Time Series Coursework

tmp120 01854740

December 2022

1 Question 1

1.1 A

$G(z) = \frac{1-\theta z}{1-\phi z} = 1 + (\phi - \theta)\sum_{k=1}^{\infty} \phi^{k-1} z^k$
$X_t = \epsilon_t + (\phi - \theta)\sum_{k=1}^{\infty} \phi^{k-1}\epsilon_{t-k}$
$g_0 = 1, g_k = (\phi - \theta)\sum_{k=1}^{\infty} \phi^{k-1}, k >= 1$
$Var(X_0) = \sigma_\epsilon^2 \sum_{k=0}^{\infty} g_k^2 = \sigma_\epsilon^2(1 + (\phi - \theta)^2 \sum_{k=0}^{\infty} \phi^{2k}) = \sigma_\epsilon^2(1 + \frac{(\phi-\theta)^2}{1-\phi^2})$
$Var(\epsilon_0) = \sigma_\epsilon^2$
$Cov(X_0, \epsilon_0) = Cov(\epsilon_0, X_0) = E[\epsilon_0^2 + \epsilon_0(\phi - \theta)\sum_{k=1}^{\infty} \phi^{k-1}\epsilon_{-k}] = E[\epsilon_0^2] + (\phi - \theta)\sum_{k=1}^{\infty} \phi^{k-1} E[\epsilon_0 \epsilon_{-k}] = \sigma_\epsilon^2 + 0$

```
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
import scipy.signal as signal

def ARMA11(phi,theta,sigma2,N):
    X = np.zeros(N)
    #define D matrix
    D =  np.zeros((2,2))
    D[0][0] = sigma2*(1+(phi-theta)**2/(1-phi**2))
    D[1][1] = sigma2
    D[0][1] = sigma2
    D[1][0] = sigma2
    #get Cholesky decomposition and standard Gaussians
    C = np.linalg.cholesky(D)
    Y1 = np.random.normal(0,1)
    Y2 = np.random.normal(0,1)
    Y = np.array([Y1,Y2])
    #define X0, epsilon0 that satisfy stationarity
    X0, eps0 = np.matmul(C,Y)[0], np.matmul(C,Y)[1]
    eps_tminus1 = eps0
    X_tminus1 = X0
    #use the ARMA formula recursively for the remaining Xi
    for t in range(1,N+1):
        eps_t = np.random.normal(0,np.sqrt(sigma2))
        X_t = phi*X_tminus1 +eps_t - theta*eps_tminus1
        X[t-1] = X_t
```

```
        X_tminus1, eps_tminus1 = X_t, eps_t
    return X
```

1.2 B

```
def acvs(X,tau):
    N=len(X)
    X_bar = 1/N * np.sum(X)
    output = np.array([])
    #for each value in the tau list
    for t_ in tau:
        #find each autocovariance
        output = np.append(output, 1/N *sum([(X[t-1]-X_bar)*(X[t-1+
                                            abs(t_)]-X_bar) for t in
                                            range(1,N-abs(t_)+1)]))
    return output
```

1.3 C

```
def periodogram(X):
    #find the periodogram and shift to centre
    periodogram = (1/len(X))*np.abs(np.fft.fftshift(np.fft.fft(X)))
                                        **2
    #find the frequencies the periodogram is worked out at
    frequencies = np.fft.fftshift(np.fft.fftfreq(len(X)))
    return periodogram,frequencies
```

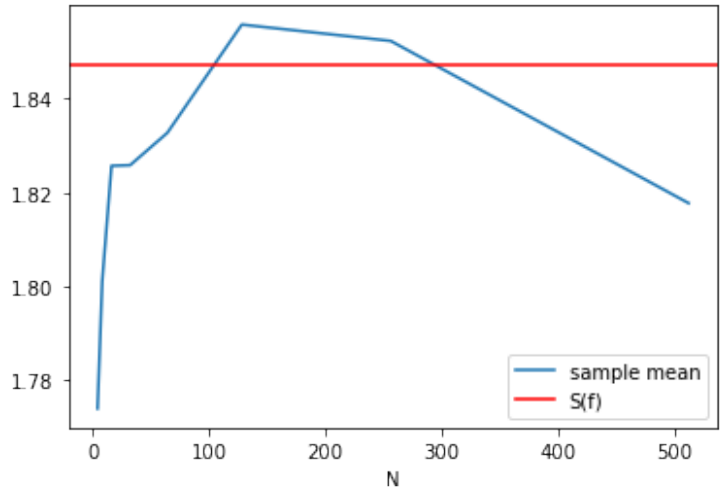
2 Question 2

2.1 A

```
#set my parameters to ARMA(1,1)
phi = 0.49
theta = 0.63
sigma_epsilon2 = 1.64

#set Ns to iterate over
n_values = [4,8,16,32,64,128,256,512]
N_r=10000

sample_means=np.array([])
sample_var=np.array([])
sample_p=np.array([])
sample_arrays = []
for N in n_values:
    sequence1=np.array([])
    sequence2=np.array([])
    #get N_r samples
    for i in range(N_r):
        X = ARMA11(phi,theta,sigma_epsilon2,N)
```

```
        period = periodogram(X)[0]
        #store sequences of periodograms at 2 different indices
        sequence1 = np.append(sequence1,period[N//4])
        sequence2 = np.append(sequence2,period[N//4-1])
    #keep the sample mean of sequence1, variance, correlation with
                                    2
    sample_means = np.append(sample_means,np.mean(sequence1))
    sample_var = np.append(sample_var,np.var(sequence1,ddof=1))
    sample_p = np.append(sample_p,np.corrcoef(sequence1,sequence2)[
                                    0][1])
    sample_arrays.append(sequence1)

#define the spectral density function which is used for large
                                    sample result
def S(f):
    return sigma_epsilon2 * np.abs((1-theta*np.exp(-1j*2*np.pi*f)))
                                    **2/(np.abs((1-phi*np.exp(-1j
                                    *2*np.pi*f)))**2)

#plot the sample mean with a horizontal line of S(1/4)
plt.plot(n_values,sample_means)
plt.axhline(Sf,color="r")
plt.xlabel("N")
plt.legend(["sample mean","S(f)"])
```

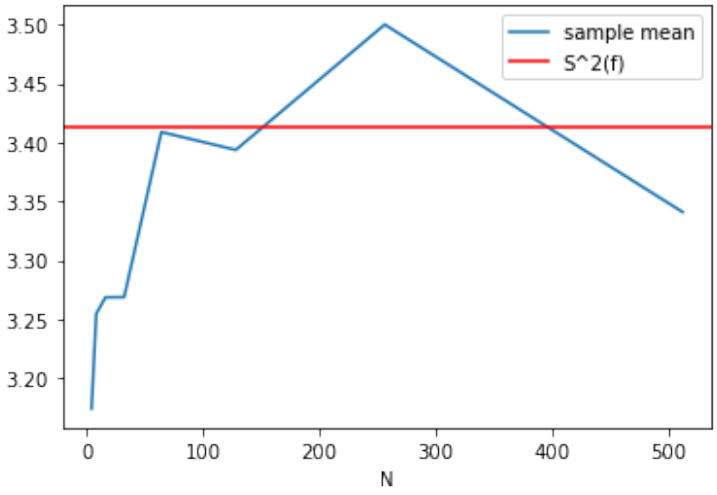
As N increases the sample mean more or less approaches the large sample result S(f).



2.2 B

```
plt.plot(n_values,sample_var)
plt.axhline(Sf**2,color="r")
plt.xlabel("N")
plt.legend(["sample variance","S^2(f)"])
```

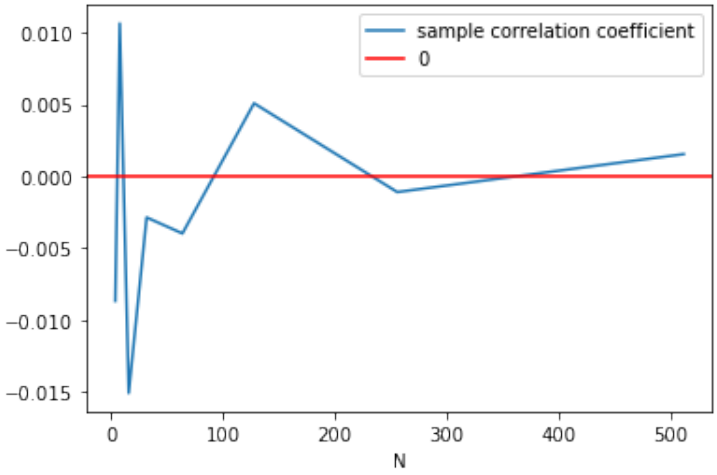
As N increases the sample variance more or less approaches the large sample result S(f) squared.



2.3 C

```
plt.plot(n_values,sample_p)
plt.axhline(0,color="r")
plt.xlabel("N")
plt.legend(["sample correlation coefficient",0])
```

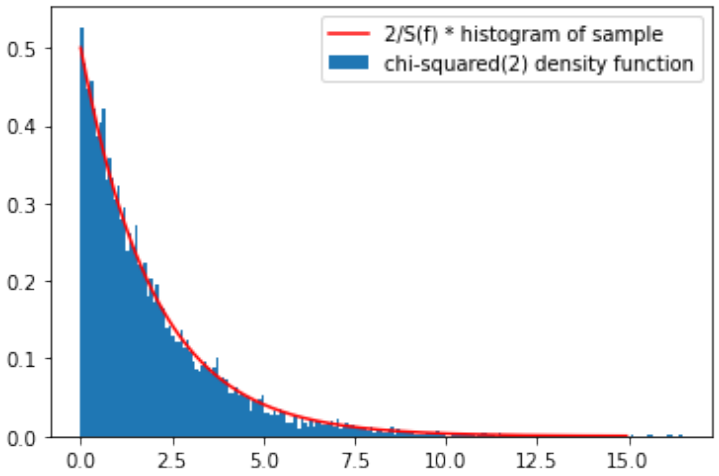
The sample correlation between the two sequences tends to 0 as N increases.



2.4 D

```
#extract the N=4 stored samples
#scale the histogram by 2/S(f) to compare to asymptotic result
plt.hist((2/Sf)*sample_arrays[0],bins=200, density =True )
x=np.arange(0,15,0.05)
from scipy.stats import chi2
plt.plot(x,chi2.pdf(x,df=2),color="r")
plt.legend(["2/S(f) * histogram of sample","chi-squared(2) density
                                    function"])
```

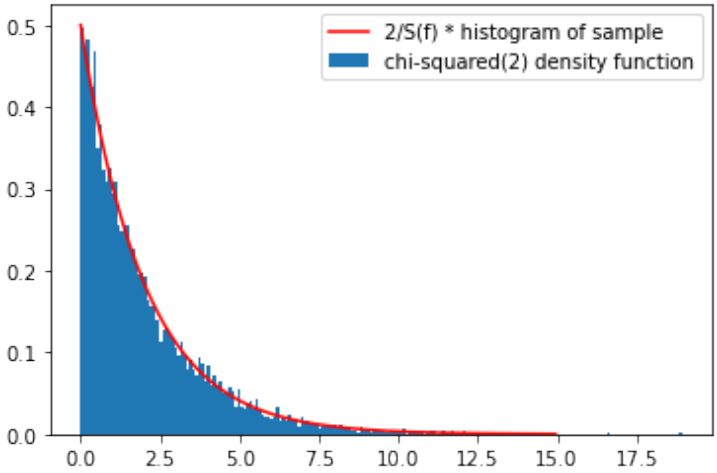
We find the sample follows the asymptotic distribution expected, has a chi-squared(2) shape scaled by half the spectral density function.



2.5 E

```
# index to get N=32 samples
plt.hist((2/Sf)*sample_arrays[3],bins=200, density =True )
x=np.arange(0,15,0.05)
from scipy.stats import chi2
plt.plot(x,chi2.pdf(x,df=2),color="r")
plt.legend(["2/S(f) * histogram of sample","chi-squared(2) density
                                    function"])
```

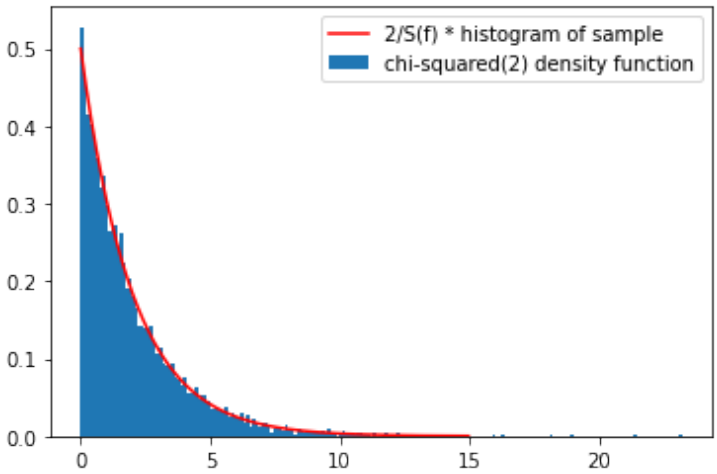
Again, there is a very good fit to the asymptotic distribution, even slightly better for larger N.



2.6 F

```
#index to extract the samples for N=256
plt.hist((2/Sf)*sample_arrays[6],bins=200, density =True )
x=np.arange(0,15,0.05)
from scipy.stats import chi2
plt.plot(x,chi2.pdf(x,df=2),color="r")
plt.legend(["2/S(f) * histogram of sample","chi-squared(2) density
                                    function"])
```

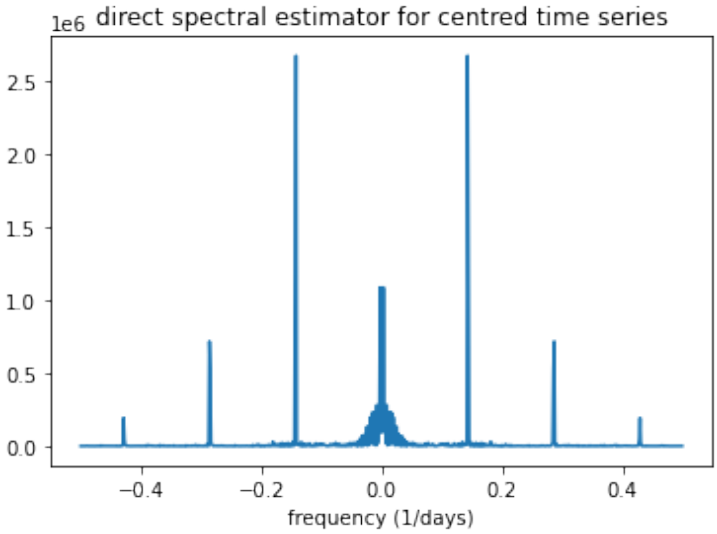
The fit of the samples scaled to the chi-squared(2) is very close, even better for N=256.

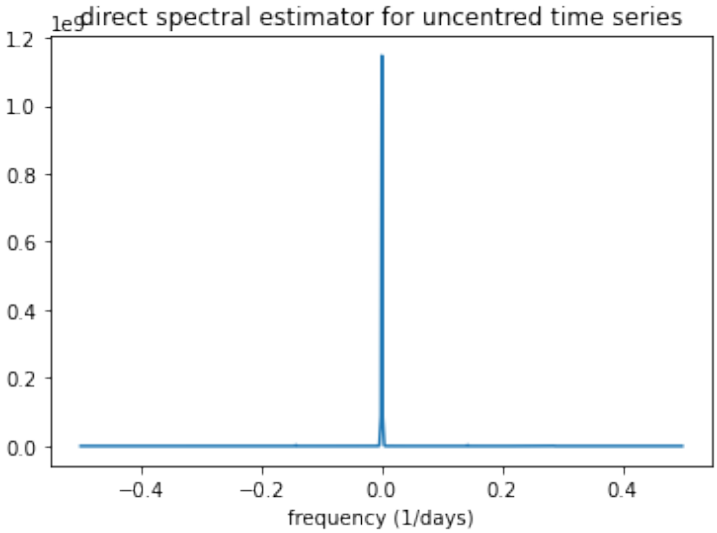


3 Question 3

3.1 A

```
#extract the time series from csv and centre it
import csv
import pandas as pd
df  = pd.read_csv("time_series_120.csv")
df.rename(columns=lambda x: x[:6],inplace = True)
X = df.columns.astype(np.float).to_numpy()
X_centred = X-X.mean()

import math as math
def cosine_taper(X):
    #function that applies a 50% cosine taper to the time series
    N = len(X)
    taper =  np.zeros(len(X))
    for t in range(len(X)):
        if t<=math.floor(0.5*N)/2-1:
            taper[t]= 0.5*(1-np.cos(2*np.pi*(t+1)/(math.floor(0.5*N
                                                    )+1)))
        elif t>math.floor(0.5*N)/2 -1 and t<N+1-math.floor(0.5*N)/2
                                                -1:
            taper[t]= 1
        else:
            taper[t] = 0.5*(1-np.cos(2*np.pi*(N+1-(t+1))/(math.
                                                    floor(0.5*N)+1)))
    #scale the taper so it has square sum 1
    taper = taper/np.sqrt(np.sum(taper**2))
    #apply the taper to the time series
    X1 = np.multiply(X,taper)
    return X1

#taper the centred data and get the periodogram
new_data = cosine_taper(X_centred)
periodo, freq = periodogram(new_data)
#direct spectral estimate is N*periodogram
plt.plot(freq,len(X_centred)*periodo)
plt.xlabel("frequency (1/days)")
plt.title("direct spectral estimator for centred time series")

#non centred
new_data = cosine_taper(X)
periodo, freq = periodogram(new_data)
plt.plot(freq,len(X)*periodo)
plt.xlabel("frequency (1/days)")
plt.title("direct spectral estimator for uncentred time series")
```





For the centred time series, the largest peaks are at around 1/7 and -1/7 which correspond most likely to weekly trends, stronger correlation between data 1 week apart. There seems to be another smaller peak between 1/3 and 1/4 and -1/3 and -1/4 corresponding perhaps to half-weekly patterns. There are tiny peaks near 0.4 and -0.4 corresponding to a more frequent pattern (around every 2 days). The second largest peak size is at 0, corresponding to very long-term patterns e.g. yearly trends.

If you do not centre the mean, there is only 1 very high peak at 0, perhaps as all values then have a small percentage difference to each other and so other peaks corresponding to trends which can no longer be seen disappear.

3.2 B

```
def max_likelihood(X,p):
    N = len(X)
    #define F matrix
    F = np.zeros((N-p,p))
    for i in range(0,p):
        for j in range(0,N-p):
            F[j,i]=X[p+j-i-1]
    X_ = X[p:]
    #calculate phis and sigma2 of the white noise
    phi_ = np.linalg.inv(((F.T)@F))@(F.T)@X_
    sigma2_ = ((X_-F@phi_).T)@(X_-F@phi_)/(N-2*p)
    return phi_,sigma2_

#redefine the autocovariance function for pre-tapered time series
def acvs2(X,tau):
    N=len(X)
    X_bar = 1/N * np.sum(X)
    output = np.array([])
    for t_ in tau:
        #do not divide by the length of X
        output = np.append(output, sum([(X[t-1]-X_bar)*(X[t-1+abs(
                                        t_)]-X_bar) for t in
                                        range(1,N-abs(t_)+1)]))
    return output

from scipy import linalg as lg
def yule_walker(X,p):
    #first pass time series through the taper
    X = cosine_taper(X)
    tau = np.arange(p+1)
    acvs_est = acvs2(X,tau)
    #use the new autocovariance function to fill the Toeplitz
                                    matrix and gamma vector
    toep = lg.toeplitz(acvs_est[:p])
    gamma = acvs_est[1:p+1]
    #calculate sigma2 of white noise and phis
    phi_ = lg.inv(toep)@gamma
    sigma2_ = acvs_est[0]
    for i in range(1,p+1):
        sigma2_-=phi_[i-1]*acvs_est[i]
    return phi_,sigma2_
```

3.3 C

```
#function to calculate a single residual
def residual1(X, p, t):
    #obtain the fitted phis for a given p
    phis = max_likelihood(X, p)[0]
    return X[t-1] - np.sum([phis[i] * X[t - i-2] for i in range(p)]
                                    )

#function for the test statistic given the sequence of residuals
def L(res):
```

```
    h = 14
    #define n as length of residual sequence
    n=len(res)
    tau = [i for i in range(1,h+1)]
    s0 = acvs(res,[0])
    L1 = n*(n+2)*sum([((acvs(res,[k])/s0)**2)/(n-k) for k in range(
                                        1,h+1)])
    return L1[0]

h=14
alpha = 0.05
#define term that if smaller than the test statistic we reject the
                                    null hypothesis
chi_term = chi2.ppf(1 - alpha, h)

#initialise p as 1
p = 1
#get the list of residuals
residuals = [residual1(X_centred, p, t) for t in range(p+1, len(
                                    X_centred)+1)]
while L(residuals) > chi_term:
    #increment p until the null hypothesis fails to be rejected
    p+=1
    residuals = [residual1(X_centred, p, t) for t in range(p+1, len
                                        (X_centred)+1)]
#return the optimal p and the associated parameters
print(p)
print(max_likelihood(X_centred, p))
```

For the maximum likelihood method, I obtain p=22 and:
phi= [0.66523412, -0.03374059, 0.07177895, -0.12253324, 0.15702461, 0.07975202, 0.36443913, -0.3065897 , 0.01455166, 0.04429335, 0.05859316, -0.16736826, 0.04276669, 0.19270168, -0.14888418, -0.03588637, -0.09121873, 0.03128844, -0.00514974, -0.10253231, 0.34292401, -0.15183074]
sigma2 = 3980.4297264184784 as parameters for AR(p).

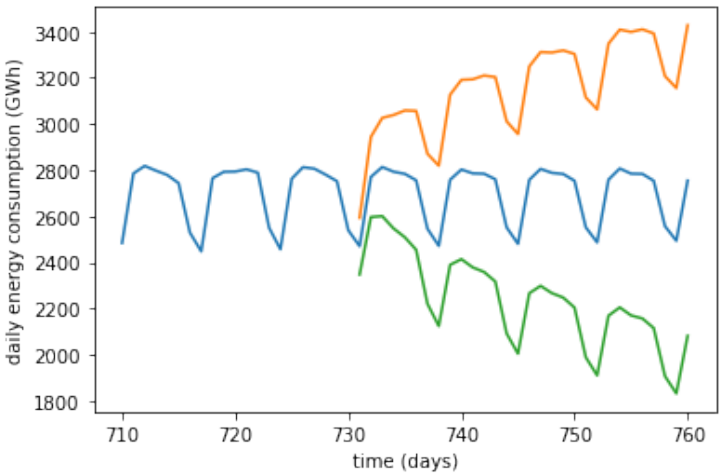
```
def residual(X, p, t):
    #obtain the fitted phis for yule walker
    phis = yule_walker(X, p)[0]
    return X[t-1] - np.sum([phis[i] * X[t - i-2] for i in range(p)]
                                        )

def L(res):
    h = 14
    n=len(res)
    tau = [i for i in range(1,h+1)]
    s0 = acvs(res,[0])
    L1 = n*(n+2)*sum([((acvs(res,[k])/s0)**2)/(n-k) for k in range(
                                        1,h+1)])
    return L1[0]
h=14
alpha = 0.05
chi_term = chi2.ppf(1 - alpha, h)

p = 1
residuals = [residual(X_centred, p, t) for t in range(p+1, len(
```

```
                                    X_centred)+1)]
while L(residuals) > chi_term:
    p+=1
    print(p)
    residuals = [residual(X_centred, p, t) for t in range(p+1, len(
                                        X_centred)+1)]
print(p)
print(yule_walker(X_centred, p))
```

For tapered yule walker, we get p=28, and:
phi = [0.71896084, -0.06933608, 0.10243464, -0.15277905, 0.1924825 , 0.09307033, 0.32124375, -0.31394785, -0.02983087, 0.09433763, 0.02787297, -0.16537239, 0.0403368 , 0.17567644, -0.16147629, 0.00375555, -0.06424761, 0.01931207, 0.04840125, -0.09063827, 0.25496836, -0.19309067, 0.05247075, -0.11761049, 0.08234566, -0.08738359, -0.0368928 , 0.16870564]
sigma2 = 3711.239051882394

3.4 D

```
#find the appropriate parameters before forecasting
p=22
phi,sigma2=max_likelihood(X_centred,p)
#find residuals for maximum  likelihood with p=22
residuals = [residual1(X_centred, 22, t) for t in range(p+1, len(
                                    X_centred)+1)]
sample_sd = np.std(residuals,ddof=1)
lower = np.array([])
upper = np.array([])
for l in range(1,31):
    #forecast new values iteratively
    Xi = np.dot(phi,X_centred[-p:][::-1])
    #find the upper and lower bounds of confidence interval
    low = Xi-1.96*sample_sd*np.sqrt(l)
    high = Xi +1.96*sample_sd*np.sqrt(l)
    X_centred = np.append(X_centred,Xi)
    lower = np.append(lower,low)
    upper = np.append(upper,high)
#include the mean back in
X_centred+=mean
lower+=mean
upper+=mean
plt.plot([i for i in range(710,761)],X_centred[709:])
plt.plot([i for i in range(731,761)],upper)
plt.plot([i for i in range(731,761)],lower)
```



I would report the point estimates however it is more important to report the confidence interval around it and the probability that the real value will lie within the forecasted interval. Perhaps I could find a narrower, more confident interval and report this one too.