Ph21 Problem Set 2

Thomas Alford

February 5, 2019

Mathematical Introduction- Part I

1

If we set $h(x) = \delta(x - x_0)$, we have $h(x) = \sum_{k=-\infty}^{\infty} \frac{1}{L} e^{-2\pi i x f_k} \left(\int_0^L \delta\left(x - x_0\right) e^{2\pi i x f_k}\, dx \right)$.

Using the sifting theorem for the delta function, the right side is equivalent to

$$\sum_{k=-\infty}^{\infty} \frac{1}{L} e^{2\pi i x_0 f_k} e^{-2\pi i x f_k}$$

.

$$= \sum_{k=-\infty}^{\infty} \frac{1}{L} e^{2\pi i x_0 f_k} e^{-2\pi i x f_k}$$

$$= \sum_{k=-\infty}^{\infty} \frac{1}{L} e^{-2\pi i (x - x_0) f_k}$$

Now, we'll assume that each partition is small enough that we can evaluate this sum as an integral, and so we obtain

$$\frac{1}{L} \int_{-\infty}^{\infty} e^{-\frac{2\pi i k (x - x_0)}{L}}\, dk$$

Now, we have the formula $\delta(x - a) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ip(x-a)}\, dp$. Using this, we obtain

$$= \frac{1}{L} (2\pi) \frac{1}{2\pi} L \delta\left(x - x_0\right)$$

$$= \delta(x - x_0)$$

So this result is indeed consistent, at least for h(x) as a delta function.

2

Here we start out with the complex linear combination of exponentials:

$$(a + ib)e^{-\frac{2\pi ix}{L}} + (a - ib)e^{\frac{2\pi ix}{L}}$$

Expanding this out in terms of trig function, we obtain

$$2a\cos\left(\frac{2\pi x}{L}\right) + 2b\sin\left(\frac{2\pi x}{L}\right)$$

Now, setting $a = \frac{1}{2}A\sin(\varphi)$ and $b = \frac{1}{2}A\cos(\varphi)$, we obtain through the sine sum formula

$$A\sin(\frac{2\pi x}{L} + \varphi)$$

. Thus, we can create any generic sine function with given frequency out of the two fourier terms of the given frequency.

3

Here we have that $e^{\frac{2\pi ixf_k}{L}} = \cos(\frac{2\pi ixf_k}{L}) + i\sin(\frac{2\pi ixf_k}{L})$.

$$= \cos(\frac{2\pi ixf_k}{L}) + i\sin(\frac{2\pi ixf_k}{L})$$

Additionally,

$$e^{\frac{-2\pi ixf_k}{L}} = \cos(\frac{2\pi ixf_k}{L}) - i\sin(\frac{2\pi ixf_k}{L})$$

.

Thus, we see that the real parts of these expressions are the same, and the imaginary parts differ by a negative sign. Looking at the expression for the Fourier coefficients, $\tilde{h_k} = \frac{1}{L}\int_0^L h(x)e^{\frac{2\pi ixf_k}{L}}$, we have that if h(x) is real, then we have that $\tilde{h}_{-k} = \tilde{h}_k$.

4

Here we have that H(x) = h$^{(1)}$(x) h$^{(2)}$(x).

So, $\tilde{H}_k = \left(\sum_{k=-\infty}^{\infty} \tilde{h}^1{}_k e^{-\frac{2\pi i k x}{L}}\right) \sum_{k=-\infty}^{\infty} \tilde{h}^2{}_k e^{-\frac{2\pi i k x}{L}}$.

Expanding out over values of k, for each e-term we get coefficients multiplied of all the different ways to add up to k. This can be any combination of (k', k - k'), where k' goes from -∞ to +∞, since k runs from -∞ to +∞.

Thus, we simply obtain

$$\sum_{k'=-\infty}^{\infty} e^{-\frac{2\pi i k x}{L}} \tilde{h}^2{}_{k-k'} \tilde{h}^1{}_{k'}$$

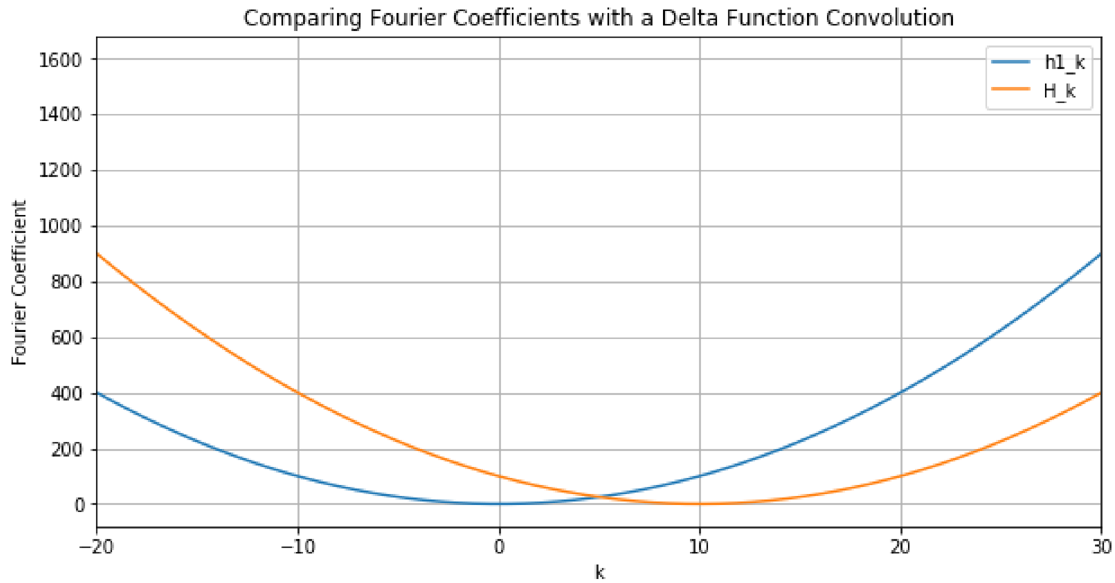
.

For the next part, if we try obtaining a coefficients for generic k, we get for $k' \neq 10$, $\tilde{h}^1{}_{k-k'} \tilde{h}^2{}_{k'} = 0$ due to $h^{(2)}$'s unit pulse coefficients centered around 10. If $k' = 10$, we get $\tilde{h}^{(1)}_{k-10}$. So, we're effectively just shifting all the Fourier coefficeints of the smooth function centered at 0 over by 10, giving us a smooth function centered at 10 for the Fourier coefficients of $\tilde{H}(x)$.

For example, let's say $h^{(1)}(x) = x^2$ and plot H.

```
from astropy.stats import LombScargle
from astropy.io.votable import parse
import numpy as np
import scipy
from scipy.signal import lombscargle
import matplotlib.pyplot as plt
%matplotlib inline
```

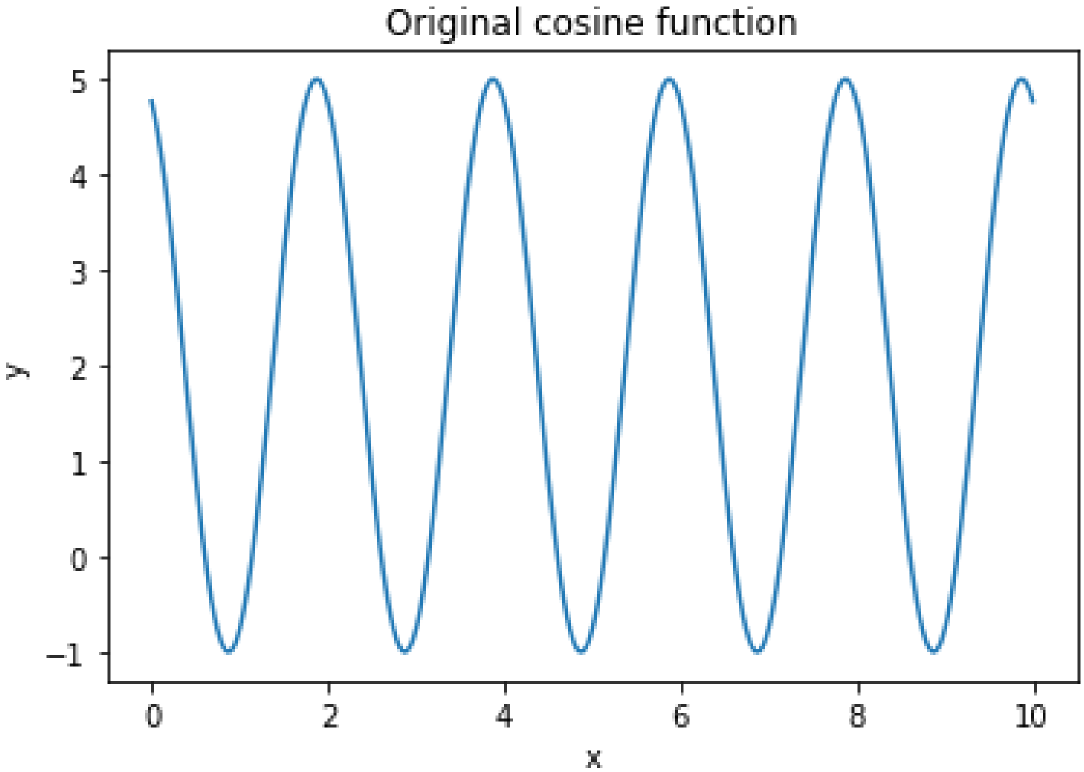
```
X = np.linspace(-30, 30, 1000)
plt.figure(figsize=(10, 5))
plt.plot(X, X ** 2, label = 'h1_k')
plt.plot(X, (X - 10) ** 2, label='H_k')
plt.legend()
plt.grid(True)
plt.xlabel('k')
plt.xlim([-20, 30])
plt.ylabel('Fourier Coefficient')
plt.title('Comparing Fourier Coefficients with a Delta Function Convolution')
plt.show()
```



5

We'll create a cosine function $C + Acos(2\pi ft + \phi)$, with $f = f_k = \frac{k}{L}$, with $L = 10, k = 5, C = 2, A = 3, \phi = \frac{\pi}{2}$.

```
def cosine(t, C, A, k, L, phi):
    return C + A * np.cos(2 * np.pi * (k / L) * t + phi)

# here we'll set C = 2, A = 3, k = 5, L = 10, phi = pi/8
X = np.linspace(0, 10, 10000)
Y = cosine(X, 2., 3., 5., 10., np.pi / 8)
plt.plot(X, Y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Original cosine function')
plt.show()
```



Here it looks like we do have 5 periods in every interval of 10. First we'll find these coefficients analytically in Mathematica:

$$\text{args} = \left\{A \to 3, C \to 2, \varphi \to \frac{\pi}{8}\right\};$$

$$\text{hk(k\_, h\_)}:= \frac{1}{10}\int_0^{10} h(x) e^{\frac{2}{10}\pi i k x}\, dx$$

$$h(\text{x\_}):= A\cos\left(\varphi + \frac{2\ 5\pi x}{10}\right) + C$$

$$\text{coefs} = |\text{Table}[N[\text{hk}(a, h)/.\ \text{args}], \{a, -10, 10\}]|$$

$$\{0., 0., 0., 0., 0., 1.5, 0., 0., 0., 0., 2., 0., 0., 0., 0., 1.5, 0., 0., 0., 0., 0.\}$$

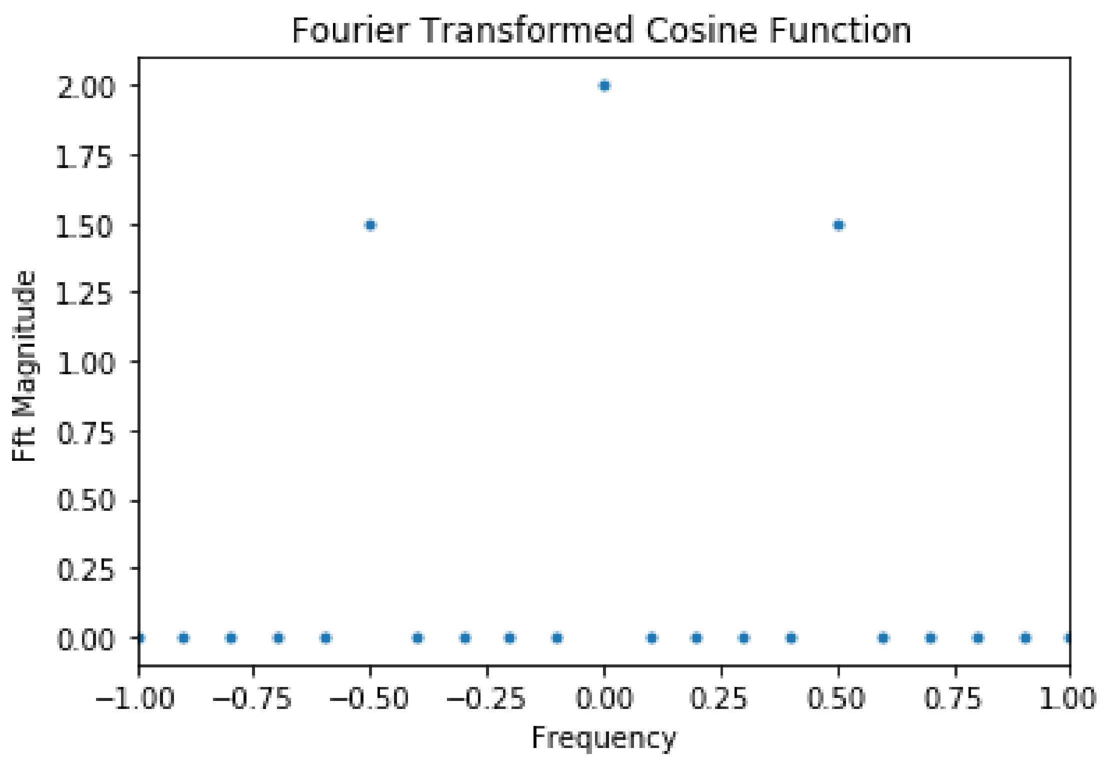
So here, it looks like we only get coefficients at frequencies of $\frac{1}{2}, -\frac{1}{2}$, and 0, with magnitudes of 1.5, 1.5, and 2.

Now let's check these values with numpy's fft:

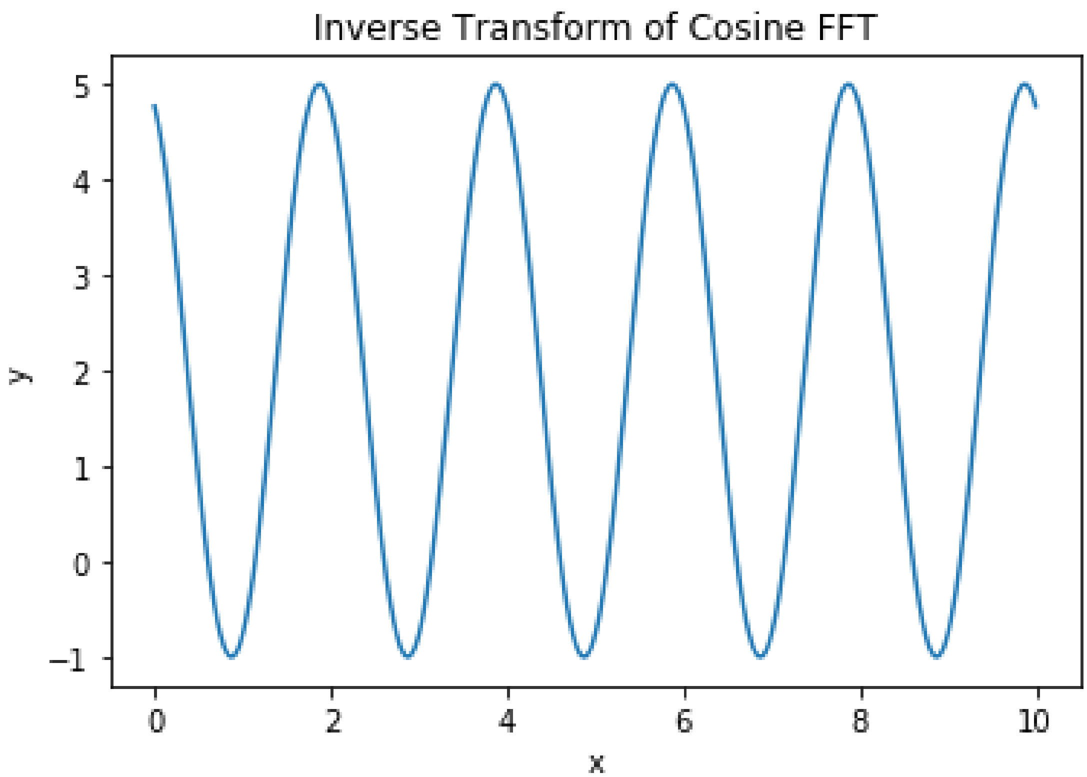
```python
def fft_and_freq(Y, d):
    fft = np.fft.fft(Y) / len(Y)
    freqs = np.fft.fftfreq(len(Y), d)
    return fft, freqs

def ifft(fftY, l):
    return np.fft.ifft(fftY * l)
```

```python
cosfft, cosfreqs = fft_and_freq(Y, (X[1] - X[0]))
```

```python
plt.plot(cosfreqs, np.abs(cosfft), '.')
plt.xlim([-1, 1])
plt.xlabel('Frequency')
plt.ylabel('Fft Magnitude')
plt.title('Fourier Transformed Cosine Function')
plt.show()
```

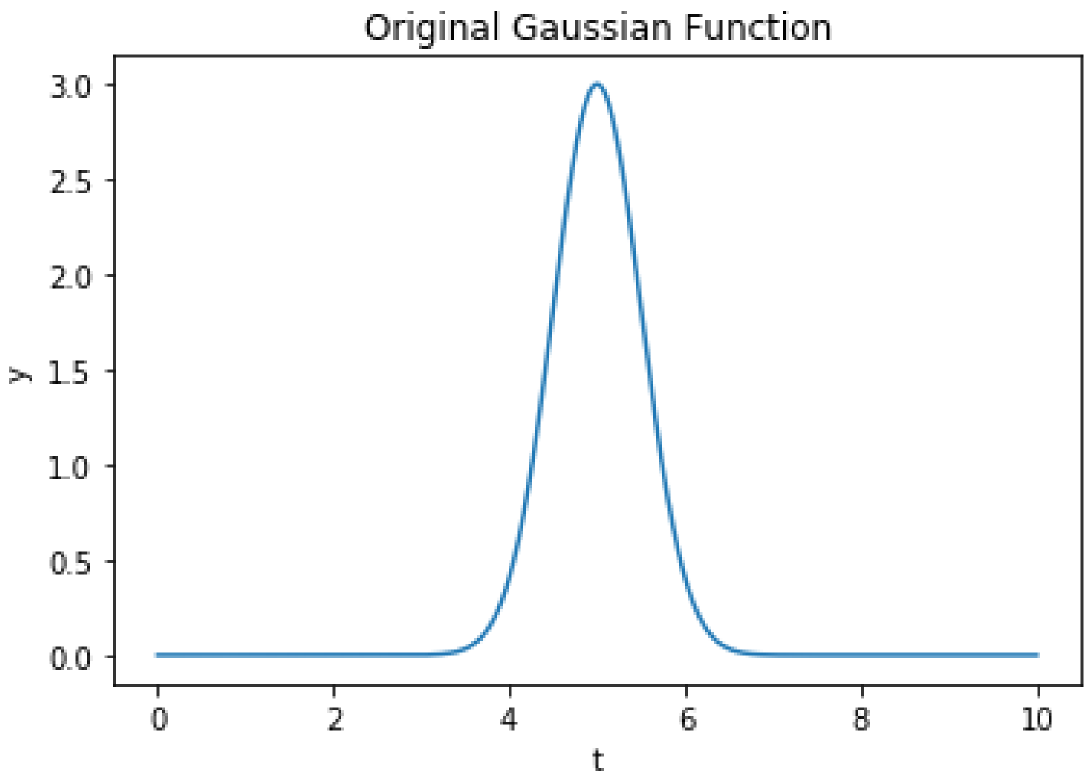



```python
plt.plot(X, ifft(cosfft, len(Y)))
plt.xlabel('x')
plt.ylabel('y')
plt.title('Inverse Transform of Cosine FFT')
plt.show()
```



Here the inverse again gives us the same thing we had before. Now let's do the same thing for the Gaussian function:

```
def gaussian(t, A, B, L):
    return A * np.exp(-B * ((t - L / 2) ** 2))

# here we'll set A = 3, B = 2, L = 10
Xg = np.linspace(0, 10, 10000)
Yg = gaussian(Xg, 3., 2., 10.)
plt.plot(Xg, Yg)
plt.xlabel('t')
plt.ylabel('y')
plt.title('Original Gaussian Function')
plt.show()
```



Finding the coefficients again in Mathematia, we see:

$$\text{h2}(\text{x\_}) := Ae^{-B\left(x-\frac{L}{2}\right)^2}$$

$$\text{args2} = \{A \to 3, B \to 2, L \to 10\};$$

$$\text{coefsGaussian} = |\text{Table}[N[\text{hk}(a, \text{h2})/.\,\text{args2}], \{a, -10, 10\}]|$$

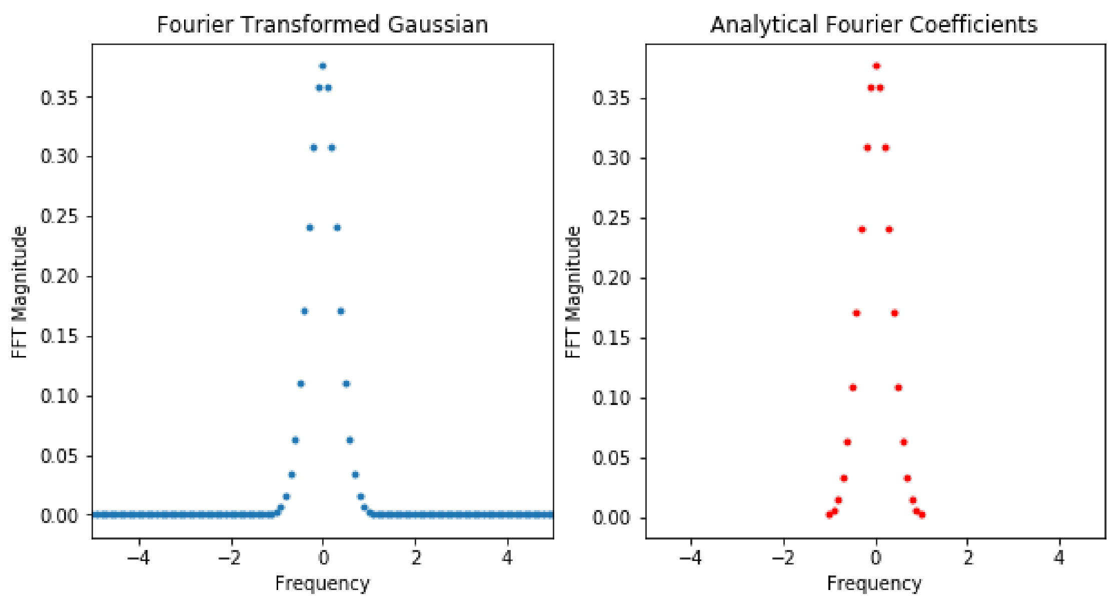
$$\{0.00270411, 0.00690595, 0.0159794, 0.0334992, 0.0636275, 0.109494,$$

$$0.170717, 0.241155, 0.308642, 0.35789, 0.375994, 0.35789, 0.308642, 0.241155, 0.170717,$$

$$0.109494, 0.0636275, 0.0334992, 0.0159794, 0.00690595, 0.00270411\}$$

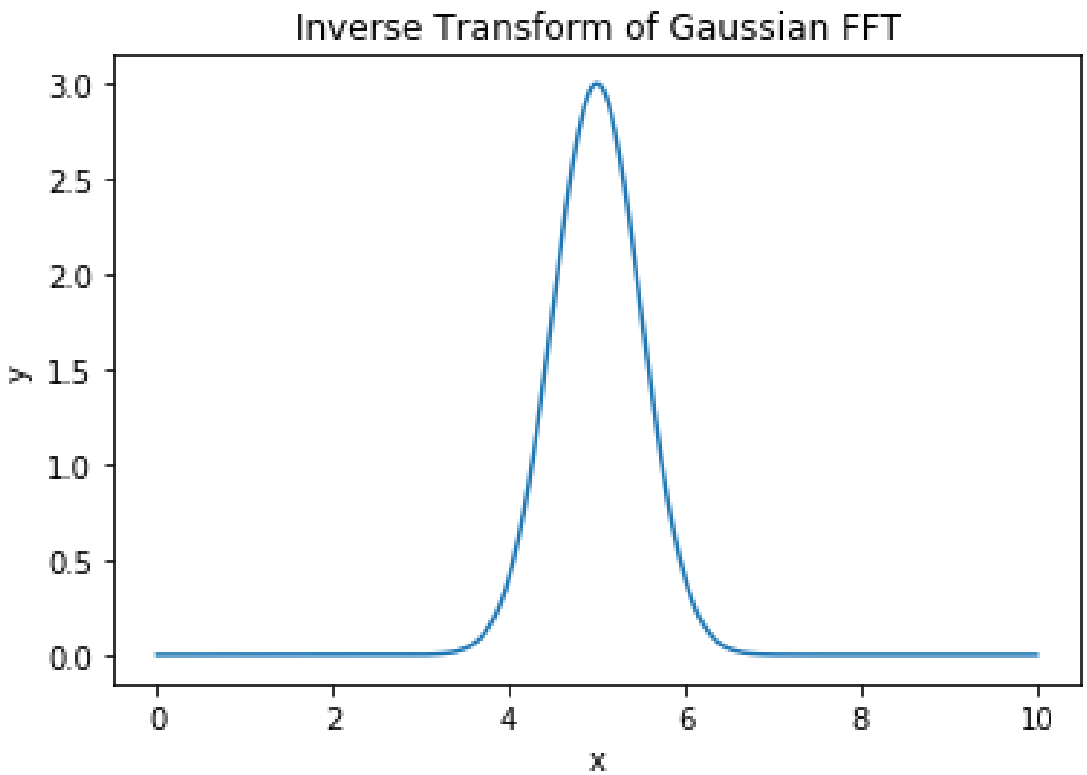
Now, checking these in numpy, we get:

```
mathematica_coefs = [0.00270411, 0.00690595, 0.0159794, 0.0334992, 0.0636275,
                     0.109494, 0.170717, 0.241155, 0.308642, 0.35789, 0.375994,
                     0.35789, 0.308642, 0.241155, 0.170717, 0.109494, 0.0636275,
                     0.0334992, 0.0159794, 0.00690595, 0.00270411]
mathematica_freqs = (1/10.) * np.linspace(-10, 10, 21)

gaussfft, gaussfreqs = fft_and_freq(Yg, (Xg[1] - Xg[0]))
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

ax1.plot(gaussfreqs, np.abs(gaussfft), '.')
ax1.set_title('Fourier Transformed Gaussian')

ax2.plot(mathematica_freqs, mathematica_coefs, '.', color='red')
ax2.set_title('Analytical Fourier Coefficients')

for ax in [ax1, ax2]:
    ax.set_xlabel('Frequency')
    ax.set_ylabel('FFT Magnitude')
    ax.set_xlim([-5, 5])
plt.show()
```



Here we do see that the Fourier transform of our Gaussian is indeed another Gaussian. And the normalizations do line up with our analytical coefficients.

```
plt.plot(Xg, ifft(gaussfft, len(Yg)))
plt.xlabel('x')
plt.ylabel('y')
plt.title('Inverse Transform of Gaussian FFT')
plt.show()
```



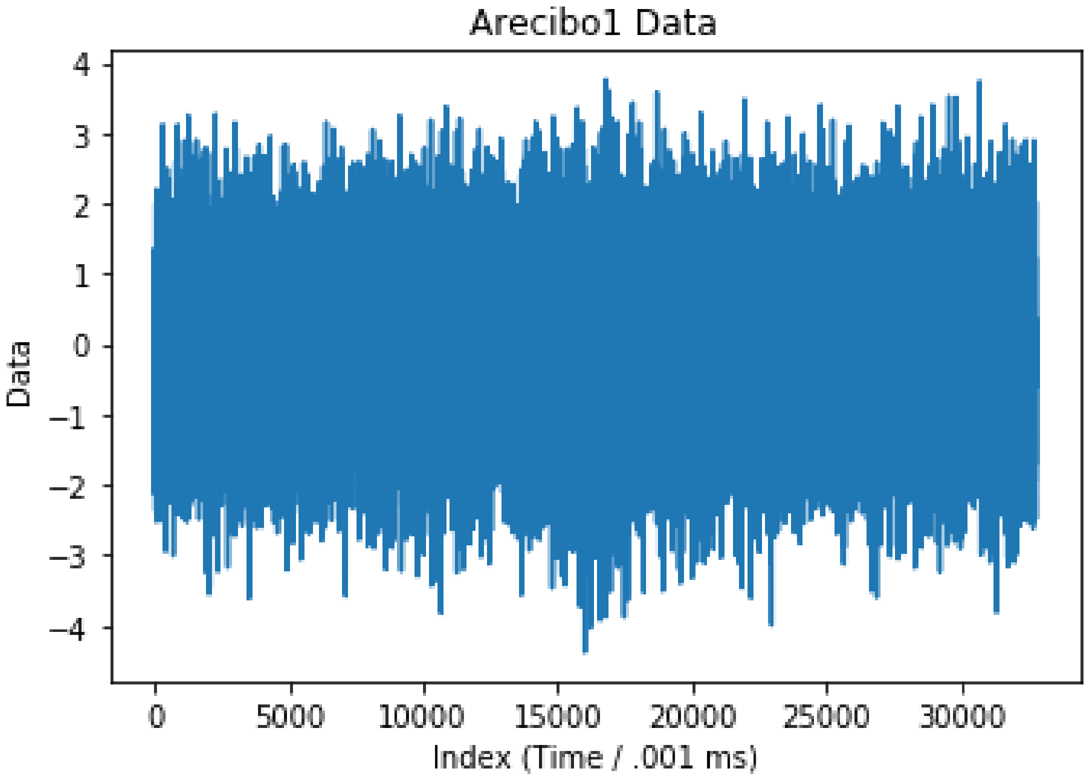
So here we also find the inverse giving us the same function.

Part II

(1)

```
dataset1 = np.genfromtxt("/Users/tommyalford/Documents/Ph21/Set2/arecibo1.txt")
```

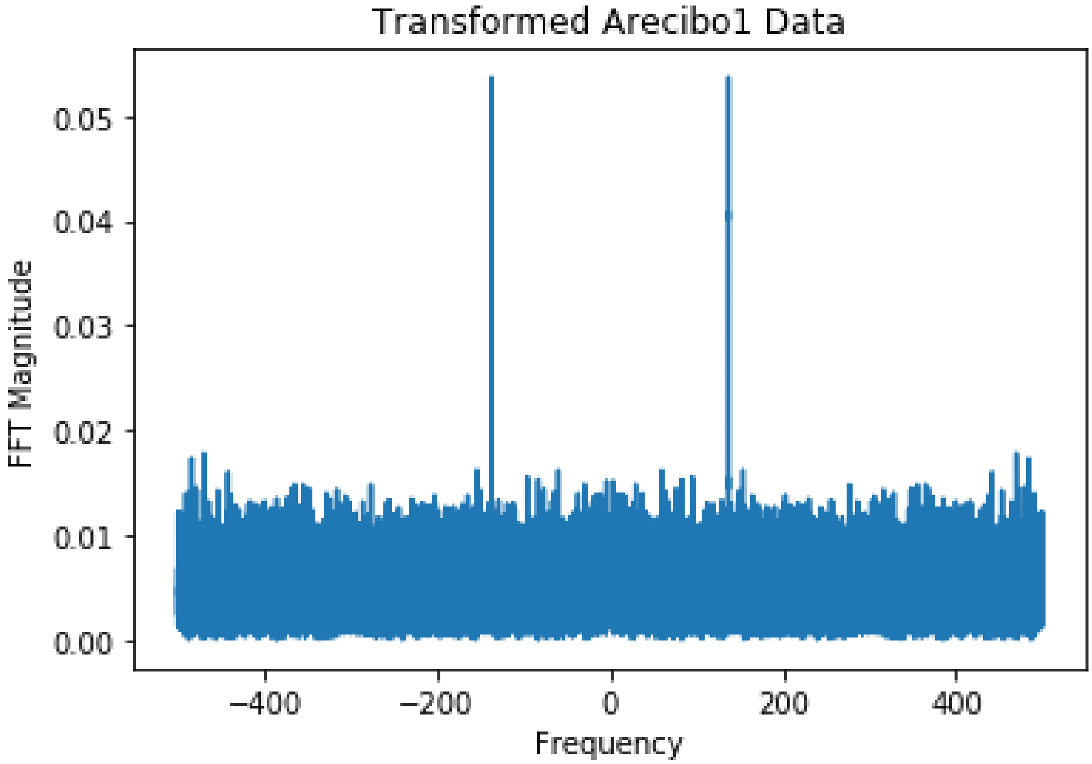
```
plt.plot(dataset1)
plt.ylabel('Data')
plt.xlabel('Index (Time / .001 ms)')
plt.title('Arecibo1 Data')
plt.show()
```



Definitely looks pretty noisy here. Let's try ffting it.

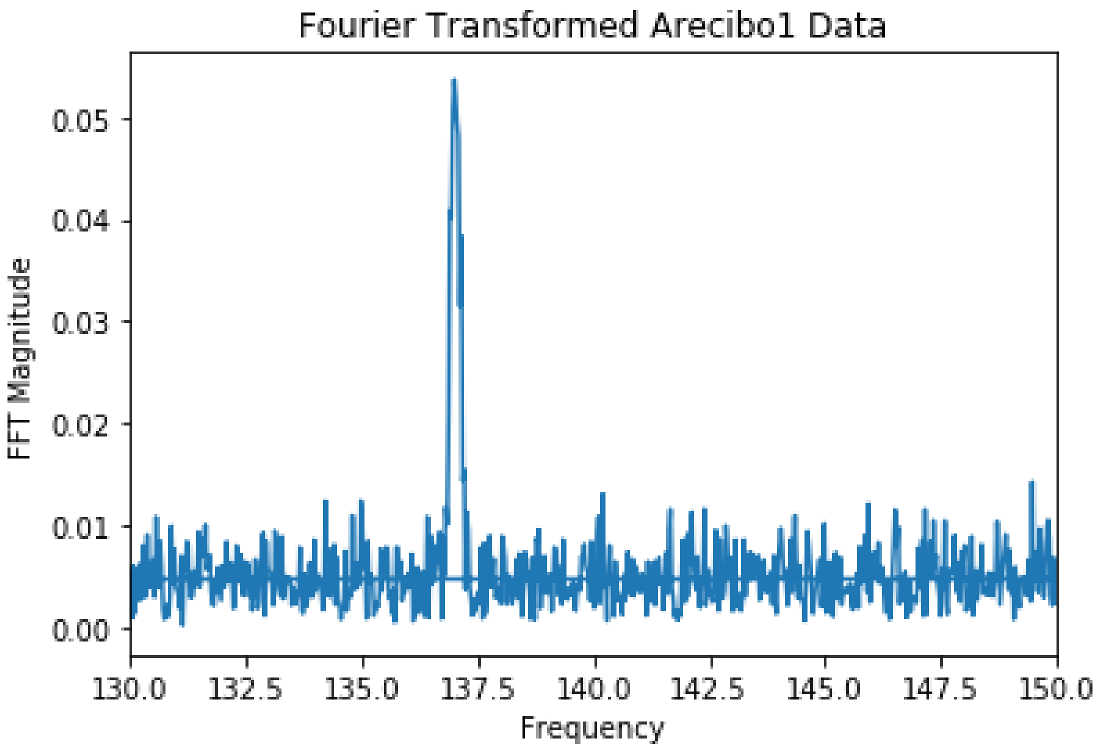
```
set1fft, set1freqs = fft_and_freq(dataset1, d=.001)
```

```
plt.plot(set1freqs, np.abs(set1fft))
plt.ylabel('FFT Magnitude')
plt.xlabel('Frequency')
plt.title('Transformed Arecibo1 Data')
plt.show()
```



Here we clearly see most of the signal near the frequency of 150Hz. Let's zoom in on the data and then find this frequency more accurately:

```
plt.plot(set1freqs, np.abs(set1fft))
plt.xlim([130, 150])
plt.xlabel('Frequency')
plt.ylabel('FFT Magnitude')
plt.title('Fourier Transformed Arecibo1 Data')
plt.show()
```



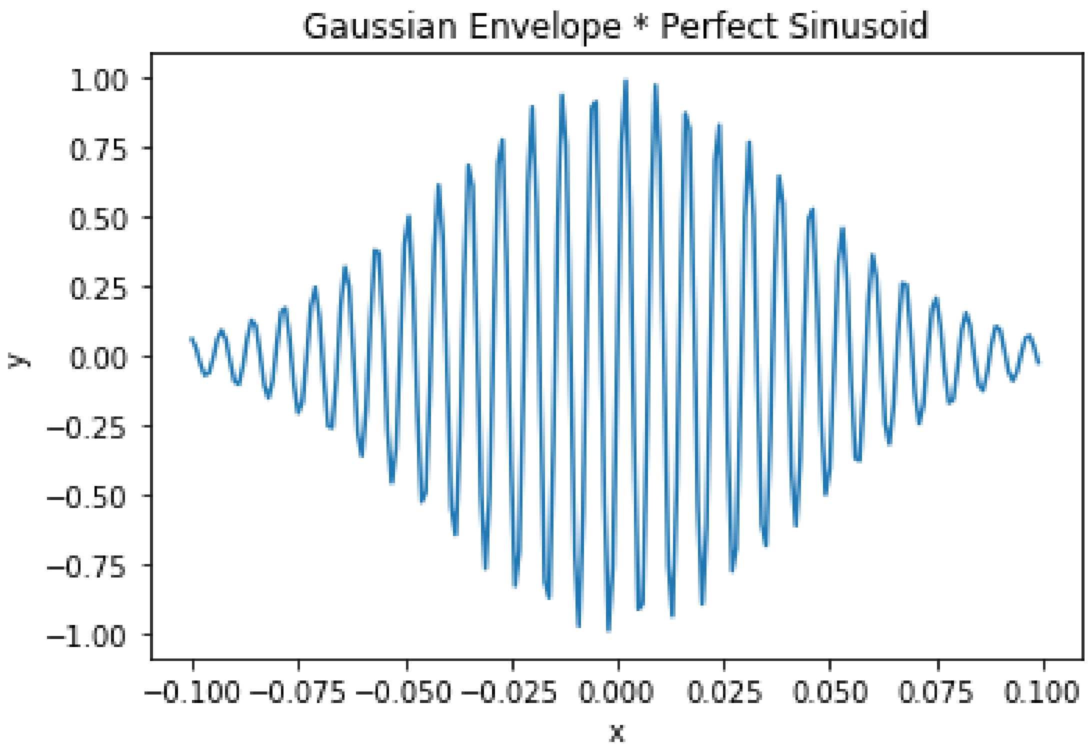

```
set1freqs[np.argmax(np.abs(set1fft))]
```

```
136.993408203125
```

So, it looks like our signal has a frequency of 137 Hz!

(2)

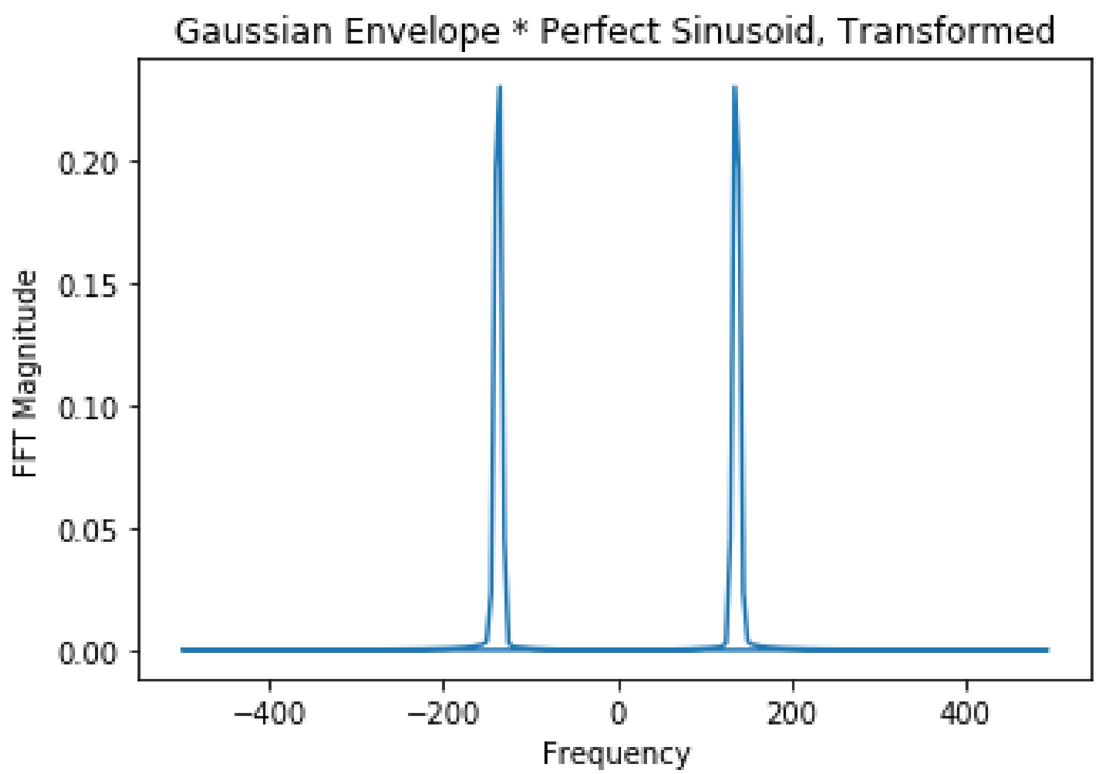
```
def gaussian_envelope(t, t0, deltat):
    return np.exp((-(t - t0) ** 2) / (2 * deltat) ** 2)

def perfect_sin(t, f):
    return np.sin(2 * np.pi * f * t)

dt=.03
X = np.arange(-.1, .1, step=.001)
Y = perfect_sin(X, 137) * gaussian_envelope(X, 0, dt)

plt.plot(X, Y)
plt.title('Gaussian Envelope * Perfect Sinusoid')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



This looks right. Now we'll try Fourier transforming it to see what we get:

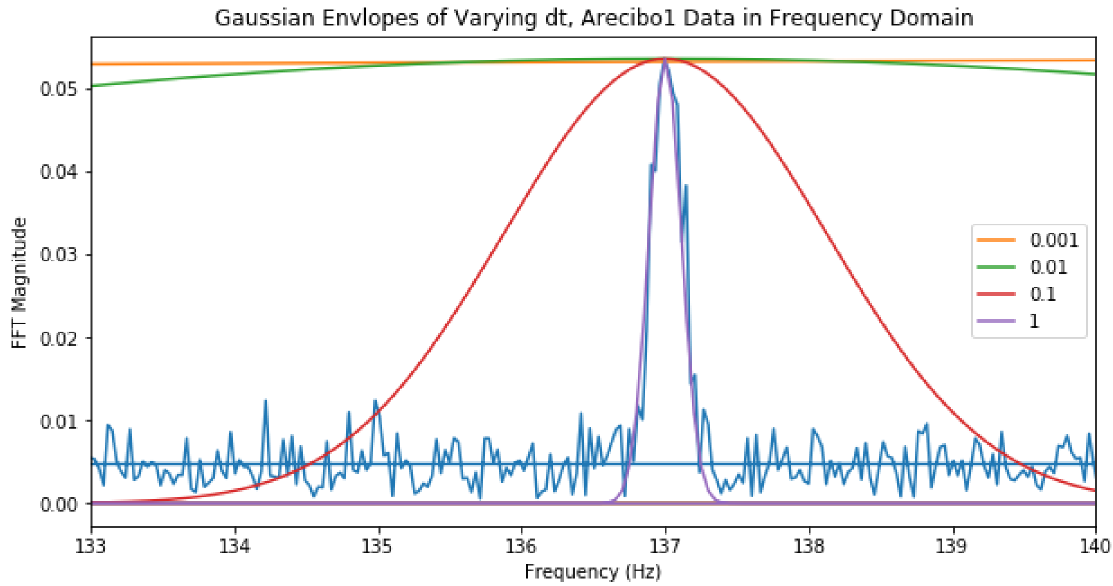
```
envfft, envfreqs = fft_and_freq(Y, d=.001)
```

```
plt.plot(envfreqs, np.abs(envfft))
plt.title('Gaussian Envelope * Perfect Sinusoid, Transformed')
plt.xlabel('Frequency')
plt.ylabel('FFT Magnitude')
plt.show()
```



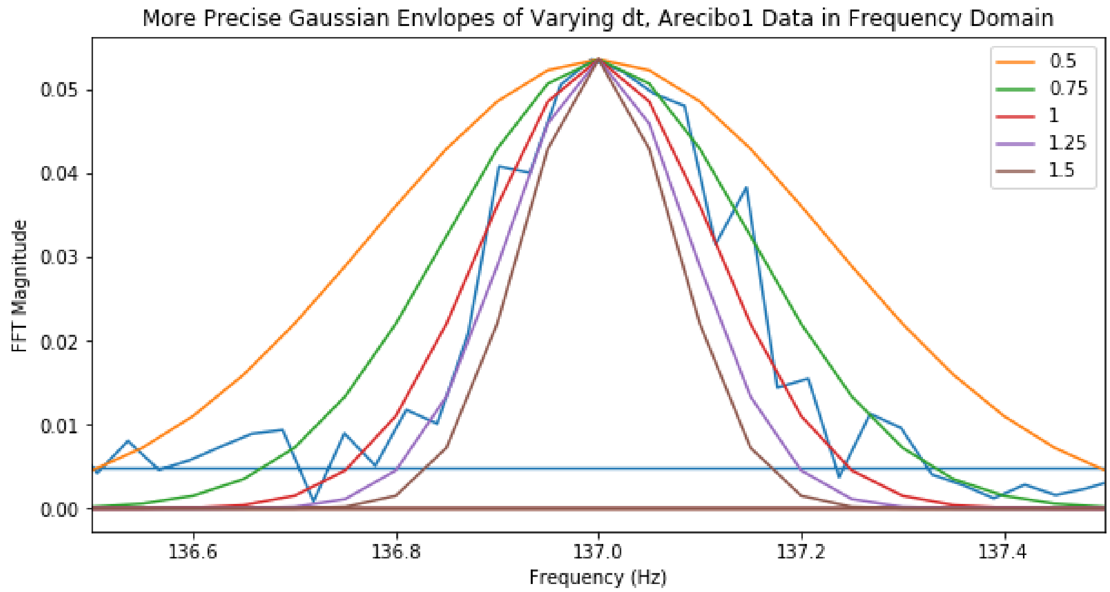
Now we'll loop through a bunch of different Δt values and plot superimpose plots of them over the original fft data.

```
# need to normalize so that our max magnitude is the same as the first
# that way we can directly compare widths really
max_mag = np.max(np.abs(set1fft))
plt.figure(figsize=(10, 5))
plt.plot(set1freqs, np.abs(set1fft))
plt.xlim([133, 140])

X = np.arange(-10, 10, step=.001)
dtvals = [.001, .01, .1, 1]

def plot_envelope(dt):
    Yset = perfect_sin(X, 137) * gaussian_envelope(X, 0, dt)
    fft, freqs = fft_and_freq(Yset, d=.001)
    plt.plot(freqs, np.abs(fft) * (max_mag / np.max(np.abs(fft))), label=dt)

for dt in dtvals:
    plot_envelope(dt)

plt.legend()
plt.title('Gaussian Envlopes of Varying dt, Arecibo1 Data in Frequency Domain')
plt.xlabel('Frequency (Hz)')
plt.ylabel('FFT Magnitude')
plt.show()
```



Looks like $\Delta t = 1$ actually gives us a pretty close approximation. We can try some more values near that:

```
plt.figure(figsize=(10, 5))
plt.plot(set1freqs, np.abs(set1fft))
plt.xlim([133, 140])
for dt in [.5, .75, 1, 1.25, 1.5]:
    plot_envelope(dt)

plt.legend()
plt.title('More Precise Gaussian Envlopes of Varying dt, Arecibo1 Data '
          'in Frequency Domain')
plt.xlabel('Frequency (Hz)')
plt.ylabel('FFT Magnitude')
plt.xlim([136.5, 137.5])
plt.show()
```



This is pretty hard to tell at this point. Seems to be pretty close to 1. Fitting using a real fit would be much more convienient.

(4)

Part III

(1)

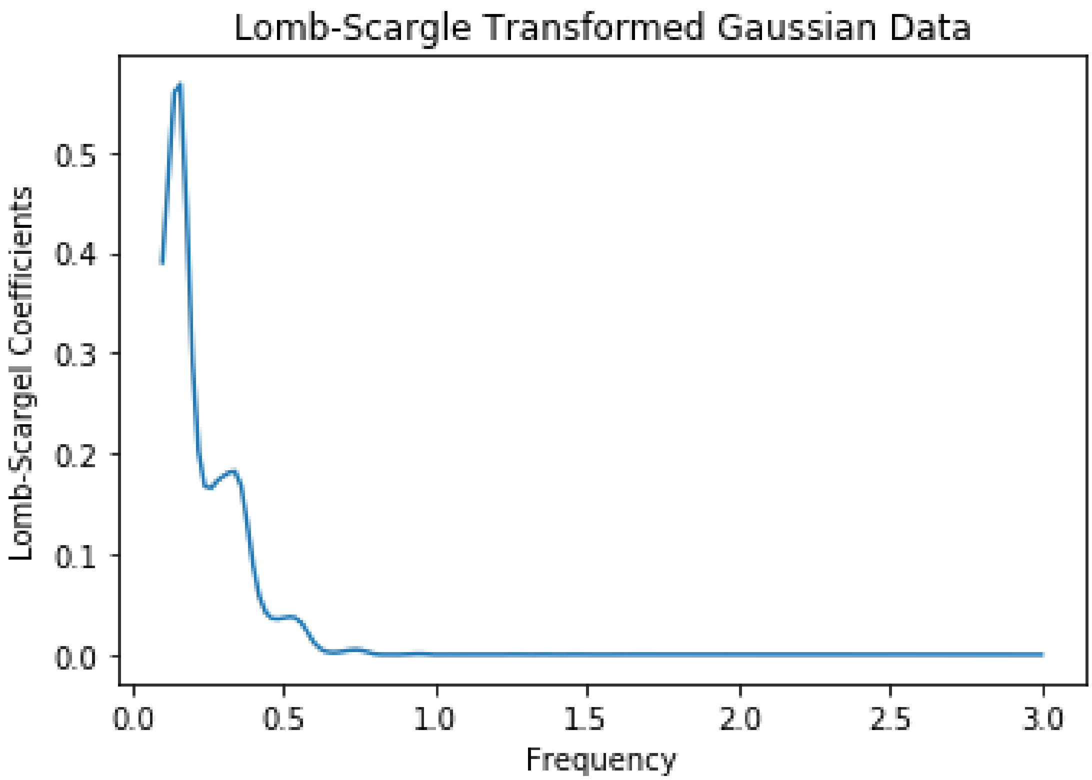
We'll be using the astopy lombscargle algorithm now.

(2)

Gaussian

```
# from before, we had Xg, Yg. Will take similar freqs here
gauss_lomb_freqs, gausslomb = LombScargle(Xg, Yg, .01).autopower(
    minimum_frequency=0.1, maximum_frequency=3)
```

```
plt.plot(gauss_lomb_freqs, np.abs(gausslomb))
plt.xlabel('Frequency')
plt.ylabel('Lomb-Scargel Coefficients')
plt.title('Lomb-Scargle Transformed Gaussian Data')
plt.show()
```

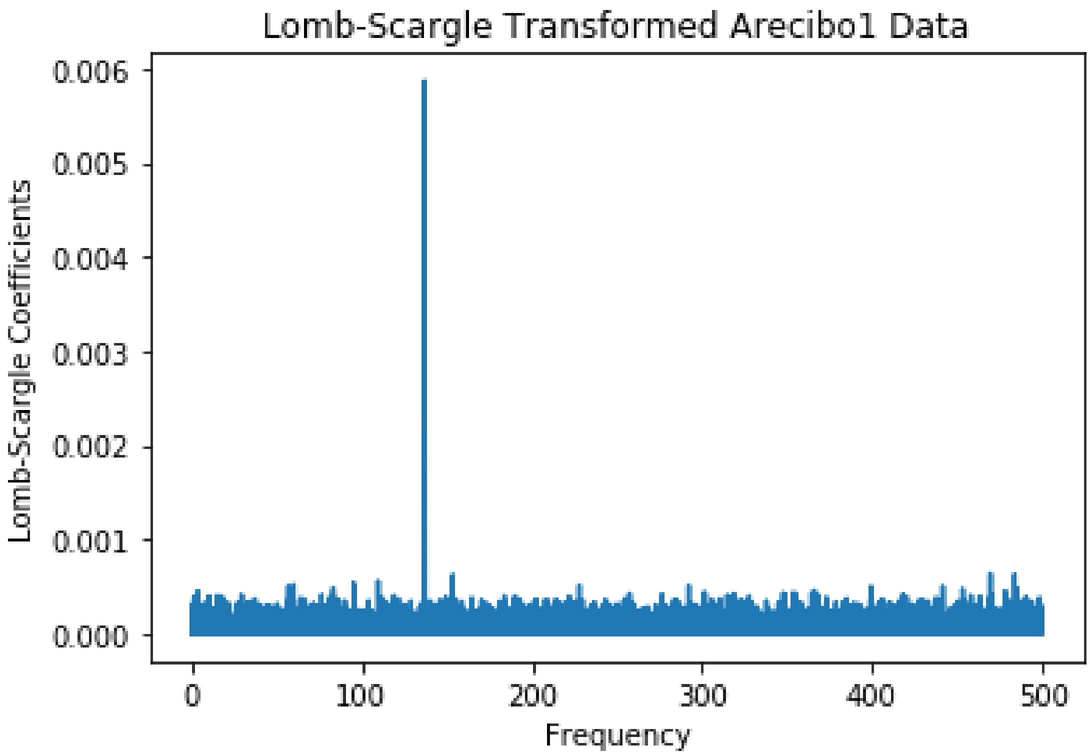


Here we can still see the Gaussian behavior, just with a little bit of noise added in.

Part II Data

```
set1_times = np.arange(len(dataset1) * .001, step=.001)
set1_lomb_freqs, set1_lomb = LombScargle(set1_times, dataset1, .01).autopower(
    minimum_frequency=0.1, maximum_frequency=500)
```

```
plt.plot(set1_lomb_freqs, np.abs(set1_lomb))
plt.xlabel('Frequency')
plt.ylabel('Lomb-Scargle Coefficients')
plt.title('Lomb-Scargle Transformed Arecibo1 Data')
plt.show()
```



Here we can now find the 137 Hz signal as before!

3

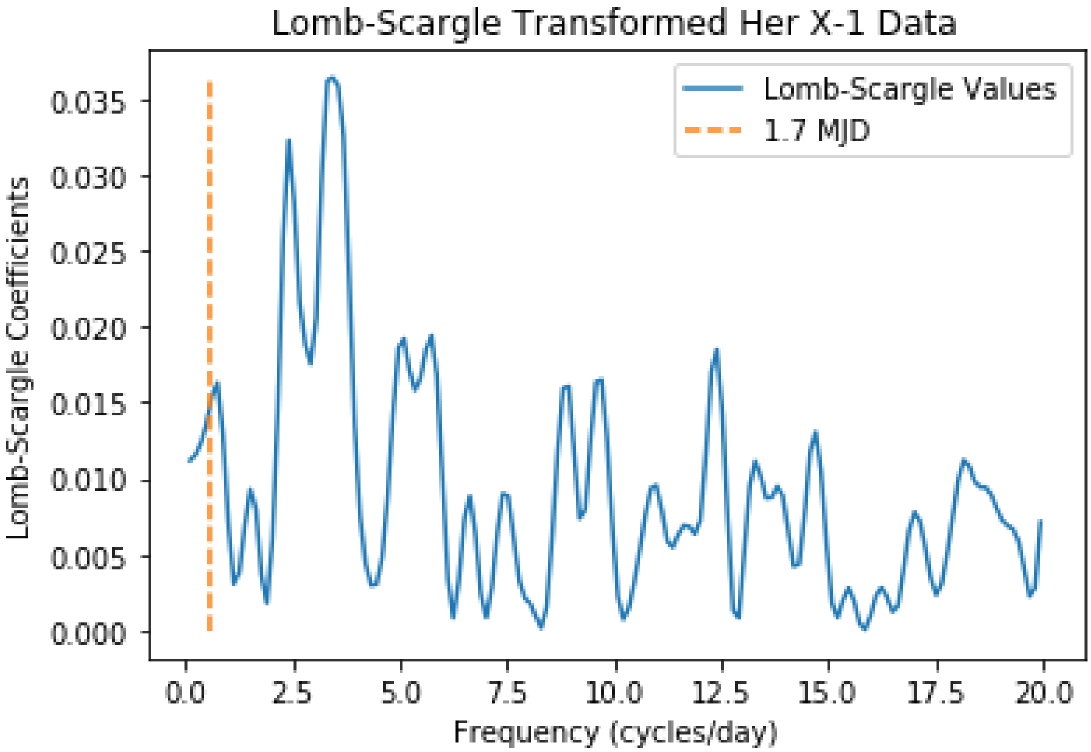
```
votable = parse("/Users/tommyalford/Documents/Ph21/set1/result_web_fileDR8owK.vot",
                pedantic=False)
vo_data = votable.get_first_table().to_table()
```

```
vo_mags = np.array(vo_data['Mag']).astype('float64')
vo_errs = np.array(vo_data['Magerr']).astype('float64')
vo_MJDs = np.array(vo_data['ObsTime']).astype('float64')
```

```
# subtract mean from mag data
mags = (vo_mags - np.mean(vo_mags)).flatten()
days = (vo_MJDs).flatten()
dy = .0001
vo_freqs, vo_lomb = LombScargle(mags, days, dy, center_data=True).autopower(
    minimum_frequency=.1, maximum_frequency=20.)
```

```
plt.plot(vo_freqs, vo_lomb, label='Lomb-Scargle Values')
plt.plot(1000 * [1 / 1.7], np.linspace(0, np.max(vo_lomb), 1000),
         '--', label='1.7 MJD')
```

```python
plt.xlabel('Frequency (cycles/day)')
plt.ylabel('Lomb-Scargle Coefficients')
plt.title('Lomb-Scargle Transformed Her X-1 Data')
plt.legend()
plt.show()
```



Here this might be the 1.7 MJD period that we want. Otherwise there are some significant beats near frequencies of 2.5, 5, 10, 12.5, etc cylces per day.