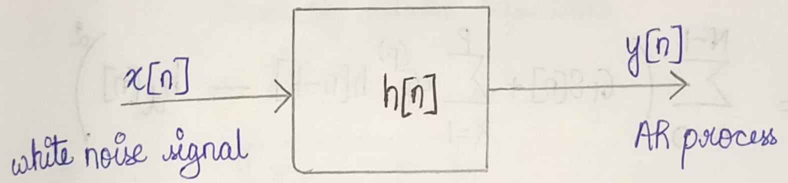
Assignment-4

J. Kalyan Raman

17EE35004

Levinson Durbin Algorithm to calculate LPC :

LD algorithm is used most commonly to estimate the all-pole AR model parameters, because the design equations used to obtain the best-fit AR model are simpler than those used for MA or ARMA modelling.



consider a general all-pole filter (order P) : $H(z) = \dfrac{Y(z)}{X(z)}$

$$\frac{Y(z)}{X(z)} = \frac{G}{1 - \sum_{k=1}^{P} \alpha_k^{(P)} \cdot z^{-k}}$$

applying inverse z transform, we get.

$$y[n] = G\, x[n] + \sum_{k=1}^{P} \alpha_k^{(P)} \cdot y[n-k]$$

Now we want to obtain a filter T.F to an arbitrary desired filter T.F : $H_d(z)$. This is done by minimizing the average square error between magnitude of frequency response of desired filter $H_d(e^{j\omega})$ and all pole filter $H(e^{j\omega})$

$$e^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| H(e^{j\omega}) - H_d(e^{j\omega}) \right|^2 d\omega$$

applying parseval's theorem :

$$e^2 = \sum_{n=0}^{N-1} \left(h[n] - h_d[n] \right)^2$$

Since h[n] is impulse response ; system response to $\delta[n]$

$$h[n] = G \cdot \delta[n] + \sum_{k=1}^{P} \alpha_k^{(P)} \cdot h[n-k]$$

$$\Rightarrow \quad e^2 = \sum_{n=0}^{N-1} \left(G\,\delta[n] + \sum_{k=1}^{P} \alpha_k^{(P)} \cdot h[n-k] - h_d[n] \right)^2$$

for error to be minimum :

$$\frac{de^2}{da_K} = 0$$

$$\sum_{n=0}^{N-1} 2\left(G\,\delta[n] + \sum_{l=1}^{P} \alpha_l^{(P)} \cdot h[n-l] - h_d[n] \right) \cdot h[n-k] = 0$$

$$\sum_{n=0}^{N-1} \left(G\,\delta[n] \cdot h[n-k] + \sum_{l=1}^{P} \alpha_l^{(P)} \cdot h[n-l] \cdot h[n-k] \right)$$

$$= \sum_{n=0}^{N-1} h_d[n] \cdot h[n-k]$$

Since system is causal, G won't enter solution.

$$\sum_{l=1}^{P} \alpha_l^{(p)} \cdot \left(\frac{\sum_{n=0}^{N-1} h[n-l] \cdot h[n-k]}{N} \right) = \frac{\sum_{n=0}^{N-1} h_d[n] \cdot h[n-k]}{N}$$

let $\phi_{yy}[m] = \frac{1}{N} \sum_{n=0}^{N-1} h[n] \cdot h[n-m] = \phi[m]$

$$\Rightarrow \boxed{\sum_{l=1}^{P} \alpha_l^{(p)} \cdot \phi[k-l] = \phi[k]}$$

for $k = 1, 2, ---, P$.

P set of linear equations.

$$\Rightarrow \boxed{R \underline{\alpha} = \underline{P}}$$

where $R = \begin{bmatrix} \phi[0] & \phi[1] & --- & \phi[P-1] \\ \phi[1] & \phi[0] & --- & \phi[P-2] \\ \vdots & \vdots & & \vdots \\ \phi[P-1] & \phi[P-2] & --- & \phi[0] \end{bmatrix}$

$\underline{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_P \end{bmatrix}$, $\underline{P} = \begin{bmatrix} \phi[1] \\ \phi[2] \\ \vdots \\ \phi[P] \end{bmatrix}$

* a direct solution is given by [Complexity: $O(N^3)$]

$\underline{\alpha} = R^{-1}\underline{P}$, but it is tough to compute inverse for large number. So instead we

But in this algorithm, we find solution in a recursive method to reduce the complexity.

The basic idea of the recursion is to find the solution $\underline{\alpha}_{P+1}$ for the $(P+1)^{st}$ order case from the solution $\underline{\alpha}_P$ for the p^{th} order case.

$$R_{P+1} = \left[\begin{array}{cccc|c} & & & & \phi[P] \\ & & R_P & & \phi[P-1] \\ & & & & \vdots \\ & & & & \phi[1] \\ \hline \phi[P] & \phi[P-1] & \cdots & \phi[1] & \phi[0] \end{array}\right]$$

let $\underline{P} = \underline{r}$

$$\underline{r}_{P+1} = \begin{bmatrix} \underline{r}_P \\ \hline \phi[P+1] \end{bmatrix} \quad \text{and} \quad \underline{f}_P = \begin{bmatrix} \phi[P] \\ \phi[P-1] \\ \vdots \\ \phi[1] \end{bmatrix}$$

$$\Rightarrow \quad R_{p+1} = \left[\begin{array}{c|c} R_p & \underline{\rho}_p \\ \hline \underline{\rho}_p^T & \phi[0] \end{array}\right]$$

$$\underline{\alpha}_{p+1} = \left[\begin{array}{c} \underline{\alpha}_p \\ \hline \alpha_{p+1} \end{array}\right] = \left[\begin{array}{c} \underline{\alpha}_p \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} \underline{\epsilon}_p \\ \hline k_{p+1} \end{array}\right]$$

where $\underline{\epsilon}_p$ is correction term

k_{p+1} is new $\alpha_{p+1} \rightarrow$ reflection coefficients

$$\therefore \quad R_{p+1} \cdot \underline{\alpha}_{p+1} = \underline{r}_{p+1}$$

$$\begin{bmatrix} R_p & \underline{\rho}_p \\ \underline{\rho}_p^T & \phi[0] \end{bmatrix} \left(\begin{bmatrix} \underline{\alpha}_p \\ 0 \end{bmatrix} + \begin{bmatrix} \underline{\epsilon}_p \\ k_{p+1} \end{bmatrix} \right) = \begin{bmatrix} \underline{r}_p \\ \phi[p+1] \end{bmatrix}$$

$$R_p \underline{\alpha}_p + R_p \underline{\epsilon}_p + \underline{\rho}_p \cdot k_{p+1} = \underline{r}_p$$

and

$$\underline{\rho}_p^T \underline{\alpha}_p + \underline{\rho}_p^T \underline{\epsilon}_p + \phi[0]\, k_{p+1} = \phi[p+1]$$

On simplifying and approximating

$$\underline{\epsilon}_p = \cancel{-k_p \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}} + k_p \cdot \underline{\bar{\rho}}_p$$

$$k_{p+1} = \frac{-\phi[p+1] - \underline{\rho}_p^T \cdot \underline{\alpha}_p}{E_p}$$

where $E_p = (1-k_p)^2 E_{p-1}$

$E_0 = \phi[0]$

$\therefore$ Recursion algorithm is as follows.

$E_0 = \phi[0]$

$$k_i = \frac{-\phi[i] - \sum_{j=1}^{i-1} a_j^{(i-1)} \cdot \phi[i-j]}{E_{i-j}} \quad \text{for } 1 \leq i \leq P$$

$$a_i^{(i)} = k_i$$

$$a_j^{(i)} = a_j^{(i-1)} + k_i \cdot a_{i-j}^{(i-1)} \quad \text{for } j = 1, 2, \ldots, i-1$$

$$E_i = (1 - k_i^2) E_{i-1}$$

after P steps, we arrive at p^{th} order estimate

here $E_p = \left[\prod_{i=1}^{P} (1 - k_i^2) \right] E_0 = \left[\prod_{i=1}^{P} (1 - k_i^2) \right] \phi[0]$

gives the estimate of variance of $x[n]$.

Levinson

Reflection coefficients $\xleftrightarrow{\text{interconversion}}$ AR parameters

$\phi[0], k_1, k_2, \ldots k_p$ — $a[1], a[2], \ldots a[P], \sigma^2$

Step Down

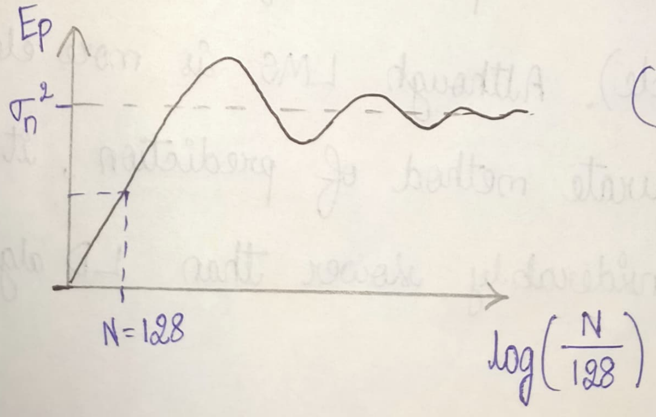
Note:

→ For all pole AR process generator to be stable, the poles must all lie inside unit circle in the Z-plane.

→ Time Complexity of LD algorithm is $O(N^2)$.
Space Complexity is $O(N)$

RESULTS:

→ For lower length, the estimation is too weak; but as length increases the estimation is becoming more perfect.



(ROUGHLY)

Comparison of LD with other algorithms :

→ LD V/S Cholesky decomposition :

* The cholesky decomposition is a method used to find inverse of matrix which has Hermitian Symmetry.

Computational Complexity , Space

LD – $O(N^2)$, $O(N)$

Cholesky – $O(N^3)$, $O(N^2)$

* LD exploits the fact that LPC analysis has Toeplitz Symmetry.

→ LD V/S LMS algorithm.

* LMS algo is an adaptive filter technique. But it does not guarantee minimum phase systems and stability while LD does. (minimumphase system – all poles and zeros lie within unit circle). Although LMS is more elegant and accurate method of prediction, it is considerably slower than LD algorithm.

MATLAB CODE :

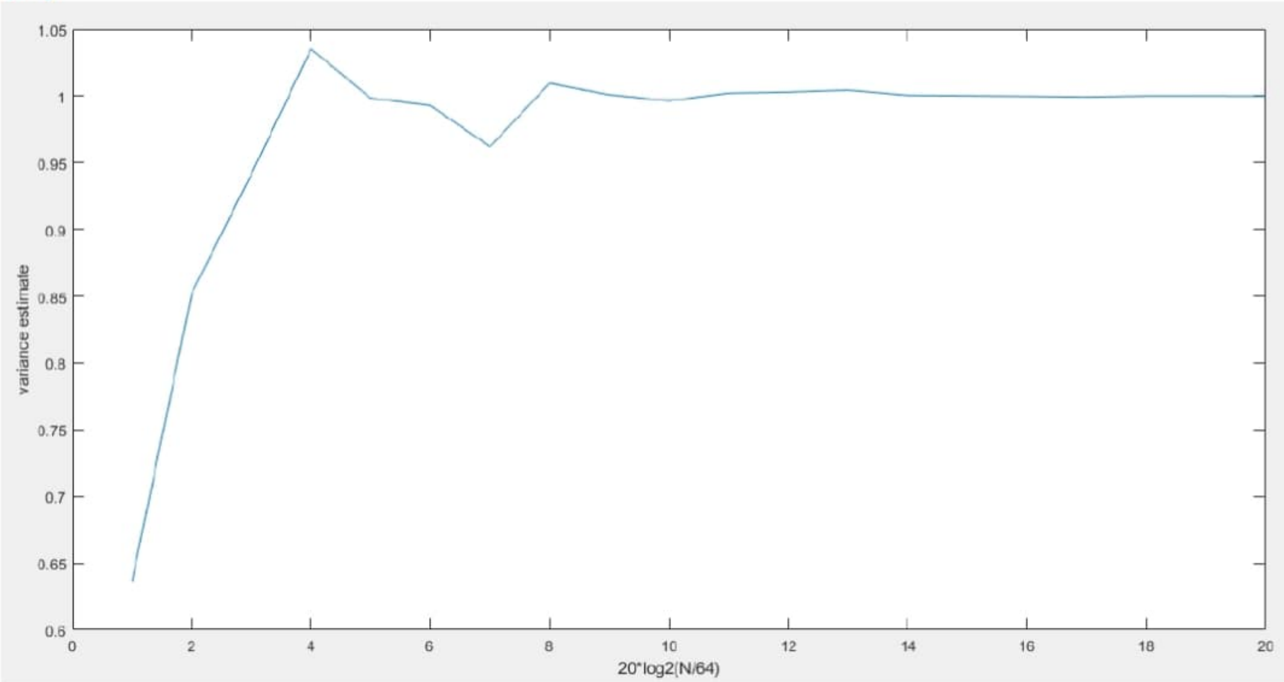
RTSP_Assg_3_17EE35004.m :

```
%% Levinson Durbin Recursive algorithm
% LD algorithm is used for Linear prediction filter coefficients
clc;
clear;

%% Initialization of global parameters
P = 3;                  % order of filter used
max_len = 20;           % maximum length of white noise signal
var = 1;                % variance of white noise signal

% Poles of the filter
% All poles are chosen to be within unit circle for STABILITY
p1 = 0.4;
p2 = 0.5;
p3 = 0.6;

% Finding filter coefficients as per given poles
a1 = -(p1+p2+p3);
a2 = (p1*p2+p2*p3+p3*p1);
a3 = -p1*p2*p3;

% AR parameters found by designed LD algorithm
alpha = zeros(max_len,P+1);
% AR parameters found by matlab inbuilt function
alpha1 = zeros(max_len,P+1);

% reflection coefficients in designed LD algorithms
k = zeros(max_len,P);
% reflection coefficients in matlab inbuilt function
k1 = zeros(max_len,P);

% variance estimate in designed LD algorithm
err = zeros(max_len,1);
% variance estimate in matlab inbuilt function
err1 = zeros(max_len,1);

% Loop running across all possible lengths
for n = 1:1:max_len

N = 128*2^(n-1);                    % length of signal
v = wgn(N,1,10*log10(var));         % noise signal with given specs
u1 = zeros(1,N+3)';

for i = 1:1:N
         % code for AR process generation
         u1(i+3) = -a1*u1(i+2)-a2*u1(i+1)-a3*u1(i)+v(i);
end

u = u1(4:N+3);

Rx = zeros(P+1,1);  % Autocorrelation sequence
```

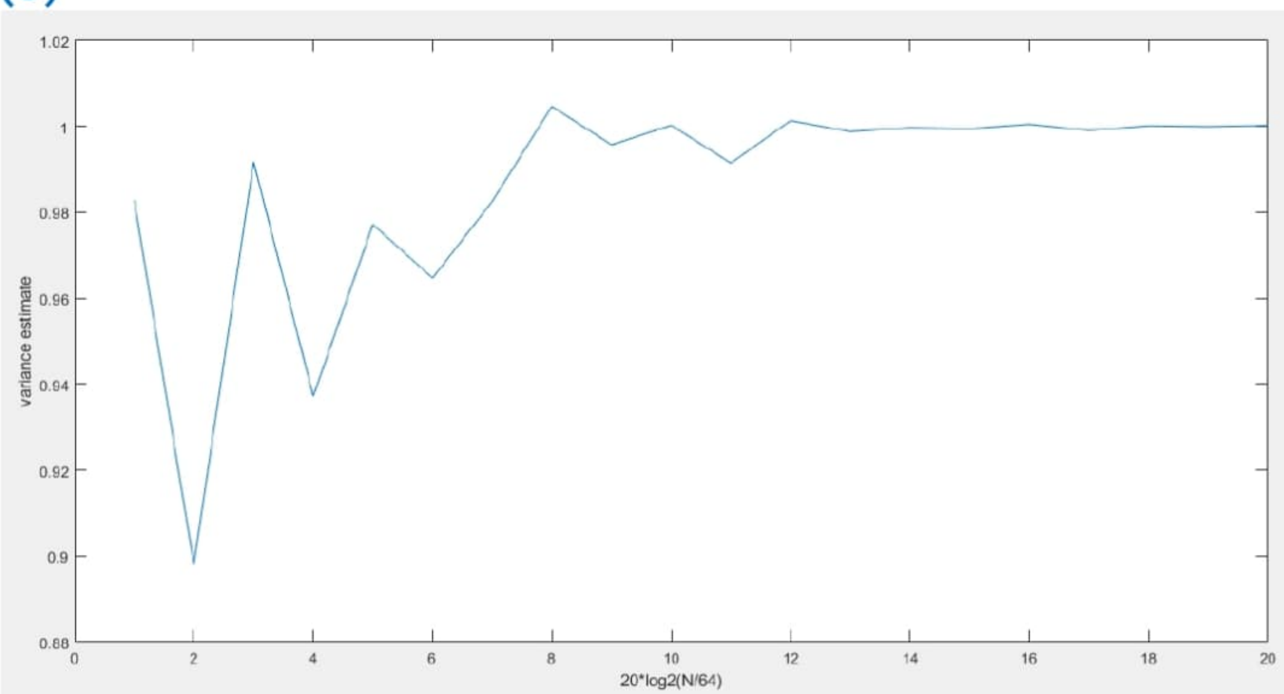
```
for i=1:P+1
    for j=i+1:N+1
        Rx(i) = Rx(i) + u(j-1)*u(j-i);
    end
    Rx(i) = Rx(i)/(N-i+1);
end

temp = zeros(P,P);  % temporary filter coefficients variable
E = zeros(P+1,1);   % temporary variance collecting variable
sum = 0;            % used to execute the vector product


% i=0 : zero level iteration
E(1) = Rx(1);               % initializing estimate
k(n,1) = -Rx(2)/Rx(1);      % first reflection coefficient
temp(1,1) = k(n,1);
E(2) = (1-(k(n,1))^2)*E(1); % first level variance estimate


% Iteration or LD recursion
for i = 2:1:P

    sum =0;
    for j=1:1:i-1
        % dot product of autocorrelation seq and filter coefficients
        sum = sum + temp(i-1,j)*Rx(i+1-j);
    end

    k(n,i) = -(Rx(i+1)+sum)/E(i);  % finding new reflection coefficient

    temp(i,i) = k(n,i); % assigning it to next order filter coefficient

    for j=1:1:i-1
        % updating lower order filter coefficients
        temp(i,j) = (temp(i-1,j) + k(n,i)*temp(i-1,i-j));
    end

    E(i+1) = (1-((k(n,i))^2))*E(i); % updating variance estimate

end

alpha(n,:) = [1,temp(3,:)];     % final filter coefficients solution
err(n) = E(P+1);                % final variance estimate

% using matlab inbuilt levinson durbin function
[alpha1(n,:),err1(n),k1(n,:)] = levinson(Rx,3);

end

%% Plotting the results

figure;
plot(1:1:max_len,err);
xlabel('20*log2(N/64)');
ylabel('variance estimate');
```

Plot results :

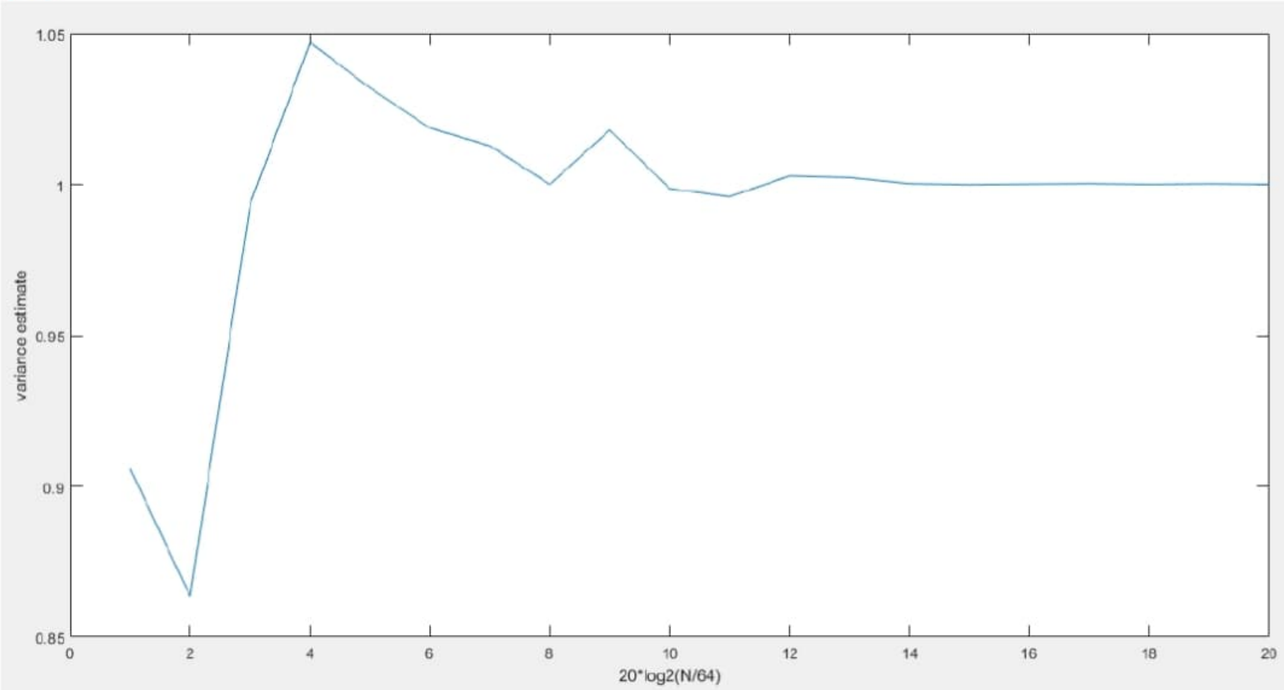
(a)



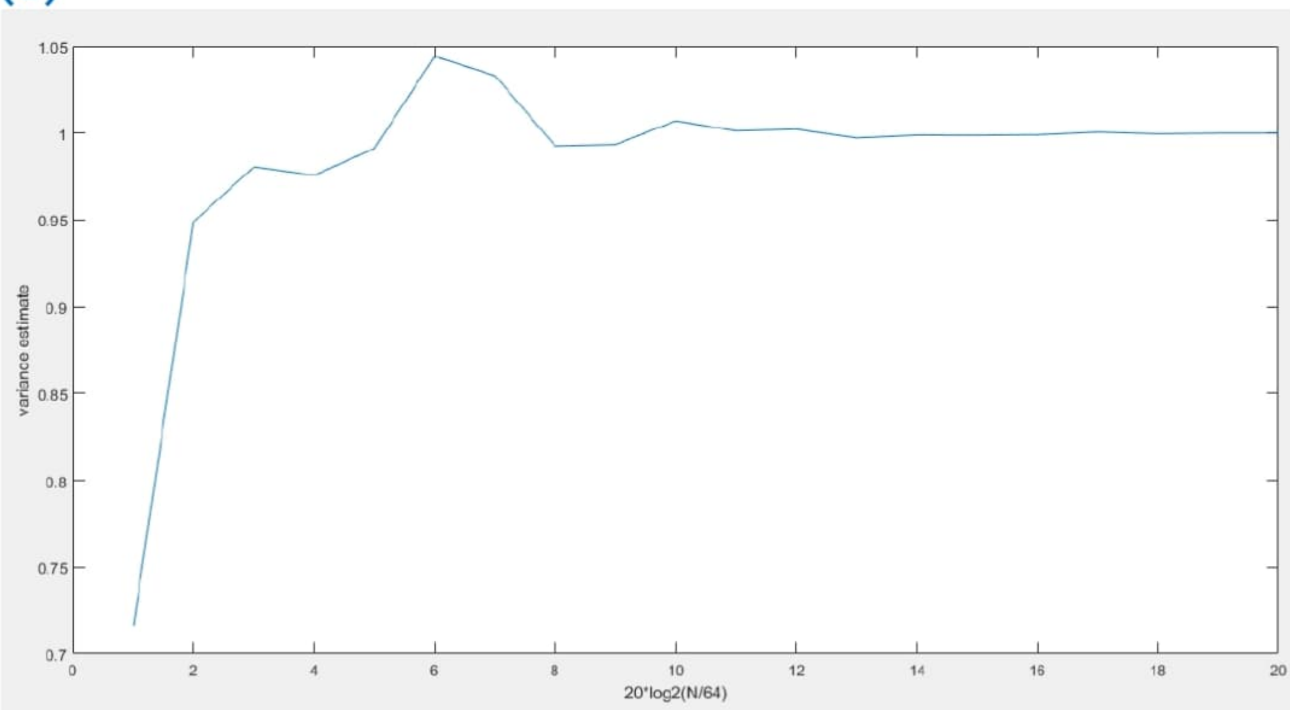
(b)



(c)

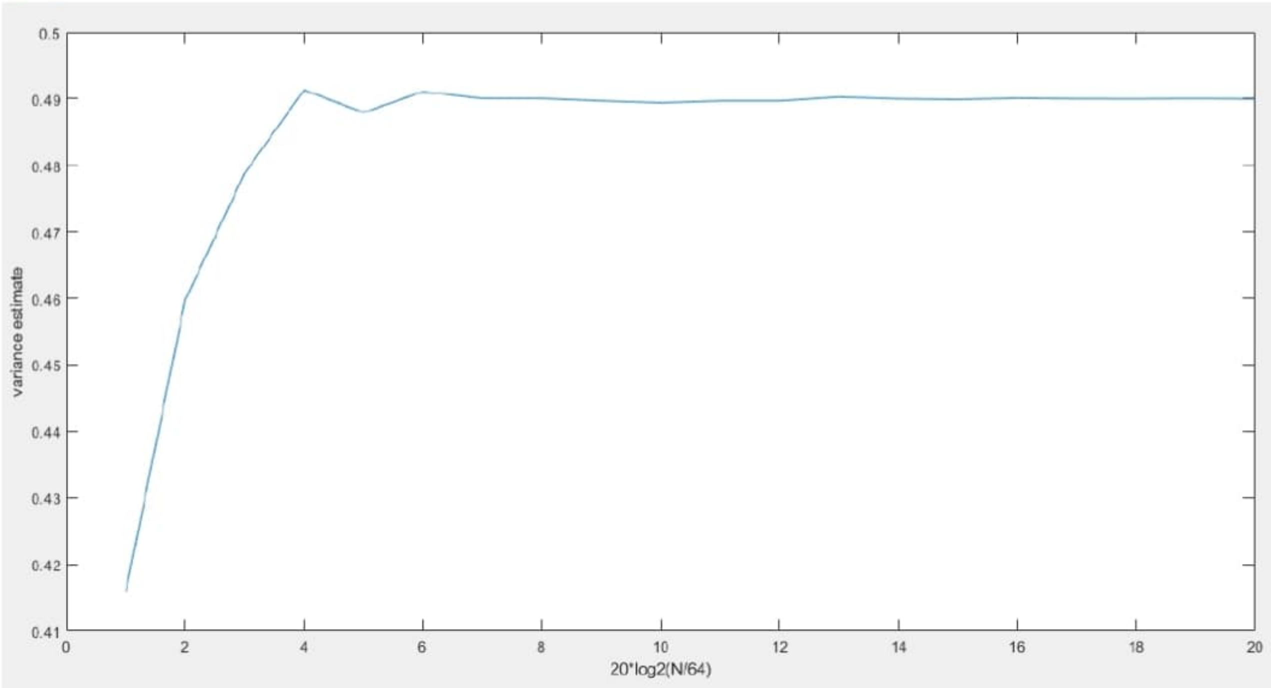

1.05
1
0.95
0.9
0.85
variance estimate
0
2
4
6
8
10
12
14
16
18
20
20*log2(N/64)

(d)


1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
variance estimate
0
2
4
6
8
10
12
14
16
18
20
20*log2(N/64)

This experiment is done multiple times and plot shows the average of variance estimate of each experiment to detect the statistical behaviour of variance estimate of LD algorithm:

(a) 50 times



(b) 100 times