PS5

October 30, 2021

```
[2]: import numpy as np
     import matplotlib.pyplot as plt
```

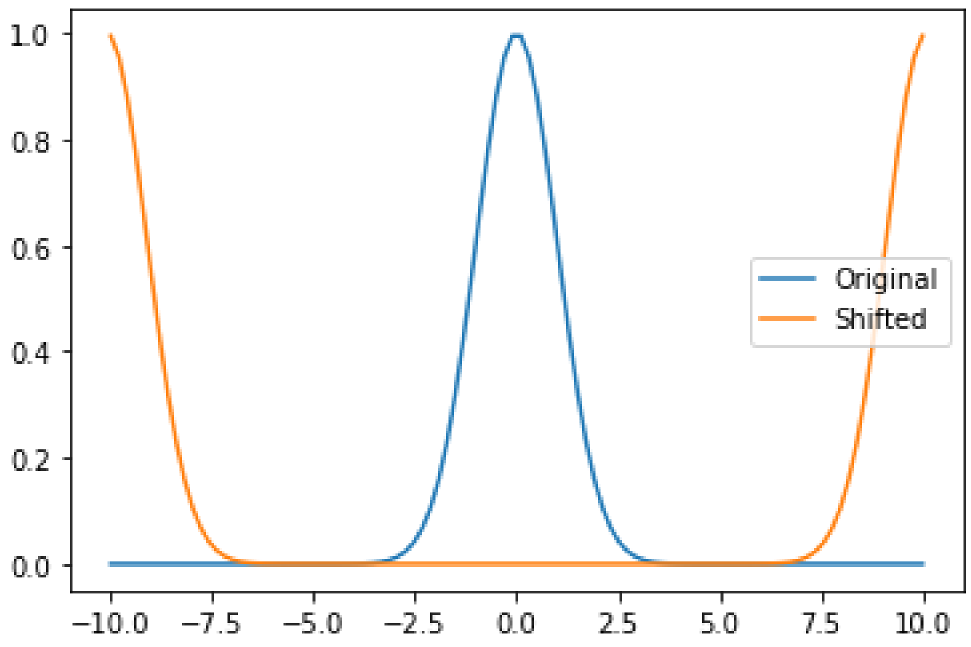
0.1 Question 1

```
[155]: def Shift(array, dx):
           N = len(array)
           k = np.arange(N)
           ft = np.fft.fft(array)
           ft_new = ft*np.exp(-2j*np.pi*k*dx/N)
           new_array = np.fft.ifft(ft_new)
           return new_array
```

```
[352]: x = np.linspace(-10, 10, 100)
       y = np.exp(-x**2/(2*1**2))
       dx = len(y)/2
       y_new = Shift(y, dx)
```

```
[353]: plt.plot(x, y, label = 'Original')
       plt.plot(x, y_new, label = 'Shifted')
       plt.legend()
```

```
[353]: <matplotlib.legend.Legend at 0x118e735c0>
```

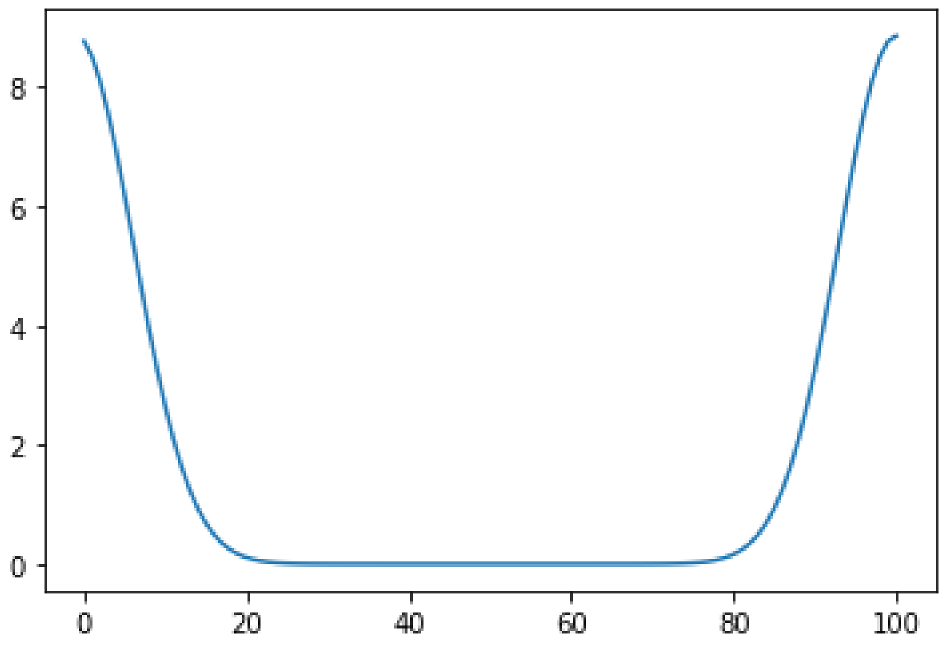


0.2 Question 2

[273]:
```python
def Corr(fun1, fun2):
    ft1 = np.fft.fft(fun1)
    ft2 = np.fft.fft(np.flip(fun2))
    return np.real(np.fft.ifft(ft1*ft2))
```

[274]:
```python
x = np.linspace(-10,10,101)
fun1 = np.exp(-x**2/(2))
corr = Corr(fun1, fun1)
plt.plot(corr)
```

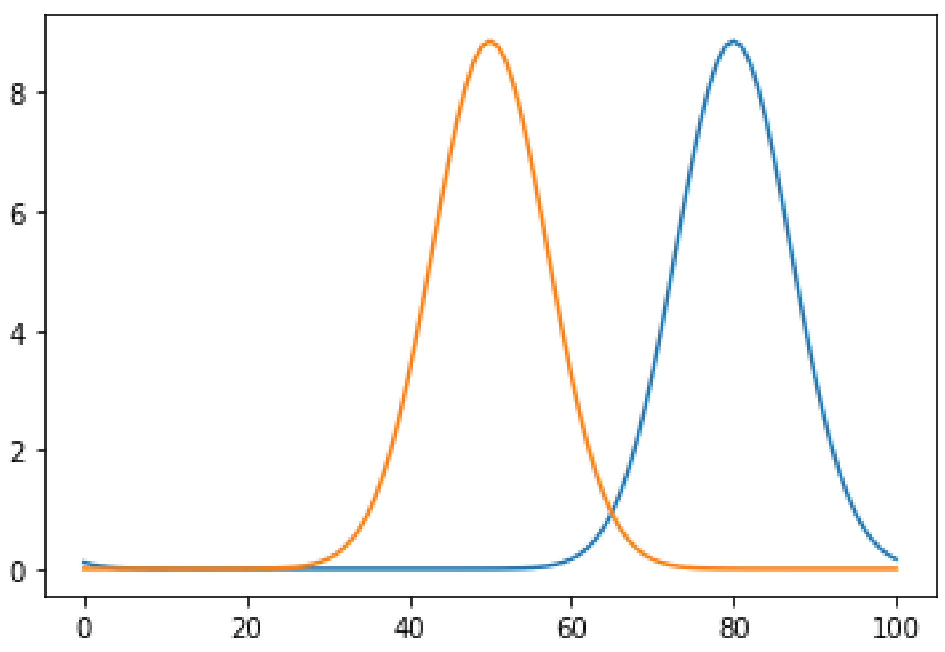
[274]:
```
[<matplotlib.lines.Line2D at 0x11955d278>]
```



0.3 Question 3

[851]:
```python
def ShiftCorr(fun, dx):
    shift = Shift(fun, dx)
    corr = Corr(fun, shift)
    return corr
```

[852]:
```python
x = np.linspace(-10, 10, 101)
gauss = np.exp(-x**2/2)
shift1 = ShiftCorr(gauss, 20)
plt.plot(shift1)
shift2 = ShiftCorr(gauss, 50)
plt.plot(shift2)
plt.show()
```



0.4 Question 4

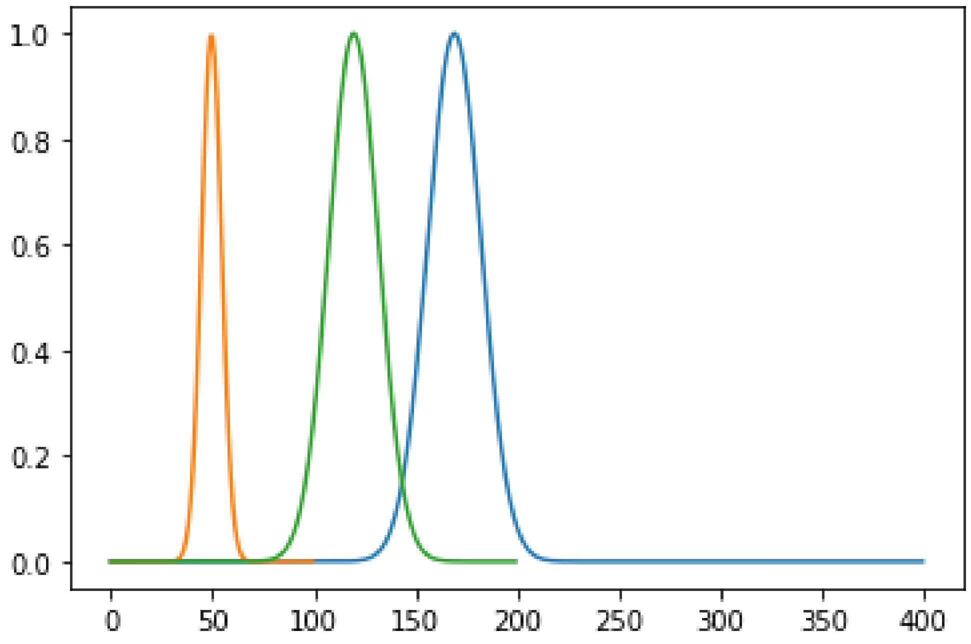
[456]:
```
def conv_safe(f,g):
    diff = abs(len(f)-len(g))
    n = len(f)
    if len(f) > len(g):
        g = np.concatenate((g,np.zeros(diff)))
        n = round(len(f))
    elif len(f) < len(g):
        f = np.concatenate((f,np.zeros(diff)))
        n = round(len(g))
    f = np.concatenate((f,np.zeros(n)))
    g = np.concatenate((g,np.zeros(n)))
    fft = np.fft.fft(f)
    gft = np.fft.fft(g)
    conv = np.real(np.fft.ifft(fft*gft))
    return conv
```

[850]:
```
x = np.linspace(-10, 10, 100)
x2 = np.linspace(-10, 10, 200)
gauss = np.exp(-x**2/2)
gauss2 = np.exp(-(x2-2)**2/3)
#sin1 = np.sin(x)
#sin2 = np.sin(x2)
conv = conv_safe(gauss, gauss2)
print(len(conv))
```

```
plt.plot(conv/max(conv), label = 'Conv')
plt.plot(gauss, label = 'Function 1')
plt.plot(gauss2, label = 'Function 2')
```

```
400
```

[850]:
```
[<matplotlib.lines.Line2D at 0x131b66f28>]
```



Here we add first pad the smaller array with enough zeros so that the 2 arrays are the same size as we need for future operations. Then since we need both arrays to be zero padded, we add zeros equal to the size of the array to the arrays so that our output array is twice the size of the longest array passed in.

0.5 Question 5

0.5.1 a.) $\sum_{x=0}^{N-1} exp(-2\pi ikx/N) = \sum_{x=0}^{N-1} exp(-2\pi ik/N)^x$

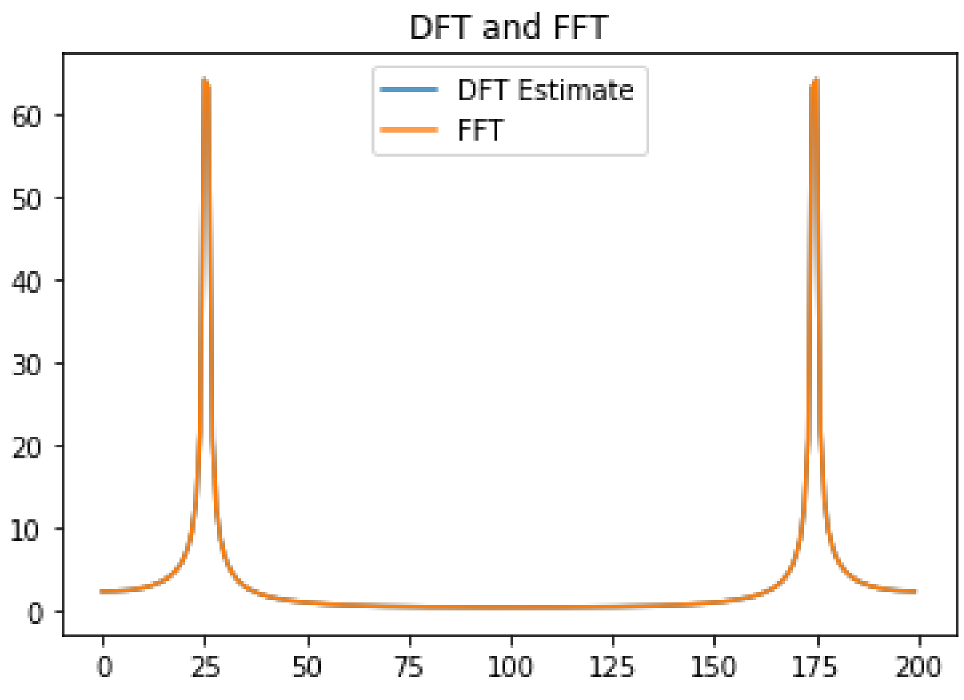
Recall Geometric Series: $\sum_{k=0}^{n} ar^k = a(\frac{1-r^{n+1}}{1-r})$ $\sum_{x=0}^{N-1} exp(-2\pi ik/N)^x = \frac{1-exp(-2\pi ik/N)^{N-1+1}}{1-exp(-2\pi ik/N)} = \frac{1-exp(-2\pi ik)}{1-exp(-2\pi ik/N)}$

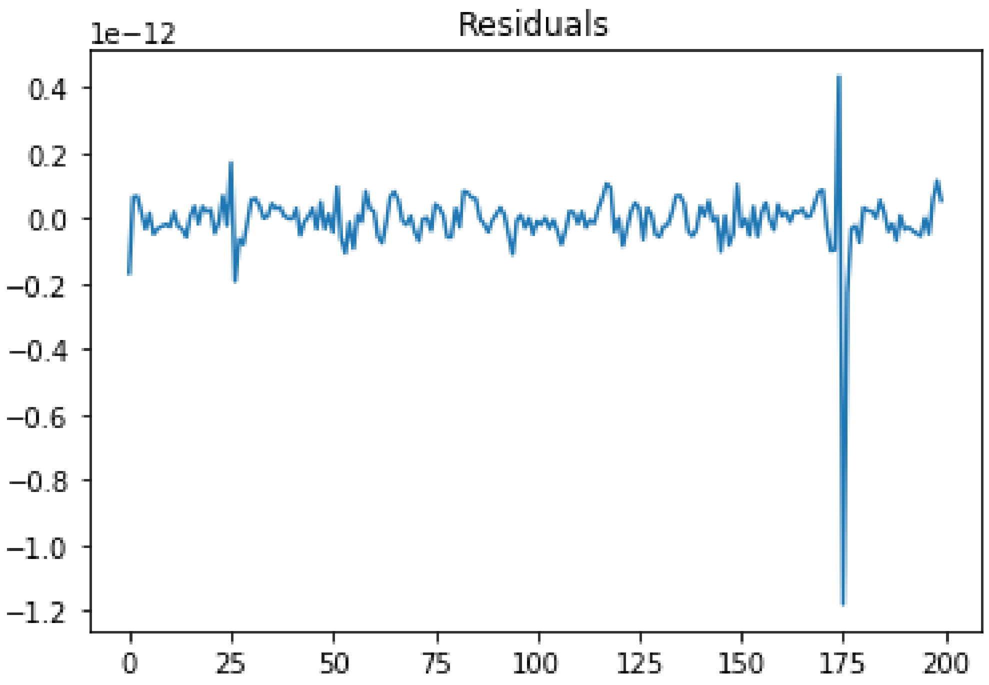
0.5.2 b.)

Using l'hopital's rule: $\lim_{k\to 0} \frac{1-exp(-2\pi ik)}{1-exp(-2\pi ik/N)} = \lim_{k\to 0} \frac{2\pi iexp(-2\pi ik)}{2\pi i/Nexp(-2\pi ik/N)} = N$ $1 - exp(2\pi ik) = 0, k \in \mathbb{Z}$ Therefore, if k is not a multiple of N, $1 - exp(-2\pi ik/N) \neq 0$ Hence, $\frac{1-exp(-2\pi ik)}{1-exp(-2\pi ik/N)} = 0$ if k is an integer, but not a multiple of N.

0.5.3 c.)

[859]:
```
k2 = np.arange(200)
k = 51/2
N = len(k2)
def dft_sin(k, k2):
    N = len(k2)
    num_1 = 1-np.exp(2j*np.pi*(k-k2))
    bot_1 = 1-np.exp(2j*np.pi*(k-k2)/N)
    num_2 = 1-np.exp(-2j*np.pi*(k+k2))
    bot_2 = 1-np.exp(-2j*np.pi*(k+k2)/N)
    return np.real(1/(2j)*((num_1/bot_1)-(num_2/bot_2)))
dft = dft_sin(k=k, k2=k2)
plt.plot(abs(dft), label = 'DFT Estimate')
fft = np.fft.fft(np.sin(2*np.pi*k*k2/N))
plt.plot(abs(fft), label = 'FFT')
plt.title('DFT and FFT')
plt.legend()
plt.show()
plt.plot(abs(fft)-abs(dft))
plt.title('Residuals')
plt.show()
```



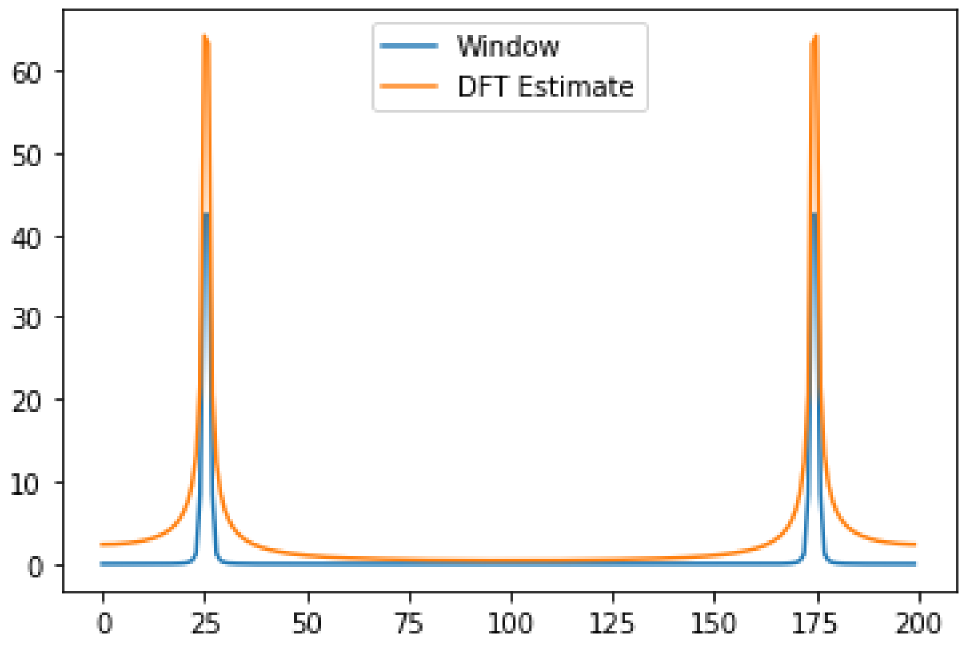


By plotting the residuals between the our analytic estimate of the DFT and FFT, we see that we have errors on the order of magnitude of -12. We plot our DFT and observe that the curve looks like a wide delta function.

0.5.4 d.)

[844]:
```
k2 = np.arange(200)
k = 51/2
N = len(k2)
window = 0.5 - 0.5*np.cos(2*np.pi*k2/N)
fft = np.fft.fft(window*np.sin(2*np.pi*k*k2/N))
plt.plot(abs(fft), label = 'Window')
plt.plot(abs(dft), label = 'DFT Estimate')
plt.legend()
```

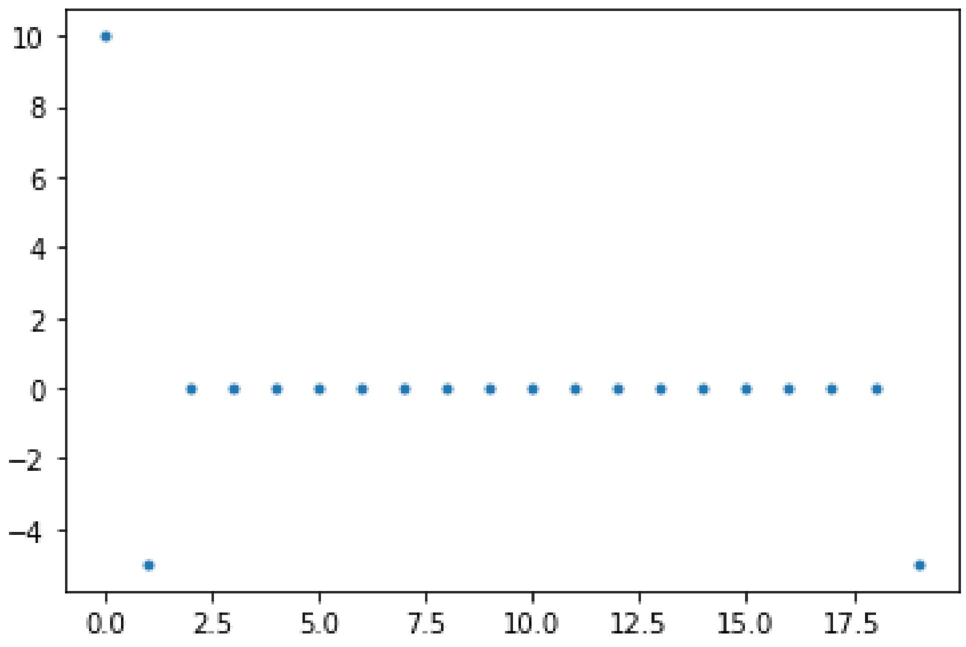
[844]:
```
<matplotlib.legend.Legend at 0x1313b1470>
```



0.5.5 e.)

[572]:
```python
k2 = np.arange(20)
N = len(k2)
window = 0.5 - 0.5*np.cos(2*np.pi*k2/N)
winfft = np.fft.fft(window)
plt.plot(np.real(winfft), '.')
print(np.round(np.real(winfft)))
```

```
[10. -5.  0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0.
  0. -5.]
```

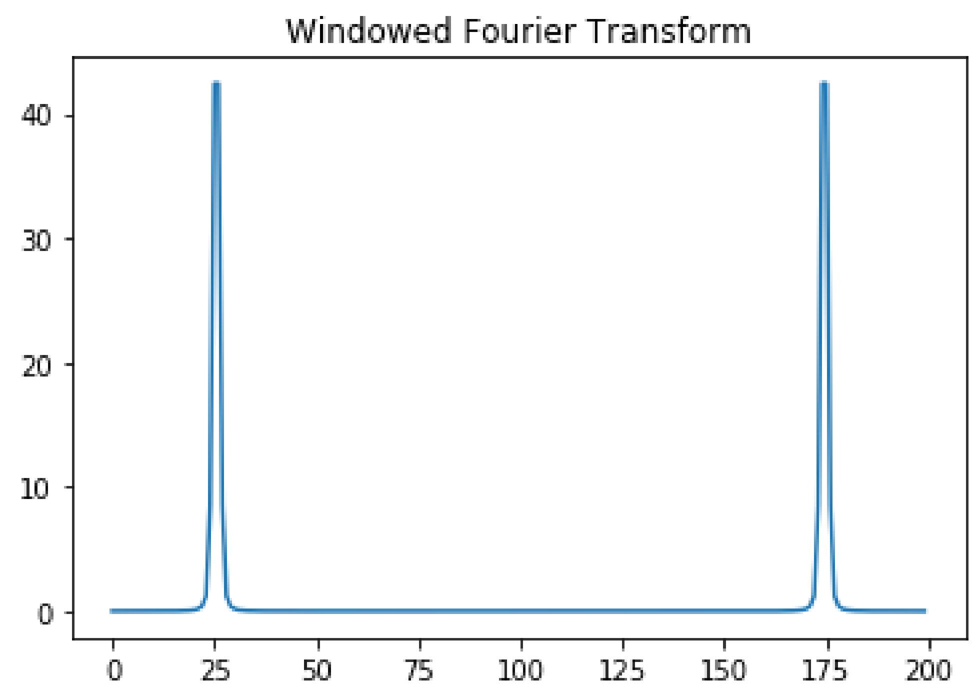


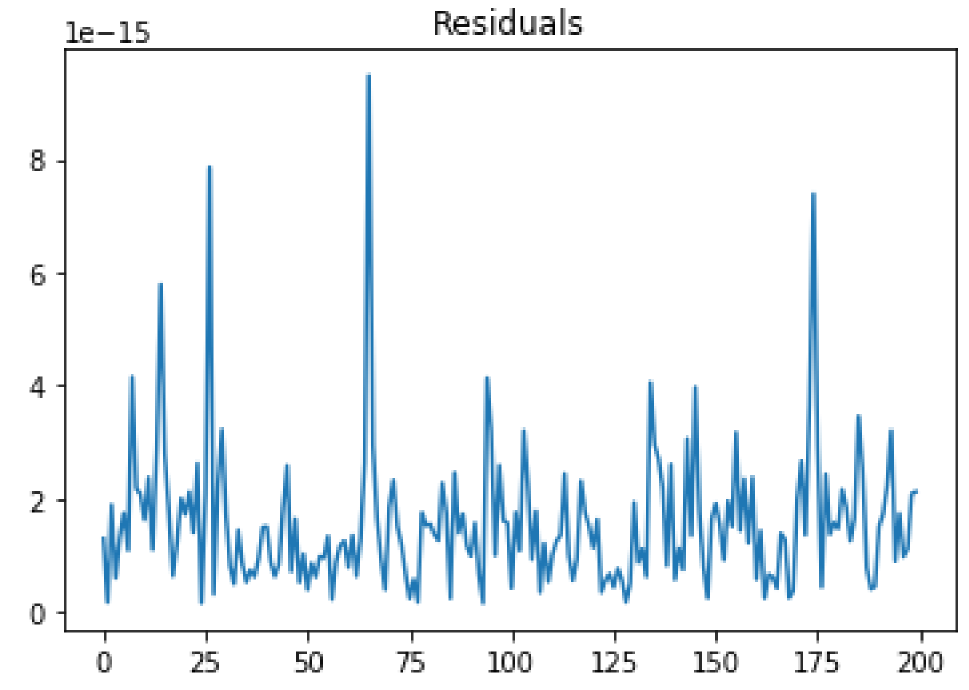
Since we have N = 20, we expect the first, second and last elements of the array to have values of 20/2, -20/4, -20/4 respectively, which we observe in the plot/print statement. For this next part, we know that the Fourier Transform (FT) of the window tells us how to sample points in order to apply the window in Fourier Space. So our windowed sin function is simply: $F(N) = -\frac{F(N-1)}{4} + \frac{F(N)}{2} - \frac{F(N+1)}{4}$ where $F(N)$ is the FT of our function.

[860]:
```
x = np.arange(200)
k = 51/2
N = len(x)


rfft = np.fft.fft(np.sin(2*np.pi*k*x/N))
windowed = -np.roll(rfft, -1)/4 + rfft/2 - np.roll(rfft, 1)/4
plt.plot(np.abs(windowed))
plt.title('Windowed Fourier Transform')
plt.show()


window = 0.5 - 0.5*np.cos(2*np.pi*k2/N)
fft = np.fft.fft(window*np.sin(2*np.pi*k*k2/N))
plt.plot(np.abs(windowed - fft))
plt.title('Residuals')
plt.show()
```


Windowed Fourier Transform
40
30
20
10
0
0
25
50
75
100
125
150
175
200


Residuals
1e−15
8
6
4
2
0
0
25
50
75
100
125
150
175
200

0.6 Question 6.)

0.6.1 a.)

[ ]:
```
```