Guillaume Payeur (260929164)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 200
plt.rcParams.update({"text.usetex": True})
```

Q1

Here we use the fact that the FT of a shifted f(x) is

$$F(k) = \exp(2\pi ikdx/N)\sum f(x)\exp(-2\pi ikx/N), \tag{1}$$

So to shift the array we need to multiply the FT by a phase and then take the IFT. We can do that via a convolution, using the convolution theorem

$$f * g = \mathrm{IFT}(\mathrm{FT}(f)\mathrm{FT}(g)) \tag{2}$$

Letting our initial array be f, what we need to do is find a function g which has a fourier transform that is just

$$G(k) = \exp(2\pi ikdx/N) \tag{3}$$

The function in question is

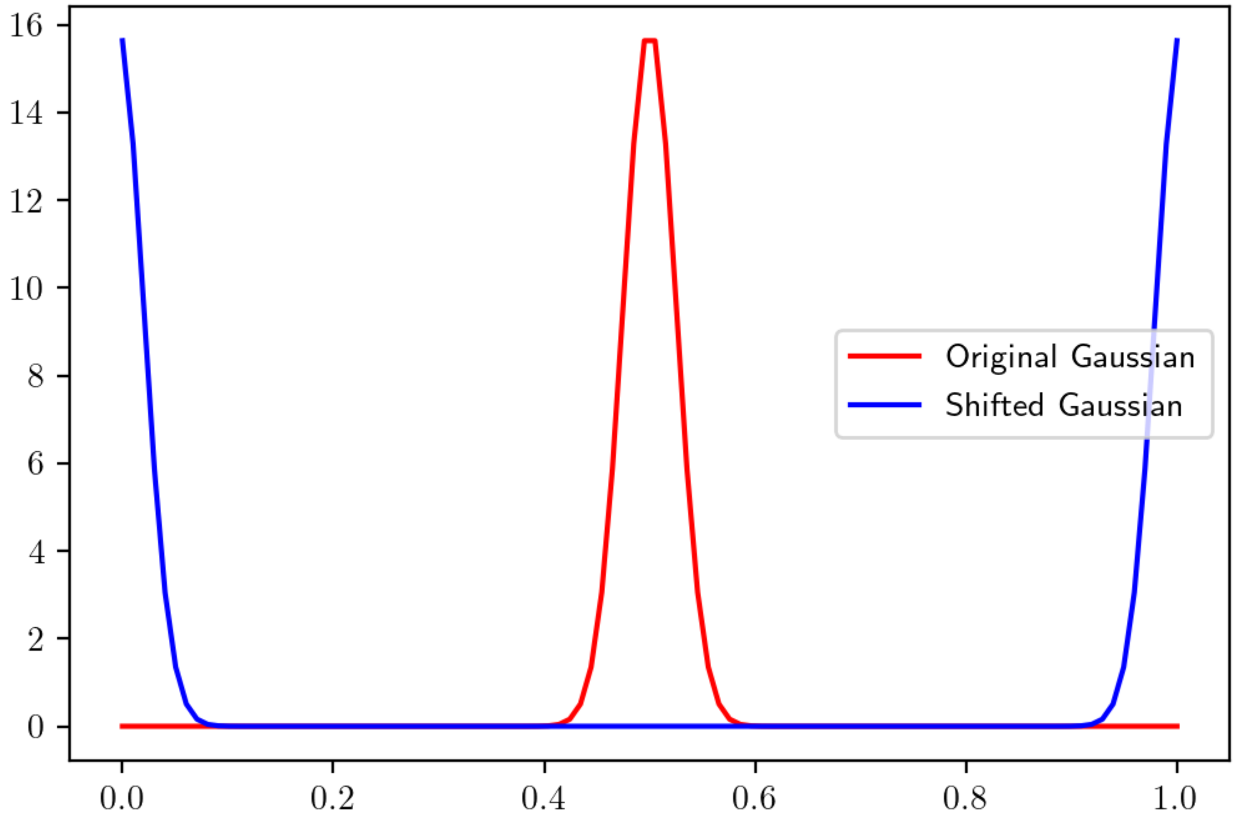
$$g(x) = \delta(dx - x) \tag{4}$$

In [2]:
```
# Function to do discrete convolution with circular padding
def conv(x1, x2, N):
    n, m = np.ogrid[:N, :N]
    return (x1[:N] * x2[(n - m) % N]).sum(axis=1)

# Function to shift array by ammount dx
def shift(f,dx):
    dx = int(dx)
    g = np.zeros((f.shape[0]))
    g[-dx] = 1
    shifted_f = conv(f,g,int(f.shape[0]))
    return shifted_f
```

In [3]:
```
# Making a Gaussian array
x = np.linspace(0,1,100)
sigma = 0.025
mu = 0.5
gaussian = 1/(np.sqrt(2*np.pi)*sigma)*np.exp((-1/2)*(x-mu)**2/sigma**2)
# Shifting the Gaussian array
shifted_gaussian = np.real(shift(gaussian,gaussian.shape[0]/2))
```

In [4]:
```
# Plotting both arrays
plt.plot(x,gaussian,color='red',label='Original Gaussian')
plt.plot(x,shifted_gaussian,color='blue',label='Shifted Gaussian')
plt.legend(frameon=True)
```

Out[4]: <matplotlib.legend.Legend at 0x276f69139c8>



Q2

a)

We implement the correlation function as

$$f \star g = \mathrm{IFT}(\mathrm{FT}(f)\mathrm{FT}(g)^*) \tag{5}$$

In [5]:
```
# Function to take correlation function of two arrays
def corr(f,g):
    F = np.fft.fft(f)
    G = np.fft.fft(g)
    G_conj = np.conjugate(G)
    corr = np.fft.ifft(F*G_conj)
    return np.real(corr)
```

In [6]:
```
# Making a Gaussian array
x = np.linspace(0,1,100)
sigma = 0.025
mu = 0.5
```

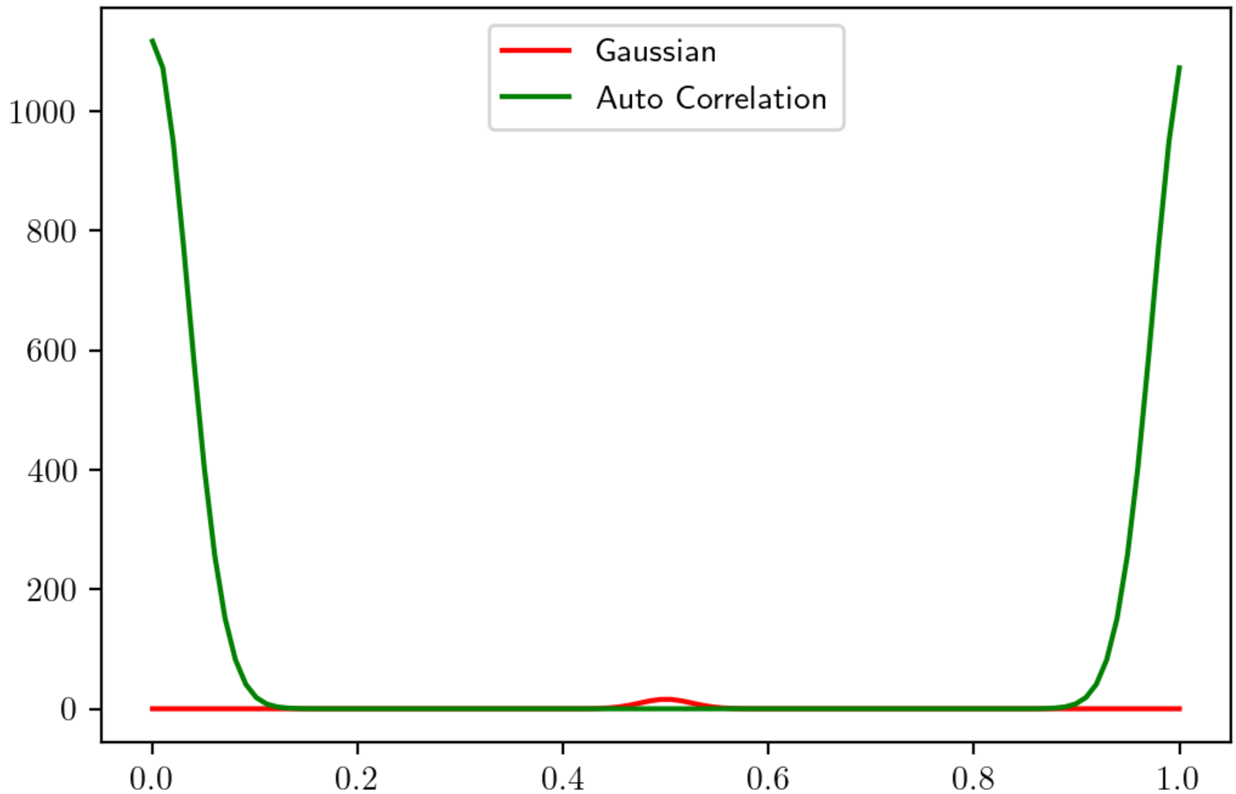
```
gaussian = 1/(np.sqrt(2*np.pi)*sigma)*np.exp((-1/2)*(x-mu)**2/sigma**2)
# Making the correlation function of the gaussian with itself
gaussian_corr = corr(gaussian,gaussian)
```

In [7]:

```
# Plotting the gaussian and its autocorrelation
plt.plot(x,gaussian,color='red',label='Gaussian')
plt.plot(x,gaussian_corr,color='green',label='Auto Correlation')
plt.legend(frameon=True)
```

Out[7]:

```
<matplotlib.legend.Legend at 0x276f745d8c8>
```



As expected the correlation function is largest at $x = 0$ and decreases away from $x = 0$. It is also Gaussian.

b)

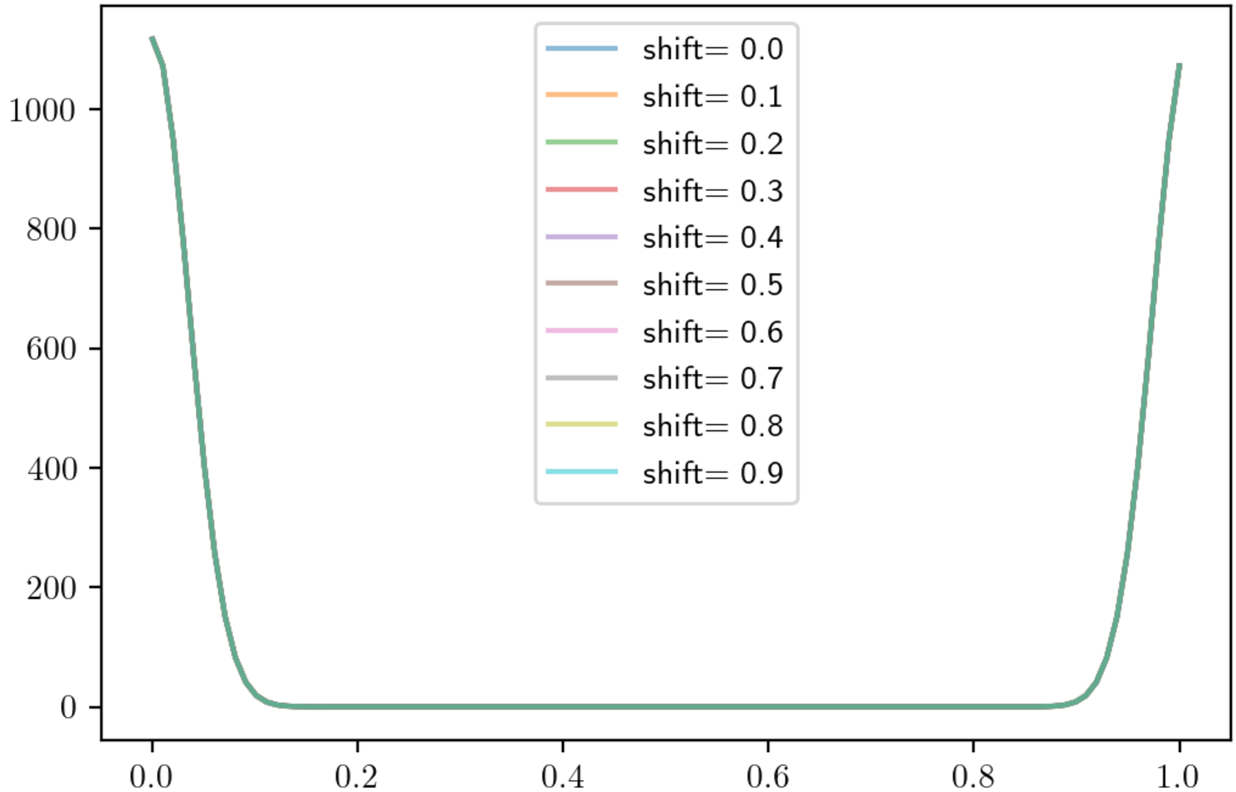
Now we plot the correlation function of a shifted gaussian with itself, for various values of shift

In [8]:

```
for dx in np.arange(0,gaussian.shape[0],10):
    shifted_gaussian = shift(gaussian,dx)
    gaussian_corr = corr(shifted_gaussian,shifted_gaussian)
    plt.plot(x,gaussian_corr,label='shift= {}'.format(dx/gaussian.shape[0]),alpha=0.5)
plt.legend(frameon=True)
```

Out[8]:

```
<matplotlib.legend.Legend at 0x276f74b5348>
```



The correlation function does not change as a function of the shift, as expected.

Q3

We write a function that takes a FFT using the convolution theorem

$$f * g = \mathrm{IFT}(\mathrm{FT}(f)\mathrm{FT}(g)) \tag{6}$$

but we pad f and g with zeros and keep the center of the resulting convolution

In [9]:
```
def conv(f,g):
    # padding f and g with zeros
    f_padded = np.zeros((3*f.shape[0]-2))
    f_padded[f.shape[0]-1:2*f.shape[0]-1] = f
    g_padded = np.zeros((3*g.shape[0]-2))
    g_padded[g.shape[0]-1:2*g.shape[0]-1] = g

    # using convolution theorem
    F = np.fft.fft(f_padded)
    G = np.fft.fft(g_padded)
    convolution = np.fft.ifft(F*G)
    if f.shape[0]%2 == 0:
        convolution = np.concatenate((convolution[int(-f.shape[0]//2)-1:],convolution[:
    else:
        convolution = np.concatenate((convolution[int(-f.shape[0]//2):],convolution[:in
    return np.real(convolution)
```

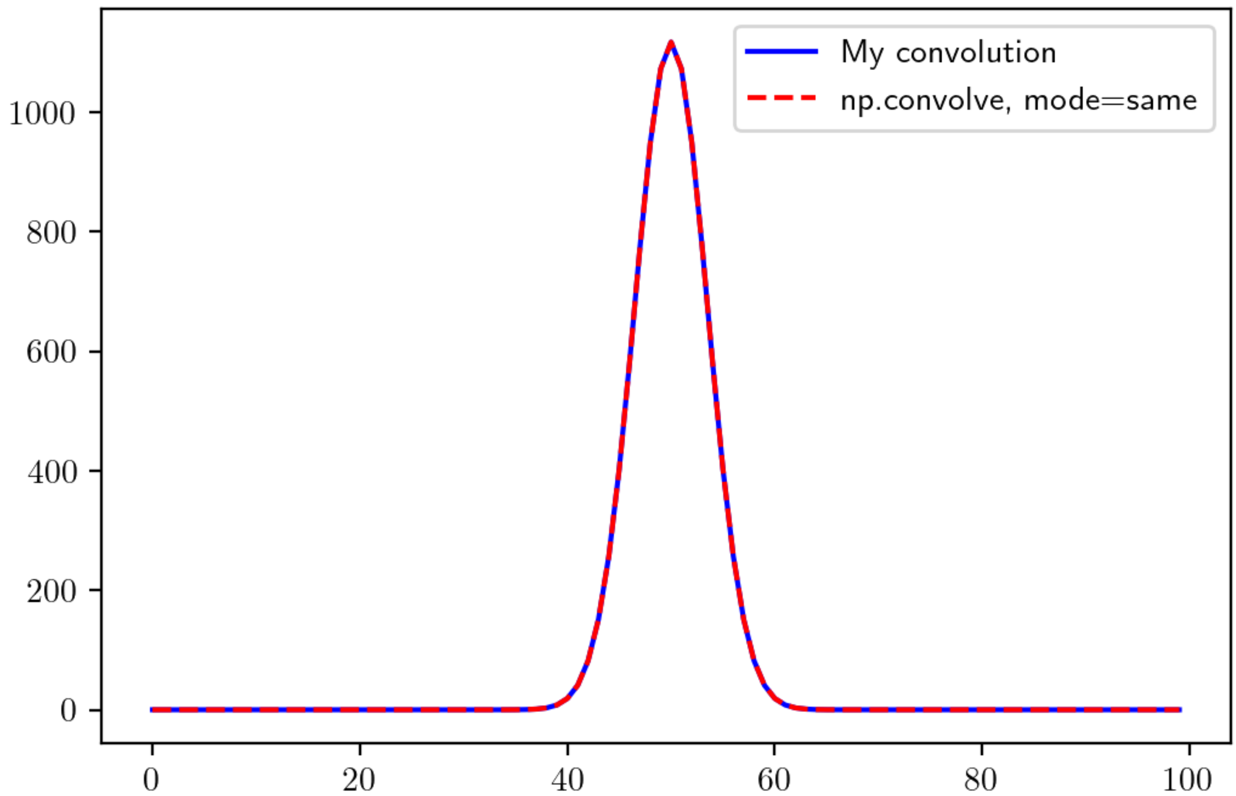
In [10]:
```
x = np.linspace(0,1,100)
sigma = 0.025
```

```
mu = 0.5
gaussian = 1/(np.sqrt(2*np.pi)*sigma)*np.exp((-1/2)*(x-mu)**2/sigma**2)
```

In [11]:

```
plt.plot(conv(gaussian,gaussian),label='My convolution',color='blue')
plt.plot(np.convolve(gaussian,gaussian,mode='same'), label='np.convolve, mode=same',col
plt.legend(frameon=True)
```

Out[11]: <matplotlib.legend.Legend at 0x276f3815dc8>



Q4

a)

Write

$$
\begin{aligned}
\sum_{x=0}^{N-1} \exp(-2\pi ikx/N) &= \sum_{x=0}^{N-1} \exp(-2\pi ik/N)^x && (7)\\
&= \frac{1-\exp(-2\pi ik/N)^{(N-1+1)}}{1-\exp(-2\pi ik/N)} && (8)\\
&= \frac{1-\exp(-2\pi ik)}{1-\exp(-2\pi ik/N)} && (9)
\end{aligned}
$$

b)

Using L'Hopital's rule we get for $k \to 0$

$$\lim_{k\to 0}\sum_{x=0}^{N-1}\exp(-2\pi ikx/N) = \lim_{k\to 0}\frac{\frac{d}{dk}(1-\exp(-2\pi ik))}{\frac{d}{dk}(1-\exp(-2\pi ik/N))} \tag{10}$$

$$= \lim_{k\to 0}\frac{2\pi ik\exp(-2\pi ik)}{2\pi ik/N\exp(-2\pi ik/N)} \tag{11}$$

$$= N \tag{12}$$

while for any integer k that is not a multiple of N, the numerator in

$$\sum_{x=0}^{N-1}\exp(-2\pi ikx/N) = \frac{1-\exp(-2\pi ik)}{1-\exp(-2\pi ik/N)} \tag{13}$$

is 0, while the denominator is not, so that the net result is

$$\sum_{x=0}^{N-1}\exp(-2\pi ikx/N) = 0 \tag{14}$$

c)

We can compute analytically the DFT of a sine wave with angular frequency k_0 as such

$$F(k) = \sum_{x=0}^{N-1} sin(k_0 2\pi x/N)e^{-2\pi ikx/N} \tag{15}$$

$$= \sum_{x=0}^{N-1}\frac{i}{2}\left(e^{ik_0 2\pi x/N} - e^{-ik_0 2\pi x/N}\right)\left(e^{-2\pi ikx/N}\right) \tag{16}$$

$$= \frac{i}{2}\sum_{x=0}^{N-1} e^{-2\pi i(k-k_0)x/N} - e^{-2\pi i(k+k_0)x/N} \tag{17}$$

$$= \frac{i}{2}\left(\frac{1-e^{-2\pi i(k-k_0)}}{1-e^{-2\pi i(k-k_0)/N}} - \frac{1-e^{-2\pi i(k+k_0)}}{1-e^{-2\pi i(k+k_0)/N}}\right) \tag{18}$$

Now we compare this to the DFT obtained with a FFT

In [12]:

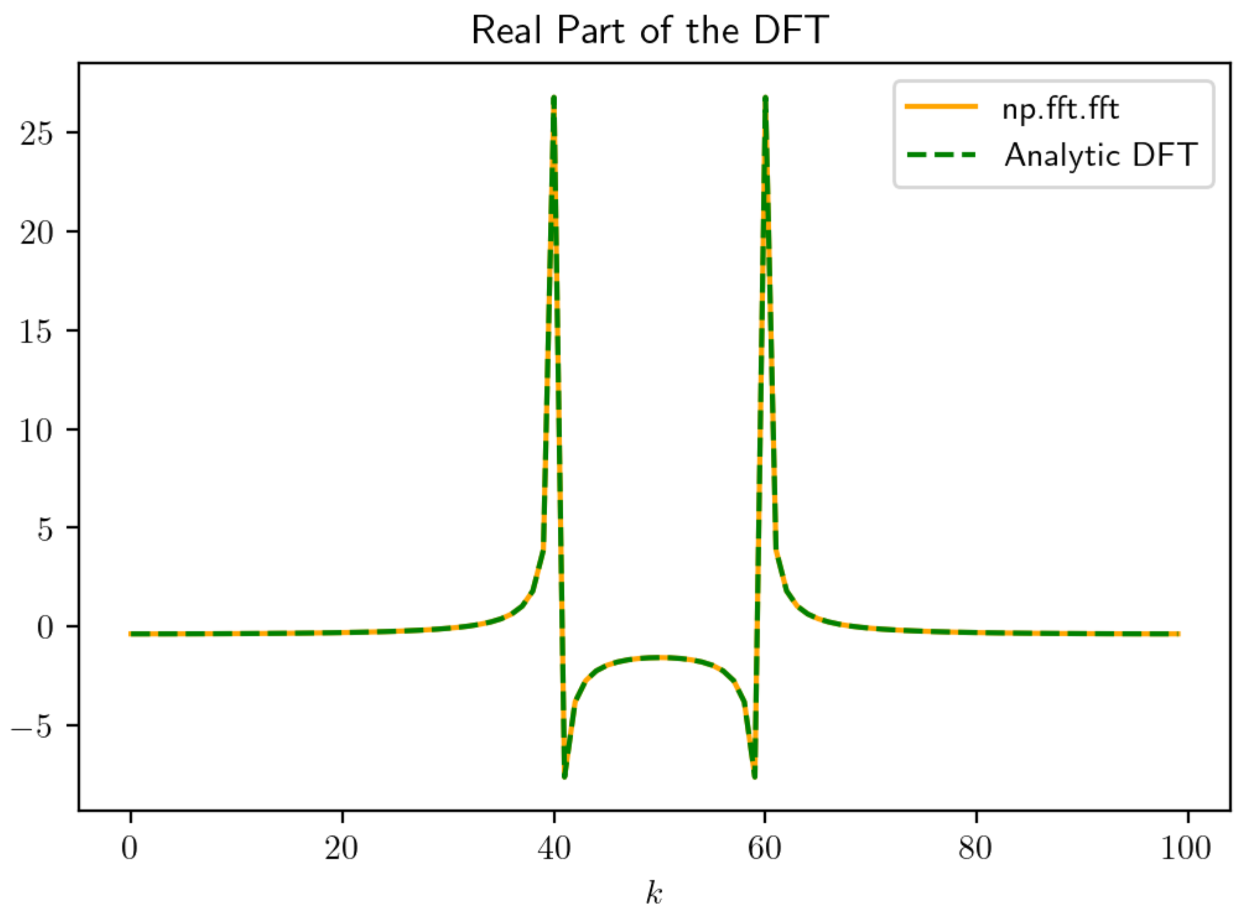
```
def analytic_DFT(k,k0,N):
    return (-1j/2)*(1-np.exp(-2*np.pi*1j*(k-k0)))/(1-np.exp(-2*np.pi*1j*((k-k0)/N))) -

# Making a sine array
N = 100
x = np.linspace(0,1,N+1)[0:N]
k = np.arange(N)
k0 = 40.2
sine = np.sin(k0*x*2*np.pi)
```

In [13]:

```
plt.plot(k,np.real(np.fft.fft(sine)),label='np.fft.fft',color='orange')
plt.plot(k,np.real(analytic_DFT(k,k0,N)),label='Analytic DFT',color='green',ls='--')
plt.title('Real Part of the DFT')
```

```
plt.xlabel('$k$')
plt.legend(frameon=True)
```

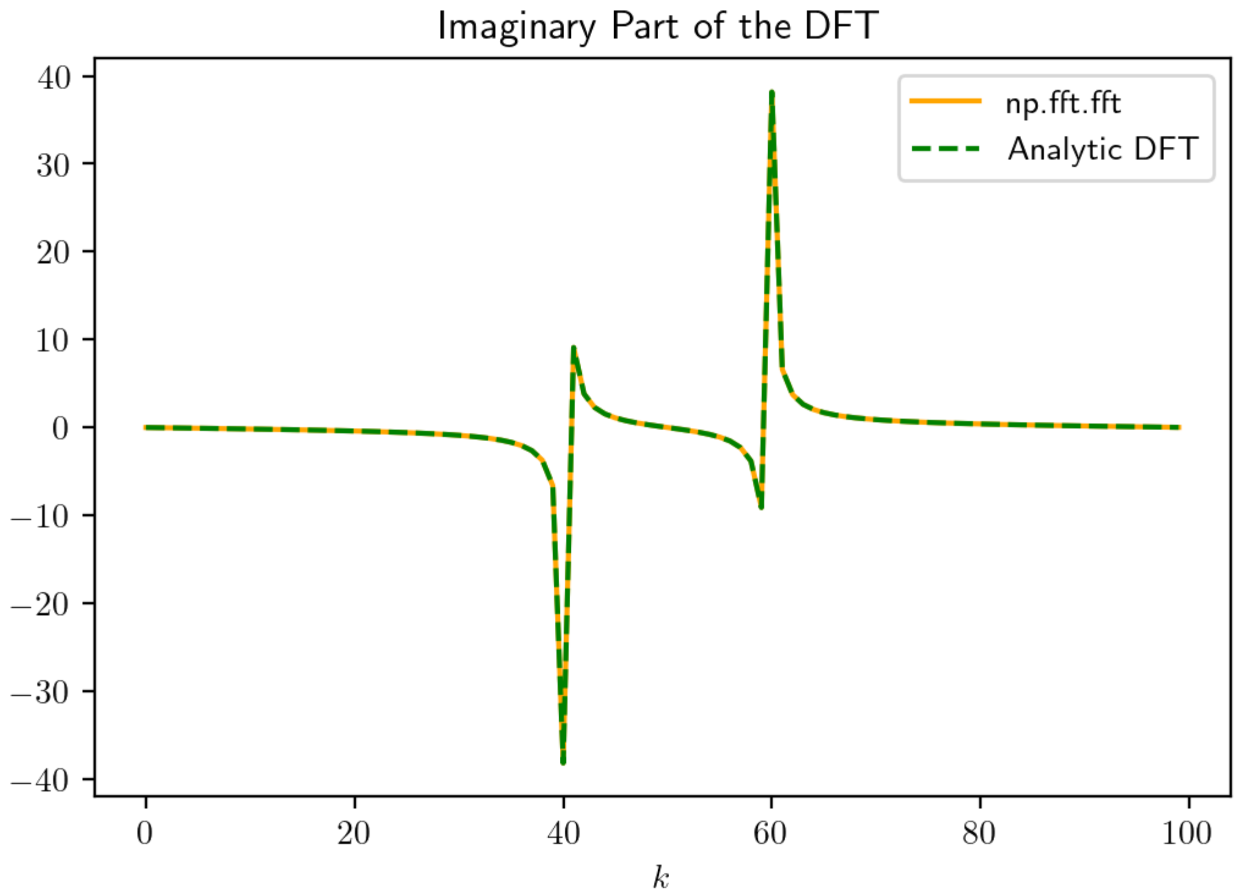
Out[13]: <matplotlib.legend.Legend at 0x276f7c4f608>



In [14]:

```
plt.plot(k,np.imag(np.fft.fft(sine)),label='np.fft.fft',color='orange')
plt.plot(k,np.imag(analytic_DFT(k,k0,N)),label='Analytic DFT',color='green',ls='--')
plt.title('Imaginary Part of the DFT')
plt.xlabel('$k$')
plt.legend(frameon=True)
```

Out[14]: <matplotlib.legend.Legend at 0x276f7cde488>



They look identical, and the mean absolute difference is

In [15]:

```
print(np.mean(np.abs(analytic_DFT(k,k0,N)-np.fft.fft(sine))))
```

```
9.570381475201317e-14
```

which is roughly machine precision.

d)

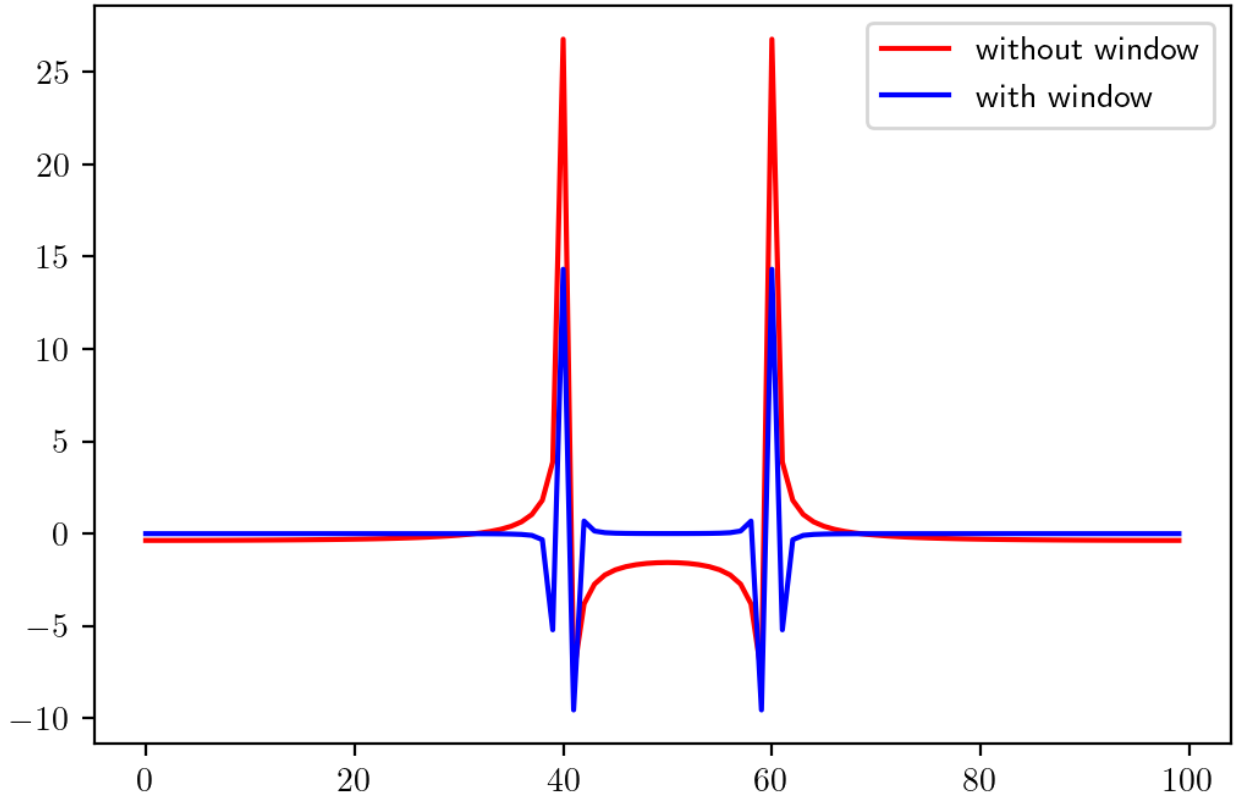
I multiply by a window and plot the FT side by side with the FT that didn't use a window

In [16]:

```
# cosine window
def window(x,N):
    return 0.5-0.5*np.cos(2*np.pi*x)

plt.plot(k,np.real(np.fft.fft(sine)),color='red',label='without window')
plt.plot(k,np.real(np.fft.fft(sine*window(x,N))),color='blue',label='with window')
plt.legend(frameon=True)
```

Out[16]:

```
<matplotlib.legend.Legend at 0x276f7daad08>
```



Indeed the spectral leakage has decreased a lot.

e)

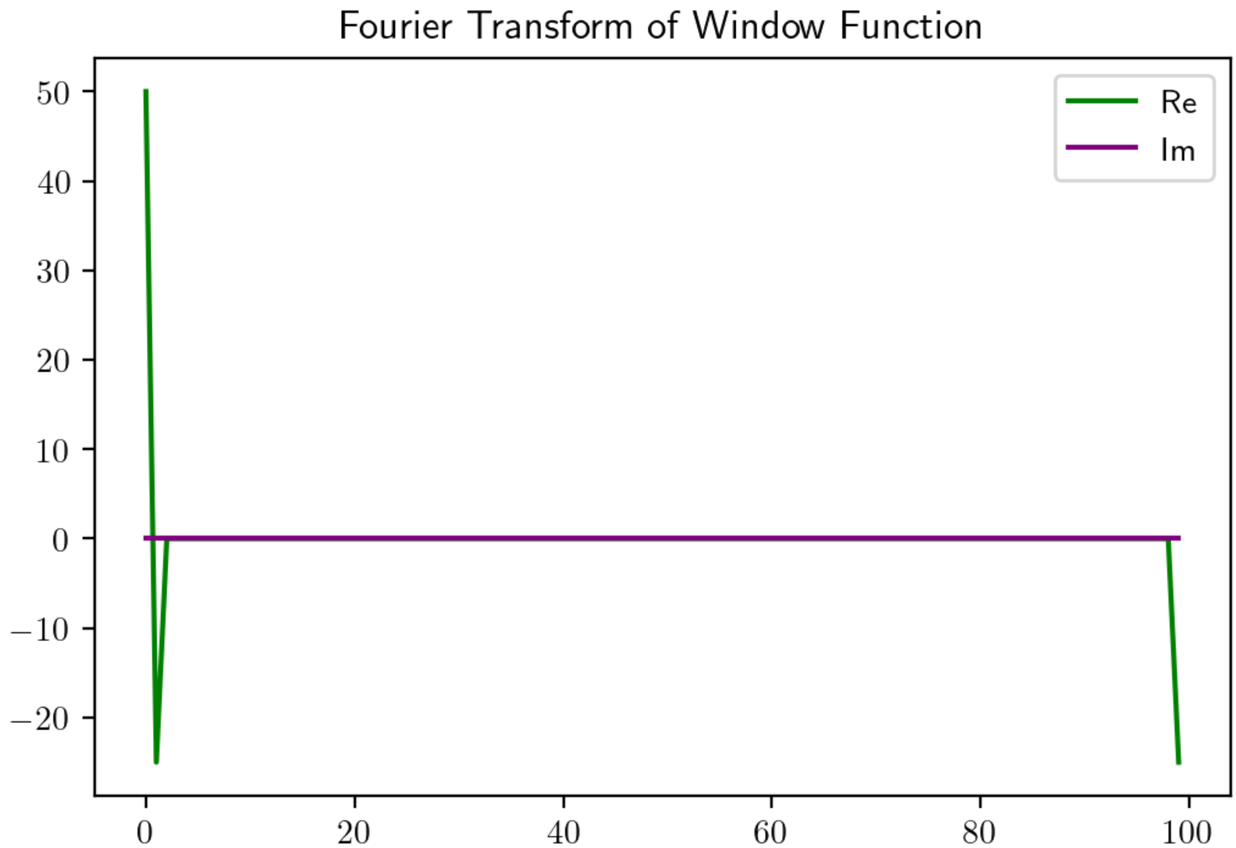
We fourier transform the window function and plot/print the result

In [17]:

```
plt.plot(np.real(np.fft.fft(window(x,N))),color='green',label='Re')
plt.plot(np.imag(np.fft.fft(window(x,N))),color='purple',label='Im')
plt.title('Fourier Transform of Window Function')
plt.legend(frameon=True)
```

Out[17]:

```
<matplotlib.legend.Legend at 0x276f7e2be48>
```



In [18]:
```
print(np.round(np.real(np.fft.fft(window(x,N))),2))
print(np.round(np.imag(np.fft.fft(window(x,N))),2))
print(N)
```

```
[ 50. -25.  -0.  -0.   0.  -0.  -0.   0.   0.   0.  -0.  -0.   0.   0.
   0.   0.  -0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  -0.  -0.
  -0.   0.   0.   0.   0.   0.  -0.  -0.   0.   0.   0.   0.   0.   0.
  -0.   0.  -0.   0.  -0.  -0.  -0.   0.   0.   0.  -0.  -0.  -0.   0.
  -0.  -0.  -0.   0.   0.  -0.   0.   0.   0.   0.  -0.   0.   0.  -0.
   0.   0.  -0.   0.  -0.  -0.   0.  -0.   0.   0.   0.  -0.   0.   0.
  -0.   0.   0.   0.   0.  -0.  -0.  -0.   0.   0.  -0.  -0.   0.  -0.
  -0. -25.]
[ 0.  0. -0. -0. -0.  0. -0. -0.  0.  0. -0. -0. -0. -0.  0.  0. -0.  0.
  0. -0.  0.  0.  0. -0. -0.  0.  0.  0. -0.  0. -0. -0. -0. -0.  0.  0.
 -0. -0.  0. -0.  0.  0.  0. -0. -0. -0.  0. -0. -0.  0.  0. -0.  0.  0.
 -0.  0.  0.  0. -0.  0.  0.  0. -0.  0.  0. -0. -0. -0.  0.  0.  0. -0.
  0. -0. -0. -0.  0.  0. -0.  0.  0.  0. -0. -0.  0. -0. -0.  0.  0. -0.
  0.  0. -0.  0.  0. -0.  0.  0.  0. -0.]
100
```

So we get that the fourier transform of the window is

$$[N/2, -N/4, 0, \ldots, 0, -N/4] \tag{19}$$

\ Now, as a result of the Convolution theorem,

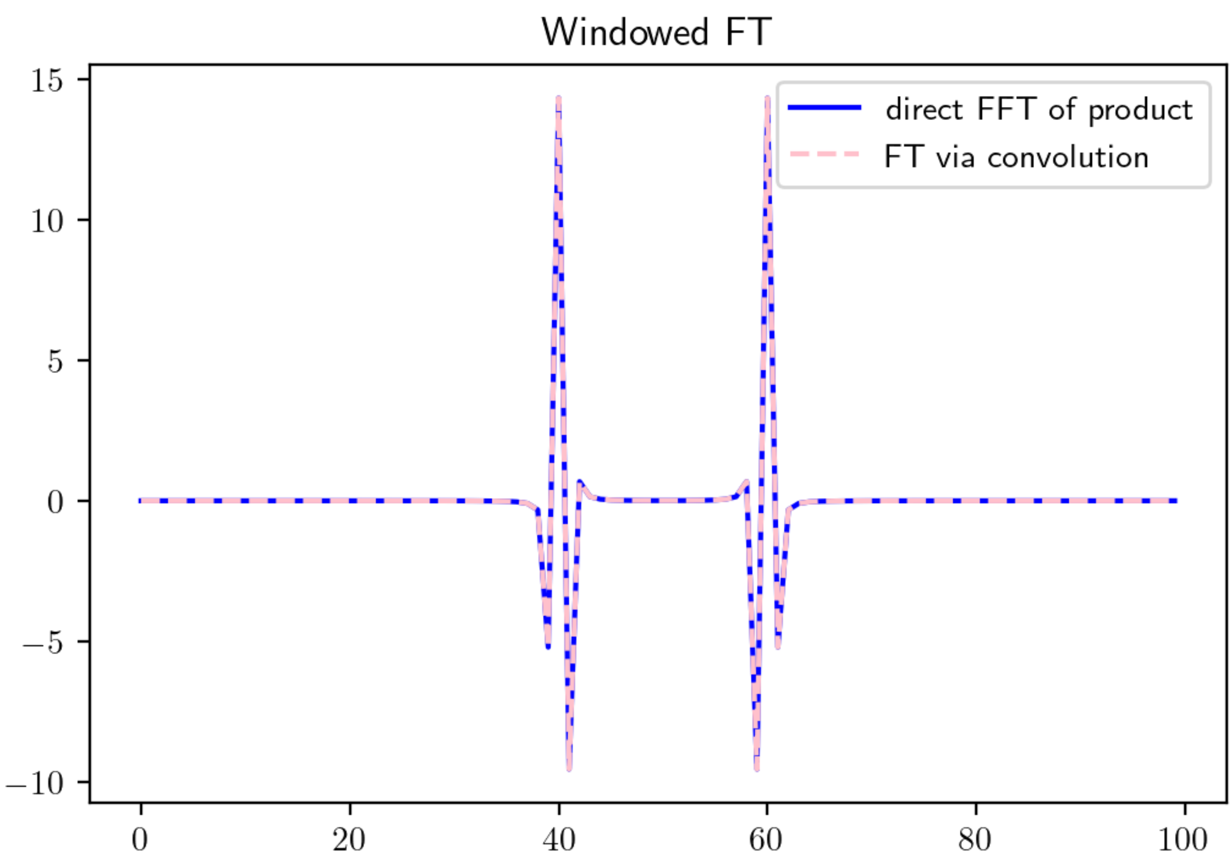
$$FT(f.g) = F * G \tag{20}$$

So letting f be our sine wave, and g be our window function, we see that we can take the FT of their product by taking the convolution of their FT. The convolution at any poiny will only involve neighboring points as a result of the convolution filter being [N/2, -N/4, 0, \dots, 0, -N/4].

In [19]:
```
# Function to do discrete convolution with circular padding
def conv(x1, x2, N):
    n, m = np.ogrid[:N, :N]
    return (x1[:N] * x2[(n - m) % N]).sum(axis=1)

# Getting the fourier transform including the window function using a convolution
FTsine = np.fft.fft(sine)
FTwindow = np.fft.fft(window(x,N))
FT = np.real(conv(FTsine,FTwindow,N))/N
```

In [20]:
```
plt.plot(k,np.real(np.fft.fft(sine*window(x,N))),color='blue',label='direct FFT of prod
plt.plot(FT,color='pink',label='FT via convolution',ls='--')
plt.title('Windowed FT')
plt.legend(frameon=True)
```

Out[20]:
```
<matplotlib.legend.Legend at 0x276f7e902c8>
```



we indeed got the same thing as before.

Q5

In [21]:
```
dir_ligo = 'ligo_data'
```

a)

To get a noise model for the Hangford detectors I

1. Load all 4 GW observations
2. Take the DFT of all 4 GW observations, with a Tukey Window as window (this one has an extended flat period at the center)
3. Smoothen each noise spectrum using a gaussian kernel like we did in class
4. Take the average of all 4 noise spectra

I then repeat this for the Livingston detector

In [22]:

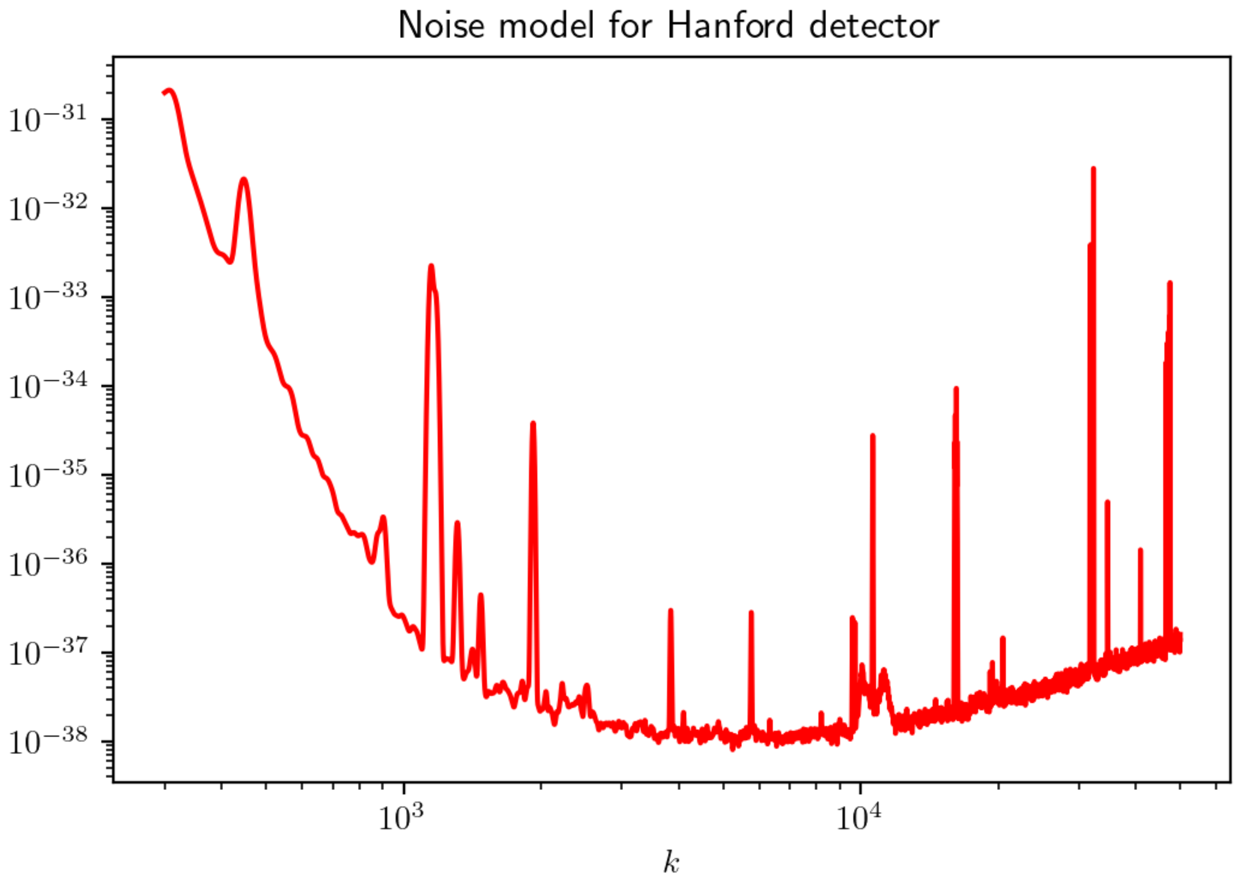
```
# This cell is code written in class
import numpy as np
from matplotlib import pyplot as plt
import h5py
import glob
plt.ion()

def read_template(filename):
    dataFile=h5py.File(filename,'r')
    template=dataFile['template']
    tp=template[0]
    tx=template[1]
    return tp,tx

def read_file(filename):
    dataFile=h5py.File(filename,'r')
    dqInfo = dataFile['quality']['simple']
    qmask=dqInfo['DQmask'][...]

    meta=dataFile['meta']
    #gpsStart=meta['GPSstart'].value
    gpsStart=meta['GPSstart'][()]
    #print meta.keys()
    #utc=meta['UTCstart'].value
    utc=meta['UTCstart'][()]
    #duration=meta['Duration'].value
    duration=meta['Duration'][()]
    #strain=dataFile['strain']['Strain'].value
    strain=dataFile['strain']['Strain'][()]
    dt=(1.0*duration)/len(strain)

    dataFile.close()
    return strain,dt,utc

def smooth_vector(vec,sig):
    n=len(vec)
    x=np.arange(n)
    x[n//2:]=x[n//2:]-n
    x = np.clip(x,-20000,20000)
    kernel=np.exp(-0.5*x**2/sig**2) #make a Gaussian kernel
    kernel=kernel/kernel.sum()
    vecft=np.fft.rfft(vec)
    kernelft=np.fft.rfft(kernel)
    vec_smooth=np.fft.irfft(vecft*kernelft) #convolve the data with the kernel
    return vec_smooth
```

In [23]:
```
# 1. Loading all 4 data files from Hanford detector
names=[]
names.append(dir_ligo+'/H-H1_LOSC_4_V1-1167559920-32.hdf5')
names.append(dir_ligo+'/H-H1_LOSC_4_V2-1126259446-32.hdf5')
names.append(dir_ligo+'/H-H1_LOSC_4_V2-1128678884-32.hdf5')
names.append(dir_ligo+'/H-H1_LOSC_4_V2-1135136334-32.hdf5')

strains=np.zeros((4,131072))
for index,fname in enumerate(names):
    print('reading data file ',fname)
    strains[index,:]=read_file(fname)[0]
```

```
reading data file  ligo_data/H-H1_LOSC_4_V1-1167559920-32.hdf5
reading data file  ligo_data/H-H1_LOSC_4_V2-1126259446-32.hdf5
reading data file  ligo_data/H-H1_LOSC_4_V2-1128678884-32.hdf5
reading data file  ligo_data/H-H1_LOSC_4_V2-1135136334-32.hdf5
```

In [24]:
```
# Creating the Tukey window
window = np.cos(np.linspace(-np.pi/2,np.pi/2,strains[0].shape[0]))

x=np.arange(strains[0].shape[0])
window= np.zeros((strains[0].shape[0]))
window[int(window.shape[0]/4):int(3*window.shape[0]/4)]=1
window[0:int(window.shape[0]/4)]=(1/2)*(1-np.cos(x*np.pi/(window.shape[0]/4)))[0:int(wi
window[int(3*window.shape[0]/4):]=(1/2)*(1+np.cos(x*np.pi/(window.shape[0]/4)))[0:int(w

# 2. and 3. Take the DFT of each spectrum and smoothen each one
noises_smooth=[0,0,0,0]
for i in range(4):
    ps = np.fft.fft(window*strains[i])
    noises_smooth[i] = smooth_vector(np.abs(ps)**2,10)
    noises_smooth[i] = noises_smooth[i][:len(ps)//2+1]
noises_smooth = np.array(noises_smooth)
```

In [25]:
```
# 4. Average the noise spectra
noise_smooth = np.average(noises_smooth,axis=0)
noise_smooth[0:300]=np.inf
noise_smooth[50000:]=np.inf
plt.plot(noise_smooth,color='red')
plt.xscale('log')
plt.yscale('log')
plt.title('Noise model for Hanford detector')
plt.xlabel('$k$')
noise_smooth_hanford = noise_smooth
```

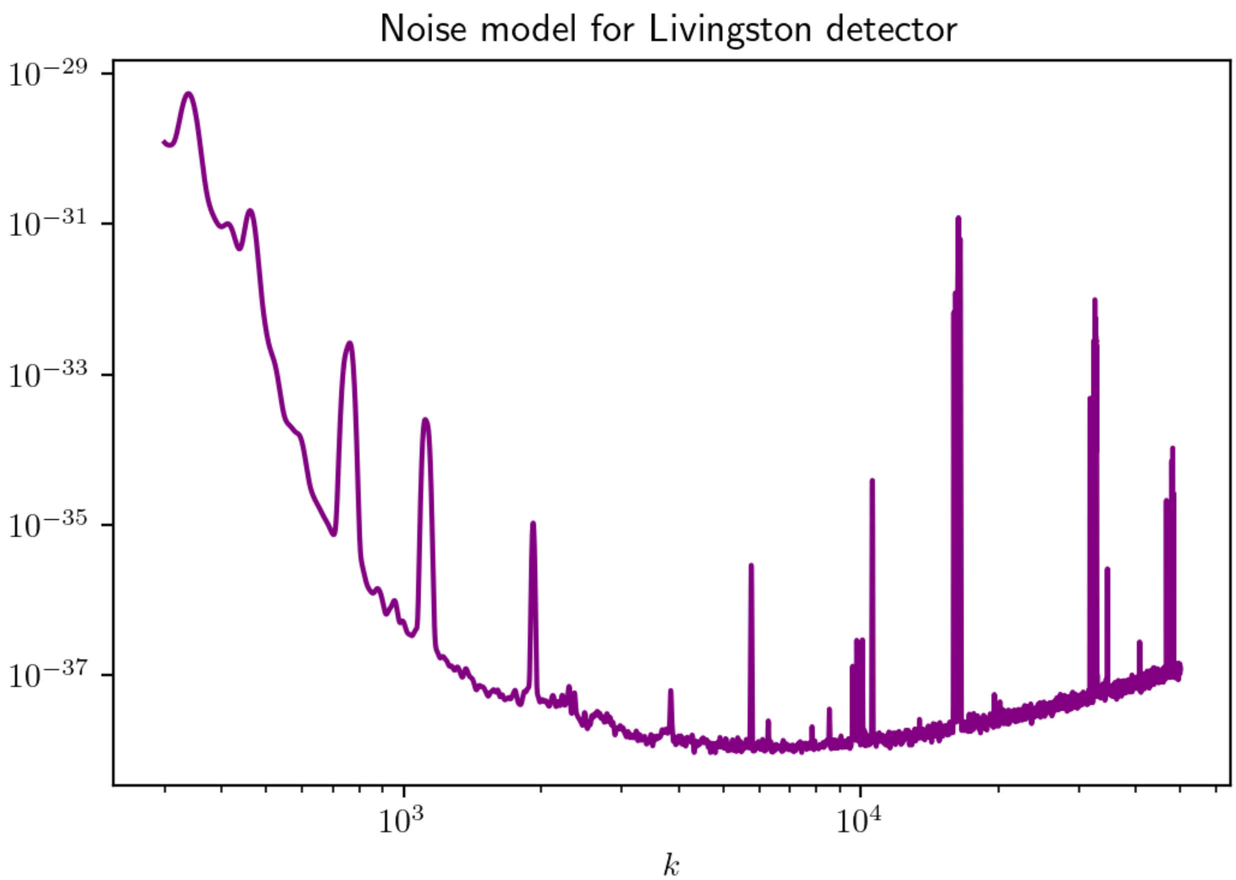


In [26]:

```
# Now I just repeat the 4 steps for the Livingston detector

# 1. Loading all 4 data files from Hanford detector
names=[]
names.append(dir_ligo+'/L-L1_LOSC_4_V1-1167559920-32.hdf5')
names.append(dir_ligo+'/L-L1_LOSC_4_V2-1126259446-32.hdf5')
names.append(dir_ligo+'/L-L1_LOSC_4_V2-1128678884-32.hdf5')
names.append(dir_ligo+'/L-L1_LOSC_4_V2-1135136334-32.hdf5')

strains=np.zeros((4,131072))
for index,fname in enumerate(names):
    print('reading data file ',fname)
    strains[index,:]=read_file(fname)[0]

# Creating the Tukey window
window = np.cos(np.linspace(-np.pi/2,np.pi/2,strains[0].shape[0]))

x=np.arange(strains[0].shape[0])
window= np.zeros((strains[0].shape[0]))
window[int(window.shape[0]/4):int(3*window.shape[0]/4)]=1
window[0:int(window.shape[0]/4)]=(1/2)*(1-np.cos(x*np.pi/(window.shape[0]/4)))[0:int(wi
window[int(3*window.shape[0]/4):]=(1/2)*(1+np.cos(x*np.pi/(window.shape[0]/4)))[0:int(w

# 2. and 3. Take the DFT of each spectrum and smoothen each one
noises_smooth=[0,0,0,0]
for i in range(4):
    ps = np.fft.fft(window*strains[i])
    noises_smooth[i] = smooth_vector(np.abs(ps)**2,10)
    noises_smooth[i] = noises_smooth[i][:len(ps)//2+1]
noises_smooth = np.array(noises_smooth)
```

```
# 4. Average the noise spectra
noise_smooth = np.average(noises_smooth,axis=0)
noise_smooth[0:300]=np.inf
noise_smooth[50000:]=np.inf
plt.plot(noise_smooth,color='purple')
plt.xscale('log')
plt.yscale('log')
plt.title('Noise model for Livingston detector')
plt.xlabel('$k$')
noise_smooth_livingston = noise_smooth
```

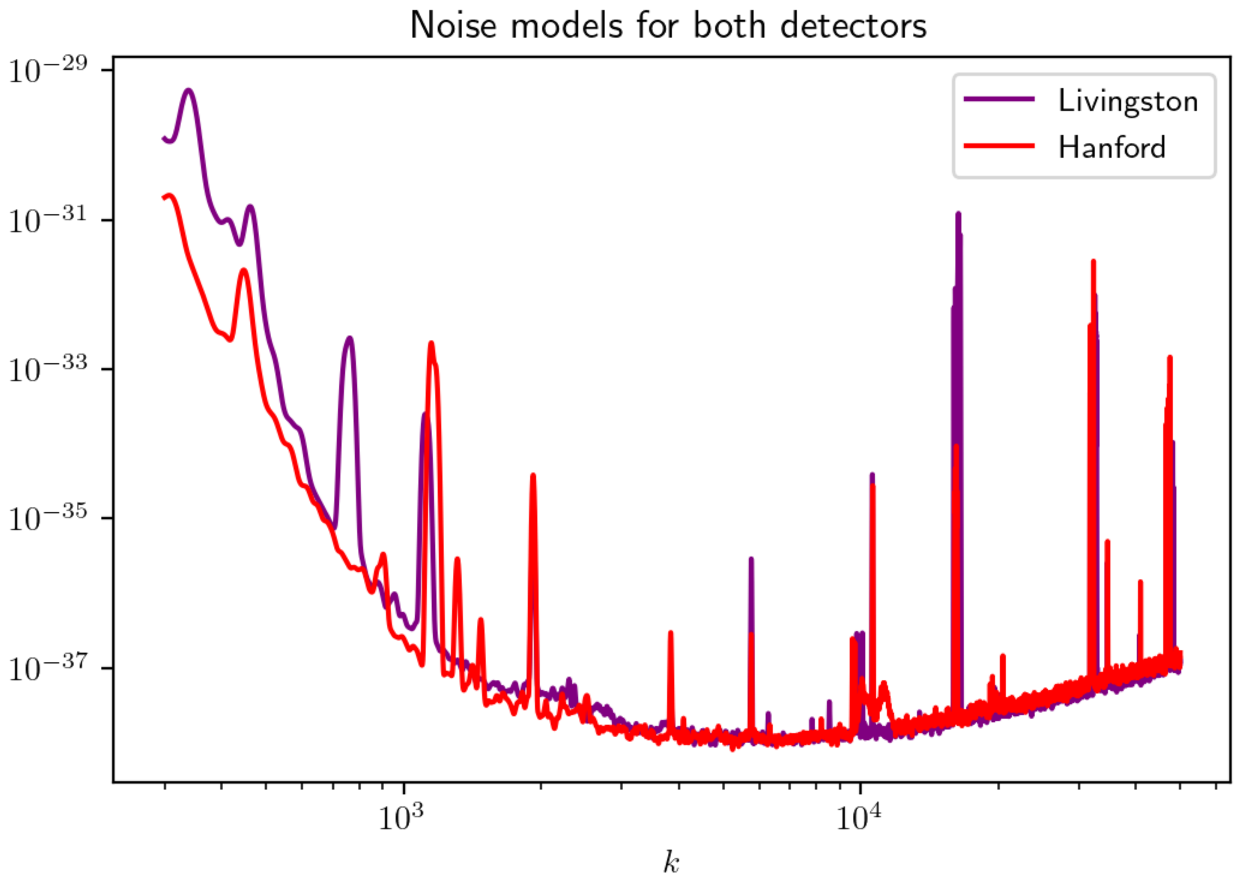
```
reading data file  ligo_data/L-L1_LOSC_4_V1-1167559920-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1126259446-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1128678884-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1135136334-32.hdf5
```



In [27]:

```
# Now I just plot the two noise models together
plt.plot(noise_smooth_livingston,label='Livingston',color='purple')
plt.plot(noise_smooth_hanford,label='Hanford',color='red')
plt.xscale('log')
plt.yscale('log')
plt.title('Noise models for both detectors')
plt.xlabel('$k$')
plt.legend(frameon=True)
```

Out[27]: <matplotlib.legend.Legend at 0x276f9b75fc8>



b)

Now we search for a signal in the data. I make a function that takes in a template, a datafile, a window and a noise model and returns the output of a match filter. This function is almost entirely code we wrote in class.

In [28]:

```
def match_filter(tp,strain,dt,window,noise_model):
    tobs=dt*len(strain)
    dnu=1/tobs
    nu=np.arange(len(noise_model))*dnu
    nu[0]=0.5*nu[1]

    Ninv=1/noise_model
    Ninv[nu>1500]=0
    Ninv[nu<20]=0

    template_ft=np.fft.rfft(tp*window)
    template_filt=template_ft*Ninv
    data_ft=np.fft.rfft(strain*window)
    rhs=np.fft.irfft(data_ft*np.conj(template_filt))
    return rhs
```

Using this function I search for a GW in all 4 data files, for both detectors

In [29]:

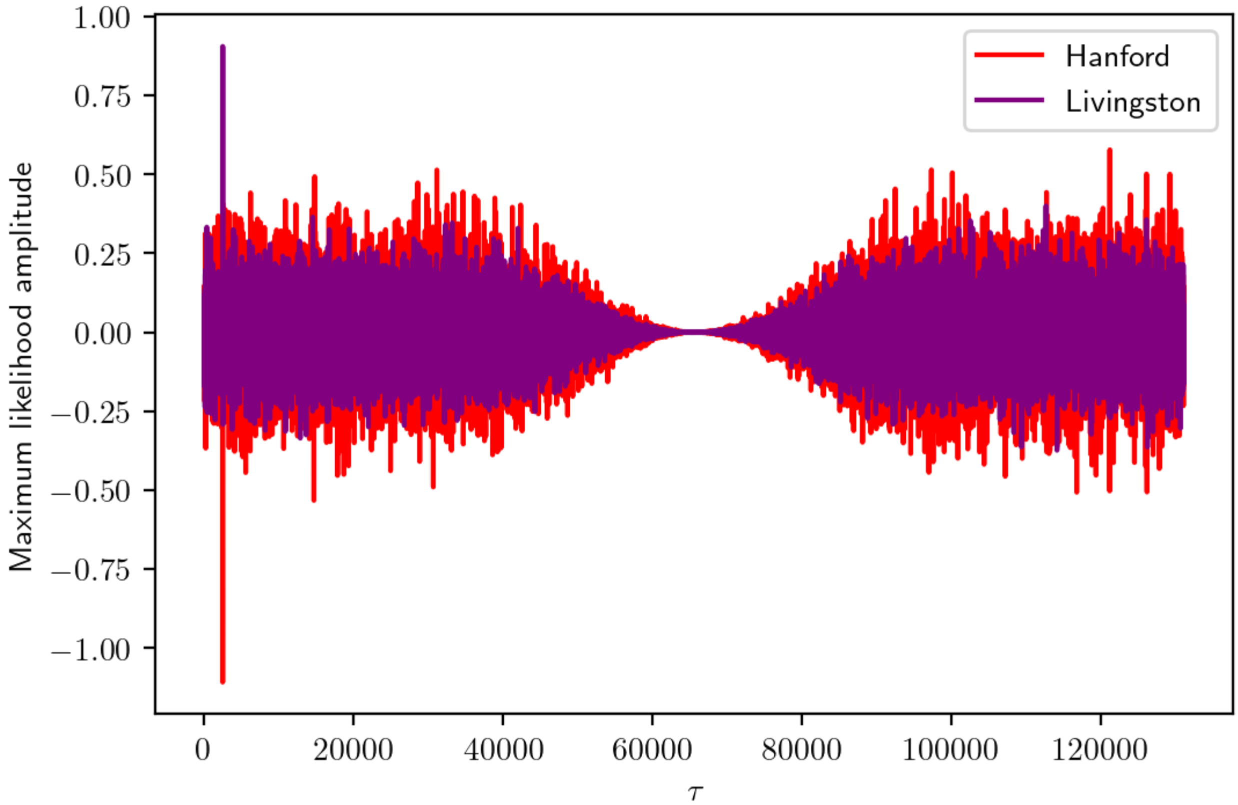
```
def search_GW(fname_hanford,fname_livinston,template_name):
    print('reading data file ',fname_hanford)
    strain_hanford,dt_hanford,utc_hanford=read_file(fname_hanford)
```

```
    print('reading data file ',fname_livingston)
    strain_livingston,dt_livingston,utc_livingston=read_file(fname_livingston)

    print('reading template file', template_name)
    tp,tx=read_template(template_name)

    match_filter_hanford = match_filter(tp,strain_hanford,dt_hanford,window,noise_smoot
    match_filter_livingston = match_filter(tp,strain_livingston,dt_livingston,window,no
    plt.plot(match_filter_hanford,color='red',label='Hanford')
    plt.plot(match_filter_livingston,color='purple',label='Livingston')
    plt.xlabel('$\\tau$')
    plt.ylabel('Maximum likelihood amplitude')
    plt.legend(frameon=True)
```

In [30]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V1-1167559920-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V1-1167559920-32.hdf5'
template_name=dir_ligo+'/GW170104_4_template.hdf5'
search_GW(fname_hanford,fname_livingston,template_name)
```

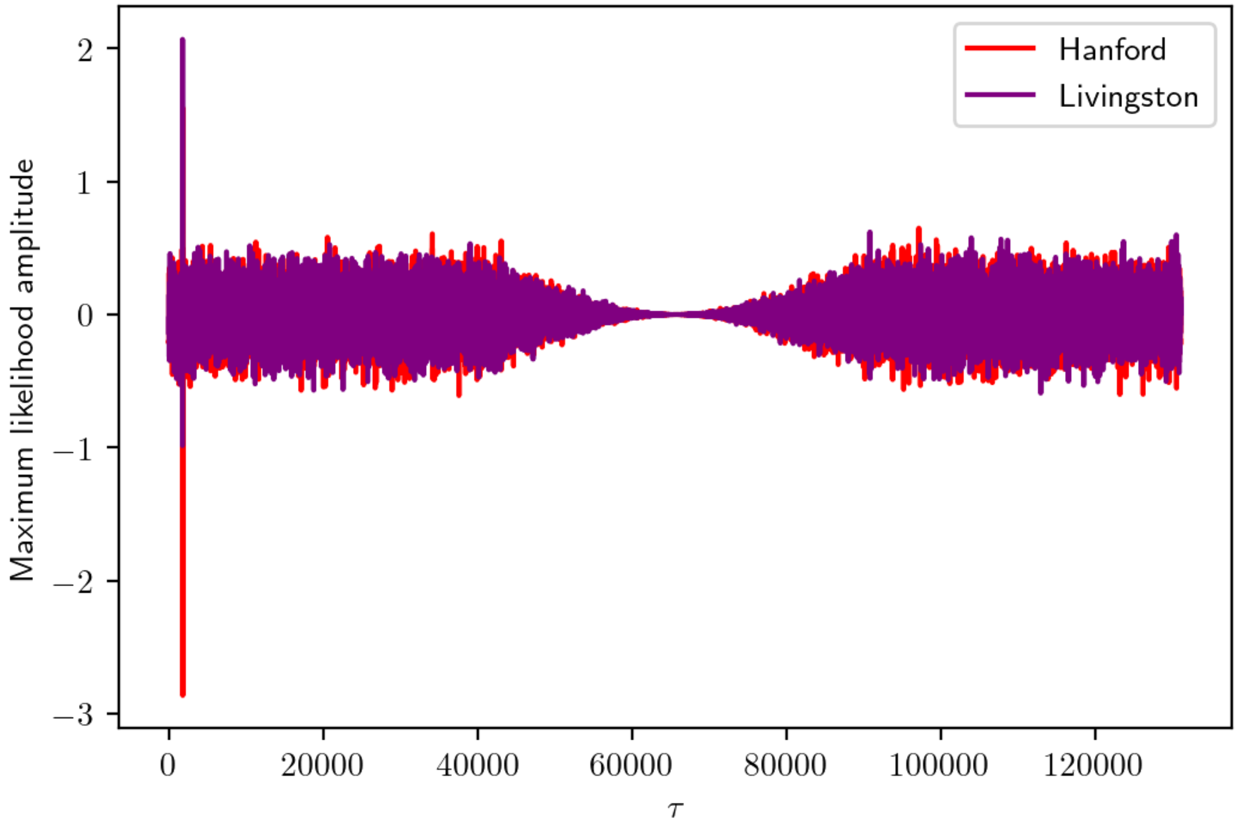
```
reading data file  ligo_data/H-H1_LOSC_4_V1-1167559920-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V1-1167559920-32.hdf5
reading template file ligo_data/GW170104_4_template.hdf5
```



In [31]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1126259446-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1126259446-32.hdf5'
template_name=dir_ligo+'/GW150914_4_template.hdf5'
search_GW(fname_hanford,fname_livingston,template_name)
```

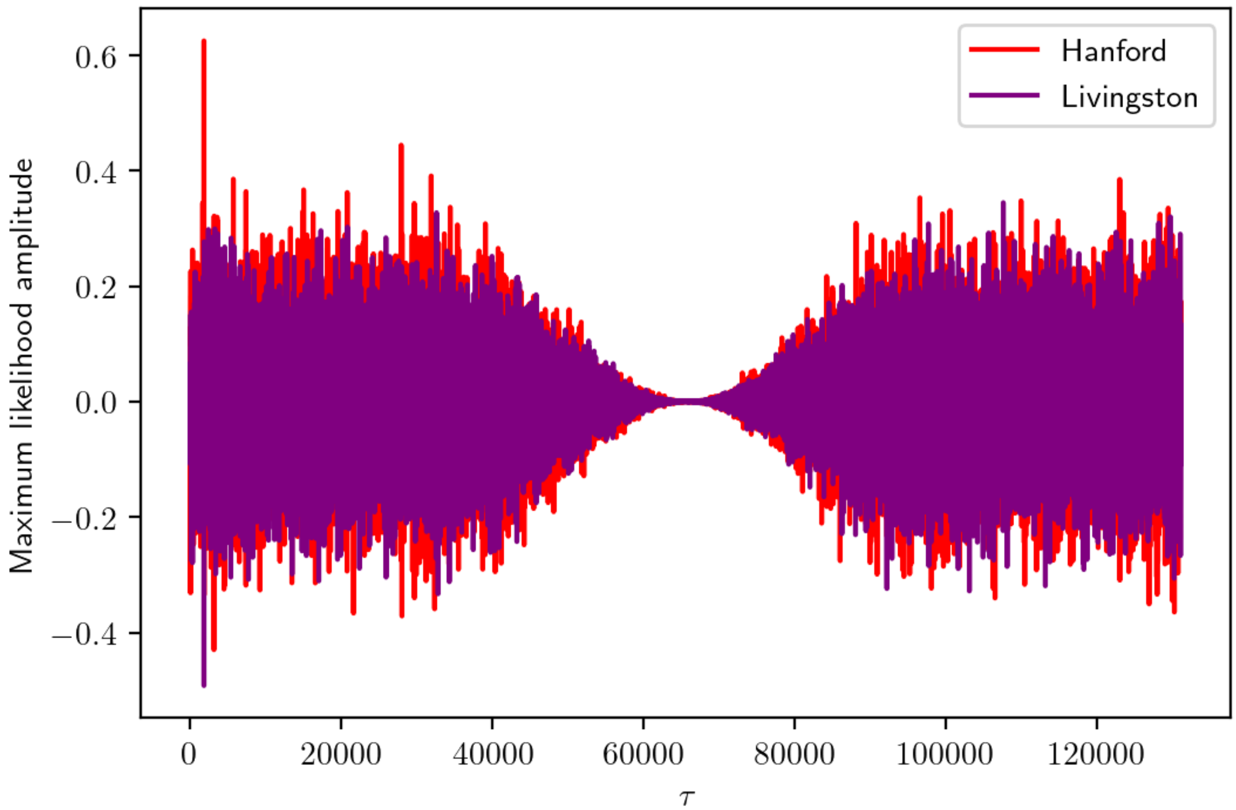
```
reading data file  ligo_data/H-H1_LOSC_4_V2-1126259446-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1126259446-32.hdf5
reading template file ligo_data/GW150914_4_template.hdf5
```



In [32]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1128678884-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1128678884-32.hdf5'
template_name=dir_ligo+'/LVT151012_4_template.hdf5'
search_GW(fname_hanford,fname_livingston,template_name)
```

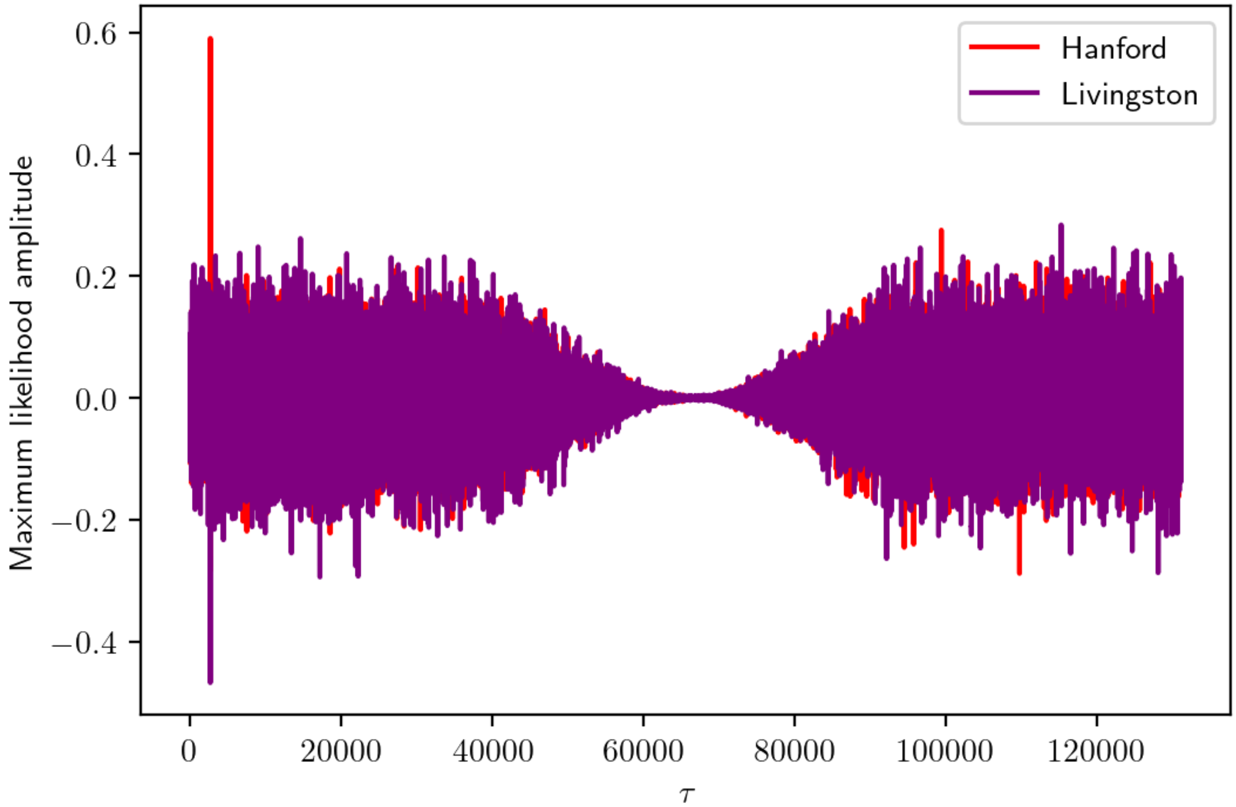
```
reading data file  ligo_data/H-H1_LOSC_4_V2-1128678884-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1128678884-32.hdf5
reading template file ligo_data/LVT151012_4_template.hdf5
```



In [33]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1135136334-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1135136334-32.hdf5'
template_name=dir_ligo+'/GW151226_4_template.hdf5'
search_GW(fname_hanford,fname_livingston,template_name)
```

```
reading data file  ligo_data/H-H1_LOSC_4_V2-1135136334-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1135136334-32.hdf5
reading template file ligo_data/GW151226_4_template.hdf5
```



c)

Now I remake the match_filter function so that it returns the maximum amplitude and the associated error. The error is calculated by taking the standard deviation of the match filter array (making sure to ignore values near the center since they are pushed towards 0 by the window function).

In [34]:

```
def search_GW_with_error(fname_hanford,fname_livinston,template_name):
    print('reading data file ',fname_hanford)
    strain_hanford,dt_hanford,utc_hanford=read_file(fname_hanford)
    print('reading data file ',fname_livingston)
    strain_livingston,dt_livingston,utc_livingston=read_file(fname_livingston)

    print('reading template file', template_name)
    tp,tx=read_template(template_name)

    amplitude_hanford,e_amplitude_hanford = match_filter_with_error(tp,strain_hanford,d
    amplitude_livingston,e_amplitude_livingston = match_filter_with_error(tp,strain_liv
    return amplitude_hanford,e_amplitude_hanford,amplitude_livingston,e_amplitude_livin

def match_filter_with_error(tp,strain,dt,window,noise_model):
    tobs=dt*len(strain)
    dnu=1/tobs
    nu=np.arange(len(noise_model))*dnu
    nu[0]=0.5*nu[1]

    Ninv=1/noise_model
    Ninv[nu>1500]=0
    Ninv[nu<20]=0
```

```
    template_ft=np.fft.rfft(tp*window)
    template_filt=template_ft*Ninv
    data_ft=np.fft.rfft(strain*window)
    rhs=np.fft.irfft(data_ft*np.conj(template_filt))
    amplitude = np.max(np.abs(rhs))

    e_amplitude = np.std(rhs[0:40000])
    return(amplitude,e_amplitude)
```

In [35]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V1-1167559920-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V1-1167559920-32.hdf5'
template_name=dir_ligo+'/GW170104_4_template.hdf5'
amplitude_hanford,e_amplitude_hanford,amplitude_livingston,e_amplitude_livingston = sea
print(amplitude_hanford,e_amplitude_hanford)
print(amplitude_livingston,e_amplitude_livingston)
print('SNR_hanford: ',amplitude_hanford/e_amplitude_hanford)
print('SNR_livingston: ',amplitude_livingston/e_amplitude_livingston)
```

```
reading data file  ligo_data/H-H1_LOSC_4_V1-1167559920-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V1-1167559920-32.hdf5
reading template file ligo_data/GW170104_4_template.hdf5
1.1082746893740416 0.1324326837567538
0.9050732240458457 0.09335441972259138
SNR_hanford:  8.368588915782068
SNR_livingston:  9.695022760950456
```

In [36]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1126259446-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1126259446-32.hdf5'
template_name=dir_ligo+'/GW150914_4_template.hdf5'
amplitude_hanford,e_amplitude_hanford,amplitude_livingston,e_amplitude_livingston = sea
print(amplitude_hanford,e_amplitude_hanford)
print(amplitude_livingston,e_amplitude_livingston)
print('SNR_hanford: ',amplitude_hanford/e_amplitude_hanford)
print('SNR_livingston: ',amplitude_livingston/e_amplitude_livingston)
```

```
reading data file  ligo_data/H-H1_LOSC_4_V2-1126259446-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1126259446-32.hdf5
reading template file ligo_data/GW150914_4_template.hdf5
2.8642046702379726 0.1671464095616782
2.0694626253018327 0.15784837836906457
SNR_hanford:  17.135903054986418
SNR_livingston:  13.11044590184659
```

In [37]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1128678884-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1128678884-32.hdf5'
template_name=dir_ligo+'/LVT151012_4_template.hdf5'
amplitude_hanford,e_amplitude_hanford,amplitude_livingston,e_amplitude_livingston = sea
print(amplitude_hanford,e_amplitude_hanford)
print(amplitude_livingston,e_amplitude_livingston)
print('SNR_hanford: ',amplitude_hanford/e_amplitude_hanford)
print('SNR_livingston: ',amplitude_livingston/e_amplitude_livingston)
```

```
reading data file  ligo_data/H-H1_LOSC_4_V2-1128678884-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1128678884-32.hdf5
reading template file ligo_data/LVT151012_4_template.hdf5
0.6253187060154224 0.10119188095263694
0.4921720132850558 0.0891462089592485
```

```
SNR_hanford:  6.1795343670714455
SNR_livingston:  5.520952814830778
```

In [38]:

```
fname_hanford=dir_ligo+'/H-H1_LOSC_4_V2-1135136334-32.hdf5'
fname_livingston=dir_ligo+'/L-L1_LOSC_4_V2-1135136334-32.hdf5'
template_name=dir_ligo+'/GW151226_4_template.hdf5'
amplitude_hanford,e_amplitude_hanford,amplitude_livingston,e_amplitude_livingston = sea
print(amplitude_hanford,e_amplitude_hanford)
print(amplitude_livingston,e_amplitude_livingston)
print('SNR_hanford: ',amplitude_hanford/e_amplitude_hanford)
print('SNR_livingston: ',amplitude_livingston/e_amplitude_livingston)
```

```
reading data file  ligo_data/H-H1_LOSC_4_V2-1135136334-32.hdf5
reading data file  ligo_data/L-L1_LOSC_4_V2-1135136334-32.hdf5
reading template file ligo_data/GW151226_4_template.hdf5
0.5902131634914305 0.061201411393756656
0.46731724267931474 0.07075374566733185
SNR_hanford:  9.643783534567962
SNR_livingston:  6.604841033809503
```

So the SNR we got for all events are

GW170104: 8.36 (Hanford) 9.69 (Livingston)

GW150914: 17.13 (Hanford) 13.11 (Livingston)

LVT151012: 6.17 (Hanford) 5.52 (Livingston)

GW151226: 9.64 (Hanford) 6.60 (Livingston)

I ran out of time to do the rest