In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import periodogram, freqz, convolve
```

Question 1.

a)

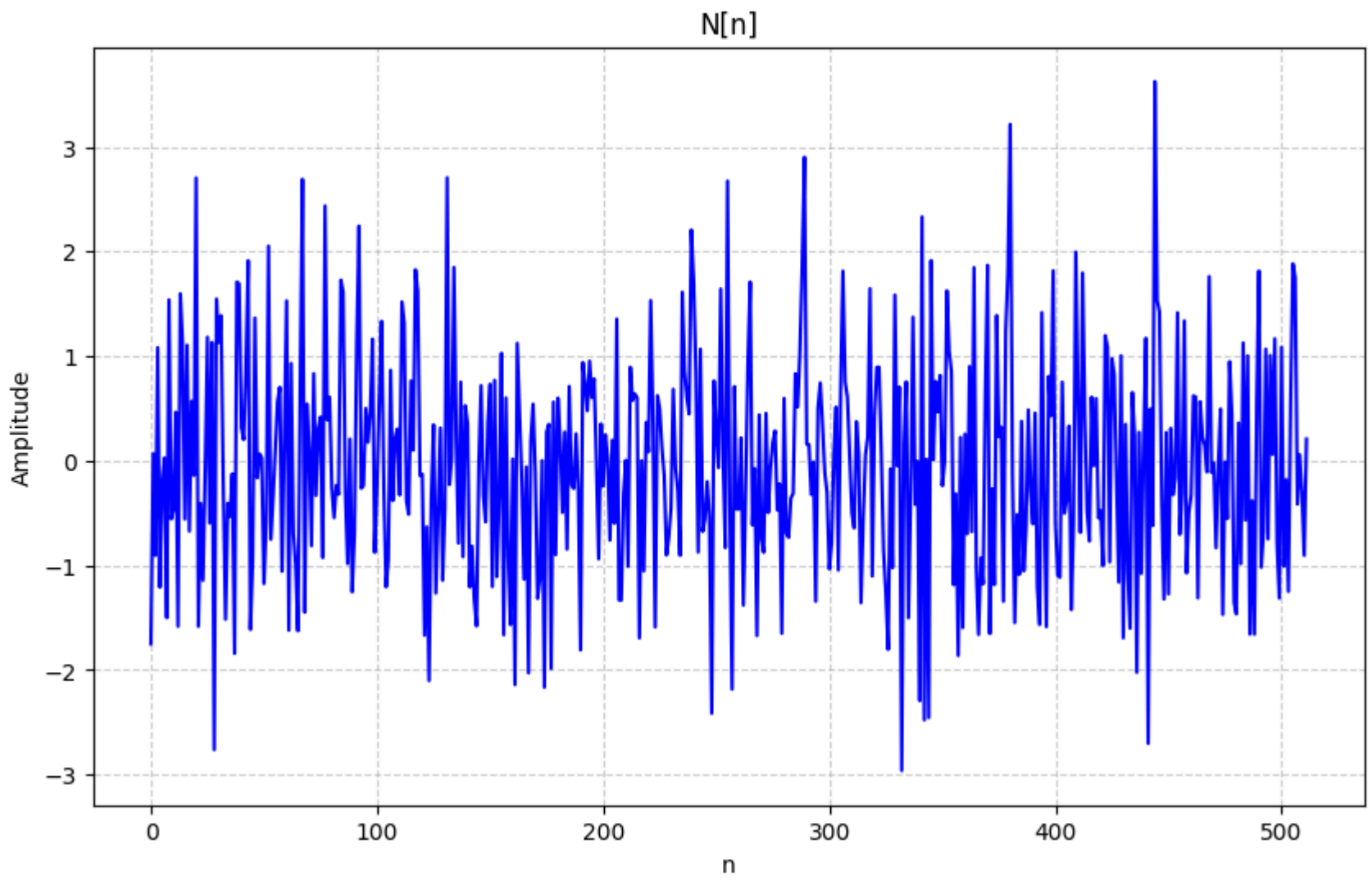
In [ ]:
```
np.random.seed(36)
Nn = np.random.normal(loc=0, scale=1, size=600)
Nn[:5]
```

Out[ ]:
```
array([ 0.67641327,  1.52109919, -0.51187625,  1.15019464, -0.59071681])
```

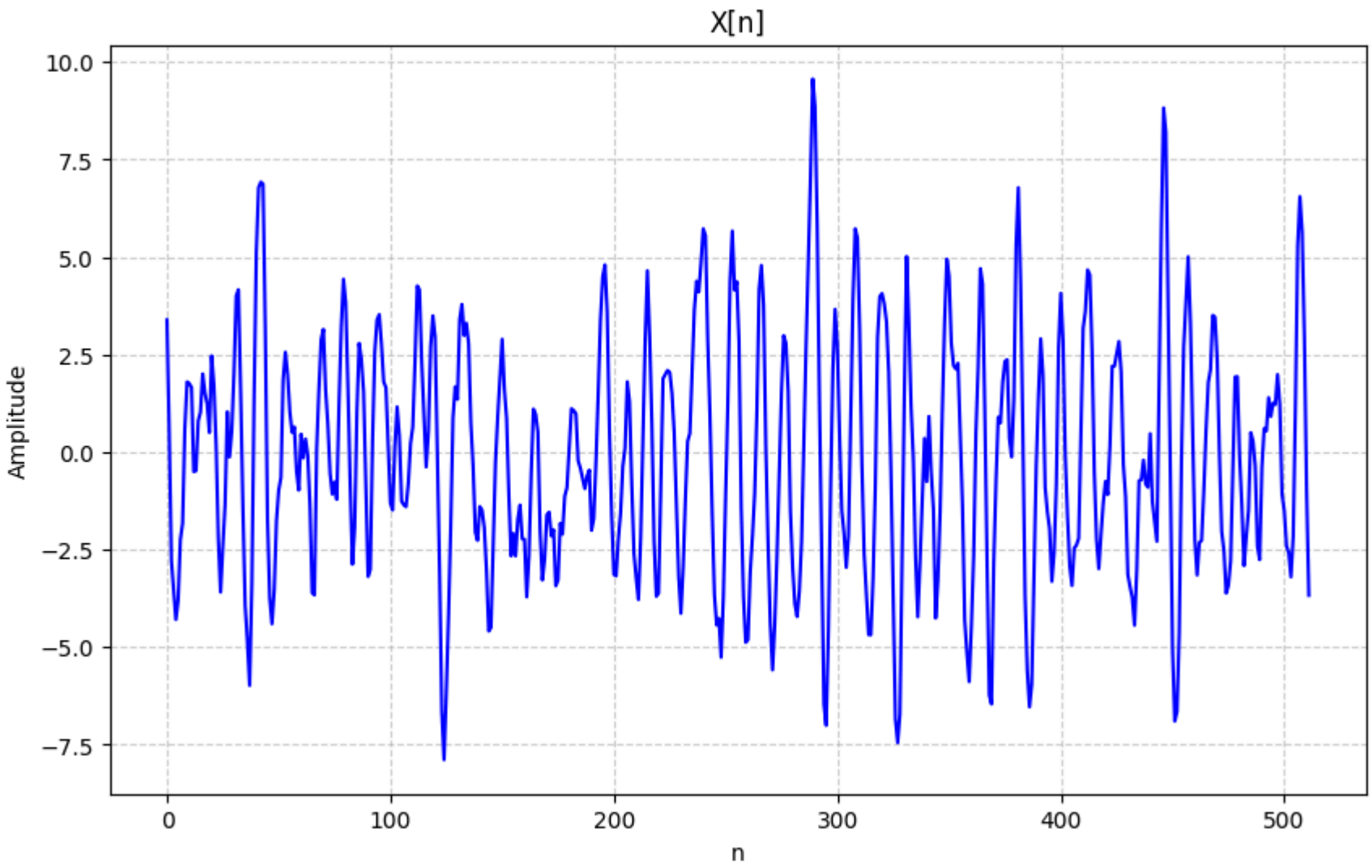
In [ ]:
```
def get_Xn(Nn):
    Xn = np.zeros(600)
    for i in range(len(Nn)):
        if i == 0:
            Xn[i] = Nn[i]
        elif i == 1:
            Xn[i] = 1.5*Xn[i-1] + Nn[i]
        else:
            Xn[i] = 1.5*Xn[i-1] - 0.8*Xn[i-2] + Nn[i]
    return Xn
Xn = get_Xn(Nn)[88:]
print(len(Xn))
```

```
512
```

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(Nn[88:], color='blue')
plt.title("N[n]")
plt.xlabel("n")
plt.ylabel("Amplitude")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



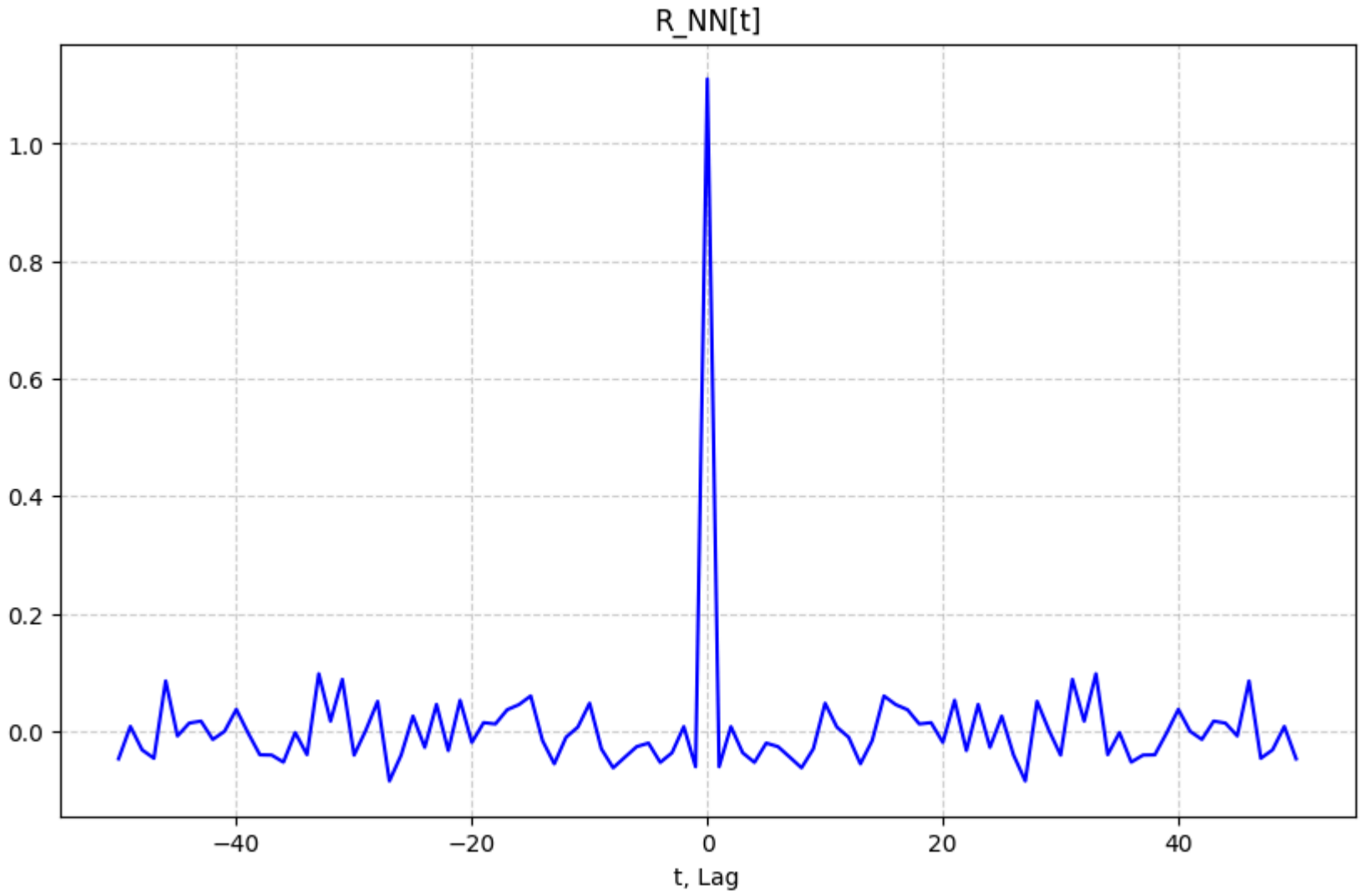
In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(Xn, color='blue')
plt.title("X[n]")
plt.xlabel("n")
plt.ylabel("Amplitude")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



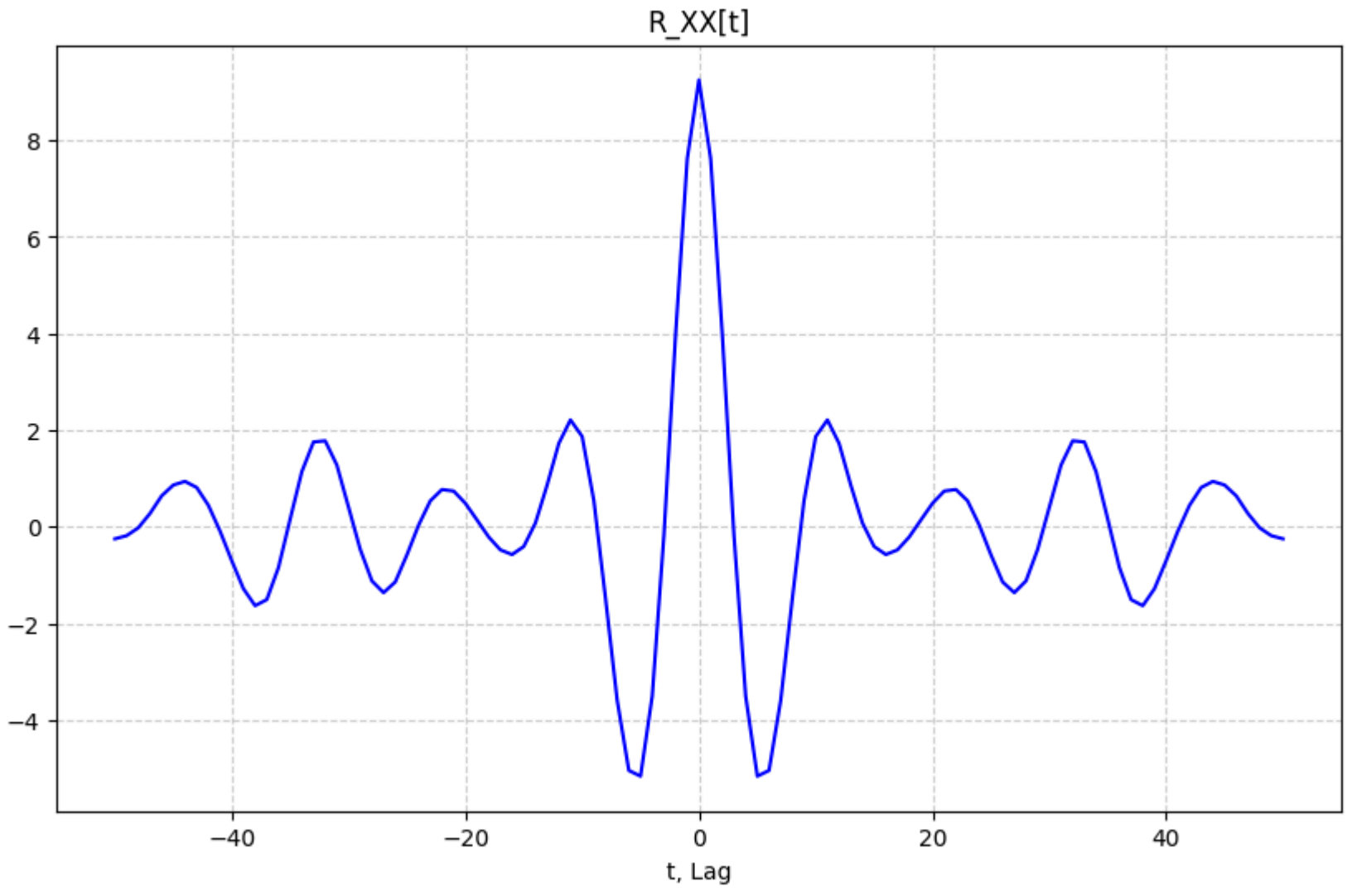
Comments: N[n] behaves as expected, resembling uncorrelated white gaussian noise where most values are in the range -1 to 1. X[n] shows an amplified version of the input signal due to accumulation like X[n-1] and its scaling factor. X[n] also shows less variations compared to N[n].

b)

In [ ]:
```
def autocorr(X, nlags):
    R = np.zeros(nlags + 1)
    n = len(X)
    for i in range(len(R)):
        t = 0
        for j in range(n-i):
            t = t + (X[j]*X[j+i])
        R[i] = t/(n-i)
    R_inv = R[1:]
    R_inv = R_inv[::-1]
    conc = np.concatenate((R_inv, R))
    return conc
R_nn = autocorr(Nn, 50)
R_xx = autocorr(Xn, 50)

L = list(range(-50,51))
plt.figure(figsize=(10, 6))
plt.plot(L, R_nn, color='blue')
plt.title("R_NN[t]")
plt.xlabel("t, Lag")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(L, R_xx, color='blue')
plt.title("R_XX[t]")
plt.xlabel("t, Lag")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



In [ ]:
```
Pn = R_nn[50]
Px = R_xx[50]
print(f"Power of N: {Pn}, Power of X: {Px}")
```

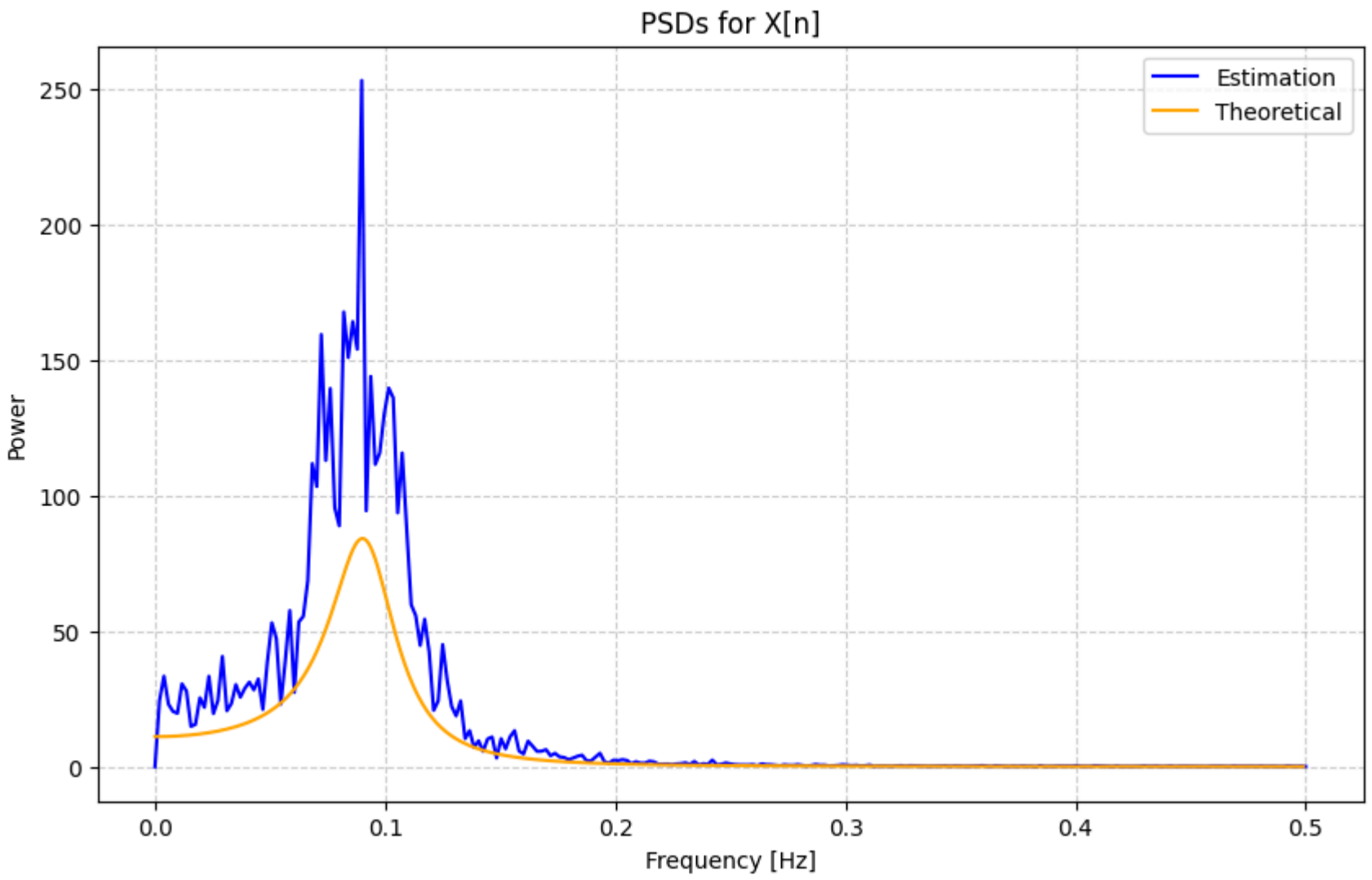
```
Power of N: 1.1101595646472882, Power of X: 9.250149426474632
```

Comments: The graph of R_NN[t] shows that N[n] is uncorrelated as the value of it is close to 0 at lags other than 0. For R_XX[t], smaller lags (X closer to each other) tend to have larger correlation. The AR formula suggests this relation.

c)

In [ ]:
```
Xns = []
for i in range(10):
```

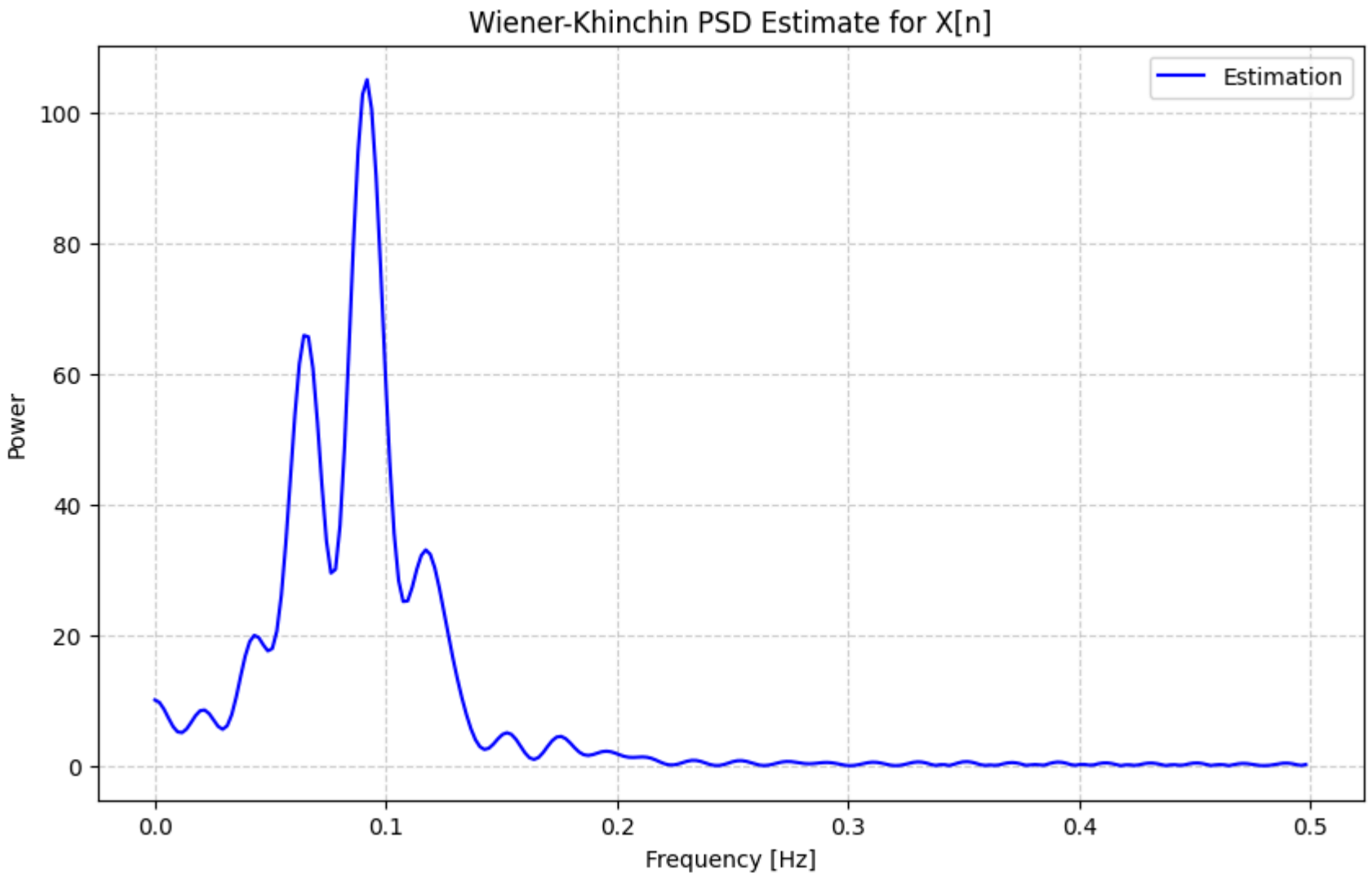
```
    Nn = np.random.normal(loc=0, scale=1, size=600)
    Xn = get_Xn(Nn)[88:]
    Xns.append(Xn)

periodograms = []
frequencies = None
for Xn in Xns:
    freq, power = periodogram(Xn, scaling='density')
    periodograms.append(power)
    if frequencies is None:
        frequencies = freq

average_power = np.mean(periodograms, axis=0)

numerator = [1.0]
denominator = [1.0, -1.5, 0.8]
w, H = freqz(numerator, denominator)

freq = w / (2 * np.pi)

plt.figure(figsize=(10, 6))
plt.plot(frequencies, average_power, label="Estimation", color='blue')
plt.plot(freq, np.abs(H)**2, label="Theoretical", color="orange")
plt.title("PSDs for X[n]")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Power")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



Comments: The estimate and true values follow the same trend with different amplitudes.

d)

In [ ]:
```
S_xx = np.fft.fft(np.concatenate((R_xx, np.zeros(512 - len(R_xx)))))
S_xx = np.fft.fftshift(S_xx)  # 512-pt fft

Nsig_x = len(S_xx)
freq_x = np.fft.fftfreq(Nsig_x)
freq_x = np.fft.fftshift(freq_x)
freq_x = freq_x[Nsig_x // 2:]

plt.figure(figsize=(10, 6))
plt.plot(freq_x, np.abs(S_xx)[Nsig_x // 2:], label="Estimation", color='blue')
plt.title("Wiener-Khinchin PSD Estimate for X[n]")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Power")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



Comments: The estimation is able to followed the same trend as the theoretical values, while having similar amplitudes unlike the periodogram. In addition, it also has less spike compared to the periodogram.

e)

In [ ]:
```
def r_x(p):
    return R_xx[51:51+p]

def R_x(p):
    return np.column_stack([R_xx[50-k:50+p-k] for k in range(p)])

p3 = np.linalg.solve(R_x(3), r_x(3))

p4 = np.linalg.solve(R_x(4), r_x(4))

p5 = np.linalg.solve(R_x(5), r_x(5))

print(f"3 element predictor: {p3}")
print(f"4 element predictor: {p4}")
print(f"5 element predictor: {p5}")
```

```
3 element predictor: [ 1.40779428 -0.66979296 -0.07317281]
4 element predictor: [ 1.40625941 -0.68384252 -0.04364294 -0.02097598]
5 element predictor: [ 1.40584533 -0.68470406 -0.05714247  0.00678457 -0.0197407 ]
```

Comments: From above, the first 2 coefficients are almost the same as the coefficients in the AR filter, [1.5, -0.8]. The rest of the coefficients in the predictors are close to 0 as the true filter only has 2 components.

Question 2.

In [ ]:
```
np.random.seed(5)
Nn = np.random.normal(loc=0, scale=1, size=600)
Nn[:5]
```

Out[ ]:
```
array([ 0.44122749, -0.33087015,  2.43077119, -0.25209213,  0.10960984])
```

In [ ]:
```
def get_Sn(Nn):
    Sn = np.zeros(600)
    for i in range(len(Sn)):
        if i == 0:
            Sn[i] = Nn[i]
        if i == 1:
            Sn[i] = 0.2*Sn[i-1] + Nn[i]
        else:
            Sn[i] = (0.2*Sn[i-1]) - (0.8*Sn[i-2]) + Nn[i]
    return Sn

Sn = get_Sn(Nn)[88:]
```

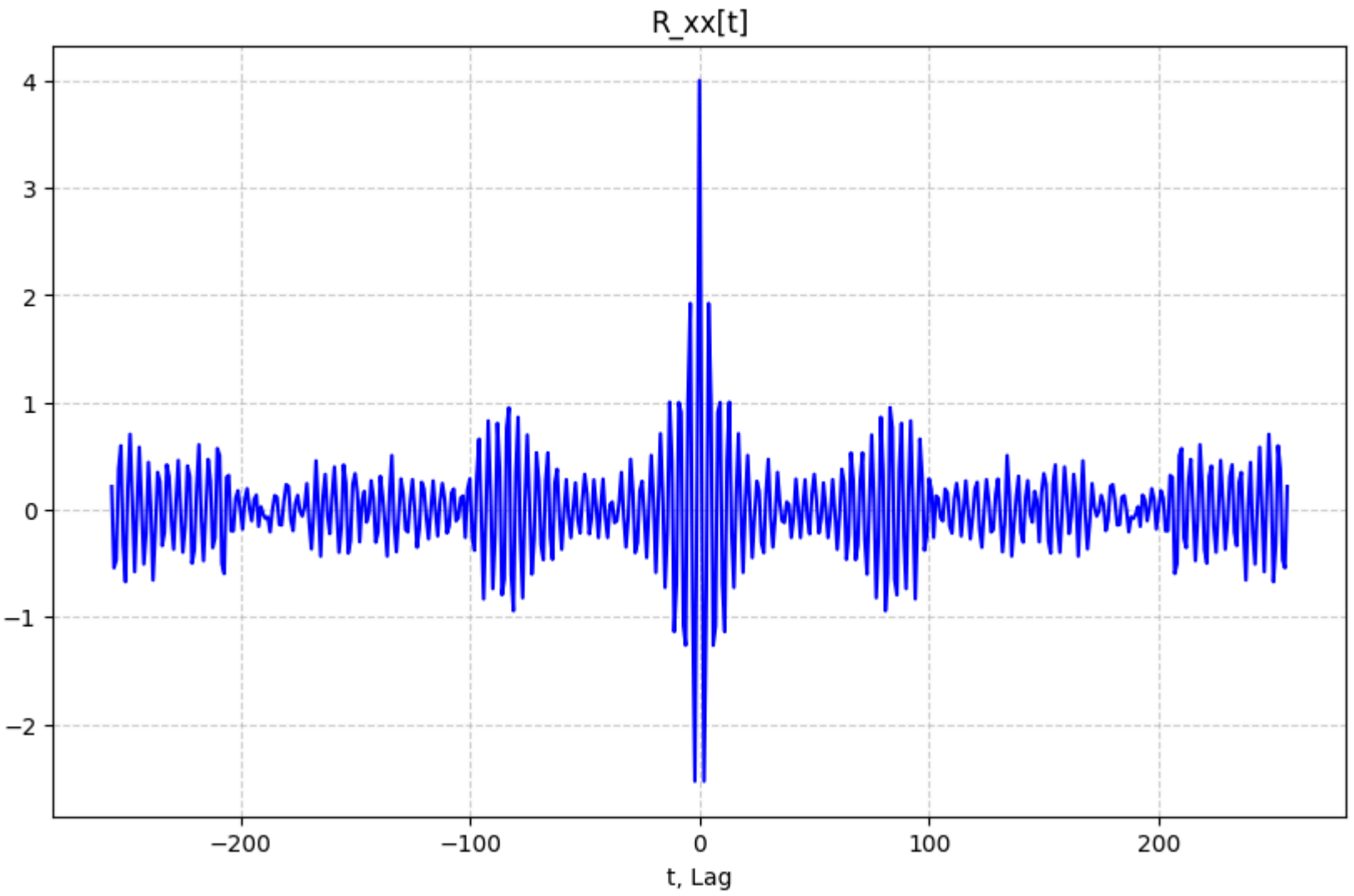
a)

In [ ]:
```
Wn = np.random.normal(loc=0, scale=1, size=len(Sn))
Xn = Sn + Wn
Xn[:5]
```

Out[ ]:
```
array([-0.19421741,  1.39742493,  3.10359799, -3.19676241, -3.41441519])
```

b)

In [ ]:
```
R_xx = autocorr(Xn, 256)

L = list(range(-256,257))
plt.figure(figsize=(10, 6))
plt.plot(L, R_xx, color='blue')
plt.title("R_xx[t]")
plt.xlabel("t, Lag")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

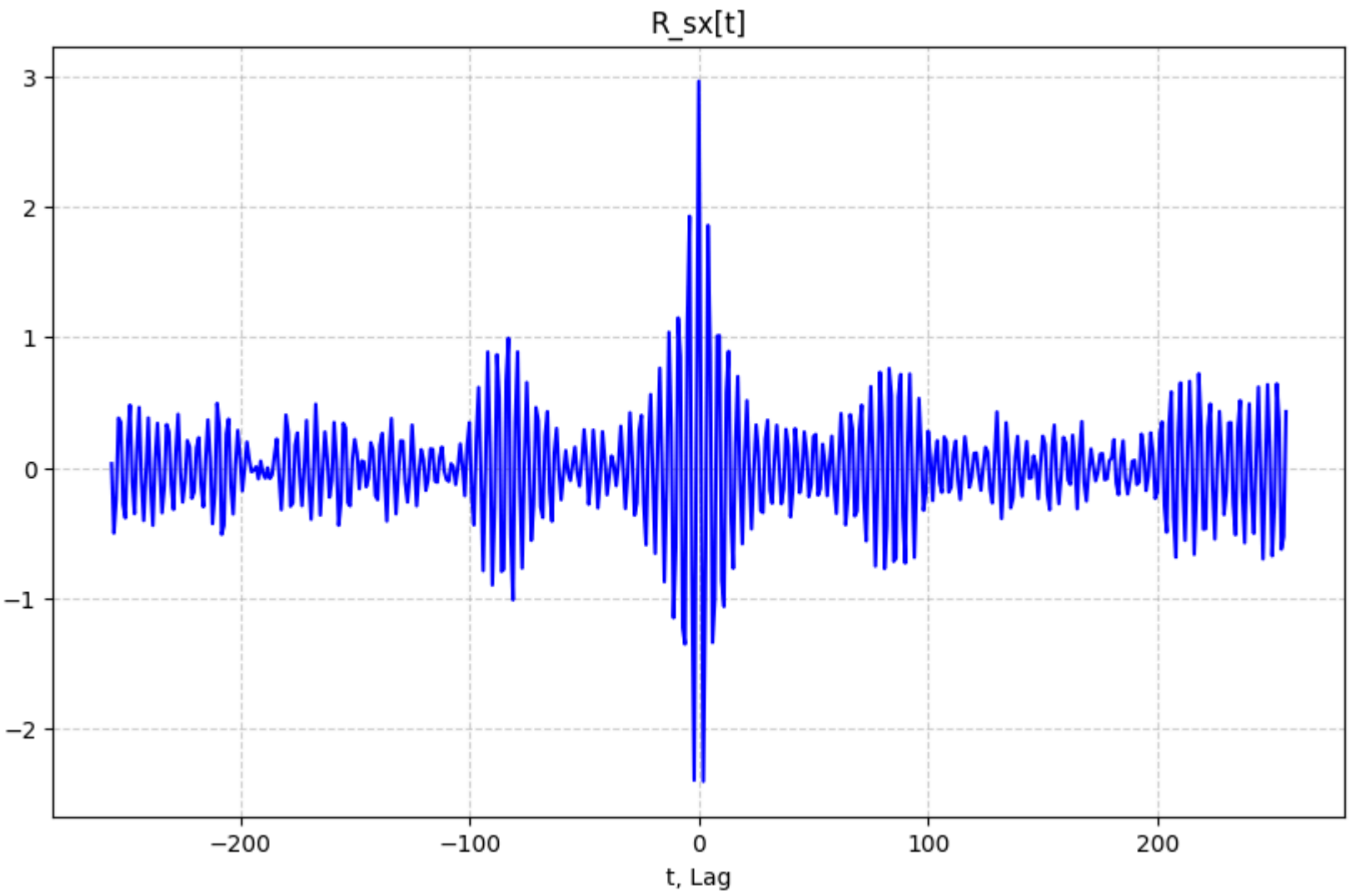


Comments: From the graph, we see that R_xx[t] is an even function. This pattern can be seen as the process is WSS.

c)

In [ ]:
```
def crosscor(X, Y, nlags):
    R_xy = np.zeros(nlags+1)
    R_xy_inv = np.zeros(nlags+1)
    n = len(X)
    for i in range(len(R_xy)):
        t = 0
        t_inv = 0
        for j in range(n-i):
            t = t + (X[j] * Y[j+i])
            k = j + i
            t_inv = t_inv + (X[k] * Y[k-i])
        t = t / (n-i)
        t_inv = t_inv / (n-i)
        R_xy[i] = t
        R_xy_inv[i] = t_inv
    R_xy_inv = R_xy_inv[1:]
    R_xy_inv = R_xy_inv[::-1]
    conc = np.concatenate((R_xy_inv, R_xy))
    return conc

R_sx = crosscor(Sn,Xn, 256)

L = list(range(-256,257))
plt.figure(figsize=(10, 6))
plt.plot(L, R_sx, color='blue')
plt.title("R_sx[t]")
plt.xlabel("t, Lag")
```

```
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



Comments: By visually inspecting the plot, the waveform appears to exhibit symmetry about t = 0. Peaks and valleys at positive lags are mirrored at corresponding negative lags, suggesting that R_sx[t] is approximately even. This is the characteristic of jointly WSS processes which can also be assumed from the relation of X[n] and S[n] where X[n] is a modified version of S[n] due to independent noise.

d)

In [ ]:
```
def r_sx(p):
    return R_sx[256:256 + p]
def R_x(p):
    return np.column_stack([R_xx[256-k:256+p-k] for k in range(p)])

p7 = np.linalg.solve(R_x(7), r_sx(7))
print(list(p7))
```

```
[0.5761980604153849, 0.037317294645904484, -0.18076014972878213, -0.02619216793448504, 0.0529548976401698, -0.00020009674162225184, -0.046817884208807656]
```

e)

In [ ]:
```
Zn = convolve(p7, Xn, mode='full')
np.var(Sn - Zn[:len(Sn)])
```

Out[ ]:
```
0.5603398558403406
```

In [ ]:
```
Ps = autocorr(Sn, 50)[50]
mse = Ps - np.dot(p7, R_sx[256:256+7])
mse
```

Out[ ]:
```
0.5744157885701822
```

Comments: The optimal linear estimator closely approaches the minimum mean squared error.

In [ ]:
```
%%shell
jupyter nbconvert --to html /content/Colab_downladed_file_name.ipynb
```