Imperial College London

COURSEWORK 1

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Time Series

Author:
(CID: 01844537)

Date: December 16, 2022

Question 1

1(a)

For this question, let us first consider what values $Var\{X_0\}$, $Cov\{X_0, \epsilon_0\}$ and $Var\{\epsilon_0\}$ take, so that we can construct matrix D. We already know what $Var\{\epsilon_0\}$, since the value for σ_ϵ^2 is provided as one of the arguments for our **ARMA11** function. Therefore, we just need to determine what $Var\{X_0\}$ and $Cov\{X_0, \epsilon_0\}$ are. To do so, let us consider what **ARMA(1,1)** is written in General linear form:

$$X_t - \phi X_{t-1} = \epsilon_t - \theta\epsilon_{t-1}$$
$$\implies X_t = \frac{\Theta(B)}{\Phi(B)}\epsilon_t$$
$$\Phi(B) = 1 - \phi B \qquad \Theta(B) = 1 - \theta B$$

For this to be invertible and stationary, we must have $|\theta| < 1$, $|\phi| < 1$. Consider $G(z) = \frac{\Theta(z)}{\Phi(z)} = \frac{1-\theta z}{1-\phi z}$. Since we have $|\phi| < 1$, we can expand $G(z)$ (from lecture notes):

$$\frac{1-\theta z}{1-\phi z} = (1-\theta z)\sum_{k=0}^{\infty}(\phi z)^k = 1 + (\phi - \theta)\sum_{k=1}^{\infty}\phi^{k-1}z^k$$

and X_t can be written as (from lecture notes):

$$X_t = \sum_{k=0}^{\infty} g_k\epsilon_{t-k}, \qquad g_k = \begin{cases} 1 & \text{if } k = 0 \\ (\phi-\theta)\phi^{k-1} & \text{if } k \geq 1 \end{cases}$$

and $E\{X_t\} = 0$ since it's just a sum of zero-mean white noise Gaussian processes.

Therefore, we have:

$$Cov\{X_0, \epsilon_0\} = E\{X_0\epsilon_0\} - E\{X_0\}E\{\epsilon_0\} = E\left\{\epsilon_0 \sum_{k=0}^{\infty} g_k\epsilon_{t-k}\right\} = E\left\{g_0\epsilon_0^2\right\} = \sigma_\epsilon^2$$
$$Var\{X_0\} = \sigma_\epsilon^2\left(1 + \frac{(\phi-\theta)^2}{1-\phi^2}\right) \quad \text{(From lecture notes)}$$

After we obtain the matrix C from the Cholesky decomposition of D, the product CY would give us the simulated values for X_0 and ϵ_0, and from then we can simulate the rest of the values by applying the ARMA formula. Y is the length-2 vector whose entries are independent samples from standard normal distribution.

Below is the code for this question.

```
// Time_Series.py
import numpy as np

def ARMA11(phi, theta, sigma2, N):
    """
    Simulates N values from an zero mean Gaussian ARMA(1,1) process, with
    given values of phi, theta and sigma squared

```

```
    :param phi: phi value used in the ARMA process
    :param theta: theta value used in the ARMA process
    :param sigma2: variance of the white noise process
    :param N: length of process simulated

    :return X: the length N simulated time series
    """
    D = sigma2 * np.ones((2, 2)) # Initialise D
    D[0][0] *= (1 + (phi-theta) ** 2 / (1 - phi ** 2))
    Y = np.random.normal(size = (2))
    C = np.linalg.cholesky(D) # Find Cholesky decomposition
    X0, epsilon = (C @ Y)[0], (C @ Y)[1]
    X = [X0] # Create list to store output
    for i in range(N):
        Et = np.random.normal(0, np.sqrt(sigma2))
        Xi = phi * X[-1] + Et - theta * epsilon
        X.append(Xi) # Keeping track of the Xt values
        epsilon = Et # Keeping track of the epsilon_t values
    X.pop(0)
    return X
```

1(b)

The code for this **acvs** is provided below:

```
def acvs(X, tau):
    """
    Returns the autocovariance sequence estimator for a time
    series stored in vector X at lags in vector
    tau

    :param X: array to estimate the autocovariance of
    :param tau: vector of lags

    :return acvs_arr: array of estimated autocovariance values at     given
   lags
    """
    N, tau =len(X)
    tau = np.abs(tau) # Converts the lags to positive values
    acvs_arr = []
    for i in tau: # Iterates through differnet lag values
        if np.abs(i) >= N: # Sets lags outside of range to 0
            acvs_arr.append(0)
        else:
            estimator_sum = 0
            X_bar = np.mean(X)
            for t in range(N-i):
                estimator_sum += (X[t] - X_bar) * (X[t+i] - X_bar)
                acvs_arr.append(estimator_sum / N)
    return np.array(acvs_arr)
```

1(c)

The code for this **periodogram** is provided below:

```
def periodogram(X):
    """
    Computes the estimated periodogram values at Fourier frequencies
```

```

    :param X: vector to compute the periodogram for
    :param taper: indicates whether a taper is used or not

    :return S_shift: An array of values for the periodogram at
    Fourier frequencies
    :return f: An array that consists of Fourier frequencies
    """
    N = len(X)
    S_hat = (1 / N) * np.abs(np.fft.fft(X)) ** 2 # Apply FFT
    f = np.fft.fftfreq(N) # Obtain the Fourier frequencies
    S_shift = np.fft.fftshift(S_hat) # Apply shift to recenter
    f = np.fft.fftshift(f) # Apply the same shift to frequency array
    return S_shift, f
```

Question 2

2(a), 2(b), 2(c)

```
def Q2(i):
    """
    This function calculates the required statistics for an ARMA
    process of a specified length
    :param i: used to specify the length of simulated ARMA process

    :returns np.mean(SN4): the sample mean of SN4; SN4 is the
    array of Periodogram values at Fourier frequency f_{N/4}
    :returns np.var(SN4, ddof=1): the sample variance of SN4
    :returns SN4: the array of values at Fourier frequency f_{N/4}
    for ARMA processes of length N
    :returns SN4_star: the array of values at Fourier frequency
    f_{(N/4)+1} for ARMA processes of length N
    """
    phi, theta, sigma2 = 0.59, -0.8, 2.38 # Assigning parameter
    # values
    SN4, SN4_star = [], [] # Used to store outputs
    N_list = [4, 8, 16, 32, 64, 128, 256, 512] # List of lengths to
    # used to simulate ARMA process
    N = N_list[i]
    for j in range(10000):
        Xvec = ARMA11(phi, theta, sigma2, N) # simulate the process
        Periodogram = periodogram(Xvec)[0] # calculate the periodogram
        N4 = int(N/4)
        SN4.append(Periodogram[N4]) # Get value associated to f_{N/4}$
        SN4_star.append(Periodogram[N4+1]) # f_{N/4 + 1} values
    return np.mean(SN4), np.var(SN4, ddof=1), SN4, SN4_star

def Sf(f):
    """
    Calculates the spectral density of an ARMA(1,1) process at a given
    frequency

    :param f: frequency to calculate spectral density at
    """
    fv = sigma2 * (np.abs((1-theta*(np.exp(-2*np.pi*1j*f))) / (1-phi*(np.exp
    (-2*np.pi*1j*f)))))**2
    return fv

```

```python
SN4_mean_arr, SN4_var_arr, SN4_sample_arr, SN4_star_sample_arr = [], [], [], []
# Generating values for different lengths of ARMA processes
for i in range(8):
    V = Q2(i)
    SN4_mean_arr.append(V[0]) # Getting sample means
    SN4_var_arr.append(V[1]) # Getting sample variances
    SN4_sample_arr.append(V[2]) # Storing S^(p)(f_{N/4}) values
    SN4_star_sample_arr.append(V[3]) # Storing S^(p)(f_{N/4 + 1}) values

x_arr2 = [2 ** (i+2) for i in range(8)] # Generate values of N to plot on x-axis
fig1 = plt.figure(figsize=(15, 5))
fig1.suptitle('Plots for parts a, b and c', y = 1.03, fontsize='15')

ax1 = plt.subplot(1, 3, 1)
ax1.set_title('Sample mean and its large\n sample result for different N')
plt.plot(x_arr2, SN4_mean_arr, label="Sample mean of $\hat{S}^{(p)}(f_{\\frac{N}{4}})$")
plt.axhline(Sf(1/4), color='red', label="S(1/4)") # Large sample result for mean
plt.legend(fontsize='12'), plt.xlabel("N")

ax2 = plt.subplot(1, 3, 2)
ax2.set_title('Sample variance and its large\n sample result for different N')
plt.plot(x_arr2, SN4_var_arr, label="Sample variance of $\hat{S}^{(p)}(f_{\\frac{N}{4}})$")
plt.axhline(Sf(1/4)**(2), color = 'red', label="$S^{2}$(1/4)") # Large sample result for variance
plt.legend(fontsize='12'), plt.xlabel("N")

cor = [] # Calculate the correlation cofficients at frequencies f_{N/4} and f_{N/4+1}
for i in range(8):
    cor.append(np.corrcoef(SN4_sample_arr[i], SN4_star_sample_arr[i])[0][1])

ax3 = plt.subplot(1, 3, 3)
ax3.set_title('Sample correlation coefficient and\n its large sample result for different N')
plt.plot(x_arr2, cor, label='corr{$\hat{S}^{(p)}(f_{\\frac{N}{4}}), \hat{S}^{(p)}(f_{\\frac{N}{4}+1})$}')
plt.axhline(0, color='red', label='0') # Large sample result for correlation
plt.legend(fontsize='12'), plt.xlabel("N")
```

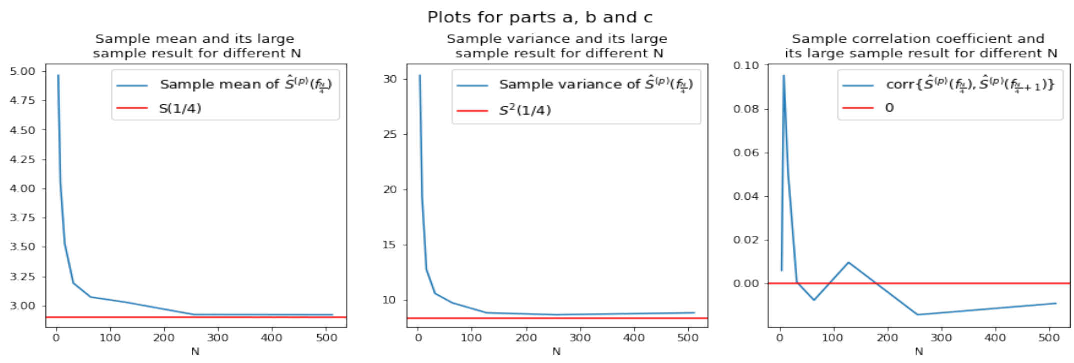

Figure 1: Plots for parts a, b and c

Comments: when accessing the value at Fourier frequency $f_{\frac{N}{4}}$, the index used was $\frac{N}{4}$, despite Python being 0-indexed. If we consider the output array when calling **periodogram**, the outputted array is the array of estimated values for $S(f)$ at Fourier frequencies f_k, which in our case are $-0.5, -0.499, -0.498\dots$. We are interested in the value at $f_{\frac{N}{4}}$. By definition of Fourier frequency, $f_{\frac{N}{4}}$ is just equal to $\frac{1}{4}$. Also, since periodogram is symmetric around y-axis, we have $\hat{S}^{(p)}(\frac{1}{4}) = \hat{S}^{(p)}(-\frac{1}{4})$. $\hat{S}^{(p)}(-\frac{1}{4})$ is the $(\frac{N}{4}+1)$-th element in the outputted array, hence why we take the value at index $\frac{N}{4}$.

The formula for spectral density at frequency f for an ARMA(1,1) process is given by: $S(f) = \sigma_\epsilon^2 \left|\frac{1-\theta e^{-i2\pi f}}{1-\phi e^{-i2\pi f}}\right|^2$ and is what's used in the function **Sf** in the code. In the figures, we can see that the statistics in our time series do tend towards their predicted asymptotic behaviours when N becomes large. Whilst the graph for correlation doesn't quite look as convincing as the others, we can see that this is due to the scaling on the y-axis. By observing the scale, we can see that indeed the sample correlation is still converging towards 0, with $corr \approx -0.01$ when $N = 512$, very close to 0.

2(d), 2(e), 2(f)

```
fig2 = plt.figure(figsize=(15, 5)), from scipy.stats import chi2
x_arr2 = np.linspace(0,15,30)
fig2.suptitle('Plots for parts d, e and f', y = 1.05, fontsize='15')

ax1 = plt.subplot(1, 3, 1)
ax1.set_title('Histogram of $\\frac{2}{S(f)}\hat{S}^{(p)}(f)$ versus $\chi_{2}^{2}$, \nN=4, f=1/4'
               , fontsize='12')
# Plotting the scaled histogram
plt.hist((2/Sf(1/4)) * np.array(SN4_sample_arr[0]), bins=200, density = True, label=
         '$\\frac{2}{S(1/4)}\hat{S}_{j}^{(p)}(\\frac{1}{4})$, N=4')
# Plotting chi-squared on the same scale
plt.plot(x_arr2, chi2.pdf(x_arr2, df=2), label = '$\chi_{2}^{2}$')
plt.legend(fontsize=12)

ax2 = plt.subplot(1, 3, 2)
ax2.set_title('Histogram of $\\frac{2}{S(f)}\hat{S}^{(p)}(f)$ versus $\chi_{2}^{2}$, \nN=32, f=1/4'
               , fontsize='12')
# Plotting the scaled histogram
plt.hist((2/Sf(1/4)) * np.array(SN4_sample_arr[2]), bins = 200, density = True, label=
         '$\\frac{2}{S(1/4)}\hat{S}_{j}^{(p)}(\\frac{1}{4})$, N=32')
# Plotting chi-squared on the same scale
plt.plot(x_arr2, chi2.pdf(x_arr2, df=2), label = '$\chi_{2}^{2}$')
plt.legend(fontsize=12)

ax3 = plt.subplot(1, 3, 3)
ax3.set_title('Histogram of $\\frac{2}{S(f)}\hat{S}^{(p)}(f)$ versus $\chi_{2}^{2}$, \nN=256, f=1/4'
               , fontsize='12')
# Plotting the scaled histogram
plt.hist((2/Sf(1/4)) * np.array(SN4_sample_arr[-2]), bins = 200, density = True, label=
         '$\\frac{2}{S(1/4)}\hat{S}_{j}^{(p)}(\\frac{1}{4})$, N=256')
```

```python
# Plotting chi-squared on the same scale
plt.plot(x_arr2, chi2.pdf(x_arr2, df=2), label = '$\chi_{2}^{2}$')
plt.legend(fontsize=12)
```

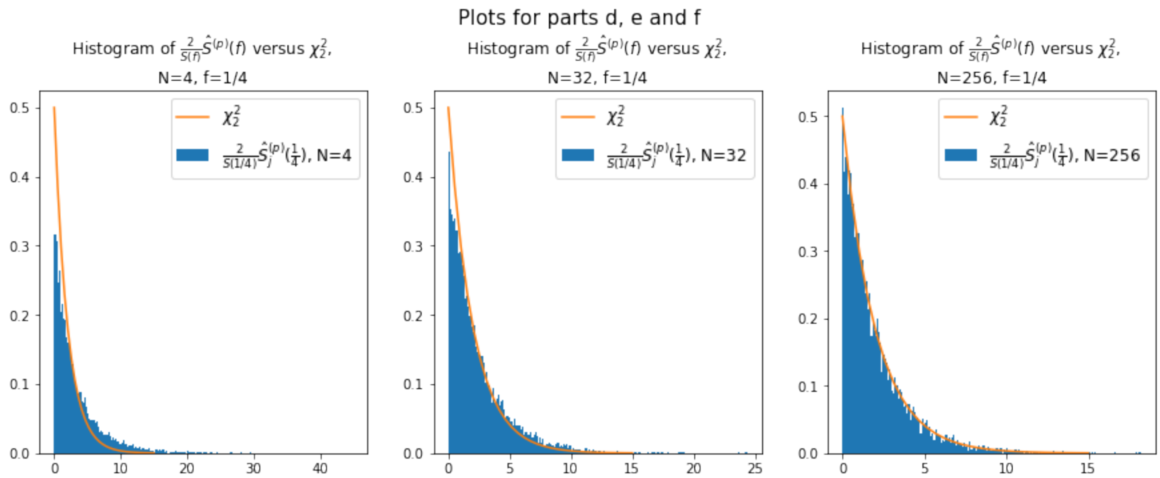

Figure 2: Plots for parts d, e and f

Comments: in the plots, we compared the pdf of χ_2^2 with the histogram of sampled values of $\{\hat{S}_j^{(p)}(\frac{1}{4})\}$ divided by $\frac{S(\frac{1}{4})}{2}$. We can do that, since $\hat{S}^{(p)}(f) \overset{d}{=} \frac{S(f)}{2}\chi_2^2 \Leftrightarrow \frac{2}{S(f)}\hat{S}^{(p)}(f) \overset{d}{=} \chi_2^2$. Expected asymptotic behaviour is again shown when we increase N, as evidenced in the graphs. We can see that the fit of the histograms to χ_2^2 become better as we increase N.

Question 3

3(a)

We first introduce the function **cos_taper**, which returns the tapered array $\{h_t X_t\}$ for a given array $\boldsymbol{X}$, using a 50% cosine taper. We also propose the following changes to functions **acvs** and **periodogram** so that they also work for centered tapered data (docstrings and comments for the modified functions are omitted to save space):

```python
from scipy.linalg import toeplitz

def cos_taper(X):
    """
    Creates a vector of real value constants that applies 50% cosine taper
    for a given vector

    :param X: vector to apply taper to

    :return X1: tapered vector
    """
    N = len(X)
    ht = np.zeros(N) # Creates an array to store the taper sequence
    p = 0.5
    j = int(np.floor(p*N))
    for i in range(N): # Checking condition on the indices
        if i <= j/2 - 1:
            ht[i] = (1/2 * (1-np.cos(2*np.pi*(i+1) / (j+1))))
```

```
        elif j/2-1 < i and i < (N - j/2):
            ht[i] = 1
        else:
            ht[i] = (1/2 * (1-np.cos(2*np.pi*(N+1-(i+1)) / (j+1))))
    Norm = np.linalg.norm(ht) # Calculate the norm of the taper
    ht = ht / Norm # Scale the sequence so that it has norm 1
    X1 = np.multiply(ht, X) # multiply X by the sequence element wise
    return X1
def acvs(X, tau, taper=False):
    N, tau =len(X)
    tau = np.abs(tau)
    acvs_arr = []
    for i in tau:
        if np.abs(i) >= N:
            acvs_arr.append(0)
        else:
            estimator_sum = 0
            X_bar = np.mean(X)
            for t in range(N-i):
                estimator_sum += (X[t] - X_bar) * (X[t+i] - X_bar)
            if taper:
                acvs_arr.append(estimator_sum)
            else:
                acvs_arr.append(estimator_sum / N)
    return np.array(acvs_arr)

def periodogram(X, taper=False):
    N = len(X)
    if taper:
        S_hat = np.abs(np.fft.fft(X)) ** 2
        f = np.fft.fftfreq(N)
        S_shift = np.fft.fftshift(S_hat)
        f = np.fft.fftshift(f)
        return S_shift, f
    S_hat = (1 / N) * np.abs(np.fft.fft(X)) ** 2
    f = np.fft.fftfreq(N)
    S_shift = np.fft.fftshift(S_hat)
    f = np.fft.fftshift(f)
    return S_shift, f
```

The modified **acvs** has an option to take in a centered tapered array (i.e. $\boldsymbol{X} = \{h_t X_t\}$, $\sum_{t=1}^{N} h_t^2 = 1$) and returns its modified autocovariance sequence estimator. Note that in the code, $\bar{X}$ isn't removed, despite the formula for modified autocovariance sequence estimator is defined as $\hat{s}_\tau = \sum_{t=1}^{N-|\tau|} h_t X_t h_{t+|\tau|} X_{t+|\tau|}$. This is because the centered data has sample mean 0, Hence despite that $\bar{X}$ being still present in the tapered **acvs** function, it won't affect the results since $\bar{X} = 0$ and doesn't contribute to calculations.

We run the following code block to extract our data and generate a plot for the time series:

```
import pandas as pd
df = pd.read_csv(r'/Users/larrywang/Downloads/time_series_85.csv', header=
    None)
arr = df.to_numpy() # Read the data
arr = arr.flatten()
plt.plot(np.linspace(1, 730, 730), arr), plt.title('Plot of daily energy
    consumption in Germany') # Plot the time series
plt.xlabel('Day number'), plt.ylabel('Energy Consumption(GWh)')
```

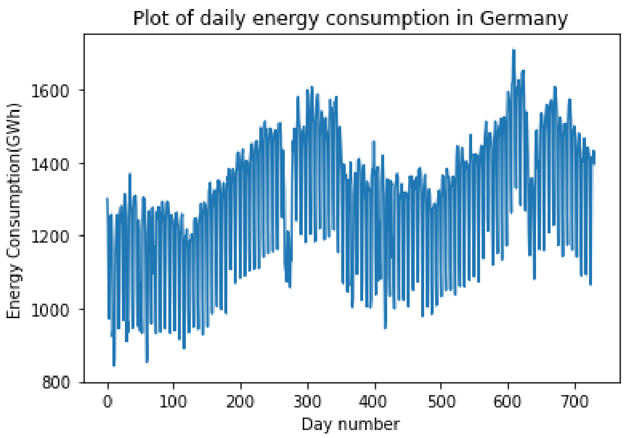

Figure 3: Plot of the time series

Now we center our data and generate the direct spectral estimator:

```
T_mean = np.mean(arr) # Remove the mean
arr -= T_mean
arr1 = cos_taper(arr) # Create a new tapered array
Sd = periodogram(arr1, taper=True)[0] # Generate periodogram
plt.plot(np.linspace(-1/2, 1/2, 730), Sd)
# Highlighting important frequencies with vertical dotted lines
plt.axvline(1/365, ymin=0.05, ymax=0.65, ls='--', color='red', label='freq=$\pm 1/365$')
plt.axvline(1/7, ymin=0.05, ymax=0.99, ls='--', color='yellow', label='freq=$\pm 1/7$')
plt.axvline(-1/7, ymin=0.05, ymax=0.99, ls='--', color='yellow')
plt.axvline(1/3.5, ymin=0.05, ymax=0.27, ls='--', color='purple', label='freq=$\pm 2/7$')
plt.axvline(-1/3.5, ymin=0.05, ymax=0.27, ls='--', color='purple')
plt.axvline(3/7, ymin=0.05, ymax=0.15, ls='--', color='black', label='freq=$\pm 3/7$')
plt.axvline(-3/7, ymin=0.05, ymax=0.15, ls='--', color='black')
plt.title("Direct Spectral Estimator"), plt.xlabel("freq (1/d)"), plt.legend()
```

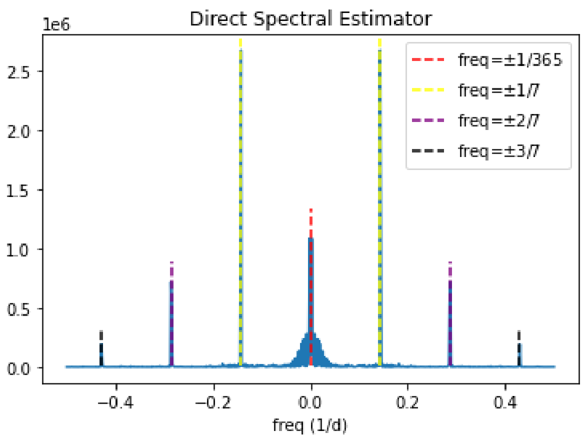

Figure 4: Direct Spectral Estimator

If we don't remove the mean from the data, then we would get a huge peak that occurs at

0 that dominates over all other peaks, which makes it difficult to perform any meaningful analysis. This is because the value of the Fourier Transform of a vector $\boldsymbol{X} = \{X_t\}$ of length N at $f = 0$ is just a sum over all of $\boldsymbol{X}$'s entries, which is equal to $N\bar{X}$ ($\bar{X}$ is the sample mean of $\boldsymbol{X}$). During the calculation for the direct spectral estimator, we are taking the absolute value of $N\bar{X}$ and then squaring it, giving us $(N\bar{X})^2$. If we have a non-zero mean, the term $(N\bar{X})^2$ would blow up (since $N = 730$ in our time series), which in turn produces a huge peak at 0. Hence it is essential that we center our data before calculating the estimator.

In the plots, important peaks are highlighted with vertical dotted lines. We can see that the most dominant peaks occur at $freq = \pm\frac{1}{7}$. This makes sense, as it demonstrates cyclicality at a weekly level; one would expect household electricity consumption to show strong periodic behaviour in weekly windows. The next highest peak occurs at $freq = \pm\frac{1}{365}$. This also makes sense, as it demonstrates cyclicality at a yearly level; one would also expect household electricity consumption to show strong periodic behaviour in yearly windows. Finally, we also observe that lower peaks occur at $freq = \pm\frac{2}{7}, \pm\frac{3}{7}$ i.e. multiples of $\frac{1}{7}$. These lower peaks occur most likely as a byproduct of the dominant peaks at $\pm\frac{1}{7}$. We can interpret $\pm\frac{1}{7}$ as the fundamental frequency, and $\pm\frac{2}{7}, \pm\frac{3}{7}$ are harmonic frequencies that arise as a result of the fundamental frequency. This would also explain why there is a cluster of low peaks around $\pm\frac{1}{365}$. Those can be interpreted as harmonics that arise from the fundamental frequency at $\pm\frac{1}{365}$.

3(b)

The code block below is for Yule-Walker method:

```python
def YW_fit_AR(X, p):
    """
    Fits an AR(p) model, given X1, X2... XN and p

    :param X: set of values to fit AR model on
    :param p: number of paramter values to fit for

    :return phi: fitted set of parameter values
    :return sigma2_hat: estimator for sigma^2
    """
    X1 = cos_taper(X) # Apply taper to the data
    i_arr = np.linspace(0, p, p+1, dtype='int32')
    acvs_arr = acvs(X1, i_arr, True) # Acvs at lag 0 to lag p
    A = np.array(acvs_arr[1:p+1])
    C = toeplitz(acvs_arr[:p]) # Creates the Toeplitz matrix
    phi = np.linalg.inv(C) @ A # Calculate phi
    sigma2_hat = acvs_arr[0]
    for i in range(1,p+1): # Calculate sigma2 estimator
        sigma2_hat -= phi[i-1] * acvs_arr[i]
    return phi, sigma2_hat
```

Listing 1: Yule Walker Code

The code block below is for Maximum Likelihood method:

```python
def ML_fit_AR(X, p):
    """
    Fits an AR(p) model, given X1, X2... XN and p, using Maximum
    Likelihood method

    :param X: set of values to fit AR model on
```

```
    :param p: number of paramter values to fit for

    :return phi: fitted set of parameter values
    :return sigma2_hat: estimator for sigma^2
    """
    X1 = X[p:] # Slice X to get entries from Xp onwards
    N = len(X)
    F = np.ones((N-p, p)) # Creates a matrix to store outputs
    for i in range(N-p):
        F[i,:] = np.flip(X[i:i+p])
    phi = np.linalg.inv(F.T @ F) @ F.T @ X1 # Calculate phi
    sigma2_hat = ((X1 - F @ phi).T @ (X1 - F @ phi)) / (N-2*p) # Calculates
    sigma2 estimator
    return phi, sigma2_hat
```

Listing 2: Max Likelihood Code

3(c)

The code below is used to apply the Ljunge-Box test for given p, h and α values.

```
from scipy.stats import chi2

def LB_test(X, p, option, h, alpha):
    """
    Implements the Ljunge-Box test for X for given p and method

    :param X: array to implement the test on
    :param p: order of the AR process
    :param option: indicates whether Yule-Walker method or Maximum
    Likelihood is used. 'YW' indicates
    Yule Walker and 'ML' indicate Maximum Likelihood
    :param h: degrees of freedom in the chi-squared distribution
    :param alpha: used to specify the quantile of the chi-squared
    distribution

    :return False: returns False if null hypothesis is rejected
    :return phi, sigma2: returns the corresponding phi parameters and the
    variance of residuals when null hypothesis isn't rejected
    """
    N = len(X)
    # Obtain the (1-alpha)-th quantile from chi-squared distribution
    # with h degrees of freedom
    Bound = stats.chi2.ppf(1-alpha, h)
    if option == 'YW': # Obtain phi and sigma2
        phi, sigma2 = YW_fit_AR(X, p)
    else:
        phi, sigma2 = ML_fit_AR(X, p)
    F = np.ones((N-p, p))
    # Calculate the residual
    for i in range(N-p):
        F[i,:] = np.flip(X[i:i+p])
    Res = X[p:] - F @ phi
    std = np.std(Res, ddof=1) # Calculate sample std of residual
    n = len(Res)
    # Creates the test statistic
    L = 0
    s_0 = acvs(Res, np.array([0]))
    for k in range(1, h+1):
        # Calculate the autocorrelations for residual array at lag-k
```

```python
        rho_k = acvs(Res,np.array([k])) / s_0
        L += (n*(n+2)*(rho_k)**2)/(n-k)
    if L > Bound:
        return False
    else:
        return phi, sigma2, std

p, h, a = 1, 14, 0.05
Test1 = LB_test(arr, p, 'YW', h, a)
while not LB_test(arr, p, 'YW', h, a):
    p += 1
    Test1 = LB_test(arr, p, 'YW', h, a)
print(p, Test1) # Obtain the smallest p and parameters values for YW method

p = 1
Test2 = LB_test(arr, p, 'ML', h, a)
while not LB_test(arr, p, 'ML', h, a):
    p += 1
    Test2 = LB_test(arr, p, 'ML', h, a)
print(p, Test2) # Obtain the smallest p and parameters values for ML method
```

Upon running the above code block, we discover that for both Yule-Walker and Maximum Likelihood method, we get that $p = 22$ is the smallest p such that we fail to reject the null hypothesis. The estimated parameter values we obtain are:

$$
\begin{aligned}
\hat{\phi}_{YW} = [&0.64314405, \; 0.01333134, \; 0.15580555, -0.11004447, \; 0.03447730, \\
&0.08693418, \; 0.37209604, -0.26268911, -0.08278800, -0.01020829, \\
&0.11230805, -0.07150802, -0.08593831, \; 0.23165883, -0.11843832, \\
&0.02724599, -0.12268959, -0.02942018, \; 0.03624358, \; 0.01307124, \\
&0.32334694, -0.2123953] \\
\hat{\phi}_{ML} = [&0.64044323, \; 0.03248817, \; 0.10657801, -0.08814136, \; 0.02807658, \\
&0.07724485, \; 0.36821482, -0.22697947, -0.07227281, \; 0.02421634, \\
&0.07908561, -0.06989360, -0.10114759, \; 0.22674965, -0.11720951, \\
&0.01218540, -0.12158637, \; 0.00286982, \; 0.02172707, \; 0.05268403, \\
&0.33456135, -0.25512776] \\
(\hat{\sigma}^2_\epsilon)_{YW} =& 2751.9789415281152 \\
(\hat{\sigma}^2_\epsilon)_{ML} =& 2941.610693303918
\end{aligned}
$$

3(d)

```python
p = 22
L = LB_test(arr, p, 'ML', 14, 0.05)
Phi, sd = L[0], L[2] # Get the phi parameters and sample std of residual
X_Forecast = arr[-p:] # Slice out last p entries of X for forecasting

for t in range(30):
    Xi = np.dot(Phi, X_Forecast[-p:][::-1]) # Generate future values
    X_Forecast = np.append(X_Forecast, Xi) # Continue to update our data by appending forecasted values
X_Forecast += T_mean # Adding the mean back
```

```
Upper, Lower = [], []
# Getting values of upper and lower bounds, starting at x_731
for i in range(30):
    Upper.append(X_Forecast[p+i] + 1.96*sd*np.sqrt(i+1))
for i in range(30):
    Lower.append(X_Forecast[p+i] - 1.96*sd*np.sqrt(i+1))
plt.title('Observed and forecasted values \nfrom $X_{710}$ to $X_{760}$ with prediction intervals')
plt.plot(np.linspace(710, 760, 51), X_Forecast[(p-21):], label="Observed and forecasted values") # Plot X_{710} to X_{760}
plt.plot(np.linspace(731, 760, 30), Upper, label="Upper bound for prediction interval") # Plot upper prediction interval
plt.plot(np.linspace(731, 760, 30), Lower, label="Lower bound for prediction interval") # Plot lower prediction interval
plt.legend(), plt.xlabel('Day Number'), plt.ylabel('Daily energy Consumption (GWh)')
```

The code above generates the following figure.

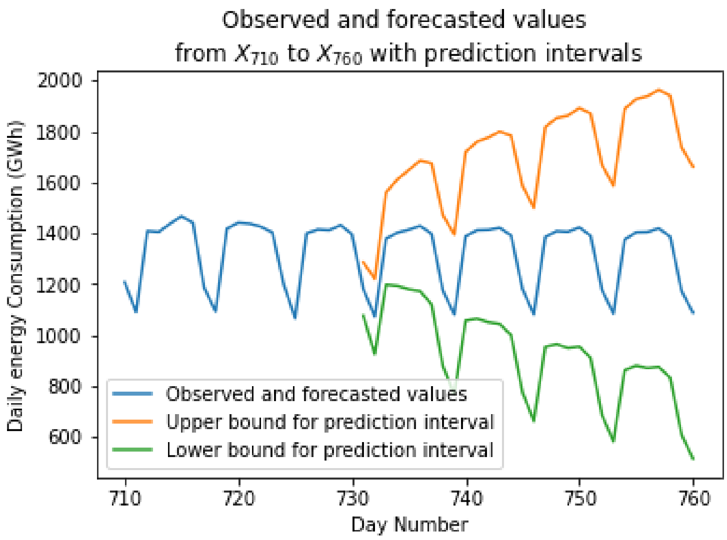

Figure 5: Plot of X_{710} to X_{760} with 95% prediction interval

We can see that, our forecast for X_{731} to X_{760} demonstrates largely similar behaviour to observed data, which is good news, as it indicates that our fitted model is fairly accurate. In particular, we can observe the same weekly periodic behaviour: in each 7-day window, there is a 5-day peak in energy consumption at around $1400GWh$, followed by a 2-day trough at around $1100 \sim 1200GWh$. We can interpret this as energy consumption during weekdays are typically higher than energy consumption during the weekends.

One thing to note about our predictions, is that the sample standard deviation for the residuals is relatively high (around 53), which means our prediction interval widens quite quickly the further we go into the future. At X_{760}, the range of the prediction interval is around 1150, which is massive, as its magnitude almost exceeds the predicted value of X_{760}. This vast decrease in certainty suggests that our model may not be best for large forecast steps, and the forecasted values may not be the best representation of what may occur.