Time Series TSIA202a
Practical works

IQBI Hamza

19 October 2022

1 Exercise 1 : Computing averages and covariances

1.1 White Noise :

For $Z \sim WN(0, \sigma^2)$, its autocovariance function is :

$$\gamma_Z(t) = \sigma^2 \delta_0(t), \qquad (\forall t \in \mathbb{Z}).$$

1.2 AR(1) with a coefficient $\phi \in]-1, 1[$:

Let Y be a random process AR(1) s.t : $(t \in \mathbb{Z}), Y_t = Z_t + \phi Y_{t-1}$, with $\phi \in]-1, 1[$ and $Z \sim WN(0, \sigma^2)$.
We can get by induction that $Y = (\sum_{k=0}^{\infty} \phi^k Z_{t-k})_{k\in\mathbb{Z}}$
Let $t, s \in \mathbb{Z}$ s.t $s \leq t$.

$$\begin{aligned}
Cov(Y_t, Y_s) &= Cov(\sum_{k=0}^{\infty} \phi^k Z_{t-k}, \sum_{j=0}^{\infty} \phi^j Z_{t-j}) && (1)\\
&= \sum_{k,j=0}^{\infty} \phi^{k+j} \underbrace{(Cov(Z_{s-k}, Z_{t-j}))}_{=\gamma_Z((s-k)-(t-j))} && (2)\\
&= \sigma^2 \sum_{k,j=0}^{\infty} \phi^{k+j} \delta_{s-k,t-j} && (3)\\
&= \sigma^2 \sum_{k=0}^{\infty} \phi^{k+(t-s+k)} && (4)\\
&= \sigma^2 \frac{\phi^{t-s}}{1-\phi^2} && (5)
\end{aligned}$$

Then, by symmetry of $t\&s$, we get that : $(\forall t, s \in \mathbb{Z}), Cov(Y_t, Y_s) = \sigma^2 \frac{\phi^{|t-s|}}{1-\phi^2}$.
Hence

$$\gamma_Y(t) = \sigma^2 \frac{\phi^{|t|}}{1-\phi^2}, \qquad (\forall t \in \mathbb{Z}).$$

1.3 A Sinusoidal Process :

We consider a sinusoidal process $X = (A_0 \cos(\lambda_0 t + \Phi_0) + Z_t)_{t\in\mathbb{Z}}$.

$$\text{Where} \begin{cases} \lambda_0 \in [0, \pi[\\ \Phi_0 \text{ is a uniform random variable in } [0, 2\pi] \text{ independent of Z} \\ Z \sim WN(0, \sigma^2) \end{cases}$$

Let $s, t \in \mathbb{Z}$

$$\begin{aligned}
Cov(X_t, X_s) &= Cov(A_0 \cos(\lambda_0 t + \Phi_0), A_0 \cos(\lambda_0 s + \Phi_0)) + Cov(Z_t, Z_s) \qquad (\Phi_0 \perp\!\!\!\perp Z) && (6)\\
&= A_0^2 Cov(\cos(\lambda_0 t + \Phi_0), \cos(\lambda_0 s + \Phi_0)) + \sigma^2 \delta_{t-s} && (7)\\
&= \frac{A_0^2}{2} Cov(\cos(\lambda_0 (t-s)) \cos(\lambda_0 (t+s) + 2\Phi_0)) + \sigma^2 \delta_{t-s} && (8)\\
&= \frac{A_0^2}{2} \cos(\lambda_0 (t-s)) + \sigma^2 \delta_{t-s} && (9)
\end{aligned}$$

Hence, we have that

$$\gamma_X(t) = \frac{A_0^2}{2} \cos(\lambda_0 t) + \sigma^2 \delta_0(t), \qquad (\forall t \in \mathbb{Z}).$$

1.4 The results of plotting the estimations :

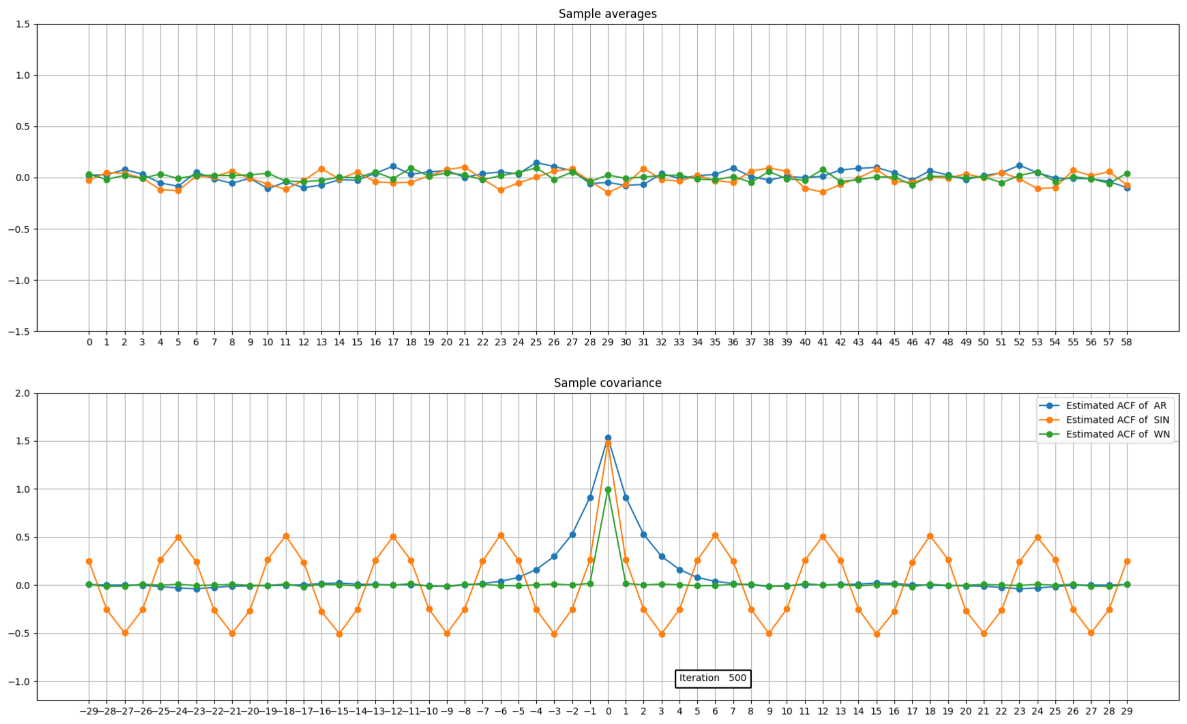


Figure 1 – Estimation of the average and covariance of the 3 specified weakly stationnary processes

The parameters taken are : $\sigma = 1, \phi = 0.6, A_0 = 1, \lambda_0 = \frac{\pi}{3}$.

So we have that, theoretically, $\gamma_Z(0) = 1, \gamma_Y(0) = \frac{\sigma^2}{1-\phi^2} \approx 1.5625, \gamma_X(0) = \frac{A_0^2}{2} + \sigma^2 = \frac{3}{2}$.

Both the shape of the estimated ACF and its value at zero permit us to conclude that the theoretical ACFs of these 3 processes are well estimated by this simulation.

We can see also that the average converges to zeros after 500 iterations, since the processes considered are centered.

1.5 Plotted properties of the transfer function of the specified AR process :

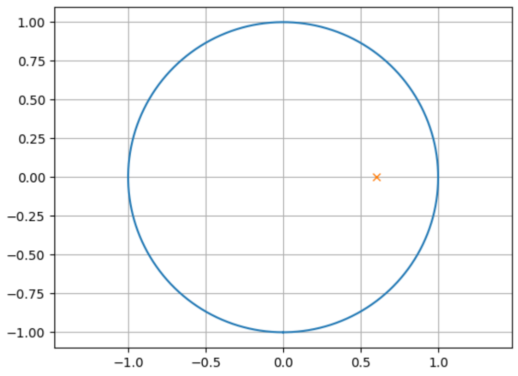


FIGURE 2 – Poles of the AR transfer function

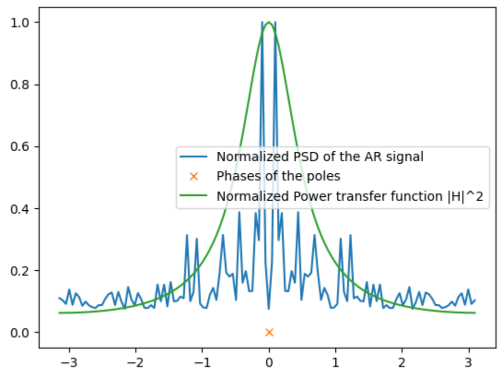


FIGURE 3 – PSD and phases of the poles

2 Exercise 2 : Spectral density estimation and Periodogram

2.1

We have that :

$$I_n(\lambda) = \frac{1}{2\pi}\sum_{h=-n+1}^{n-1} \widehat{\gamma}_n(h)e^{-i\lambda h} \tag{10}$$

$$= \frac{1}{2\pi n}\sum_{h=-n+1}^{n-1}\sum_{t=0}^{n-1-h}(X_t-\hat{\mu}_n)(X_{t+h}-\hat{\mu}_n)e^{-i\lambda h} \tag{11}$$

$$= \frac{1}{2\pi n}\sum_{t=0}^{n-1}\sum_{k=0}^{n-1}(X_t-\hat{\mu}_n)(X_k-\hat{\mu}_n)e^{-i\lambda(t-k)} \qquad (h=t-k) \tag{12}$$

$$= \frac{1}{2\pi n}\sum_{t=0}^{n-1}\sum_{k=0}^{n-1}((X_t-\hat{\mu}_n)e^{-i\lambda t})((X_k-\hat{\mu}_n)e^{-i\lambda k})^H \text{ (X is a real process)} \tag{13}$$

$$I_n(\lambda) = \frac{1}{2\pi n}\left|\sum_{t=0}^{n-1}(X_t-\hat{\mu}_n)e^{-i\lambda t}\right|^2 \tag{14}$$

$\rightsquigarrow$ *The periodogram* $\mathrm{I}_n(\lambda)$ is therefore proportionate to the DTFT of $X_0-\hat{\mu}_n, X_1-\hat{\mu}_n, ..., X_{n-1}-\hat{\mu}_n$.

2.2

By the previous question, we get that :

$$(\forall k \in 0,1,...,m-1),\ I_n(2\pi\frac{k}{m}) = \frac{1}{2\pi n}\left|\sum_{t=0}^{n-1}(X_t-\hat{\mu}_n)e^{-i2\pi\frac{k}{m}t}\right|^2.$$

Where $m \geq n$ is the order of the DFT used.

We can compute the DTFT of order m of $(X_0-\hat{\mu}_n, ..., X_{n-1}-\hat{\mu}_n, 0, ..., 0) \in \mathbb{R}^m$ as described in the script below :

```
    H = 30       # number of points for covariance
    n = 2*H -1   # number of process' samples
    tc = np.arange(-(H-1),H)  # temporal axis for ACF
    m = 10*n     # The size of the computed DTFT

    # Generate an AR(1) process
    Z = np.random.normal(0,1,n)
    phi1  = 0.6
    Pcoeffs = np.array([1.])
    Qcoeffs = np.poly((phi1,))
    AR = sig.lfilter(Pcoeffs, Qcoeffs,Z)
    mu_AR = np.mean(AR)
    I_AR = np.abs(np.fft.fft(AR - mu_AR, m))**2 / (2*np.pi*n)

    # Generate a Harmonic process
    A_0 = 1
    omega = np.pi/3
    phi =   np.pi +2*np.pi*np.random.random_sample()
    SIN = A_0 * np.cos(omega*tc+phi) +  np.random.normal(0,1,n)
    mu_SIN = np.mean(SIN)
    I_SIN = np.abs(np.fft.fft(WN - mu_SIN, m))**2 / (2*np.pi*n)

    # Generate a White Noise WN(0, 1)
    sigma = 1
    WN = np.random.normal(0,sigma,n)
    mu_WN = np.mean(WN)
    I_WN = np.abs(np.fft.fft(WN - mu_WN, m))**2 / (2*np.pi*n)
```
Listing 1 – Algorithm to compute I_n

Then, by plotting the results, we get :

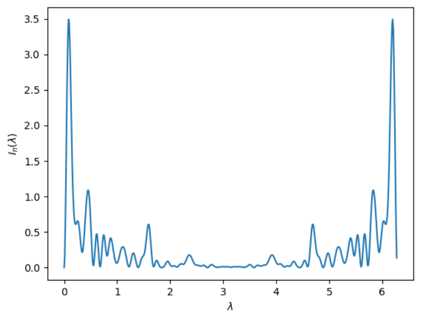


FIGURE 4 – Periodogram of AR(1)

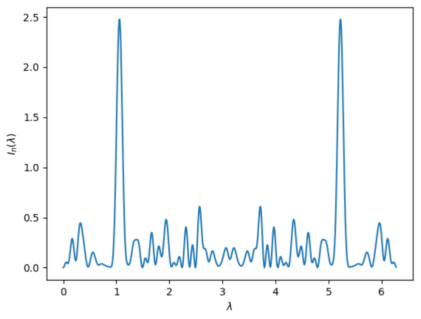


FIGURE 5 – Periodogram of the Harmonic process

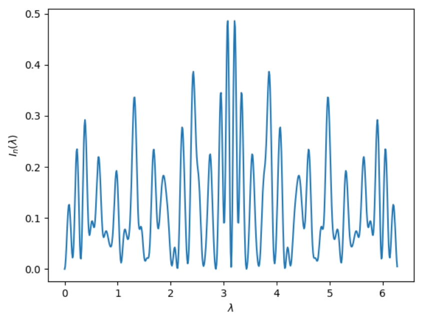


FIGURE 6 – Periodogram of WN(0, 1)

2.3

We have that $2\pi I_n(2\pi\frac{k}{m}) = \sum_{h=-n+1}^{n-1} \widehat{\gamma}_n(h)e^{-i\lambda h}$.
Thus, we can obtain $\widehat{\gamma}_n$ from $I_n(2\pi\frac{k}{m})$ by the inverse DTFT, with the condition to take $\boxed{m = 2n-1}$.
In this case we get that :

$$(\forall h \in \mathbb{Z}),\ \widehat{\gamma}_n(h) = \begin{cases} \frac{2\pi}{m}\sum_{k=0}^{m-1} I_n(2\pi\frac{k}{m})e^{i2\pi\frac{k}{m}h} & \text{if h < n} \\ 0 & \text{otherwise} \end{cases}$$

The function `acovb(X)` in the module `randproc` correctly computes the estimate of the autocovariance function from a set of samples :

- It first calculates the DTFT of order m = 2n-1 of $(X_0 - \hat{\mu}_n, ..., X_{n-1} - \hat{\mu}_n, 0, ..., 0) \in \mathbb{R}^m$. Then takes its square divided by n (not by $2\pi n$ since we're going to divide by 2π later.
- Then it calculates the inverse DTFT of order m of the quantity that was just calculated. We take its real part and discard the imaginary one, since it's theoretically zero.
- The function returns the first n, since $\widehat{\gamma}_n(h) = 0$ for all $h \geq n$.

2.4

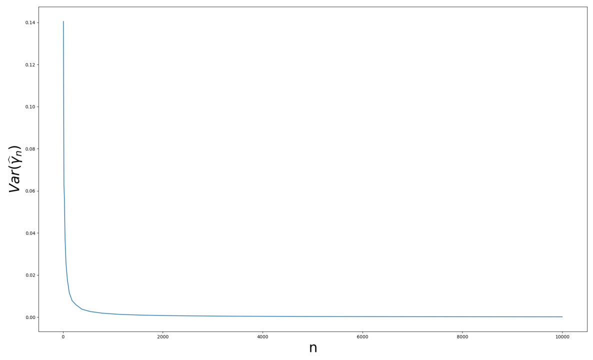
Since I_n is proportionate to the DTFT of $\widehat{\gamma}_n$, they have also proportionate variances (by Bessel's equality).

The script below permits to plot the variance of $\widehat{\gamma}_n$ for multiple values of n (which is basically the same as plotting the variance of the periodogram) :

```python
N = 50
variances = []
for n in np.logspace(1, 4, 20, dtype=int) :
    var_n = []
    for k in range(N) :
        #multiplying the experiences to calculate the empirical variance
        X = np.random.normal(0, 1, n)
        gamma = acovb(X)
        v = np.var(gamma)
        var_n.append(v)
    variances.append((sum(var_n))/len(var_n))

plt.plot(np.logspace(1, 4, 20, dtype=int), variances)
plt.xlabel('n')
plt.ylabel(r'$Var(\widehat{\gamma}_n)$')
```

Listing 2 – Plot of the variance of the periodogram

We can see that the variance of the periodogram quickly converges to zero, as n goes to infinity. In fact, as n increases, we observe the signal over longer time, which allows us to estimate γ_n with more accuracy.
The periodogram is actually a good estimator of the PSD, in the case of a white noise.



3 Exercise 3 : Filtering random processes

3.1 Part I - Yule-Walker equations

3.1.1

Let $t \in \mathbb{Z}$ and $h \geq 1$.
We have that $Z_t = X_t - \sum_{k=1}^{p} \phi_k X_{t-k}$.
So

$$\mathbb{E}[X_{t-h} Z_t] = \mathbb{E}[X_{t-h} X_t] - \sum_{k=1}^{p} \phi_k \mathbb{E}[X_{t-k} X_{t-h}] \tag{15}$$

$$= \gamma(h) - \sum_{k=1}^{p} \phi_k \gamma(h-k) \tag{16}$$

But we have also that

$$\gamma(h) = Cov(X_h, X_0) \tag{17}$$

$$= Cov(Z_h, X_0) + \sum_{k=1}^{p} \phi_k Cov(X_{h-k}, X_0) \tag{18}$$

$$= Cov(Z_h, Z_0) + \sum_{k=1}^{p} \phi_k \gamma(h-k) \quad (X_0 = Z_0 \text{ since X is causal}). \tag{19}$$

$$= \underbrace{\gamma_Z(h)}_{=0\ (h>0)} + \sum_{k=1}^{p} \phi_k \gamma(h-k) \tag{20}$$

$$\gamma_X(h) = \sum_{k=1}^{p} \phi_k \gamma(h-k) \tag{21}$$

$$\text{Hence, } \forall h \geq 1, \mathbb{E}[X_{t-h} Z_t] = 0$$

3.1.2

In the demonstration below, we have proved that :

$$(\forall h \geq 1), \quad \gamma(h) = \sum_{k=1}^{p} \phi_k \gamma(h-k)$$

3.1.3

We have that

$$\gamma(0) = Cov(X_t, X_t) \tag{22}$$

$$= Cov(\sum_{k=1}^{p} \phi_k X_{t-k} + Z_t, X_t) \tag{23}$$

$$= \sum_{k=1}^{p} \phi_k Cov(X_{t-k}, X_t) + Cov(Z_t, \sum_{k=1}^{p} \phi_k X_{t-k} + Z_t) \tag{24}$$

$$= \sum_{k=1}^{p} \phi_k \gamma(-k) + \sum_{k=1}^{p} \phi_k \underbrace{\mathbb{E}[Z_t X_{t-k}]}_{=0\ (Q1)} + Cov(Z_t, Z_t) \tag{25}$$

$$\gamma(0) = \sum_{k=1}^{p} \phi_k \gamma(-k) + \sigma^2 \tag{26}$$

3.1.4

Let

$$\Gamma_{p+1} = \begin{bmatrix} \gamma(0) & \gamma(-1) & \cdots & \gamma(-p) \\ \gamma(1) & \gamma(0) & \cdots & \gamma(-p+1) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma(p) & \gamma(p-1) & \cdots & \gamma(0) \end{bmatrix} = (\gamma(i-j))_{1\leq i,j\leq p+1}$$

By the previous questions, we get that :

$$\Gamma_{p+1} \begin{bmatrix} 1 \\ -\phi_1 \\ \vdots \\ -\phi_p \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

3.2 Part II - Estimation

We can estimate the variance and AR coefficients by the script just below :

```
#--- Generating n=1000 samples of an AR(4) process ---#
n = 1000
p = 4
std = 1
X, phi = rp.genAR(p,n,std)
coeff = np.concatenate(([1], -phi))

#--- An estimation of the Toeplitz matrix ---#
gamma =  rp.acovb(X)
Gamma_hat= la.toeplitz( gamma[:p+1])

#--- Solving the equation system ---#
v1 = np.zeros(p+1)
v1[0]=1
c = np.dot(np.linalg.inv(Gamma_hat), v1)
sigma2Est = 1/c[0]
estimated_coeff = sigma2Est*c
estimated_coeff[0] = 1

#--- Displaying the results ---#
print("The estimated coefficients are : ", estimated_coeff)
print("The actual coefficients are : ", coeff)

#--- Computing the relative error ---#
rel_err =  np.linalg.norm(err)/np.linalg.norm(coeff)
print('Relative error {0:.2%} '.format(rel_err))
```

Listing 3 – Estimating the variance and the AR coefficients

Where, while solving the equation system, we have that

$$v = \hat{\Gamma}_{n,p+1}^{-1} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} \frac{1}{\sigma^2} \\ \frac{-\phi_1}{\sigma^2} \\ \vdots \\ \frac{-\phi_p}{\sigma^2} \end{bmatrix}$$

We get as a result :

```
The estimated coefficients are : [ 1. -1.27446643 1.40728448 -0.81985054 0.4828439 ]
The actual coefficients are : [ 1. -1.2749218 1.43661759 -0.82034217 0.47993837]
Relative error 1.25 %
```

$\rightsquigarrow$ We've got a good estimation of the variance of this AR(4) process and of its AR coefficients.

We can then show the estimated PSD and estimated poles (compared to the theoretical ones) thanks to the script below :

```python
plt.figure()
nPoints = np.int(np.exp2( np.ceil(np.log2(X.size))))
nu = np.linspace(-0.5, 0.5 - 1/nPoints, nPoints)
omega = 2*np.pi*nu
w1, H1 = sig.freqz([1], coeff, omega)
w2, H2 = sig.freqz([1], estimated_coeff, omega)
plt.plot(w1,abs(H1))
plt.plot(w2,abs(H2))
plt.legend(['Amplitude Frequency Response (FR) of the AR filter','Estimated Amplitude FR' ])

plt.figure()
plt.axis('equal')
plt.grid()
zk    = np.roots(coeff)
plt.plot(np.real(zk),np.imag(zk), 'x' )
zk2 = np.roots(estimated_coeff)
plt.plot(np.real(zk2),np.imag(zk2), '+' )
plt.legend(['Poles of the AR transfer function','Poles of the estimated AR'])
t = np.linspace(-np.pi,np.pi,1000)
plt.plot(np.sin(t),np.cos(t), 'black')
```

Listing 4 – Computing the relative error & plot of the estimated PSD and poles

Thus, we get as a result :

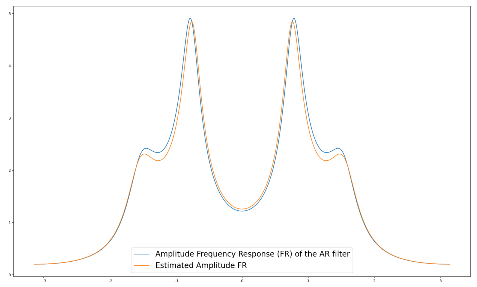


FIGURE 7 – Amplitude FR - comparison

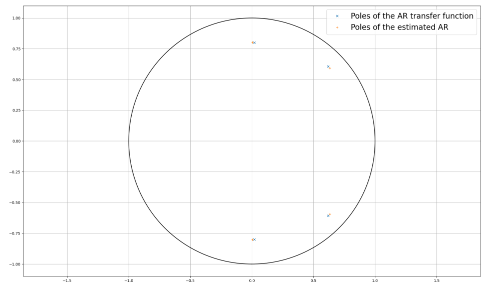


FIGURE 8 – Poles of the AR - comparison

3.3 Part III - Application to speech signal

3.3.1

Knowing that $\begin{cases} f_0 = \frac{1}{T} \in [80Hz, 400Hz] \\ \frac{1}{T} \in [\frac{1}{maxT}, \frac{1}{minT}] \\ \text{The sampling frequency is } F_s \end{cases} \Rightarrow \begin{cases} minT \geq \frac{F_s}{400} \\ maxT \geq \frac{F_s}{80} \end{cases}$

3.3.2

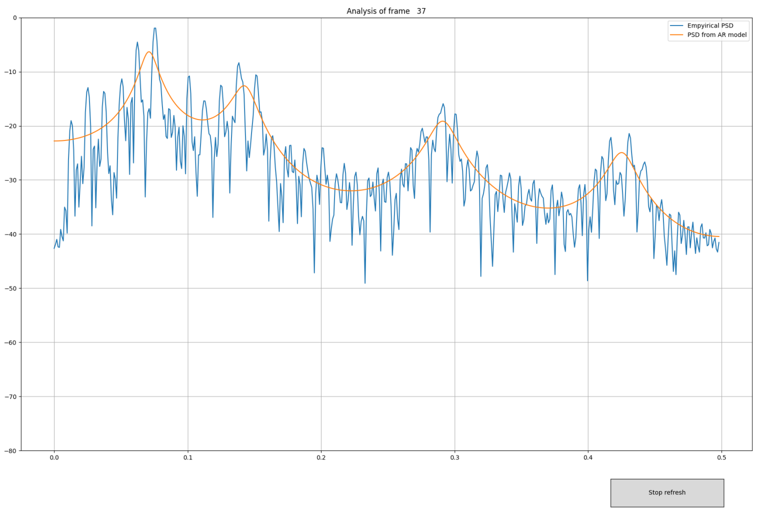
```python

#--- Load the sound as a signal ---#
filename = "../phrasezoe8k.wav"
wavObject = wave.open(filename)
nchannels, sampwidth, framerate, nframes, comptype, compname  = wavObject.getparams()
s = wavObject.readframes(nframes)
samples =  np.frombuffer(s, dtype='int16')/32768.
x = sig.lfilter([1, -0.98], [1], samples)

#--- Analysis parameters ---#
frameLen = 0.04 # frame length in seconds
```

```
    p          =   12 # AR parameter model
    overlap    =   50 # frame overlap in %
    Fmax       =  200 # maximum voicing frequency
    Fmin       =   60 # minimum voicing frequency
    Nx         = int(frameLen*framerate) # lenght of the frames in number of samples
    lag        = int(Nx * overlap / 100)
    framesToProc   = int((nframes-Nx)/lag -0.5) # floor
    minT       = int(framerate/Fmax -0.5)  # floor respects the condition in Q-3.3.1
    maxT       = int(framerate/Fmin + 0.5) # ceil respects the condition in Q-3.3.1
    synth      = np.zeros_like(samples)
    pitch      = np.zeros(framesToProc)
    coeff      = np.zeros([p,framesToProc])
    sigma2     = np.zeros(framesToProc)

    for frameIndex in  np.arange(framesToProc):
        startSample = frameIndex*lag
        stopSample  = startSample + Nx
        frame = x[startSample:stopSample]

        ### ANALYSIS
        #--- Pitch detection ---#
        pitch[frameIndex] = rp.detectPitch(frame, minT, maxT)

        ### write the code to estimate the coefficients for the current frame
        ### note that the leading "1" does not need to be stored
        gamma =  rp.acovb(frame)
        Gamma_hat = la.toeplitz(gamma[:p])
        v1 = np.zeros(p)
        v1[0]=1

        c = np.dot(np.linalg.inv(Gamma_hat), v1)
        sigma2Est = 1/c[0]
        estimated_coeff = sigma2Est*c

        coeff[:,frameIndex]= estimated_coeff
        sigma2[frameIndex]= sigma2Est
        c1 = coeff[:,frameIndex]
```

Listing 5 – Estimating the pitch and the signal parameters

We get the results as an interactive display, and for the frame 37

The model used (AR(12)) seems, graphically, to follow the shape and pattern of the PSD of the signal, which has very complex evolution. But it follows in a ”simplistic” way, since the details are taken off, as if we were taking the average of the empirical PSD.



After the synthesizing part, we get the plots :
In FIGURE 9 the plots correspond, respectively (from up to down) to
— The plot of the original signal
— The plot of the detected pitch, depending on the frame
— The plot of the estimated variance, depending on the frame

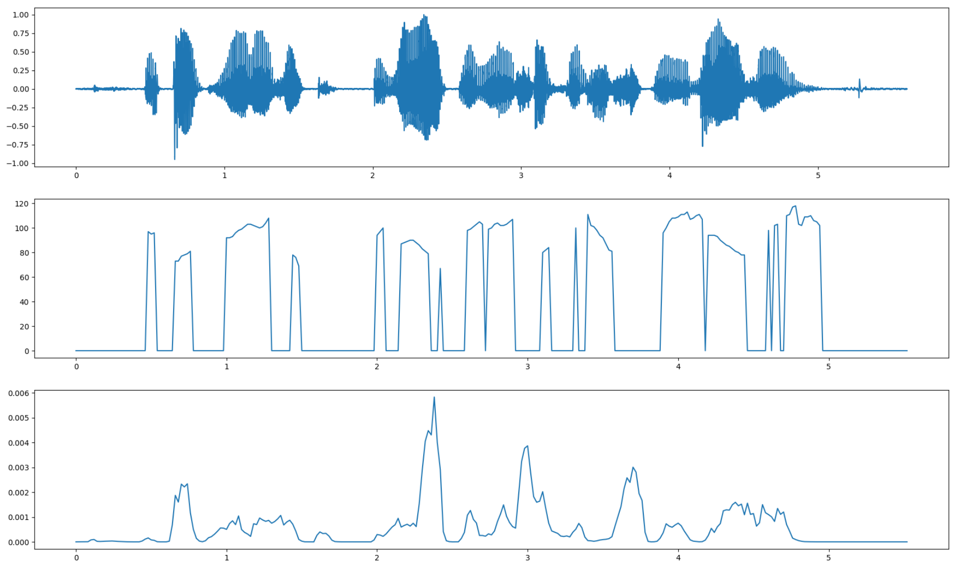


Figure 9

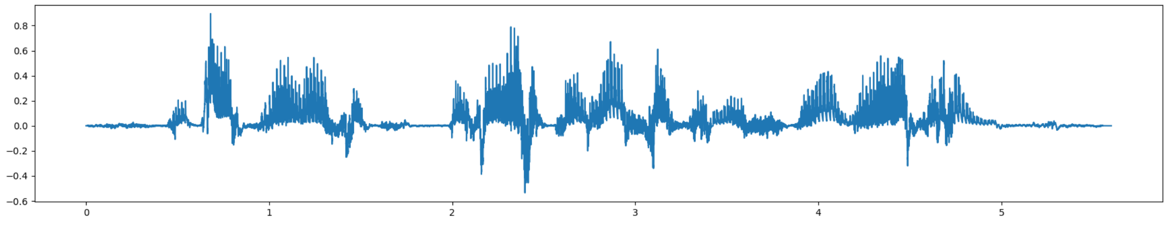


Figure 10 – Plot of the synthesized signal

After listening to the synthesized signal, one can recognize the original sound but it's so noisy. This is due to the fact that, graphically in FIGURE 9, the period is zero on considerably many intervals. Which means that we have added "too much" white noise to the synthesized signal.