Welcome to AEcroscopy for Automated and Autonomous Microscopy

Contents

Under construction (Here, we will introduce automated and autonomous experiment, the requirement of autonomous experiment, the objective of AEcroscopy…)

AEcroscopy is a cutting-edge Python package that integrates with our self-developed Application Programming Interface (API) BEPyAE,aiming to revolutionizing microscopy measurements. AEcroscopy enables easy implementation of experiments formulated with Python program……allowing users to orchestrate complex experiments, design workflows, and gather data more efficiently.

In addition, AEcroscopy allows users to integrate machine leanring models into the operating experiment, harnessing the power of AI for real-time data analysis and decision-making. This will enable rapid iterations in experiment and largely accelerate scientific discovery.

With AEcroscopy, a microscopy capable of automating operation, actively learning from on-the-fly results, adapting experiments, optimizing parameters, driving real-time decision-making… becomes a reality.

Discover the potential of AEcroscopy and experience the future of microscopy experimentation.

Yongtao Liu, June 2023

Introduction to AEcroscopy

Skip to main content

Welcome to our comprehensive guide to the AEcroscopy for scanning probe microscopy (SPM). AEcroscopy is a decdicated 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, a 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.

Yongtao Liu, *June* 2023

Guide to Essential Commands and Functionalities in AEcroscopy

Yongtao Liu

June 2023

Welcome to our comprehensive guide to the AEcroscopy for scanning probe microscopy (SPM). AEcroscopy is a decdicated 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, a 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

Skip to main content

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 Acquisition_v0_9 import Acquisition   # include the Acquistion_v.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_
```

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
                                               # executing this will also initlize AR18
```

Skip to main content

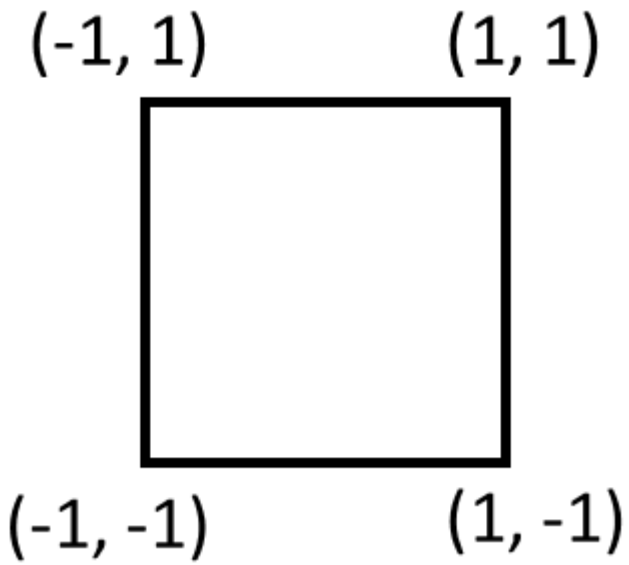
Hereinafter

- 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]




```
newexp.tip_control(tip_parms_dict = {"set_point_V_00": 1, "next_x_pos_00": 0.5, "next_y_pos_01"
                   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]
```

Skip to main content

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})
```

```
('0 Cypher AR18',
 '6124',
 4000000.0,
 10.0,
 10.0,
 'AC and DC on AO0',
 1.0,
 'none',
 'none',
 'none',
 'external')
```

```
BEPFM Measurement
```

Set BE pulse parameters

One of the most popular experiment we can perform with AEcroscopy-BEPyAE.exe is the band excition (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)
```

Skip to main content

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 fucntion is just a single BE line scan, not a raster image
- This fucntion returns 5 datasets: quick_fitting, complex spectra, and 3 channel data

```
# Do a single line scan
qk_fit, com_spec, chn1, chn2, chn3 = newexp.do_line_scan(line_scan_parms_dict = {"num_BE_pulses
                                                                                 "start_x_pos_0
                                                                                 "stop_x_pos_02
                                                                    upload_to_daq = True, do_line_scan = T
```

```
voltage offset and number of BE pulse are:  (0.0, 32)
line scan start and end positions:  (-0.5, 0.0, 0.5, 0.0)
```

BE Raster Scan

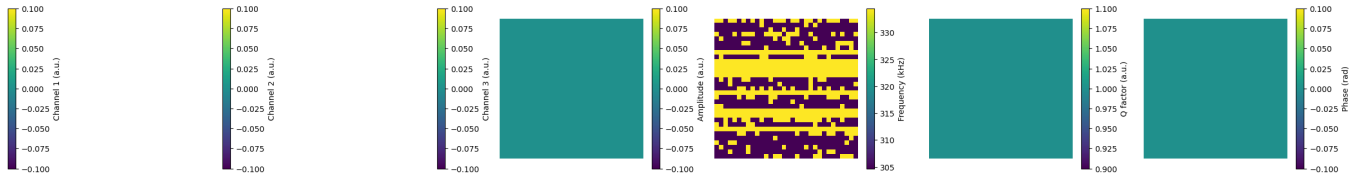
We can also perform a square raster scan BE measurement using the function $raster\_scan()$. In this function, we need to provide the raster scan pixel (i.e., $scan\_pixel$) and scan region (i.e., scan start and stop points).

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_
                                                                       "scan_y_start": -0.8,"sc
                                                                       "scan_y_stop": 0.8}, fil

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

Skip to main content



```
 [progress: 0:00:38] |**********************************  | (ETA:   0:00:01) C:\Users\yla\AppD
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```

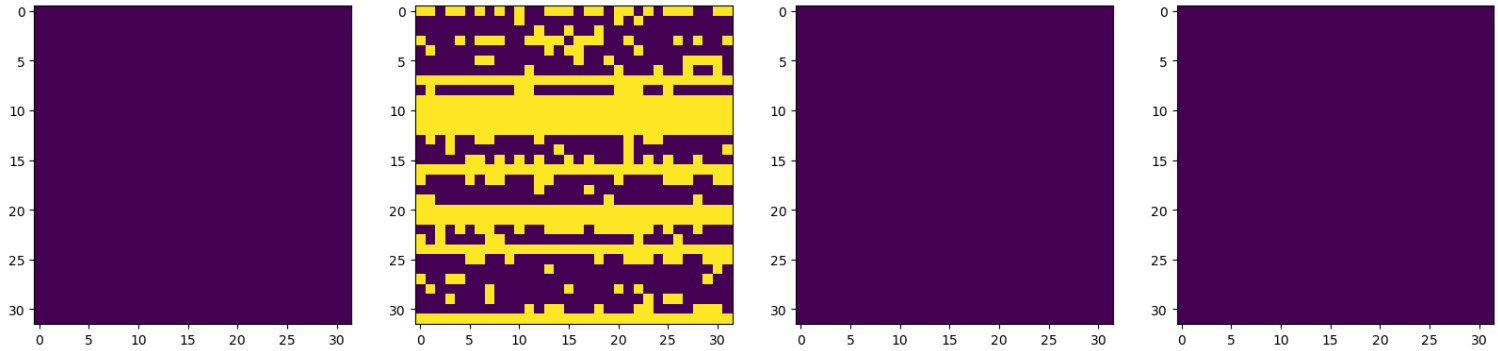
BE Raster Scan Results

We can visualize BE raster scan results inlcuding 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])
ax2.imshow(dset_pfm[:,:,1])
ax3.imshow(dset_pfm[:,:,2])
ax4.imshow(dset_pfm[:,:,3])
```

```
sidpy.Dataset of type IMAGE_STACK with:
 dask.array<array, shape=(32, 32, 5), dtype=float64, chunksize=(32, 32, 5), chunktype=numpy.ndar
 data contains: quick fit pfm (generic)
 and Dimensions:
y axis:  y axis (m) of size (32,)
x axis:  x axis (m) of size (32,)
BE responses:  channels (generic) of size (5,)
```

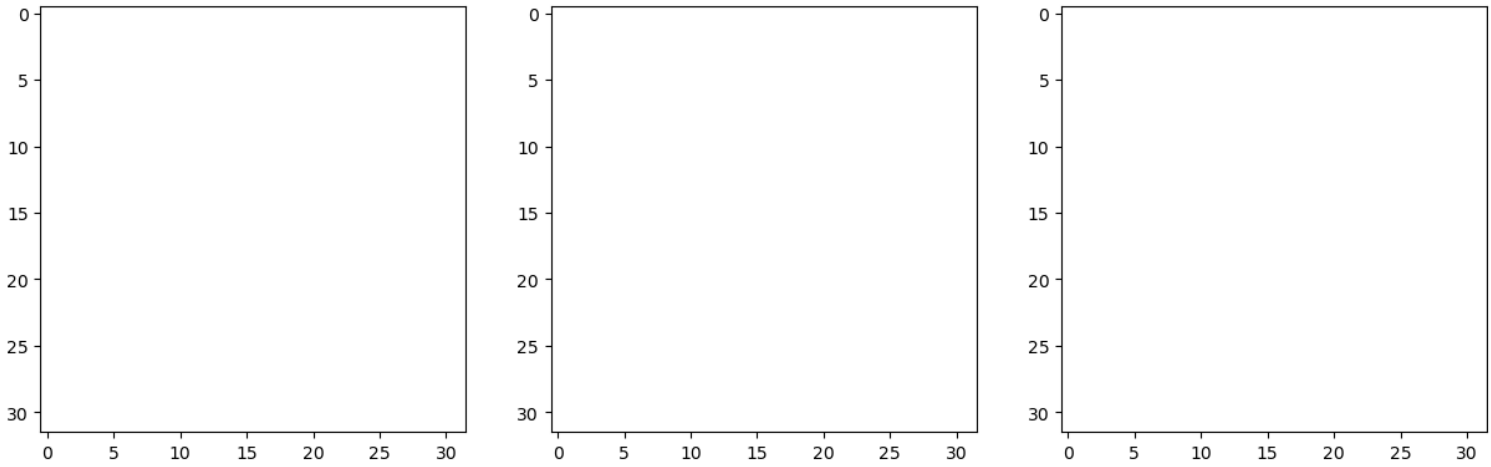
```
<matplotlib.image.AxesImage at 0x1ba0b485a50>
```



```
# 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,:,:])
```

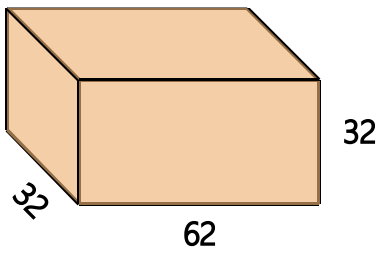
```
sidpy.Dataset of type IMAGE_STACK with:
 dask.array<array, shape=(3, 32, 32, 1), dtype=float64, chunksize=(3, 32, 32, 1), chunktype=nump
 data contains: channels (generic)
 and Dimensions:
y axis:  y axis (m) of size (3,)
x axis:  x axis (m) of size (32,)
channels images:  channels (generic) of size (32,)
d:  generic (generic) of size (1,)
```

```
<matplotlib.image.AxesImage at 0x1ba0b4448e0>
```



```
# sidpy dataset of complex spectra
dset_cs
```

	Array	Chunk
Bytes	0.97 MiB	0.97 MiB
Shape	(32, 32, 62)	(32, 32, 62)
Count	1 Tasks	1 Chunks
Type	complex128	numpy.ndarray



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)
```

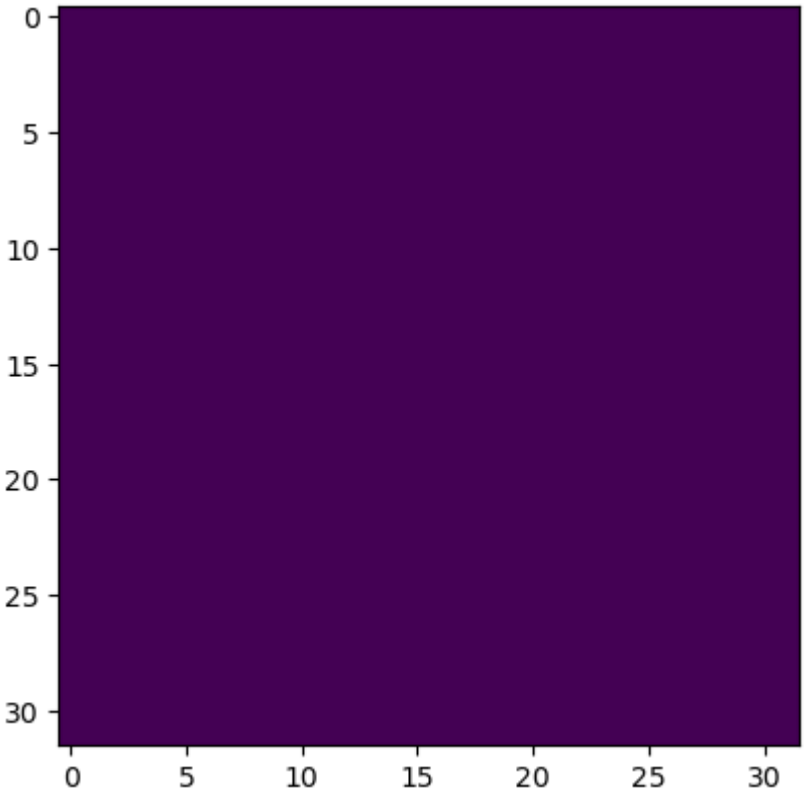
Skip to main content

```
/
├ BE Channels
  -----------
  ├ Channels
    --------
    ├ Channels
    ├ channels images
    ├ d
    ├ x axis
    ├ y axis
├ BE Complex Spectra
  ------------------
  ├ Complex Spectra
    ---------------
    ├ Complex Spectra
    ├ c
    ├ location index x
    ├ location index y
├ BE Parameters
  -------------
  ├ frequency
  ├ pulse parameters
  ├ scan size
├ BE Quick Fitting
  ----------------
  ├ Quick Fitting
    -------------
    ├ BE responses
    ├ Quick Fitting
    ├ x axis
    ├ y axis
```

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

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

```
<matplotlib.image.AxesImage at 0x1ba0c06efb0>
```



BEPS Measurements

Set BEPS parameters

In addition to BE piezoresponce force microscopy image measurement, we can also do BE piezoresponse 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 wavefrom 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": 3
                                                 "num_cycles_04": 3, "cycle_fraction_05": 0,
                                                 "cycle_phase_shift_06": 0, "measure_loops_07":
                                                 "transition_time_s_08": 1E-3, "delay_after_ste
```

Skip to main content

```
BEPS parameters are:  (8.0, 0.0, 0.0, 32, 3, 0, 0, 0, 0.001, 0.0, 0.0, 0.0, 1, 0, 8.0, 8.0, 7.0,
```

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 $beps\_grid\_parms\_dict()$

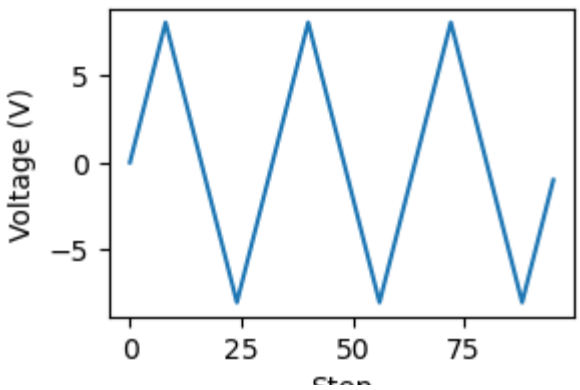
```
# do BEPS
beps_waveform, beps_quick_fit, beps_cx, beps_chns = newexp.do_beps_grid(beps_grid_parms_dict =

                                                                        file_name = "BEPS_grid")
```

```
 [progress: 0:00:39] |**********************************  | (ETA:   0:00:01) C:\Users\yla\AppD
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```

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")
```

```
Text(0.5, 0, 'Step')
```

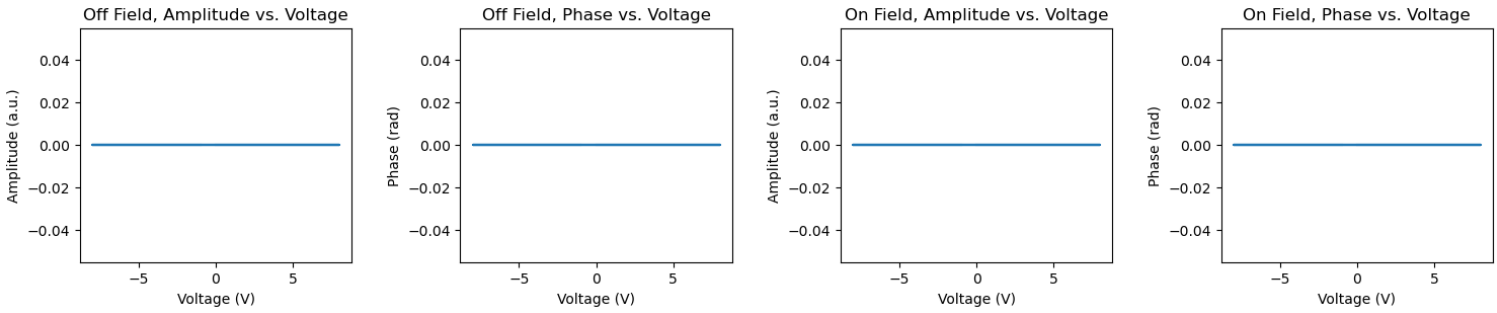


Skip to main content

Druing 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 seperate 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)")
```

```
Text(0, 0.5, 'Phase (rad)')
```



Skip to main content

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.as
                                                                            file_name = "BEPS_a
```

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 coordiante 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,
print(converted_coor)
```

```
[-0.21875   0.015625]
```

After we have the coordiante 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 = conve
                                                                            file_name = "BEPS_a
```

```
 [progress: 0:00:01] |                                          | (ETA:  --:--:--) C:\Users\yla\AppD
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```

Apply a pulse

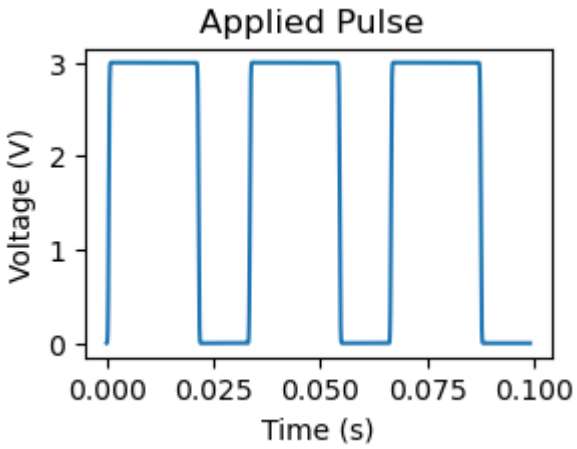
Skip to main content

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, "pu
                                                            "pulse_final_amplitude_V_02": 0, "p
                                                            "rise_time_s_05": 1E-3, "pulse_fina
                                                            "pulse_repeats_06": 3},
                                        do_create_pulse = True, do_upload_pulse = True, do_appl
# 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)")
```

```
Text(0, 0.5, '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.

Skip to main content

```
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)
```

```
[progress: 0:00:02] |****************************        | (ETA:   0:00:00)
```

Capability with FPGA

In AEcroscopy-BEPyAE.exe system, users have various options of scan trajectories when equiped with FPGA, such as sprial 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.

```
sprial_results = newexp.fpga_spiral_scan(spiral_parms_dict = {"spiral_inner_radius_x_V_00": 0,
                                                              "spiral_inner_radius_y_V_02": 0,
                                                              "spiral_N_cycles_04": 5, "spiral_
                                                              "spiral_dose_distribution_06": 1,
                                                              "spiral_return_opt_08": 0, "scan_
                                                              "scan_y_offset_V": 0, "scan_rotat
                                         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.

Skip to main content

```
# 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"
                                                                       "amplitude_V_02": 1, "ph
                                                                       "repeats_04": 4, "req_pu
                                                      do_create_be_waveform = True,
                                                      spiral_parms_dict = {"spiral_inner_radius
                                                                           "spiral_outer_radius
                                                                           "spiral_inner_radius
                                                                           "spiral_outer_radius
                                                                           "spiral_N_cycles_04"
                                                      num_BE_pulse = 128, do_scan_update = True
                                                      do_BE_arb_line_scan_01 = True, spiral_rec
```

```
BE parameters are:  (335000.0, 100000.0, 1.0, 1.0, 4, 0.004, 1, 3352.2952763920002, 0.121594590
```

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
                                             "final_x_position_V_02": 0.5, "final_y_position_V_
                        make_cur_pos_start_pos=False, do_probe_move_update=True, do_probe_move=
```

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_
                        make_cur_pos_start_pos=True, do_probe_move_update=True, do_probe_move=T
```

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

shown below.

```
raster_full = newexp.fpga_linebyline_raster_scan(line_by_line_raster_dict = {"raster_scan_size_
                                                                              "raster_scan_size_
                                                                              "raster_N_scan_lin
                                                                              "raster_line_durat
                                                                              "scan_x_offset_V_0
                                                                              "scan_y_offset_V_0
                                                                              "scan_rotation_deg
                                                 initialize_line_by_line_raster=True, do_full_r
                                                 wait_to_advance_to_next_line=False, do_next_ra
                                                 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_line = newexp.fpga_linebyline_raster_scan(line_by_line_raster_dict = {"raster_scan_size_
                                                                              "raster_scan_size_
                                                                              "raster_N_scan_lin
                                                                              "raster_line_durat
                                                                              "scan_x_offset_V_0
                                                                              "scan_y_offset_V_0
                                                                              "scan_rotation_deg
                                                 initialize_line_by_line_raster=True, do_full_r
                                                 wait_to_advance_to_next_line=True, do_next_ras
                                                 stop_full_raster_scan=False)
```

High Throughput Experimentation

This chapter is about high throughput experiment workflows

...Under construction (we will introduce high-throughput experiment here)...

Yongtao Liu, *June* 2023

High Throughput Domain Writing Workflow

Yongtao Liu

Skip to main content

Install and Import

```
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 Acquisition_v0_9 import Acquisition   # include the Acquistion_v.py in the same directory
```

Start BEPyAE.exe and set VI

- Start BEPyAE.ext
- 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_
# exe_path is the directory of BEPyAE;
```

Initialize Igor AR18

- Set offline development
- Build a connection between BEPyAE and AR18
- Get parameters in AR18

```
newexp.init_BEPyAE(offline_development = True) # set offline_development=True if doing offline
                                               # executing this will also initlize AR18
```

> Hereinafter
>
> - If no parameters are provided in a function, executing the function will utilize the default parameters within $BEPyAE.exe$.
> - Certain functions provide feedback after execution, which can be disabled by setting

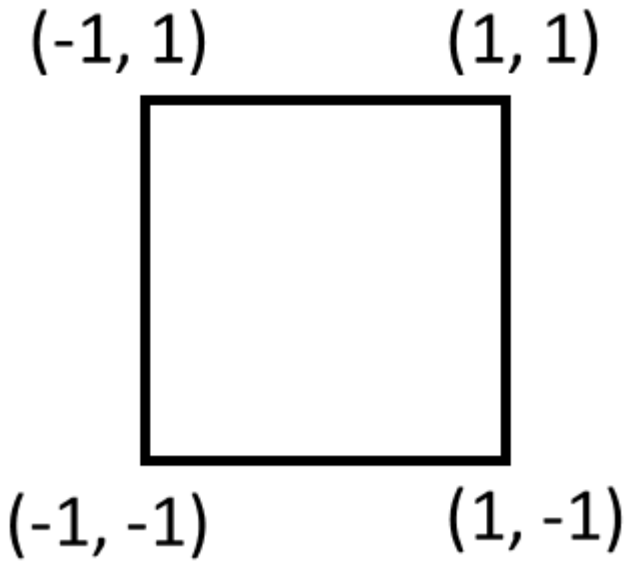
Skip to main content

instances when it is not required.

- Note: For tip locations, -1 corresponds to the left-hand side on the x-axis and the bottom side on the y-axis, while 1 corresponds to the right-hand side on the x-axis and the top side on the y-axis.

Set tip parameters

- set setpoint, tip locations




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

```
Setpoint is:  1.0
Tip parameters are:  (-0.5, 0.5, 0.2)
```

Set IO

This defines IO parameters, such as AFM platform: AR18, amplifiers, channel data types, etc

Skip to main content

```
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":
```

```
('0 Cypher AR18',
 '6124',
 4000000.0,
 10.0,
 10.0,
 'AC and DC on AO0',
 10.0,
 'topography',
 'current',
 'aux',
 'external')
```

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)
```

```
BE parameters are:  (335000.0, 100000.0, 1.0, 1.0, 4, 0.004, 1, 3352.2952763920002, 0.121594590
```

Run a BE Line scan to test parameters

- This is a single BE line scan
- This returns 5 datasets: quick_fitting, complex spectra, and 3 channels

```
# Do a single line scan
qk_fit, com_spec, chn1, chn2, chn3 = newexp.do_line_scan(line_scan_parms_dict = {"num_BE_pulses
                                                                                 "start_x_pos_0
                                                                                 "stop_x_pos_02
                                                                    upload_to_daq = True, do_line_scan = T
```

Skip to main content

```
voltage offset and number of BE pulse are:  (0.0, 32)
line scan start and end positions:  (-0.5, 0.0, 0.5, 0.0)
```

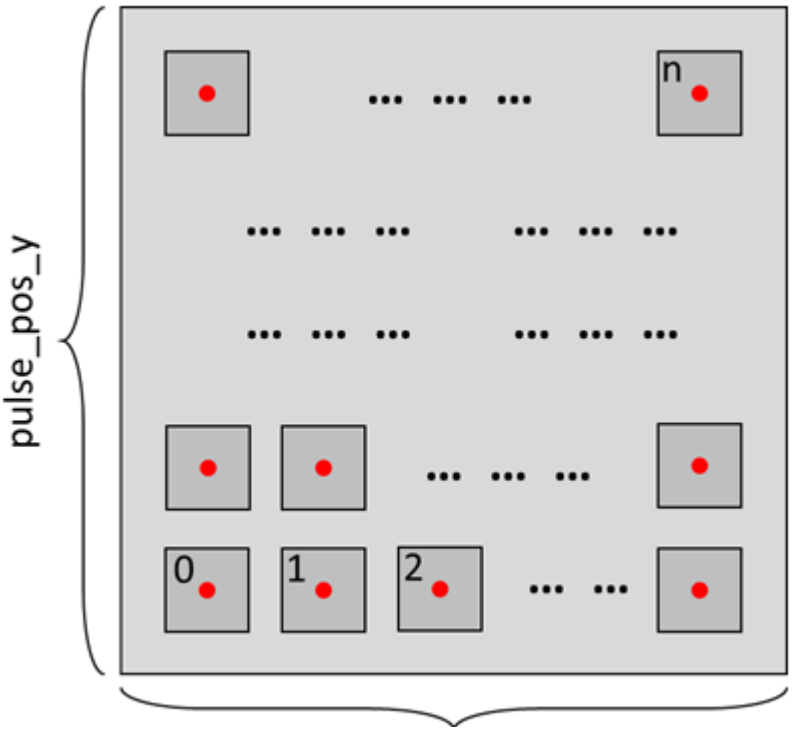
> *Tests Done*

Experiment Starts

> In this experiment, we begin by applying a DC pulse to switch the ferroelectric polarization. Subsequently, a BEPFM (Bias-Enhanced Piezoresponse Force Microscopy) measurement is conducted to image the domain structure.

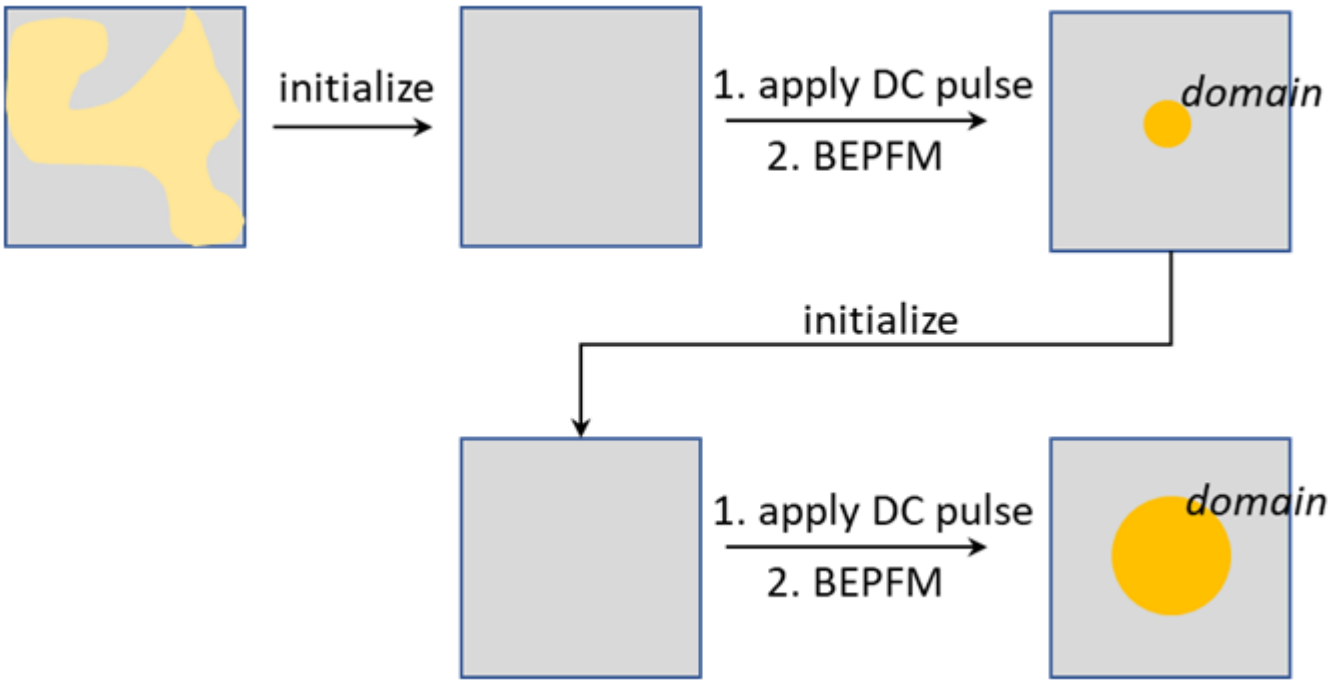
1. To initiate the measurement process, we first need to determine the location for each individual measurement. There are two scenarios to consider:

> - For each measurement, a new location is chosen, requiring a location array to record all the measurements as demonstrated below.

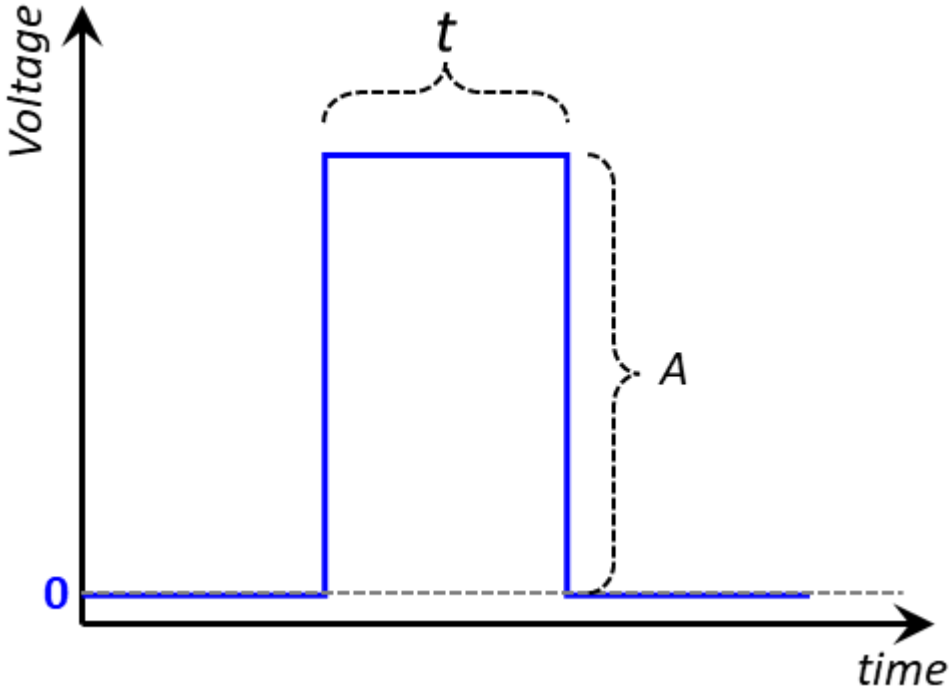


Skip to main content

- Alternatively, all measurements are conducted at the same location. In this case, the measurement location needs to be initialized each time a new measurement is started, as shown below.



2. Prior to experiments, we also need to establish the DC pulse parameters including pulse magnitude A and pulse length t , as shown below. Again, there are two scenarios to consider here:



- The pulse parameters can be pre-defined, e.g., the parameter values can be uniformly distributed within a specified range or customized to suit the experimental requirements.
- The pulse parameters can be random values within a defined space, typically these random values is a uniform distribution across that space in principle.

Experiment 1. Perform each measurement at a new location with pre-defined pulse parameters

Prior to expeirment, set a directory to save data

```
os.chdir(r"C:\Users\yla\Dropbox (ORNL)\My Files\AEcroscopy_BEPyAE")
```

Step 1. Generate a location array

Skip to main content

```
# All locations span across [start_point_x, end_point_x] in x-direction and [start_point_y, end_
# There are num_x rows and num_y columns in the locations array

start_point_x = -0.9   # Define location array parameters
end_point_x = 0.9
start_point_y = -0.9
end_point_y = 0.9
num_x = 2
num_y = 2

# Generate location array
pos_x = np.linspace(-0.9, 0.9, num_x)
pos_y = np.linspace(-0.9, 0.9, num_y)
pulse_pos = np.meshgrid(pos_x, pos_y)
pulse_pos_x = pulse_pos[0].reshape(-1)
pulse_pos_y = pulse_pos[1].reshape(-1)  # pulse_pos_x and pulse_pos_y are the coordinates of al

# Set BEPFM image size
img_size = 0.1

# Check
if img_size > np.abs(pos_x[0]-pos_x[1]):
    print ("Alert: there will be image overlap along x-direction")
elif img_size > np.abs(pos_y[0]-pos_y[1]):
    print ("Alert: there will be image overlap along y-direction")
else:
    print("{} locations are ready for experiments".format(len(pulse_pos_x)))
```

```
4 locations are ready for experiments
```

Step 2. Establish pulse parameters

```
# uniformly distributed pulse parameters
min_voltage = 5
max_voltage = 9
Vdc_amp = np.linspace(min_voltage, max_voltage, num_x)  # pulse magnitude

min_time_log = -4
max_time_log = 1
Vdc_time = np.linspace(min_time_log, max_time_log, num_y, dtype = np.float32())
Vdc_time = np.power(10, Vdc_time)                 # pulse time

# Establish pulse parameters
Vdc = np.meshgrid(Vdc_amp, Vdc_time)
Vdc_amp = Vdc[0].reshape(-1)
Vdc_time = Vdc[1].reshape(-1)

if len(Vdc_amp) > len(pulse_pos_x):
    print ("Error: No enough locations to test all pulse conditions")
else:
    print ("{} pulse parameters are ready for expierments".format(len(Vdc_amp)))

# save pulse condition
np.save("Vdc_list.npy", np.asarray([Vdc_amp, Vdc_time]))
```

```
4 pulse parameters are ready for expierments
```

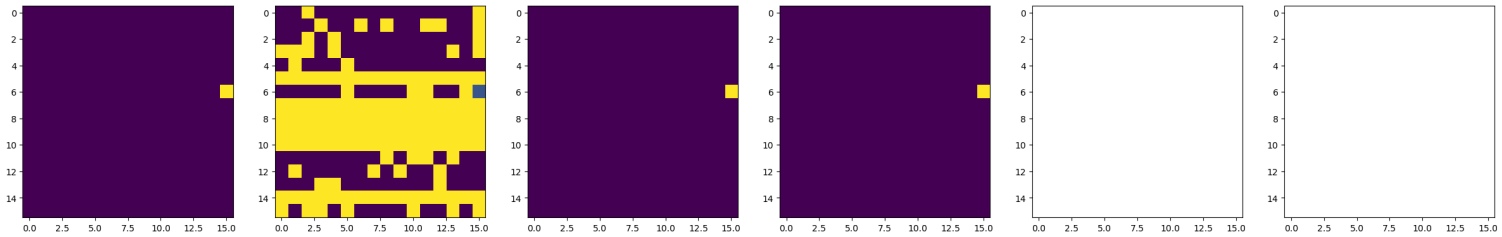
Step 3. Start experiment

```
for i in tqdm(range(len(Vdc_amp))):
    ####################---------- Move tip to the pulse location ----------##################
    newexp.tip_control(tip_parms_dict = {"set_point_V_00": 1,
                                         "next_x_pos_00": pulse_pos_x[i],
                                         "next_y_pos_01": pulse_pos_y[i]},
                       do_move_tip = True, do_set_setpoint = True)
    time.sleep(0.2)

    ####################---------- Apply pulse ----------####################
    V_amp = Vdc_amp[i]
    V_time = Vdc_time[i]
    newexp.define_apply_pulse(pulse_parms_dict = {"pulse_init_amplitude_V_00": 0, "pulse_mid_am
                                                  "pulse_final_amplitude_V_02": 0, "pulse_on_du
                                                  "rise_time_s_05": 1E-4, "pulse_final_duration
                                                  "pulse_repeats_06": 1},
                              do_create_pulse = True, do_upload_pulse = True, do_apply_pulse =
    #
    time.sleep(1)
    newexp.define_apply_pulse(pulse_parms_dict = {"pulse_init_amplitude_V_00": 0, "pulse_mid_am
                                                  "pulse_final_amplitude_V_02": 0, "pulse_on_du
                                                  "rise_time_s_05": 1E-4, "pulse_final_duration
                                                  "pulse_repeats_06": 1},
                              do_create_pulse = True, do_upload_pulse = True, do_apply_pulse =

    time.sleep(2)

    ####################---------- Do BEPFM to image domain ----------####################
    dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {"scan_pixel": 16,
                                                                           "scan_x_start": puls
                                                                           "scan_y_start": puls
                                                                           "scan_x_stop": pulse
                                                                           "scan_y_stop": pulse
                                                          file_name = "Domain_Writing_{}".format(i)
                                                          progress_on = False, ploton = False)
    time.sleep(0.5)

    # Plot BEPFM images
    f, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6, figsize = (30, 5), dpi = 100)
    ax1.imshow(dset_pfm[:,:,0])
    ax2.imshow(dset_pfm[:,:,1])
    ax3.imshow(dset_pfm[:,:,2])
    ax4.imshow(dset_pfm[:,:,3])
    ax5.imshow(dset_chns[0,:,:])
    ax6.imshow(dset_chns[1,:,:])
    plt.show()
```

```
  0%|
```

```
Setpoint is:  1.0
Tip parameters are:  (-0.9, -0.9, 0.5)
```

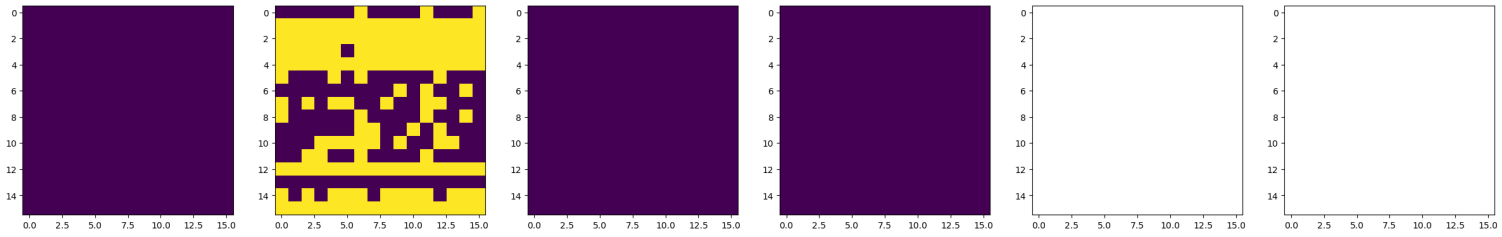
```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
 25%|████████████████████                                                                | 1/4 [
```

```
Setpoint is:  1.0
Tip parameters are:  (0.9, -0.9, 0.5)
```

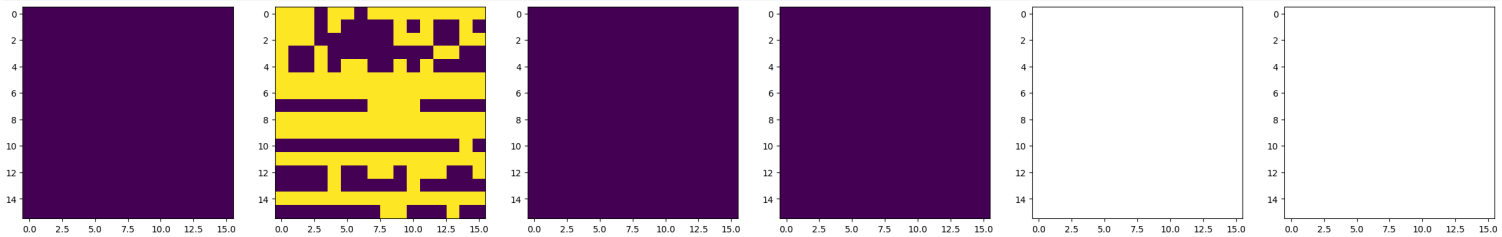
```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
 50%|████████████████████████████████████████                                            | 2/4 [
```

```
Tip parameters are:  (-0.9, 0.9, 0.5)
```

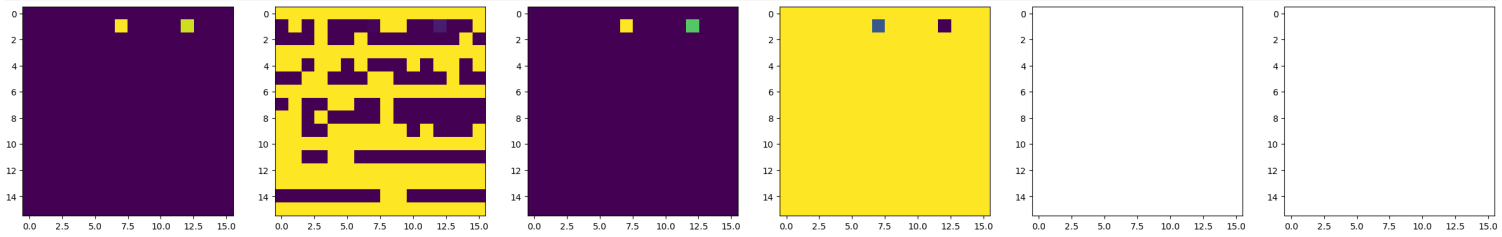
```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
 75%|███████████████████████████████████████████████████████████████████████████                | 3/4 [
```

```
Setpoint is:  1.0
Tip parameters are:  (0.9, 0.9, 0.5)
```

```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```

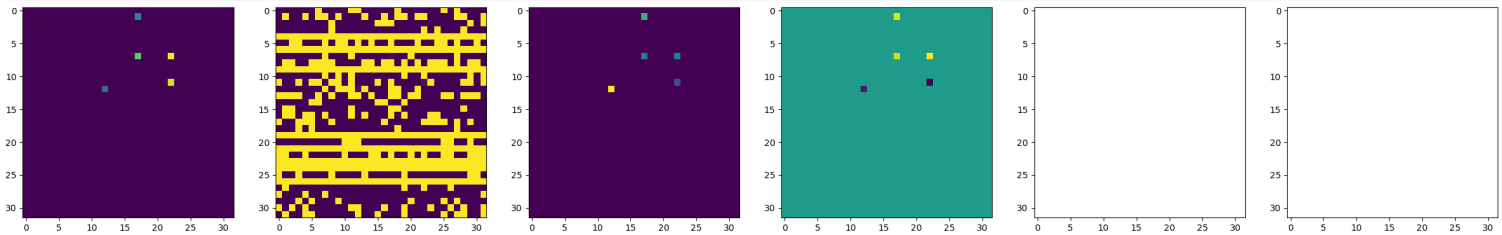


```
100%|███████████████████████████████████████████████████████████████████████████████| 4/4 [
```

Step 4. Do a BEPFM at the whole experiment area

```
dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {"scan_pixel": 32, "scan_
                                                                       "scan_y_start": -1.0,"sc
                                                                       "scan_y_stop": 1.0},
                                                  file_name = "pfm_whole", ploton = False)

f, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6, figsize = (30, 5), dpi = 100)
ax1.imshow(dset_pfm[:,:,0])
ax2.imshow(dset_pfm[:,:,1])
ax3.imshow(dset_pfm[:,:,2])
ax4.imshow(dset_pfm[:,:,3])
ax5.imshow(dset_chns[0,:,:])
ax6.imshow(dset_chns[1,:,:])
plt.show()
```

```
 [progress: 0:00:35] |**********************************  | (ETA:   0:00:01) C:\Users\yla\AppD
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



Experiment 2. Perform measurement with random pulse parameters

Prior to expeirment, set a directory to save data

```
os.chdir(r"C:\Users\yla\Dropbox (ORNL)\My Files\AEcroscopy_BEPyAE")
```

Step 1. Generate a location array

```
# All locations span across [start_point_x, end_point_x] in x-direction and [start_point_y, end_
# There are num_x rows and num_y columns in the locations array

start_point_x = -0.9   # Define location array parameters
end_point_x = 0.9
start_point_y = -0.9
end_point_y = 0.9
num_x = 2
num_y = 2

# Generate location array
pos_x = np.linspace(-0.9, 0.9, num_x)
pos_y = np.linspace(-0.9, 0.9, num_y)
pulse_pos = np.meshgrid(pos_x, pos_y)
pulse_pos_x = pulse_pos[0].reshape(-1)
pulse_pos_y = pulse_pos[1].reshape(-1)  # pulse_pos_x and pulse_pos_y are the coordinates of al

# Set BEPFM image size
img_size = 0.1

# Check
if img_size > np.abs(pos_x[0]-pos_x[1]):
    print ("Alert: there will be image overlap along x-direction")
elif img_size > np.abs(pos_y[0]-pos_y[1]):
    print ("Alert: there will be image overlap along y-direction")
else:
    print("{} locations are ready for experiments".format(len(pulse_pos_x)))
```

```
4 locations are ready for experiments
```

Step 2. Establish pulse space

```
# pulse magnitude space
min_voltage = 5
max_voltage = 9

# pulse time space
min_time_log = -4
max_time_log = 1
```

Step 3. Start experiment

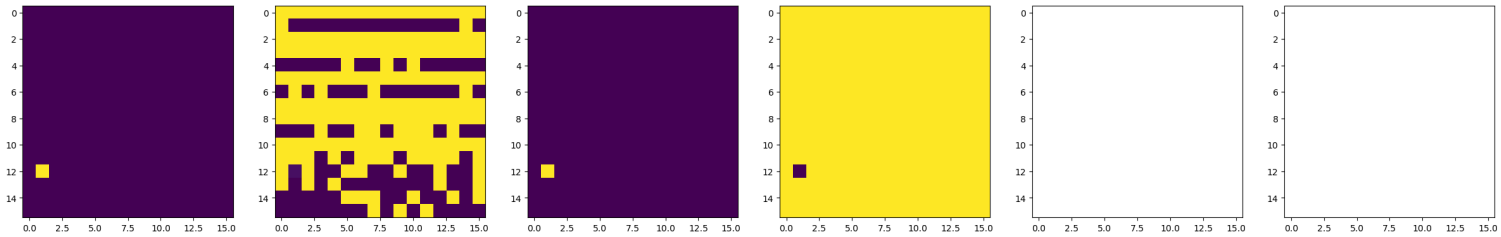
```
# Set measurement iteration number
iterations = 100
# Set lists to save pulse parameters
Vdc_amp = []
Vdc_time = []

# Measurement starts
for i in tqdm(range(iterations)):
    ######################---------- Move tip to the pulse location ----------#################
    newexp.tip_control(tip_parms_dict = {"set_point_V_00": 1,
                                         "next_x_pos_00": pulse_pos_x[i],
                                         "next_y_pos_01": pulse_pos_y[i]},
                       do_move_tip = True, do_set_setpoint = True)
    time.sleep(0.2)

    ######################---------- Apply pulse ----------######################
    # Random pulse parameters
    V_amp = np.random.uniform(min_voltage, max_voltage)
    V_time = np.random.uniform(min_time_log, max_time_log)
    V_time = np.power(10, V_time)
    # Add parameters to Vdc lists
    Vdc_amp.append(V_amp)
    Vdc_time.append(V_time)

    # Apply pulse
    newexp.define_apply_pulse(pulse_parms_dict = {"pulse_init_amplitude_V_00": 0, "pulse_mid_am
                                                  "pulse_final_amplitude_V_02": 0, "pulse_on_du
                                                  "rise_time_s_05": 1E-4, "pulse_final_duration
                                                  "pulse_repeats_06": 1},
                              do_create_pulse = True, do_upload_pulse = True, do_apply_pulse =
    #
    time.sleep(1)
    newexp.define_apply_pulse(pulse_parms_dict = {"pulse_init_amplitude_V_00": 0, "pulse_mid_am
                                                  "pulse_final_amplitude_V_02": 0, "pulse_on_du
                                                  "rise_time_s_05": 1E-4, "pulse_final_duration
                                                  "pulse_repeats_06": 1},
                              do_create_pulse = True, do_upload_pulse = True, do_apply_pulse =

    time.sleep(2)

    ######################---------- Do BEPFM to image domain ----------####################
    dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {"scan_pixel": 16,
                                                                           "scan_x_start": puls
                                                                           "scan_y_start": puls
                                                                           "scan_x_stop": pulse
                                                                           "scan_y_stop": pulse
                                                           file_name = "Domain_Writing_{}".format(i)
                                                           progress_on = False, ploton = False)
    time.sleep(0.5)

    # Plot BEPFM images
    f, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6, figsize = (30, 5), dpi = 100)
    ax1.imshow(dset_pfm[:,:,0])
```

```
    ax4.imshow(dset_pfm[:,:,3])
    ax5.imshow(dset_chns[0,:,:])
    ax6.imshow(dset_chns[1,:,:])
    plt.show()

# Save pulse parameters
np.save("Vdc_list.npy", np.asarray([Vdc_amp, Vdc_time]))
```

```
  0%|
```

```
Setpoint is:  1.0
Tip parameters are:  (-0.9, -0.9, 0.5)
```

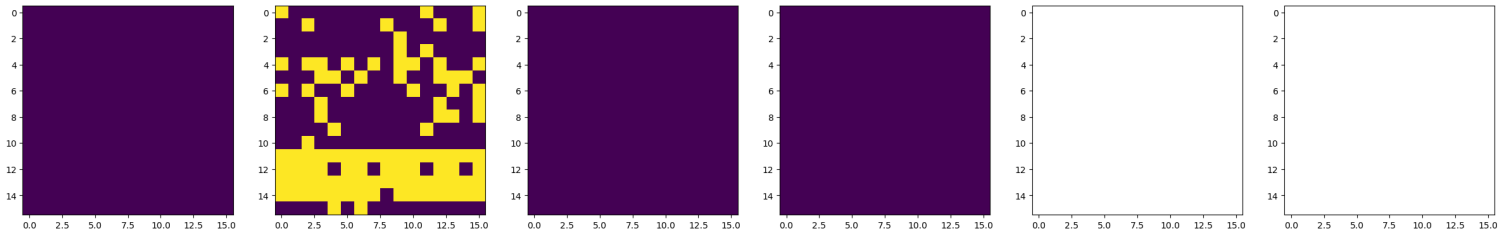
```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
  1%|▉                                                                                | 1/100 [0
```

```
Setpoint is:  1.0
Tip parameters are:  (0.9, -0.9, 0.5)
```

```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```

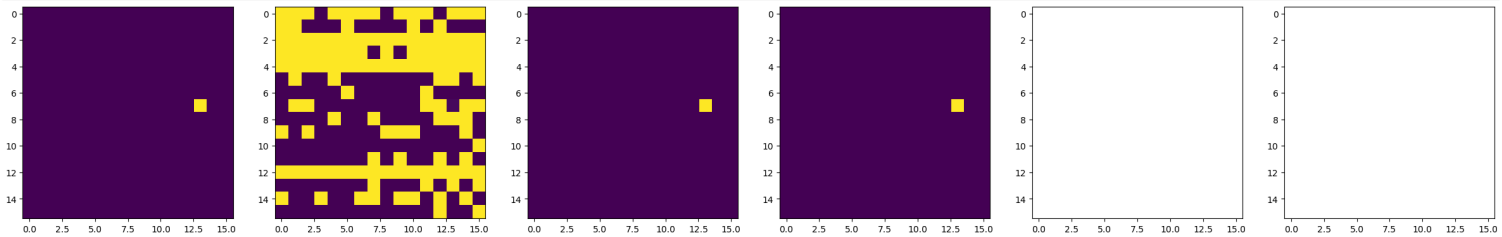


```
  2%|█▏                                                                               | 2/100 [01
```

Skip to main content

```
Tip parameters are:  (-0.9, 0.9, 0.5)
```

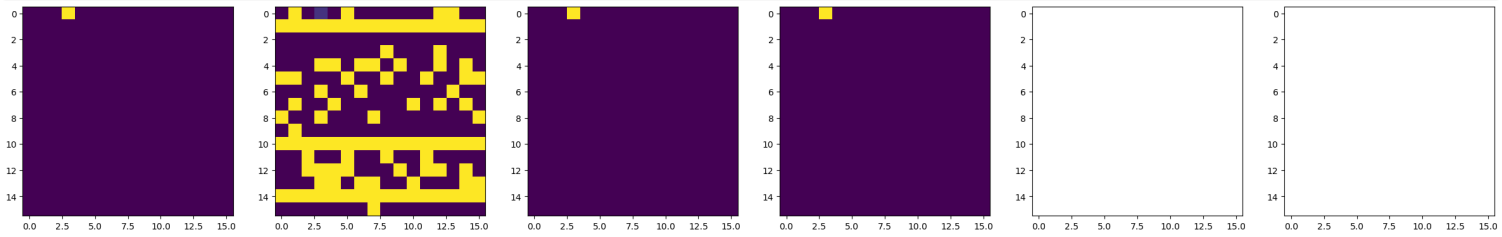
```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
  3%|███                                                                  | 3/100
```

```
Setpoint is:  1.0
Tip parameters are:  (0.9, 0.9, 0.5)
```

```
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
C:\Users\yla\AppData\Local\anaconda3\lib\site-packages\pyNSID\io\hdf_utils.py:376: FutureWarning
  warn('validate_h5_dimension may be removed in a future version',
```



```
  4%|███▍                                                                 | 4/100
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[19], line 11
      7 # Measurement starts
      8 for i in tqdm(range(iterations)):
```

```
---> 11                                                 "next_x_pos_00": pulse_pos_x[i],
     12                                                 "next_y_pos_01": pulse_pos_y[i]},
     13                           do_move_tip = True, do_set_setpoint = True)
     14     time.sleep(0.2)
     16     ######################---------- Apply pulse ----------######################
     17     # Random pulse parameters

IndexError: index 4 is out of bounds for axis 0 with size 4
```

Step 4. Do a BEPFM at the whole experiment area

```
dset_pfm, dset_chns, dset_cs = newexp.raster_scan(raster_parms_dict = {"scan_pixel": 32, "scan_
                                                                       "scan_y_start": -1.0,"sc
                                                                       "scan_y_stop": 1.0},
                                                  file_name = "pfm_whole", ploton = False)

f, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6, figsize = (30, 5), dpi = 100)
ax1.imshow(dset_pfm[:,:,0])
ax2.imshow(dset_pfm[:,:,1])
ax3.imshow(dset_pfm[:,:,2])
ax4.imshow(dset_pfm[:,:,3])
ax5.imshow(dset_chns[0,:,:])
ax6.imshow(dset_chns[1,:,:])
plt.show()
```

Notebook for high throughput domain writing analysis

YongtaoLiu

*June*15, 2023

```
# You may need to sidpy if you did not do before
# !pip install sidpy
```

Import

Skip to main content

```
import numpy as np
import matplotlib.pyplot as plt
import os
import h5py
import sidpy
import cv2
import imutils
```

Set the directory and root name of your dataset

```
# directory
path = r"C:\Users\yla\Dropbox (ORNL)\My Files\AEcroscopy_BEPyAE\AEcroscopy_data\Experiment1"

# name
file_name = "Domain_Writing_"

# spot numbers
num_x = 8
num_y = 8
```

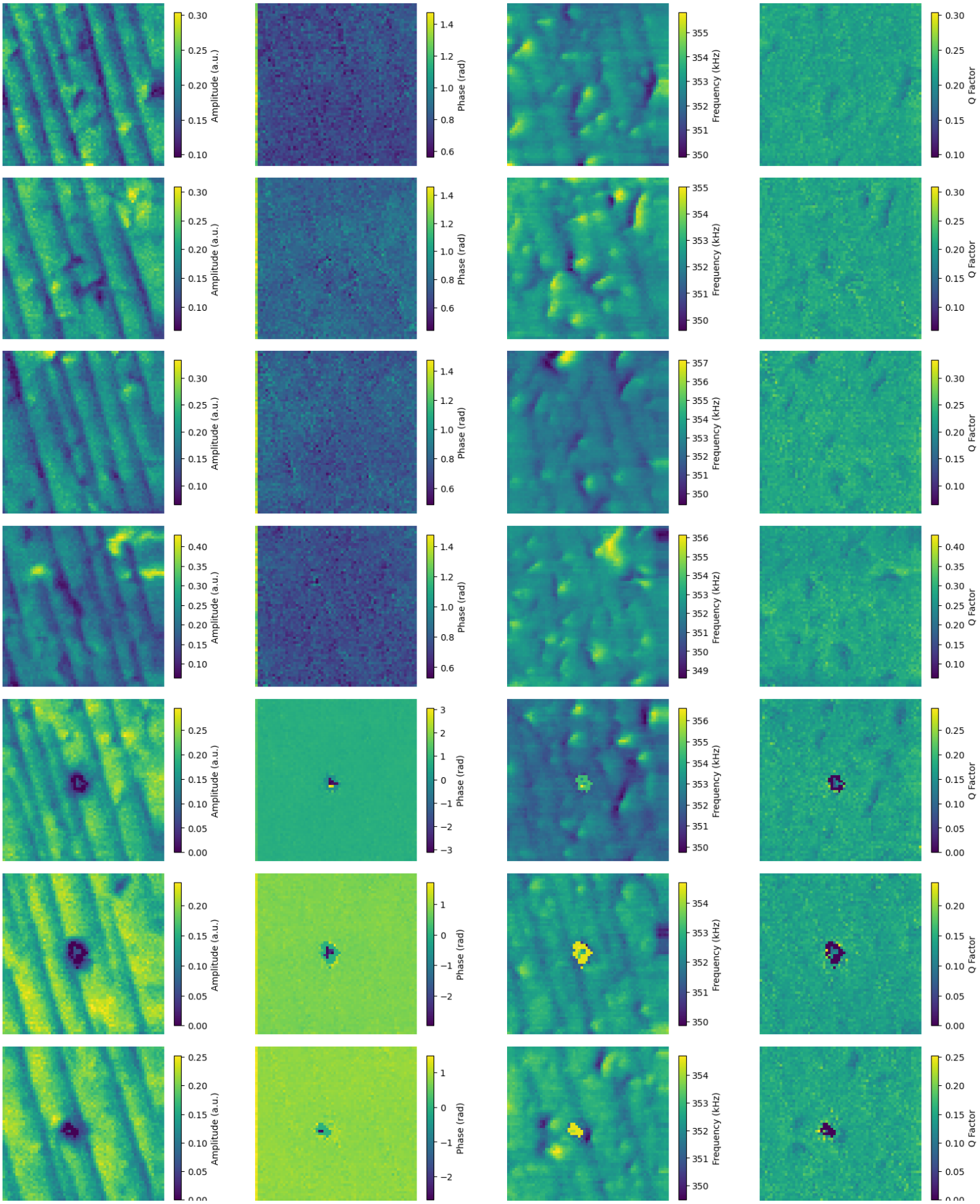
Load all data

```
count_imgs = 64 # variable of how many images you have in the directory
pixel = 64  # pixel of your image

# change working directory
os.chdir(path)

# create arrays for all data
amplitude = np.zeros((count_imgs, pixel, pixel))
phase = np.zeros((count_imgs, pixel, pixel))
frequency = np.zeros((count_imgs, pixel, pixel))
qfactor = np.zeros((count_imgs, pixel, pixel))
topography = np.zeros((count_imgs, pixel, pixel))

for i in range (count_imgs):
    h5 = h5py.File('Domain_Writing_{}_0.hf5'.format(i), 'r+')
    be_qf = h5["BE Quick Fitting/Quick Fitting/Quick Fitting"]
    be_ch = h5["BE Channels/Channels/Channels"]
    amplitude[i,] = be_qf[:,:,0]
    phase[i,] = be_qf[:,:,3]
    frequency[i,] = be_qf[:,:,1]
    qfactor[i,] = be_qf[:,:,2]
    topography[i,] = be_ch[0,:,:,0]

nor_amplitude = (amplitude-amplitude.min())/amplitude.ptp()
nor_phase = (phase - phase.min())/phase.ptp()
nor_frequency = (frequency - frequency.min())/frequency.ptp()
nor_topography = (topography - topography.min())/topography.ptp()
```

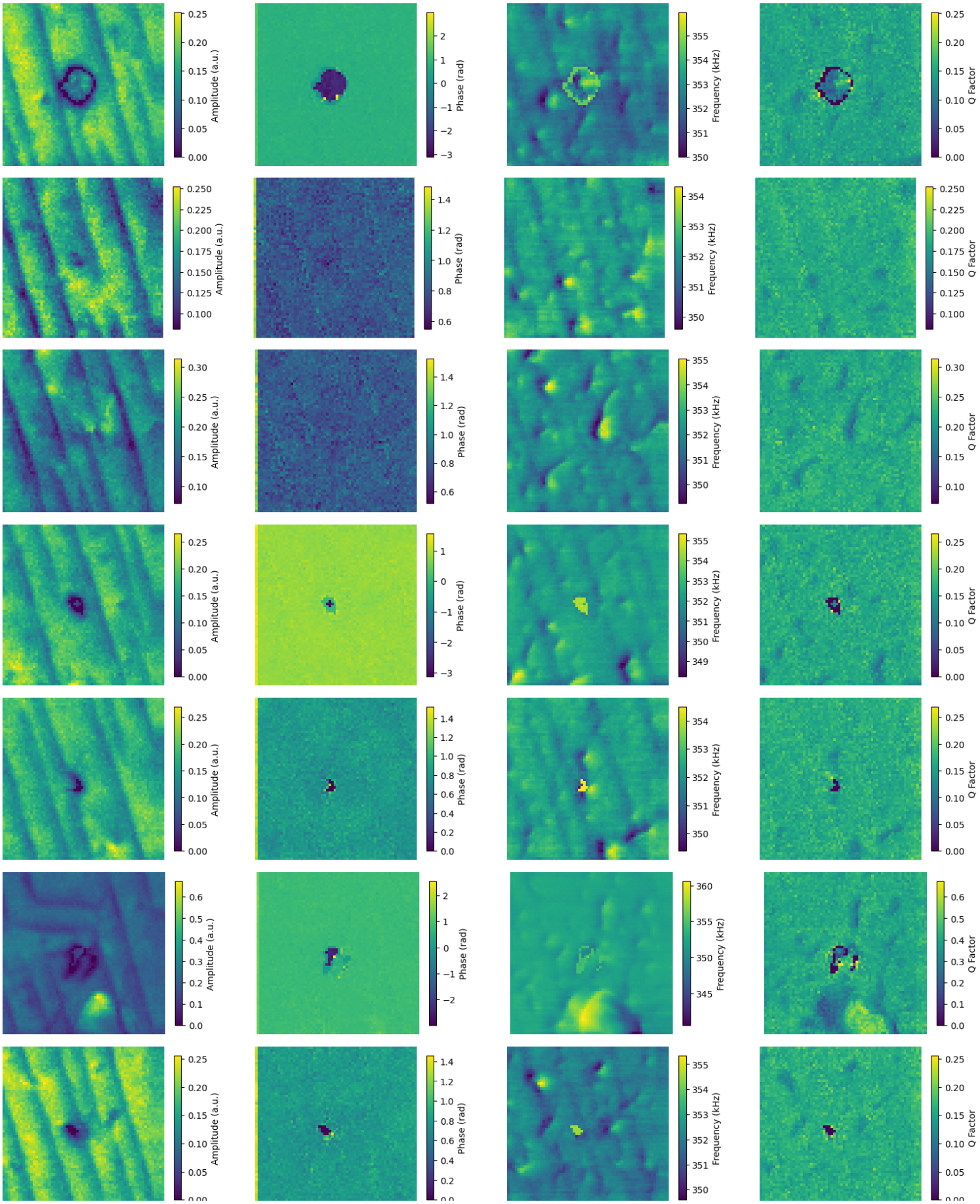
Plot data

Plot all results

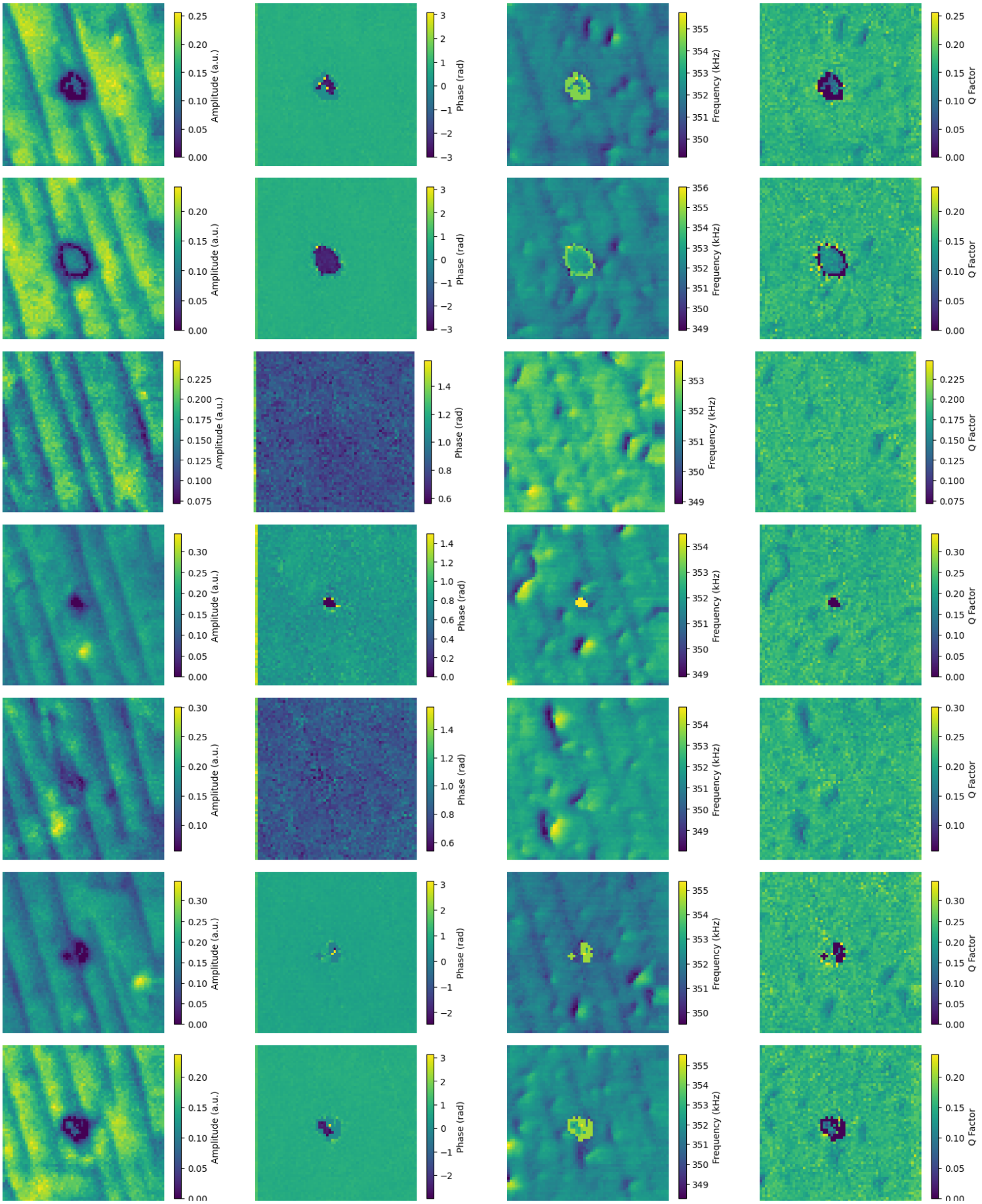
```
for i in range (count_imgs):

    fig, axs = plt.subplots(1, 4, figsize=(16, 4), dpi = 100)
    fig.subplots_adjust(left=0.02, bottom=0.06, right=0.95, top=0.99, wspace=0.25)
    cm = 'viridis'
    shrink = 0.6

    im0 = axs[0].imshow(amplitude[i, ]*1000, origin = "lower", interpolation='nearest', cmap=cm
    fig.colorbar(im0, ax=axs[0], shrink = shrink, label = "Amplitude (a.u.)")
    axs[0].axis("off")

    im1 = axs[1].imshow(phase[i, ], origin = "lower", interpolation='nearest', cmap=cm)
    fig.colorbar(im1, ax=axs[1], shrink = shrink, label = "Phase (rad)")
    axs[1].axis("off")

    im2 = axs[2].imshow(frequency[i, ]/1000, origin = "lower", interpolation='nearest', cmap=cm
    fig.colorbar(im2, ax=axs[2], shrink = shrink, label = "Frequency (kHz)")
    axs[2].axis("off")

    im3 = axs[3].imshow(qfactor[i, ], origin = "lower", vmin = 0, vmax = 250, interpolation='ne
    fig.colorbar(im0, ax=axs[3], shrink = shrink, label = "Q Factor")
    axs[3].axis("off")

    plt.show()
```



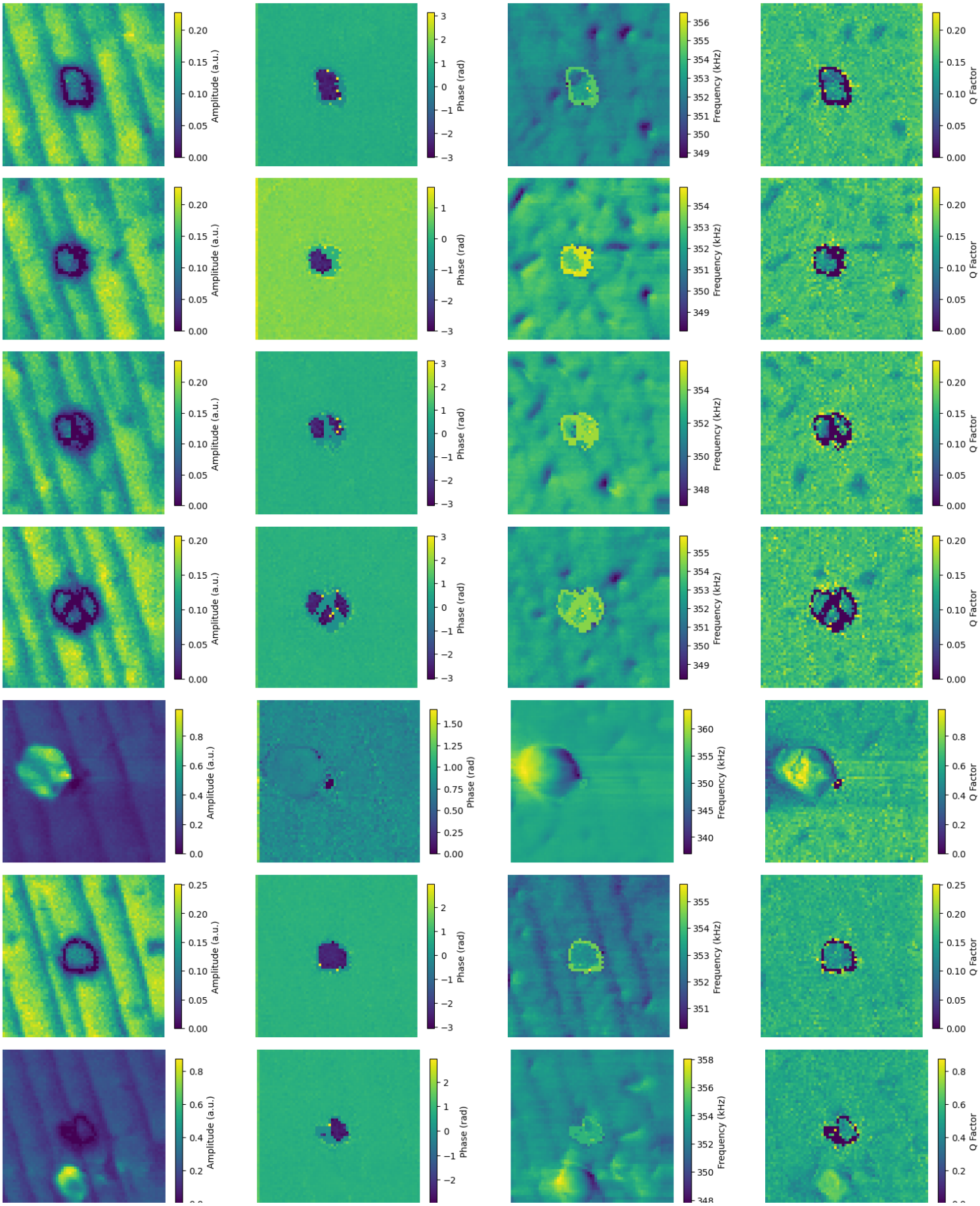
Skip to main content



Skip to main content



Skip to main content

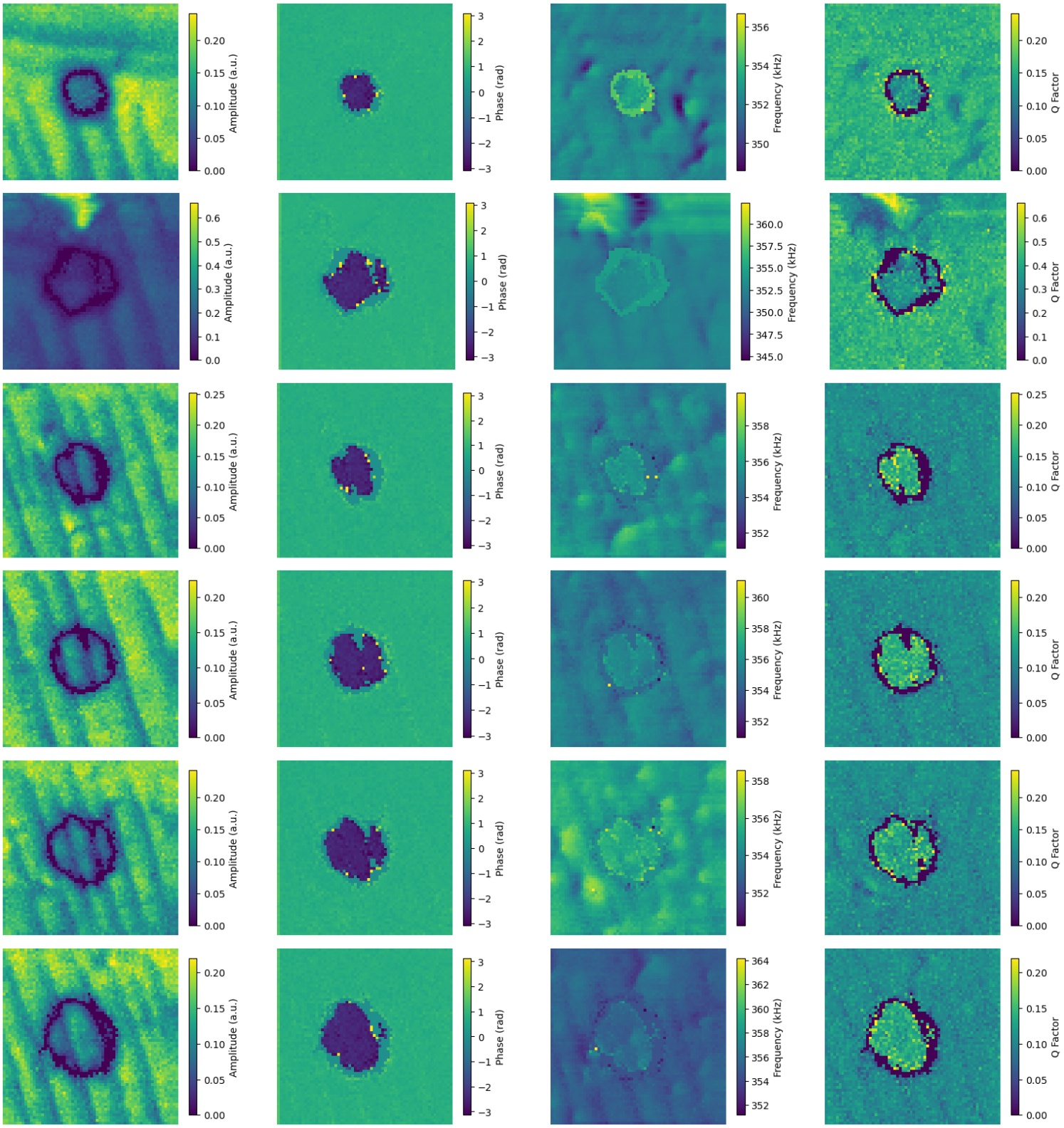


Skip to main content

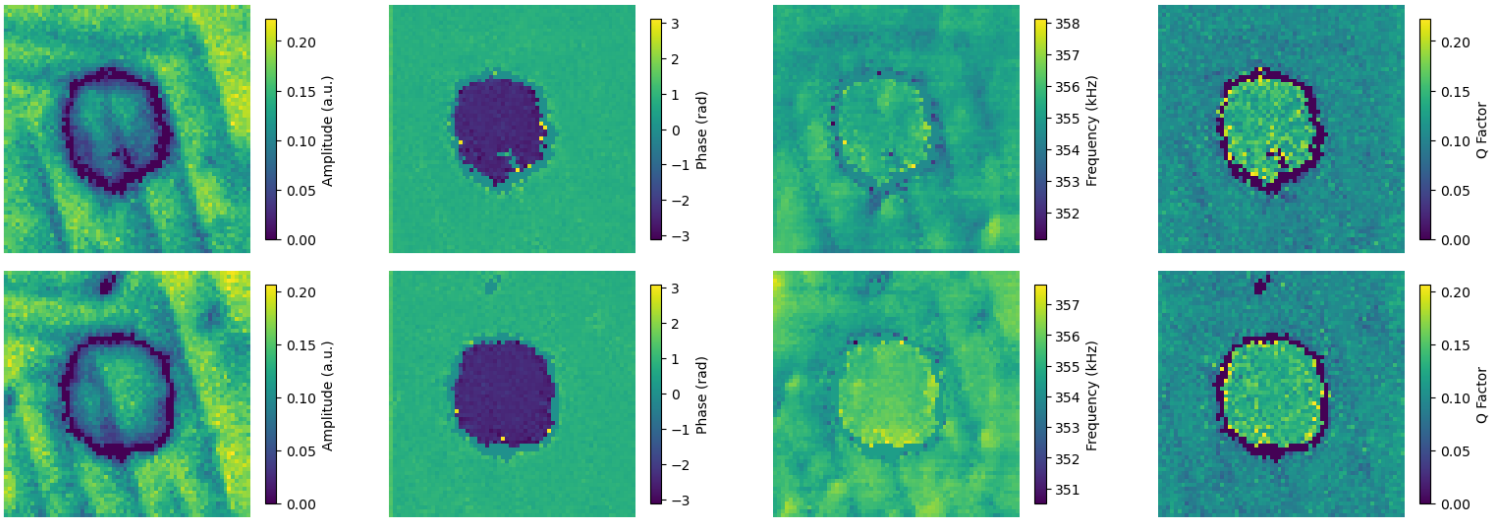
Skip to main content

Skip to main content

Skip to main content



Skip to main content



Plot a specific channel together

```
fig, axes = plt.subplots(num_y, num_x, figsize=(num_x, num_y),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.02, wspace=0.02))

for ax, i in zip(axes.flat, range(count_imgs)):
    ax.imshow(amplitude[i,], origin = "lower") # We are plotting amplitude now, you can change
```