the voltage to be applied. However, the guideline you provided does not explicitly mention a function for a standard spiral scan with voltage application via the tip.

Typically, spiral scans and voltage application are separate functionalities. If AEcroscopy supports combining these actions (i.e., performing a spiral scan while applying a specific voltage via the tip), this would typically be done through a specialized function or by setting parameters in a spiral scan function.

Given the information available, I'll outline a general approach. Please note that you may need to adjust this code based on the specific functions and capabilities of AEcroscopy, as well as your experimental setup.

```
python  Copy code

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the Aecroscopy Acquisi↓
newexp = Acquisition(exe_path = "C:\Path\To\BEPyAE.exe") # Replace wi
```

```
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = "C:\Path\To\BEPyAE.exe") # Replace wi

# Initialize BEPyAE with ARIB (set offline_development=True for offline
newexp.init_BEPyAE(offline_development=True)

# Assuming a function like 'fpga_spiral_scan' exists and allows setting
# Set parameters for spiral scan with 5V applied via the tip
spiral_results = newexp.fpga_spiral_scan(spiral_parms_dict = {"spiral_v
# ... other parameters
},
# additional flags and parameters
)

# Note: Replace 'fpga_spiral_scan' and its parameters with the actual function and parameters
# based on AEcroscopy's capabilities for performing a spiral scan with 5V applied via tip
```

#### Instructions:

1. **Path to BEPyAE.exe:** Update the 'exe\_path' with the actual path to your 'BEPyAE.exe' file.
2. **Spiral Scan Parameters:** You need to replace 'fpga\_spiral\_scan' and its parameters ('spiral\_parms\_dict') with the actual function and parameters that AEcroscopy uses for spiral scans with voltage application. This is a placeholder based on typical function names and parameter structures.

Please review the AEcroscopy documentation or contact the developers/support team for precise details on performing such a specific task, especially if it involves combining spiral scanning with voltage application. The actual function names and parameter configurations might differ from this generalized example.

Figure S2. Converting natural language instruction to script. Here the command is to perform a spiral scan with 5V applied via tip.

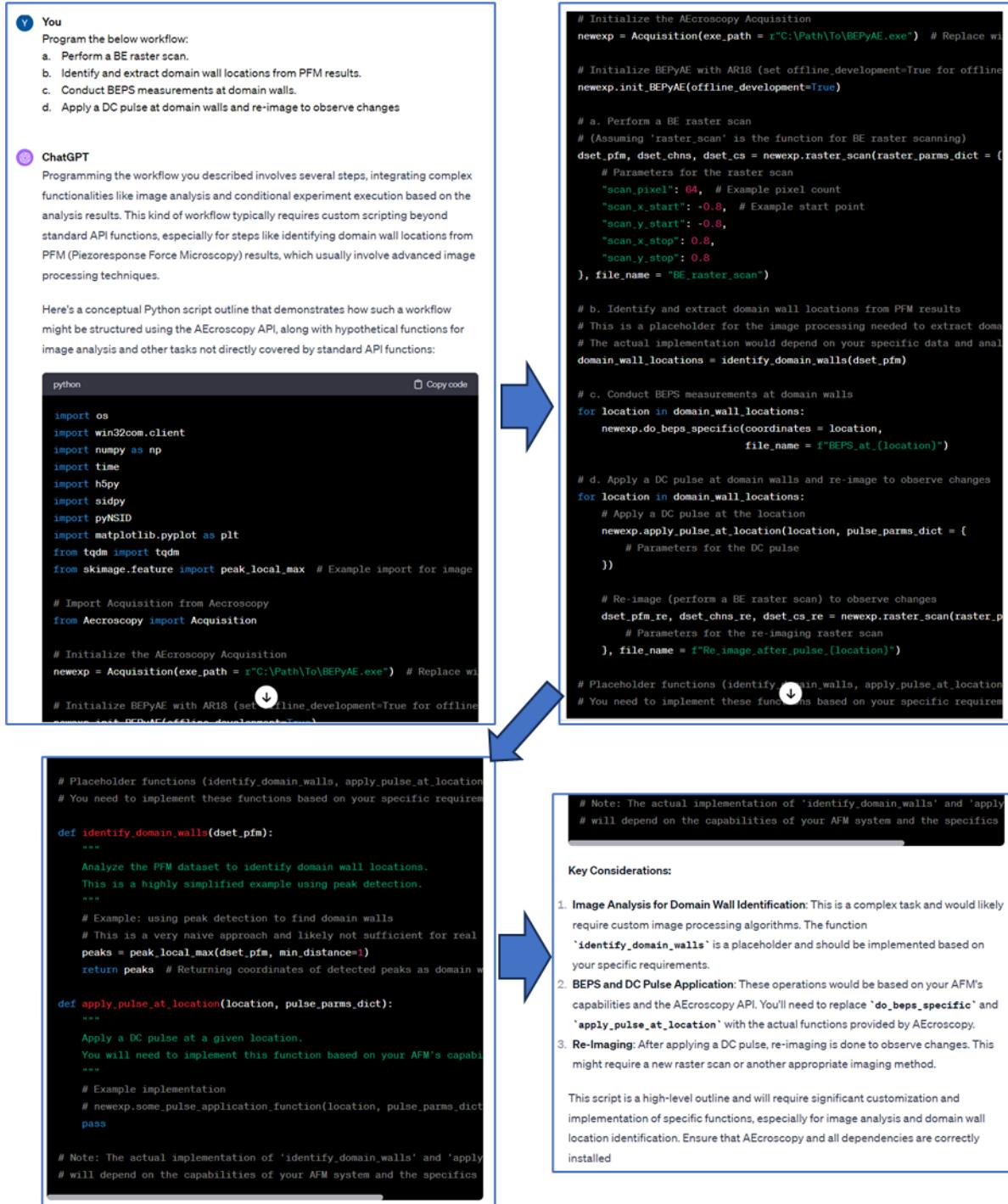


Figure S3. AE-GPT's ability to program a multifaceted workflow. In AE-GPT's response, it added placeholders for tasks required advanced analysis methods. Figure S4 shows how AE-GPT completes these tasks with human guidance.

You

Please define a function to identify domain walls using edge detector

ChatGPT

To define a function that identifies domain walls using an edge detection algorithm in Python, you can utilize image processing libraries like OpenCV or scikit-image. Edge detection is a common technique in image processing used to find boundaries or edges within images. It can be particularly useful for identifying domain walls in microscopy images, depending on the contrast and quality of the image.

Below is an example function that uses the Canny edge detector from the 'cv2' library (OpenCV) to identify edges in an image, which could correspond to domain walls:

```
python Copy code
import cv2
import numpy as np
import matplotlib.pyplot as plt

def identify_domain_walls(image, low_threshold=50, high_threshold=150,
    """
    Identify domain walls using the Canny edge detector.

    :param image: Input image (2D array).
    :param low_threshold: Lower threshold for the hysteresis procedure.
    :param high_threshold: Higher threshold for the hysteresis procedure.
    :param plot: If True, the function will plot the original and processed images.
    :return: Edge-detected image.
    """
    # Convert the image to grayscale if it's not already
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to smooth the image
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply the Canny edge detector
    edges = cv2.Canny(blurred_image, low_threshold, high_threshold)
```

```
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

# Apply the Canny edge detector
edges = cv2.Canny(blurred_image, low_threshold, high_threshold)

if plot:
    plt.figure(figsize=(12, 8))

    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(edges, cmap='gray')
    plt.title('Edges Detected')
    plt.axis('off')

    plt.show()

return edges

# Example usage:
# Load an image (replace 'path_to_image' with the path to your actual image)
image = cv2.imread('path_to_image', cv2.IMREAD_GRAYSCALE)

# Identify domain walls
edges = identify_domain_walls(image, plot=True)
```

```
# Identify domain walls
edges = identify_domain_walls(image, plot=True)
```

#### Key Points:

- \* This function uses the Canny edge detector, which is effective for many edge detection applications. It requires two threshold values, which you may need to adjust based on your specific images.
- \* The function includes an option to plot the original and edge-detected images for visualization purposes.
- \* The function assumes the input image is a 2D grayscale image. If it's a color image, the function converts it to grayscale.

#### Dependencies:

- \* You need to have OpenCV ('cv2') installed in your Python environment. You can install it via pip ('pip install opencv-python-headless' for a minimal installation without GUI functionalities).
- \* If your domain wall images have specific characteristics, you may need to fine-tune the parameters or use more sophisticated image processing techniques.

Remember that edge detection for identifying domain walls can be sensitive to noise and image quality. Additional image processing steps (like filtering or morphological operations) might be necessary for optimal results, depending on your data.

Figure S4. Human guided design for domain wall identification. Human suggests using edge detector for wall detection and AE-GPT employs the Canny filter for this task.

 You

Please define a function to apply pulse at locations using the tip control function and apply pulse function, here users can move tip to domain wall first and apply a pulse then

 ChatGPT

To define a function that moves the AFM tip to specified locations (presumably domain walls) and then applies a pulse at each location, we can create a function that utilizes the `tip\_control` function for moving the tip and another function, possibly named `apply\_pulse`, for applying the pulse. This function would be a part of the workflow using the AEcroscopy API, as outlined in your guide.

Here's an example of how such a function could be structured:

```
python  Copy code

def apply_pulse_at_domain_walls(acquisition, locations, pulse_params, tip_move_params):
    """
    Moves the AFM tip to specified locations and applies a pulse at each.

    :param acquisition: The Acquisition object from the AEcroscopy API.
    :param locations: List of coordinates where the pulse will be applied.
    :param pulse_params: Dictionary containing parameters for the pulse.
    :param tip_move_params: Dictionary containing parameters for moving the tip.
    """

    for location in locations:
        # Move the tip to the current location (domain wall)
        tip_params = tip_move_params.copy()
        tip_params.update({"next_x_pos_00": location[0], "next_y_pos_01": location[1]})

        acquisition.tip_control(tip_params, dict=pulse_params, do_move_tip=True)

        # Apply the pulse at the current location
        acquisition.apply_pulse(pulse_params, dict=pulse_params, do_create=True)

    # Example usage
    # Assuming 'newexp' is an instance of the Acquisition class and already
    # initialized with the necessary parameters.
```

```
# Apply the pulse at the current location
acquisition.apply_pulse(pulse_params, dict=pulse_params, do_create=True)

# Example usage
# Assuming 'newexp' is an instance of the Acquisition class and already
locations = [(x1, y1), (x2, y2), ...] # Replace with actual domain wall locations
pulse_params = {
    "pulse_amplitude_V": 5, # Example amplitude
    "pulse_duration_s": 0.02 # Example duration
    # ... other necessary pulse parameters ...
}
tip_move_params = {
    # Parameters for moving the tip
    "set_point_V_00": 1 # Example setpoint voltage
    # ... other necessary tip control parameters ...
}

apply_pulse_at_domain_walls(newexp, locations, pulse_params, tip_move_params)
```

**Key Points:**

- The function `apply\_pulse\_at\_domain\_walls` iterates over a list of locations, moves the tip to each location, and then applies a pulse.
- `locations` should be a list of tuples or lists, where each tuple/list represents the x and y coordinates of a domain wall location.
- `pulse\_params` and `tip\_move\_params` are dictionaries containing the necessary parameters for pulse application and tip movement, respectively. These should be defined based on the specific requirements of your experiment and the capabilities of your AFM system.
- The function assumes that `acquisition` (an instance of the `Acquisition` class from the AEcroscopy API) is already initialized and configured for use.

This is a conceptual example and might require adjustments to fit the specific functions and parameters of the AEcroscopy API you are using. Additionally, error handling and specific AFM behavior should be considered for robust and reliable operation.

Figure S5. Human guided design for applying pulse at domain walls. According to human instruction, AE-GPT programs a function of using tip\_control() and apply\_pulse() to move tip at domain wall and subsequently apply a DC pulse.

**You**  
Describe and program potential experiments in PFM to study ferroelectric domain wall dynamics using AEcrosoPy

**ChatGPT**  
To study ferroelectric domain wall dynamics using Piezoresponse Force Microscopy (PFM) with AEcroscopy, a variety of experiments can be designed. Here, I'll outline a few potential experiments and provide a conceptual Python programming approach for each using the AEcroscopy API. Remember that the specific code might need adjustments based on the exact capabilities and functions available in your version of AEcroscopy.

### Experiment 1: Domain Wall Imaging and Manipulation

**Objective:** Visualize domain walls and manipulate them using an electric field.

**Python Programming Outline:**

1. **PFM Imaging:** Perform initial PFM imaging to visualize the domain structure.
2. **Domain Manipulation:** Apply an electric field to manipulate domain walls.
3. **Post-manipulation Imaging:** Re-image the area to observe changes in domain structure.

```
python
from AECroscopy import Acquisition

# Initialize AEcroscopy Acquisition
newexp = Acquisition(exe_path='path_to_BEPyAE.exe')

# 1. PFM Imaging
# Parameters for initial PFM imaging
pfm_image_params = {
    # Parameters for PFM imaging
}
initial_pfm_image = newexp.pfm_scan(pfm_image_params)

# 2. Domain Manipulation
# Apply an electric field
# 3. Post-manipulation Imaging
# Re-image to observe changes
post_pfm_image = newexp.pfm_scan(pfm_image_params)
```

### Experiment 2: Frequency-Dependent Domain Wall Response

**Objective:** Investigate the frequency-dependent response of domain walls.

**Python Programming Outline:**

1. **Spectroscopic PFM:** Conduct spectroscopic PFM measurements at different frequencies across the domain wall.

```
python
# Spectroscopic PFM at various frequencies
frequency_ranges = [1e5, 2e5, 3e5, 4e5] # Example frequencies in Hz

for freq in frequency_ranges:
    spectroscopic_params = {
        'frequency': freq,
        # other parameters
    }
    response = newexp.spectroscopic_pfm(spectroscopic_params)
    # Process and store the response data
```

### Experiment 3: Time-Resolved Domain Wall Dynamics

**Objective:** Observe domain wall motion over time under an external stimulus.

**Python Programming Outline:**

1. **Time-Lapse PFM Imaging:** Perform PFM imaging over time while applying an external stimulus.

```
python
# Time-lapse PFM Imaging
time_points = range(0, 3600, 300) # Every 5 minutes for 1 hour

for time_point in time_points:
    # Apply stimulus if needed (e.g., electric field, temperature change)
    newexp.apply_stimulus(stimulus_params)

    # Perform PFM imaging
    time_lapse_image = newexp.pfm_scan(pfm_image_params)

    # Perform PFM imaging
    time_lapse_image = newexp.pfm_scan(pfm_image_params)
    # Store or process the image
```

### General Notes:

- \* **Path to BEPyAE:** Replace `path\_to\_BEpyAE.exe` with the actual path to your BEPyAE executable.
- \* **Function Parameters:** The functions `pfm\_scan`, `apply\_electric\_field`, `spectroscopic\_pfm`, and `apply\_stimulus` are placeholders. Replace them with the actual AEcroscopy API functions and appropriate parameters for your experiments.
- \* **Data Processing:** After each measurement, you may need to process the data to extract meaningful information about domain wall dynamics.
- \* **Error Handling:** Consider adding error handling to check the success of each operation and handle any exceptions that may occur during the experiment.

These outlines provide a starting point for programming PFM experiments to study ferroelectric domain wall dynamics using AEcroscopy. Tailor these scripts to fit the specific capabilities of your AEcroscopy setup and the details of the experiments you wish to conduct.

Figure S6. AE-GPT suggests three experiments for investigating ferroelectric domain walls dynamics using AEcrosoPy: domain wall imaging and manipulation, Frequency-dependent domain wall response, and time-resolved domain wall dynamics.

You

Please draft Python code that would reproduce the experiment detailed in the logger file.

ChatGPT

Based on the details from the logger file, the Python code to reproduce the AEcroscopy experiment would include the following steps:

1. Initialization and Import of Necessary Packages
2. Tip Control Setup
3. Input/Output (IO) Cluster Definition
4. Band Excitation (BE) Parameters Definition

Here is a draft of the Python code:

```

python
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Acroscopy
from Acroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = "path_to_BEPyAE.exe") # Replace with your actual path

# Tip Control
newexp.tip_control(tip_parms_dict={"set_point_V_00": 1, "next_x_pos_00": 0, "do_move_tip": True, "do_set_setpoint": True, "feedbackon": False})

# IO Cluster Definition
newexp.define_io_cluster(IO_cluster_parms_dict={"analog_output_amplifier": 1, "channel_01_type_07": 1, "channel_02_type_08": 2, "channel_03_type_09": 3})

```

You

log\_exp.txt  
Document

Summarize the AEcroscopy experiment detailed in the attached logger file using natural language for clarity.

ChatGPT

The AEcroscopy experiment detailed in the logger file involved three main steps:

1. **Tip Control:** The experiment began with controlling the AFM tip. The setpoint voltage was set to 1V, and the tip was moved to the position [0.5, 0.5]. Both moving the tip and setting the setpoint were executed successfully, with feedback turned off.
2. **IO Cluster Definition:** Next, the Input/Output (IO) cluster was defined. The parameters set included the analog output amplifier, and three different channel types. These IO settings were successfully applied with feedback enabled.
3. **Band Excitation (BE) Parameters Definition:** Finally, the parameters for band excitation (BE) were defined. This included setting the center frequency at 335 Hz, a bandwidth of 100 Hz, an amplitude of 1V, along with other parameters like phase variation, number of repeats, required pulse duration, and auto smooth ring. The creation of the BE waveform was executed with feedback turned off.

Each of these steps was crucial for the proper setup and execution of the experiment, ensuring precise control and measurement capabilities for the study.

```

# Tip Control
newexp.tip_control(tip_parms_dict={"set_point_V_00": 1, "next_x_pos_00": 0, "do_move_tip": True, "do_set_setpoint": True, "feedbackon": False})

# IO Cluster Definition
newexp.define_io_cluster(IO_cluster_parms_dict={"analog_output_amplifier": 1, "channel_01_type_07": 1, "channel_02_type_08": 2, "channel_03_type_09": 3})

# Band Excitation Parameters Definition
newexp.define_be_parms(be_parms_dict={"center_frequency_Hz_00": 335, "bandwidth_Hz_01": 100, "amplitude_V_02": 1, "phase_variation_03": 0, "repeats_04": 4, "req_pulse_duration_ms_05": 1, "auto_smooth_ring_06": 1, "do_create_be_waveform": True, "feedbackon": False})

```

Replace "`"path_to_BEPyAE.exe"`" with the actual path to your `"BEPyAE.exe"` file. This code assumes that the functions and parameters in the AEcroscopy API match those detailed in the logger file. Adjustments may be needed based on the specific version and configuration of the AEcroscopy package you are using.

Figure S7. AE-GPT writes code to reproduce experiment detailed in logger file.

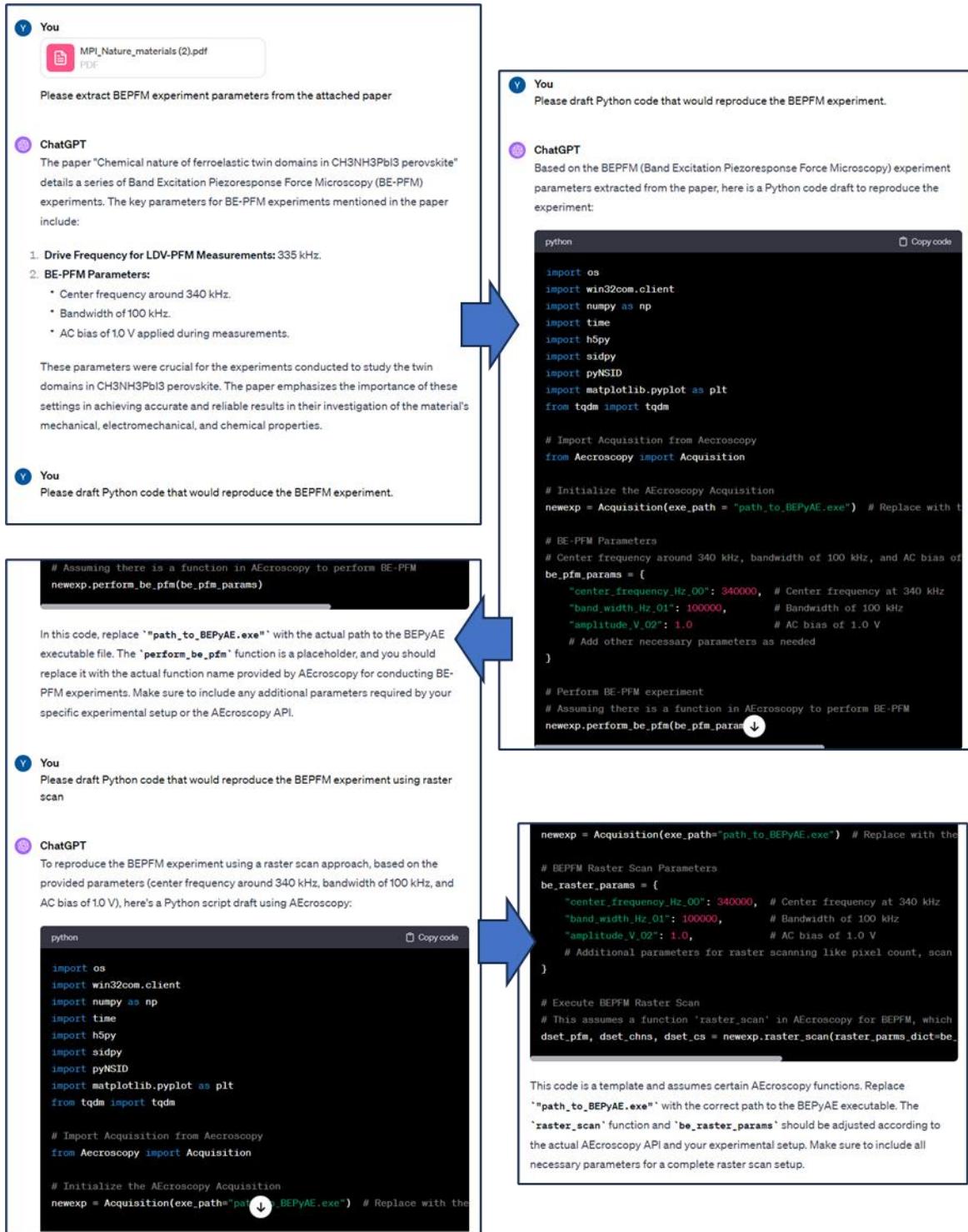


Figure S8. AE-GPT writes code to reproduce experiment reported in published paper.

```

sidpy.hdf.hdf_utils.print_tree(h5_file)

/
├ Measurement_000
│   -----
│   ├ Channel_000
│   │   -----
│   │   ├ Bin_FFT
│   │   ├ Bin_Frequencies
│   │   ├ Bin_Indices
│   │   ├ Bin_Step
│   │   ├ Bin_Wfm_Type
│   │   ├ Excitation_Waveform
│   │   ├ Noise_Floor
│   │   ├ Position_Indices
│   │   ├ Position_Values
│   │   ├ Raw_Data
│   │   └ Raw_Data-SHO_Fit_000
│   │       -----
│   │       ├ Guess
│   │       ├ Spectroscopic_Indices
│   │       ├ Spectroscopic_Values
│   │       ├ h5_sho_spec_inds_fixed
│   │       ├ h5_sho_spec_vals_fixed
│   │       └ Raw_Data-SHO_Fit_001
│   │           -----
│   │           ├ Fit
│   │           ├ Guess
│   │           ├ Spectroscopic_Indices
│   │           ├ Spectroscopic_Values
│   │           ├ completed_fit_positions
│   │           ├ completed_guess_positions
│   └ Spatially_Averaged_Plot_Group_000
        -----
        ├ Bin_Frequencies
        ├ Mean_Spectrogram
        ├ Spectroscopic_Parameter
        └ Step_Averaged_Response
        Spectroscopic_Indices
        Spectroscopic_Values
        UDVS
        UDVS_Indices
    └ Channel_001
        -----
        ├ Position_Indices
        ├ Position_Values
        ├ Raw_Data
        ├ Spectroscopic_Indices
        ├ Spectroscopic_Values
    └ Channel_002
        -----
        ├ Position_Indices
        ├ Position_Values
        ├ Raw_Data
        ├ Spectroscopic_Indices
        ├ Spectroscopic_Values
    └ Channel_003
        -----
        ├ Position_Indices
        ├ Position_Values
        ├ Raw_Data
        ├ Spectroscopic_Indices
        └ Spectroscopic_Values

```

Figure S9. The detailed structure of the BEPFM.h5 data uploaded to ChatGPT4, where Channel 0 is the piezoresponse data, Channel 1 is topography, and Channel 2 is deflection. Channel 3 is empty.

Figure S10-1

You  
BEPFM\_2um\_0006.h5

Please plot the amplitude and phase image in the attached file, note that it may need to be reshaped to 256\*256 before plotting. Please use 'jet' as map color.

ChatGPT  
The HDF5 file contains multiple datasets. I will examine the structure and contents of these datasets to determine which ones contain the amplitude and phase data required for plotting. After identifying the relevant datasets, I will reshape them to 256×256 and plot them using the 'jet' colormap as requested. Let's proceed with examining the datasets. [→]

Error analyzing.  
It appears that some of the items in the HDF5 file are groups, not datasets, which is why we encountered an error when trying to access their shape and data type. I will modify the approach to only examine the items that are datasets and ignore the groups. Let's try again. [→]

We have identified various datasets within the HDF5 file. To plot the amplitude and phase images, we typically look for datasets that contain amplitude and phase information. Based on the dataset information, it seems the relevant data might be in the 'Measurement\_000/Channel\_000/Raw\_Data-SHO\_Fit\_000/Guess' dataset, which contains amplitude and phase information.

I will now extract the amplitude and phase data from this dataset, reshape them to 256×256, and plot them using the 'jet' colormap. Let's proceed with this task. [→]

Amplitude      Phase

Here are the plots for the amplitude and phase data from the provided HDF5 file:

- On the left, you see the amplitude plot, reshaped to 256×256 and displayed using the 'jet' colormap.
- On the right, the phase plot is displayed, also reshaped to 256×256 and using the 'jet' colormap.

These visualizations should provide insights into the spatial variation of amplitude and phase in the dataset. If you need any further analysis or different visualizations, feel free to let me know! [→]

< 2 / 2

You  
What is the band excitation center frequency in this data

ChatGPT  
The 'UDVS' dataset contains an array with the shape (1, 7), indicating seven parameters. However, the values in this dataset are mostly 1s, with the first value being 0. This dataset doesn't seem to provide clear information about the band excitation center frequency.

Typically, band excitation center frequency can be derived from the spectroscopic values, such as those in the 'Spectroscopic\_Values' dataset, which often contains frequency-related data. I will check this dataset to see if we can find the center frequency information there. Let's proceed with this approach. [→]

The 'Spectroscopic\_Values' dataset contains an array with frequency values. The shape of this array is (1, 62), indicating 62 frequency values. These values are likely the frequencies used during the band excitation process.

To find the center frequency, we can calculate the average of the minimum and maximum frequency values from this dataset. Let's compute this center frequency. [→]

The band excitation center frequency in this data is approximately 374,267.59 Hz. This value is calculated as the average of the minimum and maximum frequency values found in the 'Spectroscopic\_Values' dataset. [→]

You  
found in the 'Spectroscopic\_Values' dataset. [→]

ChatGPT  
The data in channel1 is topography, please plot the image and calculate roughness.

Topography

You  
Is there piezo z sensitivity in the data?

ChatGPT  
There is no information about piezo z sensitivity in the provided HDF5 file, at least not under the path 'Measurement\_000/Channel\_000/Piezo\_Z\_Sensitivity'. Piezo z sensitivity data might be stored under a different name or path, or it might not be included in this particular file.

If you have specific knowledge about where this information might be located within the file structure, or if it is labeled differently, please let me know, and I can search for it accordingly. [→]

You  
Please use 3E-8 as the piezo z sensitivity and correct the topography value, then calculate roughness again

ChatGPT  
After correcting the topography values using the provided piezo z sensitivity of  $3 \times 10^{-8}$  meters per unit, the calculated roughness of the corrected topography is approximately  $1.35 \times 10^{-9}$  meters (or 1.35 nanometers).

This value represents the standard deviation of the height values in the corrected topography data, giving a measure of surface roughness at the nanoscale. [→]

You  
What is the excitation waveform of this data

ChatGPT  
Excitation Waveform

Figure S10-2

**You**

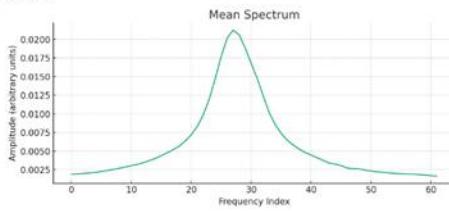
There are 256\*256 raw spectroscopic data here, please plot the mean spectrum, use the spectroscopic value as the frequency data.

This waveform is typically used in band excitation experiments to stimulate the sample or the measurement system. The exact nature of the waveform (e.g., sinusoidal, sawtooth, square, etc.) and its characteristics (e.g., frequency, amplitude) can be crucial for interpreting the measurement results and understanding the system's response. [–]

**You**

There are 256\*256 raw spectroscopic data here, please plot the mean spectrum

**ChatGPT**



Here is the plot of the mean spectrum derived from the raw spectroscopic data in the provided HDF5 file. This graph shows the average amplitude of the spectroscopic response across all 256\*256 data points for each frequency index, with the amplitude presented in arbitrary units.

The mean spectrum can provide insights into the average response characteristics of the sample under study, highlighting dominant frequencies or resonances. This is particularly useful in understanding material properties or system behaviors in spectroscopic measurements. [–]

**You**

There are 256\*256 raw spectroscopic data here, please plot the mean spectrum, use the spectroscopic value as the frequency data

**You**

There are 256\*256 raw spectroscopic data here, please plot the mean spectrum, use the spectroscopic value as the frequency data.

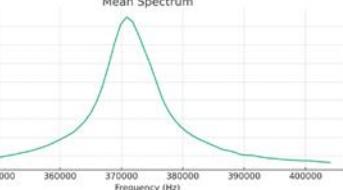
**ChatGPT**



It appears there was a mismatch in the dimensions between the frequency data and the mean spectrum data. The frequency data has 62 points, while the mean spectrum seems to have a different dimension.

I will re-examine the approach to correctly calculate the mean spectrum so that it matches the dimensions of the frequency data. Let's correct this and plot the mean spectrum again. [–]

**Mean Spectrum**



The mean spectrum plot, now correctly aligned with the frequency data, is displayed above. This graph shows the average amplitude response of the spectroscopic data across all 256\*256 points for each frequency, measured in Hertz.

This visualization is essential for understanding the overall spectral behavior of the sample, indicating dominant frequency responses and overall trends in the spectroscopic data. [–]

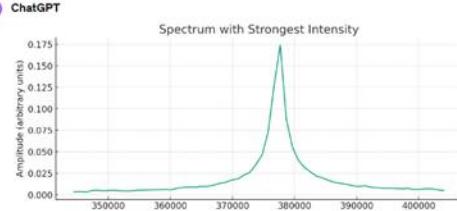
**You**

Please plot the spectrum with the strongest intensity

**You**

Please plot the spectrum with the strongest intensity

**ChatGPT**



The plot above represents the spectrum with the strongest intensity from the spectroscopic data. This graph shows the amplitude response at each frequency (in Hertz) for the specific data point that exhibited the highest overall amplitude.

This particular spectrum can provide insights into the resonant characteristics or other significant features of the sample at the point of strongest response. It's often useful in identifying specific behaviors or properties that are most pronounced in the dataset. [–]

Figure S10. The full process of AE-GPT analyzing a BEPFM dataset.

**Supplementary Note I: Record of using ChatGPT to program AEcroscopy workflow.**



**Anonymous**

Please study the AEcroscopy API guideline, then assist in drafting Python program according to the users' request. Below is the guideline:

```
# Experiments
```

```
## Guide to Essential Commands and Functionalities for Experiment in AEcroscopy
```

```
$_{Yongtao}$$$_{Liu,}$$$_{liuy3@ornl.gov}$
```

```
$_{June}$$$_{2023}$
```

Welcome to our comprehensive guide to the AEcroscopy for scanning probe microscopy (SPM). AEcroscopy is a dedicated Python package incorporating a wide range of commands and functions that facilitate basic microscope operations with BEPyAE.exe, enabling users to perform tasks with ease.

Here, we will introduce you to the essential Python commands and functions available within the AEcroscopy. We will explore their capabilities, syntax, and usage, empowering you to harness the full potential of the AEcroscopy-BEPyAE-SPM system.

Whether you are a seasoned Python developer or new to the world of programming, experienced SPM users or new to the microscope society, this chapter will serve as a foundation for your journey into the exciting realm of Python-controlled SPM.

By utilizing AEcroscopy, you will gain the ability to write and execute SPM experiments in Python, revolutionizing the way we perform the microscope experiments.

```
### Install and Import
```

First thing first, we need to install and import necessary packages, including AEcroscopy.

```
import os
```

First thing first, we need to install and import necessary packages, including Aecroscopy.

```
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

## import acquition.py
from Aecroscopy import Acquisition # include the Aecroscopy.py in the same directory

### Start BEPyAE.exe and set VI
```

Then, we need to start the Python API BEPyAE.exe.

- \* Start BEPyAE.exe
- \* Set VI of BEPyAE; if this version includes PyScanner, also set VIs for PyScanner

```
newexp = Acquisition(exe_path = r"C:\Users\yla\Dropbox (ORNL)\My
Files\AECroscopy_BEPyAE\BEPyAE_Labview\BEPyAE 060523 01\BEPyAE.exe") # exe_path is the
directory of BEPyAE;
```

### Initialize Igor AR18

Here, we connect BEPyAE with the microscope

- \* Set offline development. If you are doing offline development, set offline\_development=True, otherwise, if you are running microscope measurement, set offline\_development=False

Executing \$init\\_BEPyAE()\$ command will also:

- \* Build a connection between BEPyAE and AR18
- \* Get parameters in AR18

```
newexp.init_BEPyAE(offline_development = True) # set offline_development=True if doing offline
development
```

# executing this will also initialize AR18

```
development
    # executing this will also initlize AR18

>##### Hereinafter
>* if no parameter is input in a function, when you execute the function, it will take the default
parameters in BEPyAE.exe
>* Some functions print feedback after execution. This feedback can be turned off by setting
feedbackon = False. You can turn off feedback when you include this function in some iterations.
>* Note: Tip locations: -1 is the left handside for x-axis and bottom side for y-axis, 1 is the right handside
for x-axis and top side for y-axis
```

### ### Set tip parameters

Here, we use \$tip\\_control()\$ function to set setpoint.

\$tip\\_control()\$ function also allows us to move tip to a specific location [next\_x\_pos\_00, next\_y\_pos\_01], e.g., in the below example, the tip will be moved to locaiton [0.5, 0.5]

```
```{image} ../images/chapter1_image1.png
:width: 300px
:align: center
```

newexp.tip_control(tip_parms_dict = {"set_point_V_00": 1, "next_x_pos_00": 0.5, "next_y_pos_01": 0.5},
    do_move_tip = True,
    do_set_setpoint = True,
    feedbackon=False) # Executing this code will set setpoint to 1 V,
    # and move tip to location [0.5, 0.5]
```

### ### Set IO

In order for the Python code to provide the correct commands, it is essential to provide the hardware components (e.g., AFM platform, voltage amplifiers, channel data, etc.) involved in our experiment. Thus, here we set information about hardwares with function \$define\\_io\\_cluster()\$.

```
newexp.define_io_cluster(IO_cluster_parms_dict = {"analog_output_amplifier_06": 1,
    "channel_01_type_07": 1,
    "channel_02_type_08": 2,
    "channel_03_type_09": 3})
```

```
    channel_02_type_00 : 2,  
    "channel_03_type_09": 3})
```

#### #### Logger

Logger function will log any operations (including parameters) performed during an experiments.  
Reload the logger file also allows to reproduce previous experiment.

```
newexp._log
```

```
# Load previous experiment records  
f = open('log_exp.txt', 'r')  
exp = f.read()  
print(exp)
```

---

## BEPFM Measurement

#### #### Set BE pulse parameters

One of the most popular experiment we can perform with AEcroscopy-BEPyAE.exe is the band excitation (BE) piezoresponse force microscopy experiments. Here we use \$define\\_be\\_parms()\$ to set BE pulse parameters.

```
# set BE parameters  
newexp.define_be_parms(be_parms_dict = {"center_frequency_Hz_00": 335, "band_width_Hz_01": 100,  
                                         "amplitude_V_02": 1, "phase_variation_03": 1,  
                                         "repeats_04": 4, "req_pulse_duration_s_05": 4,  
                                         "auto_smooth_ring_06": 1},  
                         do_create_be_waveform = True, feedbackon=False)
```

#### #### BE Line scan

After setting BE pulse parameters, we can start to do BE line measurement with the function \$do\\_line\\_scan()\$. In this function, we will need to provide the BE line scan pixel (i.e., num\_BE\_Pulses\_01) and BE line scan locations (i.e. start location and stop location).

Note that

- \* This function is just a single BE line scan, not a raster image
- \* This function returns 5 datasets: quick\_fitting, complex spectra, and 3 channel data

```
# Do a sinale line scan
```

```
# Do a single line scan
qk_fit, com_spec, chn1, chn2, chn3 = newexp.do_line_scan(line_scan_parms_dict =
{"num_BE_pulses_01": 32,
 "start_x_pos_00": -0.5, "start_y_pos_01": 0,
 "stop_x_pos_02": 0.5, "stop_y_pos_03": 0},
 upload_to_daq = True, do_line_scan = True)
```

#### #### BE Raster Scan

We can also perform a square raster scan BE measurement using the function `$raster\_scan()`. In this function, we need to provide the raster scan pixel (i.e., `$scan\_pixel$`) and scan region (i.e., scan start and stop points).

Note that

- \* `raster_scan` returns 3 sidpy datasets: BEPFM quick fitting, channels, and BE complex spectra
- \* `raster_scan` also saves these 3 sidpy dataset in a h5 file named `$file\_name$`

#### # Do a 64\*64 raster scan

```
dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {"scan_pixel": 32,
 "scan_x_start": -0.8,
 "scan_y_start": -0.8, "scan_x_stop": 0.8,
 "scan_y_stop": 0.8}, file_name = "test")
```

```
# if you see below error, check if you set IO channels manually---this has to be done manually now
### TypeError: When specifying values over which a parameter is varied, values should not be an
empty array
```

#### #### BE Raster Scan Results

We can visualize BE raster scan results including quick fitting images and channel images.

```
# Quick fit BE images
print(dset_pfm) # sidpy dataset of BE quick fit

# plot BEPFM quick fit data
f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize = (20, 5), dpi = 100)
ax1.imshow(dset_pfm[:, :, 0])
```

```

# plot BEPFM quick fit data
f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize = (20, 5), dpi = 100)
ax1.imshow(dset_pfm[:, :, 0])
ax2.imshow(dset_pfm[:, :, 1])
ax3.imshow(dset_pfm[:, :, 2])
ax4.imshow(dset_pfm[:, :, 3])

# Channel images
print(dset_chns) # sidpy dataset of channels

# plot channel data
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (15, 5), dpi = 100)
ax1.imshow(dset_chns[0, :, :])
ax2.imshow(dset_chns[1, :, :])
ax3.imshow(dset_chns[2, :, :])

# sidpy dataset of complex spectra
dset_cs

```

We can also load the saved h5 file and analyze it after experiments

```

hf = h5py.File('test_0.hf5', 'r+')
sidpy.hdf.hdf_utils.print_tree(hf)

be = hf["BE Quick Fitting/Quick Fitting/Quick Fitting"]

be = np.asarray(be)
plt.imshow(be[:, :, 0])

```

---

#### BEPS Measurements

### Set BEPS parameters

In addition to BE piezoresponce force microscopy image measurement, we can also do BE piezoresponce spectroscopy (BEPs) measurement with the function \$do\\_BEPs\\_xxx()\$.

However, before doing BEPS measurement, we first need to define BEPS measurement parameters with the function \$define\\_BEPs\\_parameters()\$. In this function, we need to provide BEPS waveform parameters such as amplitude, steps, cycle numbers, etc.

However, before doing BEPS measurement, we first need to define BEPS measurement parameters with the function `$define\_BEPS\_parameters()`. In this function, we need to provide BEPS waveform parameters, such as amplitude, steps, cycle numbers, etc.

```
# Define BEPS parameters
newexp.define_BEPS_parameters(beps_parms_dict = {"amplitude_V_00": 8, "offset_V_01":0,
                                                 "read_voltage_V_02": 0, "step_per_cycle_03": 32,
                                                 "num_cycles_04": 3, "cycle_fraction_05": 0,
                                                 "cycle_phase_shift_06": 0, "measure_loops_07": 0,
                                                 "transition_time_s_08": 1E-3, "delay_after_step_s_09": 0},
                               do_create_waveform = True)
```

#### Do grid BEPS

After defining BEPS parameters, we can execute `$do\_BEPS\_grid()` function to perform a BEPS measurement. In the `$do\_BEPS\_grid()` function, we can provide grid pixel number in `$beps\_grid\_parms\_dict()`

```
# do BEPS
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_grid(beps_grid_parms_dict =
{"pixel_num_x": 5,
                                         "pixel_num_y": 5},
                                         file_name = "BEPS_grid")
```

Plot the BEPS waveform used in the experiments.

```
f, ax = plt.subplots(figsize = (3, 2))
ax.plot(beps_waveform)
ax.set_ylabel("Voltage (V)")
ax.set_xlabel("Step")
```

During experiments, BEPS loops can be visualized in `$BEPyAE.exe`. After experiments, we can also plot all BEPS loops as below.

```
# convert sidpy dataset to numpy array
quick_fit = np.asarray(beps_quick_fit)

# if measure_loops_07 is on and off, we need to separate on field and off field responses.
on_field_quick_fit = np.zeros((quick_fit.shape[0], quick_fit.shape[1], len(beps_waveform)))
off_field_quick_fit = np.zeros((quick_fit.shape[0], quick_fit.shape[1], len(beps_waveform)))
```

---

```

# if measure_loops_07 is on and off, we need to separate on field and off field responses.
on_field_quick_fit = np.zeros((quick_fit.shape[0], quick_fit.shape[1], len(beps_waveform)))
off_field_quick_fit = np.zeros((quick_fit.shape[0], quick_fit.shape[1], len(beps_waveform)))
for i in range (len(beps_waveform)):
    on_field_quick_fit[:, :, i] = quick_fit[:, :, 2*i]
    off_field_quick_fit[:, :, i] = quick_fit[:, :, 1+2*i]

##### Plot on field and off responses #####
idx = 0 # index of the loop to plot
f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize = (18, 3), dpi = 100)
f.subplots_adjust(wspace = 0.4)
ax1.set_title("Off Field, Amplitude vs. Voltage")
ax1.plot(beps_waveform, off_field_quick_fit[idx, 0, :])
ax1.set_xlabel("Voltage (V)")
ax1.set_ylabel("Amplitude (a.u.)")

ax2.set_title("Off Field, Phase vs. Voltage")
ax2.plot(beps_waveform, off_field_quick_fit[idx, 3, :])
ax2.set_xlabel("Voltage (V)")
ax2.set_ylabel("Phase (rad)")

ax3.set_title("On Field, Amplitude vs. Voltage")
ax3.plot(beps_waveform, on_field_quick_fit[idx, 0, :])
ax3.set_xlabel("Voltage (V)")
ax3.set_ylabel("Amplitude (a.u.)")

ax4.set_title("On Field, Phase vs. Voltage")
ax4.plot(beps_waveform, on_field_quick_fit[idx, 3, :])
ax4.set_xlabel("Voltage (V)")
ax4.set_ylabel("Phase (rad)")

### Do BEPS at a specific locations

```

In addition to BEPS measurement at grid locations, the \$do\\_BEPS\\_specific()\$ function also enables BEPS measurement at specified locations by providing specified coordinates to \$coordinates\$

```

# do BEPS at specific locations
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_specific(coordinates =
np.asarray([0.5, 0.5]),
                                file_name = "BEPs at specific location")

```

```
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_specific(coordinates =
np.asarray([0.5, 0.5]),
                                         file_name = "BEPS_at_specific_location")
```

However, when you want to want to measure a feature observed in an image, you often need to convert the location from the image to the coordinate for tip. Here we can use the \$convert\coordinates()\$ function.

```
# e.g. convert the location [100, 130] at an image to the coordinate of tip.
locations = np.asarray([100, 130])
converted_coor = newexp.convert_coordinates(original_coordinates = locations, num_pix_x = 256,
num_pix_y = 256)
print(converted_coor)
```

After we have the coordinate for tip location, we can use the \$do\BEPS\specific()\$ to perform BEPS measurement here

```
# do BEPS at specific locations
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_specific(coordinates =
converted_coor,
                                         file_name = "BEPS_at_specific_location")
```

---

```
### Apply a pulse
```

In addition to characterization, SPM also allows to manipulate objects. For example, applying a DC pulse via SPM tip to a ferroelectric film allows to flip the ferroelectric polarization.

In AECroscopy, we can apply a pulse by using \$define\apply\pulse()\$ function. We can set the pulse amplitude and pulse duration in this function, as shown below.

```
# Apply DC pulse
pulse, time = newexp.define_apply_pulse(pulse_parms_dict = {"pulse_init_amplitude_V_00": 0,
"pulse_mid_amplitude_V_01": 3,
"pulse_final_amplitude_V_02": 0, "pulse_on_duration_s_03": 20E-3,
"rise_time_s_05": 1E-3, "pulse_final_duration_s_04": 10E-3,
"pulse_repeats_06": 3},
```

```

        "pulse_final_amplitude_V_02": 0, "pulse_on_duration_s_03": 20E-3,
        "rise_time_s_05": 1E-3, "pulse_final_duration_s_04": 10E-3,
        "pulse_repeats_06": 3},
    do_create_pulse = True, do_upload_pulse = True, do_apply_pulse = False)
# Plot applied pulse
f, ax = plt.subplots(figsize = (3, 2))
f.suptitle("Applied Pulse")
ax.plot(time, pulse)
ax.set_xlabel("Time (s)")
ax.set_ylabel("Voltage (V)")

#### Progress bar
```

AEcroscopy is designed for automated and autonomous microscopy measurements, so we also include a progress function `$progress_bar()` that assists users to track the experiment progress. For example, when we perform 5 iterations BEPS measurement, we can use the progress bar to track the experiment progress as below.

```

iteration = 5
progress_bar = newexp.progress_bar(max_value = iteration)
for i in range (iteration):
    newexp.define_BEPS_parameters(do_VS_waveform = True, feedbackon = False)
    progress_bar.update(i)
```

#### Capability with FPGA

In AEcroscopy-BEPyAE.exe system, users have various options of scan trajectories when equipped with FPGA, such as spiral scan or any customized trajectories.

#### #### Spiral Scan

Here we will show how to do spiral scan with the `$fpga\_\spiral\_\scan()` function. Using this function, we set the spiral parameters, e.g., inner radius, outer radius, spiral cycles, duration, etc; the function will return the spiral result and save the result as a H5 file.

```

spiral_results = newexp.fpga_spiral_scan(spiral_parms_dict = {"spiral_inner_radius_x_V_00": 0,
    "spiral_outer_radius_x_V_01": 2,
    "spiral_inner_radius_y_V_02": 0, "spiral_outer_radius_y_V_03": 2,
    "spiral_N_cycles_04": 5, "spiral_duration_05": 5}
```

```

        "scan_y_offset_V": 0, "scan_rotation_deg": 0},
do_scan_update = True, do_scan = True,
file_name = "spiral_scan")

```

### ### BE Spiral Scan

If we apply BE excitation waveform during spiral scan, we can also perform BE spiral measurement. The function to perform BE spiral measurement is \$fpga\\_spiral\\_scan\\_BE()\$. The BE waveform parameters can be set either in this function or beforehand.

```

# set BE parameters, set spiral parameters, and do spiral BE
fpga_results, be_results = newexp.fpga_spiral_scan_BE(be_parms_dict = {"center_frequency_Hz_00": 335, "band_width_Hz_01": 100,
                                                                    "amplitude_V_02": 1, "phase_variation_03": 1,
                                                                    "repeats_04": 4, "req_pulse_duration_s_05": 4},
                                                       do_create_be_waveform = True,
                                                       spiral_parms_dict = {"spiral_inner_radius_x_V_00": 0,
                                                                    "spiral_outer_radius_x_V_01": 0.5,
                                                                    "spiral_inner_radius_y_V_02": 0,
                                                                    "spiral_outer_radius_y_V_03": 0.5,
                                                                    "spiral_N_cycles_04": 10},
                                                       num_BE_pulse = 128, do_scan_update = True, do_BE_arb_line_update_00 =
True,
                                                       do_BE_arb_line_scan_01 = True, spiral_reconstruction = False)

```

### ### Tip Control by FPGA

FPGA can also drive the probe to a specific location with the function \$fpga\\_tip\\_control()\$. We need to input the tip location parameters when using this function, as shown below, executing the below command will move tip from [0, 0] to [0.5, 0.5].

```

newexp.fpga_tip_control(fpga_tip_parms_dict={"strat_x_position_V_00": 0, "strat_y_position_V_01": 0,
                                              "final_x_position_V_02": 0.5, "final_y_position_V_03": 0.5},
                           make_cur_pos_start_pos=False, do_probe_move_update=True, do_probe_move=True)

```

If we are moving the tip from current location, we can set the \$make\\_cur\\_pos\\_start\\_pos = True\$ and provide the final location, as shown below.

```

newexp.fpga_tip_control(fpga_tip_parms_dict={"final_x_position_V_02": 0.5, "final_y_position_V_03": 0.5},

```

```
"final_x_position_V_02": 0.5, "final_y_position_V_03": 0.5},  
make_cur_pos_start_pos=False, do_probe_move_update=True, do_probe_move=True)
```

If we are moving the tip from current location, we can set the `$make\_cur\_pos\_start\_pos = True$` and provide the final location, as shown below.

```
newexp.fpga_tip_control(fpga_tip_parms_dict={"final_x_position_V_02": 0.5, "final_y_position_V_03": 0.5},  
                         make_cur_pos_start_pos=True, do_probe_move_update=True, do_probe_move=True)
```

### ### FPGA Driven Line by Line Raster Scan

We can also use \$fpga\\_linebyline\\_raster\\_scan()\$ function to perform slow raster scan or line by line scan.

\* when we set \$do\\_full\\_raster\\_scan = True\$, \$wait\\_to\\_advance\\_to\\_next\\_line = False\$, and \$do\\_next\\_raster\\_line\\_only = False\$, this function can be used to perform a full raster scan, as shown below.

```
raster_full = newexp.fpga_linebyline_raster_scan(line_by_line_raster_dict = {"raster_scan_size_x_V_00":  
1,  
"raster_scan_size_y_V_01": 1,  
"raster_N_scan_lines_02": 64,  
"raster_line_duration_s_03": 1,  
"scan_x_offset_V_04": 0,  
"scan_y_offset_V_05": 0,  
"scan_rotation_deg_06": 0},  
initialize_line_by_line_raster=True, do_full_raster_scan=True,  
wait_to_advance_to_next_line=False, do_next_raster_line_only=False,  
stop_full_raster_scan=False)
```

\* Alternatively, when we set \$do\\_full\\_raster\\_scan = False\$, \$wait\\_to\\_advance\\_to\\_next\\_line = True\$, and \$do\\_next\\_raster\\_line\\_only = True\$, this function can be used to perform scan line by line, as shown below.

```
        "raster_N_scan_lines_02": 64,
        "raster_line_duration_s_03": 1,
        "scan_x_offset_V_04": 0,
        "scan_y_offset_V_05": 0,
        "scan_rotation_deg_06": 0},
    initialize_line_by_line_raster=True, do_full_raster_scan=False,
    wait_to_advance_to_next_line=True, do_next_raster_line_only=True,
    stop_full_raster_scan=False)
```



### ChatGPT

The AEcroscopy API guideline you've provided is a comprehensive guide for performing scanning probe microscopy (SPM) experiments using the AEcroscopy Python package. It covers a wide range of functionalities, from basic setup and initialization to conducting advanced experiments like BE (Band Excitation) piezoresponse force microscopy, BEPS (Band Excitation Piezoresponse Spectroscopy), and various scanning methods using FPGA (Field-Programmable Gate Array).

Based on this guideline, we can create a Python program to perform a specific SPM experiment using AEcroscopy. To help you better, could you please specify the type of experiment or task you want to accomplish with this program? For example, do you want to perform a basic setup and image acquisition, conduct a BE experiment, or execute a more complex task like BEPS measurements or FPGA-driven scans? Your specific requirements will help in drafting an appropriate Python script for your needs.



### Anonymous

Please show the Python code to move the AFM tip



### ChatGPT

Certainly! To move the Atomic Force Microscopy (AFM) tip using the AEcroscopy API as per the guidelines you provided, we can write a Python script that uses the `tip\_control()` function from the AEcroscopy package. This function allows you to control various parameters of the AFM tip, including its position.

Here's a basic Python script to move the AFM tip to a specific location:

```
python
import os
```

Copy code

python

 Copy code

```
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Set tip parameters and move the tip
# Here we are setting the setpoint to 1 V and moving the tip to location [0.5, 0.5]
newexp.tip_control(tip_parms_dict={"set_point_V_00": 1, "next_x_pos_00": 0.5, "next_y_pos_01": 0.5,
                                    do_move_tip=True,
                                    do_set_setpoint=True,
                                    feedbackon=False})
```

#### Instructions:

1. **Path to BEPyAE.exe:** Replace `r"C:\Path\To\BEPyAE.exe"` with the actual path where `BEPyAE.exe` is located on your system.
2. **Setpoint and Position:** The `tip\_parms\_dict` contains parameters for controlling the tip. You can adjust the values of `set\_point\_V\_00`, `next\_x\_pos\_00`, and `next\_y\_pos\_01` as needed.
  - `set\_point\_V\_00`: Setpoint voltage for the tip.
  - `next\_x\_pos\_00` and `next\_y\_pos\_01`: The x and y coordinates where you want to move the tip.
3. **Offline Development:** If you're running this script for offline development or testing, make sure

- `'"next\_x\_pos\_00"'` and `'"next\_y\_pos\_01"'`: The x and y coordinates where you want to move the tip.
3. **Offline Development:** If you're running this script for offline development or testing, make sure `offline\_development` is set to `True`. For actual microscope measurements, set it to `False`.

Remember that this script is based on the guideline you provided and assumes that you have the AEcroscopy package and all its dependencies properly installed and configured in your Python environment. Additionally, the actual path to `BEPyAE.exe` needs to be specified correctly for the script to work.

 **Anonymous**

Please move the AFM tip to the center of the scanning region and set the setpoint to 2

 **ChatGPT**

To move the AFM tip to the center of the scanning region and set the setpoint to 2 V using the AEcroscopy API, we can modify the `tip\_control()` function parameters accordingly. Assuming the scanning region is normalized (with coordinates ranging from -1 to 1), the center would typically be at `[0, 0]`. Here's the updated Python script:

```
python
Copy code

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Set tip parameters and move the tip
```

```
newexp.init_BEPyAE(offline_development=True)

# Set tip parameters and move the tip
# Set the setpoint to 2 V and move the tip to the center of the scanning region ([0, 0])
newexp.tip_control(tip_parms_dict={"set_point_V_00": 2, "next_x_pos_00": 0, "next_y_pos_01": 0,
                                    do_move_tip=True,
                                    do_set_setpoint=True,
                                    feedbackon=False})
```

#### Key Points:

1. **Path to BEPyAE.exe:** Replace `r"C:\Path\To\BEPyAE.exe"` with the correct path to your `BEPyAE.exe` file.
2. **Setpoint and Position:**
  - The setpoint is now set to 2 V (`"set\_point\_V\_00": 2`).
  - The tip is moved to the center of the scanning region, which is assumed to be at `[0, 0]` (`"next\_x\_pos\_00": 0, "next\_y\_pos\_01": 0`).
3. **Offline Development Mode:** The `offline\_development` parameter is set to `True` for development or testing. Change it to `False` for actual measurements.

Ensure that your Python environment is properly set up with the AEcroscopy package and its dependencies, and that `BEPyAE.exe` is correctly located at the specified path. This script follows the guideline you provided and is tailored for a typical scanning setup. Adjustments may be needed depending on the specific configuration of your AFM system.



#### Anonymous

Provide Python code for conducting a BE line scan measurement from point [-1, -1] to [1, 1] with a center frequency of 400kHz



#### ChatGPT

To conduct a Band Excitation (BE) line scan measurement using the AEcroscopy API, as per your request, we need to set up both the line scan parameters and the BE parameters. For your specific case, we'll set the line scan to go from point `[-1, -1]` to `[1, 1]` and set the center frequency of the BE pulse to `400 kHz`.

Here's the Python code to achieve this:

Here's the Python code to achieve this:

```
python
Copy code

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Set BE parameters
# Set the center frequency to 400 kHz (400000 Hz)
newexp.define_be_parms(be_parms_dict = {"center_frequency_Hz_00": 400000, "band_width_Hz_01": 10000, "amplitude_V_02": 1, "phase_variation_03": 1, "repeats_04": 4, "req_pulse_duration_s_05": 4, "auto_smooth_ring_06": 1}, do_create_be_waveform = True, feedbackon=False)

# Conduct a BE line scan
# The line scan will go from [-1, -1] to [1, 1]
qk_fit, com_spec, chn1, chn2, chn3 = newexp.do_line_scan(line_scan_parms_dict = {"num_BE_p": 100, "start_x_px_07": -1, "stop_x_px_08": 1, "start_y_px_09": -1, "stop_y_px_10": 1, "upload_to_daq": True, "do_line_scans": True})
```

### Instructions:

1. **Path to BEPyAE.exe:** Update the `exe\_path` with the actual path to your `BEPyAE.exe` file.
2. **BE Parameters:** The `define\_be\_parms` function is set with a center frequency of 400 kHz. You may need to adjust other parameters like `band\_width\_Hz\_01`, `amplitude\_V\_02`, etc., based on your specific requirements.
3. **Line Scan Parameters:** The `do\_line\_scan` function's parameters are set for a scan from `[-1, -1]` to `[1, 1]`. The `num\_BE\_pulses\_01` parameter (number of BE pulses) can be adjusted based on your requirements.

Ensure that all dependencies (like AEcroscopy package, win32com, etc.) are correctly installed in your Python environment. Also, adjust any parameters according to the specific requirements of your AFM system and the BE line scan experiment.



#### Anonymous

How to perform a spiral scan with 5V applied via the tip



#### ChatGPT

To perform a spiral scan with a voltage of 5V applied via the tip using the AEcroscopy API, you need to use the appropriate function for conducting a spiral scan and set the necessary parameters, including the voltage to be applied. However, the guideline you provided does not explicitly mention a function for a standard spiral scan with voltage application via the tip.

Typically, spiral scans and voltage application are separate functionalities. If AEcroscopy supports combining these actions (i.e., performing a spiral scan while applying a specific voltage via the tip), this would typically be done through a specialized function or by setting parameters in a spiral scan function.

Given the information available, I'll outline a general approach. Please note that you may need to adjust this code based on the specific functions and capabilities of AEcroscopy, as well as your experimental setup.

python

Copy code

```
import os
import win32com.client
import numpy as np
```

```

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Assuming a function like 'fpga_spiral_scan' exists and allows setting a tip voltage
# Set parameters for spiral scan with 5V applied via the tip
spiral_results = newexp.fpga_spiral_scan(spiral_parms_dict = {"spiral_voltage_V": 5,
                                                               # ... other necessary parameters
                                                               },
                                         # additional flags and parameters as needed
                                         )

# Note: Replace 'fpga_spiral_scan' and its parameters with the actual function name and parameters
# based on AEcroscopy's capabilities for performing a spiral scan with voltage application

```

**Instructions:**

1. **Path to BEPyAE.exe:** Update the `exe\_path` with the actual path to your `BEPyAE.exe` file.
2. **Spiral Scan Parameters:** You need to replace `fpga\_spiral\_scan` and its parameters (`spiral\_parms\_dict`) with the actual function and parameters that AEcroscopy uses for spiral scans with voltage application. This is a placeholder based on typical function names and parameter structures.

Please review the AEcroscopy documentation or contact the developers/support team for precise details on

voltage application. This is a placeholder based on typical function names and parameter structures.

Please review the AEcroscopy documentation or contact the developers/support team for precise details on performing such a specific task, especially if it involves combining spiral scanning with voltage application. The actual function names and parameter configurations might differ from this generalized example.



### Anonymous

Program a spiral scan for tracking domain wall evolution, including applying a DC pulse at domain walls between scans.



### ChatGPT

To program a spiral scan for tracking domain wall evolution, including applying a DC pulse at domain walls between scans using AEcroscopy API, you will need to combine multiple steps: performing a spiral scan, identifying domain walls, and applying a DC pulse at those locations. This process involves a sequence of operations which might include image analysis to locate domain walls, and then applying pulses based on those locations.

Here's a conceptual outline of how such a program might look, assuming AEcroscopy provides the necessary functions for these operations. The specifics would depend on the exact capabilities and functions of the AEcroscopy API, which might require adjustments to this code:

python

Copy code

```
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path
```

```

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Perform a spiral scan to track domain wall evolution
# (Assuming 'fpga_spiral_scan' or a similar function exists for this purpose)
spiral_scan_results = newexp.fpga_spiral_scan(spiral_parms_dict = {
    # ... appropriate parameters for the spiral scan ...
})

# Process the scan results to identify domain wall locations
# This step would likely involve image processing techniques
# and is highly dependent on the specifics of your data and domain wall characteristics
domain_wall_locations = identify_domain_walls(spiral_scan_results) # This is a placeholder

# Apply a DC pulse at the identified domain wall locations
# Loop through each identified domain wall location
for location in domain_wall_locations:
    # Apply pulse at the location
    # Assuming a function like 'apply_pulse_at_location' exists for this purpose
    newexp.apply_pulse_at_location(location, pulse_parms_dict = {
        "pulse_amplitude_V": 5, # Example amplitude
        "pulse_duration_s": 0.02 # Example duration
        # ... other necessary parameters ...
    })

    # Optionally, perform another spiral scan after applying the pulse
    # if needed to observe changes

# Note: 'identify_domain_walls' and 'apply_pulse_at_location' are hypothetical functions.
# The actual implementation will depend on the available functions and capabilities of AE

```

#### Key Points:

1. **Spiral Scan:** Replace `fpga\_spiral\_scan` with the actual function name and parameters for performing a spiral scan.

1. **Spiral Scan:** Replace `fpga\_spiral\_scan` with the actual function name and parameters for performing a spiral scan.
2. **Domain Wall Identification:** This is a complex task typically involving image processing. The function `identify\_domain\_walls` is a placeholder and would need to be implemented based on your specific requirements and data.
3. **Applying DC Pulse:** Replace `apply\_pulse\_at\_location` with the actual function name and parameters for applying a DC pulse. The parameters will depend on your specific requirements for the pulse.

This outline assumes a certain level of automation and analysis capability within AEcroscopy which might not be present. You may need to perform some steps manually or develop custom scripts for image analysis. Additionally, consult the AEcroscopy documentation for exact function names, parameters, and capabilities.



#### Anonymous

Perform a 10x10 grid BEPS measurement with 3 cycles of drive waveform and an 8V DC amplitude



#### ChatGPT

To perform a 10x10 grid BEPS (Band Excitation Piezoresponse Spectroscopy) measurement with 3 cycles of a drive waveform and an 8V DC amplitude using the AEcroscopy API, you would typically follow these steps:

1. Define BEPS parameters including the amplitude, the number of cycles, and other waveform characteristics.
2. Perform a grid BEPS measurement over the specified grid size.

Based on the AEcroscopy API guideline you provided, here's a Python script that should accomplish this task:

```
python
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from AEcroscopy
```

```

from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# Define BEPS parameters
newexp.define_BEPS_parameters(beps_parms_dict = {
    "amplitude_V_00": 8,                                # 8V DC amplitude
    "offset_V_01": 0,                                   # Offset voltage
    "read_voltage_V_02": 0,                             # Read voltage
    "step_per_cycle_03": 32,                            # Steps per cycle
    "num_cycles_04": 3,                                # 3 cycles of the drive waveform
    "cycle_fraction_05": 0,                            # Cycle fraction
    "cycle_phase_shift_06": 0,                           # Cycle phase shift
    "measure_loops_07": 0,                            # Measurement loops
    "transition_time_s_08": 1E-3,                      # Transition time
    "delay_after_step_s_09": 0,                         # Delay after each step
},
do_create_waveform = True)

# Perform a 10x10 grid BEPS measurement
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_grid(beps_grid_parms_dict = {
    "pixel_num_x": 10,      # 10 pixels along x-axis
    "pixel_num_y": 10       # 10 pixels along y-axis
},
file_name = "BEPS_grid_measurement")

# Note: This code assumes that the functions and parameters as described in your guide are
# You may need to adjust parameter names and values based on the exact version and capability

```

**Instructions:**

#### Instructions:

1. **Path to BEPyAE.exe:** Make sure you replace `r"C:\Path\To\BEPyAE.exe"` with the actual path to your `BEPyAE.exe` file.
2. **BEPS Parameters:** Set the required parameters for the BEPS measurement in `define\_BEPS\_parameters`. Adjust these parameters as needed for your specific experiment.
3. **Grid Measurement:** The `do\_beps\_grid` function is configured to perform a measurement over a 10x10 grid.

Ensure that AEcroscopy and all its dependencies are properly installed in your Python environment. The script is based on the guideline you provided; however, the actual function names and parameters might differ based on the version and configuration of your AEcroscopy package. Always refer to the latest documentation for accurate and detailed information.



#### Anonymous

Program the below workflow:

- a. Perform a BE raster scan.
- b. Identify and extract domain wall locations from PFM results.
- c. Conduct BEPS measurements at domain walls.
- d. Apply a DC pulse at domain walls and re-image to observe changes



#### ChatGPT

Programming the workflow you described involves several steps, integrating complex functionalities like image analysis and conditional experiment execution based on the analysis results. This kind of workflow typically requires custom scripting beyond standard API functions, especially for steps like identifying domain wall locations from PFM (Piezoresponse Force Microscopy) results, which usually involve advanced image processing techniques.

Here's a conceptual Python script outline that demonstrates how such a workflow might be structured using the AEcroscopy API, along with hypothetical functions for image analysis and other tasks not directly covered by standard API functions:

```
python
```

Copy code

```
import os
import win32com.client
import numpy as np
```

```
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm
from skimage.feature import peak_local_max # Example import for image processing

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = r"C:\Path\To\BEPyAE.exe") # Replace with your actual path

# Initialize BEPyAE with AR18 (set offline_development=True for offline development)
newexp.init_BEPyAE(offline_development=True)

# a. Perform a BE raster scan
# (Assuming 'raster_scan' is the function for BE raster scanning)
dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {
    # Parameters for the raster scan
    "scan_pixel": 64, # Example pixel count
    "scan_x_start": -0.8, # Example start point
    "scan_y_start": -0.8,
    "scan_x_stop": 0.8,
    "scan_y_stop": 0.8
}, file_name = "BE_raster_scan")

# b. Identify and extract domain wall locations from PFM results
# This is a placeholder for the image processing needed to extract domain wall locations
# The actual implementation would depend on your specific data and analysis requirements
domain_wall_locations = identify_domain_walls(dset_pfm)

# c. Conduct BEPS measurements at domain walls
for location in domain_wall_locations:
    newexp.do_beps_specific(coordinates = location,
```

```

for location in domain_wall_locations:
    newexp.do_beps_specific(coordinates = location,
                           file_name = f"BEPS_at_{location}")

# d. Apply a DC pulse at domain walls and re-image to observe changes
for location in domain_wall_locations:
    # Apply a DC pulse at the location
    newexp.apply_pulse_at_location(location, pulse_parms_dict = {
        # Parameters for the DC pulse
    })

    # Re-image (perform a BE raster scan) to observe changes
    dset_pfm_re, dset_chns_re, dset_cs_re = newexp.raster_scan(raster_parms_dict = {
        # Parameters for the re-imaging raster scan
    }, file_name = f"Re_image_after_pulse_{location}")

# Placeholder functions (identify_domain_walls, apply_pulse_at_location)
# You need to implement these functions based on your specific requirements

def identify_domain_walls(dset_pfm):
    """
    Analyze the PFM dataset to identify domain wall locations.
    This is a highly simplified example using peak detection.
    """
    # Example: using peak detection to find domain walls
    # This is a very naive approach and likely not sufficient for real data
    peaks = peak_local_max(dset_pfm, min_distance=1)
    return peaks # Returning coordinates of detected peaks as domain wall locations

def apply_pulse_at_location(location, pulse_parms_dict):
    """
    Apply a DC pulse at a given location.
    You will need to implement this function based on your AFM's capabilities.
    """
    # Example implementation
    # newexp.some_pulse_application_function(location, pulse_parms_dict)
    pass

# Note: The actual implementation of 'identify_domain_walls' and 'apply_pulse_at_location'

```

```
pass

# Note: The actual implementation of 'identify_domain_walls' and 'apply_pulse_at_location'
# will depend on the capabilities of your AFM system and the specifics of your experiment.
```

#### Key Considerations:

1. **Image Analysis for Domain Wall Identification:** This is a complex task and would likely require custom image processing algorithms. The function `identify\_domain\_walls` is a placeholder and should be implemented based on your specific requirements.
2. **BEPS and DC Pulse Application:** These operations would be based on your AFM's capabilities and the AEcroscopy API. You'll need to replace `do\_beps\_specific` and `apply\_pulse\_at\_location` with the actual functions provided by AEcroscopy.
3. **Re-Imaging:** After applying a DC pulse, re-imaging is done to observe changes. This might require a new raster scan or another appropriate imaging method.

This script is a high-level outline and will require significant customization and implementation of specific functions, especially for image analysis and domain wall location identification. Ensure that AEcroscopy and all dependencies are correctly installed



#### Anonymous

Please define a function to identify domain walls using edge detector



#### ChatGPT

To define a function that identifies domain walls using an edge detection algorithm in Python, you can utilize image processing libraries like OpenCV or scikit-image. Edge detection is a common technique in image processing used to find boundaries or edges within images. It can be particularly useful for identifying domain walls in microscopy images, depending on the contrast and quality of the image.

Below is an example function that uses the Canny edge detector from the `cv2` library (OpenCV) to identify edges in an image, which could correspond to domain walls:

```
python
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Copy code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def identify_domain_walls(image, low_threshold=50, high_threshold=150, plot=False):
    """
    Identify domain walls using the Canny edge detector.

    :param image: Input image (2D array).
    :param low_threshold: Lower threshold for the hysteresis procedure in Canny detector.
    :param high_threshold: Higher threshold for the hysteresis procedure in Canny detector.
    :param plot: If True, the function will plot the original and processed images.
    :return: Edge-detected image.
    """

    # Convert the image to grayscale if it's not already
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to smooth the image
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply the Canny edge detector
    edges = cv2.Canny(blurred_image, low_threshold, high_threshold)

    if plot:
        plt.figure(figsize=(12, 5))

        plt.subplot(1, 2, 1)
        plt.imshow(image, cmap='gray')
        plt.title('Original Image')
        plt.axis('off')

        plt.subplot(1, 2, 2)
        plt.imshow(edges, cmap='gray')
        plt.title('Edges Detected')
        plt.axis('off')

    plt.show()
```

```
plt.show()

return edges

# Example usage:
# Load an image (replace 'path_to_image' with the path to your actual image file)
image = cv2.imread('path_to_image', cv2.IMREAD_GRAYSCALE)

# Identify domain walls
edges = identify_domain_walls(image, plot=True)
```

#### Key Points:

- This function uses the Canny edge detector, which is effective for many edge detection applications. It requires two threshold values, which you may need to adjust based on your specific images.
- The function includes an option to plot the original and edge-detected images for visualization purposes.
- The function assumes the input image is a 2D grayscale image. If it's a color image, the function converts it to grayscale.

#### Dependencies:

- You need to have OpenCV (`cv2`) installed in your Python environment. You can install it via pip (`pip install opencv-python-headless` for a minimal installation without GUI functionalities).
- If your domain wall images have specific characteristics, you may need to fine-tune the parameters or use more sophisticated image processing techniques.

Remember that edge detection for identifying domain walls can be sensitive to noise and image quality. Additional image processing steps (like filtering or morphological operations) might be necessary for optimal results, depending on your data.



#### Anonymous

Please define a function to apply pulse at locations using the tip control function and apply pulse function, here users can move tip to domain wall first and apply a pulse then



#### ChatGPT

To define a function that moves the AFM tip to specified locations (presumably domain walls) and then applies a pulse at each location. we can create a function that utilizes the `tip control` function for moving



ChatGPT

To define a function that moves the AFM tip to specified locations (presumably domain walls) and then applies a pulse at each location, we can create a function that utilizes the `tip\_control` function for moving the tip and another function, possibly named `apply\_pulse`, for applying the pulse. This function would be a part of the workflow using the AEcroscopy API, as outlined in your guide.

Here's an example of how such a function could be structured:

python

Copy code

```
def apply_pulse_at_domain_walls(acquisition, locations, pulse_params, tip_move_params):
    """
    Moves the AFM tip to specified locations and applies a pulse at each location.

    :param acquisition: The Acquisition object from the AEcroscopy API.
    :param locations: List of coordinates where the pulse will be applied.
    :param pulse_params: Dictionary containing parameters for the pulse.
    :param tip_move_params: Dictionary containing parameters for moving the tip.
    """

    for location in locations:
        # Move the tip to the current location (domain wall)
        tip_params = tip_move_params.copy()
        tip_params.update({"next_x_pos_00": location[0], "next_y_pos_01": location[1]})
        acquisition.tip_control(tip_params_dict=tip_params, do_move_tip=True, feedbackon=False)

        # Apply the pulse at the current location
        acquisition.apply_pulse(pulse_params_dict=pulse_params, do_create_pulse=True, do_update=True)

# Example usage
# Assuming 'newexp' is an instance of the Acquisition class and already initialized
locations = [(x1, y1), (x2, y2), ...] # Replace with actual domain wall coordinates
pulse_params = {
    "pulse_amplitude_V": 5, # Example amplitude
    "pulse_duration_s": 0.02 # Example duration
    # ... other necessary pulse parameters ...
}
tip_move_params = {
    # Parameters for moving the tip
    "set_point_V_00": 1 # Example setpoint voltage
}
```

```
tip_move_params = {  
    # Parameters for moving the tip  
    "set_point_V_00": 1 # Example setpoint voltage  
    # ... other necessary tip control parameters ...  
}  
  
apply_pulse_at_domain_walls(newexp, locations, pulse_params, tip_move_params)
```

#### Key Points:

- The function `apply\_pulse\_at\_domain\_walls` iterates over a list of locations, moves the tip to each location, and then applies a pulse.
- `locations` should be a list of tuples or lists, where each tuple/list represents the x and y coordinates of a domain wall location.
- `pulse\_params` and `tip\_move\_params` are dictionaries containing the necessary parameters for pulse application and tip movement, respectively. These should be defined based on the specific requirements of your experiment and the capabilities of your AFM system.
- The function assumes that `acquisition` (an instance of the `Acquisition` class from the AEcroscopy API) is already initialized and configured for use.

This is a conceptual example and might require adjustments to fit the specific functions and parameters of the AEcroscopy API you are using. Additionally, error handling and specific AFM behavior should be considered for robust and reliable operation.



#### Anonymous

What are the methods to enhance weak BE signals?



#### ChatGPT

Enhancing weak Band Excitation (BE) signals in the context of Piezoresponse Force Microscopy (PFM) or other scanning probe microscopy techniques often involves both experimental setup optimization and post-acquisition data processing. Here are some common methods:

## Experimental Setup Optimization

### 1. Improving Signal-to-Noise Ratio (SNR):

- **Use of Low-Noise Electronics:** Employing low-noise amplifiers and minimizing electronic interference in the measurement setup can significantly improve SNR.

#### **1. Improving Signal-to-Noise Ratio (SNR):**

- **Use of Low-Noise Electronics:** Employing low-noise amplifiers and minimizing electronic interference in the measurement setup can significantly improve SNR.
- **Optimized Detector Sensitivity:** Using detectors or amplifiers with adjustable gain settings can help amplify weak signals.

#### **2. Optimization of Driving Frequency:**

- **Resonance Enhancement:** Operating at or near the resonance frequency of the cantilever or the system can amplify the response signal.
- **Frequency Sweeping:** Sometimes, sweeping through a range of frequencies can help identify the optimal frequency that provides the best response.

#### **3. Probe Selection:**

- **Quality and Type of Probe:** The choice of probe can significantly affect signal quality. Probes with higher mechanical Q-factors may provide better resonance enhancement.

#### **4. Environment Control:**

- **Vibration Isolation:** Minimizing external vibrations with an isolated table or environment can reduce noise.
- **Temperature and Humidity Control:** Stable environmental conditions can reduce signal drift and noise.

#### **5. Optimized Scanning Parameters:**

- **Tip Velocity and Force:** Adjusting the tip's scanning speed and the force applied to the sample can help enhance the quality of the acquired signal.
- **Feedback Loop Tuning:** Careful tuning of the feedback loop parameters can optimize the tip-sample interaction for better signal acquisition.

## **Data Processing Techniques**

#### **1. Signal Filtering:**

- **Bandpass Filtering:** Using bandpass filters to isolate the frequency range of interest while eliminating noise from other frequencies.
- **Lock-in Amplification:** Employing lock-in techniques to extract the signal at a specific reference frequency, enhancing the signal quality.

#### **2. Signal Averaging:**

- **Time Averaging:** Repeatedly measuring the signal over time and averaging can reduce random noise.
- **Ensemble Averaging:** If the experiment is repeatable, averaging the results of multiple similar

noise.

- **Ensemble Averaging:** If the experiment is repeatable, averaging the results of multiple similar experiments can enhance signal strength.
3. **Advanced Data Analysis Methods:**
    - **Fourier or Wavelet Transform:** Transforming the signal into the frequency or time-frequency domain can help isolate important features.
    - **Principal Component Analysis (PCA):** PCA and other multivariate analysis techniques can help in isolating signal components from noise.
  4. **Software Phase-Locked Loop (PLL):**
    - **Digital PLL Techniques:** Software-based PLL can track phase and amplitude changes more accurately in weak signals.

## Other Considerations

1. **Calibration:** Regular calibration of the equipment to ensure that measurements are accurate and reliable.
2. **Sample Preparation:** The quality of the sample surface and its preparation can significantly influence signal strength.
3. **Collaboration with Experts:** Consulting with microscopy and materials science experts can provide insights into optimizing experimental setups for specific materials or phenomena.

In summary, enhancing weak BE signals usually involves a combination of optimizing the experimental setup to maximize signal acquisition and employing sophisticated data processing techniques to extract meaningful information from the acquired data. The specific methods to be used depend greatly on the nature of the sample, the details of the measurement system, and the type of information being sought.



**Anonymous**

Suggest methods for investigating domain walls in ferroelectric samples



**ChatGPT**

Investigating domain walls in ferroelectric materials is a crucial aspect of understanding their electrical and mechanical properties. Several experimental techniques and methodologies can be employed to study these domain walls:

### 1. Piezoresponse Force Microscopy (PFM)

STUDY THESE DOMAIN WALLS.

## 1. Piezoresponse Force Microscopy (PFM)

- **PFM Imaging:** A powerful tool for visualizing and manipulating ferroelectric domains at the nanoscale. PFM can directly image domain structures and measure local piezoelectric responses.
- **Spectroscopic PFM:** Allows for the study of local electromechanical coupling at domain walls by analyzing the amplitude and phase of the piezoresponse as a function of the applied voltage.

## 2. Scanning Probe Microscopy Techniques

- **Conductive Atomic Force Microscopy (c-AFM):** Useful for studying the conductive properties of domain walls.
- **Kelvin Probe Force Microscopy (KPFM):** Can provide information about surface potential variations at domain walls.

## 3. Optical Techniques

- **Raman Spectroscopy:** Can be used to investigate lattice distortions and stress fields around domain walls.
- **Second Harmonic Generation (SHG):** A nonlinear optical technique sensitive to non-centrosymmetry, thus suitable for studying ferroelectric domain walls.

## 4. Transmission Electron Microscopy (TEM)

- **High-Resolution TEM:** Provides atomic-scale information about the structure of domain walls.
- **Electron Holography:** Allows for the visualization of electric field distributions in and around domain walls.

## 5. X-Ray Diffraction Techniques

- **Synchrotron X-Ray Diffraction:** Can be used to study the strain fields and structural distortions at domain walls.
- **X-Ray Photoelectron Spectroscopy (XPS):** Useful for investigating the chemical states at the domain walls.

## 6. Electrical Characterization

- **Hysteresis Loop Measurements:** Analyzing the polarization-electric field (P-E) hysteresis loops can

## **6. Electrical Characterization**

- **Hysteresis Loop Measurements:** Analyzing the polarization-electric field (P-E) hysteresis loops can provide insights into the behavior of domain walls under an electric field.
- **Impedance Spectroscopy:** Can be used to study the frequency-dependent electrical response of domain walls.

## **7. Band Excitation Methods**

- **Band Excitation Piezoresponse Force Spectroscopy:** A method to study the dynamic response of domain walls over a range of frequencies, providing insights into their kinetic properties.

## **8. Theoretical and Computational Approaches**

- **Phase-Field Modeling:** Useful for simulating domain evolution and understanding the dynamics of domain walls.
- **Density Functional Theory (DFT) Calculations:** Can provide insights into the electronic structure of domain walls.

## **9. Mechanical Characterization**

- **Nanoindentation:** Can provide information on the mechanical properties of domain walls and how they differ from the bulk material.

## **10. Multifrequency AFM Techniques**

- Combining different AFM modalities (like PFM, c-AFM, KPFM) in a single scan to correlate electrical, mechanical, and topographical information at domain walls.

## **Considerations for Experimental Design**

- **Sample Preparation:** High-quality sample preparation is crucial, especially for techniques like TEM or AFM.
- **Environmental Conditions:** Some measurements might require controlled environments (e.g., vacuum, temperature control).
- **Calibration and Standardization:** Ensuring that the measurement techniques are well-calibrated and standardized for reproducibility.

Each of these techniques offers unique insights into the properties of domain walls in ferroelectric

standardized for reproducibility.

Each of these techniques offers unique insights into the properties of domain walls in ferroelectric materials, and often, a combination of several methods is used to gain a comprehensive understanding. The choice of method depends on the specific properties of interest, the scale of investigation, and the available instrumentation.



#### Anonymous

Describe and program potential experiments in PFM to study ferroelectric domain wall dynamics using AEcrosoPy



#### ChatGPT

To study ferroelectric domain wall dynamics using Piezoresponse Force Microscopy (PFM) with AEcroscopy, a variety of experiments can be designed. Here, I'll outline a few potential experiments and provide a conceptual Python programming approach for each using the AEcroscopy API. Remember that the specific code might need adjustments based on the exact capabilities and functions available in your version of AEcroscopy.

## Experiment 1: Domain Wall Imaging and Manipulation

**Objective:** Visualize domain walls and manipulate them using an electric field.

**Python Programming Outline:**

1. **PFM Imaging:** Perform initial PFM imaging to visualize the domain structure.
2. **Domain Manipulation:** Apply an electric field to manipulate domain walls.
3. **Post-manipulation Imaging:** Re-image the area to observe changes in domain structure.

```
python
from Aecroscopy import Acquisition

# Initialize AEcroscopy Acquisition
newexp = Acquisition(exe_path='path_to_BEPyAE.exe')

# 1. PFM Imaging
# Parameters for initial PFM imaging
pfm_image_params = [
    # Parameters for PFM imaging
```

```

# Parameters for initial PFM imaging
pfm_image_params = {
    # Parameters for PFM imaging
}
initial_pfm_image = newexp.pfm_scan(pfm_image_params)

# 2. Domain Manipulation
# Apply an electric field
electric_field_params = {
    # Parameters to apply the electric field
}
newexp.apply_electric_field(electric_field_params)

# 3. Post-manipulation Imaging
# Re-image to observe changes
post_pfm_image = newexp.pfm_scan(pfm_image_params)

```

## Experiment 2: Frequency-Dependent Domain Wall Response

**Objective:** Investigate the frequency-dependent response of domain walls.

**Python Programming Outline:**

1. **Spectroscopic PFM:** Conduct spectroscopic PFM measurements at different frequencies across the domain wall.

```

python                                         Copy code

# Spectroscopic PFM at various frequencies
frequency_ranges = [1e5, 2e5, 3e5, 4e5] # Example frequencies in Hz

for freq in frequency_ranges:
    spectroscopic_params = {
        'frequency': freq,
        # other parameters
    }
    response = newexp.spectroscopic_pfm(spectroscopic_params)
    # Process and store the response data

```

```
response = newexp.spectroscopic_pfm(spectroscopic_params)
# Process and store the response data
```

## Experiment 3: Time-Resolved Domain Wall Dynamics

**Objective:** Observe domain wall motion over time under an external stimulus.

**Python Programming Outline:**

1. **Time-Lapse PFM Imaging:** Perform PFM imaging over time while applying an external stimulus.

```
python
# Time-lapse PFM Imaging
time_points = range(0, 3600, 300) # Every 5 minutes for 1 hour

for time_point in time_points:
    # Apply stimulus if needed (e.g., electric field, temperature change)
    newexp.apply_stimulus(stimulus_params)

    # Perform PFM imaging
    time_lapse_image = newexp.pfm_scan(pfm_image_params)
    # Store or process the image
```

### General Notes:

- **Path to BEPyAE:** Replace `path\_to\_BEPyAE.exe` with the actual path to your BEPyAE executable.
- **Function Parameters:** The functions `pfm\_scan`, `apply\_electric\_field`, `spectroscopic\_pfm`, and `apply\_stimulus` are placeholders. Replace them with the actual AEcroscopy API functions and appropriate parameters for your experiments.
- **Data Processing:** After each measurement, you may need to process the data to extract meaningful information about domain wall dynamics.
- **Error Handling:** Consider adding error handling to check the success of each operation and handle any exceptions that may occur during the experiment.

These outlines provide a starting point for programming PFM experiments to study ferroelectric domain wall dynamics using AEcroscopy. Tailor these scripts to fit the specific capabilities of your AEcroscopy setup and the details of the experiments you wish to conduct.

These outlines provide a starting point for programming AEC microscopy experiments to study ferromagnetic domain wall dynamics using AEcroscopy. Tailor these scripts to fit the specific capabilities of your AEcroscopy setup and the details of the experiments you wish to conduct.

 **Anonymous**



Summarize the AEcroscopy experiment detailed in the attached logger file using natural language for clarity.

 **ChatGPT**

The AEcroscopy experiment detailed in the logger file involved three main steps:

1. **Tip Control:** The experiment began with controlling the AFM tip. The setpoint voltage was set to 1V, and the tip was moved to the position [0.5, 0.5]. Both moving the tip and setting the setpoint were executed successfully, with feedback turned off.
2. **IO Cluster Definition:** Next, the Input/Output (IO) cluster was defined. The parameters set included the analog output amplifier, and three different channel types. These IO settings were successfully applied with feedback enabled.
3. **Band Excitation (BE) Parameters Definition:** Finally, the parameters for band excitation (BE) were defined. This included setting the center frequency at 335 Hz, a bandwidth of 100 Hz, an amplitude of 1V, along with other parameters like phase variation, number of repeats, required pulse duration, and auto smooth ring. The creation of the BE waveform was executed with feedback turned off.

Each of these steps was crucial for the proper setup and execution of the experiment, ensuring precise control and measurement capabilities for the study.

 **Anonymous**

Please draft Python code that would reproduce the experiment detailed in the logger file.

 **ChatGPT**

Based on the details from the logger file, the Python code to reproduce the AEcroscopy experiment would include the following steps:

1. Initialization and Import of Necessary Packages
2. Tip Control Setup
3. Input/Output (IO) Cluster Definition

2. Tip Control Setup
3. Input/Output (IO) Cluster Definition
4. Band Excitation (BE) Parameters Definition

Here is a draft of the Python code:

```
python
import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = "path_to_BEPyAE.exe") # Replace with your actual path

# Tip Control
newexp.tip_control(tip_parms_dict={"set_point_V_00": 1, "next_x_pos_00": 0.5, "next_y_pos_00": 0.5, "do_move_tip=True, do_set_setpoint=True, feedbackon=False})

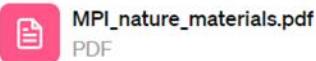
# IO Cluster Definition
newexp.define_io_cluster(IO_cluster_parms_dict={"analog_output_amplifier_06": 1,
                                                "channel_01_type_07": 1,
                                                "channel_02_type_08": 2,
                                                "channel_03_type_09": 3})

# Band Excitation Parameters Definition
newexp.define_be_parms(be_parms_dict={"center_frequency_Hz_00": 335, "band_width_Hz_01": 1, "amplitude_V_02": 1, "phase_variation_03": 1, "repeats_04": 4, "req_pulse_duration_s_05": 4, "auto_smooth_ring_06": 1}).
```

```
        "amplitude_V_02": 1, "phase_variation_03": 1,  
        "repeats_04": 4, "req_pulse_duration_s_05": 4,  
        "auto_smooth_ring_06": 1},  
    do_create_be_waveform=True, feedbackon=False)
```

Replace `"`path_to_BEPyAE.exe`"` with the actual path to your `BEPyAE.exe` file. This code assumes that the functions and parameters in the AEcroscopy API match those detailed in the logger file. Adjustments may be needed based on the specific version and configuration of the AEcroscopy package you are using.

 Anonymous

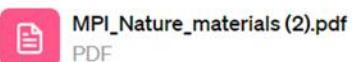


Please extract BEPFM experiment parameters from the attached paper

 ChatGPT

The file "MPI\_nature\_materials.pdf" does not contain information about BEPFM (Band Excitation Piezoresponse Force Microscopy) experiments or related parameters. Instead, it focuses on "MatSciBERT," a language model designed for text mining and information extraction in the materials science domain. The paper discusses the development, training, and application of MatSciBERT, including its use in tasks like named entity recognition, relation classification, and abstract classification. It emphasizes the model's training on a large corpus of peer-reviewed materials science publications and its improved performance over other models in specific materials science tasks.

 Anonymous



Please extract BEPFM experiment parameters from the attached paper

 ChatGPT

The paper "Chemical nature of ferroelastic twin domains in CH<sub>3</sub>NH<sub>3</sub>PbI<sub>3</sub> perovskite" details a series of Band Excitation Piezoresponse Force Microscopy (BE-PFM) experiments. The key parameters for BE-PFM experiments mentioned in the paper include:

1. **Drive Frequency for LDV-PFM Measurements:** 335 kHz.
2. **BE-PFM Parameters:**

1. **Drive Frequency for LDV-PFM Measurements:** 335 kHz.

2. **BE-PFM Parameters:**

- Center frequency around 340 kHz.
- Bandwidth of 100 kHz.
- AC bias of 1.0 V applied during measurements.

These parameters were crucial for the experiments conducted to study the twin domains in CH<sub>3</sub>NH<sub>3</sub>PbI<sub>3</sub> perovskite. The paper emphasizes the importance of these settings in achieving accurate and reliable results in their investigation of the material's mechanical, electromechanical, and chemical properties.



**Anonymous**

Please draft Python code that would reproduce the BEPFM experiment.



**ChatGPT**

Based on the BEPFM (Band Excitation Piezoresponse Force Microscopy) experiment parameters extracted from the paper, here is a Python code draft to reproduce the experiment:

```
python Copy code

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path = "path_to_BEPyAE.exe") # Replace with the actual path to your executable

# BE-PFM Parameters
# Center frequency around 340 kHz, bandwidth of 100 kHz, and AC bias of 1.0 V
```

```

# BE-PFM Parameters
# Center frequency around 340 kHz, bandwidth of 100 kHz, and AC bias of 1.0 V
be_pfm_params = {
    "center_frequency_Hz_00": 340000, # Center frequency at 340 kHz
    "band_width_Hz_01": 100000,      # Bandwidth of 100 kHz
    "amplitude_V_02": 1.0          # AC bias of 1.0 V
    # Add other necessary parameters as needed
}

# Perform BE-PFM experiment
# Assuming there is a function in AEcroscopy to perform BE-PFM
newexp.perform_be_pfm(be_pfm_params)

```

In this code, replace `""` with the actual path to the BEPyAE executable file. The `perform\_be\_pfm` function is a placeholder, and you should replace it with the actual function name provided by AEcroscopy for conducting BE-PFM experiments. Make sure to include any additional parameters required by your specific experimental setup or the AEcroscopy API.

#### Anonymous

Please draft Python code that would reproduce the BEPFM experiment using raster scan

#### ChatGPT

To reproduce the BEPFM experiment using a raster scan approach, based on the provided parameters (center frequency around 340 kHz, bandwidth of 100 kHz, and AC bias of 1.0 V), here's a Python script draft using AEcroscopy:

```

python

import os
import win32com.client
import numpy as np
import time
import h5py
import sidpy
import pyNSID
import matplotlib.pyplot as plt
from tqdm import tqdm

```

 Copy code

```

import matplotlib.pyplot as plt
from tqdm import tqdm

# Import Acquisition from Aecroscopy
from Aecroscopy import Acquisition

# Initialize the AEcroscopy Acquisition
newexp = Acquisition(exe_path="path_to_BEPyAE.exe") # Replace with the actual path to your executable

# BEPFM Raster Scan Parameters
be_raster_params = {
    "center_frequency_Hz_00": 340000, # Center frequency at 340 kHz
    "band_width_Hz_01": 100000, # Bandwidth of 100 kHz
    "amplitude_V_02": 1.0, # AC bias of 1.0 V
    # Additional parameters for raster scanning like pixel count, scan region, etc.
}

# Execute BEPFM Raster Scan
# This assumes a function 'raster_scan' in AEcroscopy for BEPFM, which might differ in actual implementation
dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict=be_raster_params, file_name="raster_scan.h5")

```

This code is a template and assumes certain AEcroscopy functions. Replace `'"path\_to\_BEPyAE.exe"'` with the correct path to the BEPyAE executable. The `raster\_scan` function and `be\_raster\_params` should be adjusted according to the actual AEcroscopy API and your experimental setup. Make sure to include all necessary parameters for a complete raster scan setup.



**Anonymous**

Please set the BE parameters according to the extracted information



**ChatGPT**

To set the Band Excitation (BE) parameters for the BEPFM experiment based on the extracted information from the paper, you would configure the parameters as follows:

1. **Center Frequency:** Set the center frequency to approximately 340 kHz.
2. **Bandwidth:** Set the bandwidth to 100 kHz.
3. **Amplitude:** Set the AC bias amplitude to 1.0 V.

Here's how you might set these parameters in Python using AEcroscopy:

3. **Amplitude:** Set the AC bias amplitude to 1.0 V.

Here's how you might set these parameters in Python using AEcroscopy:

```
python

# BEPFM Parameters
be_params = {
    "center_frequency_Hz_00": 340000, # Center frequency at 340 kHz
    "band_width_Hz_01": 100000,      # Bandwidth of 100 kHz
    "amplitude_V_02": 1.0           # AC bias of 1.0 V
    # Include other necessary parameters as needed
}

# Assuming a function in AEcroscopy to set BE parameters
newexp.set_be_params(be_params)
```

This script assumes a function `set\_be\_params` in the AEcroscopy API for setting BE parameters, which might differ in the actual API. Make sure to replace it with the correct function name and adjust the parameters according to your specific AEcroscopy setup.

 **Anonymous**

Please set the band excitation parameters according to the extracted information

 **ChatGPT**

To set the Band Excitation (BE) parameters for the BEPFM experiment according to the extracted information from the paper, you should configure the following:

1. **Center Frequency:** Approximately 340 kHz (which equates to 340000 Hz in the parameter settings).
2. **Bandwidth:** Approximately 100 kHz (100000 Hz).
3. **Amplitude:** An AC bias of 1.0 V.

In Python, using the AEcroscopy API, this configuration would look like this:

```
python

# Setting the Band Excitation parameters
be_parms_dict = {
```

```

# Setting the Band Excitation parameters
be_parms_dict = {
    "center_frequency_Hz_00": 340000, # Center frequency at 340 kHz
    "band_width_Hz_01": 100000,      # Bandwidth of 100 kHz
    "amplitude_V_02": 1.0          # AC bias amplitude of 1.0 V
    # Include any other necessary parameters
}

# Assuming 'newexp' is your AEcroscopy experiment instance
newexp.define_be_parms(be_parms_dict, do_create_be_waveform=True, feedbackon=False)

```

This code assumes that `newexp` is an instance of your experiment and `define\_be\_parms` is the function used to define BE parameters in AEcroscopy. Make sure to adjust the function name and parameters according to the actual API specifications of AEcroscopy that you are using.

 **Anonymous**

Please set band excitation amplitude to 2 and center frequency to 380 kHz

 **ChatGPT**

To update the Band Excitation (BE) parameters in your AEcroscopy experiment with an amplitude of 2V and a center frequency of 380 kHz, you can modify the parameters in your Python script as follows:

```

python                                         Copy code

# Update the Band Excitation parameters
be_parms_dict = {
    "center_frequency_Hz_00": 380000, # Center frequency set to 380 kHz
    "amplitude_V_02": 2.0          # BE amplitude set to 2 V
    # Retain the other necessary parameters as they are
}

# Assuming 'newexp' is your AEcroscopy experiment instance
newexp.define_be_parms(be_parms_dict, do_create_be_waveform=True, feedbackon=False)

```

This script assumes that `newexp` is an instance of your AEcroscopy experiment and that `define\_be\_parms` is the correct function to set BE parameters. Make sure to adjust the script according to your specific AEcroscopy setup and API version.

`'define_be_parms'` is the correct function to set BE parameters. Make sure to adjust the script according to your specific AEcroscopy setup and API version.

 **Anonymous**

How to perform a spiral scan

 **ChatGPT**

Performing a spiral scan in the context of microscopy, particularly in scanning probe microscopy like AFM (Atomic Force Microscopy), involves moving the probe in a spiral pattern over the sample surface. This method is typically used for advanced imaging or manipulation tasks. The key steps include:

1. **Setting up the microscope:** Ensure your microscope is properly calibrated and the sample is correctly positioned.
2. **Defining spiral scan parameters:** This includes the spiral's center, inner and outer radii, number of spiral cycles, and scan speed.
3. **Executing the scan:** The probe moves in a spiral trajectory, starting from the center and moving outwards or vice versa, depending on the experiment's goals.
4. **Data collection:** As the probe moves, it collects data based on the interaction with the sample surface.
5. **Data analysis:** Post-scan, analyze the collected data to extract meaningful information.

The exact implementation of a spiral scan would depend on the specific capabilities of your microscopy system and the software used to control it. It's essential to consult your microscope's user manual or software documentation for detailed instructions tailored to your specific system.

 **Anonymous**

How to perform a spiral scan using AEcroscopy

 **ChatGPT**

To perform a spiral scan using AEcroscopy, typically, you would follow these steps:

1. **Initialize the AEcroscopy System:** Start by initializing your AEcroscopy system and ensuring that all necessary components are properly configured.
2. **Define Spiral Scan Parameters:** Specify the parameters for the spiral scan, which might include the center coordinates of the spiral, inner and outer radii, the number of spiral cycles, duration, direction, and other relevant settings.
3. **Execute the Spiral Scan:** Use the AEcroscopy API to execute the spiral scan. This would typically involve a command that takes the defined parameters and controls the movement of the probe in a

3. **Execute the Spiral Scan:** Use the AEcroscopy API to execute the spiral scan. This would typically involve a command that takes the defined parameters and controls the movement of the probe in a spiral pattern over the sample.
4. **Collect and Analyze Data:** As the scan progresses, collect the data generated during the scan. After the scan, analyze this data to derive meaningful insights.

The exact method to perform a spiral scan can vary based on the specific AEcroscopy setup and the version of the software you are using. It's important to consult the AEcroscopy documentation or user manual for detailed instructions and the correct syntax for executing a spiral scan.

**Supplementary Note II: Record of using ChatGPT to interpret BEPFM data**

 You

 BEPFM\_2um\_1V\_0009.h5  
File

 BEPFM\_2um\_3V\_0009.h5  
File

 BEPFM\_2um\_5V\_0009.h5  
File

Attached are three band excitation PFM image data, please analyze these data and summarize their difference.

 ChatGPT

To analyze the three Band Excitation Piezoresponse Force Microscopy (BE-PFM) image data files you've provided, I'll first inspect their content to understand the structure and format of the data. This will help in determining the differences among the datasets, which might include variations in piezoresponse, surface topography, or other characteristics that BE-PFM can measure. Let's begin by loading and examining the data from each file.

 Error analyzing ↴

It seems there was an error in my initial approach. The data within the provided BE-PFM files are organized in a hierarchical structure, and the top-level items are groups rather than datasets, which is why attempting to access their shapes directly caused an error.

I'll adjust the analysis method to better explore the structure of these files, focusing on understanding the hierarchy and identifying the datasets within each file. This should provide a clearer picture of the contents of each file, allowing for a comparison of their differences. Let's proceed with this revised approach. [→]

The structure of the first BE-PFM file ('**BEPFM\_2um\_1V\_0009.h5**') reveals a hierarchical organization with various groups and datasets. Key findings include:

- Multiple channels (Channel\_000, Channel\_001, Channel\_002) within the 'Measurement\_000' group.
- Each channel contains datasets such as 'Raw\_Data', 'Position\_Indices', 'Position\_Values', and others.
- 'Channel\_000' has more detailed datasets like 'Bin\_FFT', 'Bin\_Frequencies', 'Excitation\_Waveform', and a sub-group 'Raw\_Data-SHO\_Fit\_000'.

This structure suggests that the file contains detailed piezoresponse force microscopy

 Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

'Position\_Values', and others.

- 'Channel\_000' has more detailed datasets like 'Bin\_FFT', 'Bin\_Frequencies', 'Excitation\_Waveform', and a sub-group 'Raw\_Data-SHO\_Fit\_000'.

This structure suggests that the file contains detailed piezoresponse force microscopy data, including raw data, position information, and possibly frequency and excitation waveform data. To compare the three files effectively, I'll now check if the other two files have a similar structure and then focus on key datasets that could highlight differences, such as 'Raw\_Data', 'Position\_Values', or frequency-related datasets. Let's proceed with this analysis. [→]

The structures of the second ('**BEPFM\_2um\_3V\_0009.h5**') and third ('**BEPFM\_2um\_5V\_0009.h5**') BE-PFM files are consistent with the first one. Each file contains a similar hierarchy of groups and datasets, such as 'Raw\_Data', 'Position\_Indices', 'Position\_Values', and 'Spectroscopic\_Values' across different channels.

Given this consistency, the differences among these files likely arise from the variations in the data values themselves, rather than their overall structure. These differences could be due to changes in experimental conditions, such as different applied voltages, as suggested by the file names (1V, 3V, 5V).

To summarize the differences in the data, I'll compare key datasets such as 'Raw\_Data' and 'Position\_Values' across the three files. This comparison will provide insights into how the piezoresponse and other properties vary with different experimental conditions. Let's proceed with this analysis. [→]

The comparison of the 'Raw\_Data' and 'Position\_Values' datasets from each BE-PFM file reveals the following:

#### 1. Raw Data Comparison:

- The 'Raw\_Data' from each file shows complex number arrays, which are likely representing the piezoresponse signals at different positions and under different conditions.
- The samples from each file show varying complex number values, indicating differences in the piezoresponse. These differences are probably due to the varying voltage conditions (1V, 3V, 5V) used during the measurements.

#### 2. Position Values Comparison:

- The 'Position\_Values' datasets from all three files are consistent, showing similar float values in a 2D array format. This suggests that the position data (likely representing the scanning positions on the sample) is the same across all three measurements.



---

0 Message ChatGPT...

---

ChatGPT can make mistakes. Consider checking important information.

To summarize, the main differences among these datasets appear to be in the 'Raw\_Data', reflecting the piezoresponse under different voltage conditions, while the 'Position\_Values' remain consistent across the files. This is consistent with the expectation that varying the voltage in BE-PFM experiments would alter the piezoresponse while keeping the spatial scanning parameters constant.

For a more detailed analysis, including visualizations and statistical comparisons, additional processing and analysis would be necessary, especially focusing on the 'Raw\_Data' to interpret the piezoresponse variations quantitatively. [→]

### Y You

Please the amplitude and phase in these three data, then compare the difference of the amplitude and phase among three data. Note that amplitude and phase need to be reshaped to 128\*128 before plotting.

### ChatGPT

To analyze and compare the amplitude and phase of the piezoresponse from the three BE-PFM data files, I will follow these steps:

1. Extract the amplitude and phase information from the 'Raw\_Data' dataset in each file. Typically, in BE-PFM data, the amplitude and phase are encoded in the complex numbers, with the amplitude being the magnitude and the phase being the angle of these complex numbers.
2. Reshape the amplitude and phase data to a 128×128 grid, as the data likely represents a 2D scan of a surface.
3. Plot the amplitude and phase for each of the three datasets, allowing for a visual comparison.

Let's begin with extracting, reshaping, and plotting the amplitude and phase data from each of the files.

### >Error analyzing

It appears that the size of the 'Raw\_Data' datasets in the BE-PFM files does not directly correspond to a 128×128 grid. This mismatch in dimensions suggests that either the data represents more than just a single scan line or the resolution of the scan is different from what was expected.

To resolve this, I will first check the actual dimensions of the 'Raw\_Data' dataset and then determine the correct way to reshape it for analysis. Let's proceed with examining the dimensions of the 'Raw\_Data' datasets in each file. [→]

### Message ChatGPT...

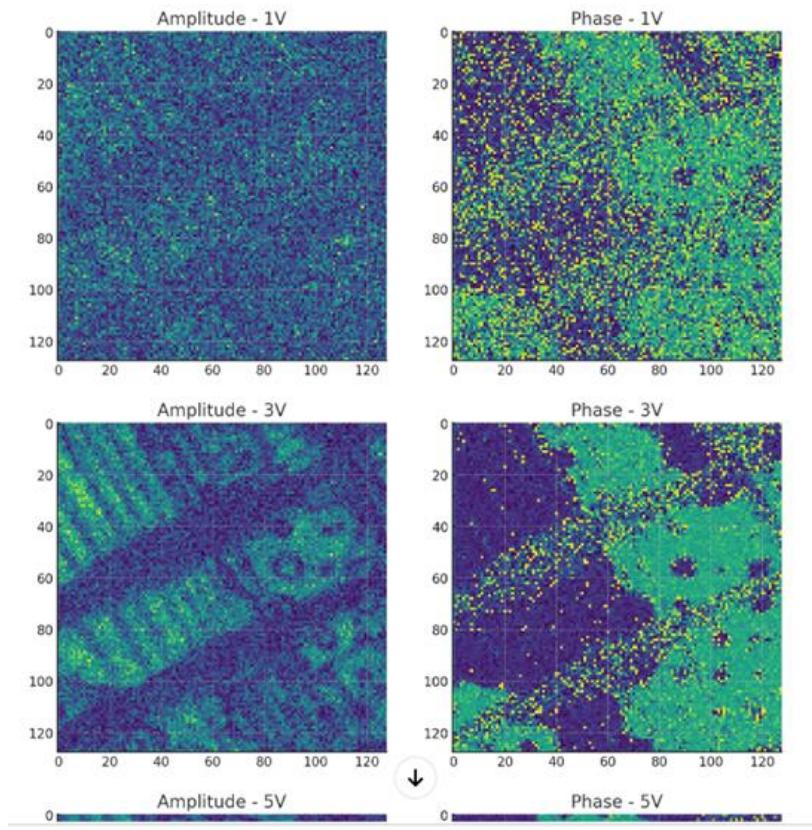
ChatGPT can make mistakes. Consider checking important information.

The 'Raw\_Data' datasets in each of the BE-PFM files have a shape of  $16384 \times 62$ . This shape does not directly correspond to a  $128 \times 128$  grid, which indicates a more complex data structure.

Given the dimensions, it seems that each dataset contains multiple scan lines or frequency sweeps. To reshape this into a  $128 \times 128$  grid, we need to understand how the data is organized. One common approach in PFM data is that one of the dimensions represents different points on a line scan, and the other represents different lines or frequencies.

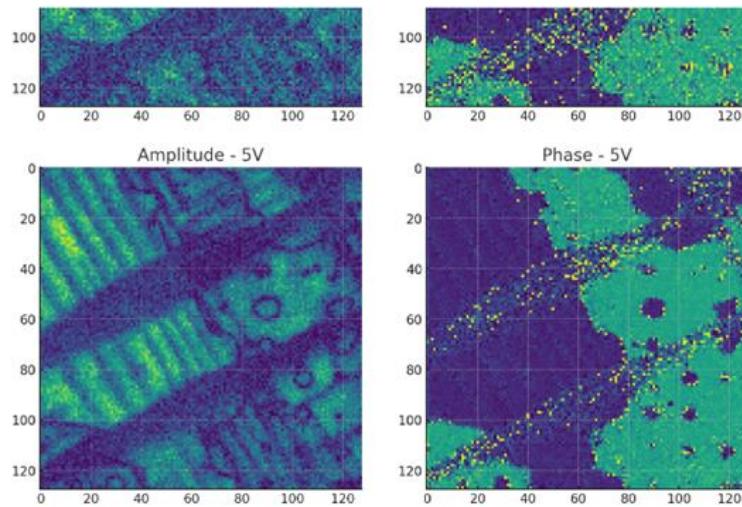
If we assume that the larger dimension (16384) represents a combination of line scans, and each line scan has a width of 128 points, we can try reshaping the data accordingly. However, this is an assumption and may not perfectly represent the actual layout of the scans. I'll proceed with this approach and reshape the data into a  $128 \times 128$  grid, focusing on a single frequency (or line) for simplicity.

Let's attempt this reshaping and then plot the amplitude and phase for a clearer comparison. [→]



Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.



Here are the amplitude and phase images for each of the BE-PFM datasets, reshaped into a 128×128 grid. Each pair of images represents one of the files, corresponding to different voltage conditions (1V, 3V, 5V):

**1. First File (1V):**

- The left image shows the amplitude.
- The right image shows the phase.

**2. Second File (3V):**

- Similarly, the left image is the amplitude, and the right image is the phase.

**3. Third File (5V):**

- Again, the left image represents the amplitude, and the right image represents the phase.

From these images, we can observe the differences in piezoresponse under different voltage conditions. The amplitude images likely represent how much the material is responding to the applied voltage, and the phase images may indicate the direction of the response.

- **Amplitude Differences:** There are variations in the intensity of the amplitude images across different voltages, suggesting that the material's piezoresponse changes with the applied voltage.
- **Phase Differences:** The phase images also show variations, indicating changes in the direction or nature of the piezoresponse.

For a detailed quantitative analysis, further processing and statistical analysis would be necessary, including comparing specific regions of interest across the different conditions. [-]

---

Message ChatGPT...

---

ChatGPT can make mistakes. Consider checking important information.

CONDITIONS. U-J

Y You

The amplitude and phase data should be in the Raw\_Data\_SHO\_Fit, please replot and analyze their difference again

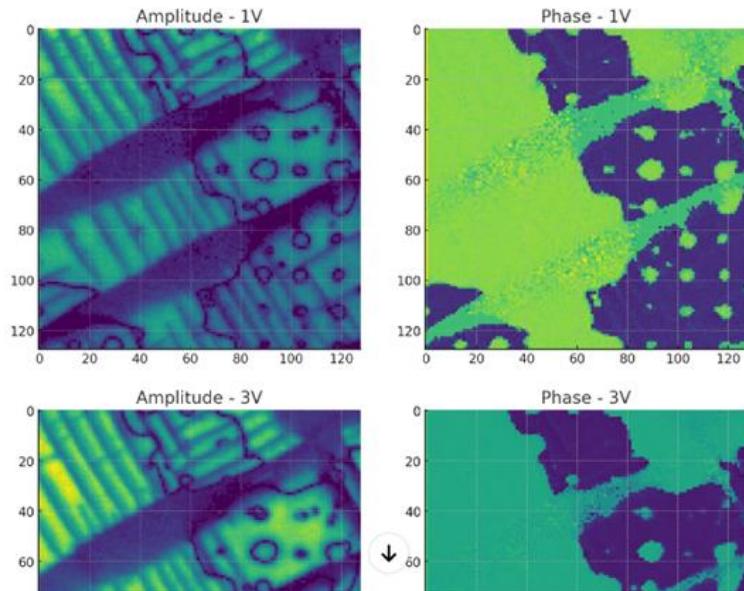
ChatGPT

! Error analyzing

It seems there was an error in the attempt to extract the amplitude and phase data from the 'Raw\_Data-SHO\_Fit\_000/Guess' dataset. The error indicates that the dataset might be structured differently than I assumed. I'll inspect the structure of this dataset to better understand how the amplitude and phase data are organized and then plot them correctly. Let's proceed with inspecting the structure of the 'Guess' dataset in one of the files. [2]

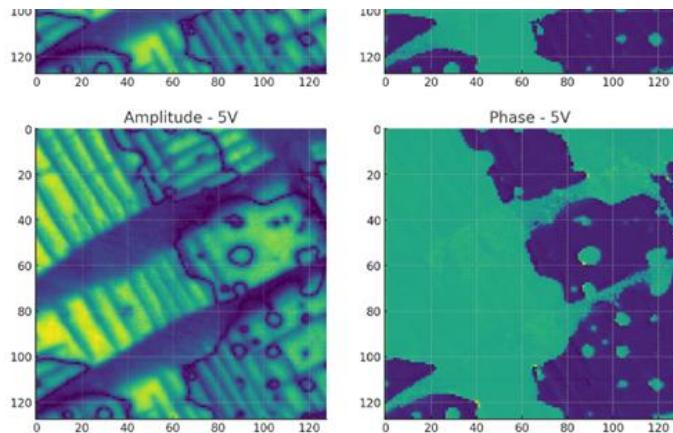
The 'Guess' dataset within the 'Raw\_Data-SHO\_Fit\_000' group of the BE-PFM file has a structure with multiple fields, as seen in the sample data. Each entry in the dataset is a structured array with several fields, including 'Amplitude [V]' and 'Phase [rad]'. The dataset shape is  $16384 \times 1$ , which indicates a single column with multiple fields.

To plot the amplitude and phase, I will extract the 'Amplitude [V]' and 'Phase [rad]' fields from this dataset, reshape them into a  $128 \times 128$  grid, and then plot them. Let's proceed with this approach for each file. [2]



Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.



The amplitude and phase images for each of the BE-PFM datasets have been re-plotted, this time using the correct 'Raw\_Data-SHO\_Fit\_000/Guess' dataset. Each pair of images corresponds to one of the files, with different voltage conditions (1V, 3V, 5V):

**1. First File (1V):**

- Left image: Amplitude.
- Right image: Phase.

**2. Second File (3V):**

- Left image: Amplitude.
- Right image: Phase.

**3. Third File (5V):**

- Left image: Amplitude.
- Right image: Phase.

Analysis of Differences:

- **Amplitude Variations:** The intensity and distribution of the amplitude images vary across the different voltages, indicating changes in the material's piezoresponse. Higher voltages might be causing more pronounced piezoresponse, as seen in the increased intensity in the amplitude images.
- **Phase Variations:** The phase images show distinct patterns for each voltage condition. These differences suggest changes in the direction or nature of the piezoresponse with varying voltage.

These observations align with the expectation that the piezoresponse of materials

---

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

These observations align with the expectation that the piezoresponse of materials changes under different applied voltages, both in magnitude (amplitude) and in the response characteristic (phase). For a more comprehensive analysis, a detailed examination of specific areas or statistical analysis of the amplitude and phase values would be beneficial. ↴

**Supplementary Note III: Record of using ChatGPT to program NanoSurf workflow**

Y You



Script Programmers Manual.pdf  
PDF

Attachment is a guideline of programmer script for NanoSurf AFM.

ChatGPT

The "Script Programmers Manual" for the NanoSurf Atomic Force Microscope (AFM) software provides comprehensive guidance on scripting and integration with the software. Here are the key aspects:

### Introduction

- The manual serves as a reference for the COM Automation interface of the NanoSurf software.
- It explains the concepts behind the interface and provides an object reference of all classes, methods, and properties published by NanoSurf [1].

### Motivation and Capabilities

- The NanoSurf control software offers a graphical user interface for general tasks and supports scripting for specific, advanced, or custom tasks.
- Script objects allow access to online microscope controls (like scan range or feedback set point) and post-processing of data [2].
- Scripting can automate repetitive tasks, write custom data analysis algorithms, extend functionality, and build complex systems [3].
- However, the scripting interface doesn't allow real-time processing or signal modification, nor the creation of new z-feedback control algorithms or operating modes [4].

### Usage and Examples

- The manual explains the general concept of scripting technology and how to integrate the software with other applications.
- A step-by-step tutorial, script examples, and a detailed object reference are provided [5].
- Embedded VBScript is used for scripting within the NanoSurf software [6].
- Example scripts demonstrate how to connect to and control various aspects of the software, like starting a scan [7].

### External Integration and Script Development

- Scripts can be developed in the integrated script editor or in external text editors like Notepad and saved with a .vbs extension [8].

U

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

Notepad, and saved with a .vbs extension [1].

- Scripts stored in a specific directory can be directly accessed and executed from the NanoSurf software menu [1].
- The NanoSurf application can act as a server, allowing integration with various programming environments and software packages through the COM Automation standard [1].
- Windows Scripting Host (WSH) is used to execute scripts, with support for VBScript and JavaScript [1].

## Integration with Specific Programming Environments

- **Visual C++:** The manual provides a guide for integrating the NanoSurf object interface using Visual C++ 6, including how to generate C++ wrapper classes [1].
- **LabView:** LabView's ActiveX function blocks can be used to control NanoSurf functionality, including methods to start the server program, send commands, and read or write properties [1].
- **Python:** Python can be used for scripting with the NanoSurf software, and the manual provides detailed instructions for setting up and writing Python scripts [1].

This comprehensive guide allows users with various levels of programming expertise to effectively utilize scripting to enhance and customize their experience with NanoSurf AFM software.



### You

Please design a Python script to move the motorized stage 100 times with an interval of 100 um.



### ChatGPT

To design a Python script for controlling a motorized stage connected to a Nanosurf system, you'll need to follow these general steps:

1. **Initialize the Nanosurf System:** Create a control object for the Nanosurf SPM (Scanning Probe Microscope) controller.
2. **Motorized Stage Control:** Access the motorized stage control object from the Nanosurf application.
3. **Movement Loop:** Implement a loop to move the stage 100 times, each time by 100 µm.
4. **Delay:** Include a delay or pause if necessary, to allow the stage to complete each movement before the next one begins.
5. **Error Handling:** Add error handling to manage any issues during the stage movements.



Here's an example script to illustrate these steps:



Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

Here's an example script to illustrate these steps:

```
python                                         ⌂ Copy code

import nanosurf
import time

# Create control object for the Nanosurf SPM controller.
spm = nanosurf.SPM()
application = spm.application

# Assuming 'Stage' is the correct class for motorized stage control.
# Replace 'Stage' with the actual class name if it's different.
motorized_stage = application.Stage

# Define the movement parameters
num_movements = 100
movement_interval_um = 100 # in micrometers

# Loop to move the stage
for i in range(num_movements):
    try:
        # Move the stage
        # Replace 'move' with the actual method name for moving the stage
        # and adjust parameters according to the method's requirements.
        motorized_stage.move(movement_interval_um)

        # Wait for the movement to complete before the next iteration
        # This delay might need adjustment based on actual stage movement
        time.sleep(1) # wait for 1 second (adjust as necessary)

    except Exception as e:
        print(f"Error occurred during movement {i}: {e}")
        break # Exit the loop if an error occurs

    # Clean up
    del motorized_stage
    del application
```

**Note:**

- Replace `'**Stage**'` and `'**move**'` with the actual class name and method name used for controlling the motorized stage in the Nanosurf system.
- The `**time.sleep(1)**` function is used to introduce a delay of 1 second between movements. This may need to be adjusted  based on the actual time it takes for the stage to move.

---

 Message ChatGPT...

---

ChatGPT can make mistakes. Consider checking important information.

- for controlling the motorized stage in the Nanosurf system.
- The `time.sleep(1)` function is used to introduce a delay of 1 second between movements. This may need to be adjusted based on the actual time it takes for the stage to move.
  - Error handling is included to catch any exceptions that might occur during the movements.
  - This script assumes that the necessary Nanosurf Python module is properly installed and configured as per the manual's instructions [1].

 You

There is no 'move' function in stage

 ChatGPT

Understood. Without a specific 'move' function for the motorized stage in the Nanosurf API, we'll have to approach this differently. We can use available methods and properties to control the stage's position. Assuming there are methods to set the X, Y (and possibly Z) coordinates of the stage, the script can be adapted as follows:

1. **Get Current Position:** First, get the current position of the stage.
2. **Calculate New Position:** Then, calculate the new position by adding the movement interval to the current position.
3. **Set New Position:** Use the method to set the new position of the stage.
4. **Repeat the Process:** Repeat this for the specified number of movements.

Here's a revised example script:

```
python  Copy code

import nanosurf
import time

# Create control object for the Nanosurf SPM controller.
spm = nanosurf.SPM()
application = spm.application

# Assuming 'Stage' is the correct class for motorized stage control.
# Replace 'Stage' with the actual class name if it's different.
motorized_stage = application.Stage

# Define the movement parameters
num_movements = 100
movement_interval_um = 100 # in micrometers
```

 Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

```

motorized_stage = application.Stage

# Define the movement parameters
num_movements = 100
movement_interval_um = 100 # in micrometers

# Loop to move the stage
for i in range(num_movements):
    try:
        # Get the current position
        current_x, current_y = motorized_stage.PositionX, motorized_stage.PositionY

        # Calculate the new position
        new_x = current_x + movement_interval_um
        new_y = current_y + movement_interval_um # Adjust if necessary

        # Move the stage to the new position
        motorized_stage.PositionX = new_x
        motorized_stage.PositionY = new_y

        # Wait for the movement to complete before the next iteration
        time.sleep(1) # Adjust based on actual stage movement speed

    except Exception as e:
        print(f"Error occurred during movement {i}: {e}")
        break # Exit the loop if an error occurs

    # Clean up
del motorized_stage
del application

```

**Note:**

- The script assumes that the `PositionX` and `PositionY` (and possibly `PositionZ`) properties are available for the motorized stage.
- The delay (here `time.sleep(1)` ) might need adjustment based on the actual movement speed and stability requirements of your setup.
- Ensure that the movement does not exceed the stage's physical limits.
- This script is a basic template and might need to be adjusted based on the exact methods and properties available in the Nanosurf API.



You

Please design a script for writing square domains in a ferroelectric thin film



Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.



You

Please design a script for writing square domains in a ferroelectric thin film



ChatGPT

Designing a script for writing square domains in a ferroelectric thin film using a Nanosurf system involves controlling the tip of the atomic force microscope (AFM) to apply an electric field across specific areas of the thin film. This is typically done by moving the AFM tip to defined positions and applying a voltage for a certain duration to switch the polarization in those areas.

Here's a conceptual outline of such a script:

1. **Initialization:** Set up the necessary objects and parameters for the AFM controller and the tip positioning system.
2. **Defining Parameters:**
  - Size of the square domain.
  - Voltage to be applied.
  - Duration of voltage application.
3. **Domain Writing:**
  - Move the tip to the start position.
  - Apply the voltage in a square pattern.
  - Incrementally move the tip in a grid-like pattern within the square.
  - Apply voltage at each point or along each line.
4. **Finalization:** Reset the system or move the tip to a safe location.

Below is a simplified Python script to illustrate this process. Please note, this script is hypothetical and depends on the specific capabilities and functions available in your Nanosurf AFM system.

```
python Copy code  
  
import nanosurf  
import time  
  
# Initialize the Nanosurf SPM controller  
spm = nanosurf.SPM()  
application = spm.application  
tip_control = application.Tip # Hypothetical object for tip control  
  
# Define parameters for square domain  
square_size = 10e-6 # 10 micromet ↓  
voltage = 10 # 10 Volts
```



Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

```

application = spm.application
tip_control = application.Tip # Hypothetical object for tip control

# Define parameters for square domain
square_size = 10e-6 # 10 micrometers
voltage = 10 # 10 Volts
pulse_duration = 0.1 # 100 milliseconds
step_size = 1e-6 # 1 micrometer

# Function to apply voltage
def apply_voltage(duration):
    tip_control.Voltage = voltage
    time.sleep(duration)
    tip_control.Voltage = 0

# Writing the square domain
try:
    for x in range(int(square_size / step_size)):
        for y in range(int(square_size / step_size)):
            # Move tip to the position
            tip_control.PositionX = x * step_size
            tip_control.PositionY = y * step_size

            # Apply voltage
            apply_voltage(pulse_duration)

except Exception as e:
    print(f"Error during domain writing: {e}")

# Clean up
tip_control.Voltage = 0 # Ensure voltage is off
del tip_control
del application

```

#### Important Notes:

- **Safety and Equipment Limits:** Ensure that the script respects the safety limits of the equipment. Overvoltage or rapid movements can damage the tip or the sample.
- **Actual API Functions:** Replace the hypothetical function calls (`Tip`, `PositionX`, `PositionY`, `Voltage`) with actual methods from the Nanosurf API.
- **Verification:** After each voltage application, you might want to verify the state of the ferroelectric domain, which can be complex and might require additional sensing or imaging steps.
- **Script Testing:** Thoroughly test the script in a controlled environment before applying it to valuable samples to ensure it behaves ↓ expected.

 Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

**Supplementary Note IV: Record of using ChatGPT to program Zurich  
Instrument workflow**

1. We trained ChatGPT with the Zurich Instruments manuals:

**You**



LabOneProgrammingManual.pdf

PDF



ziHDAWG\_UserManual.pdf

PDF



ziHF2\_UserManual\_ziControl.pdf

PDF

I am going to upload some manual of instructions of the Zurich Instruments lockin amplifier. I want you to study them and help me perform some tasks after

o

**ChatGPT**

I have now accessed the manuals for the Zurich Instruments lock-in amplifier, including the LabOne Programming Manual, ziHDAWG User Manual, and the ziHF2 User Manual for ziControl. Please let me know the specific tasks or questions you have in mind related to these documents, and I'll do my best to assist you!

2. We asked him to provide us with python scripts to perform a very easy task like apply a 2V DC step through the signal Output 1 channel:

You

Can you provide me with a python script that gets control over the labone User Interface and sends a 2V DC through the signal output 1 of the instrument



ChatGPT

Great to hear that the initial connection is working for you. Based on your updated script, it seems that you're using `zhinst.utils.autoConnect()` and `zhinst.utils.autoDetect()` to establish the connection and detect the device. Let's modify the rest of the script accordingly.

Here's the updated script that reflects these changes and sets a 2V DC output:

```
python Copy code
import zhinst.ziPython
import zhinst.utils

# Discover and connect to the device
d = zhinst.ziPython.ziDiscovery()
d.find('DEV600') # Replace 'DEV600' with your actual device ID
devProp = d.get('DEV600')
daq = zhinst.utils.autoConnect()
device = zhinst.utils.autoDetect(daq)

# Assuming 'device' now holds the identifier for your connected device

# Set the output range and offset for a 2V DC signal
# Adjust the output channel as needed
daq.setDouble(f'{device}/sigouts/0/range', 1) # Set output range to 1
daq.setDouble(f'{device}/sigouts/0/offset', 2) # Set offset to 2V for

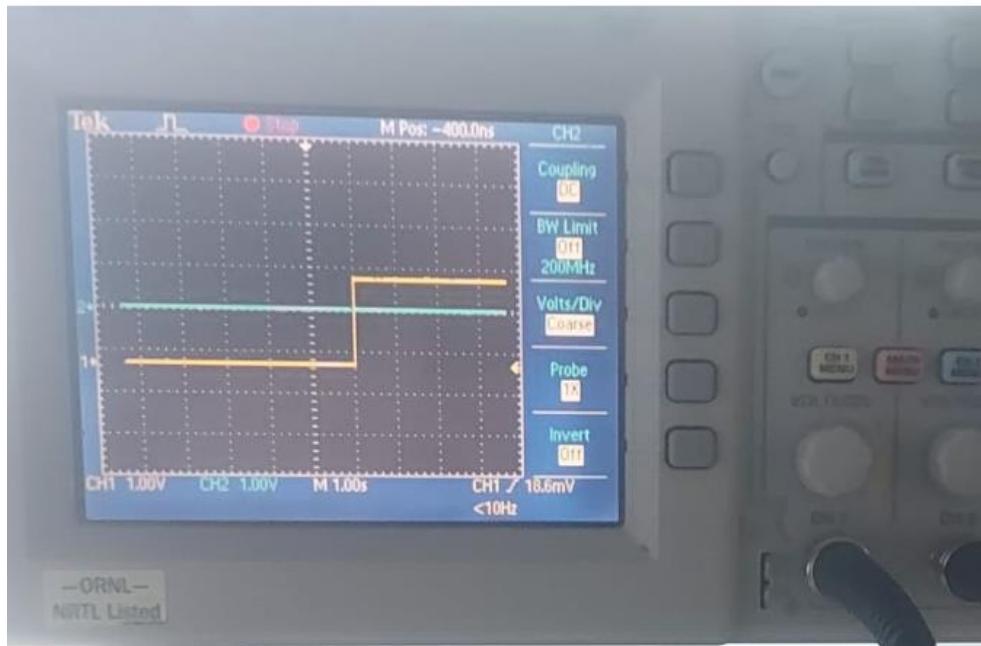
# Enable the output
daq.setInt(f'{device}/sigouts/0/on', 1)
```

In this script:

- `autoConnect()` is used to automatically connect to the data server.
- `autoDetect()` is used to automatically detect the connected device. It will return the device identifier, which is then used in the subsequent commands.
- The `daq.setDouble()` and `daq.setInt()` functions are used to set the output range, offset, and to enable the output, respectively. The `device` variable is used to dynamically insert your device's identifier into the command strings.

Make sure that the `range` and `offset` values are appropriate for your device and what you're trying to achieve. Also, if your device has a different configuration or requires additional settings, you'll need to adjust the script accordingly.

3. Then test that it works experimentally:



4. Last, we provided him with a more complex task. That is we uploaded the instruction manual to set up a rather complicated experiment like Heterodyne KPFM (which includes the combination of several instruments, not only the Zurich lockin amplifier), and asked him to provide us with the python script necessary to perform such experiment.

You



HeterodyneKPFMManual.pdf  
PDF

Now, I am uploading a pdf with the instructions to set up heterodyne KPFM using the Zurich lockin amplifier. Can you write a python that sets up heterodyne KPFM, please:

5. ChatGPT was able to understand the complexity of the task it was assigned, and it provided us with an accurate a step by step process of the steps to follow for heterodyne operation.



### ChatGPT

To set up heterodyne Kelvin Probe Force Microscopy (KPFM) using the Zurich Instruments lock-in amplifier, a specific sequence of settings and operations must be performed. This involves:

1. **Preparing the Instrument and Sample:** Setting up the sample, cantilever, and ensuring a proper approach in AC mode.
2. **Configuring the LabOne User Interface:** This includes setting up signal inputs, demodulators, output amplitudes, and signal outputs. For example, switching on AC, setting demodulator and oscillator frequencies, adjusting output ranges, and enabling outputs.
3. **Setting up the Modulation (MOD) Tab:** Here, you will manually set the mode, carrier, and sidebands, as well as manage amplitudes and enable the required settings.
4. **Adjusting the Auxiliary (AUX) Outputs:** This step involves setting the signal and offset for AUX outputs, ensuring they are appropriate for your KPFM measurement.
5. **Utilizing the Sweeper and PID Tabs:** This involves configuring sweep parameters, setting start and stop frequencies, and adjusting PID settings for feedback loops.
6. **Final Adjustments and Measurements:** This includes zeroing the phase, setting up feedback loops for open or closed loop KPFM, and monitoring input and output channels during scanning.

Translating these steps into a Python script requires a deep understanding of both the KPFM process and the Zurich Instruments API. The script would involve numerous API calls to set parameters like frequency, phase, amplitude, and feedback loops according to the specifics of the heterodyne KPFM process.

Given the complexity and the specific nature of this setup, it is highly recommended to closely follow the manual you provided and carefully translate each step into the corresponding API calls based on your exact setup and requirements. Unfortunately, without direct access to your setup and specific configuration details, providing a detailed and accurate script is challenging.

If you have specific questions about certain steps or API calls, feel free to ask!

- Finally, we asked him to provide also with a python script of the part related to the control and set up of experimental parameters for heterodyne KPFM:

**You**

I know that it involves more than just the zurich lockin amplifier settings, but can you provide the script to set up the lockin amplifier settings only

7. Then it provides with a script that performed only partially the setting up of the heterodyne KPFM:

 ChatGPT

Based on the provided information from the Heterodyne KPFM manual, a Python script to set up the Zurich Instruments lock-in amplifier for a FM Heterodyne KPFM measurement typically involves configuring various parameters such as tracking the topography at the first eigenmode, setting the electrical excitation, and detecting the electrical signal.

Here is a basic Python script template to help you get started. Please note that you'll need to adjust the specific parameters (like frequencies and device node paths) according to your experimental setup and the capabilities of your specific Zurich Instruments model.

```
python Copy code

import zhinst.ziPython
import zhinst.utils

# Discover and connect to the device
d = zhinst.ziPython.ziDiscovery()
d.find('DEV600') # Replace 'DEV600' with your actual device ID
devProp = d.get('DEV600')
daq = zhinst.utils.autoConnect()
device = zhinst.utils.autoDetect(daq)

# Assuming 'device' holds the identifier for your connected device
# Replace 'freq_omega0' and 'freq_omegai' with your actual frequencies

freq_omega0 = ... # Set your first eigenmode frequency here
freq_omegai = ... # Set your detection frequency here
freq_excitation = freq_omegai - freq_omega0

# Set the oscillator frequencies
daq.setDouble(f'{device}/oscs/0/freq', freq_omega0)
daq.setDouble(f'{device}/oscs/1/freq', freq_excitation)

# Configure demodulator settings for detecting the signal
# Adjust these parameters based on your requirements
daq.setDouble(f'{device}/demods/0/freq', freq_omegai)
daq.setDouble(f'{device}/demods/0/harmonic', 1)
daq.setDouble(f'{device}/demods/0/order', 4) # Example filter order
daq.setDouble(f'{device}/demods/0/timeconstant', 0.1) # Example time

# Enable the output for the excitation signal
daq.setInt(f'{device}/sigouts/0/on', 1)
daq.setDouble(f'{device}/sigouts/0/range', 10) # Example range in vol
```

8. We hypothesize that the main missing parts needed to fully provide us with a working script for this much more complex task are contained in the Figures (not the text) of those manuals, hindering it from ChatGPT's ability to learn such steps. However appropriate manuals targeted to LLMs learning capabilities should work better, which we consider out of the scope of this work.