1 Q1

1.1 a)

Consider the elements of matrix D.
The GLP of X_t is: $X_t = \frac{1-\theta}{1-\phi}\epsilon_t = (1-\theta)\sum_{k=0}^{\infty}(\phi\epsilon_t)^k = \epsilon_t + (\phi-\theta)\sum_{k=1}^{\infty}\phi^{n-1}\epsilon_{t-k}$, where $g_0 = 1$, and $g_k = (\phi-\theta)\phi^{k-1}$ for $k \geq 1$.
Let $t = 0$ and take the variance of X_t. As X_t and ϵ_t are both stationary processes, their acvs doesn't depend on τ but t.
Therefore, we have: $Var(\epsilon_0) = \sigma_\epsilon^2$, and $Var(X_0) = s_0 = Var(\epsilon_0 + (\phi - \theta)\sum_{k=1}^{\infty}\phi^{n-1}\epsilon_{-k}) = Var(\epsilon_0) + (\phi-\theta)^2\sum_{k=1}^{\infty}\phi^{2(n-1)}Var(\epsilon_{-k})) = \sigma_\epsilon^2(1+\frac{(\phi-\theta)^2}{1-\phi^2})$.
Now for $Cov(X_0, \epsilon_0)$: note that $E(\epsilon_t) = 0$ and that $E(\epsilon_0\epsilon_\tau) = 0$ for any $\tau \neq 0$, $Cov(X_0, \epsilon_0) = E(X_0\epsilon_0) - E(X_0)E(\epsilon_0) = E(X_0\epsilon_0) = E(\epsilon_0^2 + (\phi - \theta)\sum_{k=1}^{\infty}\phi^{n-1}E(\epsilon_{-k}\epsilon_0)) = E(\epsilon_0^2) = \sigma_\epsilon^2$.
Therefore, D $= \sigma_\epsilon^2 \begin{bmatrix} 1 + \frac{(\phi-\theta)^2}{1-\phi^2}) & 1 \\ 1 & 1 \end{bmatrix}$

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
from scipy.linalg import toeplitz
plt.style.use('classic')
```

```
def ARMA11(phi, theta, sigma2, N):
    samples = np.array([])
    D = sigma2 * np.array([[1 + (phi - theta)**2 / (1 - phi**2), 1]
                                , [1, 1]])
    C = np.linalg.cholesky(D)  # use cholesky to decompose
    y1 = np.random.normal(0, 1)
    y2 = np.random.normal(0, 1)
    x0, eps0 = np.dot(C,np.array([y1,y2])) # initial x0,eps0: C@y
    for i in range(N):
        eps1 = np.random.normal(0, np.sqrt(sigma2))
        x1 = phi*x0 + eps1 - theta*eps0
        samples = np.append(samples, x1)
        x0 = x1
        eps0 = eps1
    return samples
```

1.2 b)

Using the formula $\hat{s}_\tau^{(p)} = \frac{1}{N}\sum_{t=1}^{N-|\tau|}\left(X_t - \bar{X}\right)\left(X_{t+|\tau|} - \bar{X}\right)$, where $\tau = 0, \pm1, \ldots, \pm(N-1)$. The implementation is:

```
def acvs(X, tau):
    tau, N = abs(tau), len(X)
    mu = np.sum(X)/N
    return sum([(X[i] - mu)*(X[i+tau] - mu) for i in range(N - tau)
                                ]) / N
```

1.3 c)

Using the formula $\hat{S}^{(p)}(f) = \frac{1}{N}\left|\sum_{t=1}^{N} X_t e^{-i2\pi ft}\right|^2$, the implementation is:

```
def periodogram(X):
    N = len(X)
    shifted_sum = np.fft.fftshift(np.fft.fft(X, N))
    spec_est = np.abs(shifted_sum)**2 / N
    freqs = -1/2 + np.arange(0,N)/N
    return spec_est, freqs
```

2 Q2

2.1 A)B)C)

Defined a function "seq-generator" to generate the required $\hat{S}_j^{(p)}(f_{\frac{N}{4}})$ and $\hat{S}_j^{(p)}(f_{\frac{N}{4}+1})$:

```
phi, theta, sigma2 = -0.45, 0.90, 2.25
N = np.array([4, 8, 16, 32, 64, 128, 256, 512])  # N for length
N_t = 10000

def seq_generator(lth, N_t):
    S_1 = np.array([])
    S_2 = np.array([])
    for i in range(N_t):
        ts = ARMA11(phi, theta, sigma2, lth)
        spec_est, freqs = periodogram(ts)
        S_1 = np.append(S_1, spec_est[int(lth/4)])
        S_2 = np.append(S_2, spec_est[int(lth/4 + 1)])
    return S_1, S_2
```

Given $E(\hat{S}^{(p)}(f)) \approx S(f)$, we want to obtain $S(f_{\frac{N}{4}})$ for the large-sample result. And given the of GLP for $ARMA(1,1)$, and $f_{\frac{N}{4}} = \frac{\frac{N}{4}}{4} = \frac{1}{4}$, we have $S_X(\frac{1}{4}) = \sigma_\epsilon^2 \frac{|1-\theta e^{-i2\pi/4}|^2}{|1-\theta e^{-i2\pi/4}|^2}$, which is implemented by "sdf":

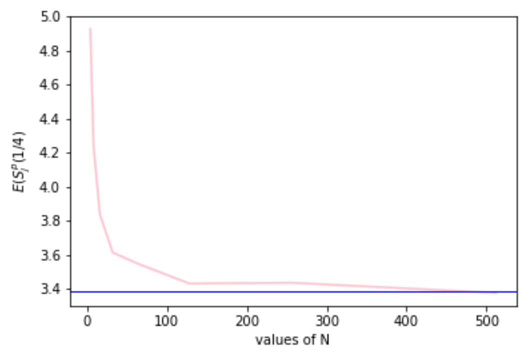
```
def sdf(f):
    return sigma2*np.abs(1 - theta*(np.exp(-2j*np.pi*f)))**2 / np.
                                        abs(1 - phi*(np.exp(-2j*np.pi
                                        *f)))**2
```

Now it's ready to do the plots.

2.2 a)

```
S1_mean = np.array([])  # to store the sample mean
for i in range(len(N)):
    seq1, seq2 = seq_generator(N[i], N_t)
    S1_mean = np.append(S1_mean, np.mean(seq1))

plt.plot(N, S1_mean, color='pink')
plt.xlabel("values of N")
```

```
plt.ylabel("$E(S_j^{p}(1/4)$")
plt.axhline(y = sdf(1/4), c="blue", linewidth=1)
plt.show()
```

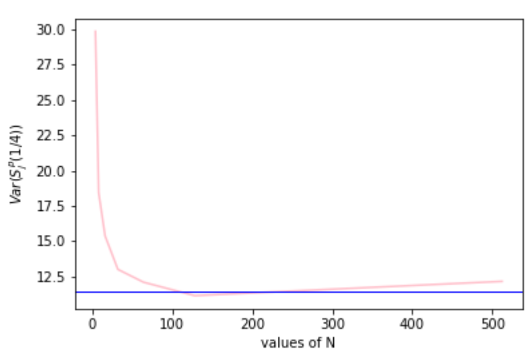


Comment: The estimator is asymptotically unbiased, according to lecture notes. So the sequence goes down quickly and approaches the line of large-sample result (blue line) at $S_X(\frac{1}{4})$.

2.3 b)

Given $Var(\hat{S}^{(p)}f) \approx S^2(f) \quad 0 < f < \frac{1}{2}$, again, we can obtain the large sample result by computing $sdf(\frac{1}{4})^2$.

```
S1_var = np.array([])
for i in range(len(N)):
    seq1, seq2 = seq_generator(N[i], N_t)
    S1_var = np.append(S1_var, np.var(seq1)*len(seq1)/(len(seq1) -
                                          1))

plt.plot(N, S1_var, color='pink')
plt.xlabel("values of N")
plt.ylabel("$Var(S_j^{p}(1/4)$")
plt.axhline(y = sdf(1/4)**2, c="blue", linewidth=1)
plt.show()
```

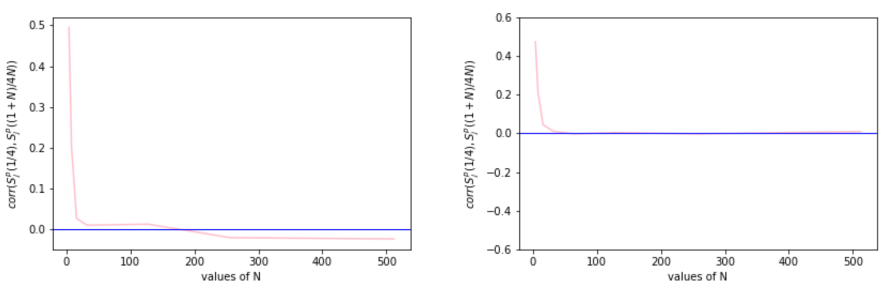


Comment: Similar to the trend of sample mean, the sample variance for increasing N goes down to the large-sample result, but it gets close more rapidly.

2.4 c)

Given $\text{corr}\left\{\hat{S}^{(p)}\left(f_j\right), \hat{S}^{(p)}\left(f_k\right)\right\} \approx 0, \quad j \neq k$ and $0 \leq j, k \leq N/2$, the horizontal line should be drawn at 0.

```
def P_cor(x, y):
    """Return the correlation of 2 sequences x and y."""
    res = 0
    for j in range(len(x)):
        res += (x[j] - np.mean(x)) * (y[j] - np.mean(y))
    res = res / np.sqrt(len(x)**2 * np.var(x) * np.var(y))
    return res

S_cor = np.array([])
for i in range(len(N)):
    seq1, seq2 = seq_generator(N[i], N_t)
    rho = P_cor(seq1, seq2)
    S_cor = np.append(S_cor, rho)

plt.plot(N, S_cor, color='pink')
plt.axhline(y=0, c="blue", linewidth=1)
plt.ylim(-0.5, 0.5)
plt.xlabel("values of N")
plt.ylabel("$corr(S_j^{p}(1/4), S_j^{p}((1+N)/4N))$")
plt.show()
```



Comment: The correlation coefficients is generally going down to 0, with a little fluctuation (with magnitude of about 0.05) around 0 for large N, which is seen from the left. Setting the y-scale to be a much broader one: (-0.5, 0.5), the fluctuation is negligible, which is seen from the right.

2.5 d)e)f)

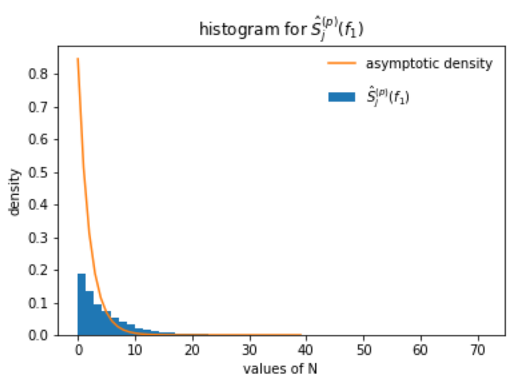
Given that as $N \longrightarrow 0$, $\hat{S}^{(p)}(f) \stackrel{\mathrm{d}}{=} \frac{S(f)}{2}\chi_2^2, \quad 0 < f < \frac{1}{2}$. Substituting f with $f_{\frac{N}{4}}$, we have the asymptotic probability density $\hat{S}^{(p)}(f_{\frac{N}{4}}) \stackrel{\mathrm{d}}{=} \frac{S(\frac{1}{4})}{2}\chi_2^2, \quad 0 < f < \frac{1}{2}$, implementing as:

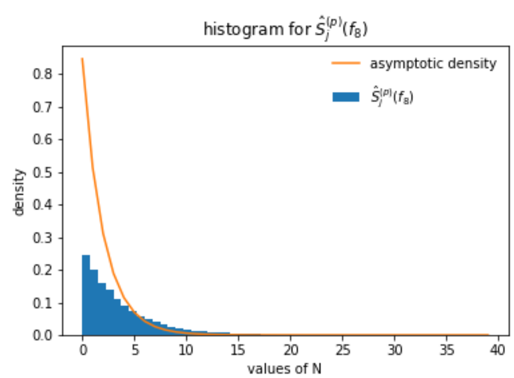
```python
from scipy.stats import chi2  # to compute chi-2 density
x = np.arange(0, 40, 1)
```

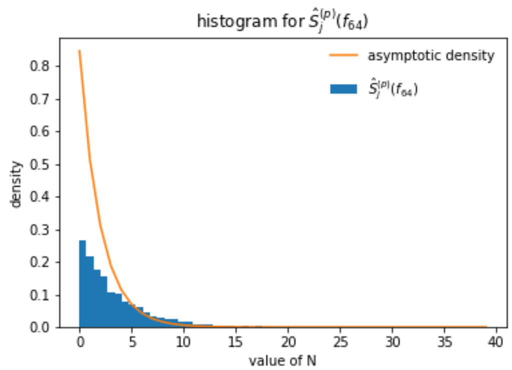
```python
# d)
seq1, seq2 = seq_generator(N[0], N_t)
plt.hist(seq1, density=True, bins=50, label="$\hat{S}_{j}^{(p)}(f_1)$")
plt.plot(x, sdf(1/4)/2*chi2.pdf(x, df=2), label="asymptotic density")
plt.legend(loc='upper right', frameon=False, labelspacing=1)
plt.xlabel("values of N")
plt.ylabel("density")
plt.title("histogram for $\hat{S}_{j}^{(p)}(f_1)$")
plt.show()
```

```python
# e)
seq1, seq2 = seq_generator(N[3], N_t)
plt.hist(seq1, density=True, bins=50, label="$\hat{S}_{j}^{(p)}(f_8)$")
plt.plot(x, sdf(1/4)/2*chi2.pdf(x, df=2), label="asymptotic density")
plt.legend(loc='upper right', frameon=False, labelspacing=1)
plt.xlabel("values of N")
plt.ylabel("density")
plt.title("histogram for $\hat{S}_{j}^{(p)}(f_8)$")
plt.show()
```

```python
# f)
seq1, seq2 = seq_generator(N[6], N_t)
plt.hist(seq1, density=True, bins=50, label="$\hat{S}_{j}^{(p)}(f_{64})$")
plt.plot(x, sdf(1/4)/2*chi2.pdf(x, df=2), label="asymptotic density")
plt.legend(loc='upper right', frameon=False, labelspacing=1)
plt.xlabel("values of N")
plt.ylabel("density")
plt.title("histogram for $\hat{S}_{j}^{(p)}(f_{64})$")
plt.show()
```







Comment: The plotted histogram approaches the ideal asymptotic probability density as N increases, by the change of the increased left tail density and the deceased right tail density.

3 Q3

3.1 a)

First read the file and store as a numpy array. And then center the time series by subtracting the sample mean from each observation.

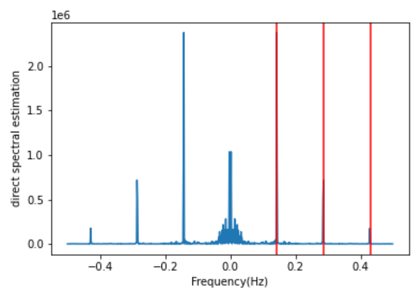
```
import csv
X = np.array([])
with open('time_series.csv','r') as f:
    reader = csv.reader(f)  # read the time series
    for row in reader:
        X = np.append(X, row)
X = np.array(row).astype(np.float64)  # store as a numeric array
N = len(X)
X = X - np.mean(X)  # centered time series
```

Implementing $0.5 * cosine$ taper, as given in lecture notes:

```
p = 0.5
h_coef = np.array([])
for i in range(1, N+1):
    if 1<= i <= int(p*N)/2:
        h_coef = np.append(h_coef, (1-np.cos(2*np.pi*i/(int(p*N)+1)
                                    ))/2)
    elif int(p*N)/2 < i < N + 1-int(p*N)/2:
        h_coef = np.append(h_coef, 1)
    else:
        h_coef = np.append(h_coef, (1-np.cos(2*np.pi*(N+1-i)/(int(p
                                    *N)+1)))/2)

C = np.sum(h_coef**2)  # the normalizing constant
h_coef = h_coef / np.sqrt(C)

plt.plot(h_coef)
plt.show()
```

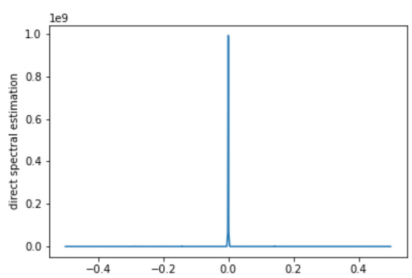
Plug the tapered series $\{h_t X_t\}$ back into "periodogram" defined in Q1:

```
estimate, freqs = periodogram(h_coef*X)
plt.plot(freqs, (N*estimate))
plt.xlabel("Frequency(Hz)")
plt.ylabel("direct spectral estimation")

# trials to match the peaks
plt.axvline(1/7, color="red")  # period = 1/f = 7
plt.axvline(1/3.5, color="red")  # period = 3.5
plt.axvline(1/2.3255, color="red")  # period = 2.3255
plt.show()  # see below
```



As a periodogram is symmetric, only meed to focus on the positive half: 3 red lines plotted represent period 7, 3.5, and 2.3255 days respectively(see code "plt.axvline" above). This means that 3 values appear periodically in this time series. And the height of the peaks is proportional to the magnitude of the observations, which means that a relatively small value appears around every 2.3 days, a larger value appears every half a week and a large value appears once a week. The peak at $f = 0$ represents $\hat{S}^{(d)}(0) = \frac{1}{N}\sum_{t=1}^{t=N} X_t^2$(a pulse at 0), which is irrelevant to any period.

Centering is an essential step in spectral analysis. If not, then $\hat{S}^{(d)}(0) = \frac{1}{N}\sum_{t=1}^{t=N} X_t^2$. Give the time series (which has a big sample mean about 1294), $\hat{S}^{(d)}(0)$ will be so big that it dominates the estimated spectrum, resulting into low-frequency and low-amplitude components obscured by leakage. This is exactly shown by the graph plotted without centering.



Therefore, centering will lead to a better estimate at neighboring frequencies as the last plot on page 8 shows.

3.2 b)

For X_1, ..., X_N, the estimation of ϕ and that of $\hat{\sigma}^2$ using ML is:

$$\hat{\phi} = (F^T F)^{-1} F^T X$$
$$\hat{\sigma}^2 = \frac{(X - F\hat{\phi})^T (X - F\hat{\phi})}{(N - 2p}$$

where $F = \begin{bmatrix} X_p & X_{p-1} & \dots & X_1 \\ X_{p+1} & X_p & \dots & X_2 \\ \vdots & & & \vdots \\ X_{N-1} & X_{N-2} & \dots & X_{N-p} \end{bmatrix}$ and $\boldsymbol{X} = \begin{bmatrix} X_{p+1} \\ X_{p+2} \\ \vdots \\ X_N \end{bmatrix}$.

```
def fill_F(X, N, p):
    """Return the matrix F in the system above."""
    F = np.zeros((N-p, p))
    for i in range(p):
        for j in range(N-p):
            F[j, i] = X[p-i+j-1]
    return F

def ML_est(X, p):
    """Return the estimated coefficients of AR(p)."""
    F = fill_F(X, len(X), p)
    phi_hat = np.linalg.inv(F.T @ F) @ F.T @ X[p: ]
    return phi_hat
```

And the estimation using tapered Yule-Walker is:

$$\hat{\phi}_p = \hat{\Gamma}_p^{-1} \hat{\gamma}_p$$

where $\hat{\Gamma}_p = \begin{bmatrix} \hat{s}_0 & \hat{s}_1 & \dots & \hat{s}_{p-1} \\ \hat{s}_1 & \hat{s}_0 & \dots & \hat{s}_{p-2} \\ \vdots & \vdots & & \vdots \\ s_{p-1} & \hat{s}_{p-2} & \dots & \hat{s}_0 \end{bmatrix}$, $\hat{\phi}_p = \left[\hat{\phi}_{1,p}, \hat{\phi}_{2,p}, \dots, \hat{\phi}_{p,p}\right]^T$, $\hat{\gamma}_p = [\hat{s}_1, \hat{s}_2, \dots, \hat{s}_p]^T$, and $\hat{s}_\tau = \sum_{t=1}^{N-|\tau|} h_t X_t h_{t+|\tau|} X_{t+|\tau|}$.

```
def tapr_acvs(X, tau):
    """Return the estimated autocovariance of X at lag tau."""
    tau = abs(tau)
    s_tau = np.sum([h_coef[i]*h_coef[i+tau]*X[i]*X[i+tau] for i in
                                                range(N-tau)])
    return s_tau

def fill_Gamma(X, p):
```

```
    """Return the matrix Gamma_p in the system above."""
    v = [tapr_acvs(X, i) for i in range(p)]
    Gamma = toeplitz(v, v)  # using Toeplitz
    return Gamma

def YW_est(X, p):
    """Return the coefficients of AR(p) using Yule-Walker."""
    gamma = [tapr_acvs(X, i) for i in range(1, p+1)]  # little
                                                     gamma_p
    Gamma = fill_Gamma(X, p)
    phi_hat = np.linalg.inv(Gamma)@gamma
    return  phi_hat
```

3.3 c)

For the test, h, α, q, the initial p, $len(X)$ are given. Use the statistic $L = n(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k}$ to obtain $\chi^2_{1-\alpha,h}$, and $\hat{\rho}_k$ (where $\hat{\rho}_k = \hat{s}_k^{(p)}/\hat{s}_0^{(p)}$ and $\hat{s}_k^{(p)} = \frac{1}{N}\sum_{t=1}^{N-|k|} X_t X_{t+|k|}$):

```
h, P, N = 14, 1, len(X)
q = scipy.stats.chi2.ppf(0.95, df=14)  # the 95th quantile of chi2
L = q + 1

def LB_test(X, h, q, p, N, L, method):
    """Return the smallest p and corresponding coefficients where
                                     H0 is not rejected."""
    while L > q:
        F = fill_F(X, len(X), p)
        if method == 0:
            phi_hat = ML_est(X, p)
        if method == 1:
            phi_hat = YW_est(X, p)
        res = X[p: ] - F @ phi_hat  # residual sequence
        n = len(res)
        L = n*(n+2)*np.sum([(acvs(res, i)/acvs(res, 0))**2/(n-i)
                                            for i in range(1, h+1)])
        p += 1
    return p-1, phi_hat
```

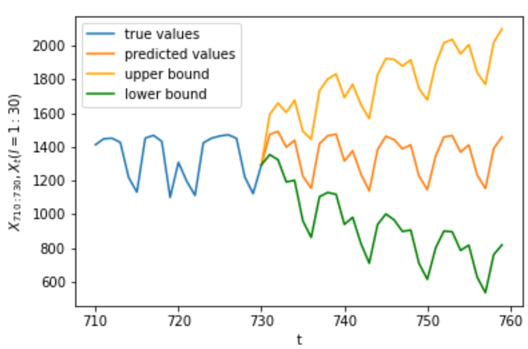
Report the smallest p and the corresponding coefficients:
For ML method:
$p = 22$, ϕ =[0.6945035, -0.0237297, 0.07616075, -0.07340594, 0.09066148, 0.10641031, 0.30209049, -0.22002768, -0.07446729, 0.06544186, 0.03335461, -0.14452815, -0.02592977, 0.24586605, -0.19601457, 0.04645478, -0.11404946, 0.01332519, 0.0473315, -0.07448256, 0.36634425, -0.22213929]
For YW method:
$p = 22$, ϕ =[0.71313326, -0.04742479, 0.07202885, -0.05510656, 0.06469044, 0.13499285, 0.32167699, -0.22531031, -0.1014689, 0.03248016, 0.05819976, -0.14454356, -0.07715938, 0.26217929, -0.20926383, 0.08641056, -0.08079289, -0.03553863, 0.06682065, -0.05594868, 0.31712052, -0.21327921]

3.4 d)

For general $AR(p)$, we use the following prediction procedures given on notes:

$$
\begin{aligned}
X_t(1) &= \phi_{1,p}X_t + \ldots + \phi_{p,p}X_{t-p+1} \\
X_t(2) &= \phi_{1,p}X_t(1) + \phi_{2,p}X_t + \cdots + \phi_{p,p}X_{t-p+2} \\
X_t(3) &= \phi_{1,p}X_t(2) + \phi_{2,p}X_t(1) + \cdots + \phi_{p,p}X_{t-p+3} \\
&\vdots
\end{aligned}
$$

```
# reported parameters using ML method
lp, phi = LB_test(X, h, q, p, N, L, method=0)
X = np.array(row).astype(np.float64)

# predictions
for i in range(30):
    xp = np.dot(phi, X[-lp:][::-1])
    X = np.append(X, xp)

# residuals and sigma_e
F = fill_F(X, len(X), lp)
res = X[lp: ] - F @ phi
sigma_e = np.std(res)

# the upper and lower bound
upp_bd = [X[730+i] + 1.96*sigma_e*np.sqrt(i) for i in range(30)]
low_bd = [X[730+i] - 1.96*sigma_e*np.sqrt(i) for i in range(30)]

plt.plot(range(710,731), X[710:731], label="true values")
plt.plot(range(730,760), X[730:761], label="predicted values")
plt.plot(range(730,760), upp_bd, color="orange", label="upper bound
                                 ")
plt.plot(range(730,760), low_bd, color="green", label="lower bound"
                                 )
plt.legend()
plt.xlabel("t")
plt.ylabel("$X_{710:730}, X_t(l=1:30)$")
plt.show()  # see below
```



Due to the construction of the interval, there's a probability of 0.95 that predicted trajectory is completely within the interval, and the boundaries of the interval exhibit the same patterns as the predicted values regarding a constant increasing/decreasing trend.

On top of that, the predicted values also oscillate periodically in the similar patterns as the observed time series, which indicates a high realization probability.