Phys 512 - PS 4

Fernanda C. Rodrigues Machado

ID# 260905170

November, 2020

P1

We can shift an array making a convolution of it with a delta function:

```
def shift_fun(f, xshift):
    n = len(f)
    k = np.arange(n)
    return np.fft.ifft( np.fft.fft(f) * np.exp(-2j*np.pi*k*xshift/n) )
```

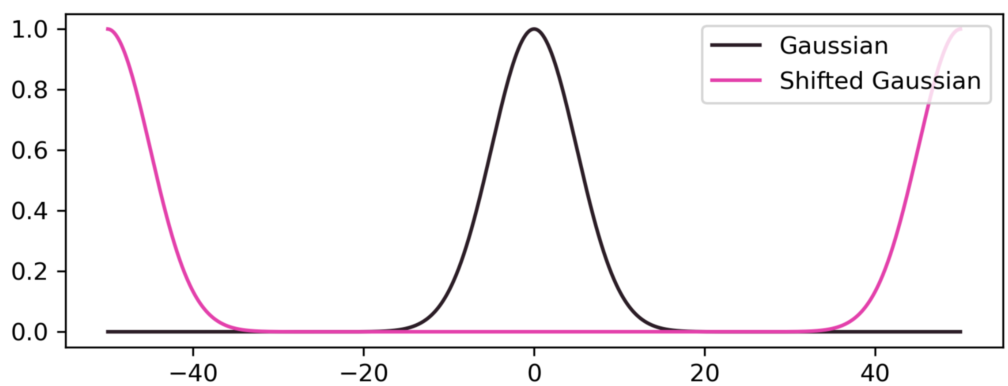

Figure 1: Convolution of a Gaussian with a delta function.

P2

Correlation function code:

```
def corr_fun(f, g):
    return np.fft.irfft( np.fft.rfft(f) * np.conj(np.fft.rfft(g)) )
```

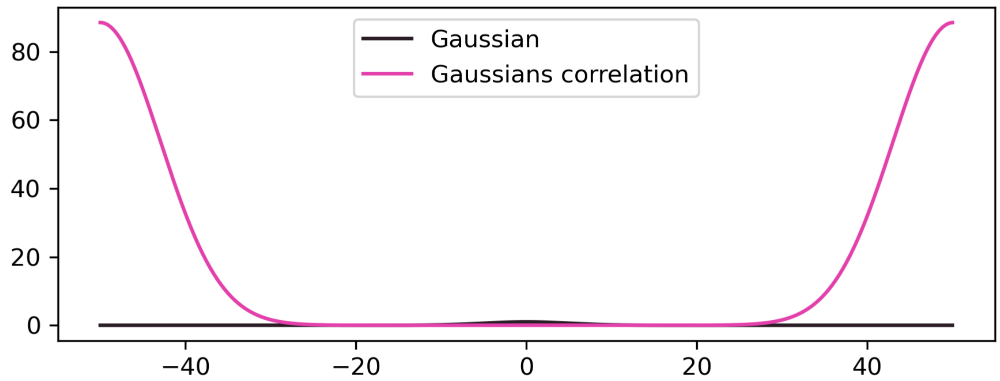

Figure 2: Correlation of a Gaussian with itself.

P3

Using the shifting (`shift_fun`) and correlation (`corr_fun`) functions defined above, the correlation of a Gaussian (shifted by an arbitrary amount) with itself can be obtained with:

```
x = np.linspace(-50,50,1000)
gauss = gaussian(x,5)
for i in range(10):
    random_shift = np.abs(np.int(np.random.randn()*500))
    shift_gauss  = shift_fun(gauss,random_shift)
    corr_gauss   = corr_fun(gauss, shift_gauss)

    shifts.append(random_shift)
    shiftgauss.append(shift_gauss)
    correlations.append(corr_gauss)
```

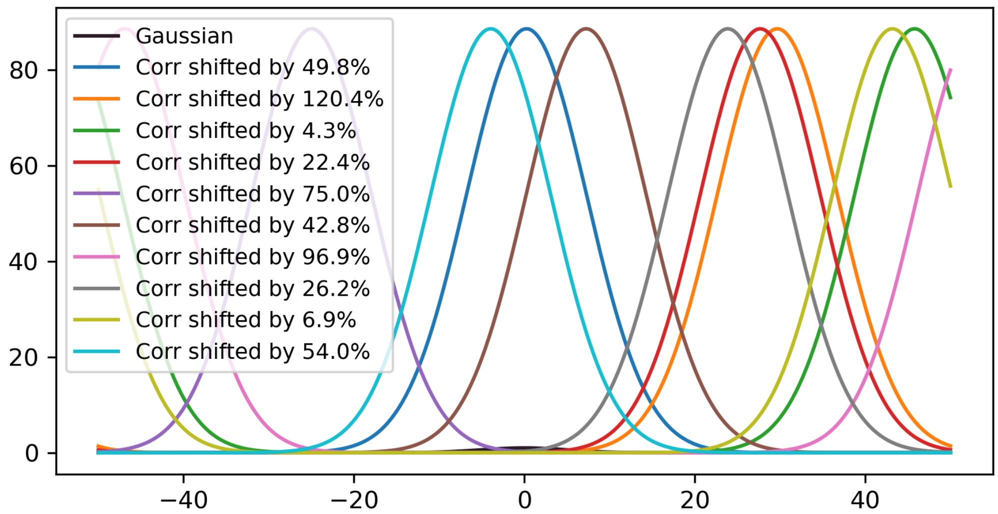

Figure 3: Correlation of a Gaussian shifted by a random amount with itself.

As expected, the center of the correlation function depends linearly on the shift of the Gaussian.

P4

We can avoid the wrapping-around of a FT by padding the function (adding zeroes to the region where the function is not defined). The convolution of two padded functions can be written as:

```
def corr_pad_fun(f, g):
    l = len(f)
    f = np.pad( f, (0,len(f)) )
    g = np.pad( g, (0,len(g)) )
    return np.fft.irfft( np.fft.rfft(f) * np.conj(np.fft.rfft(g)) )[:l]
```

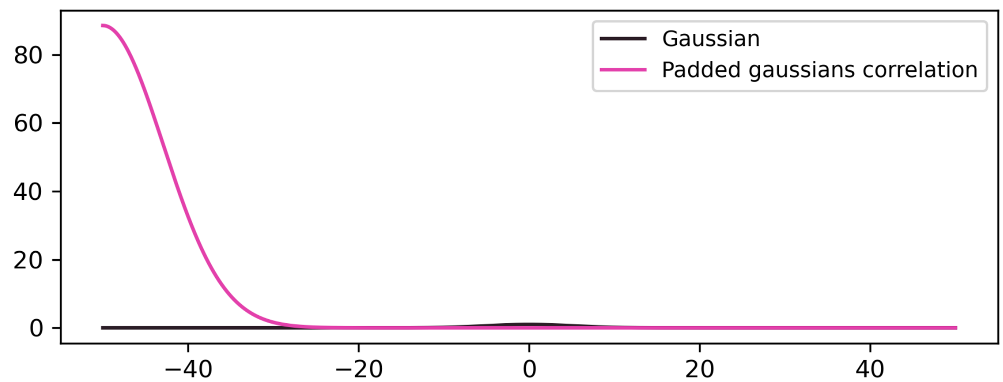

Figure 4: Correlation of two identical padded Gaussians.

P5

a)

$$\sum_{x=0}^{N-1} \mathrm{e}^{-2\pi ikx/N} = \sum_{x=0}^{N-1} \left(\mathrm{e}^{-2\pi ik/N}\right)^x \tag{1}$$

$$= \sum_{x=0}^{N-1} \alpha^x \qquad \left(\alpha = \mathrm{e}^{-2\pi ik/N}\right)$$

$$= \frac{1-\alpha^N}{1-\alpha} \qquad \text{if } |\alpha| < 1 \text{ (geometric series)}$$

$$= \frac{1-\mathrm{e}^{-2\pi ik}}{1-\mathrm{e}^{-2\pi ik/N}} \qquad \text{Q.E.D.} \tag{2}$$

b)

For $k \to 0$,

$$\lim_{k\to 0} \sum_{x=0}^{N-1} \mathrm{e}^{-2\pi ikx/N} = \sum_{x=0}^{N-1} \mathrm{e}^0$$

$$= \sum_{x=0}^{N-1} 1$$

$$= N \qquad \text{Q.E.D.}$$

For all integer k, $\mathrm{e}^{-2\pi ikx} = \mathrm{e}^0 = 1$ and the denominator of

$$\frac{1-\mathrm{e}^{-2\pi ik}}{1-\mathrm{e}^{-2\pi ik/N}}$$

is zero. As shown in Fig. 5, $\mathrm{e}^{-2\pi ik/N}$ is also equal to 1 for all integer k's that are multiples of N, resulting in a zero denominator and non-zero sum. For all $k/N \neq$integer, the denominator is non-zero and therefore the sum is zero.

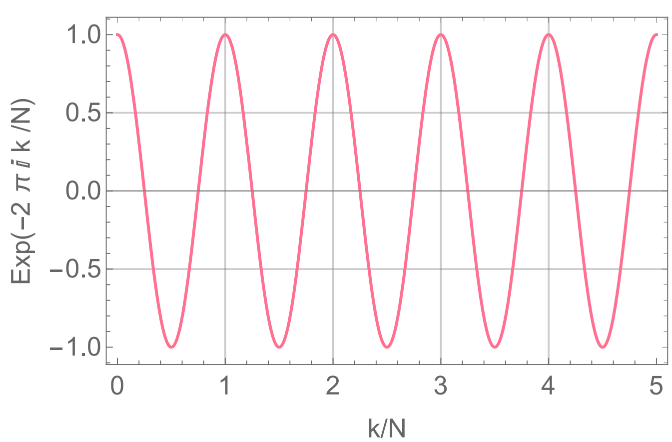

Figure 5: $\mathrm{e}^{-2\pi ik/N}$ versus k/N, determining the denominator of Eq. 2.

c)

$$
\begin{aligned}
\mathrm{DFT}\{\sin(2\pi k'x/N)\} &= \sum_{x=0}^{N-1} \sin(2\pi k'x/N)\ \mathrm{e}^{-2\pi i k x/N} \\
&= \sum_{x=0}^{N-1} \left(\frac{\mathrm{e}^{2\pi i k'x/N}}{2i} - \frac{\mathrm{e}^{-2\pi i k'x/N}}{2i} \right)\ \mathrm{e}^{-2\pi i k x/N} \\
&= \sum_{x=0}^{N-1} \frac{1}{2i} \left(\mathrm{e}^{-2\pi i(k-k')x/N} - \mathrm{e}^{-2\pi i(k+k')x/N} \right) \qquad (3) \\
&= \frac{1}{2i} \left(\frac{1-\mathrm{e}^{-2\pi i(k-k')}}{1-\mathrm{e}^{-2\pi i(k-k')/N}} - \frac{1-\mathrm{e}^{-2\pi i(k+k')}}{1-\mathrm{e}^{-2\pi i(k+k')/N}} \right)
\end{aligned}
$$

Comparing to Eq. 1 and using the findings from **5b**, the sum is equal to $N/2$ when $k' = \pm k$ and equal to zero for all other k'.

We can generate the analytical DFT of a sine (Eq. 3) with the following code:

```
def myfft(N,k):
    x    = np.arange(N)
    kvec = np.arange(N)
    FT = []
    for K in kvec:
        FTK = np.sum(1/(2J)*(np.exp(-2J*np.pi*(K-k)*x/N)-np.exp(-2J*np.pi*(K+k)*x/N)))
        FT.append(np.abs(FTK))
    return kvec, FT

n = 256
ks     = []; fts     = []
k_ints = []; ft_ints = []

for i in range(10):
    k = np.abs(np.random.randn()*100)
    ks.append(k)
    fts.append(myfft(n,k))

    k_int = np.int(k)
    k_ints.append(k_int)
    ft_ints.append(myfft(n,k_int))
```

In Fig. 6 we can see that the analytical form of DFT$\{\sin(2\pi k' x/N)\}$ given by Eq. 3 resembles a delta function for $k' = k$, but we see spectral leakage due to the non-integer values of k (negative values $k' = -k$ are folded in the spectrum). It is interesting to observe that, the closer k gets to an integer number, the closer the DFT is to a delta function, as observed in the green and lavender curves. If we take integer values of k, the DFT is indeed a delta function, as can be seen in Fig 7.

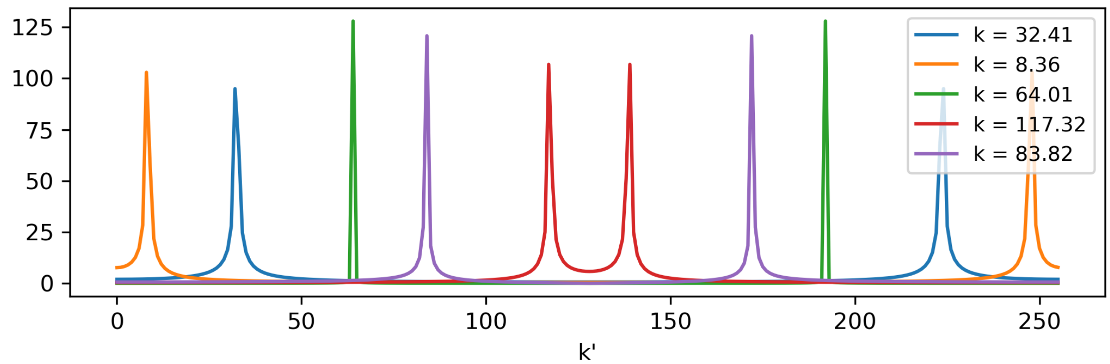

Figure 6: Analytical DFT of $\sin(2\pi k' x/N)$ for random non-integer k.

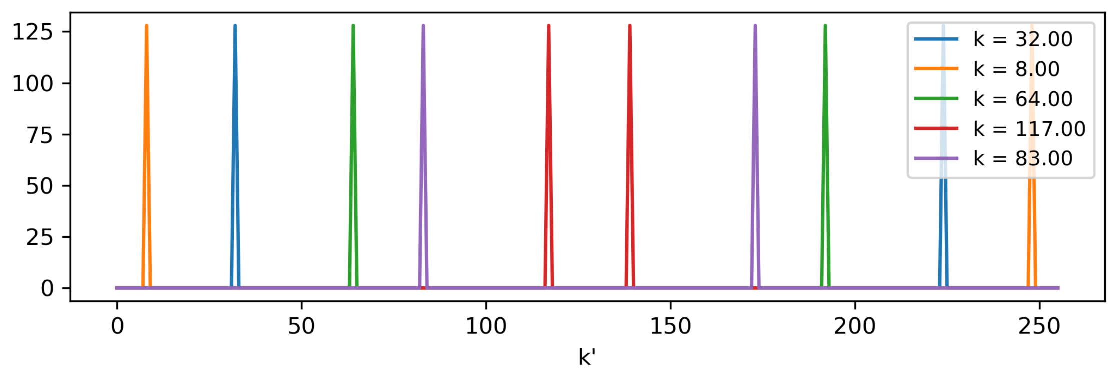

Figure 7: Analytical DFT of $\sin(2\pi k' x/N)$ for random integer k.

d)

Using a window $w = 0.5 - 0.5\cos(2\pi x/N)$ with numpy.fft reduces the spectral leakage, however it still doesn't give exactly a delta function (Fig. 8). Also, the amplitude was cut in half, so I might have done some mistake somewhere.

```
y = np.sin(2 * np.pi *x*k/n)
w = 0.5 - 0.5*np.cos(2 * np.pi *x/n)
ftnp       = np.fft.fft(y)
ftnpwindow = np.fft.fft(y*w)
```

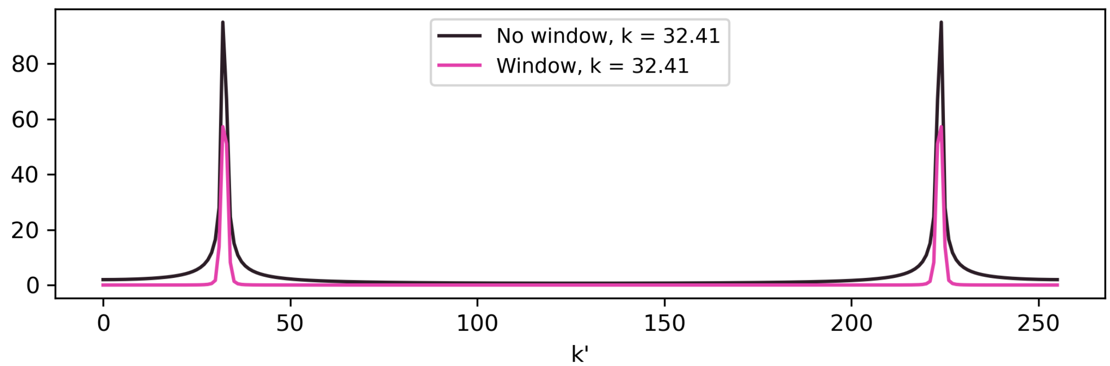

Figure 8: Numpy.fft of $\sin(2\pi k' x/N)$ for random non-integer k, with and without a window $w = 0.5 - 0.5\cos(2\pi x/N)$.

e)

Figure 9 shows the discrete Fourier transform of the window $w = 0.5 - 0.5\cos(2\pi x/N)$ for $N = 256$. We can clearly see that the amplitude of the DFT is equal to $N/2$ for $k = 0$, $-N/4$ for $k = \pm 1$ and zero otherwise.

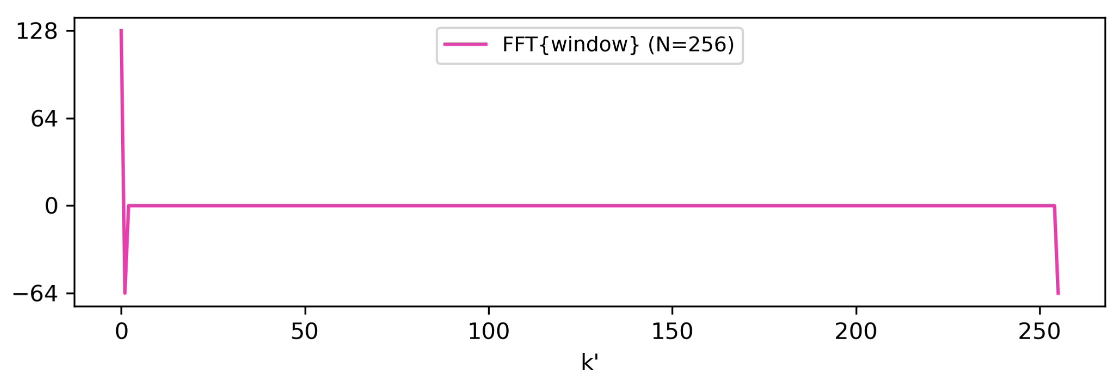

Figure 9: DFT (with numpy.fft) of a window $w = 0.5 - 0.5\cos(2\pi x/N)$ for $N = 256$.