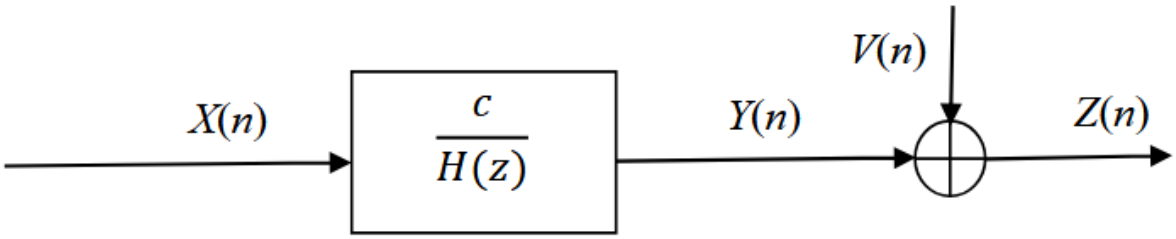
EE3005: Communication systems
Computational Assignment

Aim

To plot the variation of the ergodic Mean Square Error (MSE) with the Signal to Noise Ratio (SNR), for different coefficients of the filter used and for different samples of the random variables, which change how much the auto-regressive (AR) process is affected by its initial conditions.

Theory and formulae

- 

  where, X(n) is a discrete, white, Gaussian RP with variance 1,
  H(z) = $(1 - \alpha e^{j\theta} z^{-1})(1 - \alpha e^{-j\theta} z^{-1})(1 - \beta z^{-1})$ for $\theta = \pi/3$ radians,
  Y(n) is the output when X(n) is passed through a filter c/H(z),
  V(n) is a white Gaussian RP with variance σ_V^2 representing noise,
  Z(n) is the measured signal at the receiver; Z(n) = Y(n) + V(n)

- To ensure that the filter has gain 1, we choose c such that

$$c = (1 + h_1^2 + h_2^2 + h_3^2)^{0.5}$$

- The SNR in dB is given by

$$SNR_{dB} = 10 \log_{10} (1/\sigma_V^2)$$

  We can write the standard deviation of the noise as

$$\sigma_V = 10^{-\frac{\text{SNR (in dB)}}{20}}$$

- After normalization, the input output relationship is

$$Y(n) = h_1 Y(n-1) + h_2 Y(n-2) + h_3 Y(n-3) + cX(n)$$

  with initial conditions Y(0) = 0, Y(-1) = 0 and Y(-2) = 0.
  This implies that the useful values of X(n) occur only for $1 \leq n \leq N_0 + N$.

- To find the estimated values of h1, h2 and h3, we use the Yule-Walker equations

$$\begin{bmatrix} R(1) \\ R(2) \\ R(3) \end{bmatrix} = \begin{bmatrix} R(0) & R(1) & R(2) \\ R(1) & R(0) & R(1) \\ R(2) & R(1) & R(0) \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

where $R(k) \approx \left(\frac{1}{N-k}\right) \sum_{i=N_o+k+1}^{N_o+N} Z(i)Z(i-k)$

- The estimation error square, for each of the j = 1 to 500 Monte Carlo trials is given by

$$e^2(j) = \sum_{i=1}^{3} (h_i - \hat{h}_i(j))^2$$

- The ergodic Mean Square Error in the dB scale, averaged over the 500 trials is

$$MSE_{SNR} = 10log_{10}\left(\frac{1}{500}\sum_{j=1}^{500} e^2(j)\right)$$

Calculations and derivations

1) To find h_i in terms of α and β –

- Consider

$$Y(n) = h_1\, Y(n-1) + h_2\, Y(n-2) + h_3\, Y(n-3) + c\, X(n)$$

Taking the Z transform on both sides, we get

$$Y(z) = h_1\, z^{-1}\, Y(z) + h_2\, z^{-2}\, Y(z) + h_3\, z^{-3}\, Y(z) + c\, X(z)$$

$$Y(z) * (1 - h_1\, z^{-1} - h_2\, z^{-2} - h_3\, z^{-3}) = c\, X(z)$$

Solving further,

$$\frac{Y(z)}{X(z)} = \frac{c}{(1 - h_1\, z^{-1} - h_2\, z^{-2} - h_3\, z^{-3})} = \frac{c}{H(z)} \qquad \text{- (1)}$$

- Now, we have

$$H(z) = (1 - \alpha e^{j\theta} z^{-1})\,(1 - \alpha e^{-j\theta} z^{-1})\,(1 - \beta z^{-1})$$

$$= [1 - \alpha\, z^{-1}(e^{j\theta} + e^{-j\theta}) + \alpha^2\, z^{-2}]\,(1 - \beta z^{-1})$$

$$= [1 - \alpha\, z^{-1}(2\cos\theta) + \alpha^2\, z^{-2}]\,(1 - \beta z^{-1})$$

$$= [1 - \alpha\, z^{-1} + \alpha^2\, z^{-2}]\,(1 - \beta z^{-1}) \qquad \text{for } \theta = \pi/3 \text{ radians}$$

$$H(z) = 1 - z^{-1}\,(\alpha + \beta) + z^{-2}\,(\alpha^2 + \alpha\,\beta) - z^{-3}\,(\alpha^2\,\beta) \qquad \text{- (2)}$$

- Comparing this with $H(z) = 1 - h_1\, z^{-1} + h_2\, z^{-2} - h_3\, z^{-3}$, we get

$$h_1 = (\alpha + \beta),$$

$$h_2 = -(\alpha^2 + \alpha\,\beta),$$

$$h_3 = (\alpha^2\,\beta), \text{ and}$$

$$c = (1 + h_1^2 + h_2^2 + h_3^2)^{0.5}$$

2) Derivation of the Yule-Walker equations –

- Consider

$$Y(n) = h_1\, Y(n-1) + h_2\, Y(n-2) + h_3\, Y(n-3) + c\, X(n)$$

Multiplying by Y(n-1), we get

$$R(1) = h_1\, R(0) + h_2\, R(1) + h_3\, R(2) \qquad \text{- (3)}$$

where,

$$R(k) = E[Y(n)\, Y(n-k)] \text{ and } R(-k) = R(k)$$

- Similarly, multiplying by Y(n-2) and Y(n-3), we get

$$R(2) = h_1\, R(1) + h_2\, R(0) + h_3\, R(1) \qquad \text{- (4)}$$

$$R(3) = h_1\, R(0) + h_2\, R(1) + h_3\, R(2) \qquad \text{- (5)}$$

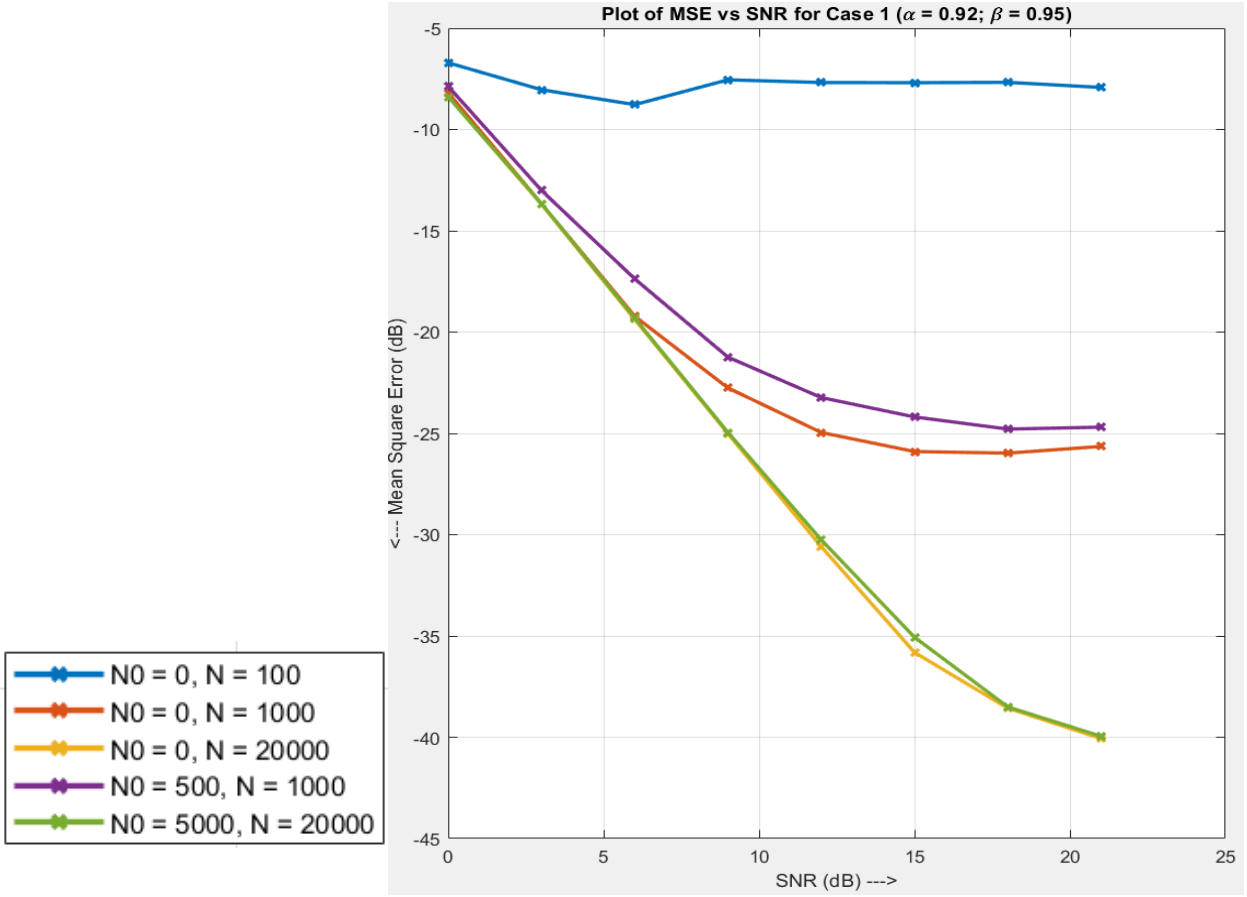
- When equations (1), (2) and (3) are written in the matrix form, we get the Yule-Walker equations

$$\begin{bmatrix} R(1) \\ R(2) \\ R(3) \end{bmatrix} = \begin{bmatrix} R(0) & R(1) & R(2) \\ R(1) & R(0) & R(1) \\ R(2) & R(1) & R(0) \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

where $$R(k) \approx \left(\frac{1}{N-k}\right) \sum_{i=N_o+k+1}^{N_o+N} Z(i)Z(i-k)$$

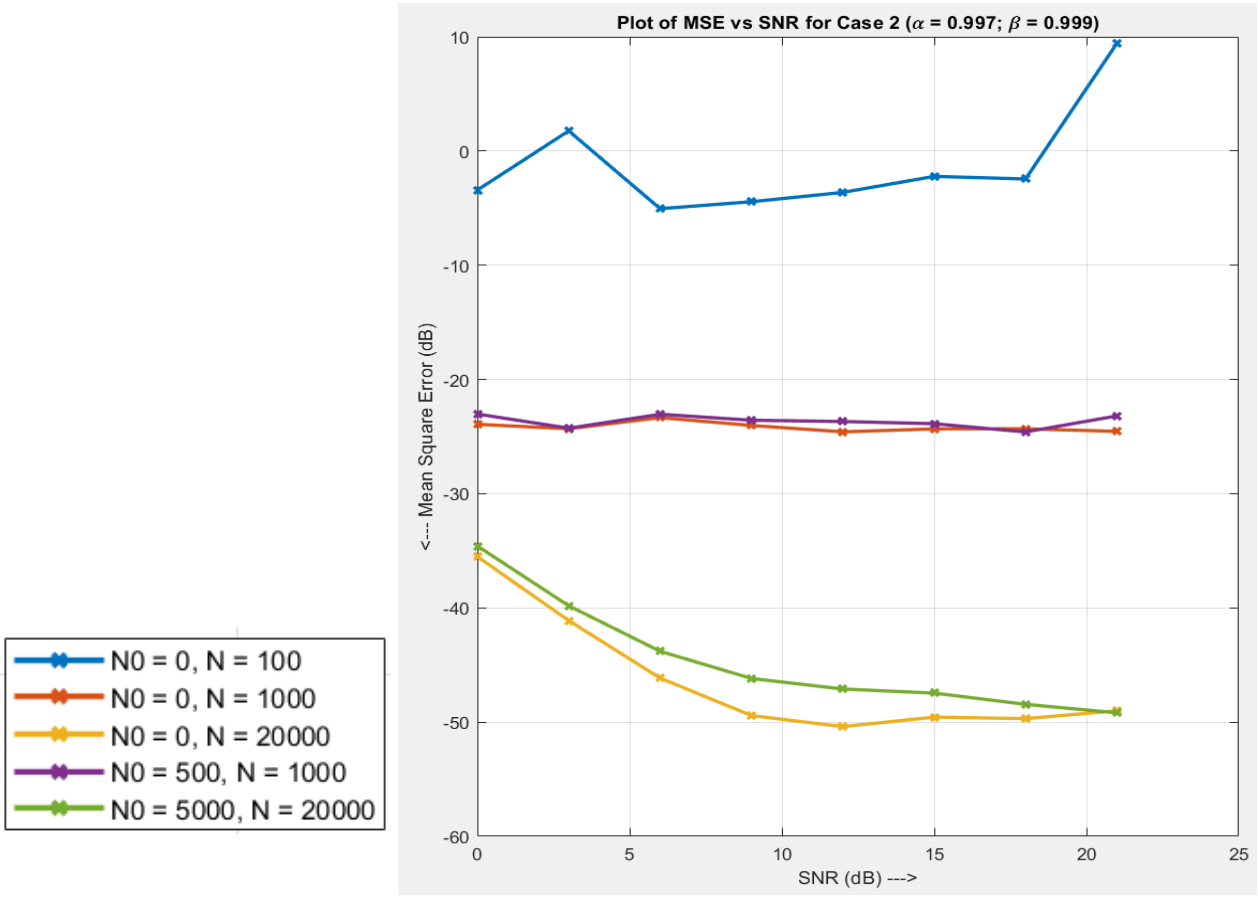
Observations and results

1) For Case 1 (α = 0.92 and β = 0.95):



	MSE (dB)				
SNR (dB)	**N_0=0, N=100**	**N_0=0, N=1000**	**N_0=0, N=20000**	**N_0=500, N=1000**	**N_0=5000, N=20000**
0	-6.706	-8.186	-8.390	-7.870	-8.391
3	-8.037	-13.662	-13.693	-13.006	-13.657
6	-8.773	-19.219	-19.376	-17.376	-19.314
9	-7.547	-22.758	-25.015	-21.237	-24.954
12	-7.681	-24.956	-30.591	-23.232	-30.257
15	-7.690	-25.897	-35.815	-24.188	-35.063
18	-7.672	-25.974	-38.564	-24.788	-38.501
21	-7.9237	-25.645	-40.062	-24.693	-39.951

2) For Case 2 (α = 0.997 and β = 0.999):



	MSE (dB)				
SNR (dB)	**N_0=0, N=100**	**N_0=0, N=1000**	**N_0=0, N=20000**	**N_0=500, N=1000**	**N_0=5000, N=20000**
0	-3.419	-23.926	-35.508	-23.033	-34.583
3	1.762	-24.317	-41.111	-24.266	-39.815
6	-5.048	-23.339	-46.145	-23.048	-43.199
9	-4.438	-24.029	-49.426	-23.570	-46.180
12	-3.621	-24.586	-50.396	-23.682	-47.103
15	-2.219	-24.327	-49.561	-23.874	-47.461
18	-2.447	-24.321	-49.693	-24.596	-48.444
21	-9.431	-24.544	-49.009	-23.190	-49.185

Conclusion and comments

1) Effect of N_0 and N on the MSE

- We can clearly see from the results that the *MSE is a lot better* (lower) when the simulation is run for a **large N**. This is because the samples father away from n=1 are affected a lot lesser by the initial conditions.
- We can also see that a **higher value of N_0** results in a much lower impact of the initial conditions on the result, due to which the *MSE is significantly lower* even for the same number of samples (N-N_0).

2) Effect of the SNR on the MSE

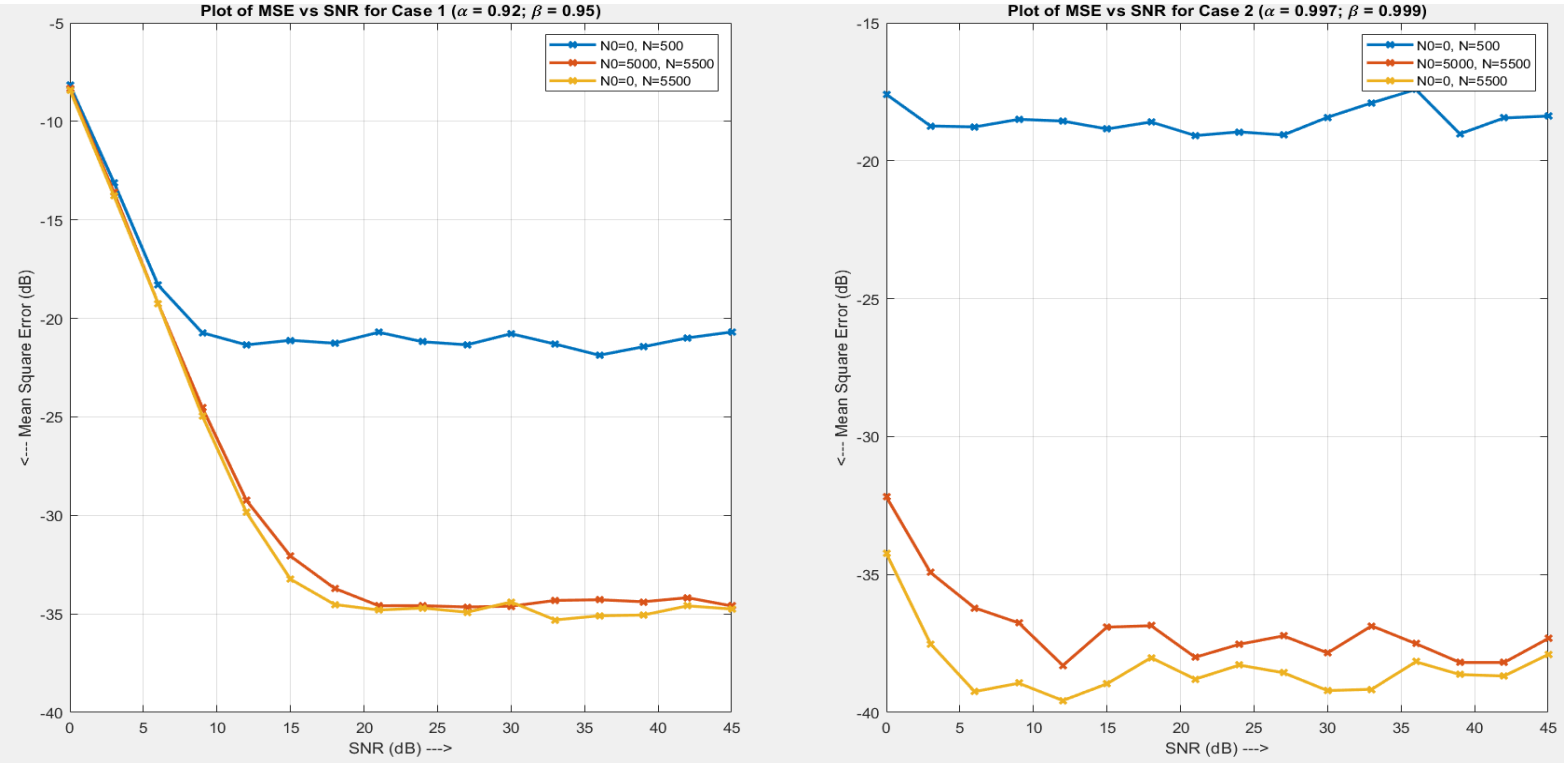
- The **MSE decreases with an increase in the SNR**, since the effect of noise is negligible for higher SNRs.
- The **MSE appears to saturate** after SNR = 21dB, likely due to the slow increase of quantities in the log scale.
- When a small number of samples near n=1 is taken, the MSE is very random due to the limited sample size. In fact, the error can sometimes be a lot higher for a higher SNR than it was for a lower SNR (as seen for N_0=0, N=100).

3) Effect of α and β on the MSE

- For a **higher α and β**, the MSE is lower for the same SNR.
- The MSE also saturates in a very smooth manner for a higher α and β.

- This can be attributed to the fact that in Case 2, that is, for a higher α and β, the **poles of the filter are a lot closer to the unit circle.**

4) The following figure can be used to illustrate the above conclusions



Brief explanation of code

- Matlab has an inbuilt function randn() to generate the required samples of our white Gaussian random process, X(n).
- Similarly, the noise V(n) can be obtained by multiplying the standard deviation σ_V to the set of points generated by the same randn() function.
- The samples of Y(n) are generated iteratively using the formula

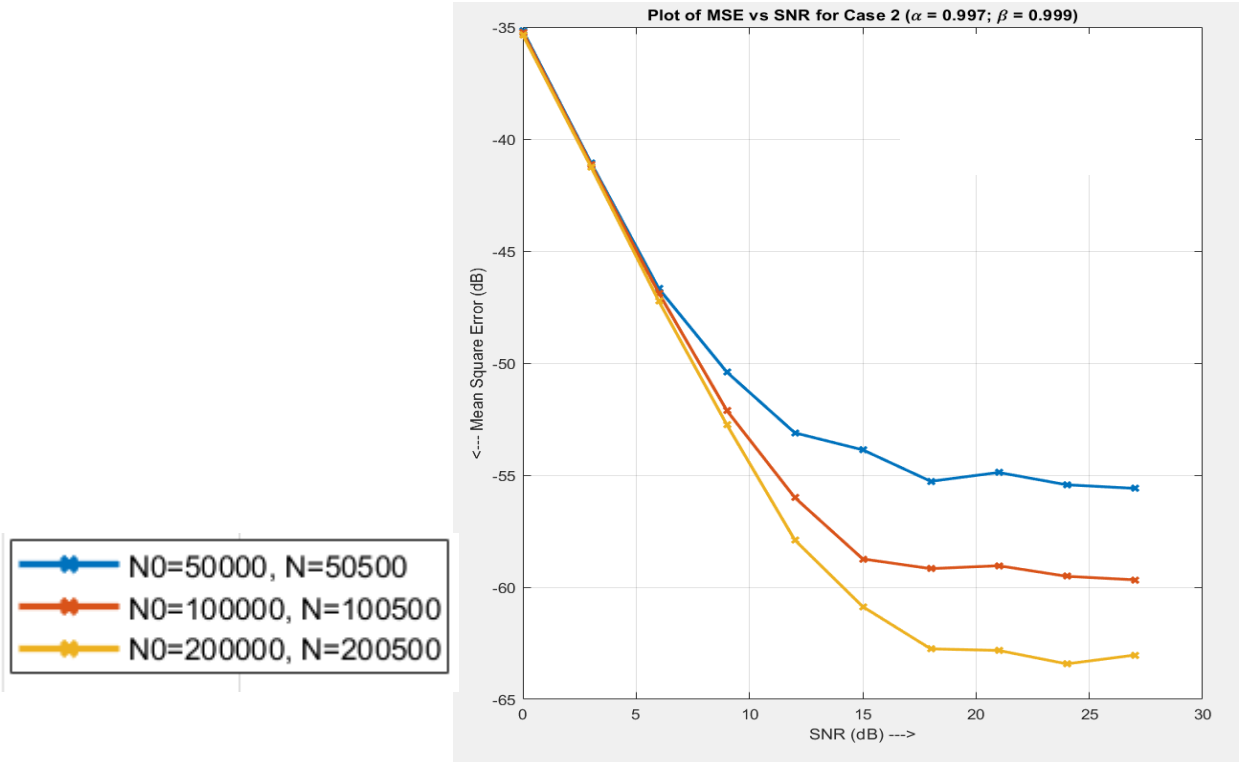
$$\mathrm{Y}(n) = h_1\,\mathrm{Y}(n-1) + h_2\,\mathrm{Y}(n-2) + h_3\,\mathrm{Y}(n-3) + c\,\mathrm{X}(n).$$

- We can then iterate over the 500 trials, the 8 different SNRs, the 5 different pairs of N_0 and N and the 2 sets of α and β using the 'For loop' in Matlab.

Bonus questions

1) Lower MSE for different values of N_0 and N

- We observe that on increasing the values of N, the MSE seems to reduce, seemingly arbitrarily.



- This can be seen from the above figure, where the values are computed for very large values.
- Thus, we can conclude that an arbitrarily high N (and N_0) can reduce the MSE arbitrarily.
- However, the simulations can take an incredibly large amount of time to run depening on the values of N_0 and N.

2) Using the Levinson-Durbin recursions to compute $h = R^{-1} r$

- Introduction –
  - To compute the inverse of an NxN matrix, Matlab's inbuilt functions have complexity of the order $O(N^3)$. However, if the matrix to be inverted is Toeplitz, it can be done in $O(N^2)$ using the Levinson-Durbin Algorithm.
  - In a Toeplitz matrix, every diagonal is constant. Thus, the matrix can be entirely represented by its first column. Thus, the algorithm can do **vector operations**, costing only $O(N^2)$ complexity.
  - The Levinson–Durbin algorithm was proposed first by Norman Levinson in 1947, improved by James Durbin in 1960, and subsequently improved to $4n^2$ and then $3n^2$ multiplications by W. F. Trench and S. Zohar, respectively.

- Algorithm –
  - The Levinson algorithm solves the system recursively, starting with a 1×1 system, then 2×2, then 3×3, and so on — building up to the full size N×N.
  - We initially start with just the top 1x1 block and make an estimate of the value at the index.
  - At each step k, we solve a larger system by using the previous solution h_{k-1} and correcting it to account for the new row and column used in the Toeplitz matrix.
  - This is done by computing a reflection coefficient that captures how the new data (the added k^{th} row from the matrix) affects our previously estimated results, and then correcting accordingly.
- Implementation in Matlab –
  - Matlab has an **inbuilt function levinson()** that takes the order of an autoregressive process n and its autocorrelation sequence r. It returns the coefficients of the AR process, h.
  - However, we can also write our own user-defined function to implement the Levinson Durbin algorithm, which returns $h = R^{-1}*r$ with order of complexity $O(N^2)$.