cBPM Position Resolution Measurement in 2023

2025 Accelerator and Beam Line Field Training

Jul 8 2025, Korea University Sejong Campus

Soohyung Lee, Ph. D (Korea University)

2023 Experiment Configurations

Data Files

- Data files are written as Python pickle files in a format of Pandas dataframe
  - With Pandas in Python, they can be easily readable

- *calibration_horizontal_2023.pickle*: Horizontal calibration data
  - 7 horizontal beam positions, 50 measurements at each position
- *calibration_vertical_2023.pickle*: Vertical calibration data
  - 7 vertical beam positions, 50 measurements at each position
- *resolution_2023.pickle*: Resolution run data
  - 1200 measurements at a beam position

Data Structure

- Columns:
  - *measure_no*: Measurement number as an index
  - *time*: List of time in a measurement [unit: s]
  - *bpm_ax*: Voltage measured at BPM-A(X) (list of voltage values over time) [unit: V]
    - *bpm_ay*, *bpm_bx*, *bpm_by*, *bpm_cx*, *bpm_cy* are other BPMs and axes, accordingly
  - *beam_charge*: Beam charge normalized by 1.6 nC (=10^{10} electrons)
  - *stripline_x*: Beam position measured by a stripline BPM (X) [unit: μm]
    - *stripline_y* is for a stripline BPM (Y)
  - *steering_current*: Steering magnet current [unit: A]

calibration_horizontal_2023

measure_no	time	bpm_ax	bpm_ay	bpm_bx	bpm_by	bpm_cx	bpm_cy	beam_charge	stripline_x	stripline_y	steering_current
1	[-2.400e-07 -2.	[0.0415039 0.03	[0.0317383 0.0 0.0512695]	[0.0145508 0.0243	[0.0366211 0.0	[0.00244141 - 0.012207]	[0.012207 0.0	0.548159122467041	-3833.99658203125	-13.020044326782200	-1.4
2	[-2.400e-07 -2.	[0.012207 0.0 0.0317383]	[0.0512695 0.03	[0.0194336 0.009	[0.0219727 0.0	[-0.00732422 0.00244141]	[-0.00244141 -0.00244141]	0.5323860049247740	-4011.67822265625	-37.040924072265600	-1.4
3	[-2.400e-07 -2.	[0.0317383 0.0 0.00732422]	[0.0268555 0.06	[0.0291992 0.0438	[0.00732422 0.	[0.012207 0.0	[0.00244141	0.5437777042388920	-3961.016845703130	-27.711624145507800	-1.4
4	[-2.400e-07 -2.	[0.012207 0.0 0.00732422]	[0.0512695 0.03	[4.78516e-03 2.4 4.78516e-03]	[0.0415039 0.0	[0.00244141 - 0.0268555]	[0.0219727 0	0.4727987349033360	-4051.92822265625	-64.27881622314450	-1.4
5	[-2.400e-07 -2.	[0.0366211 0.02	[0.0415039 0.0	[0.0389648 0.0243	[0.0219727 0.0	[-0.00732422 -0.012207]	[0.012207 0.	0.5297571420669560	-3938.2294921875	-72.16757202148440	-1.4

Reading the data files with python

```
python
(lbpm)  shlee  ~/test/acceltrain/data/2023  python
Python 3.13.0 | packaged by Anaconda, Inc. | (main, Oct  7 2024, 16:25:56) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> data = pd.read_pickle("calibration_horizontal_2023.pickle")
>>> print(data)
            measure_no                                               time  ...  stripline_y steering_current
measure_no                                                                 ...
1                    1  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -13.020044             -1.4
2                    2  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -37.040924             -1.4
3                    3  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -27.711624             -1.4
4                    4  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -64.278816             -1.4
5                    5  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -72.167572             -1.4
...                ...                                                ...  ...          ...              ...
346                346  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -53.627110             -2.6
347                347  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -30.739288             -2.6
348                348  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -12.208589             -2.6
349                349  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -70.737633             -2.6
350                350  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...  ...   -86.948952             -2.6

[350 rows x 12 columns]
>>>
```

Navigating Data

- Check data columns

```
>>> data.columns
Index(['measure_no', 'time', 'bpm_ax', 'bpm_ay', 'bpm_bx', 'bpm_by', 'bpm_cx',
       'bpm_cy', 'beam_charge', 'stripline_x', 'stripline_y',
       'steering_current'],
      dtype='object')
>>>
```

- Obtaining column values

```
>>> data["time"]
0      [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
1      [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
2      [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
3      [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
4      [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
                              ...
345    [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
346    [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
347    [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
348    [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
349    [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
Name: time, Length: 350, dtype: object
>>>
```

- Obtaining row values

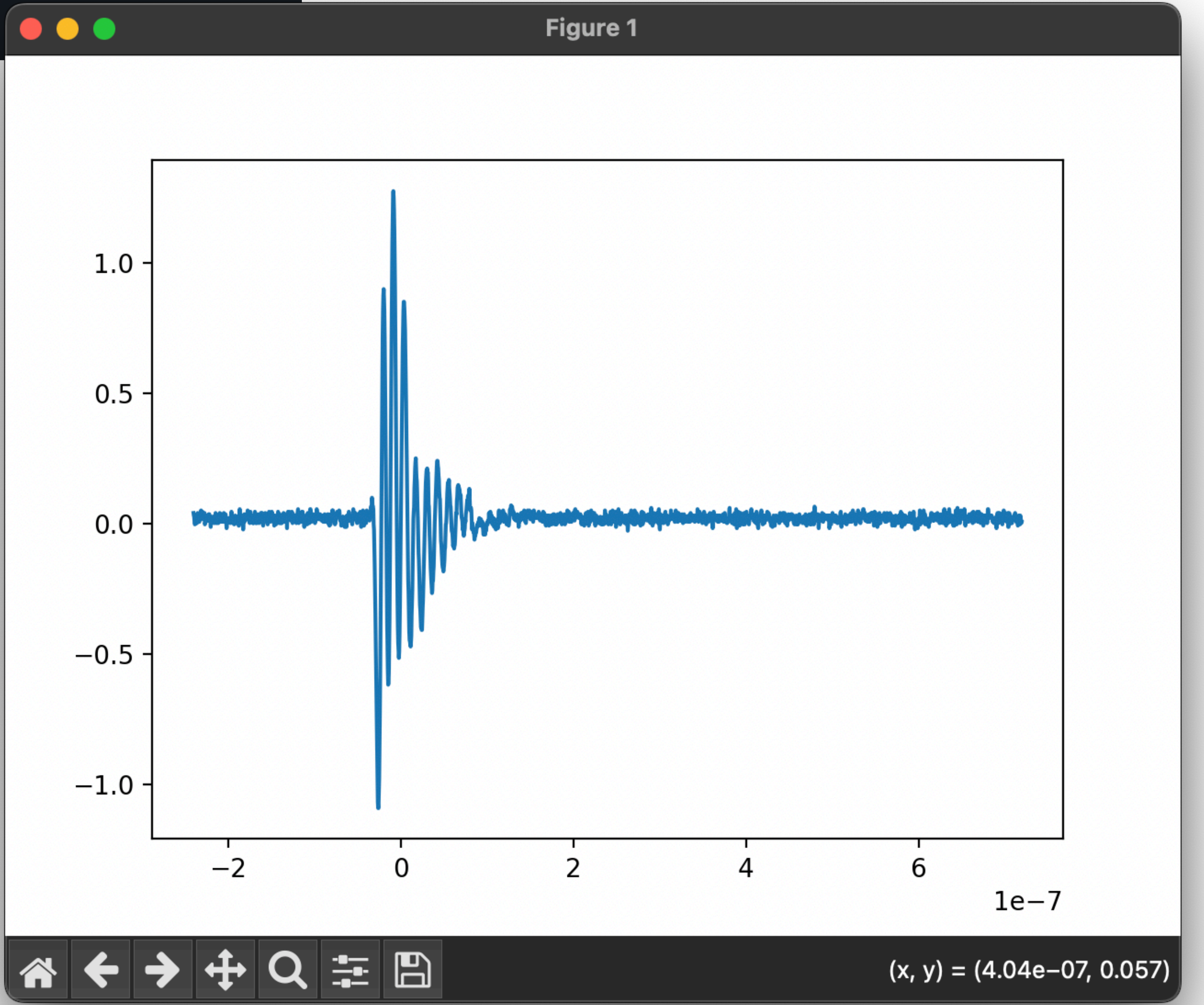
```
>>> data.iloc[0]
measure_no                                                          1
time                [-2.400e-07 -2.396e-07 -2.392e-07  ...   7.188e-...
bpm_ax              [0.0415039  0.0317383  0.0268555   ... 0.017089...
bpm_ay              [ 0.0317383   0.0268555  -0.00244141 ...   0.06...
bpm_bx              [0.0145508 0.0243164 0.0487305  ... 0.0194336 0...
bpm_by              [0.0366211 0.0463867 0.0610352  ... 0.0415039 0...
bpm_cx              [ 0.00244141 -0.0219727   0.00244141 ...   0.02...
bpm_cy              [0.012207  0.0219727 0.0268555  ... 0.012207   0...
beam_charge                                                  0.548159
stripline_x                                              -3833.996582
stripline_y                                                -13.020044
steering_current                                                 -1.4
Name: 0, dtype: object
>>>
```

Plotting Data

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(data.iloc[0]["time"], data.iloc[0]["bpm_ax"])
[<matplotlib.lines.Line2D object at 0x118ae2710>]
>>> plt.show()
```

- This example plots a waveform of a signal from BPM-A (X) over time

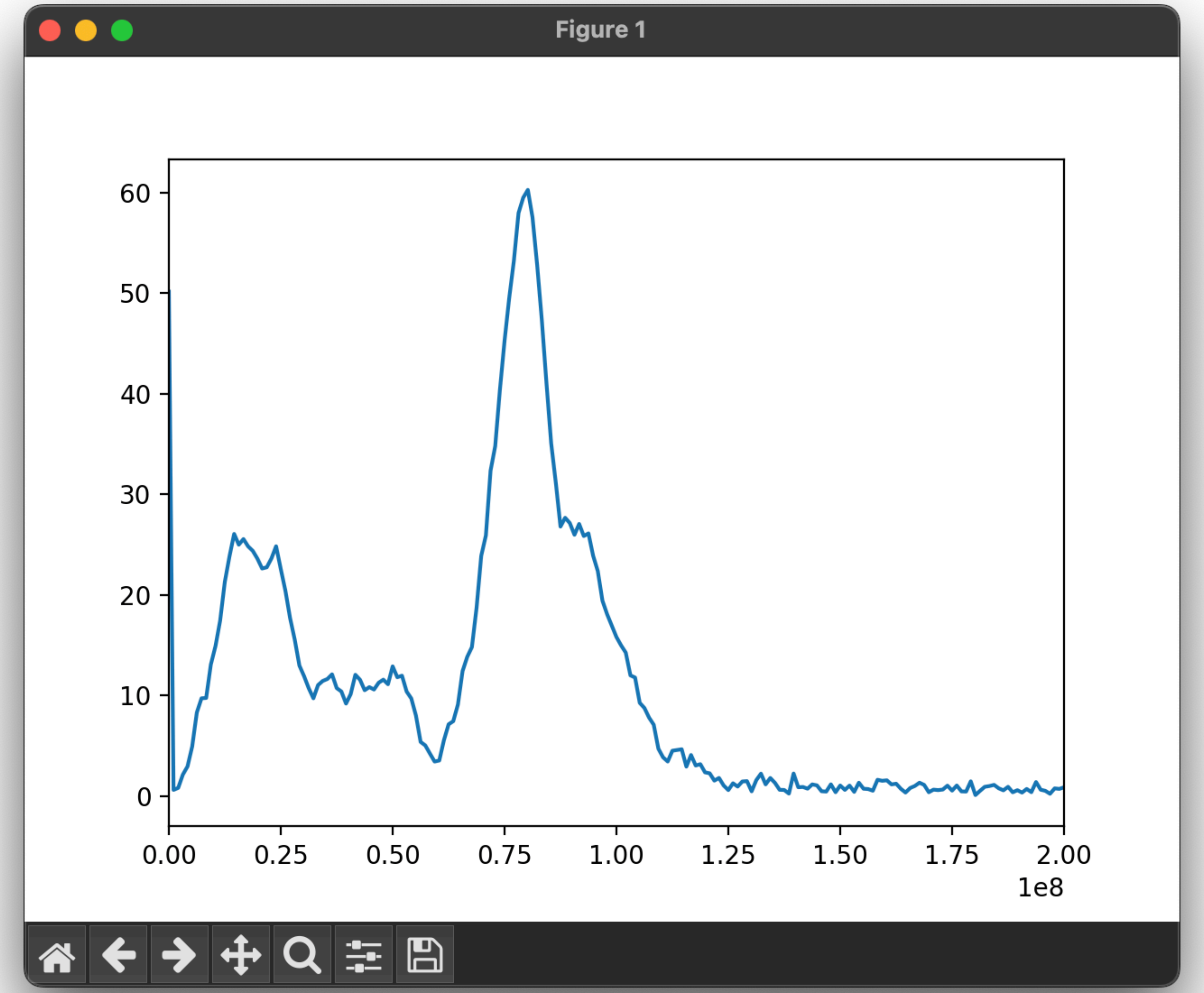
- Plot the waveforms of:
  - BPM-A(Y)
  - BPM-B(X) and BPM-B(Y)
  - BPM-C(X) and BPM-C(Y)



Fast Fourier Transform

- The previous plot (waveform) is in time-domain
  - Fast Fourier transform (FFT) translates the data from time-domain to frequency-domain
- Perform FFT to others: BPM-A(Y), BPM-B(X), BPM-B(Y), BPM-C(X), BPM-C(Y)
  - Check if there are signals at appropriate frequency
  - If not, which frequencies are they at?

```
>>> import numpy as np
>>> from scipy.fft import fft, fftfreq
>>>
>>> time = data.iloc[0]["time"]
>>> bpm_ax = data.iloc[0]["bpm_ax"]
>>>
>>> N = len(time)
>>> dt = time[1] - time[0]
>>>
>>> freq = fftfreq(N, dt)[:N // 2]
>>> fft_result = np.abs(fft(bpm_ax)[:N // 2])
>>>
>>> plt.plot(freq, fft_result)
[<matplotlib.lines.Line2D object at 0x1243bf890>]
>>> plt.xlim(0, 200e6)
(0.0, 200000000.0)
>>> plt.show()
```



Collecting FFT Amplitudes

- First, find the index where the signal is
  - 80 MHz is found at 77th index
  - Find other frequency indices if you found in the previous page

```
>>> freq_index = np.absolute(freq - 80e6).argmin()
>>> freq_index
np.int64(77)
```

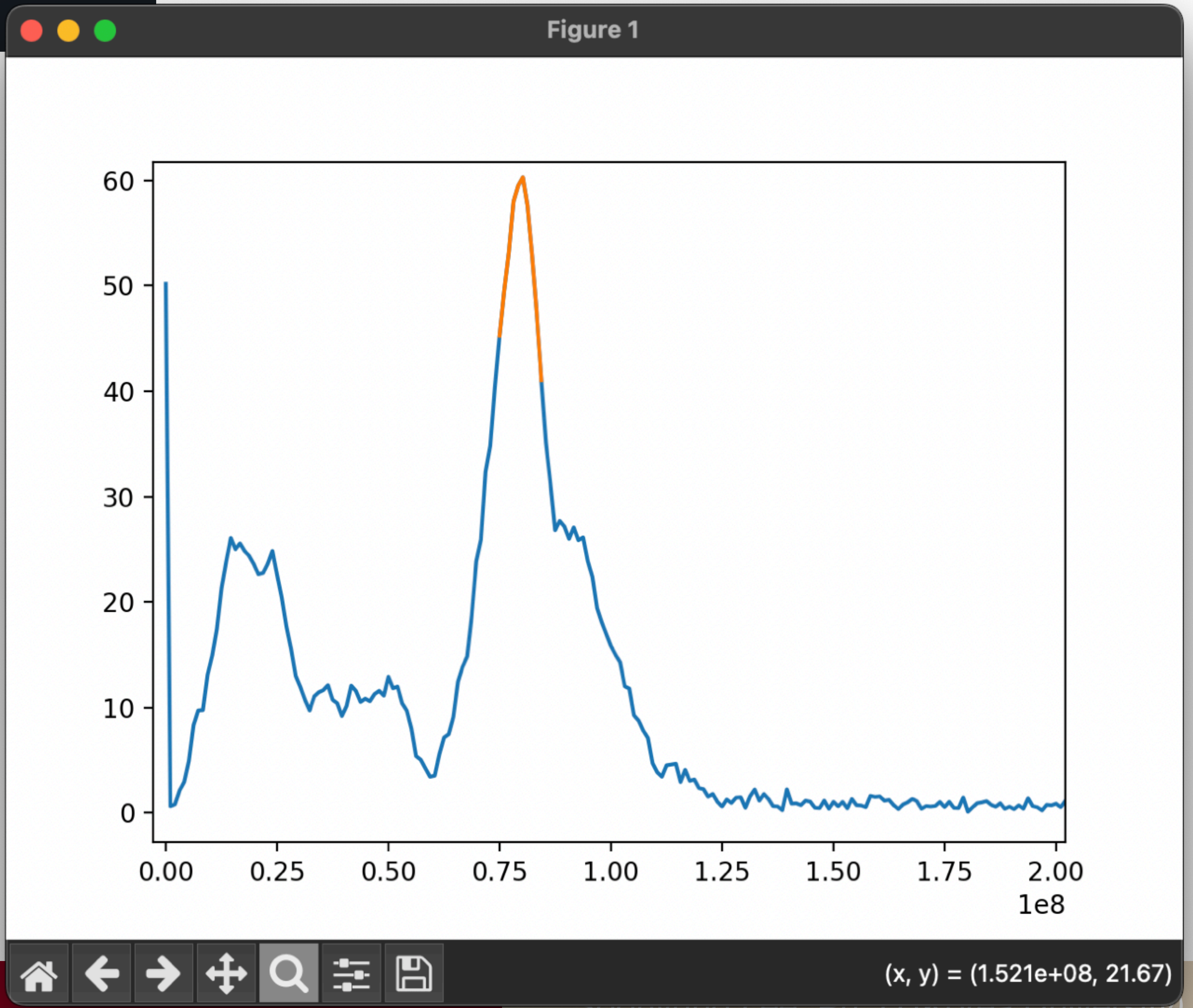
- Then, find the FFT value there

```
>>> freq[freq_index]
np.float64(80208333.33333792)
>>> fft_result[freq_index]
np.float64(60.27762775755604)
>>>
```

- Better way is integration within a range (e.g. ±5 frequency bins)

```
>>> np.sum(fft_result[freq_index - 5:freq_index + 5])
np.float64(524.4328027159542)
```

- Collect the integrated FFT values over all measurements and plot them
- Repeat the same thing for other channels



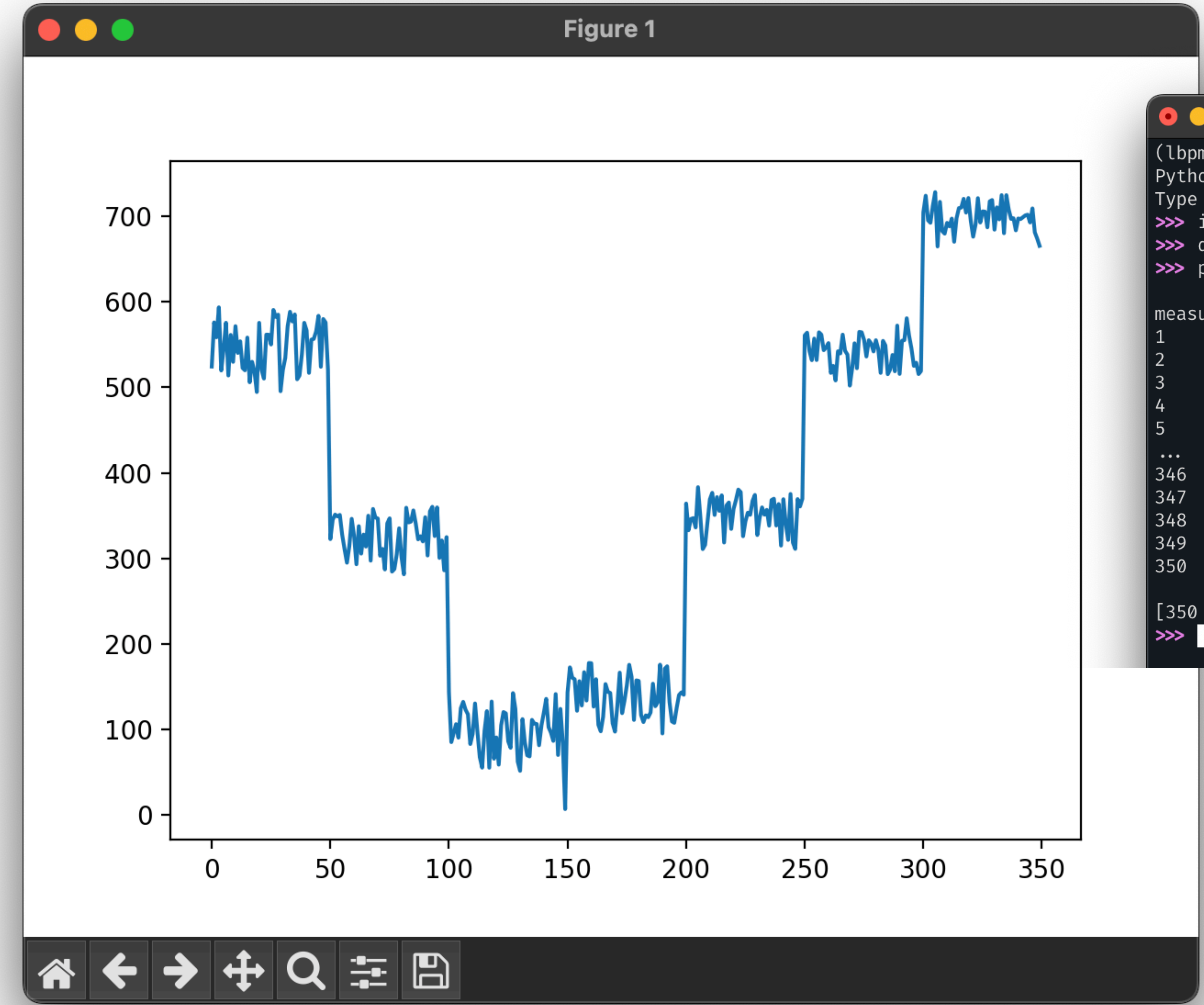
Promoting to a function

- We'll do FFT many times from now on, let's promote it as a function
  - Open Notepad and write the following (and save in the same folder)

```
import pandas as pd
import numpy as np
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt

def do_fft(time, signal):
    N = len(time)
    dt = time[1] - time[0]

    freq = fftfreq(N, dt)[:N // 2]
    fft_result = np.abs(fft(signal)[:N // 2])

    return freq, fft_result

data = pd.read_pickle("calibration_horizontal_2023.pickle")

list_fft_ax = []
list_fft_ay = []
list_fft_bx = []
list_fft_by = []
list_fft_cx = []
list_fft_cy = []

for i in range(len(data)):
    time = data.iloc[i]["time"]
    freq, fft_ax = do_fft(time, data.iloc[i]["bpm_ax"])
    _, fft_ay = do_fft(time, data.iloc[i]["bpm_ax"])
    _, fft_bx = do_fft(time, data.iloc[i]["bpm_bx"])
    _, fft_by = do_fft(time, data.iloc[i]["bpm_by"])
    _, fft_cx = do_fft(time, data.iloc[i]["bpm_cx"])
    _, fft_cy = do_fft(time, data.iloc[i]["bpm_cy"])

    freq_index = np.absolute(freq - 80.e6).argmin()

    list_fft_ax.append(np.sum(fft_ax[freq_index - 5:freq_index + 5]))
    list_fft_ay.append(np.sum(fft_ay[freq_index - 5:freq_index + 5]))
    list_fft_bx.append(np.sum(fft_bx[freq_index - 5:freq_index + 5]))
    list_fft_by.append(np.sum(fft_by[freq_index - 5:freq_index + 5]))
    list_fft_cx.append(np.sum(fft_cx[freq_index - 5:freq_index + 5]))
    list_fft_cy.append(np.sum(fft_cy[freq_index - 5:freq_index + 5]))

plt.plot(list_fft_ax)
plt.show()
```

FFT values over measurements

- For BPM-A(X) case:




```
(lbpm)  shlee  ~/test/acceltrain/data/2023  python
Python 3.13.0 | packaged by Anaconda, Inc. | (main, Oct  7 2024, 16:25:56) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> data = pd.read_pickle("calibration_horizontal_2023.pickle")
>>> print(data)
                 measure_no                                                            time  ... stripline_y steering_current
measure_no                                                                                   ...
1                         1  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -13.020044             -1.4
2                         2  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -37.040924             -1.4
3                         3  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -27.711624             -1.4
4                         4  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -64.278816             -1.4
5                         5  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -72.167572             -1.4
...                     ...                                                ...  ...         ...              ...
346                     346  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -53.627110             -2.6
347                     347  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -30.739288             -2.6
348                     348  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -12.208589             -2.6
349                     349  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -70.737633             -2.6
350                     350  [-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07,...  ...  -86.948952             -2.6

[350 rows x 12 columns]
>>>
```

Adding FFT values to DataFrame

- It is more convenient to add the FFT values we found to the DataFrame

```python
data["fft_ax"] = list_fft_ax
data["fft_ay"] = list_fft_ay
data["fft_bx"] = list_fft_bx
data["fft_by"] = list_fft_by
data["fft_cx"] = list_fft_cx
data["fft_cy"] = list_fft_cy
```

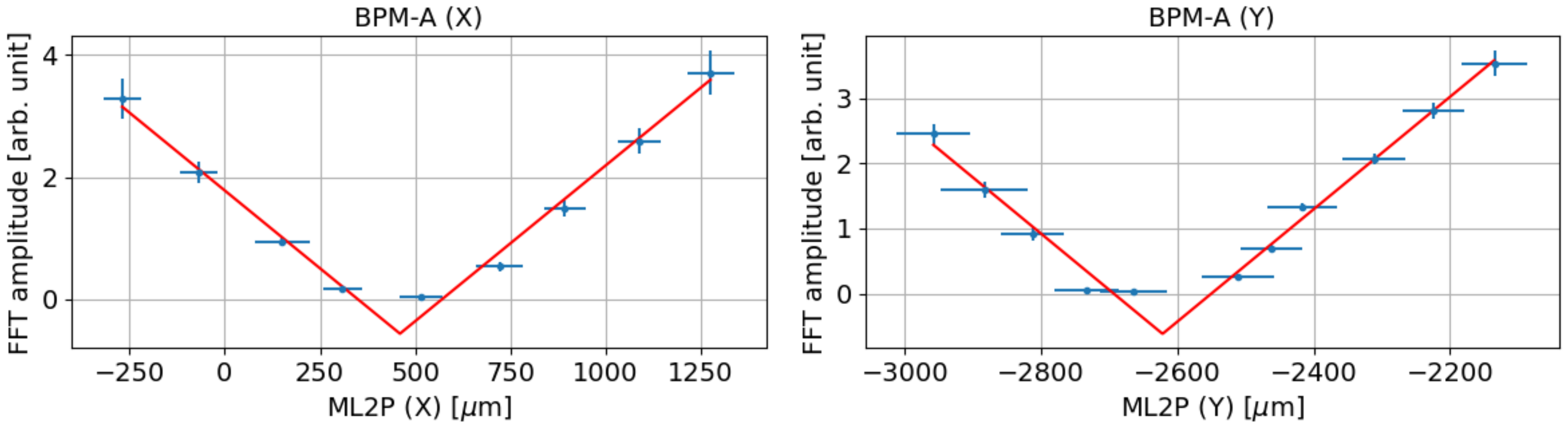
	measure_no	time	fft_ax	fft_ay	steering_current
measure_no					
1	1	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	524.432803	524.432803	-1.4
2	2	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	576.063376	576.063376	-1.4
3	3	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	558.639596	558.639596	-1.4
4	4	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	593.691518	593.691518	-1.4
5	5	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	519.816906	519.816906	-1.4
...	...	...	...	...	...
346	346	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	693.007342	693.007342	-2.6
347	347	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	709.414904	709.414904	-2.6
348	348	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	681.429921	681.429921	-2.6
349	349	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	674.188074	674.188074	-2.6
350	350	[-2.4e-07, -2.396e-07, -2.392e-07, -2.388e-07, ...	665.651316	665.651316	-2.6

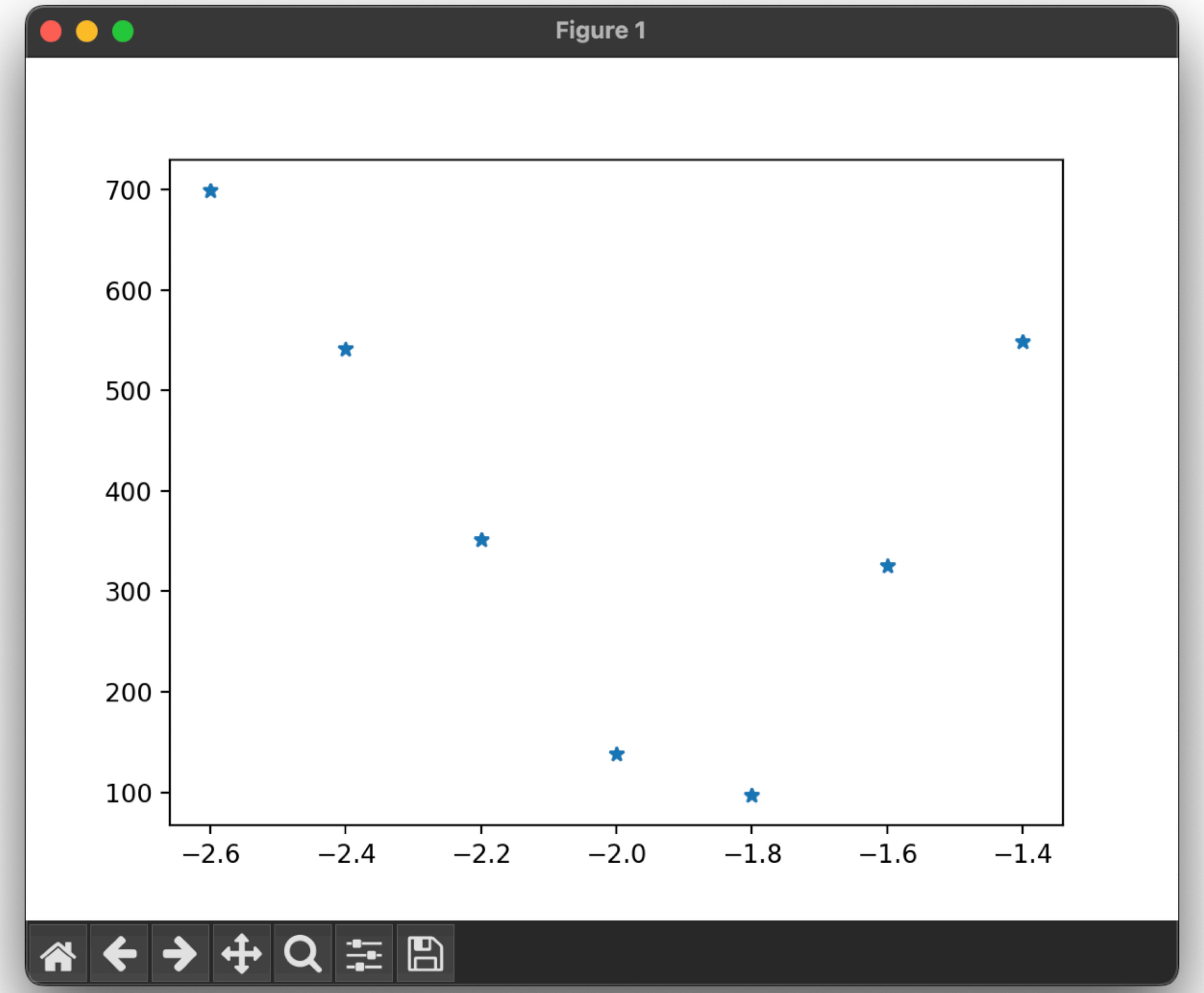
Group by steering currents

- We want to average the FFT values for each beam positions

```
steering = data.groupby("steering_current").groups.keys()
mean_ax = data.groupby("steering_current")["fft_ax"].mean()
std_ax = data.groupby("steering_current")["fft_ax"].std()
```

- Now we are ready to plot something like:



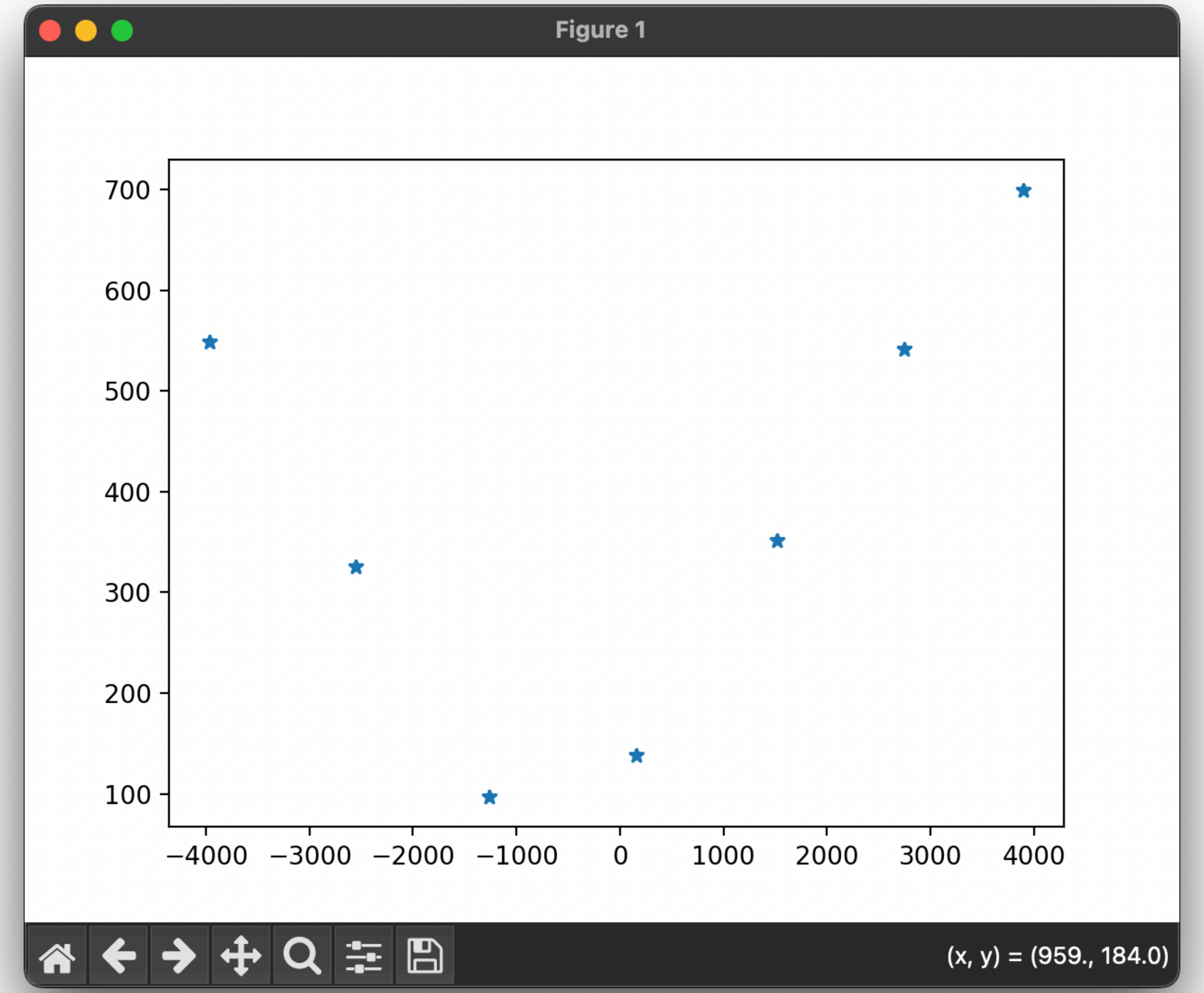



```
plt.plot(steering, mean_ax, "*")
plt.show()
```

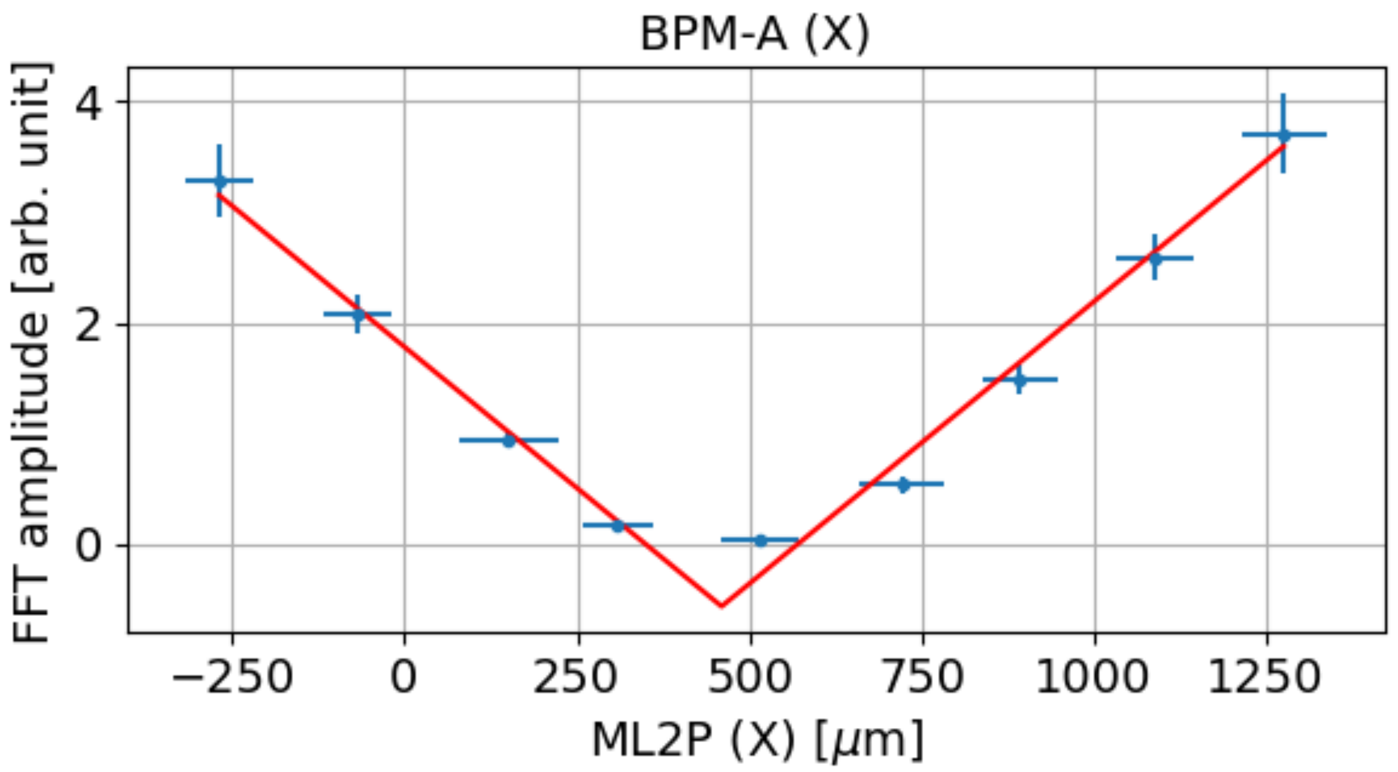
- Plot the same thing for other channels

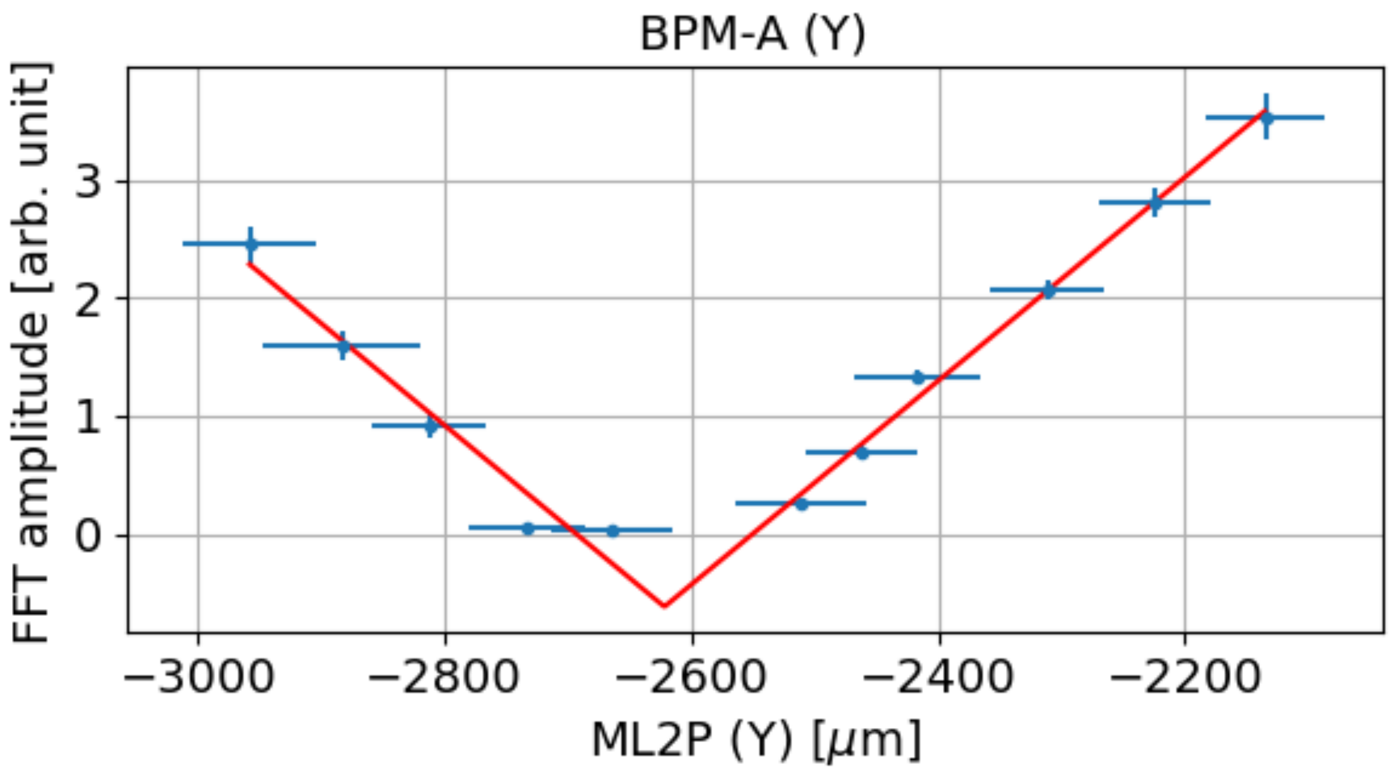
FFT values vs Stripline BPM

```python
steering = data.groupby("steering_current").groups.keys()
mean_ax = data.groupby("steering_current")["fft_ax"].mean()
std_ax = data.groupby("steering_current")["fft_ax"].std()

mean_strip = data.groupby("steering_current")["stripline_x"].mean()
std_strip = data.groupby("steering_current")["stripline_x"].std()

plt.plot(mean_strip, mean_ax, "*")
plt.show()
```



- Do the same thing for other channels
- Can you add error bars?





Fit FFT value vs stripline BPM

- Import lmfit:

```
from lmfit import Model
```

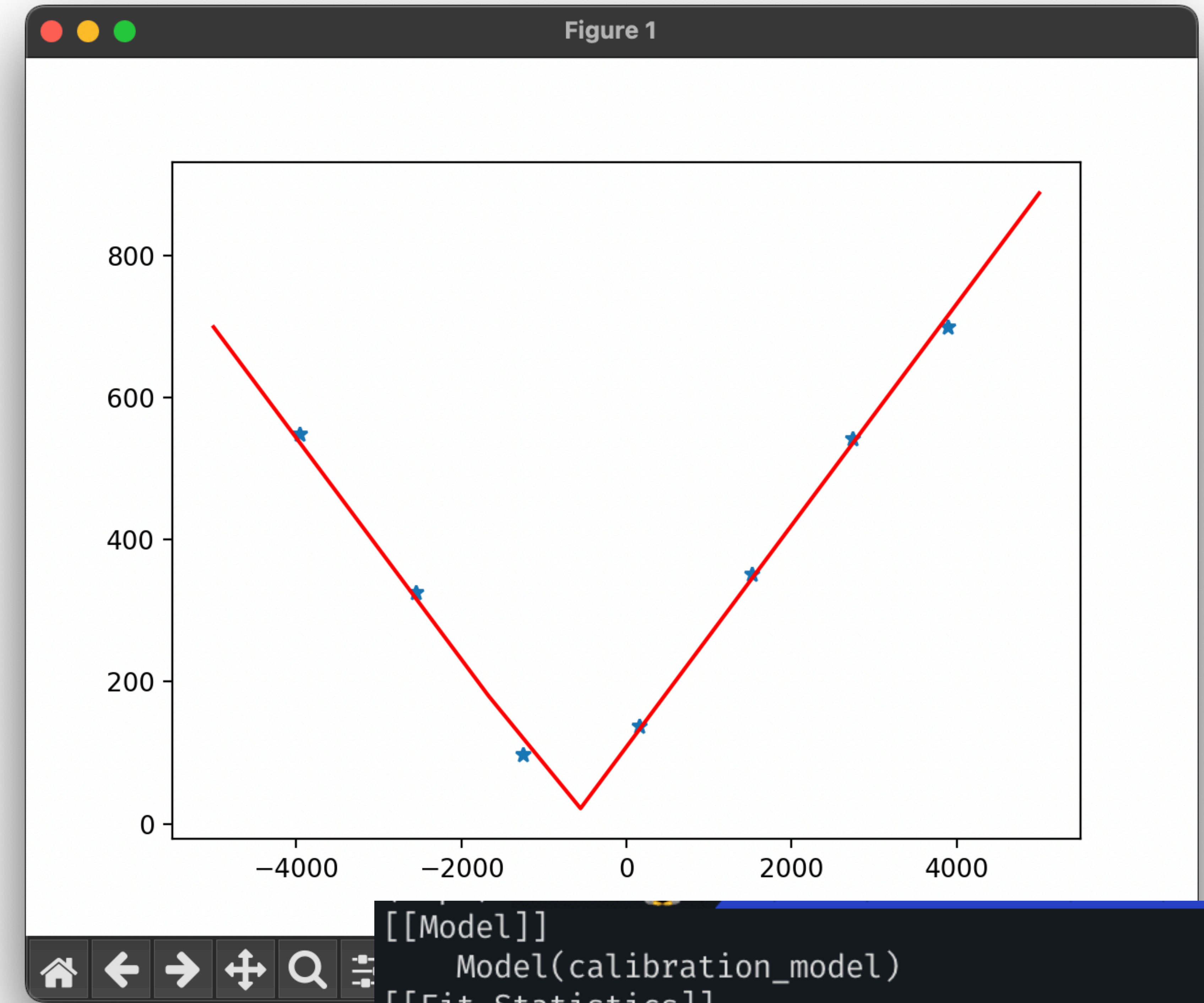
- Define fit model ($y = a\,|x - b| + c$)

```
def calibration_model(x, a, b, c):
    return a * np.abs(x - b) + c
```

- Fit the data

```
model = Model(calibration_model)
params = model.make_params(a=0.5, b=1000, c=0)
result = model.fit(mean_ax, params, x=mean_strip)

x_fit = np.linspace(-5000, 5000, 10)
y_fit = result.eval(x=x_fit)

print(result.fit_report())

plt.plot(mean_strip, mean_ax, "*")
plt.plot(x_fit, y_fit, "r-")
plt.show()
```

- Obtain the calibration factor for others




```
[[Model]]
    Model(calibration_model)
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 21
    # data points      = 7
    # variables        = 3
    chi-square         = 893.561346
    reduced chi-square = 223.390336
    Akaike info crit   = 39.9451339
    Bayesian info crit = 39.7828643
    R-squared          = 0.99700487
[[Variables]]
    a:   0.15597949 +/- 0.00442846 (2.84%) (init = 0.5)
    b: -604.508096 +/- 37.9095465 (6.27%) (init = 1000)
    c:   13.9805554 +/- 11.8113107 (84.48%) (init = 0)
[[Correlations]] (unreported correlations are < 0.100)
    C(a, c) = -0.8755
    C(a, b) = +0.2613
    C(b, c) = -0.1621
```

Resolution Analysis

- Read the resolution data file

```
data = pd.read_pickle("resolution_2023.pickle")
```

- Plot all waveforms
- Perform FFT for all channels
- Collect the integrated FFT values over all measurements and plot them

Singular Value Decomposition (SVD)

- Import necessary package

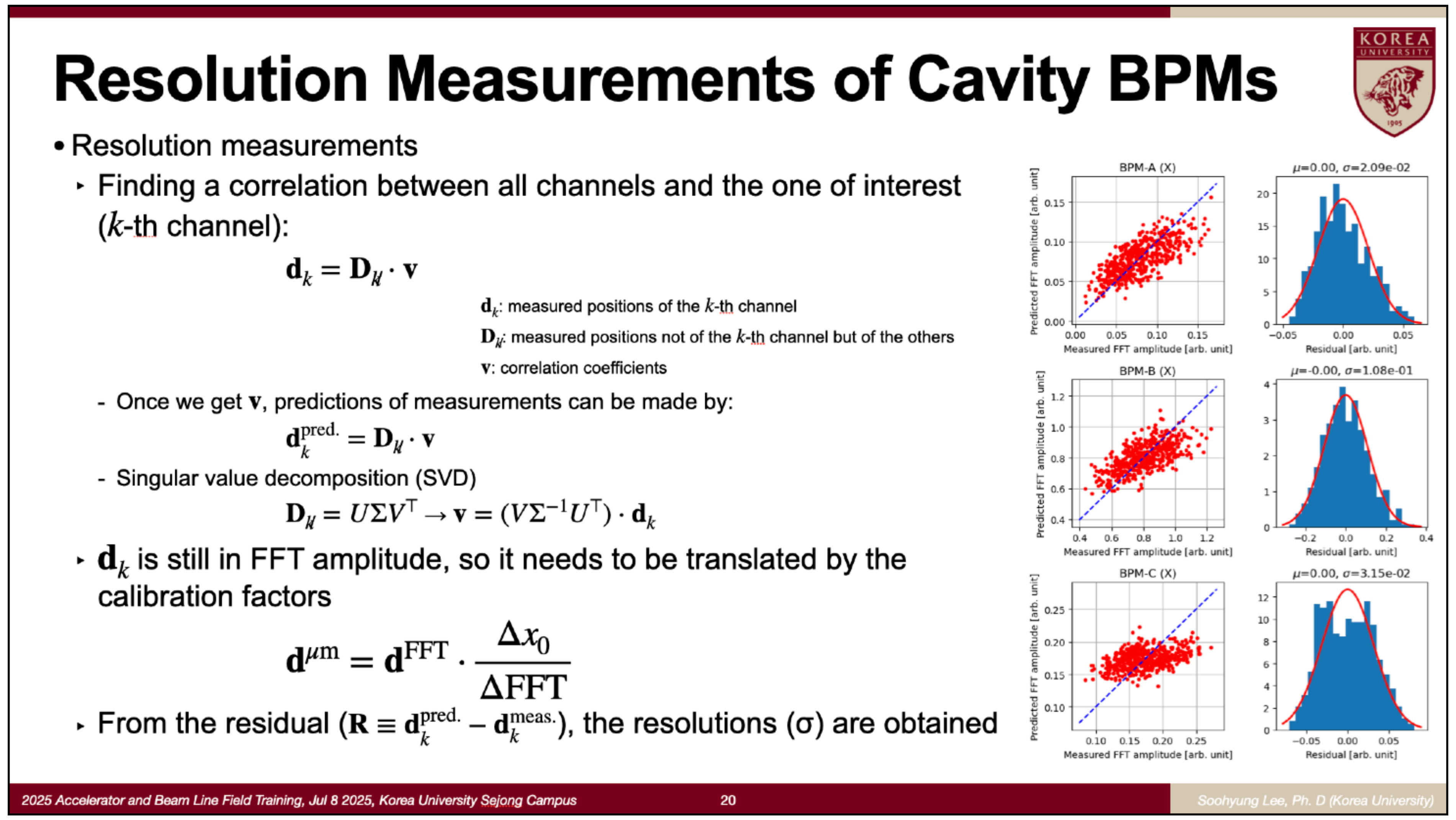
```python
from scipy.linalg import svd
```

- Construct necessary matrices

```python
mean_ax = data["fft_ax"].mean()

const_list = np.ones(len(data["fft_ax"]))

D = [
    np.column_stack((const_list, data["fft_ay"], data["fft_bx"], data["fft_by"], data["fft_cx"], data["fft_cy"])),
    np.column_stack((data["fft_ax"], const_list, data["fft_bx"], data["fft_by"], data["fft_cx"], data["fft_cy"])),
    np.column_stack((data["fft_ax"], data["fft_ay"], const_list, data["fft_by"], data["fft_cx"], data["fft_cy"])),
    np.column_stack((data["fft_ax"], data["fft_ay"], data["fft_bx"], const_list, data["fft_cx"], data["fft_cy"])),
    np.column_stack((data["fft_ax"], data["fft_ay"], data["fft_bx"], data["fft_by"], const_list, data["fft_cy"])),
    np.column_stack((data["fft_ax"], data["fft_ay"], data["fft_bx"], data["fft_by"], data["fft_cx"], const_list))
]
```

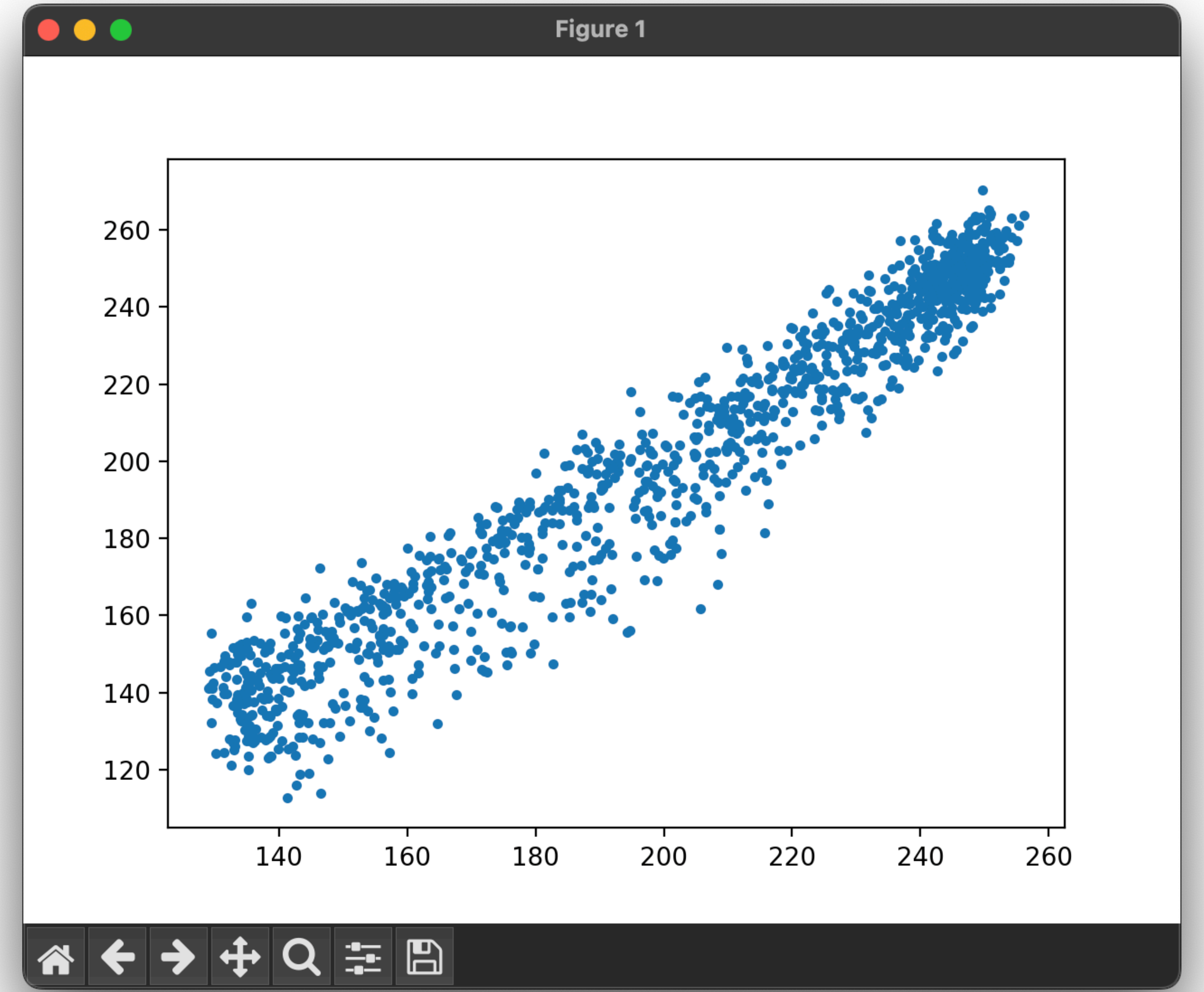
- SVD analysis

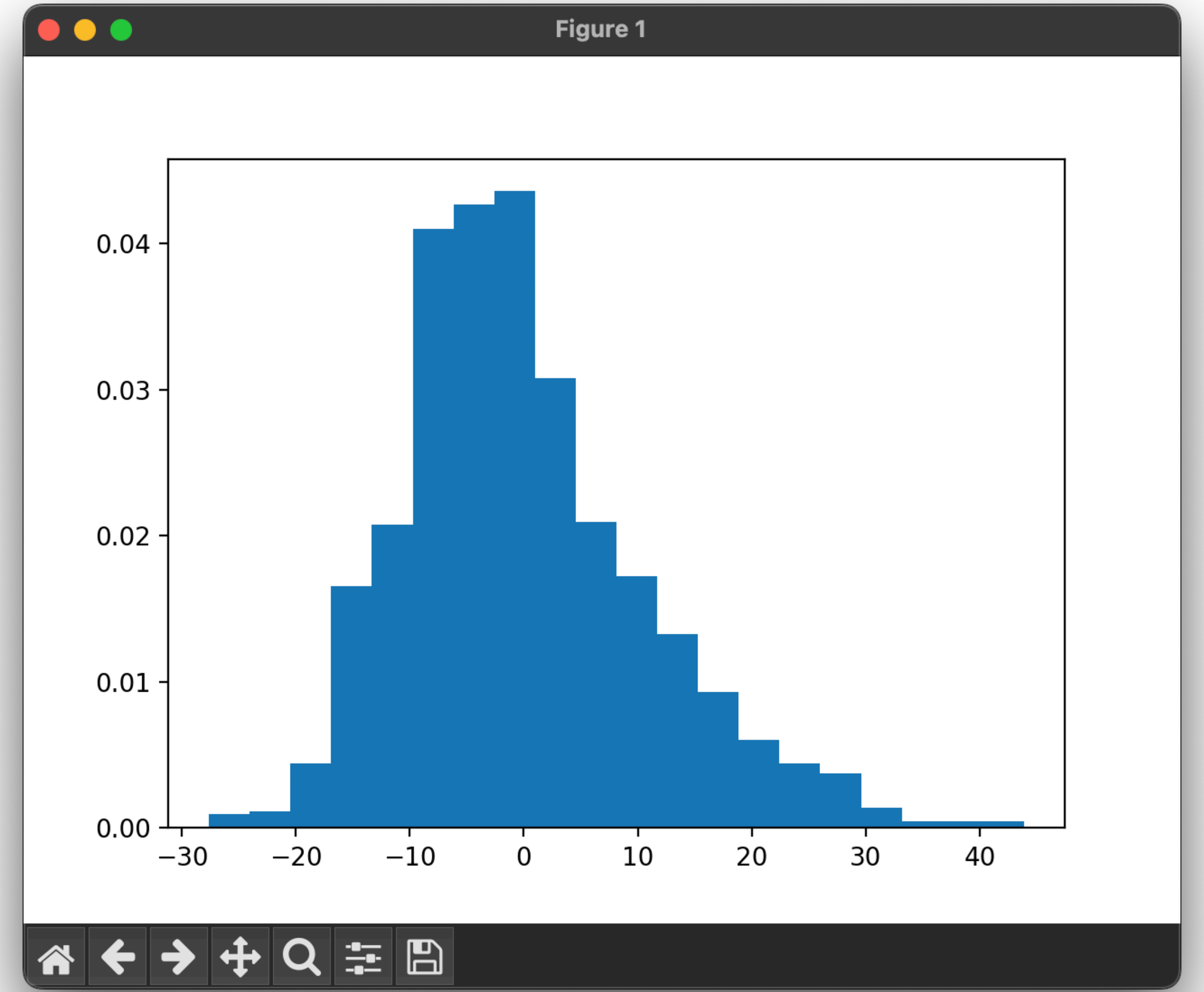
```python
U, S, Vt = svd(D[0], full_matrices=False)
x = Vt.T @ ((U.T @ data["fft_ax"]) / S)
predict = D[0] @ x
```



- Plot prediction vs measurement for BPM-A(X)
- Plot residual (prediction - measurement) distribution

Singular Value Decomposition (SVD)

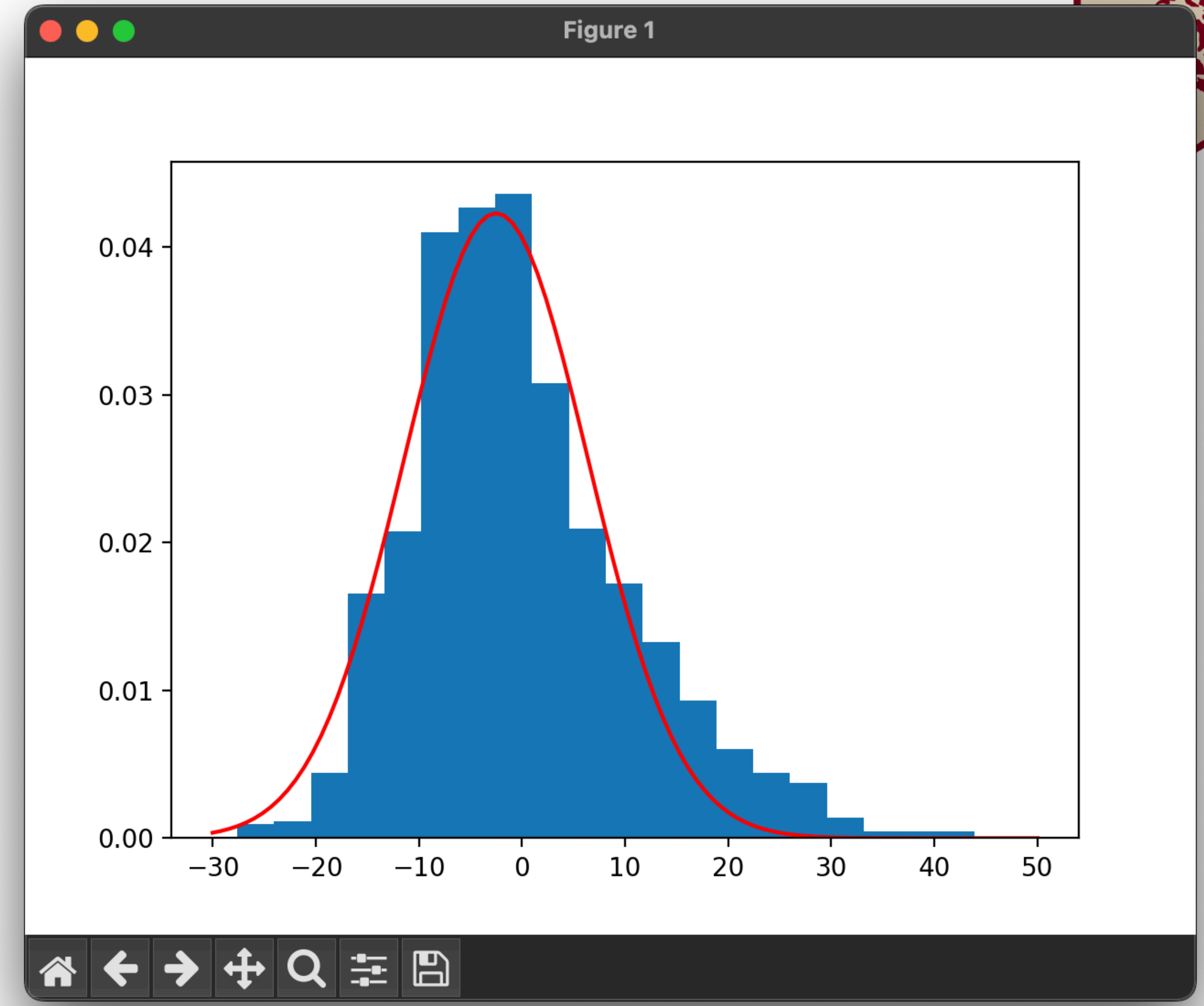




Fit residual with Gaussian

- We use lmfit again to fit the residual with Gaussian

```python
hist_y, hist_x, _ = plt.hist(predict - data["fft_ax"], bins=20, density=True)
hist_x_centers = (hist_x[:-1] + hist_x[1:]) / 2

fit = lmfit.models.GaussianModel()
params = fit.guess(hist_y, x=hist_x_centers)
result = fit.fit(hist_y, params, x=hist_x_centers)
print(result.fit_report())

fit_x = np.linspace(-30, 50, 100)
fit_y = result.eval(x=fit_x)

plt.plot(fit_x, fit_y, "r-")

plt.show()
```




```
[[Model]]
    Model(gaussian)
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 41
    # data points      = 20
    # variables        = 3
    chi-square         = 2.3723e-04
    reduced chi-square = 1.3955e-05
    Akaike info crit   = -220.843992
    Bayesian info crit = -217.856795
    R-squared          = 0.94401465
[[Variables]]
    amplitude:  0.94563893 +/- 0.04868102 (5.15%) (init = 0.69375)
    center:    -2.50625249 +/- 0.52975796 (21.14%) (init = -2.590531)
    sigma:      8.91768509 +/- 0.53076291 (5.95%) (init = 5.362576)
    fwhm:       20.9995432 +/- 1.24985112 (5.95%) == '2.3548200*sigma'
    height:     0.04230418 +/- 0.00217709 (5.15%) == '0.3989423*amplitude/max(1e-15, sigma)'
[[Correlations]] (unreported correlations are < 0.100)
    C(amplitude, sigma) = +0.5784
```

The resolution

- Residual in FFT value = 8.92
- Calibration factor = 0.16 / μm

- Position resolution = (FFT) / (calibration factor) = 8.92 / 0.16 = 55.75 μm

- This value is not final since there are further corrections that I won’t cover in this lecture

- Find the resolutions for other channels