1. The details of the code can be seen in "problem_1.py". Essentially, we take the FFT of the array as we normally would and then we add a phase of $e^{-2i\pi k\Delta x/N}$ where $k = 0, 1, 2..., N-1$. Δx is the desired shift in the x-axis. This phase term can be created using a delta function in real space that is centered at Δx, i.e. $\delta(x - \Delta x)$. In python this is done by having an array of 0's and adding a 1 at the index of Δx. We now plot a Gaussian that has been shifted by half the array-length (using an even number of points for simplicity):

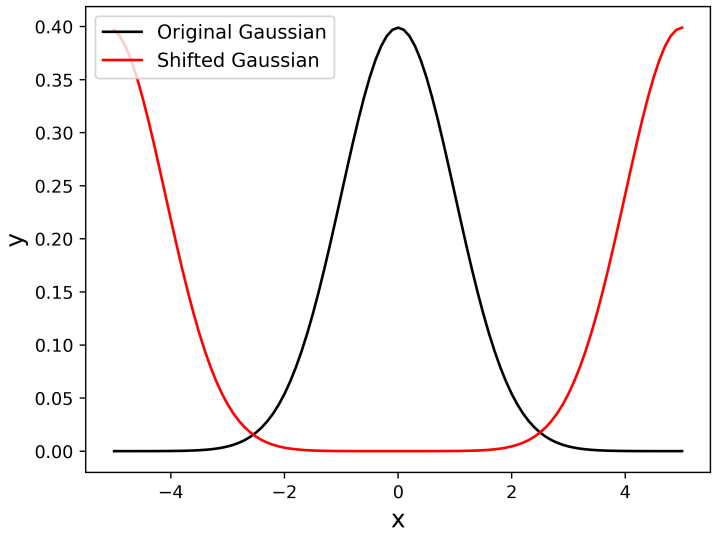


Figure 1: Graph showing a Gaussian with $\mu = 0, \sigma = 1$ from $x = [-5, 5]$ and the same Gaussian shifted by half the array length.

The Gaussian appears to be split in half, right at the peak, with both halves at the boundary as expected.

2. The details of the code can be seen in "problem_2.py". Plotting the correlation function of a Gaussian with itself, we get:

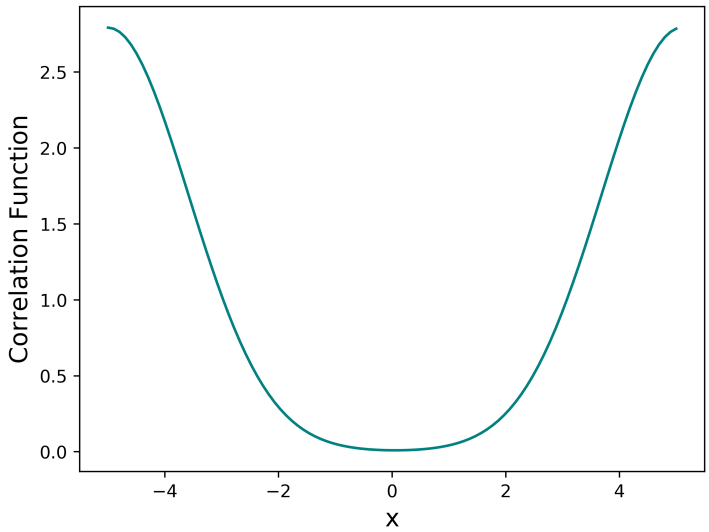


Figure 2: Graph showing the correlation function of a Gaussian ($\mu = 0, \sigma = 1$ from $x = [-5, 5]$) with itself.

3. We can make use of the functions from the last two problems to write a routine that takes the correlation function of an array with a shifted version of itself. The details of the script can be seen in "problem_3.py". We can plot a few arbitrary shifts, for example $\Delta x = \frac{1}{10}, \frac{1}{8}, \frac{1}{6}, \frac{1}{4}, \frac{1}{2}$ (as fractions of the array length):

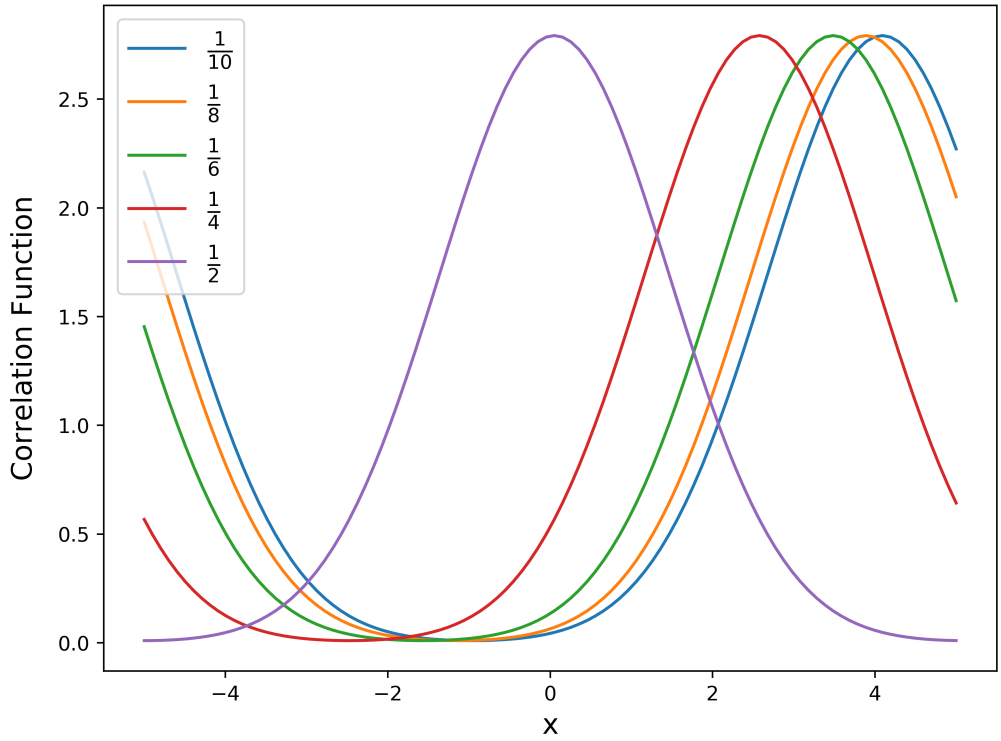
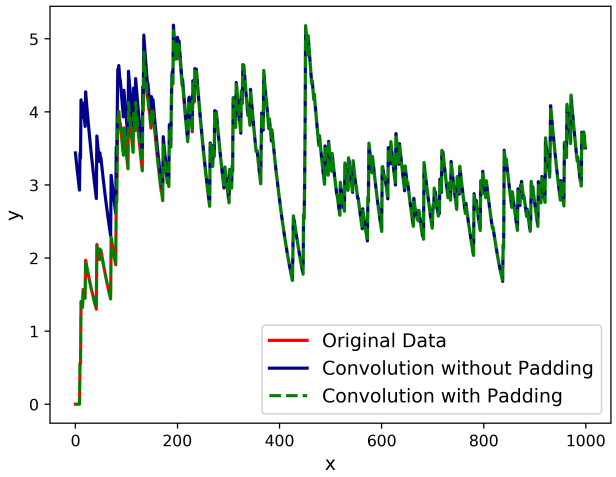


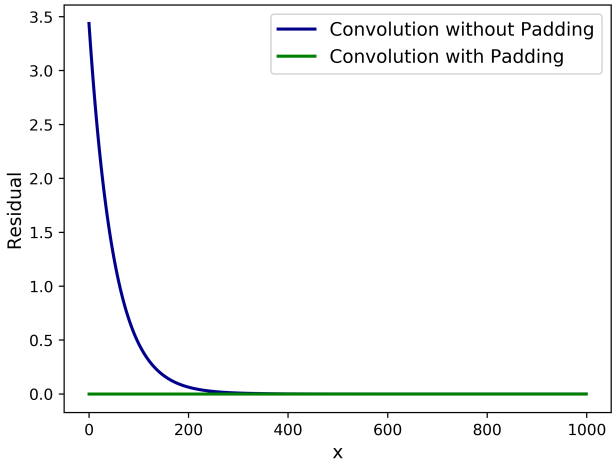
Figure 3: Graph showing the correlation function of a Gaussian ($\mu = 0, \sigma = 1$ from $x = [-5, 5]$) with a shifted version itself for various shifts.

As the shift gets larger up to $\frac{1}{2}$ of the array length, the correlation function approaches a Gaussian centered at $x = 0$. This is surprising as I would think that when the signals are the exact opposite, the correlation function would be the smallest however that seems to not be the case. Perhaps this has to do with the fact that taking the conjugate of a Fourier transform shifts the result by π in the complex plane or in other words by half the array length. Hence, an original shift of half the array length just results in the inverse Fourier transform of two Gaussian's that are stacked on top of each other, producing a spike at $x = 0$, and so on.

4. Wrapping around occurs because the DFT is periodic. As a result, if there is a jump at the end of a data set, the beginning of the transform is affected. To fix this, we can use the in-built padding provided by np.fft.fft to add 0's (equal to the length of our original arrays so the final arrays are twice their original sizes) to the end of both arrays so that there is no jump at the end of the data. We then take the convolution of both arrays as usual and then remove the second half of the data (corresponding to where all the 0's were added). The details of the script can be seen in "problem_4.py". We can test our function with the instrument example from class.



(a) Example data for our instrument.



(b) Residuals for our convolution with/without padding.

Figure 4: Plot of example data for an instrument (as done in class) produced by convolution with/without padding to compare the results.

From this we can qualitatively see that the padded input looks exactly like the original whereas the non-padded version differs from the data in the beginning as expected.

5. a) We can simplify the sum by realising that it can be expressed as the sum of a finite geometric series. The sum of a geometric series of the form $\Sigma_{k=0}^{n-1} x^k = (1-x^n)/(1-x)$. We recognise that $x = \exp\left(-2\pi i \frac{k}{N}\right)$ in our case. Therefore,

$$\sum_{x=0}^{N-1} \exp\left(-2\pi i k \frac{x}{N}\right) = \frac{1-\exp\left(-2\pi i \frac{k}{N}\right)^N}{1-\exp\left(-2\pi i \frac{k}{N}\right)}, \tag{1}$$

$$= \frac{1-\exp\left(-2\pi i k\right)}{1-\exp\left(-2\pi i \frac{k}{N}\right)}. \tag{2}$$

b) We can take the limit of the sum as $k \to 0$ by first recognising that if we take the limit of the numerator and denominator separately as $k \to 0$, they both tend to 0. Hence, we can use L'Hopital's rule as follows,

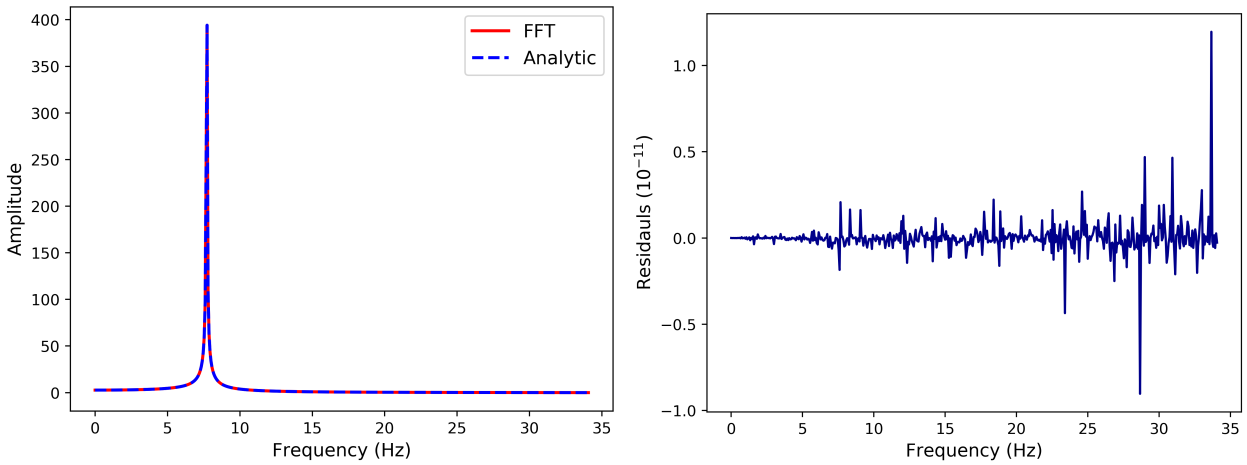
$$\lim_{k\to 0} \frac{1-\exp\left(-2\pi i k\right)}{1-\exp\left(-2\pi i \frac{k}{N}\right)} = \lim_{k\to 0} \frac{2\pi i \exp\left(-2\pi i k\right)}{2\pi i \frac{1}{N} \exp\left(-2\pi i \frac{k}{N}\right)}, \tag{3}$$

$$= N \frac{\exp\left(-2\pi i \times 0\right)}{\exp\left(-2\pi i \times \frac{0}{N}\right)}, \tag{4}$$

$$= N. \tag{5}$$

Furthermore, if we look closely at $1 - \exp(-2\pi ik)$, we recognise that if k is any integer, then $\exp -2\pi ik = 1$. As a result of this, the numerator is always 0 if k is an integer. If we look at the denominator, take $k = \alpha N$ where $\alpha \in \mathbb{R}$, therefore the denominator becomes $1 - \exp(-2\pi i\alpha)$. The exponential in the denominator is 1 only when $\alpha \in \mathbb{Z}$, otherwise $0 < \alpha < 1$. Hence, when $\alpha \in \mathbb{Z}$ the denominator is 0, and otherwise the denominator is a positive number less than 1. This means we have $\frac{0}{0}$ or $\frac{0}{\ldots} = 0$ (where ... is positive and less than 1), respectively. Therefore, the only situation when this term is not 0 is when $\alpha \in \mathbb{Z}$ or, in other words, when k is a multiple of N.

c) We arbitrarily choose a sinusoidal function with a frequency of 7.7Hz in the region $x = [-5, 10]$, such that we have a non-integer number of periods, and take $N = 1,024$. We can take the FFT using numpy and get an analytic estimate of the DFT by creating a matrix of imaginary exponential functions that we multiply by points on the sinusoid. Carrying this out in "problem_5.py" we get the following graphs. We only plot the positive frequencies for simplicity (the transform is symmetric so the negative frequencies look the same anyway).



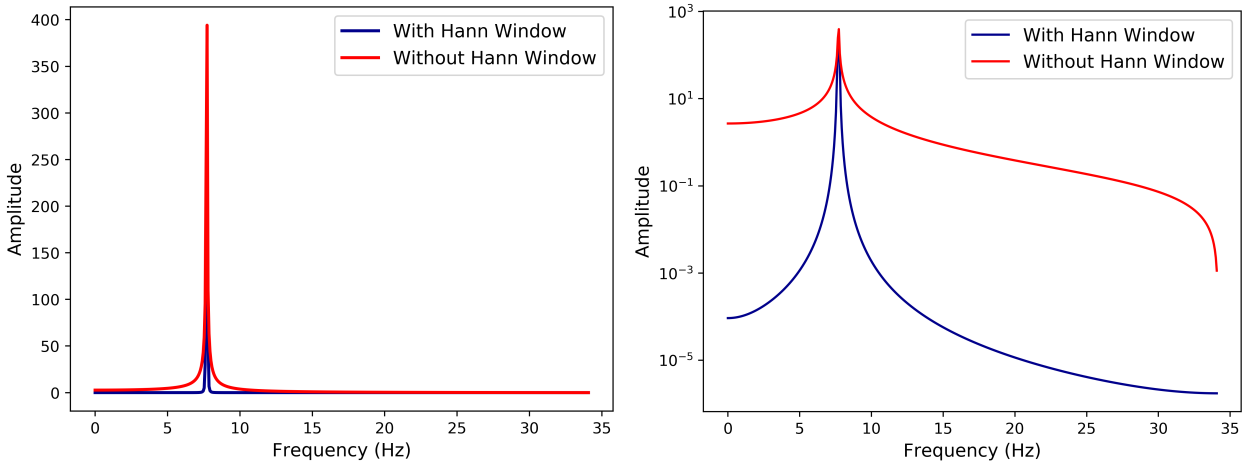
(a) Discrete Fourier transform.

(b) The residuals for our analytic estimate compared to the FFT.

Figure 5: Plot of the discrete Fourier transform of $\sin(2\pi \times 7.7x)$ produced using numpy's FFT and our analytic estimate. A residual plot is shown for comparison.

From Figure 5a, we see that our Fourier transform deviates from a delta function at $f = 7.7$Hz and appears to have wings on both sides of the peak (spectral leakage). This is the result of using a non-integer frequency and we can solve this by multiplying by a window function in the next part. Taking the mean of our residuals, we get an error of $\approx 1.46 \times 10^{-14}$ which is similar to machine precision error, showing us that our analytic is accurate.

d) We can use the well-known Hann window function, i.e. $0.5\left(1-\cos\left(\frac{2\pi x}{N}\right)\right)$. Multiplying by this window, we reduce the spectral leakage significantly. This can be seen in the following plots.



(a) Discrete Fourier transform.

(b) The log plot of the Discrete Fourier transform.

Figure 6: Plot of the discrete Fourier transform of $\sin(2\pi \times 7.7x)$ with and without applying the Hann window function to reduce the spectral leakage.

As we can see from Figure 6a, the wings of the original Fourier transform have been greatly reduced and (from Figure 6b) the amplitude drops much closer to 0 when multiplying by the Hann window. Hence, the spectral leakage has dropped significantly as a result of this application.

e) We can compute the DFT of the window analytically as follows.

$$X_k = \sum_{x=0}^{N-1} 0.5\left(1-\cos\left(\frac{2\pi x}{N}\right)\right)\exp\left(-2\pi i k\frac{x}{N}\right), \tag{6}$$

$$= 0.5\left[\sum_{x=0}^{N-1}\exp\left(-2\pi i k\frac{x}{N}\right) - 0.5\sum_{x=0}^{N-1}\left(\exp\left(-2\pi i(k+1)\frac{x}{N}\right)+\exp\left(-2\pi i(k-1)\frac{x}{N}\right)\right)\right], \tag{7}$$

$$= 0.5\left[\frac{1-\exp(-2\pi i k)}{1-\exp\left(-2\pi i\frac{k}{N}\right)} - 0.5\left(\frac{1-\exp(-2\pi i(k+1))}{1-\exp\left(-2\pi i\frac{k+1}{N}\right)}+\frac{1-\exp(-2\pi i(k-1))}{1-\exp\left(-2\pi i\frac{k-1}{N}\right)}\right)\right]. \tag{8}$$

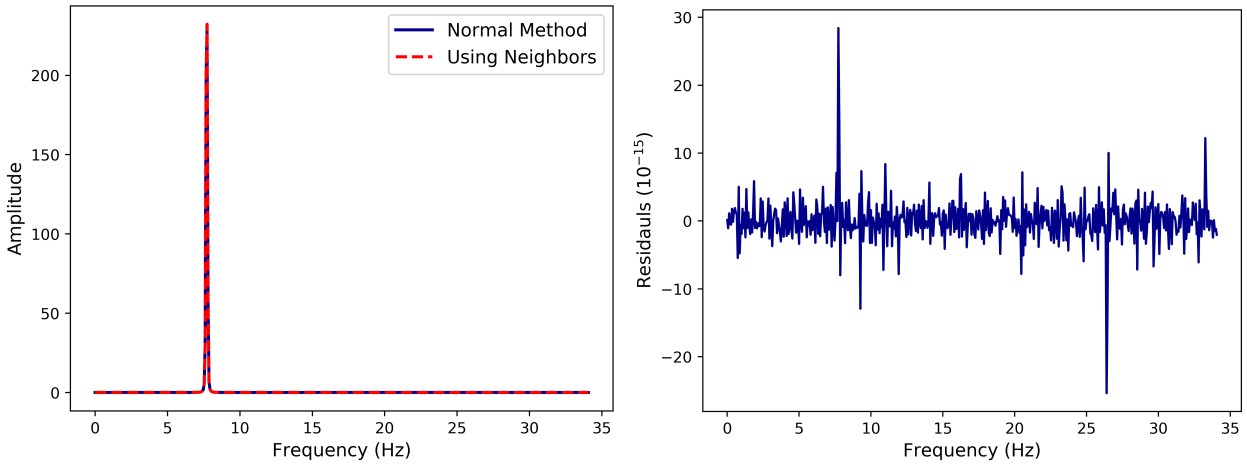
Although seemingly very complicated, it simplifies very nicely. As we know from before, if k is an integer that is not a multiple of N the answer is 0. Furthermore, as k approaches 0, the answer is just N. The two last terms have the same exact conditions, just shifted by 1 to the left or right. For example, at $k=1$ the third term approaches N while the other terms are 0, hence $X_1 = -\frac{N}{4}$. Using these two rules it is fairly easy to see that the resulting sequence where $k=[0,N-1]$ is $X=[\frac{N}{2},-\frac{N}{4},0,...,0,-\frac{N}{4}]$ (if we are looking at the amplitude the negative signs go away). We can check this in python by applying the FFT to the window. The output has been shown below. Sadly, our analytic solution is hard to check as python does not recognise that the limit exists and gives an invalid value encountered in true_divide, so the first two terms are just nan.

```
[Running] python -u "/Users/jehandastoor/Phys512/assignment4/problem_5.py"
Showing the Fourier Transform of the Window = [ 5.00000000e-01 -2.50000000e-01 -7.38579079e-18 ...  1.15467726e-18
 -7.38579079e-18 -2.50000000e-01]

[Done] exited with code=0 in 5.598 seconds
```

Figure 7: Python output showing the DFT of the Hann window for $k = [0, N-1]$.

We see that the first two values and the last value are as expected, while the other values are within machine precision (10^{-16}) and hence, we can consider them as 0. Using this we can directly calculate the windowed Fourier transform from the unwindowed version using the immediate neighbors of a point. Looking at Eq. (8), the first term is the same as we had before, where as the second and third term are a result of applying the window function. This involves subtracting $\frac{1}{4}$ of the $(k+1)$ term and $(k-1)$ term from $\frac{1}{2}$ of the original unwindowed transform. This can be done fairly easily in python. Furthermore, recognising that the DFT is circular, the first term can be calculated by taking the $(k-1)$ term to be the last term in the original transform and similar the last term can be calculated by taking the $(k+1)$ term to be the first term in the original transform. The results can be seen below,



(a) Discrete Fourier transform.

(b) Residuals to compare both methods.

Figure 8: Plot of the DFT of $\sin(2\pi \times 7.7x)$ multiplied normally by the Hann window and using the immediate neighbors of the original point in the unwindowed transform. A residual plot is shown for comparison.

Taking the average of the residuals, the difference is $\approx 5.50 \times 10^{-17}$, which is within machine precision, showing that the windowed transform can be determined in this quicker way.