Norwegian University of Science and Technology
Department of Electronics and Telecommunications

TTT4120 Digital Signal Processing - Suggested solution for problem set 7 (Python)

In [1]:
```
import numpy as np
import random
from matplotlib import pyplot as plt
from scipy import signal
```

Problem 1

(a).

The samples from the white noises are uncorrelated and drawn from the corresponding distribution. Since a white noise has zero mean and we are told that these noises have unit variance, we have all the parameters for their probability distributions. For the uniform distribution we have to find the limits (which will be symmetric because of the zero mean property):

$$\frac{1}{2a}\int_{-a}^{a} x^2 dx = 1 \implies a = \sqrt{3}$$

Python code:

In [2]:

```python
N = 100

# plot
plt.rcParams['figure.figsize'] = [12, 9]
fig, axs = plt.subplots(3)
fig.suptitle('Figure 1: Realizations of different types of white noises', 

#white binary noise
bin_vec = [-1, 1]
xn_bin = []
for i in range(N):
    xn_bin.append(bin_vec[random.randint(0,1)])
axs[0].stem(xn_bin)
axs[0].set_title('White binary noise')
axs[0].label_outer()

#white gaussian noise
xn_gauss = []
for i in range(N):
    xn_gauss.append(random.gauss(0, 1))
axs[1].stem(xn_gauss)
axs[1].set_title('White gaussian noise')
axs[1].label_outer()

#white uniform noise
xn_uni = []
for i in range(N):
    xn_uni.append(random.uniform(-np.sqrt(3),np.sqrt(3)))
axs[2].stem(xn_uni)
axs[2].set_title('White uniform noise')
plt.show()
```

Figure 1: Realizations of different types of white noises

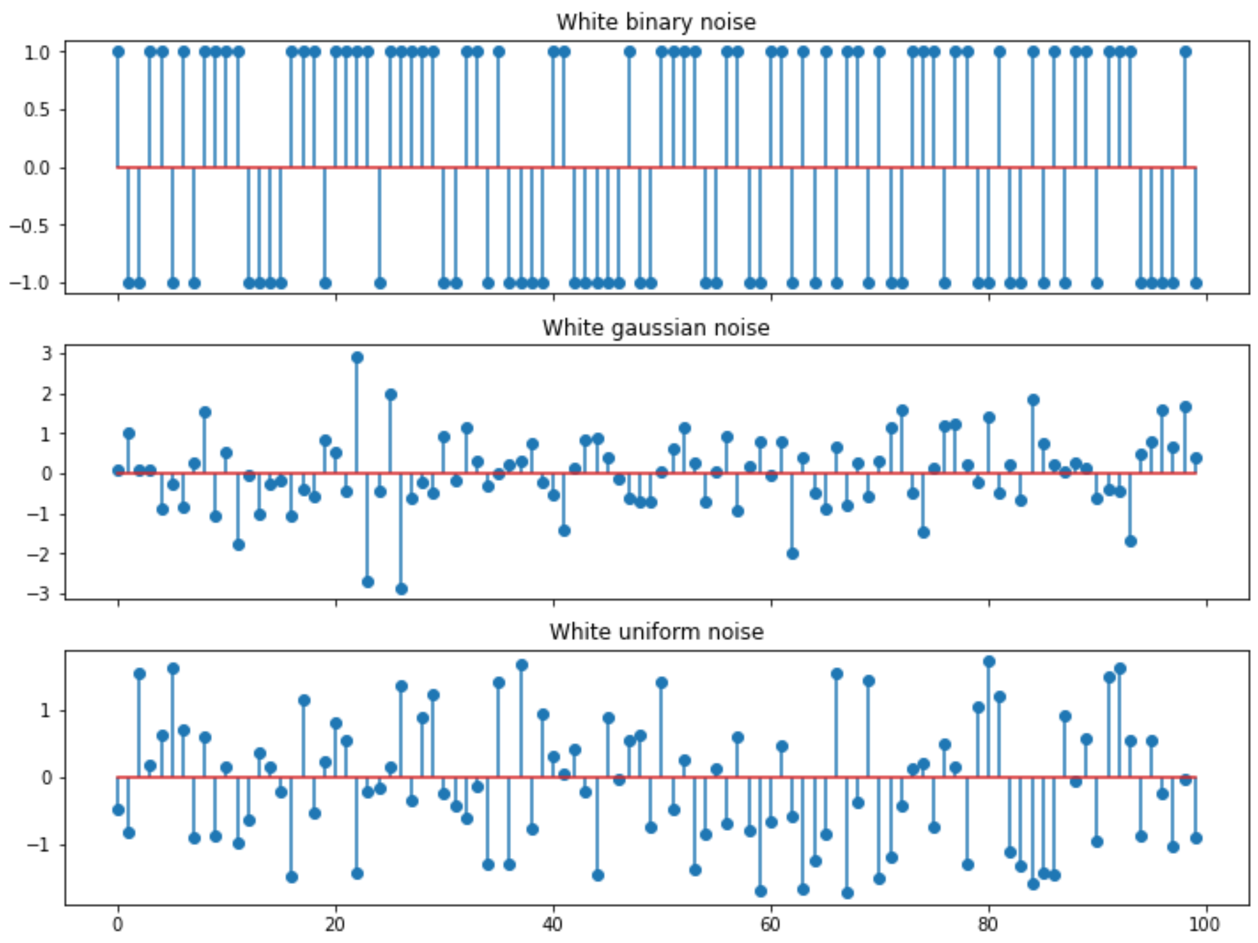


Figure 1 shows the realization of the different white noises. It is common for all the noises that the samples are uncorrelated for each of them. This means that given one sample there is no information about the value of the next one. The binary noise is limited to two different values, whereas the uniform and the gaussian noise takes values in a given interval. At first sight it is difficult to make difference between the gaussian noise and the uniform one. However it can be seen that the gaussian noise takes some values out of $(-\sqrt{3}, \sqrt{3})$. In addition, the value of the samples from the gaussian noise are concentrated in the center, where as the value of the samples of the uniform noise are equally spaced.

(b)

White binary noise:

$$P_X(x) = \begin{cases} 0.5 & \text{if } x = \pm 1 \\ 0 & \text{otherwise} \end{cases}$$

White uniform noise:

$$P_X(x) = \begin{cases} \frac{1}{2\sqrt{3}} & \text{if } x \in [-\sqrt{3}, \sqrt{3}] \\ 0 & \text{otherwise} \end{cases}$$

White gaussian noise:

$$P_X(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

To compute the mean of $X(n)$ we use the symmetry properties of $P_X(x)$:

$$\int x P_X(x) dx = 0$$

To compute the autocorrelation function we have to use the fact that the sample are uncorrelated (because they are white signals):

$$R_X[k] = E\{X[n]X[n-k]\} = \sigma^2 \delta(k)$$

The power density spectrum in the DTFT of the autocorrelation function:

$$S_X(\omega) = \text{DTFT}\{R_X[k]\} = \sigma_X^2$$

(c).

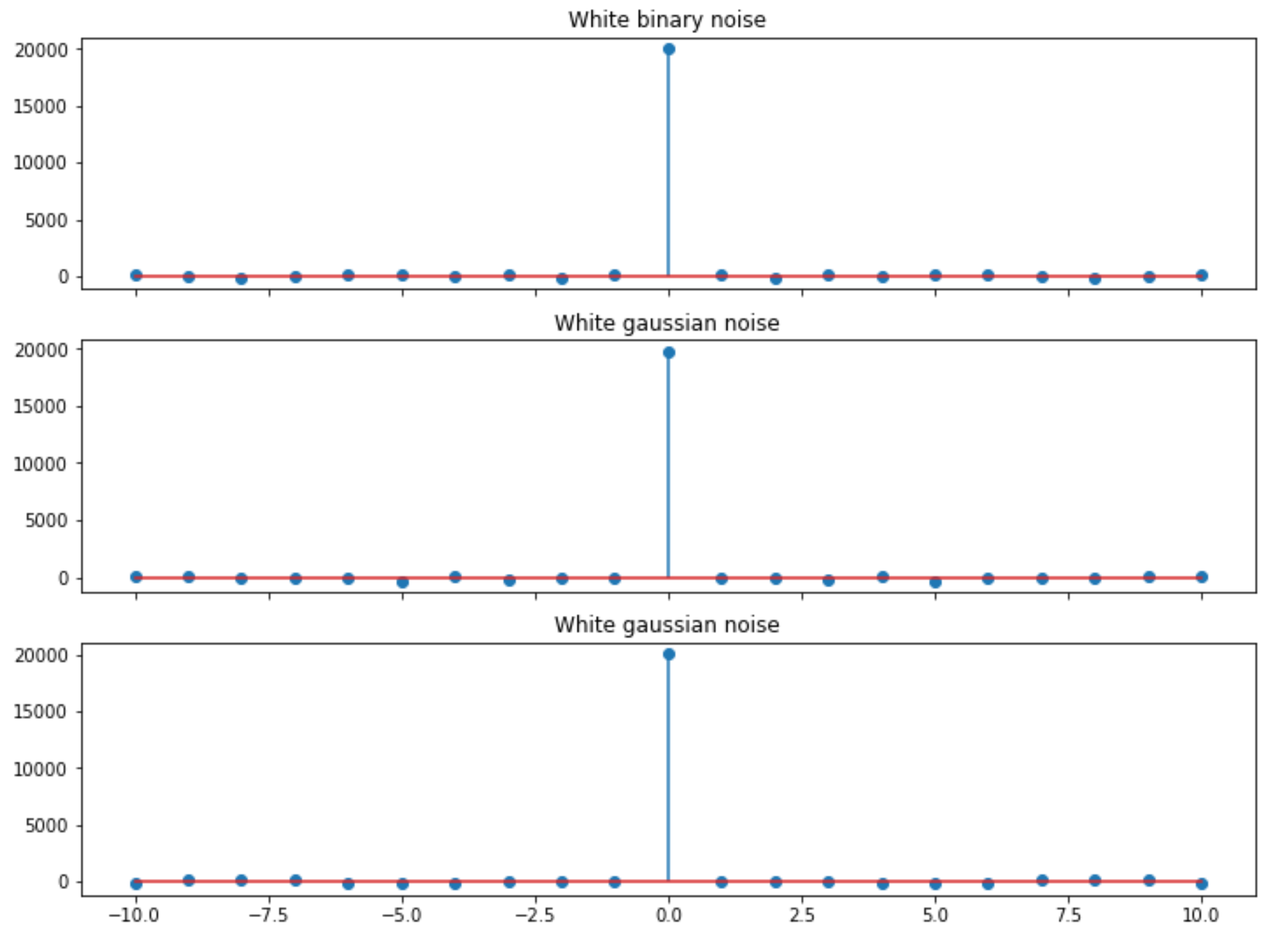
The mean estimates should be quite accurate (less than 0.001). Figure 2 shows that the estimated correlations are very accurate. They were found using the function np.convolve.

In [3]:

```
N = 20000

fig, axs = plt.subplots(3)
fig.suptitle('Figure 2: Estimates of the autocorrelation function', fontsi

#white binary noise
bin_vec = [-1, 1]
xn_bin = []

for i in range(N):
    xn_bin.append(bin_vec[random.randint(0,1)])
z = np.sum(xn_bin)/len(xn_bin)
print('mean (bin): ' +str(z))
corr = np.correlate(xn_bin, xn_bin, 'full')
axs[0].stem(np.linspace(-10,10,21), corr[19989:20010])
axs[0].set_title('White binary noise')
axs[0].label_outer()


#white gaussian noise
xn_gauss = []
for i in range(N):
    xn_gauss.append(random.gauss(0, 1))
corr = np.correlate(xn_gauss, xn_gauss, 'full')
z = np.sum(xn_gauss)/len(xn_gauss)
corr = np.correlate(xn_gauss, xn_gauss, 'full')
axs[1].stem(np.linspace(-10,10,21), corr[19989:20010])
axs[1].set_title('White gaussian noise')
axs[1].label_outer()
print('mean (gauss): ' +str(z))

#white uniform noise
xn_uni = []
for i in range(N):
    xn_uni.append(random.uniform(-np.sqrt(3),np.sqrt(3)))
corr = np.correlate(xn_uni, xn_uni, 'full')
z = np.sum(xn_uni)/len(xn_uni)
print('mean (uni): ' +str(z))
corr = np.correlate(xn_uni, xn_uni, 'full')
axs[2].stem(np.linspace(-10,10,21), corr[19989:20010])
axs[2].set_title('White gaussian noise')
plt.show()
```

```
mean (bin): -0.0122
mean (gauss): 0.009561848349589442
mean (uni): 0.006054718644892048
```

Figure 2: Estimates of the autocorrelation function



Problem 2

The mean value of the input $w(n)$ is zero, because it is a white noise process.

The autocorrelation function of the input, w(n), is given by

$$\gamma_{ww}(m) = E[w(n)w(n+m)] = \sigma_w^2\delta(m) = \frac{3}{4}\delta(m)$$

since the samples of white noise are uncorrelated. The power density spectrum of the input is given by

$$\Gamma_{ww}(f) = \text{DTFT}\{\gamma_{ww}(m)\} = \sum_{m=-\infty}^{\infty} \gamma_{ww}(m)e^{-j2\pi fm}$$
$$= \sum_{m=-\infty}^{\infty} \sigma_w^2\delta(m)e^{-j2\pi fm} = \sigma_w^2 = \frac{3}{4}$$

(a).

The mean of the signal $x(n)$ is given by

$$m_x = E[x(n)] = m_w \sum_{k=-\infty}^{\infty} h(k) = 0, \text{ since } m_w = 0.$$

The autocorrelation function of the signal x(n) is given by

$$\gamma_{xx}(m) = \gamma_{ww}(m) * r_{hh}(m)$$

where $r_{hh}(m)$ is the autocorrelation function of the unit sample response $h(n)$ of the filter (which is a deterministic signal), i.e.

$$r_{hh}(m) = \sum_{n=-\infty}^{\infty} h(n)h(n+m)$$

The unit sample response for the filter is given by

$$h(n) = Z^{-1}\{H(z)\} = \left(-\frac{1}{2}\right)^n u(n).$$

We first find the autocorrelation function $r_{hh}(m)$ for $m \geq 0$,

$$\begin{aligned} r_{hh}(m) &= \sum_{n=-\infty}^{\infty} \left(-\frac{1}{2}\right)^n u(n) \left(-\frac{1}{2}\right)^{n+m} u(n+m) \\ &= \sum_{n=-0}^{\infty} \left(-\frac{1}{2}\right)^{2n+m} = \left(-\frac{1}{2}\right)^m \sum_{n=-0}^{\infty} \left(-\frac{1}{4}\right)^n \\ &= \left(-\frac{1}{2}\right)^m \frac{1}{1-\frac{1}{4}} = \frac{4}{3}\left(-\frac{1}{2}\right)^m \end{aligned}$$

For $m < 0$, we have that $r{hh}(m) = r{hh}(–m) = \frac{4}{3} \left(-\frac{1}{2}\right)^{-m}. It follows that

$$r_{hh}(m) = \frac{4}{3}\left(-\frac{1}{2}\right)^{|m|}, \text{ for all } m.$$

The autocorrelation function of the signal $x(n)$ is now given by

$$\gamma_{xx}(m) = \gamma_{ww}(m) * r_{hh}(m) = \frac{3}{4}\delta(m) * \frac{4}{3}\left(-\frac{1}{2}\right)^{|m|} = \left(-\frac{1}{2}\right)^{|m|}$$

The power density spectrum of the signal $x(n)$ is given by

$$\Gamma_{xx}(f) = \Gamma_{ww}(f)|H(\omega)|^2$$

We can compute $|H(\omega)|^2$ from the transfer function $H(z)$ by remembering that $H(\omega) = H(z)|_{z=e^{j\omega}}$.

$$|H(\omega)|^2 = H(\omega)H^*(\omega) = \frac{1}{1+\frac{1}{2}e^{-j\omega}} \frac{1}{1+\frac{1}{2}e^{j\omega}} = \frac{4}{5+4\cos\omega}$$

It follows that

$$\Gamma_{xx}(f) = \frac{3}{4}\frac{4}{5+4\cos\omega} = \frac{3}{5+4\cos\omega}$$

The power of the signal $x(n)$ is given by

$$P_x = E[x^2(n)] = \gamma_{xx}(0) = 1,$$

and vaiance

$$\sigma_x^2 = E[(x(n)-m_x)^2] = E[x(n)^2] = P_x = 1, \text{since } m_x = 0$$

(b).

We have the following expressions for the estimates:

Mean:

$$\hat{m}_x = \frac{1}{N}\sum_{n=0}^{N-1} x(n)$$

Power:

$$\hat{P}_x = \frac{1}{N}\sum_{n=0}^{N-1} x^2(n)$$

Autocorrelation function:

$$\hat{\gamma}_{xx}(l) = \left\{\begin{array}{ll} \frac{1}{N}\sum\limits_{n=0}^{N-1} x(n)x(n+l) & \text{for } 0 \le |l| \le N-1 \\ 0 & \text{for } N \le |l| \end{array}\right\}.$$

Power spectral density:

$$\hat{\Gamma}_{xx}(l) = \text{DFT}(\hat{\gamma}_{xx}(l))$$

(c).

The signal $x(n)$ can be estimated by first generating a segment of white Gaussian noise, and then filtering it trough the filter $H(z)$.

The plots of $\gamma_{xx}(m)$ and $\hat{\gamma}_{xx}(m)$ are shown in figure 3.

In [6]:

```
N = 20000
N_hn = 100
sigma = np.sqrt(3)/2


hn =(-1/2)**(np.linspace(0, N_hn - 1, N_hn))
wn = np.random.normal(0, sigma, N - N_hn + 1)

a = [1, 1.5]
b = [1]

#xn = signal.lfilter(b, a, wn)
xn = np.convolve(wn, hn)
print('N: ', len(xn))

# find the mean & power
m_hat = np.sum(xn)/N
power = np.sum(np.square(xn))/N
print('Estimated mean: ' + str(m_hat))
print('Estimated power: ' + str(power))

# function for finding autocorrelation using the estimator given in (b).
def gamma_hat(l):
    s = 0
    for i in range(N):
        if abs(l)>N:
            continue
        else:
            if (i+l)<N:
                s += xn[i]*xn[i+l]
    return s/N

# find the estimated autocorrelation for l = [-10, 10]
autocorr_est = []
for i in range(-10,11):
    autocorr_est.append(gamma_hat(i))

# find the calculated autocorrelation function
autocorr_cal = np.correlate(xn, xn, mode = 'full')

# plot the autocorrelations
fig, axs = plt.subplots(2)
fig.suptitle('Figure 3: Autocorrelation function and its unbiased estimate
axs[0].stem(np.linspace(-10,10,21), autocorr_est)
axs[0].set(xlabel = 'm')
axs[0].set_title('Estimated autocorrelation function')
axs[1].stem(np.linspace(-10,10,21), autocorr_cal[19989:20010]/N)
axs[1].set(xlabel = 'm')
axs[1].set_title('Calculated autocorrelation function')
plt.show()

# find the power spectrum
ng = 2**6 # len of the fft
Gamma_hat = np.fft.fft(autocorr_est, ng)
plt.plot(np.linspace(0,1, ng), np.absolute(Gamma_hat))
plt.title('Figure 4: Estimate of the power spectral density.', fontsize =
plt.xlabel('Normalized frequency $f$')
plt.show()
```

```
N:  20000
```

```
N: 10000
Estimated mean: 0.0012242935052066593
Estimated power: 1.00678999045417
```

Figure 3: Autocorrelation function and its unbiased estimate.

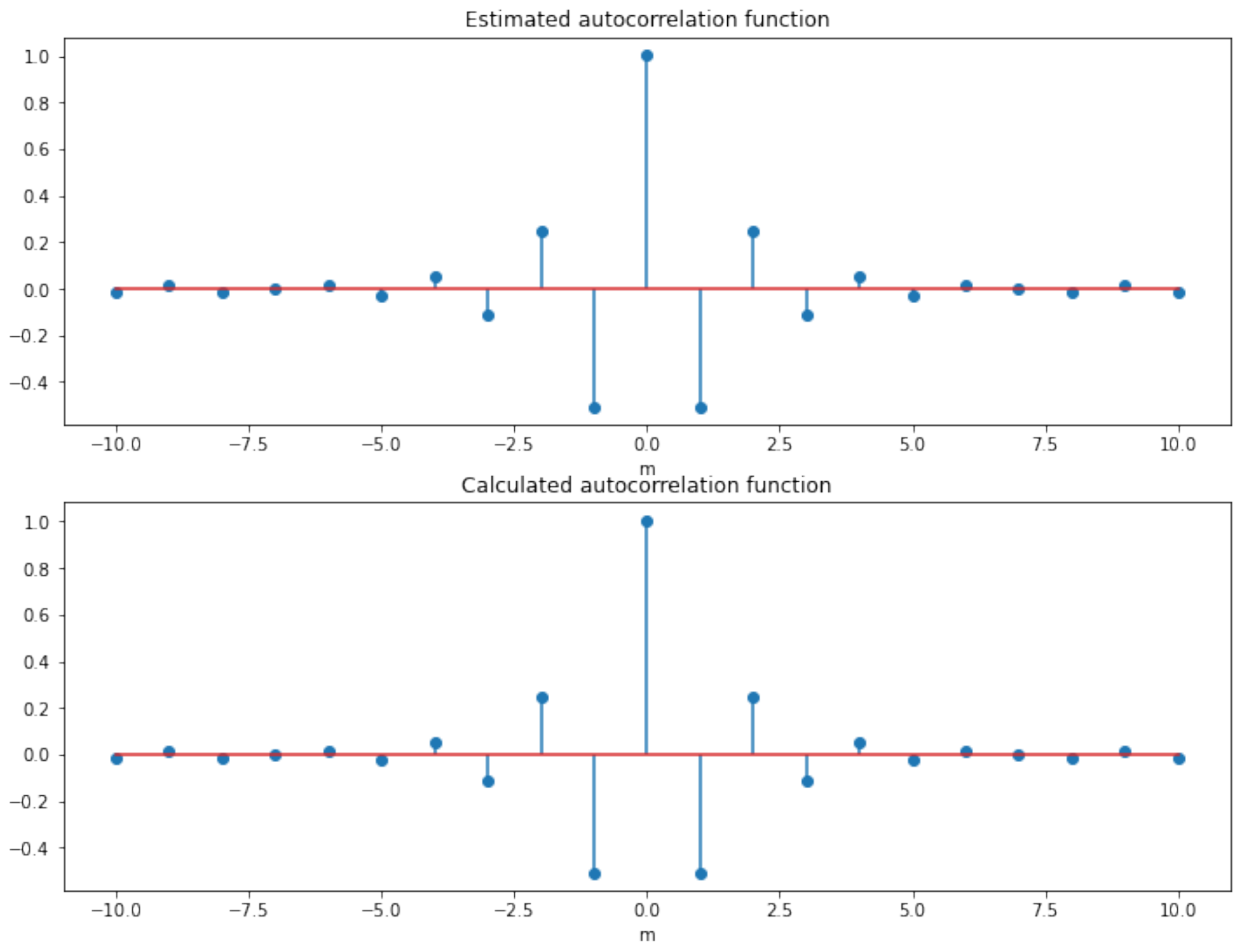
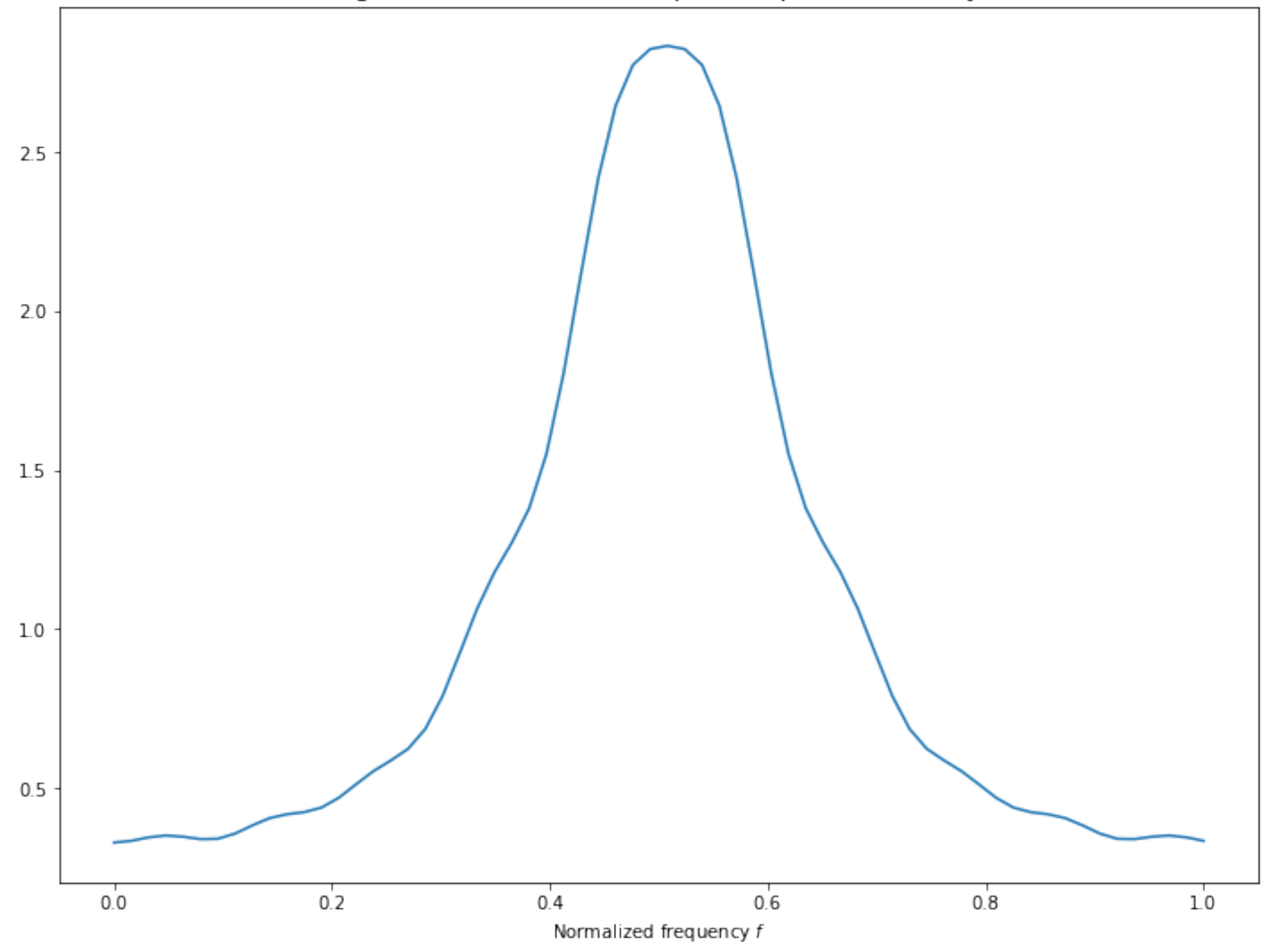


Figure 4: Estimate of the power spectral density.



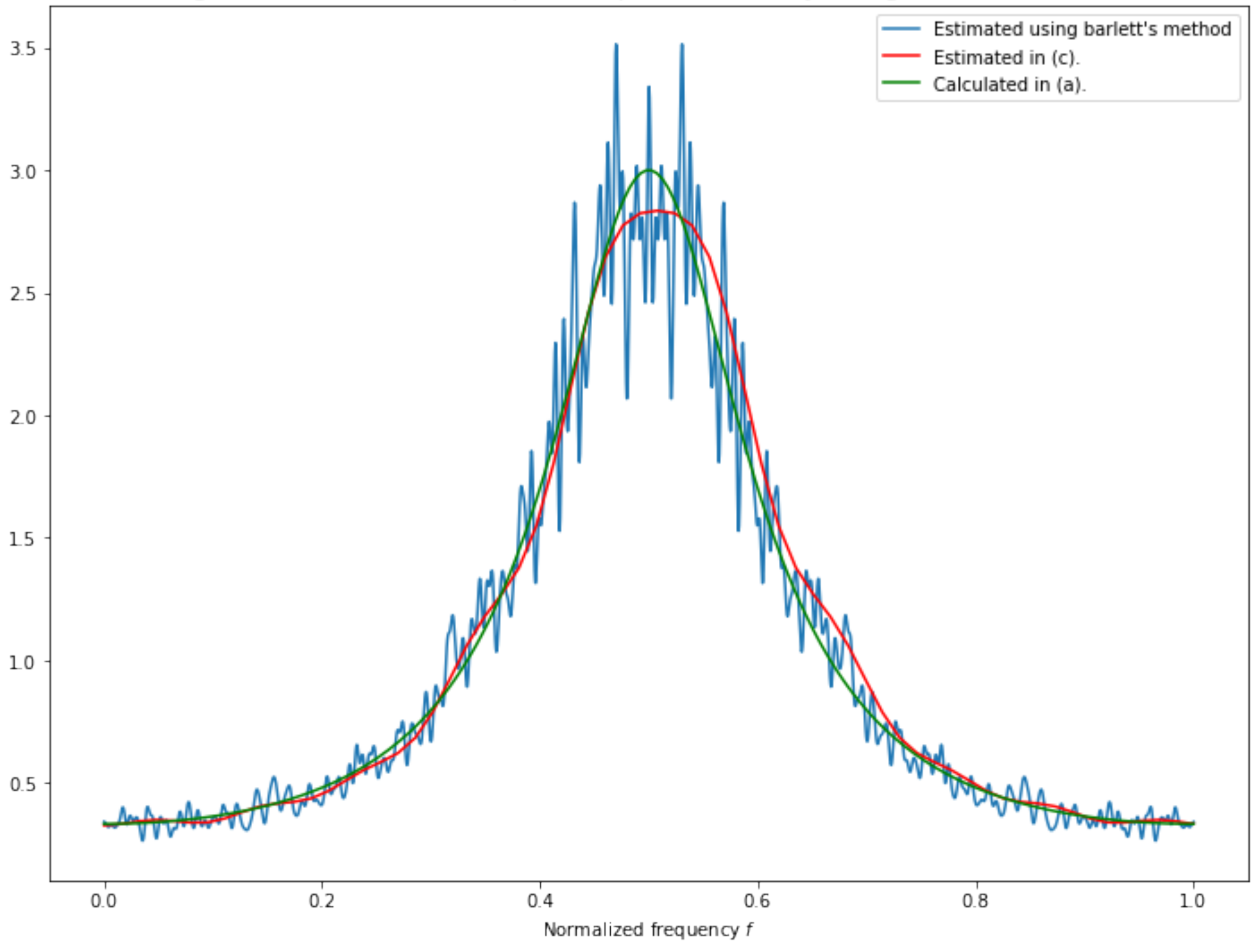
(d).

A estimate of the power density spectrum using Bartlett method are showed below.

In [7]:

```
k = 100 # number of (non-overlapping) data segments
N = 20000 # len of x[n]
l = int(N/(2*k)) # half the length of one segment

N_vec = []
# split up in k equal parts
for i in range(k):
    N_vec.append(xn[(i+1)*2*l-l:(i+1)*2*l+l])

# find the fft of each part, square and divide on the length of the data se
fft_vec = []
for i in range(len(N_vec)):
    fft = np.fft.fft(N_vec[i], 2048)
    fft_vec.append(np.square(np.abs(fft))/(2*l))

# averige the powerspectrum above for the k data segments and plot
averige = []
transposed_fft_vec = np.array(fft_vec).T
for i in range(len(transposed_fft_vec)):
    averige.append(np.sum(transposed_fft_vec[i])/len(transposed_fft_vec[0]
plt.plot(np.linspace(0,1, len(averige)), np.real(averige), label = "Estimat
plt.title("Figure 5: Estimate of the power spectral density using barlett's
plt.plot(np.linspace(0,1, ng), np.absolute(Gamma_hat), 'r', label = 'Estima
calculated_power = 3/(5+4*np.cos(2*np.pi*np.linspace(0, 1, 200)))
plt.plot(np.linspace(0, 1, 200), calculated_power, 'g', label = 'Calculated
plt.legend()
plt.xlabel('Normalized frequency $f$')
plt.show()
```

Figure 5: Estimate of the power spectral density using barlett's metod.



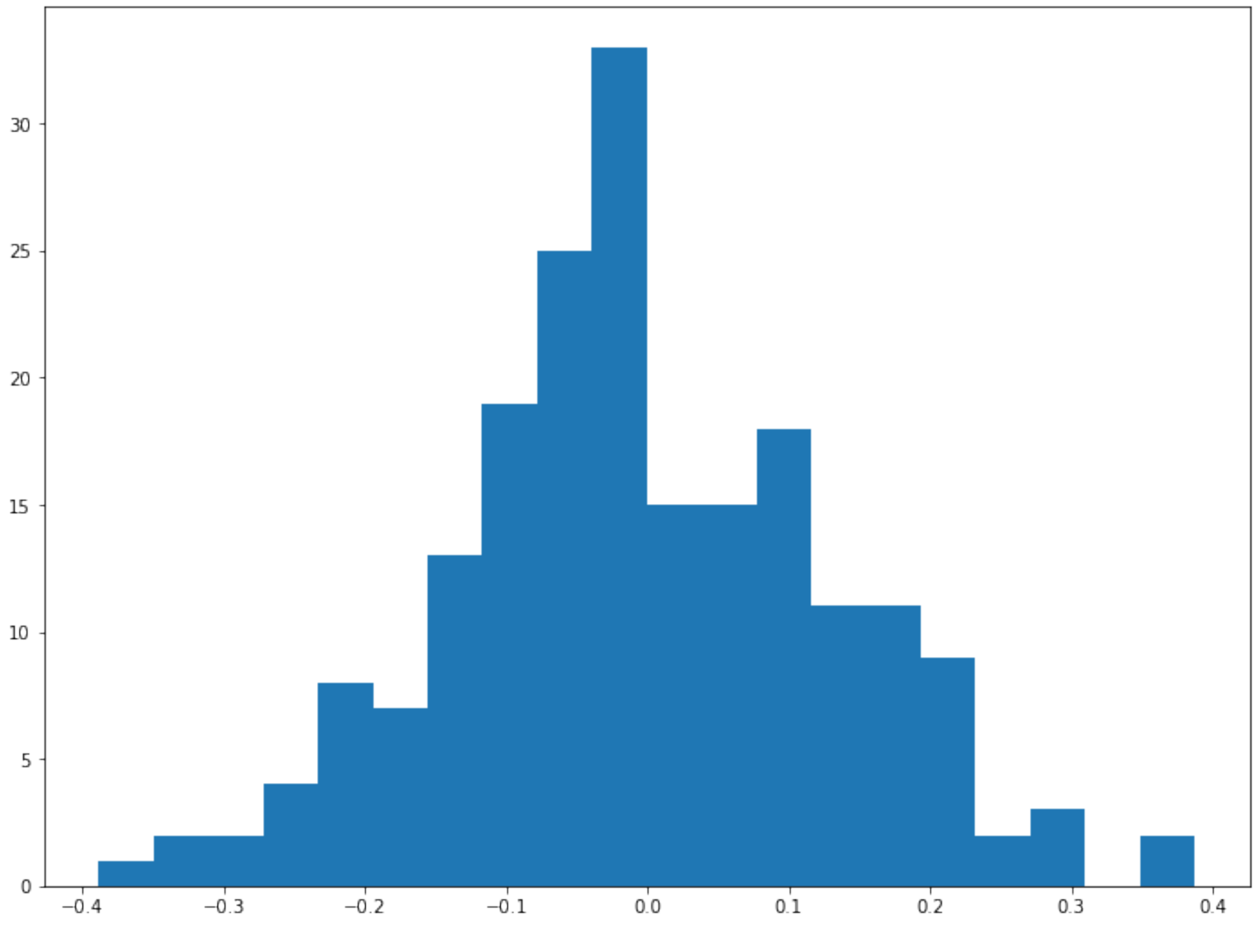
(e).

Change the parameter k in (d) and run the cell to see how the outcome changes.

Problem 3

In [8]:
```
num_of_segments = 200 # number of (non-overlapping) data segments
l = 10 # half the length of one segment (i.e. K/2)

N_vec = []

# use x[n] to generate num_of_segments segments of length K (or 2*l)
for i in range(num_of_segments):
    N_vec.append(xn[(i+1)*2*l-l:(i+1)*2*l+l])

mean_vec = []
for v in N_vec:
    mean_vec.append(np.sum(v)/len(v))

print('Mean: ' + str(np.mean(mean_vec)))
print('Var: ' + str(np.var(mean_vec)))
plt.hist(mean_vec, bins = 20)
plt.show()
```

```
Mean: -0.003964641531974683
Var: 0.018447605580906908
```



We observe that the variance of the mean estimate is reduced when the segment length is increased. This is confirmed when the variances is computed using the np.var-function.

```
In [ ]:
```