MATH60046 Time Series Analysis Coursework

CID: 01843211

December 15, 2022

1 Simulating an ARMA(1,1) process

We wish to sample $X_1 \dots X_N$ from an ARMA(1,1) process with parameters ϕ, θ and σ_ϵ^2. We will build a function in Python to do this, but first we must calculate the distribution of X_0 and ϵ_0 in order to satisfy the stationary conditions.

1.1 Sampling the initial variables X_0 and ϵ_0

We note that an ARMA(1,1) process can be written in GLP form as follows. Consider

$$X_t = \phi X_{t-1} + \epsilon_t - \theta_{t-1}\epsilon_{t-1}$$

Then we have

$$\begin{aligned}
X_t - \phi X_{t-1} &= \epsilon_t - \theta\epsilon_{t-1} \\
(1-\phi B)X_t &= (1-\theta B)\epsilon_t \\
X_t &= \frac{(1-\theta B)}{(1-\phi B)}\epsilon_t
\end{aligned}$$

where B is the backwards shift operator. We can rewrite this as

$$\begin{aligned}
X_t &= (1 + \phi B + \phi^2 B^2 \dots)(1-\theta B)\epsilon_t \\
&= (1 + (\phi-\theta)B + (\phi-\theta)\phi B^2 + (\phi-\theta)\phi^2 B^3 \dots)\epsilon_t \\
&= \epsilon_t + (\phi-\theta)\sum_{j=1}^{\infty}\phi^{j-1}\epsilon_{t-j}
\end{aligned}$$

and so

$$\begin{aligned}
\mathrm{Var}\{X_t\} &= \mathrm{Var}\{\epsilon_t + (\phi-\theta)\sum_{j=1}^{\infty}\phi^{j-1}\epsilon_{t-j}\} \\
&= \sigma_\epsilon^2 + (\phi-\theta)^2\sum_{j=1}^{\infty}(\phi^2)^{j-1}\mathrm{Var}\{\epsilon_{t-j}\} \\
&= \left(1 + \frac{(\phi-\theta)^2}{1-\phi^2}\right)\sigma_\epsilon^2
\end{aligned}$$

for all $t \in T$ (and in particular for $t = 0$) as a result of the 2nd order stationarity of the ARMA process. Now

$$\begin{aligned}
\mathrm{Cov}\{X_t, \epsilon_t\} &= \mathrm{Cov}\{\phi X_{t-1} + \epsilon_t - \theta_{t-1}\epsilon_{t-1}, \epsilon_t\} \\
&= \mathrm{Cov}\{\epsilon_t, \epsilon_t\} \\
&= \sigma_\epsilon^2
\end{aligned}$$

given that ϵ_t is independent of X_{t-1} and ϵ_{t-1} as a first-order stationary white noise process.

Now consider the vector $[X_0, \epsilon_0]$ and its variance-covariance matrix:

$$D = \begin{bmatrix} \text{Var}\{X_0\} & \text{Cov}\{X_0, \epsilon_0\} \\ \text{Cov}\{\epsilon_0, X_0\} & \text{Var}\{\epsilon_0\} \end{bmatrix}$$
$$= \sigma_\epsilon^2 \begin{bmatrix} 1 + \frac{(\phi-\theta)^2}{1-\phi^2} & 1 \\ 1 & 1 \end{bmatrix}$$

This is positive-semidefinite and hence has a Cholesky decomposition $D = CC^T$, where C is lower-triangular. In order to sample $[X_0, \epsilon_0]^T$ whilst satisfying stationarity, we consider $Y = [Y_1, Y_2]^T$ where $Y_1, Y_2 \sim \mathcal{N}(0, 1)$ independently. The variance-covariance matrix of CY is $CIC^T = CC^T = D$. So to generate a sample $[X_0, \epsilon_0]$ we sample $[Y_1, Y_2]$ and pre-multiply by C.

We have $\phi = -0.76, \theta = -1.92, \sigma_\epsilon^2 = 2.81$. We can now implement in Python an algorithm to simulate the first N values of the time series:

```
def ARMA11(phi, theta, sigma2, N):
    """Return realisation of ARMA11 process with given phi, theta, sigma^2 and N"""
    cov_x_eps = 1 + ((phi - theta) ** 2) / (1 - phi ** 2)
    D = sigma2 * np.array([[cov_x_eps, 1], [1, 1]], np.float64)
    C = np.linalg.cholesky(D)
    Y = np.array([[np.random.normal(0, 1)],[np.random.normal(0, 1)]])
    X0 = C.dot(Y)[0, 0]
    eps0 = C.dot(Y)[1, 0]
    X = X0
    eps = eps0
    series = []
    for i in range(N):
        eps_next = np.random.normal(0,(sigma2) ** (1 / 2))
        X = phi * X + eps_next - theta * eps
        eps = eps_next
        series.append(X)
    return series
```

1.2 Autocovariance estimator

We write a function to estimate the autocovariance of $\{X_t\}$ at lag τ:

```
def ACVS(series: list, tau):
    """Return estimate of ACV for time series at specified lag tau"""
    N = len(series)
    tau = abs(tau)
    x_bar = 0
    s_tau = 0
    for X in series:
        x_bar += X / N
    for t in range(N - abs(tau) - 1):
        s_tau += (series[t] - x_bar) * (series[t + tau] - x_bar) / (N - tau)
    return s_tau
```

1.3 Spectral estimation

We write a function to calculate the periodogram of a time series $\{X_t\}$ which aims to estimate the true spectrum:

$$S(f) = \sum_{\tau=-\infty}^{\infty} s_\tau e^{-i2\pi f\tau}$$

for $|f| \leq \frac{1}{2}$.

```python
def periodogram(X):
    """Return estimate of the periodogram."""
    fft = np.fft.fft(X)
    result = []
    for freq in fft:
        result.append(np.abs(freq) ** 2)
    return np.array(result) / len(X)
```

2 Large sample results for periodogram estimates

Our periodogram is defined

$$\hat{S}^{(p)}(f) = \frac{1}{N}\left|\sum_{t=1}^{N} X_t e^{-i2\pi ft}\right|^2$$

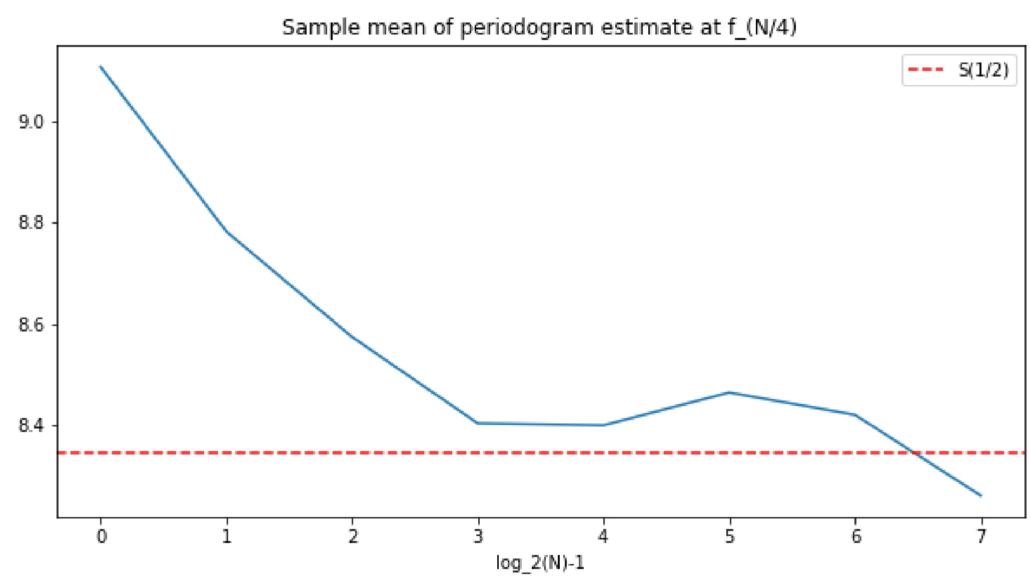
and we have that as $N \to \infty$

$$\begin{aligned} \mathrm{E}\{\hat{S}^{(p)}(f)\} &= S(f) \\ \mathrm{Var}\{\hat{S}^{(p)}(f)\} &= S^2(f) \\ \hat{S}^{(p)}(f) &\overset{d}{=} \frac{S(f)}{2}\chi_2^2 \end{aligned}$$

where $S(f)$ is the true spectral density function described above. We also have that, for Fourier frequencies $f_k = k/N$, the random variables $\hat{S}^{(p)}(f_0), \hat{S}^{(p)}(f_1) \dots \hat{S}^{(p)}(f_{\frac{N}{2}})$ are approximately pairwise uncorrelated for large enough N.

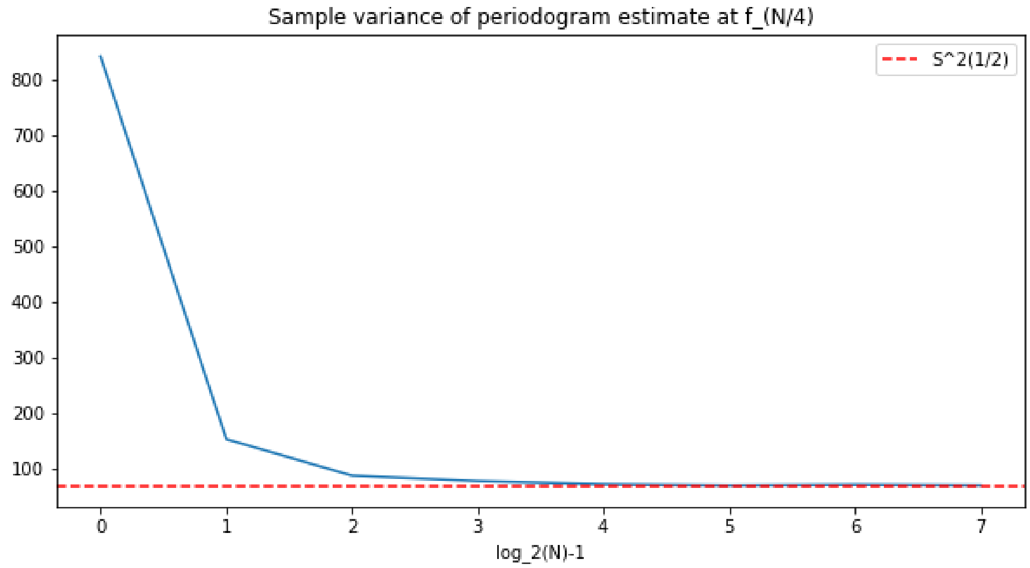
We simulate 10000 time series of length $N = 4, 8, \dots 512$ with our ARMA11 function defined previously.

2.1 Sample mean of $\hat{S}^{(p)}(f_{\frac{N}{4}})$



We can see that as $N \to \infty$ our periodogram tends towards the true value of $S(\frac{1}{2})$.

2.2 Sample variance of $\hat{S}^{(p)}(f_{N/4})$



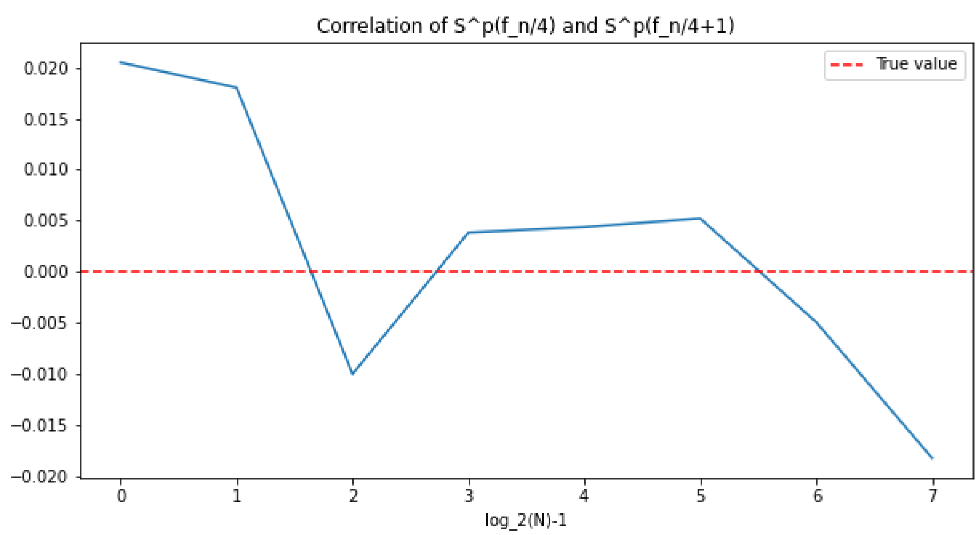
Again, we have that $\mathrm{Var}\{\hat{S}^{(p)}(f_{N\frac{N}{4}})\} \to S^2(\frac{1}{2})$. The variance for $N = 1$ is quite high, however this quickly settles to $S^2(\frac{1}{2})$ as N increases.

2.3 Sample correlation coefficient $\hat{\rho}$ between $\{S_j^{(p)}(f_{\frac{N}{4}})\}$ and $\{S_j^{(p)}(f_{\frac{N}{4}+1})\}$

The Pearson product-moment correlation coefficient is defined

$$\hat{\rho} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

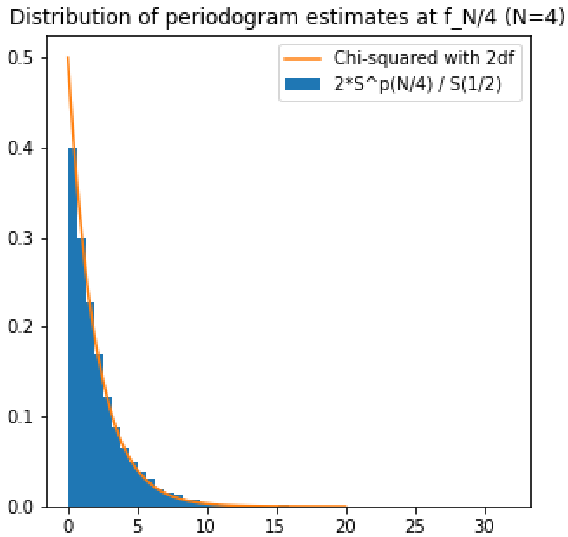
We plot $\hat{\rho}$ between $\hat{S}^{(p)}(f_{\frac{N}{4}})$ and $\hat{S}^{(p)}(f_{\frac{N}{4}+1})$:

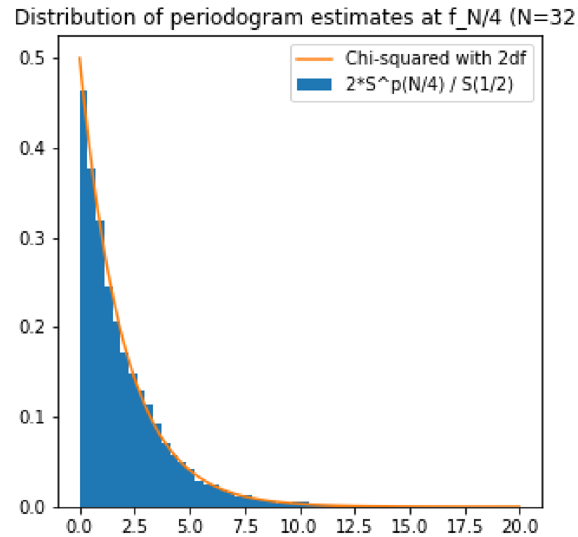


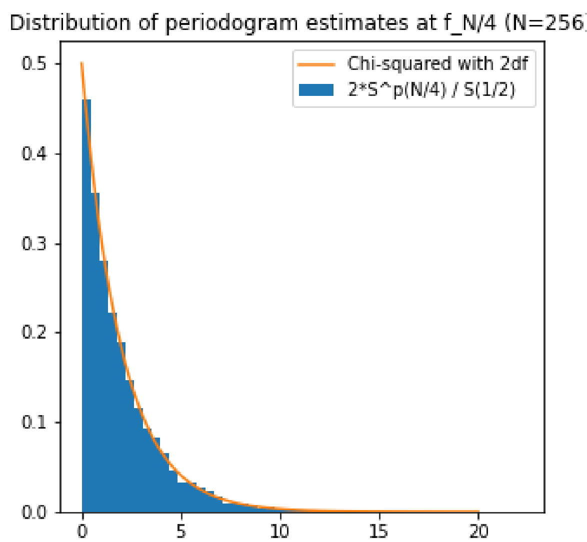
We see that the correlation is close to 0 for all values of N, however we may need larger N to see convergence to 0.

2.4 Distribution of spectral estimates

We plot a histogram of the periodogram at $f_{\frac{N}{4}}$ scaled by $\frac{2}{S(1/2)}$ for $N = 4, 32, 256$:







We see that in all cases, the estimates very closely approximate the true distribution.

2.5 Code for Question 2

```
Ns = [2 ** (i + 2) for i in range(8)]
Nr = 10000
N_total = sum(Ns) * Nr
per_n4 = []
per_n41 = []
count = 0
for N in Ns:
    #Initialise empty lists to store time series
    fn4 = []
```

```python
    fn4_1 = []
    for i in range(Nr):
        series = ARMA11(phi, theta, sigma2, N)
        freqs, per = periodogram(series)
        fn4.append(per[N // 4])
        fn4_1.append(per[N // 4 + 1])
    per_n4.append(fn4)
    per_n41.append(fn4_1)

def spectrum(phi, theta, sigma2, f):
    """Return spectrum of ARMA11 evaluated at f"""
    return sigma2 * (np.abs((1 - theta * np.exp(-2 * np.pi * np.complex(0, 1) * f)))
    ↪  ** 2) / (np.abs((1 - phi * np.exp(-2 * np.pi * np.complex(0, 1) * f)))**2)

spec = spectrum(phi,theta,sigma2, 1/4)

###############################################

#Q1A
fig1 = plt.figure(figsize=(10,5))
plt.plot([np.average(vals) for vals in per_n4])
plt.axhline(spec, color='red',linestyle='dashed', label="S(1/2)")
plt.title("Sample mean of periodogram estimate at f_(N/4)")
plt.xlabel("log_2(N)-1")
plt.legend()
plt.show()
#fig1.savefig("q1a")

###############################################

#Q1B
fig2 = plt.figure(figsize=(10,5))
plt.plot([np.var(vals) for vals in per_n41])
plt.axhline(spec**2, color='red',linestyle='dashed', label="S^2(1/2)")
plt.title("Sample variance of periodogram estimate at f_(N/4)")
plt.xlabel("log_2(N)-1")
plt.legend()
plt.show()
#fig2.savefig("q1b")

###############################################

#Q1C
def correlation(x, y):
    return np.corrcoef(x, y)[1,0]

corr = []

for i in range(8):
    corr.append(correlation(per_n4[i], per_n41[i]))

fig3 = plt.figure(figsize=(10,5))
plt.plot(corr)
plt.axhline(0, color='red',linestyle='dashed', label="True value")
plt.title("Correlation of S^p(f_n/4) and S^p(f_n/4+1)")
plt.xlabel("log_2(N)-1")
```

```
plt.legend()
plt.show()
#fig3.savefig("q1c")

###############################################

#Q1D
import math

def chisquaredpdf(x, k: int):
    return 1 / (2 ** (k / 2) * math.gamma(k / 2)) * x ** (k / 2 - 1) * np.exp(-x /
    ↪  2)

spec = spectrum(phi, theta, sigma2, 1 / 4)
xx = np.linspace(0,20,100)
fig4 = plt.figure(figsize=(5,5))
plt.hist([val / spec * 2 for val in per_n4[0]], density=True, bins=50,
↪  label="2*S^p(N/4) / S(1/2)")
plt.plot(xx, chisquaredpdf(xx, 2), label="Chi-squared with 2df")
plt.title("Distribution of periodogram estimates at f_N/4 (N=4)")
plt.legend()
#fig4.savefig("q1d")

###############################################

#Q1E
fig5 = plt.figure(figsize=(5,5))
plt.hist([val / spec * 2 for val in per_n4[3]], density=True, bins=50,
↪  label="2*S^p(N/4) / S(1/2)")
plt.plot(xx, chisquaredpdf(xx, 2), label="Chi-squared with 2df")
plt.title("Distribution of periodogram estimates at f_N/4 (N=32)")
plt.legend()
#fig5.savefig("q1e")

###############################################

#Q1F
fig6 = plt.figure(figsize=(5,5))
plt.hist([val / spec * 2 for val in per_n4[6]], density=True, bins=50,
↪  label="2*S^p(N/4) / S(1/2)")
plt.plot(xx, chisquaredpdf(xx, 2), label="Chi-squared with 2df")
plt.title("Distribution of periodogram estimates at f_N/4 (N=256)")
plt.legend()
#fig6.savefig("q1f")
```

3 Fitting an AR(p) model to a time series

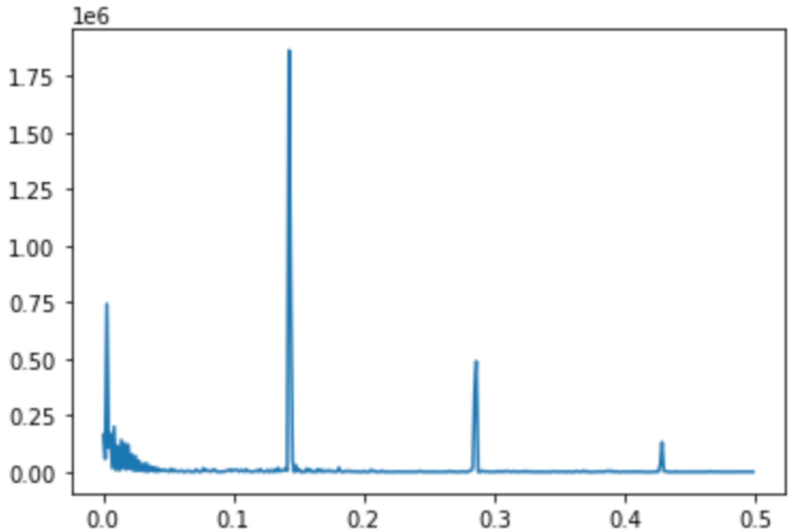
We define functions that fit an AR(p) model to a time series using approximate maximum likelihood and Yule-Walker with a 50% cosine taper.

```
import numpy as np

def fit_AR_max_likelihood(time_series, p):

    n_samples = len(time_series)
```

```
    # Set up least squares problem
    A = np.zeros((n_samples - p, p))
    b = np.zeros((n_samples - p, 1))

    for i in range(p):
        A[:, i] = time_series[i:(n_samples - p + i)]
        b[:, 0] = time_series[p:]

  # Solve least squares
    coefficients, residuals, _, _ = np.linalg.lstsq(A, b, rcond=None)

  # Calculate variance of the noise
    variance = sum(residuals) / (n_samples - p)

    return np.flip(coefficients), variance


def cosine_taper(N):
    n = np.arange(0, N)
    p = 0.5
    w = np.ones(N)
    for nn in n:
        if nn <= p*(N-1)/2:
            w[nn] = 0.5*(1+np.cos(np.pi*(2*nn/(p*(N-1))-1)))
        elif (p*(N-1)/2 < nn) and (nn <= (N-1)*(1-p/2)):
            w[nn] = 1
        elif (N-1)*(1-p/2) < nn:
            w[nn] = 0.5*(1+np.cos(np.pi*(2*nn/(p*(N-1))-2/p+1)))
    c = sum([ht ** 2 for ht in w])
    return w


import numpy as np
import scipy as sp
from scipy import linalg

def fit_AR_yule_walker(data, p):

    # Create autocorrelation matrix
    r = ACVS(data, range(p + 1))

    # Create toeplitz matrix
    toeplitz = sp.linalg.toeplitz(r[:-1])

    # Solve the Yule-Walker equations
    coeffs = linalg.solve(toeplitz, r[1:])

    # Calculate variance of the noise term
    sigma_sq = r[0] - np.sum(coeffs*r[1:])

    return coeffs, sigma_sq

h = cosine_taper(len(series))
mu = np.average(series)
N = len(series)
x_centered = [x - mu for x in series]
```

```
x_tapered = [x_centered[i] * h[i] for i in range(N)]
f, per = periodogram(x_tapered)
plt.plot(f[:730//2], per[:730//2])
```

We analyse the periodogram of the tapered series:



We see some noise around $f = 0$ and distinct peaks at $f = \frac{k}{7}$ for $k = 1, 2, 3$ corresponding to weekly peaks and troughs in the data. We now attempt to fit an AR(p) model to the data:

```
chisq = 23.685
ljungbox = 100000
phis = []
eps = 0
p = 0
while ljungbox > chisq:
    p += 1
    phis, eps = fit_AR_yule_walker(x_tapered, p)
    ljungbox = ljung_box_statistic(x_centered, yw)
print (p, phis, eps)
```

Using this code, we get $p = 23$ for Yule-Walker, with coefficients:

$\hat{\phi}_{YW}$ =[0.67695367, -0.04533059, 0.08914029, -0.13950514, 0.15968333, 0.08024158, 0.38725897, -0.31527041, 0.00433747, 0.04615454, 0.05292224, -0.1684849 , 0.0291921 , 0.19958316, -0.15048301, -0.02243638, -0.1073738 , 0.05051099, -0.00091582, -0.09655953, 0.30793795, -0.14846061, 0.00255843]

$\sigma_\epsilon^2 = 2808.29$

We now rerun the code for maximum likelihood estimation:

```
chisq = 23.685
ljungbox = 100000
phis = []
eps = 0
p = 0
while ljungbox > chisq:
    p += 1
    phis, eps = fit_AR_maximum_likelihood(x_centered, p)
    ljungbox = ljung_box_statistic(x_centered, yw)
print (p, phis, eps)
```

For maximum likelihood, we get $p = 22$ with coefficients

$\hat{\phi}_{ML}$ =[0.66576418, -0.03267086, 0.07242073, -0.12277021, 0.15658732, 0.07923709, 0.36371024, -0.30703022, 0.01266869, 0.044447 , 0.05692749, -0.16622188, 0.04178585, 0.19231146, -0.14766516, -0.035331 , -0.09074365, 0.03099957, -0.00422262, -0.10439299, 0.34356934, -0.15351423]

$\sigma_\epsilon^2 = 3855.87$