Julian Lapenna Acoustics Lab

Sept 19, 16:20

Logged on. I don't see a `BareBonesDAQ.m` file... Maybe a python file has replaced it

In [1]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

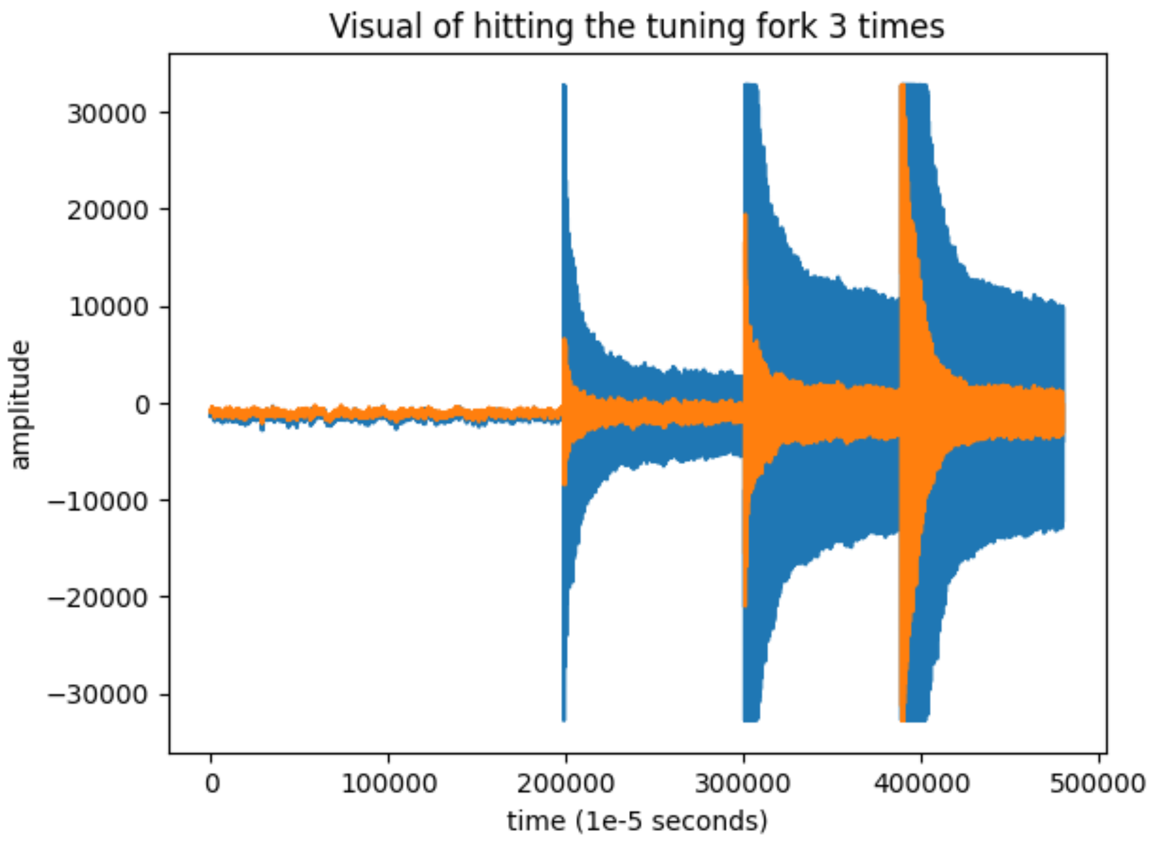
In [3]:
```
samplerate, data = wavfile.read('../data/resonant.wav')
print(data.shape)
print ("Frequency sampling:", samplerate)
```

```
(480000, 2)
Frequency sampling: 96000
```

In [4]:
```
print(len(data)/samplerate) # time sampled for
```

```
5.0
```

In [5]:
```
plt.plot(data)
plt.title('Visual of hitting the tuning fork 3 times')
plt.xlabel('time (1e-5 seconds)')
plt.ylabel('amplitude');
```



In [6]:
```
print(data[200000]) # looks about right with graph above
```

```
[8412   116]
```

File naming convention (for future):

pXXXX-fXXX-aXX-tXX-X.wav = p<position>-f<frequency>-a<amplitude>-t<time>-<close\far>.wav

where

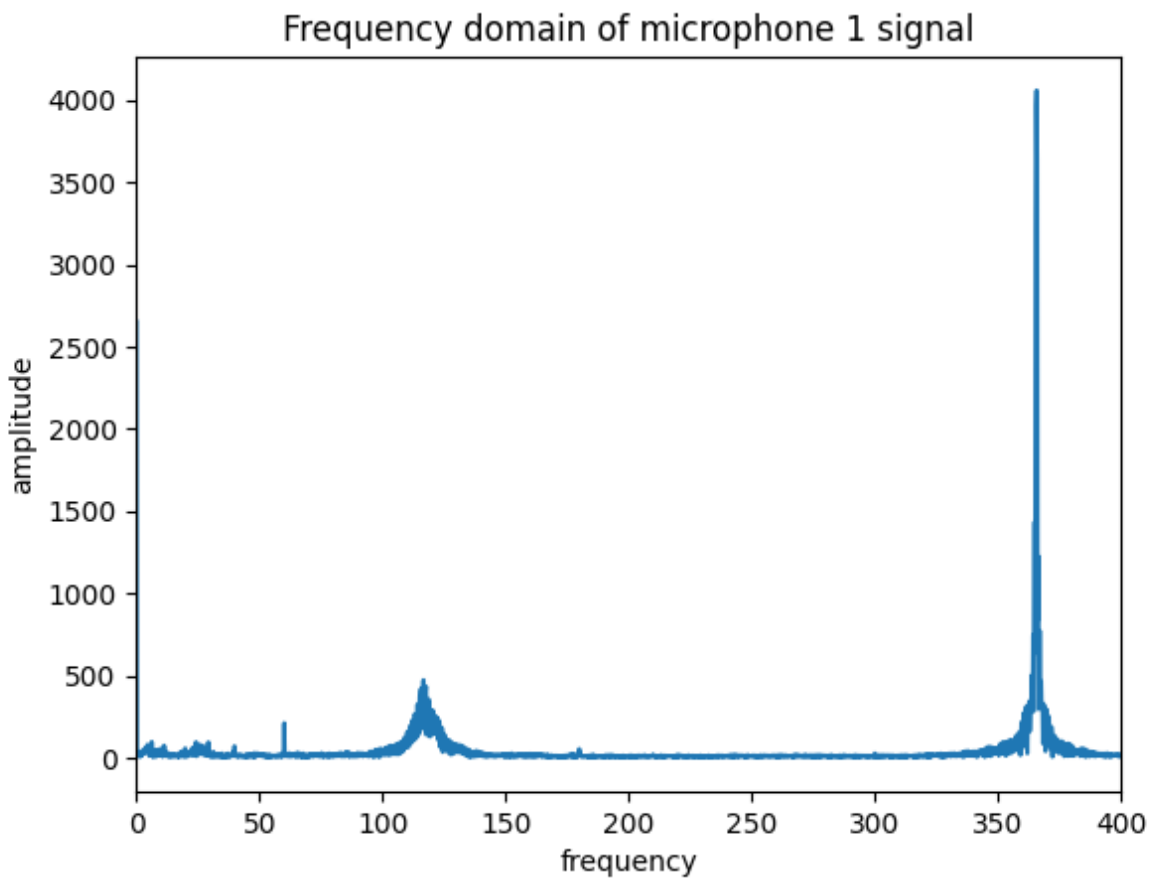
- position = number of steps of the stepper motor and tells me the angle I should be observing
- frequency = 365 Hz the fundamental frequency
- amplitude = the amplitude the motor is set to, for now probably 5
- time = how long the motor goes for, for now probably 5 seconds
- c/f = for movable mic, is it close (~1-3cm) or far (~20-50cm) away

In [7]:
```
data1 = []

for d in data:
    data1.append(d[0])

print(len(data1))
```

```
480000
```

In [4]:
```
# finding the resonant frequency
from scipy.fft import fft, fftfreq, ifft
```

In [8]:
```
N = 480000 # sample points
T = 1 / samplerate

yf = fft(data1)
xf = fftfreq(N, T)[:N//2]

# why I divide by N:
# https://electronics.stackexchange.com/questions/25900/
# scaling-fft-output-by-number-of-points-in-fft
# summary: there are N samples of each component of the signal that scale
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.title('Frequency domain of microphone 1 signal')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xlim([0,400])
plt.show()
```



In [9]:
```
print("resonant frequency:", xf[np.argmax(yf)], "Hz")
```

```
resonant frequency: 365.20000000000005 Hz
```

So the resonant frequency of the tuning fork is 365 Hz! Great :)

Sept 25, 15:25

Plan, get data from the close field, again in a 90deg sweep since the tuning fork is symmetric, then analyse the data later.

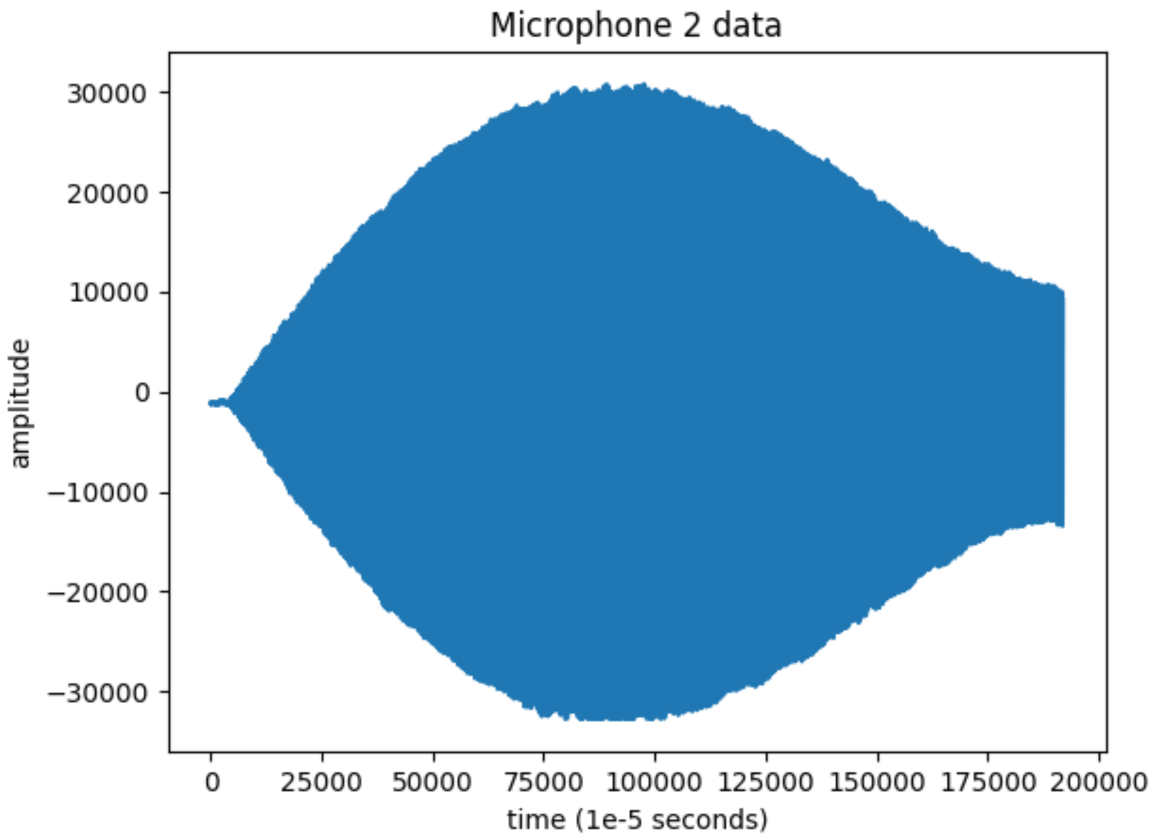
Issue that came up: The stepper motor that is used to position the mic at different positions was broken, so I took a protractor and switched to measuring the angle by hand. This will be a source of error, but I am hoping the data will still show the appropriate trends.

In [10]:
```
def show_audio_file(filename):
    samplerate, data = wavfile.read(filename)
    print(data.shape)
    print ("Frequency sampling:", samplerate)
    plt.plot(data)

def show_freqs_audio_file(filename):
    samplerate, data = wavfile.read(filename)

    data1 = []

    for d in data:
        data1.append(d[1]) # getting mic 2, the one that moves
    N = len(data1) # sample points
    T = len(data) / samplerate

    yf = fft(data1)
    xf = fftfreq(N, T)[:N//2]

    plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
    plt.title('frequency domain of signal')
    plt.xlabel('frequency')
    plt.ylabel('amplitude')
    plt.xlim([0,600])
    plt.show()
```

In [11]:

```
filename = '../data/deg05-f365-a05-t02-c.wav'
# show_audio_file(filename)
# show_freqs_audio_file(filename)

samplerate, data = wavfile.read(filename)

data1 = []

for d in data:
    data1.append(d[1]) # getting mic 2, the one that moves

plt.plot(data1)
plt.title('Microphone 2 data')
plt.ylabel('amplitude')
plt.xlabel('time (1e-5 seconds)')
print(data.shape)
print(len(data1))

for i in range(10):
    print(data1[i])

N = len(data1) # sample points
T = len(data) / samplerate

print(N,T)

# yf = fft(data1)
# xf = fftfreq(N, T)[:N//2]

# plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
# plt.title('frequency domain of signal')
# plt.xlabel('frequency')
# plt.ylabel('amplitude')
# plt.xlim([0,600])
# plt.show()
```

```
(192000, 2)
192000
-1124
-1103
-1098
-1083
-1080
-1076
-1070
-1065
-1069
-1078
192000 2.0
```



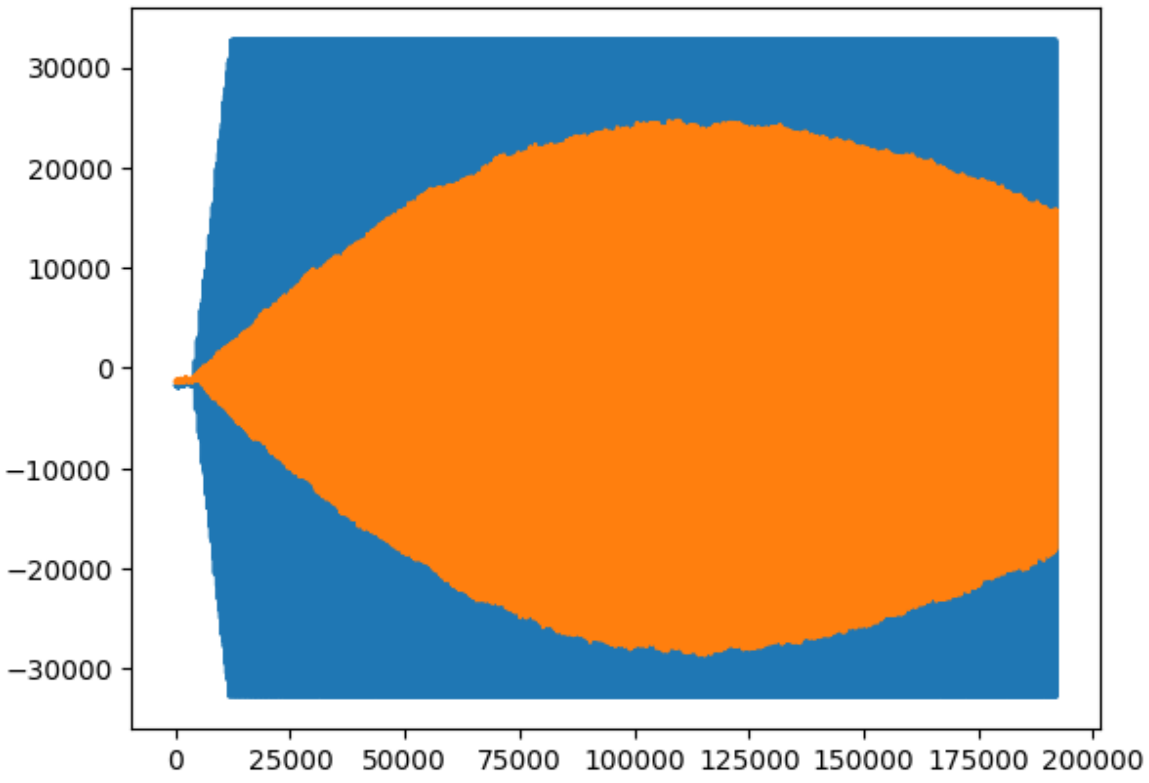
Sept 26, 15:35

Hold on, I don't even need to take the fourier transform, I'm looking at amplitude. So maybe let's look at some of the graphs of the far and close field at the expect minimums.

Close field 45deg, far field 90deg. Additionally, r for close = 1cm, while r for far = 20cm.

In [12]:
```
filename = '../data/deg45-f365-a05-t02-c.wav'
show_audio_file(filename)
```

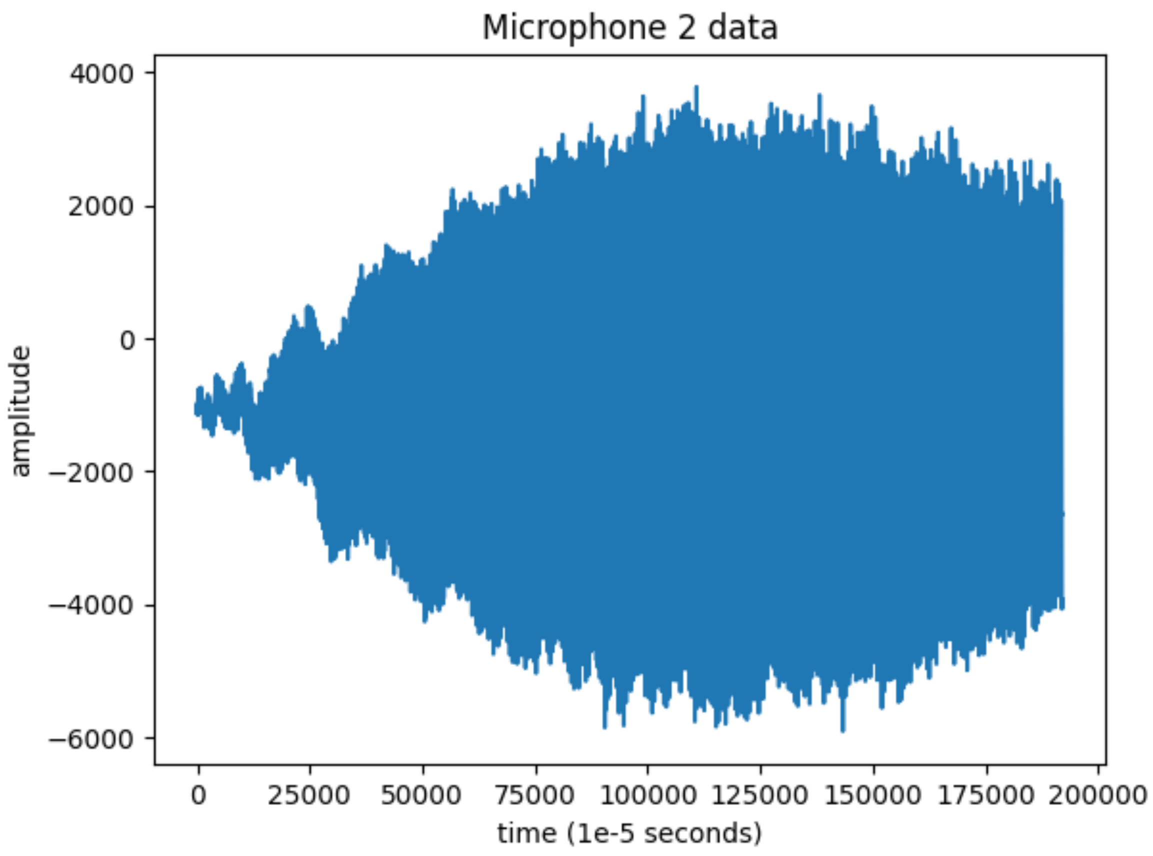
```
(192000, 2)
Frequency sampling: 96000
```



In [13]:
```
print(np.max(data))

filename = '../data/deg70-f365-a05-t02-c.wav'
samplerate, data = wavfile.read(filename)
print(data.shape)
print ("Frequency sampling:", samplerate)

plt.plot([subdata[1] for subdata in data])
plt.title('Microphone 2 data')
plt.ylabel('amplitude')
plt.xlabel('time (1e-5 seconds)')

# plt.ylim([-4e4, 4e4])
```

```
32767
(192000, 2)
Frequency sampling: 96000
```

Out[13]:
```
Text(0.5, 0, 'time (1e-5 seconds)')
```



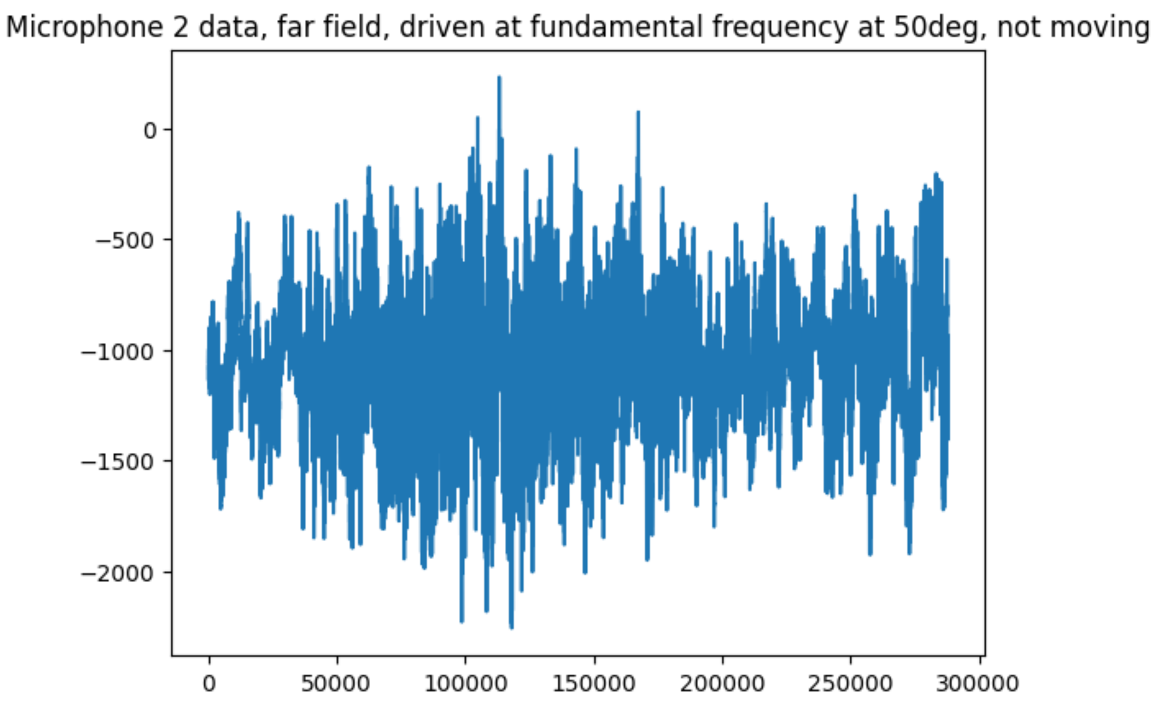
Ahhhh, so after increasing the bounds, the top and bottom are chopped, likely the mic is saturated, I will have to take data at a lower amplitude I think.

How about at another 5deg increment? How about at a maximum?

Additionally mic 1 appears to be closer to the tuning fork.

```
In [14]: filename = '../data/p800-f365-a05-t03-f.wav'
         samplerate, data = wavfile.read(filename)

         print(data.shape)
         print ("Frequency sampling:", samplerate)

         plt.plot(np.take(data, 1, axis=1))
         plt.title('Microphone 2 data, far field, driven at fundamental frequency
         print('avg val:', np.average(np.take(data, 1, axis=1)))
```

```
(288000, 2)
Frequency sampling: 96000
avg val: -1074.4259965277777
```



Maybe there is a reason to look at the frequency domain, to see the amplitude of a given frequency...

Plan

- One microphone was saturated, so take data from the other (that also makes the scale readable)
- There is a linear relationship between air pressure and intensity recorded by the microphone
- I think that a Fourier transform will let me look at how loud the resonant frequency was in a given recording
  - I want the resonant frequency amplitude because this is the frequency I am interested in!
- Test this for a few different values, then make a first attempt at a radial plot (I think that's the correct name...) like the one from the paper by Russel:

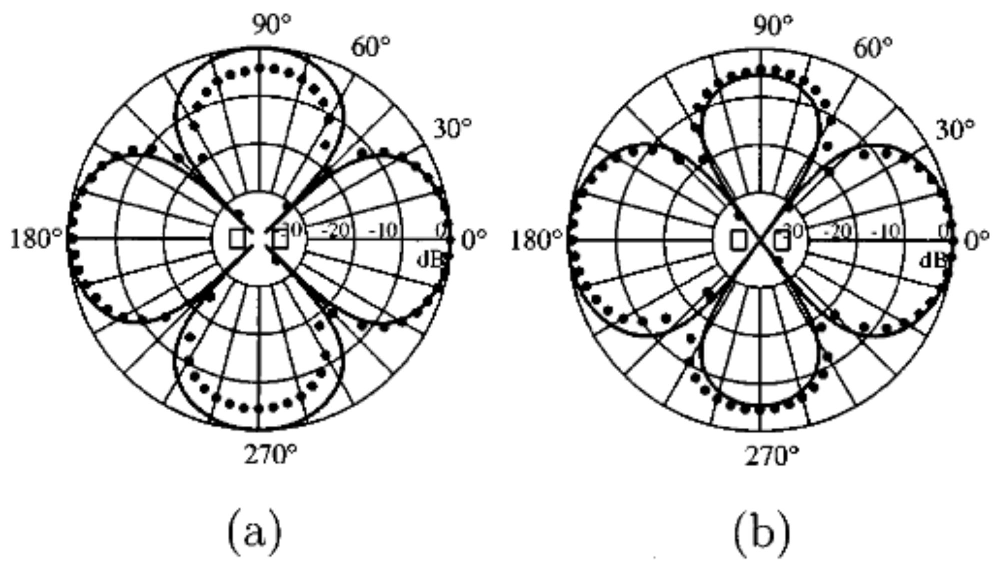
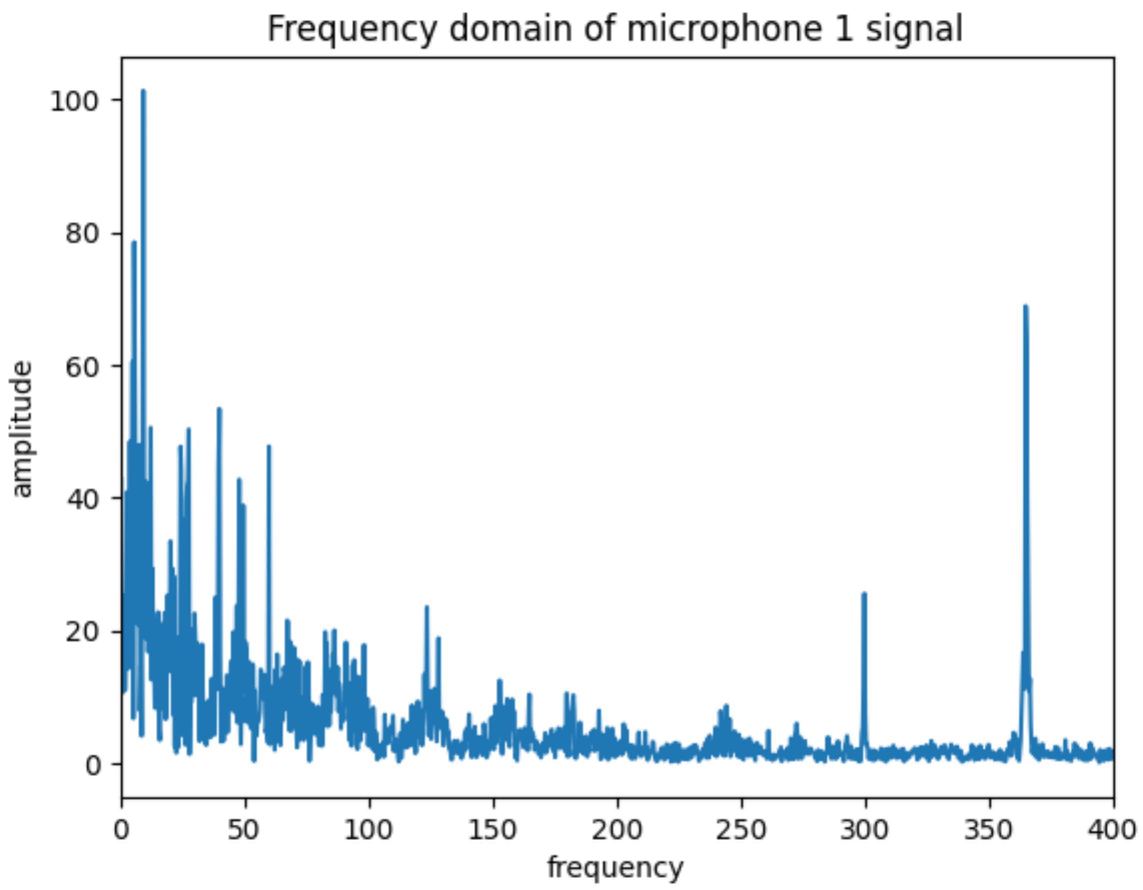


Fig. 9. Sound pressure level directivity patterns 5 cm from the small 426-Hz fork. (a) Data fit with lateral quadrupole from Eq. (2), (b) data fit with linear quadrupole from Eq. (4).

In [15]:
```
fdata = np.take(data, 1, axis=1)
fdata = fdata - np.average(fdata) # to get rid of the dc offset

N = len(fdata) # sample points
T = 1 / samplerate

print(N, T)

yf = np.abs(fft(fdata)[0:N//2])
xf = fftfreq(N, T)[:N//2]

plt.plot(xf, 2.0/N * yf)
plt.title('Frequency domain of microphone 1 signal')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xlim([0,400])
plt.show()
```

```
288000 1.0416666666666666e-05
```



In [16]:
```
max_index = np.argmax(yf)
print('max_index:', max_index)
print('frequency:', xf[max_index])
print('amp. val.:', yf[max_index]/N*2)
```

```
max_index: 28
frequency: 9.333333333333332
amp. val.: 101.33536426834647
```

Okay, I obviously know that it should be the resonant frequency, but it seems some dc offset is making it look like a dc frequency 0 is the main value. I guess I can try searching at greater than 10Hz (or rather 10 indicies).

EDIT: This was now fixed after I went and subtracted the average to get rid of the DC offset at the first step. I originally didn't do that, and thought of it afterwards, and just changed the code above.

In [17]:
```
max_index = np.argmax(yf[10:]) + 10
print('max_index:', max_index)
print('frequency:', xf[max_index])
print('amp. val.:', yf[max_index]/N*2)
```

```
max_index: 28
frequency: 9.333333333333332
amp. val.: 101.33536426834647
```

I think at this point I see a clear peak at the resonant frequency and it is probably time to build the data for the radial plot.

Something like

`radial_data = (angle, amplitude value)`
some of the data I took with a protractor and measured the angle by hand (definitely some error there), other data I moved the stepper in increments of 80 which is approximately 5 degrees:

5730 steps / 360 deg = 15.91 steps/deg

15.91 steps/deg * 5 deg = 79.58 steps ≈ 80 steps for 5 degrees

In [18]:
```
radial_data_close_field = []
radial_data_far_field = []
```

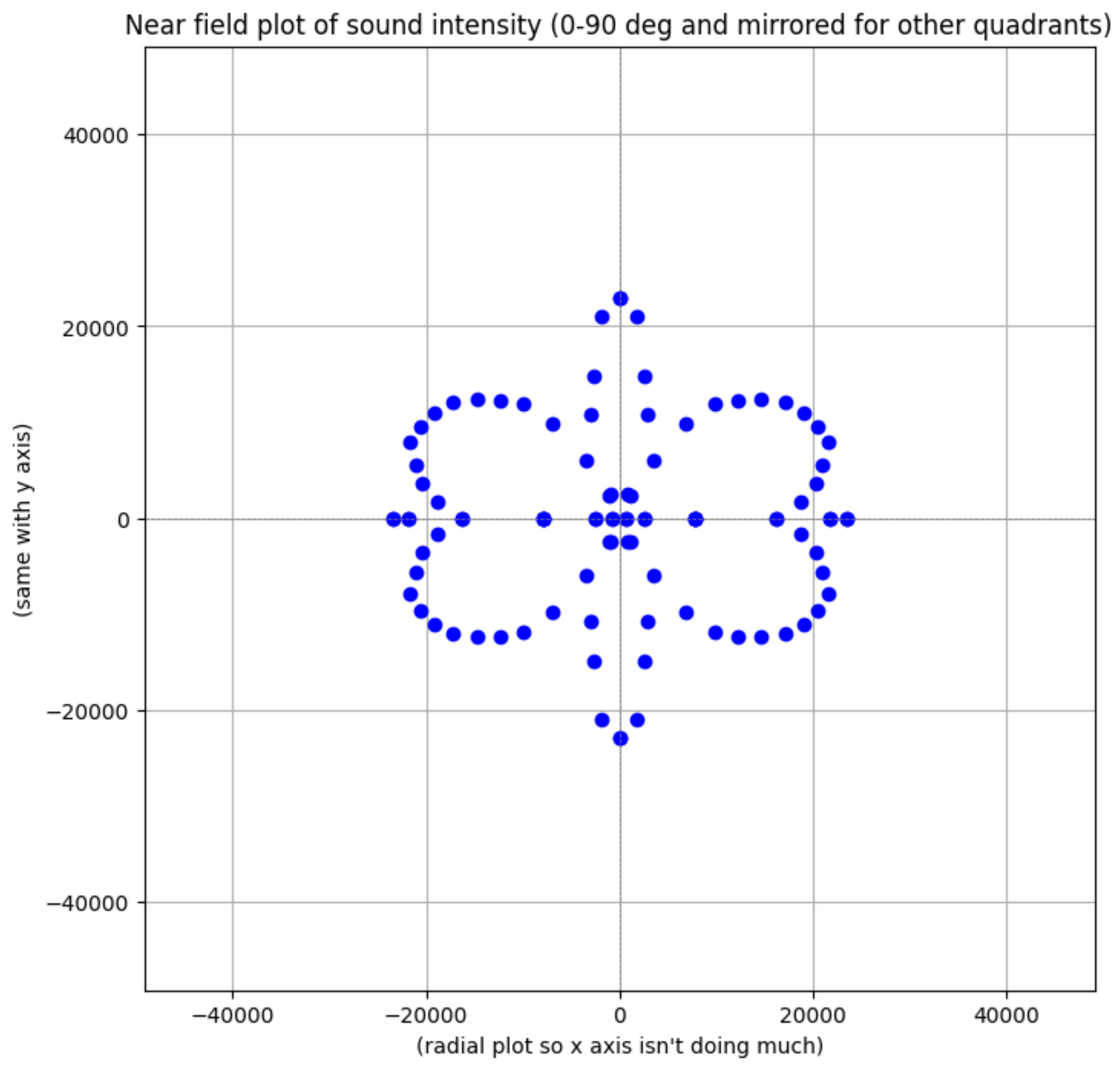
In [19]:
```
# function to get the amplitude value
# should take in a filename
def get_amplitude(filename):
    samplerate, data = wavfile.read(filename)

    fdata = np.take(data, 1, axis=1) # mic that is not saturated
    fdata = fdata - np.average(fdata) # to get rid of the dc offset

    N = len(fdata) # sample points
    T = 1 / samplerate

    yf = np.abs(fft(fdata)[0:N//2])
    xf = fftfreq(N, T)[:N//2]

    max_index = np.argmax(yf)

    # we should be looking at 365 Hz
    if not (xf[max_index] > 360):
        print(filename, xf[max_index])
    if not (xf[max_index] < 370):
        print(filename, xf[max_index])

    return yf[max_index]/N*2
```

In [20]:
```
import os
TOTAL_MOTOR_STEPS = 5730

for filename in os.listdir('../data'):
    if filename.endswith('c.wav'): # close field
        amp = get_amplitude('../data/' + filename)

        if filename.startswith('deg'):
            degree = int(filename.split('-')[0][3:])  # Skip 'deg' and ge

        if filename.startswith('p'):
            degree = int(int(filename.split('-')[0][1:])/TOTAL_MOTOR_STEP

        file_data = (amp, degree)
        radial_data_close_field.append(file_data)

    elif filename.endswith('f.wav'): # far field
        amp = get_amplitude('../data/' + filename)

        if filename.startswith('deg'):
            # Skip 'deg' and get the number
            degree = int(filename.split('-')[0][3:])
        if filename.startswith('p'):
            degree = int(int(filename.split('-')[0][1:])/TOTAL_MOTOR_STEP

        file_data = (amp, degree)
        radial_data_far_field.append(file_data)
```

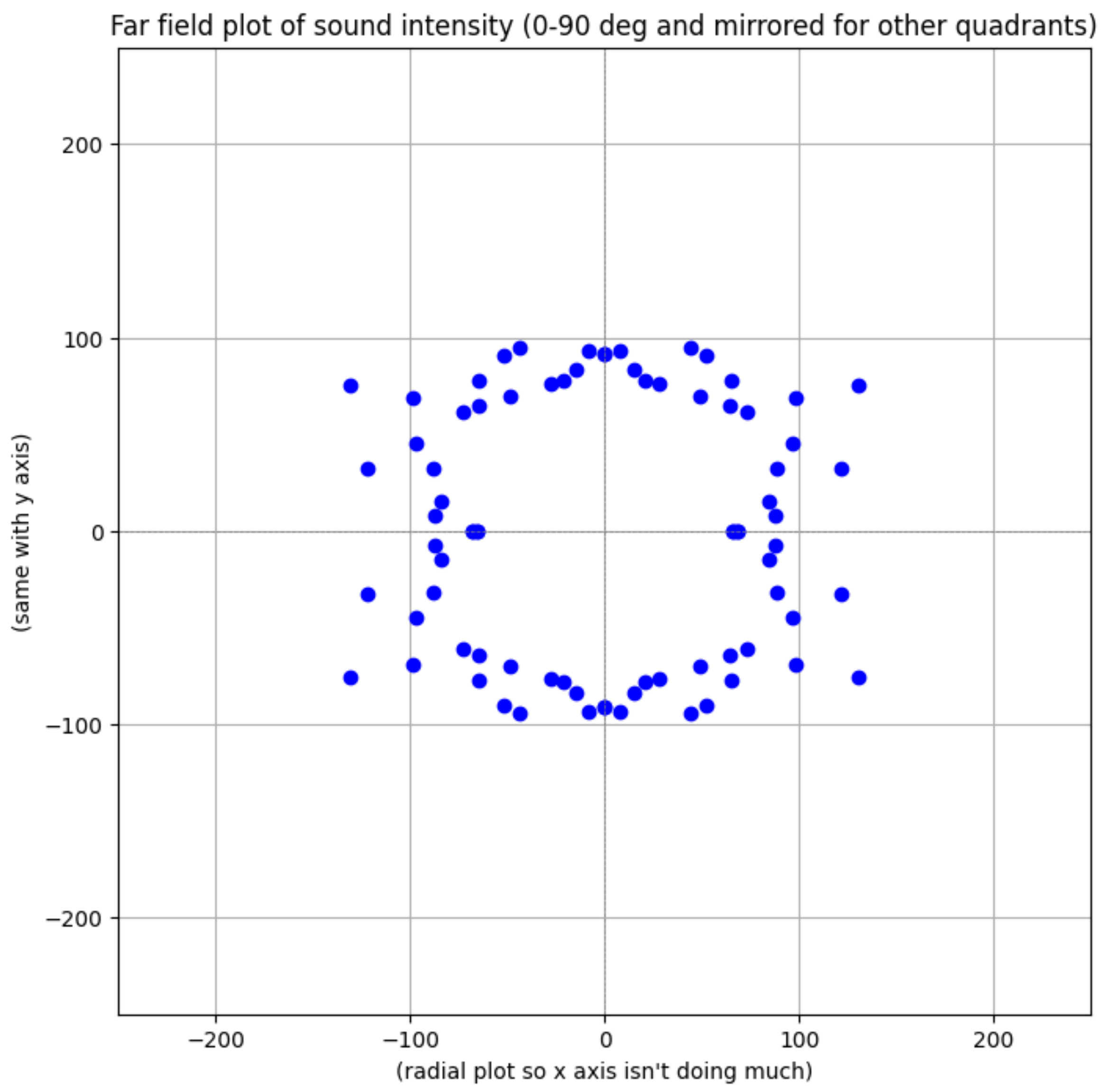
```
../data/p800-f365-a05-t03-f.wav 9.333333333333332
../data/p480-f365-a05-t02-f.wav 548.0
../data/p1040-f365-a05-t03-f.wav 4.666666666666666
../data/p00-f365-a05-t05-f.wav 8.8
../data/p1440-f365-a05-t03-f.wav 730.0
../data/p1120-f365-a05-t03-f.wav 8.666666666666666
../data/p1280-f365-a05-t03-f.wav 730.0
../data/p240-f365-a05-t02-f.wav 730.5
../data/p00-f366-a05-t05-f.wav 0.2
../data/p000-f370-a05-t05-c.wav 370.0
../data/p960-f365-a05-t03-f.wav 25.0
../data/p1200-f365-a05-t03-f.wav 6.0
../data/p1360-f365-a05-t03-f.wav 730.0
../data/p560-f365-a05-t03-f.wav 547.6666666666666
../data/p000-f380-a05-t05-c.wav 380.0
../data/p400-f365-a05-t02-f.wav 548.0
../data/p80-f365-a05-t02-f.wav 28.0
```

Okay, there's some issues to be investigated, possibly data to be retaken, but for now we can plot!

In [21]:
```
x_values = []
y_values = []

for magnitude, angle in radial_data_close_field:
    angle_rad = np.radians(angle)

    # Calculate x and y coordinates
    x = magnitude * np.cos(angle_rad)
    y = magnitude * np.sin(angle_rad)

    # Append the original and mirrored coordinates
    x_values.extend([x, -x, -x, x])
    y_values.extend([y, y, -y, -y])

plt.figure(figsize=(8, 8))
plt.plot(x_values, y_values, 'bo')  # 'bo' blue circle markers
plt.axhline(0, color='gray', lw=0.5, ls='--')
plt.axvline(0, color='gray', lw=0.5, ls='--')
plt.xlim(-max(magnitude for magnitude, angle in data)
         * 1.5, max(magnitude for magnitude, angle in data) * 1.5)
plt.ylim(-max(magnitude for magnitude, angle in data)
         * 1.5, max(magnitude for magnitude, angle in data) * 1.5)
plt.title('Near field plot of sound intensity (0-90 deg and mirrored for
plt.xlabel("(radial plot so x axis isn't doing much)")
plt.ylabel('(same with y axis)')
plt.grid()
```



In [22]:
```
x_values = []
y_values = []

for magnitude, angle in radial_data_far_field:
    angle_rad = np.radians(angle)

    # Calculate x and y coordinates
    x = magnitude * np.cos(angle_rad)
    y = magnitude * np.sin(angle_rad)

    # Append the original and mirrored coordinates
    x_values.extend([x, -x, -x, x])
    y_values.extend([y, y, -y, -y])

plt.figure(figsize=(8, 8))
plt.plot(x_values, y_values, 'bo')  # 'bo' blue circle markers
plt.axhline(0, color='gray', lw=0.5, ls='--')
plt.axvline(0, color='gray', lw=0.5, ls='--')
plt.xlim(-250, 250)
plt.ylim(-250, 250)
plt.title('Far field plot of sound intensity (0-90 deg and mirrored for o
plt.xlabel("(radial plot so x axis isn't doing much)")
plt.ylabel('(same with y axis)')
plt.grid()
```



Oct 2, 14:55

So I'm not quite sure these turned out too well. There is definitely lots of error that I can talk about, but for now I am going to get a second form of data to attempt a second plot that might be better.

Plan:

1. Record data while sweeping around 180degrees for a single plot
2. Compare that data
3. There are known issues with motor 1, if it's having trouble, do everything again but on motor 2

After testing the stepper and it looked clean, I will attempt on motor 1!

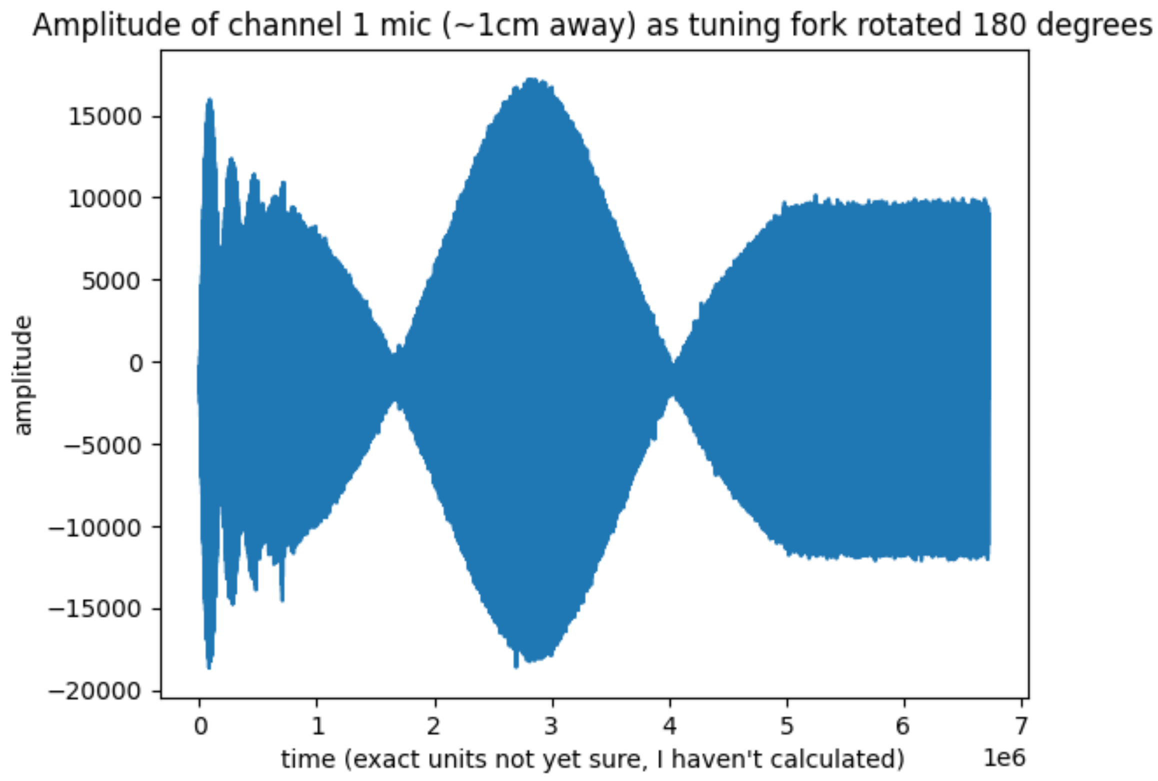
With a timer it takes ~ 1 min so I will record for 70 seconds and try to get some buffer time on either side.

Great, now save so I get my data, then try again in far field!

In [19]:
```
# not really sure what the data should look like, but let's see
filename = '../data/180sweep-m1.wav'
samplerate_1_sweep, data_1_sweep = wavfile.read(filename)

print(data_1_sweep.shape)
print ("Frequency sampling:", samplerate_1_sweep)

plt.title("Amplitude of channel 1 mic (~1cm away) as tuning fork rotated
plt.xlabel("time (exact units not yet sure, I haven't calculated)")
plt.ylabel('amplitude')
plt.plot(np.take(data_1_sweep, 1, axis=1))
```

```
(6720000, 2)
Frequency sampling: 96000
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x7646d03531a0>]
```



Hey! This actually sort of lines up with what I would want! The y-axis is the amplitude and x-axis is time, and all the while it is rotating at a constant angular velocity which I could figure out with some math.

This means when it rotated to ~1.8 on the graph, it reached a minima where the sound was very quiet and a there are maxima on either end and a larger one in the middle! This is exactly what I was expecting!!! I'm kinda shocked even though I've been studying this stuff for years lmao

Now for the far field

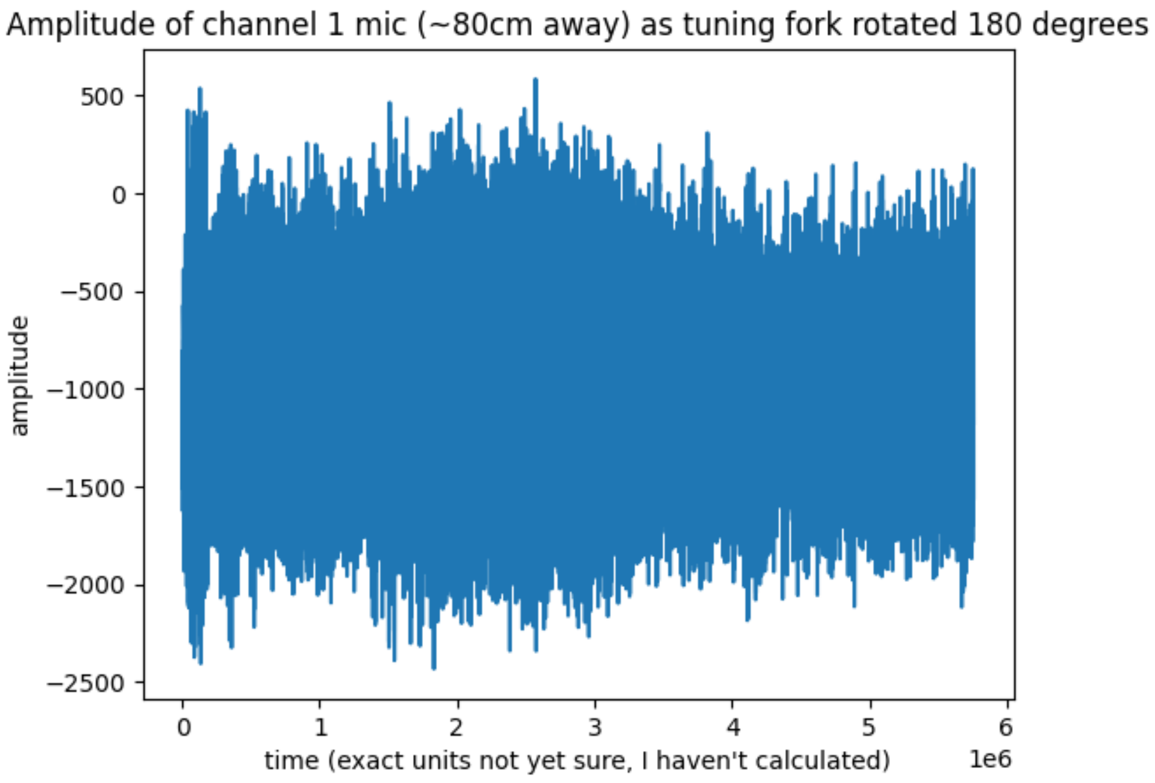
In [2]:
```
filename = '../data/180sweep-m1-far.wav'
samplerate_80_sweep, data_80_sweep = wavfile.read(filename)

print(data_80_sweep.shape)
print ("Frequency sampling:", samplerate_80_sweep)
```

```
plt.title("Amplitude of channel 1 mic (~80cm away) as tuning fork rotated
plt.xlabel("time (exact units not yet sure, I haven't calculated)")
plt.ylabel('amplitude')
plt.plot(np.take(data_80_sweep, 1, axis=1))
```

```
(5760000, 2)
Frequency sampling: 96000
```

Out[2]:

```
[<matplotlib.lines.Line2D at 0x73bc2355fa70>]
```



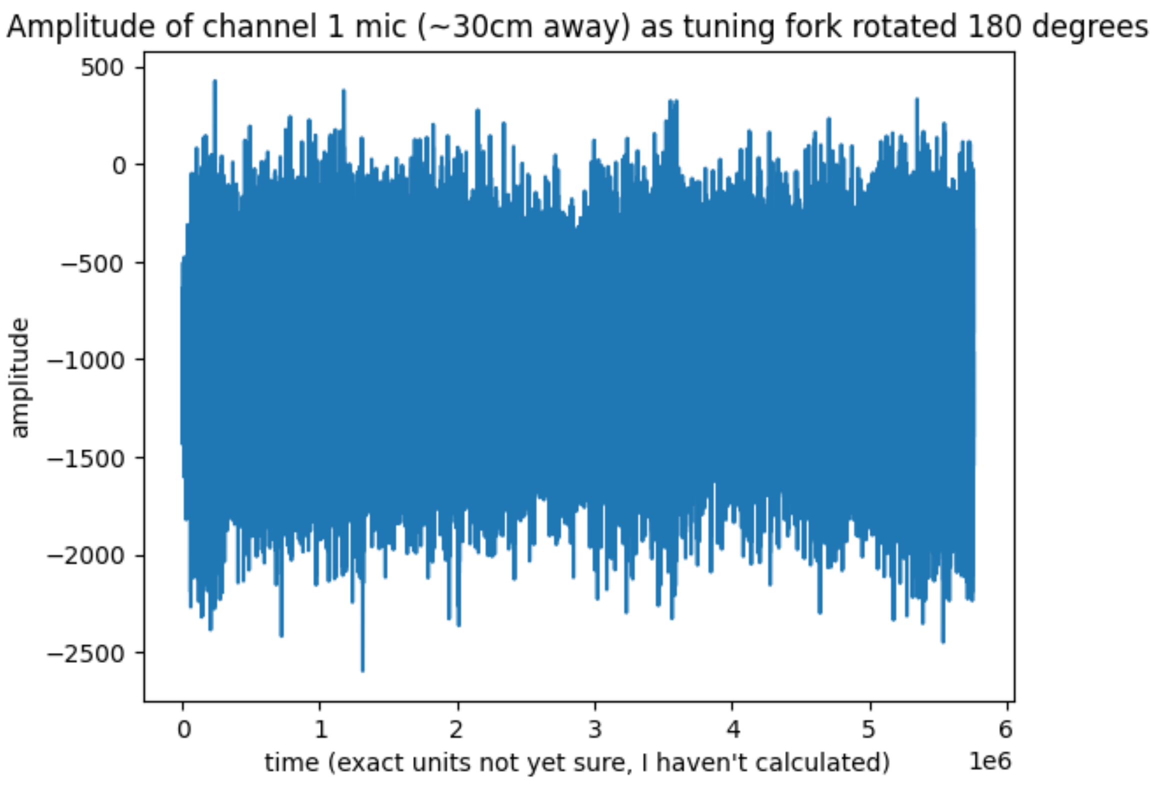
Hmmm okay this doesn't look nearly as good, but there's probably lots of noise... maybe I could take one more sample ~ 30cm away, and I can use that for a stronger signal and then try to denoise this using some filters.

In [25]:

```
filename = '../data/180sweep-m1-far30.wav'
samplerate_30_sweep, data_30_sweep = wavfile.read(filename)

plt.title("Amplitude of channel 1 mic (~30cm away) "+
          "as tuning fork rotated 180 degrees")
plt.xlabel("time (exact units not yet sure, I haven't calculated)")
plt.ylabel('amplitude')
plt.plot(np.take(data_30_sweep, 1, axis=1))
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x7ea40c374620>]
```

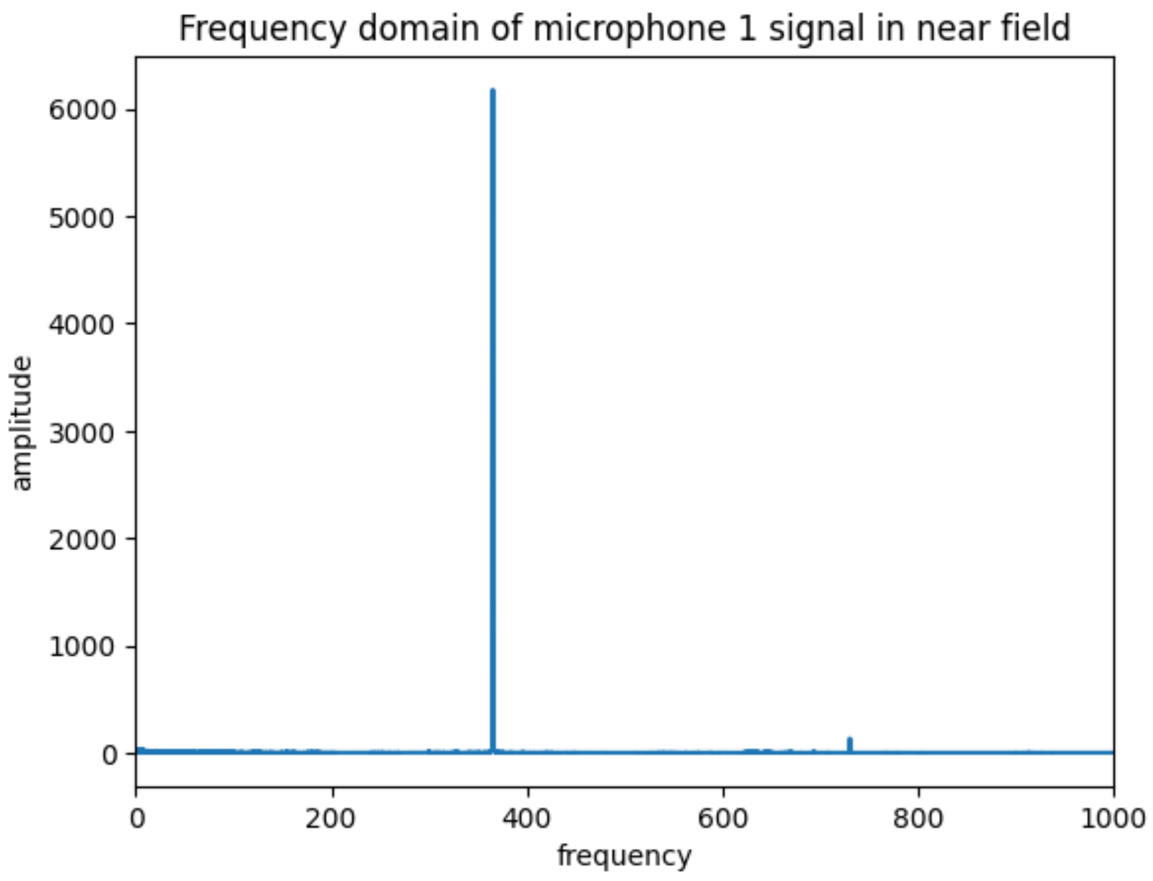


Okay, well more than enough to work with, but I'll have to do some processing I guess.

Let's check the frequency domain of each of these to make sure they're still all reading the resonant frequency and the main source of sound.

In [20]:
```
fdata = np.take(data_1_sweep, 1, axis=1)
fdata = fdata - np.average(fdata) # to get rid of the dc offset

N = len(fdata) # sample points
T = 1 / samplerate_1_sweep

print(N, T)

yf = np.abs(fft(fdata)[0:N//2])
xf = fftfreq(N, T)[:N//2]

plt.plot(xf, 2.0/N * yf)
plt.title('Frequency domain of microphone 1 signal in near field')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xlim([0,1000])
plt.show()
```

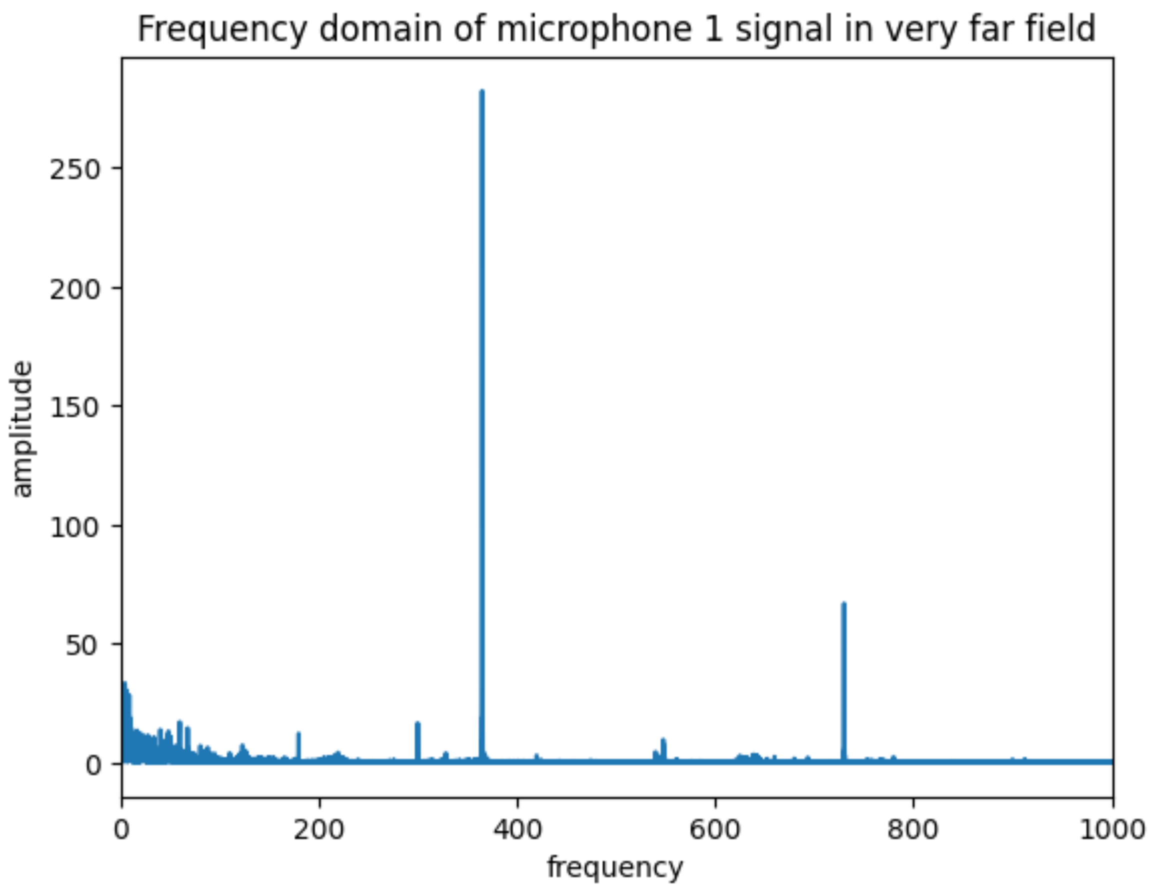
```
6720000 1.0416666666666666e-05
```



Okay, yeah awesome! Tuning fork 1 is definitely a lot clearer than tuning fork 2 when I listen so I am glad it shows in the data too

In [27]:
```
fdata = np.take(data_80_sweep, 1, axis=1)
fdata = fdata - np.average(fdata) # to get rid of the dc offset

N = len(fdata) # sample points
T = 1 / samplerate_80_sweep

print(N, T)

yf = np.abs(fft(fdata)[0:N//2])
xf = fftfreq(N, T)[:N//2]

plt.plot(xf, 2.0/N * yf)
plt.title('Frequency domain of microphone 1 signal in very far field')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xlim([0,1000])
plt.show()
```

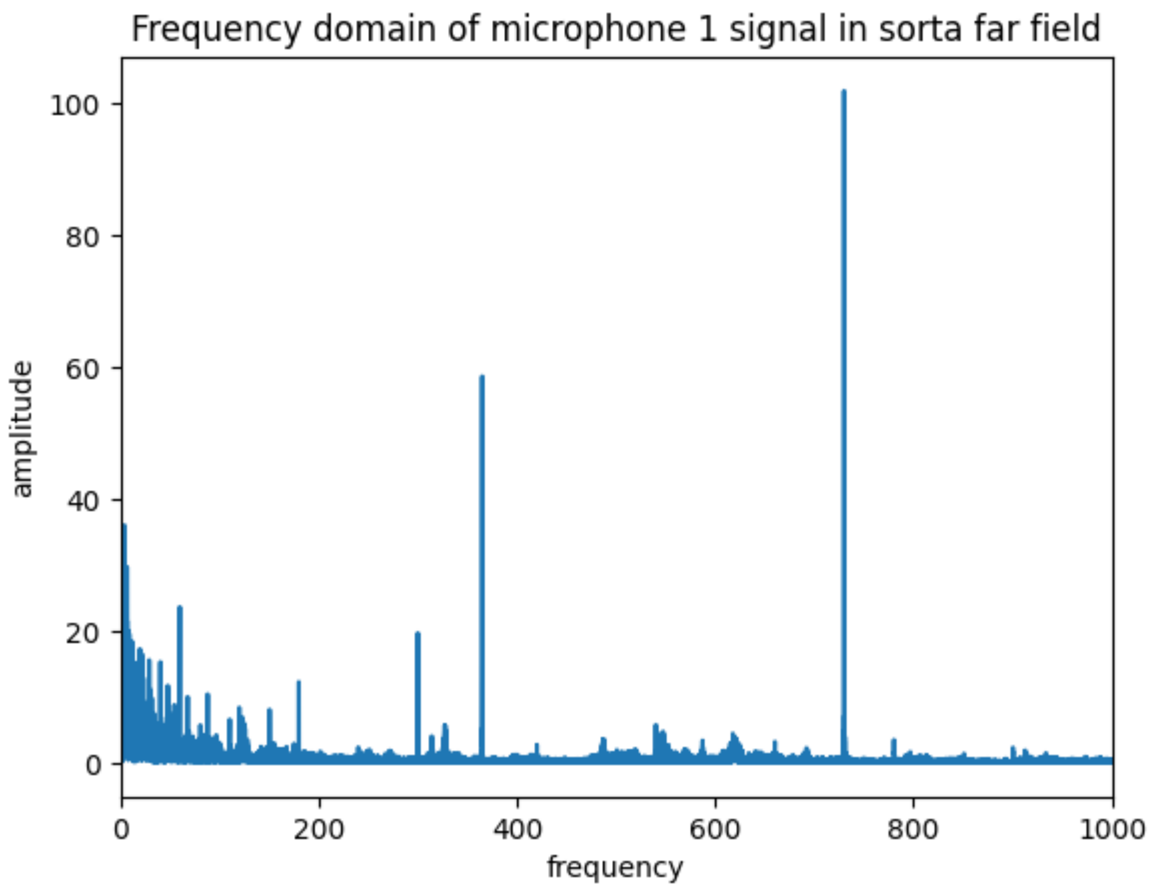
```
5760000 1.0416666666666666e-05
```



Nice, some harmonics it looks like and lots of low frequency noise that I can filter out!

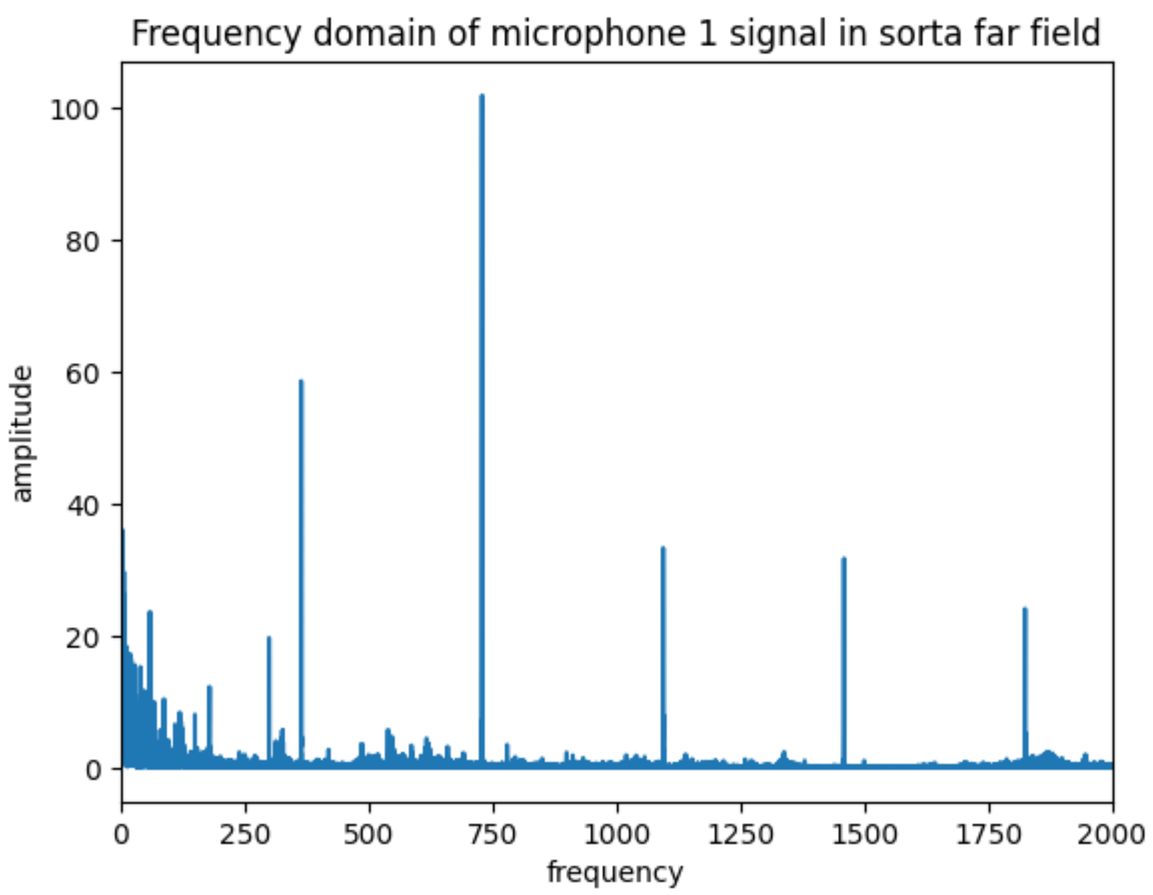
```
In [28]: fdata = np.take(data_30_sweep, 1, axis=1)
         fdata = fdata - np.average(fdata) # to get rid of the dc offset

         N = len(fdata) # sample points
         T = 1 / samplerate_30_sweep

         print(N, T)

         yf = np.abs(fft(fdata)[0:N//2])
         xf = fftfreq(N, T)[:N//2]

         plt.plot(xf, 2.0/N * yf)
         plt.title('Frequency domain of microphone 1 signal in sorta far field')
         plt.xlabel('frequency')
         plt.ylabel('amplitude')
         plt.xlim([0,1000])
         plt.show()
```

```
5760000 1.0416666666666666e-05
```



Hmmmm, that's weird, I got MORE of the harmonic than the fundamental frequency... I will have to think about why that is and ask about this in the lab tomorrow, because I didn't expect this at all.

```
In [29]: plt.plot(xf, 2.0/N * yf)
         plt.title('Frequency domain of microphone 1 signal in sorta far field')
         plt.xlabel('frequency')
         plt.ylabel('amplitude')
         plt.xlim([0,2000])
         plt.show()
```



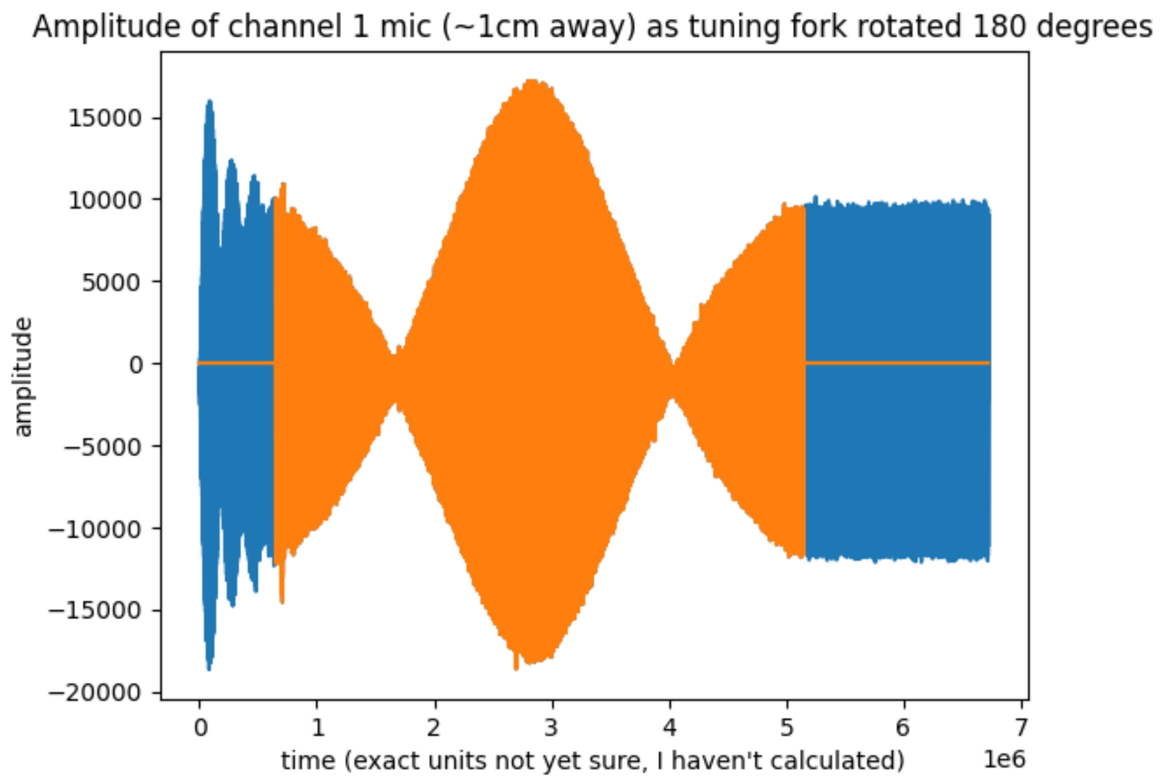
Looking at more frequencies, it looks like all the harmonics are quite strong here.

Radial plot of near field

For now, I will be attempting to get a radial plot using this new measurement technique on the near field data!

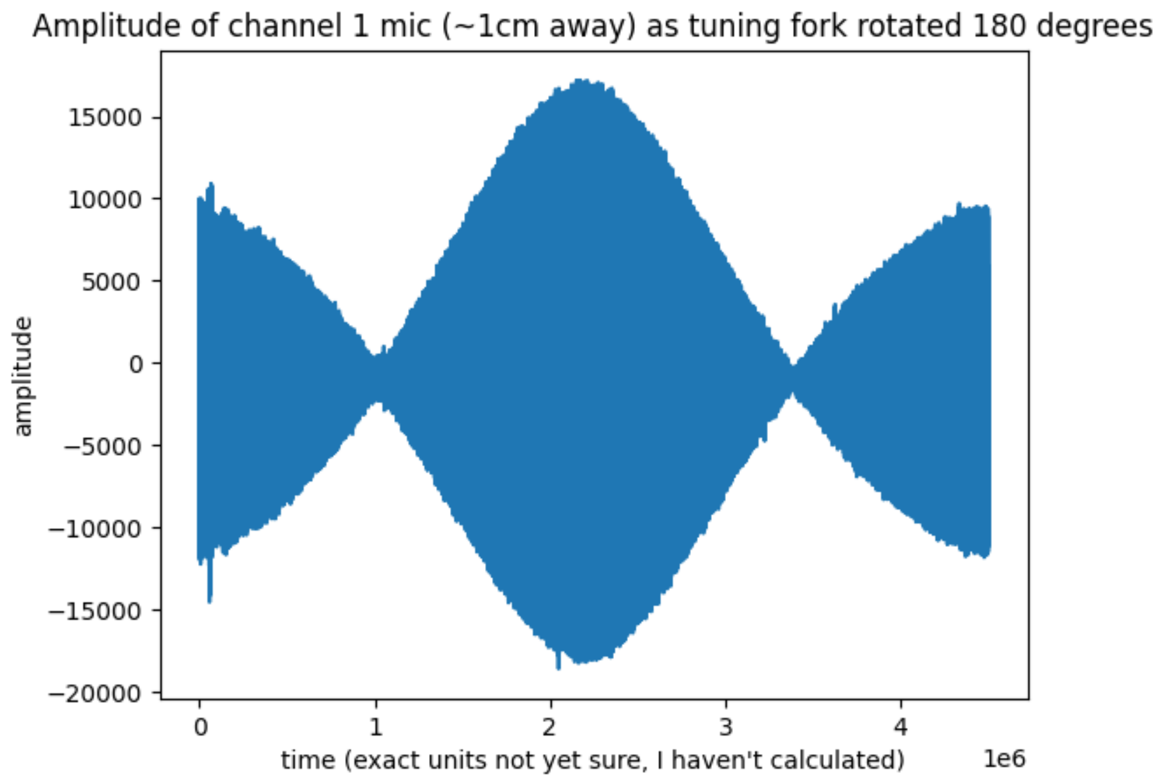
In [21]:
```
# first let's find our data range

data_1_s = np.take(data_1_sweep, 1, axis=1)

d1s_rotation = np.zeros_like(data_1_s) # default to all 0

# getting my range
d1s_rotation[650000:5150000] = data_1_s[650000:5150000]
```

In [22]:
```
plt.title("Amplitude of channel 1 mic (~1cm away) as tuning fork rotated
plt.xlabel("time (exact units not yet sure, I haven't calculated)")
plt.ylabel('amplitude')
plt.plot(data_1_s)

plt.plot(d1s_rotation)
plt.show()
```



Okay, that's a bit hacky, but the orange section looks like when I started rotating the rod, and stopped, so I will go with that.

In [25]:
```
d1s_rotation = d1s_rotation[d1s_rotation != 0]
plt.title("Amplitude of channel 1 mic (~1cm away) as tuning fork rotated
plt.xlabel("time (exact units not yet sure, I haven't calculated)")
plt.ylabel('amplitude')
plt.plot(d1s_rotation)
plt.show()
```



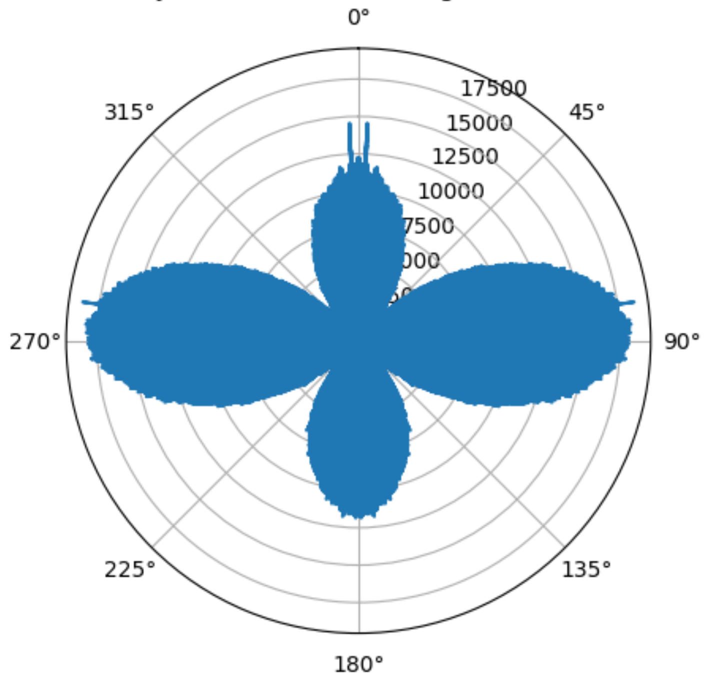
Great, now let's start getting some radial plots.

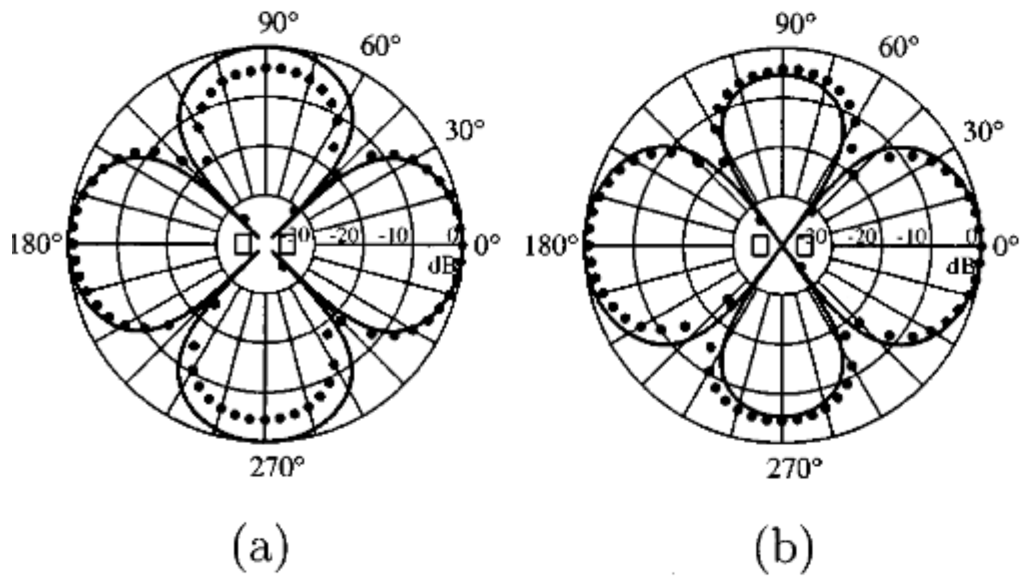
I have realised that time doesn't actually matter because it rotates at a constant speed and we only care about the angle, not the time it took to reach that angle, so I will just divide the current interval into 1800 degrees (I only did 180 degree sweep)

In [26]:
```
N = len(d1s_rotation) # number of samples
angles = np.linspace(0, np.pi, N)

# reflect on x axis for full rotation
angles_full = np.concatenate((angles, angles + np.pi))
d1s_rotation_full = np.concatenate((d1s_rotation, d1s_rotation[::-1]))
```

In [27]:
```
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d1s_rotation_full))

ax.set_theta_direction(-1)  # To make clockwise rotation
ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Radial Plot of Sound Intensity near field (0-180 deg and th

plt.show()
```

Radial Plot of Sound Intensity near field (0-180 deg and then mirrored for 180-360 deg)





Compared to this image from the readings, I think my measurement and the data is reasonable.

Filtering data from far field

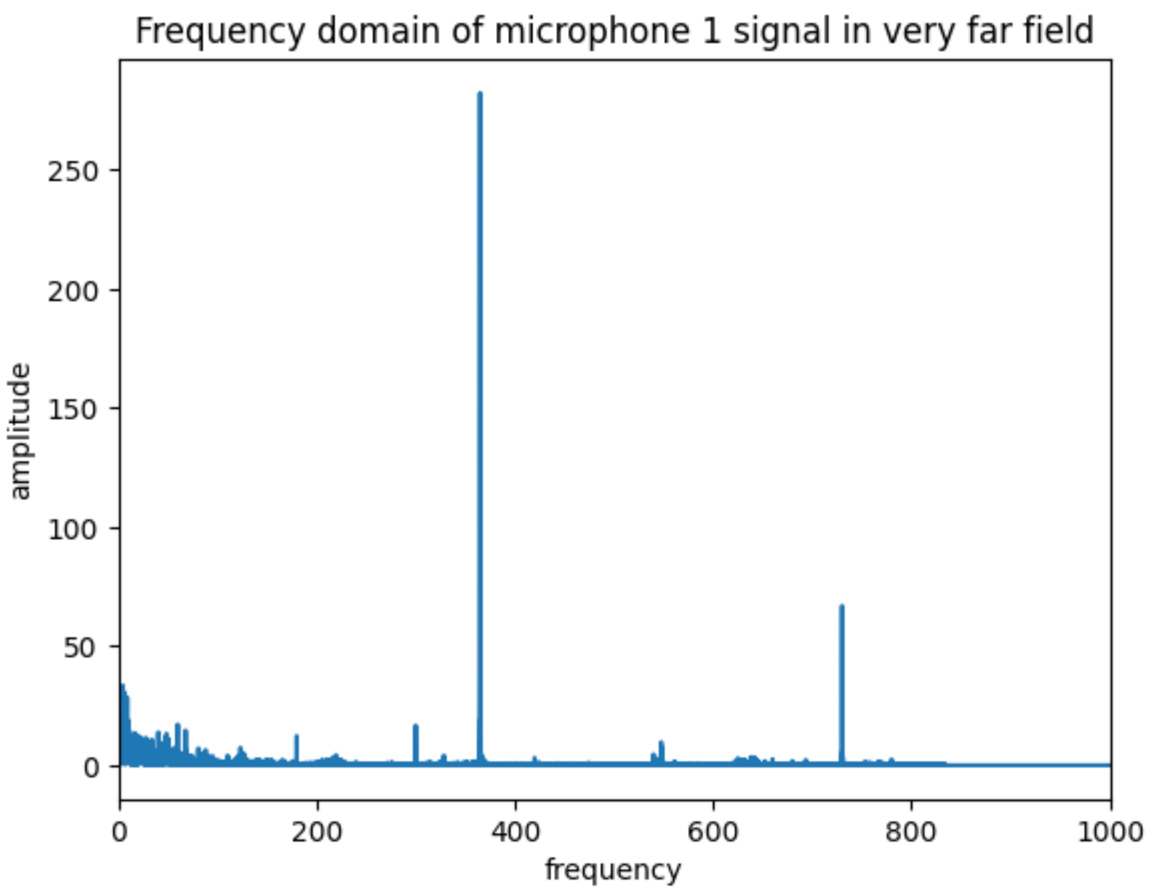
I will now check on my very far field data and try some filtering (high pass filter) and then follow a similar strategy to plot a radial plot.

I only plan to use the 30cm data as a backup i.e., if the 80cm data is completely useless.

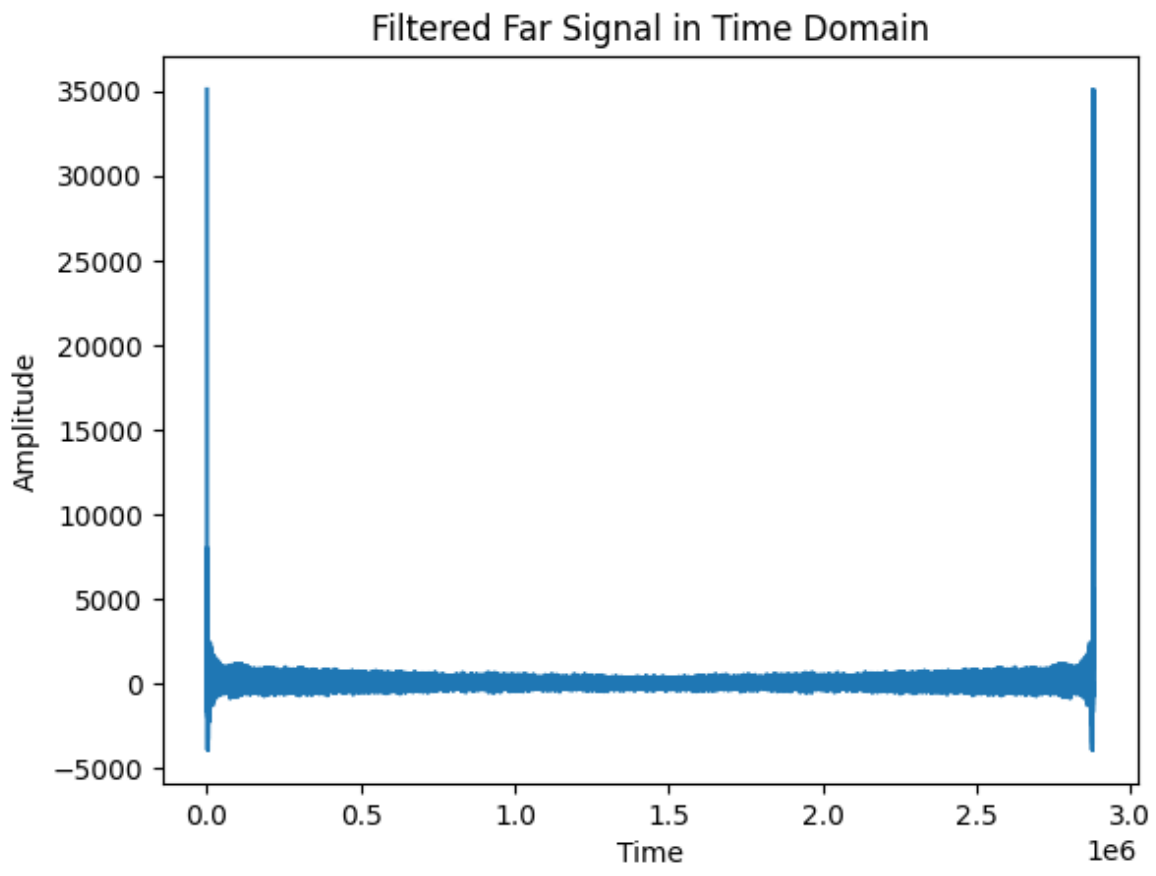
In [5]:
```
fdata = np.take(data_80_sweep, 1, axis=1)
fdata = fdata - np.average(fdata) # to get rid of the dc offset

N = len(fdata) # sample points
T = 1 / samplerate_80_sweep

print(N, T)

yf = np.abs(fft(fdata)[0:N//2])
xf = fftfreq(N, T)[:N//2]

print(len(yf))

yf_f = np.zeros_like(yf)

print(len(yf_f))
yf_f[:50000] = yf[:50000]

print(yf_f[32000])
```

```
5760000 1.0416666666666666e-05
2880000
2880000
146075.3461384756
```

In [6]:
```
plt.plot(xf, 2.0/N * yf_f)
plt.title('Frequency domain of microphone 1 signal in very far field')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xlim([0,1000])
plt.show()
```



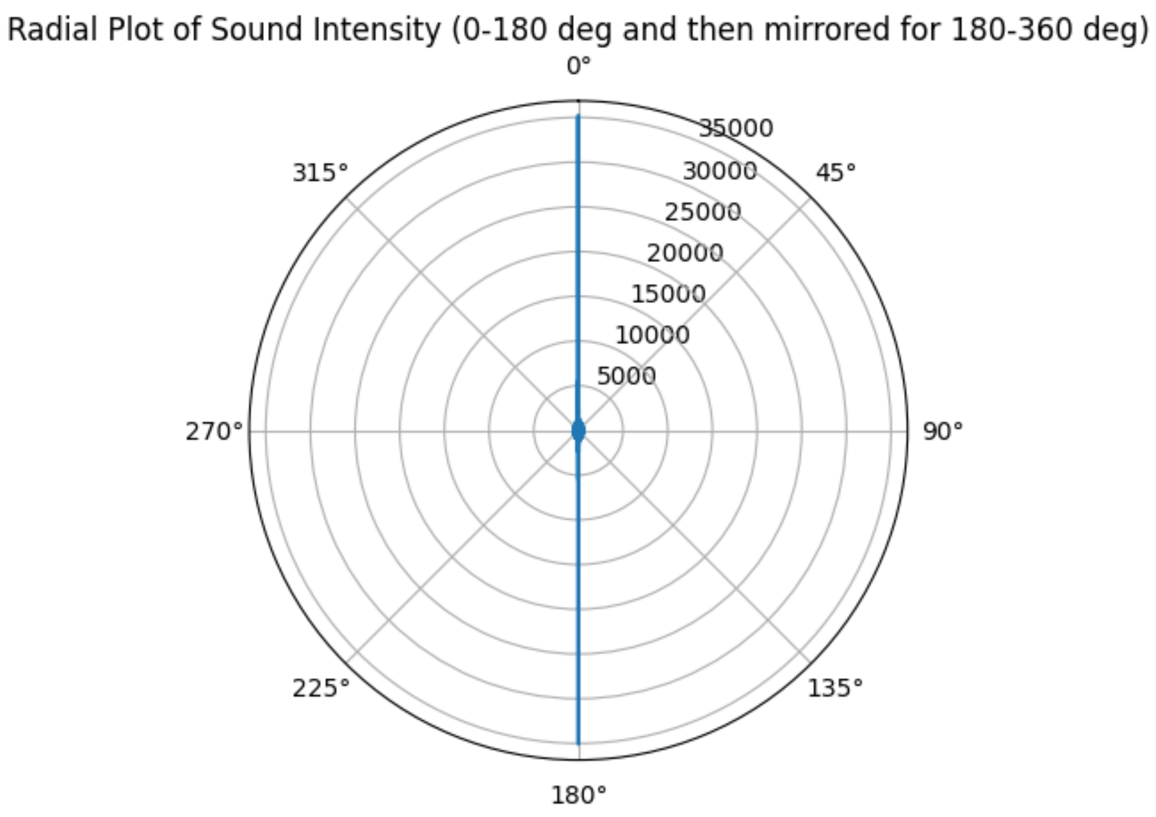
In [7]:
```
filtered_far_data = ifft(yf_f).real
plt.plot(filtered_far_data)
plt.title('Filtered Far Signal in Time Domain')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



Hmmm i'm not sure this is what I'm looking to do...

```
In [8]: N = len(filtered_far_data) # number of samples
        angles = np.linspace(0, np.pi, N)

        # reflect on x axis for full rotation
        angles_full = np.concatenate((angles, angles + np.pi))
        filtered_far_data_full = np.concatenate((filtered_far_data, filtered_far_
```

```
In [9]: fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
        ax.plot(angles_full, abs(filtered_far_data_full))

        ax.set_theta_direction(-1)  # To make clockwise rotation
        ax.set_theta_zero_location("N")  # Set 0 degrees to the top
        ax.set_title("Radial Plot of Sound Intensity (0-180 deg and then mirrored

        plt.show()
```



Maybe I should have tried it without the filter first to check... yea, I guess I can do that now.

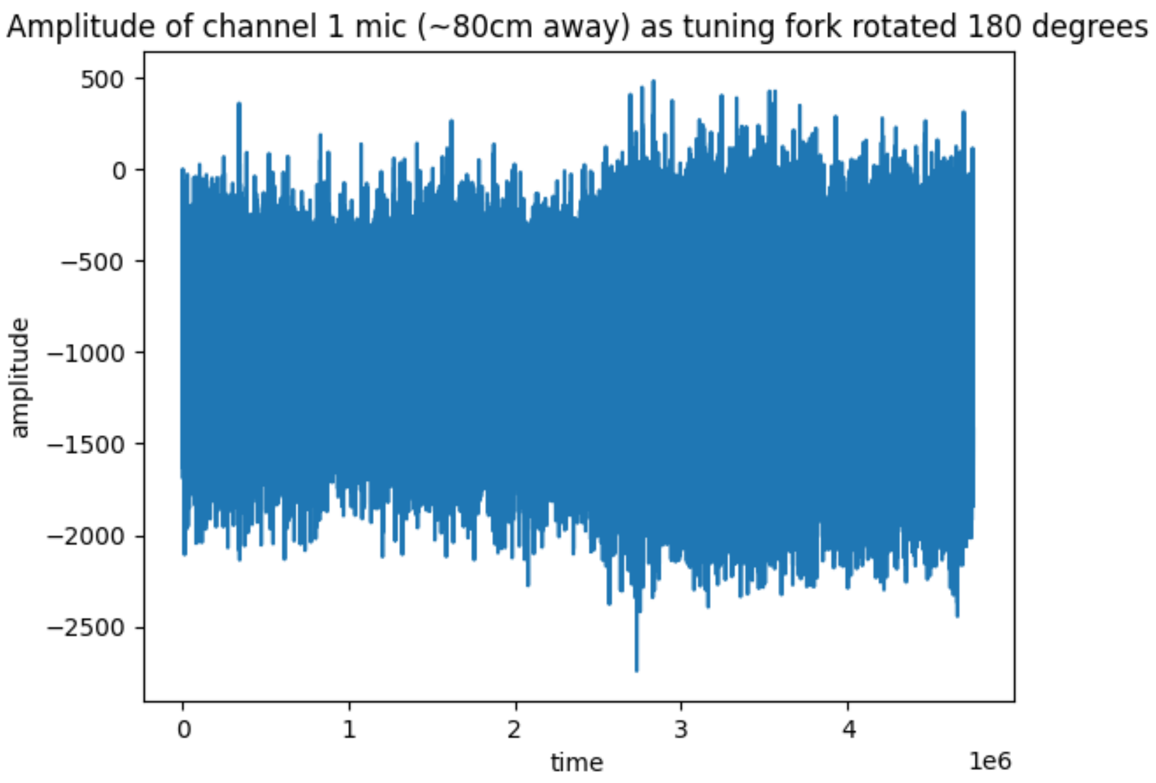
Wait a minute... I also didn't select the portion where the signal is actually being recorded while turning... okay a few things to do first before plotting, I think I jumped the gun.

Great, I now have timing! It rotates between 3.19s and 52.75s! Now I can chop and it should be good.

```
In [40]: filename = '../data/180sweep-far-timing.wav'
         sr_80g, data_80g = wavfile.read(filename)

         start_time = 3.19
         end_time = 52.75

         start_index = int(start_time * sr_80g)
         end_index = int(end_time * sr_80g)

         data_80g = data_80g[start_index:end_index]

         print(data_80g.shape)
         print ("Frequency sampling:", sr_80g)

         plt.title("Amplitude of channel 1 mic (~80cm away) as tuning fork rotated
         plt.xlabel("time")
         plt.ylabel('amplitude')
         plt.plot(np.take(data_80g, 1, axis=1))
```

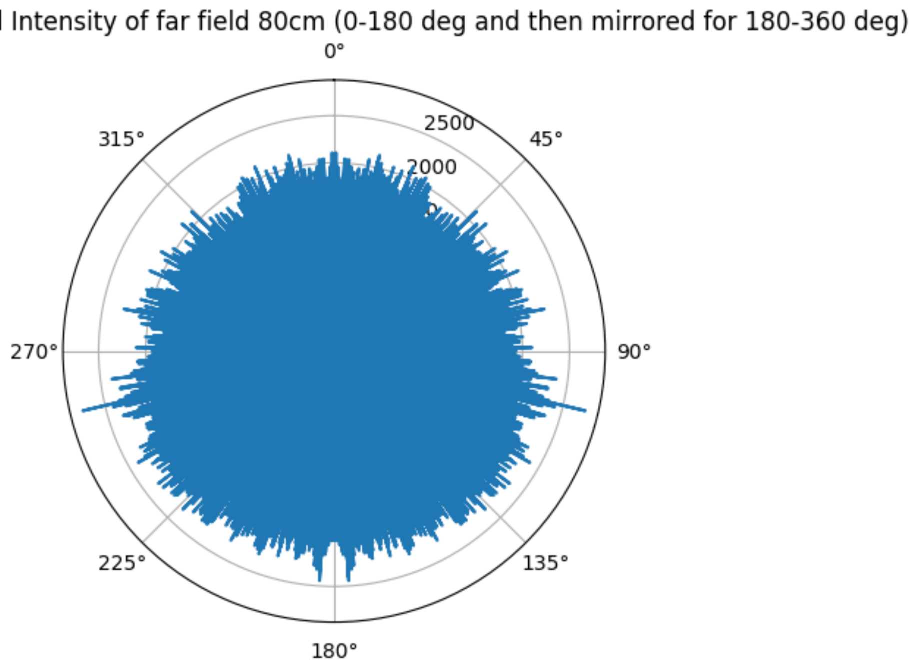
```
(4757760, 2)
Frequency sampling: 96000
```

```
Out[40]: [<matplotlib.lines.Line2D at 0x7ea40d3a96d0>]
```



In [41]:
```
d80g = np.take(data_80g, 1, axis=1)
N = len(d80g) # number of samples
angles = np.linspace(0, np.pi, N)

# reflect on x axis for full rotation
angles_full = np.concatenate((angles, angles + np.pi))
d80_rfull = np.concatenate((d80g, d80g[::-1]))
```

In [42]:
```
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d80_rfull))

ax.set_theta_direction(-1)  # To make clockwise rotation
ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Radial Plot of Sound Intensity of far field 80cm (0-180 deg

plt.show()
```

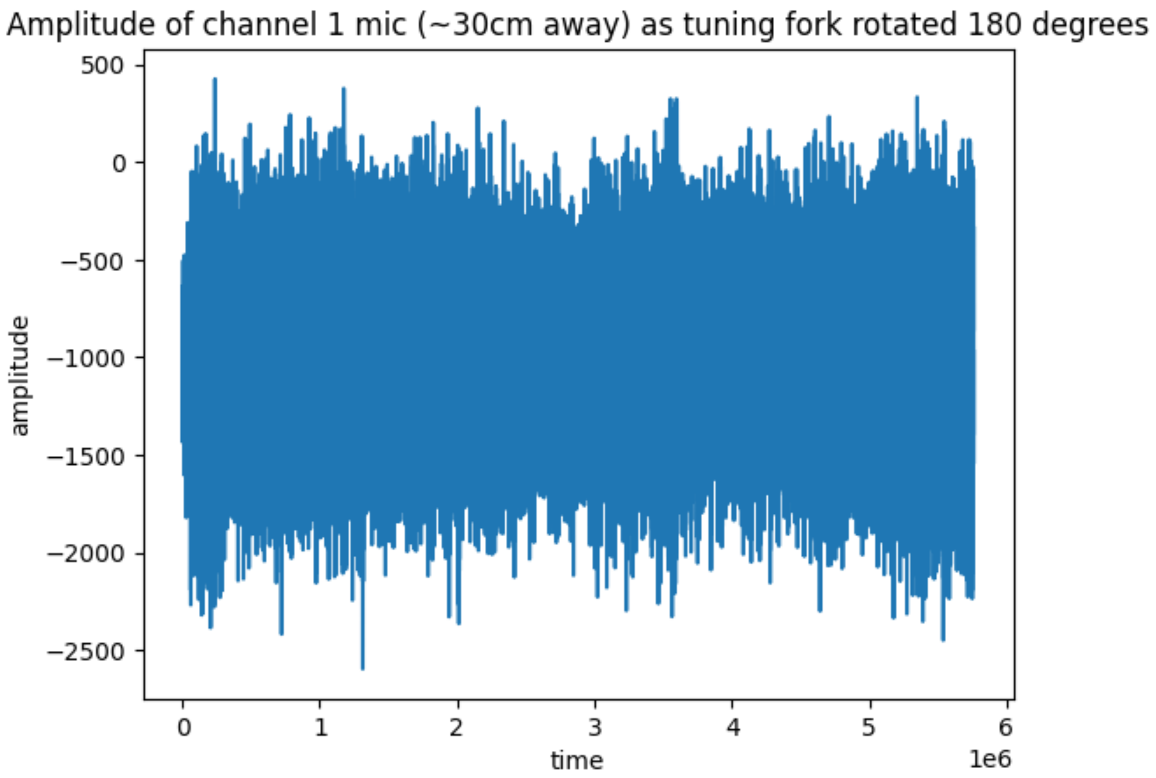


My best guess is the whirr of the motor was simply too loud, so I'll try the 30cm one.

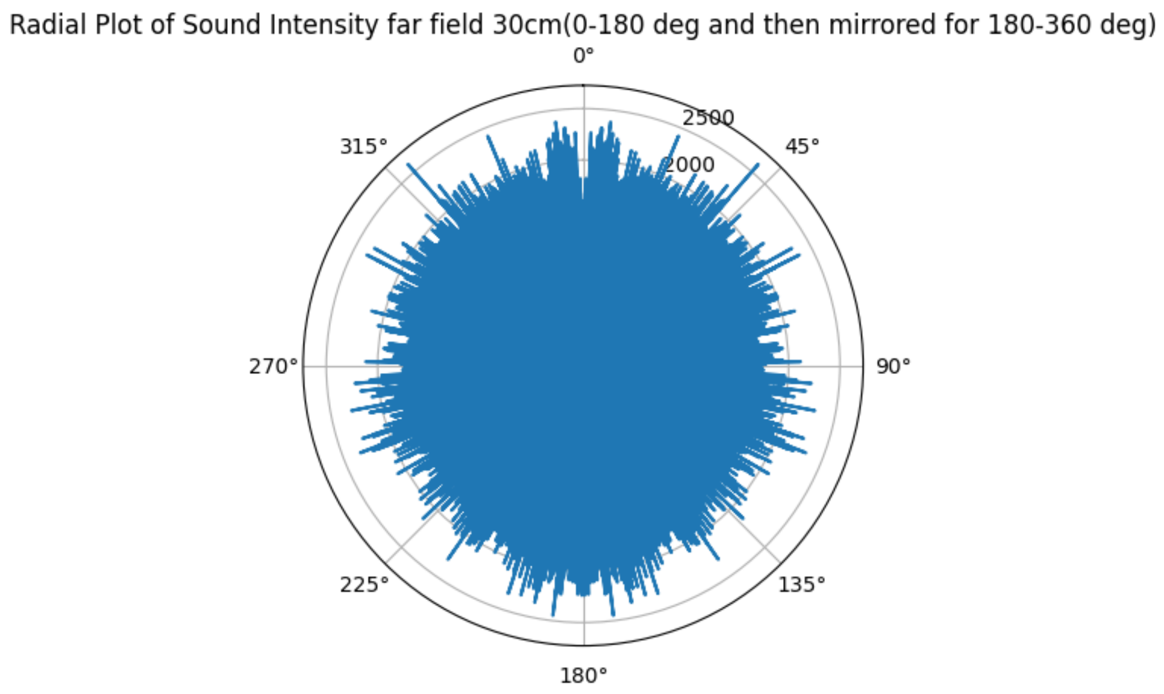
```
In [43]: filename = '../data/180sweep-m1-far30.wav'
         sr_30, d30 = wavfile.read(filename)

         print(d30.shape)
         print ("Frequency sampling:", sr_30)

         plt.title("Amplitude of channel 1 mic (~30cm away) as tuning fork rotated
         plt.xlabel("time")
         plt.ylabel('amplitude')
         plt.plot(np.take(d30, 1, axis=1))
```

```
(5760000, 2)
Frequency sampling: 96000
```

```
Out[43]: [<matplotlib.lines.Line2D at 0x7ea40d270a70>]
```



In [44]:
```
d30g = np.take(d30, 1, axis=1)
N = len(d30g) # number of samples
angles = np.linspace(0, np.pi, N)

# reflect on x axis for full rotation
angles_full = np.concatenate((angles, angles + np.pi))
d80_rfull = np.concatenate((d30g, d30g[::-1]))

fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d80_rfull))

ax.set_theta_direction(-1)  # To make clockwise rotation
ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Radial Plot of Sound Intensity far field 30cm(0-180 deg and

plt.show()
```



Summary of recent attempts

Unfortunately my first and second attempts didn't turn out as I expected, and neither did the closer attempt at the far field. It is probably that the signal was too weak, and with some filtering it could be extracted from the data, but for now I will take a break and maybe with some more thinking later I can do some checking. Otherwise I will looking at some results and analysis and possible conclusions I might be able to draw, or compare the data to a theoretical plot.

Oct 3, 15:52

Graphs from above for easy reference:

Figure 1 (below): Frequency domain of microphone 1

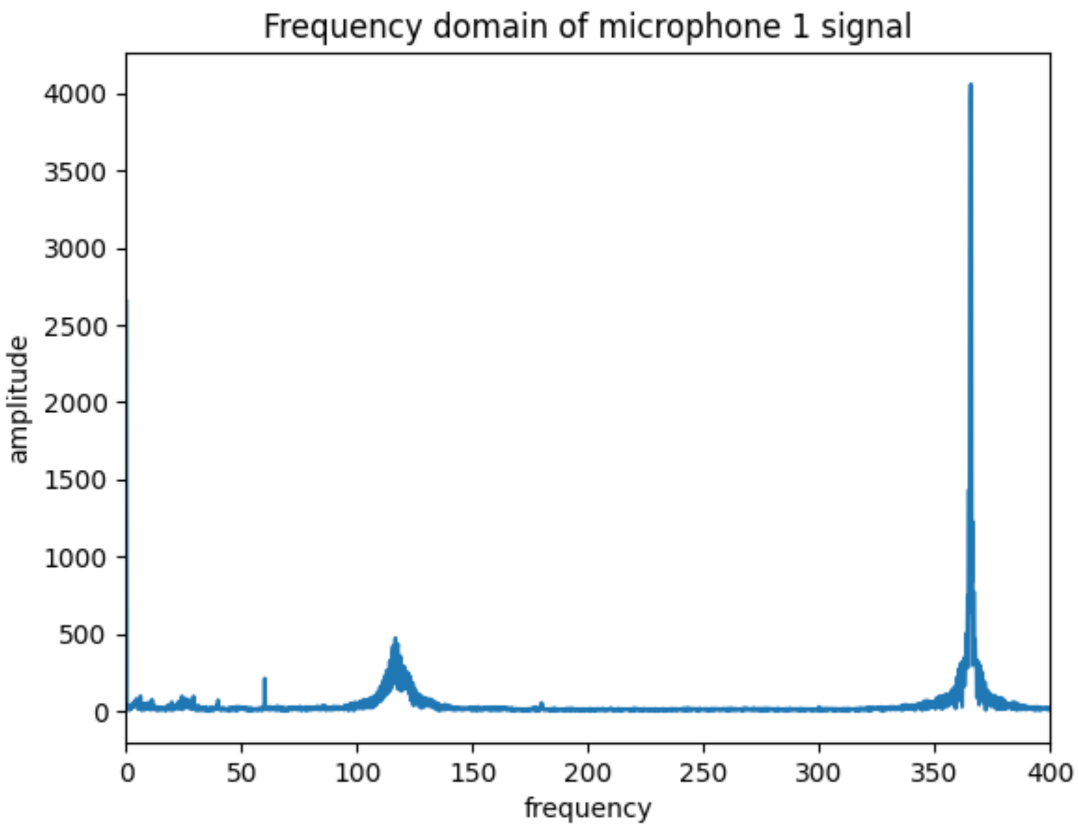


Figure 2 (below): Microphone data from driving the tuning fork at resonant frequency, still at 0-deg

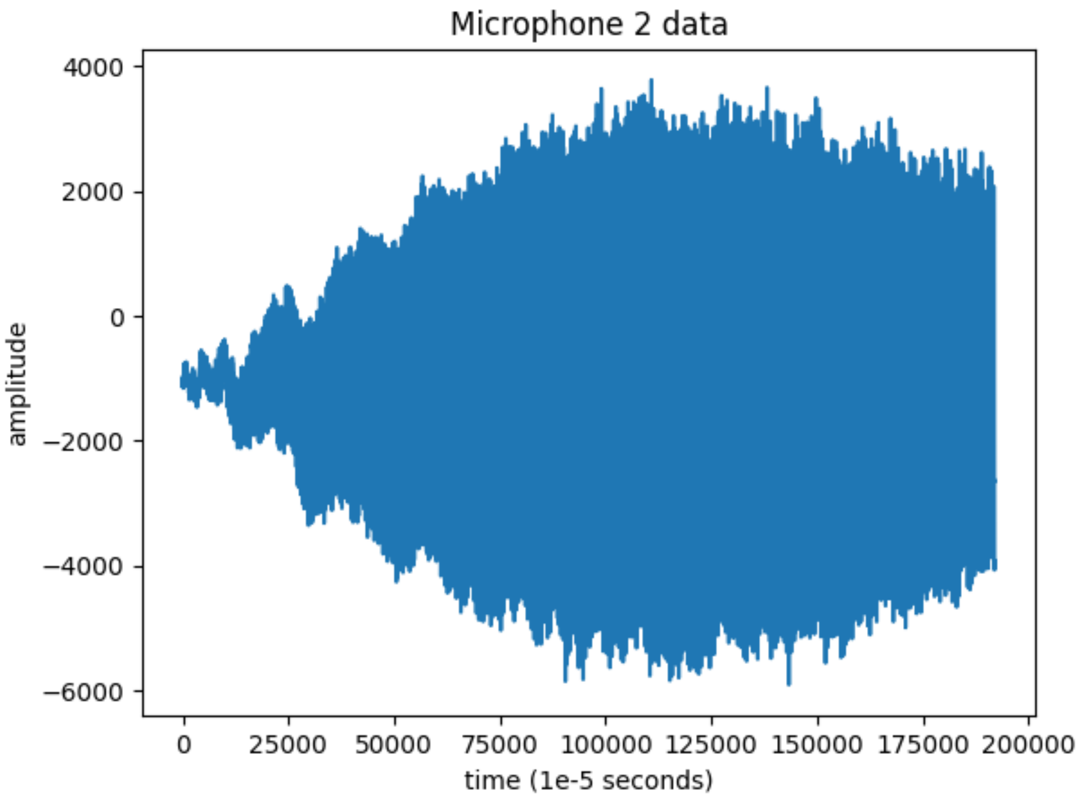


Figure 3 (below): First attempt at radial plot of the near field, using steps:

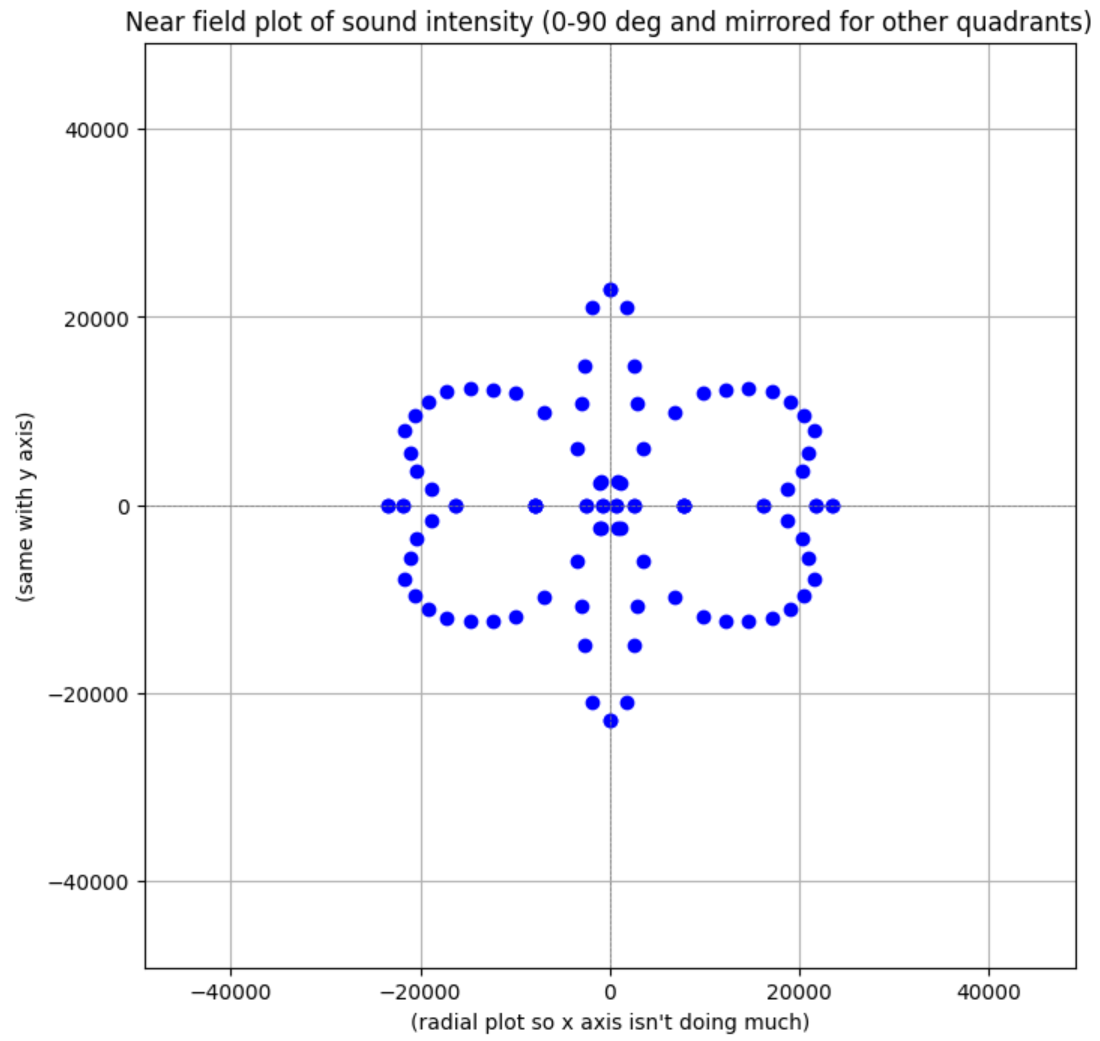


Figure 4 (below): First attempt at radial plot of the far field, using steps:

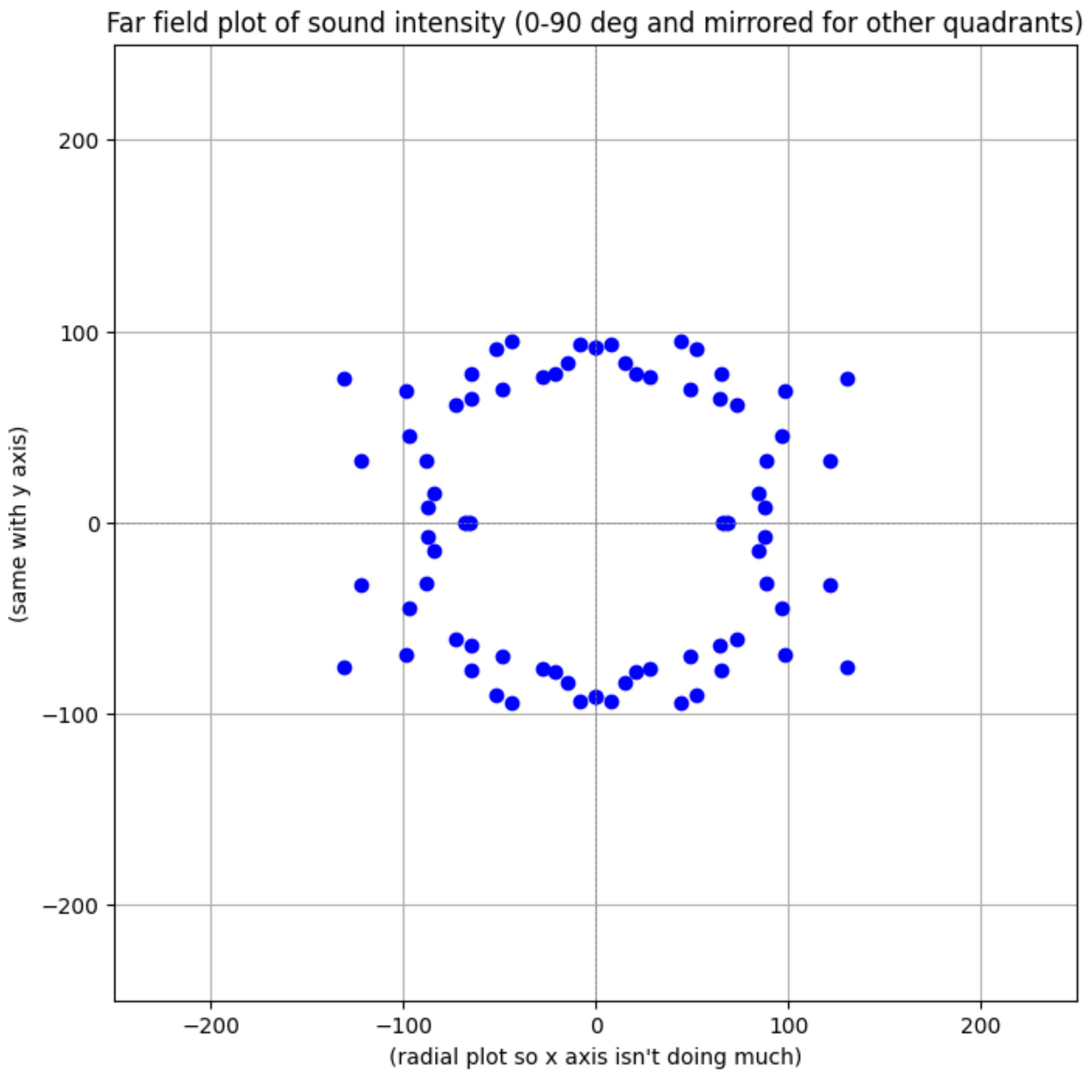


Figure 5 (below): Microphone data from driving the tuning fork while doing a continuous rotation sweep in the near field:

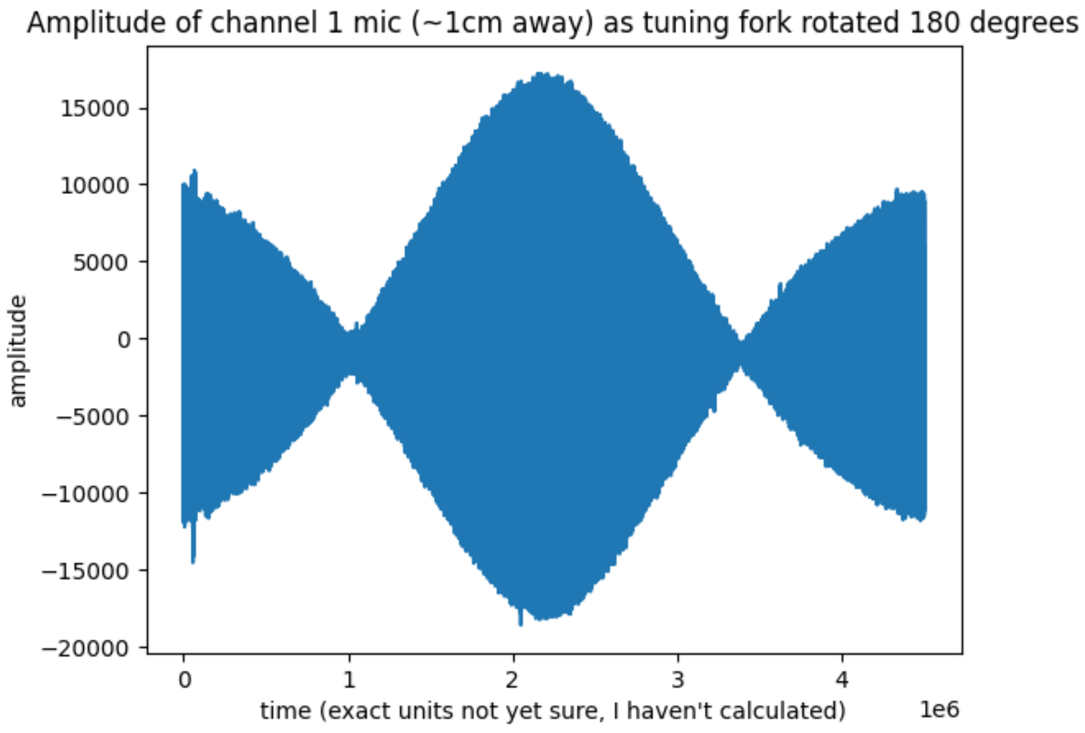


Figure 6 (below): Second attempt at radial plot of the near field, using a continuous sweep:

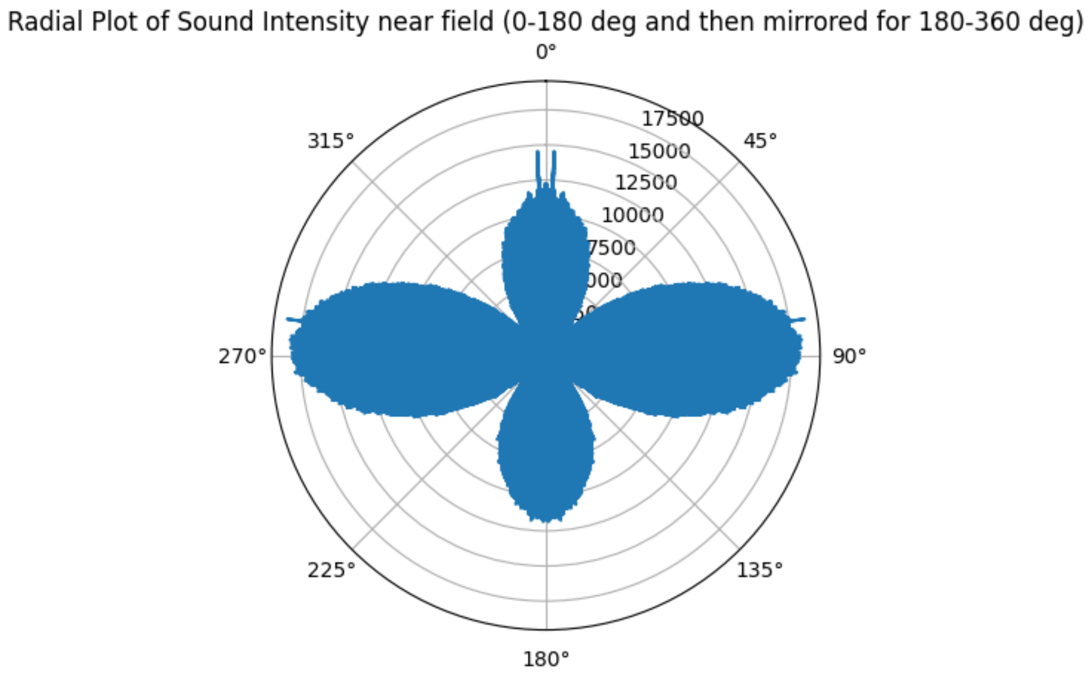


Figure 7 (below): Microphone data from driving the tuning fork while doing a continuous rotation sweep in the far field (80cm):

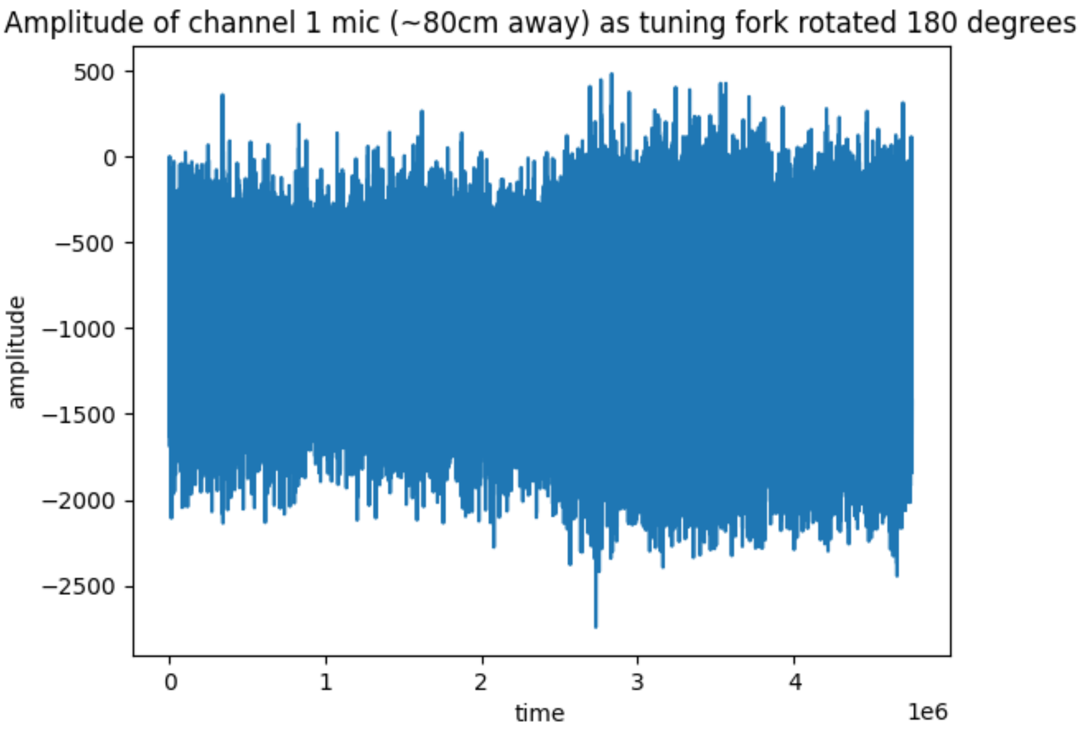


Figure 8 (below): Second attempt at radial plot of the far field, using a continuous sweep:

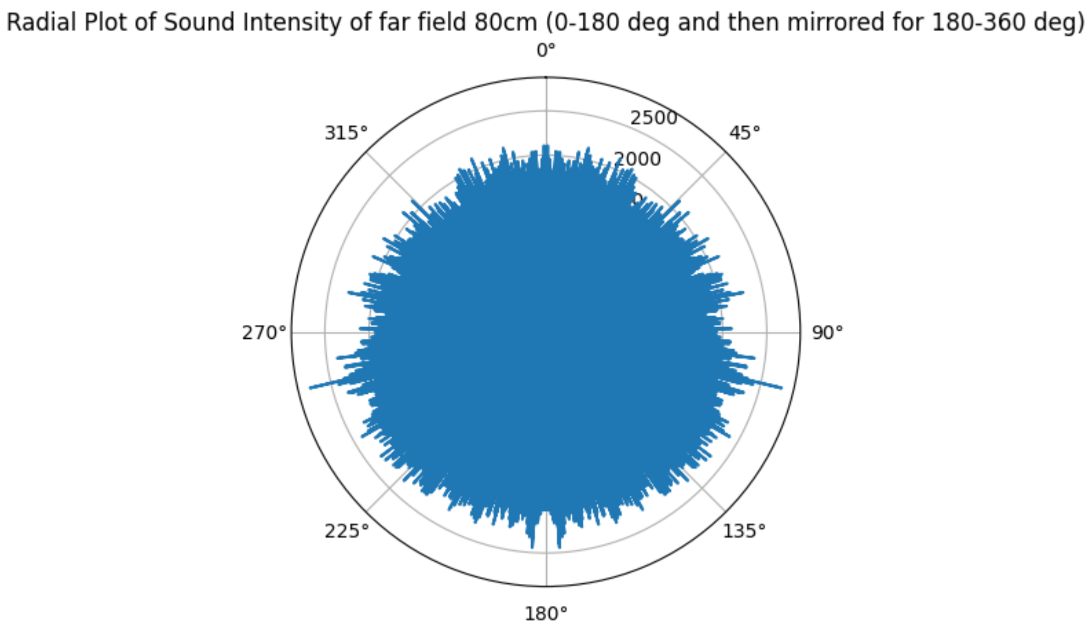


Figure 9 (below): Microphone data from driving the tuning fork while doing a continuous rotation sweep in the far field (80cm):

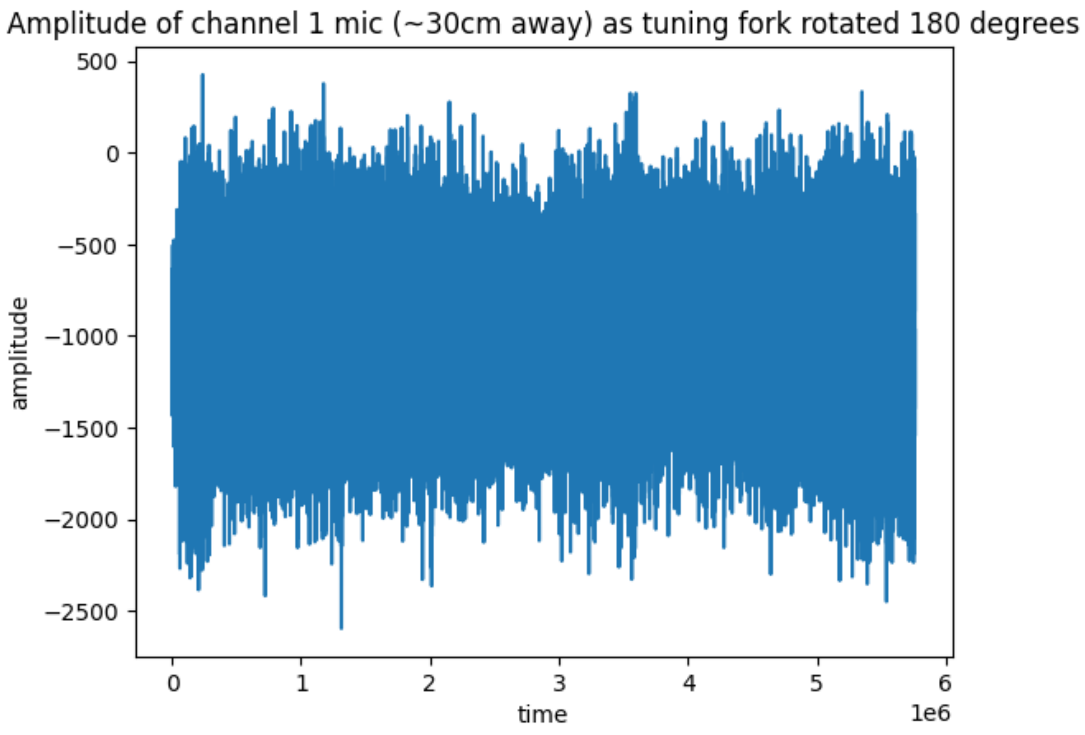
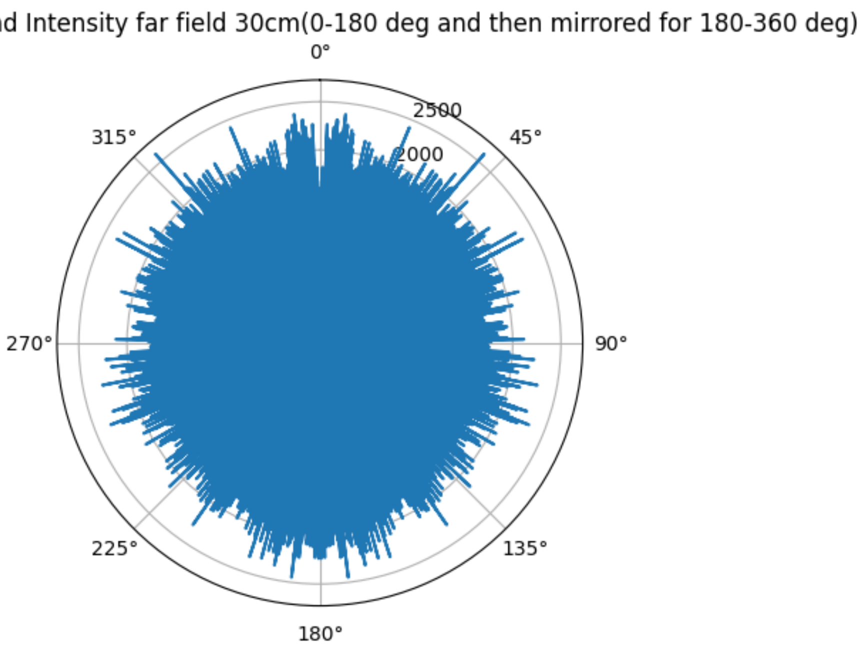


Figure 10 (below): Third attempt at a radial plot of the far field, using a continuous sweep (30cm):



The three measurements I took were at distances:

- 1 cm
- 30 cm
- 80 cm

Using the speed of sound as 343 m/s, and frequency of 365 Hz, then we get a k value of 2*pi*\365/343=6.68618844647

so for each of the three measurements we get kr values of:

- $kr_1 = 0.0669$
- $kr_2 = 2.0059$
- $kr_3 = 5.3490$

Now for plotting the theory. The lab manual mentions an L_p but the literature makes no mention of this, so I will just interpret it as a normalization constant out front.

$$p(r,\theta) = \frac{A}{r}\left[(1-3\cos^2\theta)\left(\frac{ik}{r} - \frac{1}{r^2} + \frac{k^2}{3}\right) - \frac{k^2}{3}\right].$$

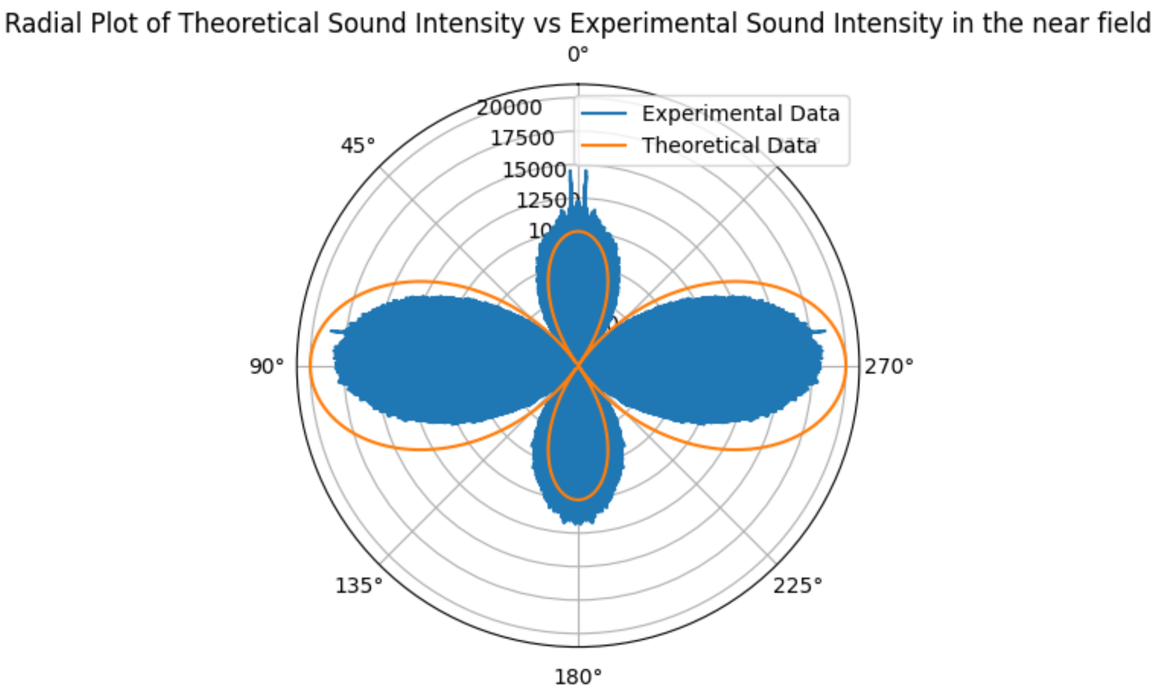
```
In [13]: def near_field_magnitude(theta, k, r, A):
             cos_theta = np.cos(theta)
             term_1 = (1 - 3 * cos_theta**2)
             term_2 = (1j * k / r) - (1 / r**2) + (k**2 / 3)
             result = (A / r) * (term_1 * term_2 - (k**2 / 3))

             return result

         def far_field_approximate(theta, k, r, A):
             cos_theta = np.cos(theta)
             result = (A * k**2 / r) * cos_theta**2
```

```
    return result
```

In [3]:
```
filename = '../data/180sweep-m1.wav'
samplerate_1_sweep, data_1_sweep = wavfile.read(filename)

data_1_s = np.take(data_1_sweep, 1, axis=1)

d1s_rotation = np.zeros_like(data_1_s) # default to all 0

# getting my range
d1s_rotation[650000:5150000] = data_1_s[650000:5150000]
d1s_rotation = d1s_rotation[d1s_rotation != 0]

N = len(d1s_rotation) # number of samples
angles = np.linspace(0, np.pi, N)

# reflect on x axis for full rotation
angles_full = np.concatenate((angles, angles + np.pi))
d1s_rotation_full = np.concatenate((d1s_rotation, d1s_rotation[::-1]))

# save my computer's memory
del samplerate_1_sweep, data_1_sweep, data_1_s, d1s_rotation, angles
```

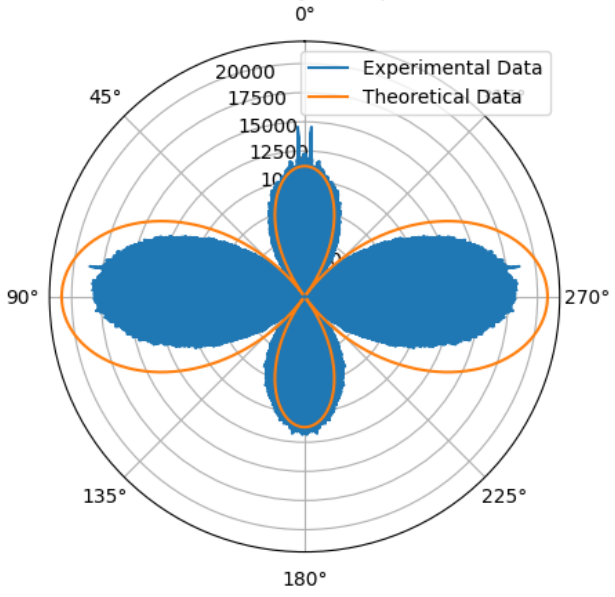
In [19]:
```
theta = np.linspace(0,2*np.pi,int(np.pi*100))

near_field_theory = near_field_magnitude(theta, 6.686, 0.01, 1e-2)
```

In [20]:
```
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d1s_rotation_full), label='Experimental Data')
ax.plot(theta+0.5*np.pi, abs(near_field_theory), label='Theoretical Data'

ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Radial Plot of Theoretical Sound Intensity vs Experimental

ax.legend(loc='upper right')

plt.show()
```



In [37]:
```
near_field_theory = near_field_magnitude(theta, 8, 0.05, 1.3)
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d1s_rotation_full), label='Experimental Data')
ax.plot(theta+0.5*np.pi, abs(near_field_theory), label='Theoretical Data'

ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Idealized Radial Plot of Theoretical Sound Intensity vs " +
             "Experimental Sound Intensity in the near field")
ax.legend(loc='upper right')

plt.show()
```

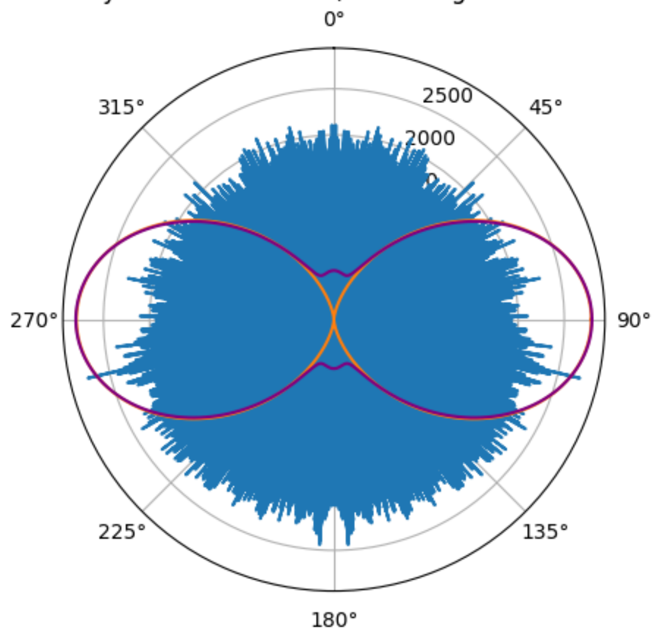
Idealized Radial Plot of Theoretical Sound Intensity vs Experimental Sound Intensity in the near field



In [5]:
```
filename = '../data/180sweep-far-timing.wav'
sr_80g, data_80g = wavfile.read(filename)

start_time = 3.19
end_time = 52.75

start_index = int(start_time * sr_80g)
end_index = int(end_time * sr_80g)

data_80g = data_80g[start_index:end_index]

d80g = np.take(data_80g, 1, axis=1)
N = len(d80g) # number of samples
angles = np.linspace(0, np.pi, N)

# reflect on x axis for full rotation
angles_full = np.concatenate((angles, angles + np.pi))
d80_rfull = np.concatenate((d80g, d80g[::-1]))

# save memory - my computer crashed multiple times :'(
del angles, sr_80g, data_80g, d80g
theta = np.linspace(0,2*np.pi,int(np.pi*100))
```

In [15]:
```
far_field_theory = far_field_approximate(theta, 6.686, 0.8, 50)
far_field_exact = near_field_magnitude(theta, 6.686, 0.8, 50)
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(angles_full, abs(d80_rfull), label='Experimental Data')
ax.plot(theta+0.5*np.pi, abs(far_field_theory), label='Theoretical Approx
ax.plot(theta+0.5*np.pi, abs(far_field_exact), color='purple', label='The

ax.set_theta_direction(-1)  # To make clockwise rotation
ax.set_theta_zero_location("N")  # Set 0 degrees to the top
ax.set_title("Radial Plot of Sound Intensity of far field 80cm (0-180 deg

plt.show()
```

Radial Plot of Sound Intensity of far field 80cm (0-180 deg and then mirrored for 180-360 deg)



Oct 4, 8:02

Topics to cover:

- Summary of experiment
- Uncertainty in the experiment (sources and suspected effects)
- Conclusion/Results (i.e., trends) and justifications

Summary of experiment

- I first found the resonant frequency of the tuning fork
  - I did this by lightly striking the tuning fork while recording
  - Then I transformed the data to the frequency domain and found the largest peak (omitting dc offset 0 Hz)
- Next, I drove the fork at this frequency while the fork was still at angles in increments of 5 degrees
  - I did this for both the near field (1cm away) and the far field (80cm away)
  - After doing a 90 deg rotation, I fourier transformed the data and took the magnitude of the resonant frequency of each recording
- This led to my first set of plots with points
  - I don't exactly know why but I think that this method was misguided even though it had proper intent

    - The reason I think it was wrong is because the shape of the plotted data didn't match what I expected and the minimum was off by ~25 degrees
    - (Again, this is likely wrong), but I think since I was driving it at the resonant frequency, each angle would still be reading it since it recorded for 2-3 seconds
    - Additionally since the resonant frequency is not exaclty an integer this might have led to beats and over a 2 second window this became loud enough for the mic
    - While the near field showed some sort of pattern, the far field seemed to have jumbled data and didn't seem to have a clear trend
- New attempt at gathering data, continuous sweep (I decided to do this after talking with Liza in the other lab and the TA)
    - Similar to the first method, I drove the fork at its resonant frequency, but this time I used the motor command to give it a continuous turn and recorded as it swept over 180 degrees
    - I did this for near field (1cm away), far field (80cm away) and far-but-not-that-far field (30cm away), and I know these are in the near or far field by calculating kr and for $kr < 1$ it is the near field, while $kr > 1$ it is the far field
    - Looking at the near field data in the time domain, it was immediately clear to me that peaks and valleys corresponded to the angles of destructive and constructive interference so I plotted the time series data as the different angles (because in addition to being time on the axis, it was also angle)
- Second set of radial plots
    - The near field data clearly displayed the physics I was trying to capture of the near field
    - The far field data again didn't really have that clear a trend as I was hoping to observe
    - I tried to counteract this by applying a filter to the far field data
    - I tried a low pass, a high pass and a bandpass, and none of them worked
    - I only kept one in my notebook because it made a lot more sense for me to edit 2 lines of code and plot again instead of making more jumbled plots... so you'll just have to take my word that they weren't that clear
- Lastly I plotted the theoretical data against the experimental data
    - The near field looks quite similar, some of the peaks/valleys are not quite the same magnitude as the theory, but angles for which the peaks and valleys should appear is essentially spot on
    - The far field (both 80cm and 30cm) did not seem to have any resemblance to either the exact theory, or the approximation
- Unfortunately at this point I had collected data many times and put more time into this experiment than my capstone project for the week, so I decided I would stop there and do my best to analyse why it might have happened and what I could improve next time (should I do this again in the future)
    - Some of these reasons are addressed here, some will be in the uncertainties section (next)

Uncertainty in the experiment

- There was lots of uncertainty in this experiment in both the dependent variable (amplitude) and independent variable (angle), some measurable, some unmeasurable
- Uncertainty in both methods:
  - The uncertainty in the measured amplitude from the microphone is unknown
    - It is a digital measurement to a precision of unit digits so I'm pretty sure that means there is uncertainty of $\pm frac12\sqrt{3}$
  - Random noises were occuring in the lab around me
    - I didn't record at the same time as my partner, but other sounds that were outside of my hearing range could have been disrupting the signal
    - This likely led to error in the far field since I didn't get a clear signal of the tuning fork
    - One disturbance was the DC offset which I could easily filter out, but other disturbances were not so obvious
  - The fundamental resonant frequency was rounded to the nearest integer
    - This resulted in "beats" which made the sound seem to oscillate loud and quiet, I did not know how to measure this
- Uncertainty in the discreet stepping of angles method:
  - The stepper motor sometimes skipped steps, so it thought it had moved 5 degrees, but had only moved 3.5 degrees for example
  - To get around this I turned off the stepper motor, rotated the tuning fork and used a protractor and measured the angle by hand
    - This gives some uncertainty in the angular position of about ± 0.5 degrees
  - After setting it, I turned the stepper motor back on, which caused a slight movement based on the electric field generated in the coils on start up (aka mechanical backlash because there is slight room between the gears)
    - This probably led to uncertainty in the angular position of around ± 0.1 degrees
  - The placement of the microphone was also done by hand using a the ruler on the table
    - This led to an uncertainty in the radial distance for a given ponit of ± 0.5 mm
- Uncertainty in the continuous sweeping of angles method:
  - Again, the motor sometimes skips steps, so (this is pretty basic) I watched as closely as I could to see if any skips occurred and I restarted when they did
  - I recorded using a stopwatch that clicked by hand start and stop to mark when the tuning fork passed 0 degrees and 180 degrees
    - With ± 0.5 seconds of uncertainty on either end, this likely led to ± 1 second of uncertainty in the range of angles I used

There were probably some other things that I decided in the moment and kept track of in the back of my mind, but I deem anything not noted here to be an insignificant amount of error.

Results

Mis-match between methods in near field

Something that most of the uncertainty in the independent variable (the angle) is a small amount of error but the results look odd. I mean small as in, even in the worst possible case, there is no way for two points (say 20 degrees, and 25 degrees) to have overlap leading to an error more than a data point away. However in comparing the near field radial plots in Figure 3 and Figure 6, the discreetly sampled points show a minima at ~70 degrees, while the continuous sweep shows it at ~50 degrees.

This error of 20 degrees doesn't make sense to me and after talking to David in the lab, I think there must be some systematic error that I can't deduce (or maybe it was in my method of taking the amplitude of the resonant frequency in the frequency domain), or there is some significant amount of noise that changed the error.

Near field sweep data follows theory

The plots comparing the near field experimental data with the theoretical data show a pretty clear correlation. Specifically they have maxima and minima at the same angles. Additionally the shape of the "leaves" (or "tear drops") are similar, appear to have similar curvature, though not quite scaled the same. Part of this difference comes from some uncertainty in the computed k value, as well as the measured radial distance r, and the scale factor that I tried different orders of magnitude until I got a decent fit.

I played around with the variables a little to get some different widths and lengths of the "leaves", but the initial fit is pretty good to begin with anyway.

I think this makes sense because the microphone was very close so I would expect the tuning fork's signal to be the dominant signal.

Far field data is a bad fit using both methods

I don't really know why this happened 😭, but summarizing above the likely factors are:

- My approach to measuring the amplitude went wrong somewhere along the way (i.e., the thing I thought I was measuring was not actually what I needed)
- There was significant noise that messed up the reading and it would have required more detailed filtering