Bilge Kaan Ateş

22102758

22.03.2024

**EEE321 SIGNALS AND SYSTEMS**

**LAB ASSIGNMENT 4**

**1) INTRODUCTION:**

In this lab, I work with ideal/imperfect integrator systems and the properties of LTI systems such as linearity, causality, memory etc. Moreover, I analyzed BIBO stability of the systems as well. I studied the impulse and unit step response of the integrator systems. In first part, I manually handled the integrator systems and analyze their LTI properties and test them in MATLAB. In the second part, I write **sumElements()** function and analyzed the systems BIBO stability via Matlab. In the third part, I investigate the differences of ideal and imperfect (exponentially decaying) integrators. In the fourth part, firstly I derived second order difference equation and implement it in Matlab. Then, I found the inverse system of it by manually and implement it in Matlab and observed these systems convolution gives me impulse response.

**2) LAB:**

* Part 1:

Part 1.1: Corresponding plots are obtained in Part 1.3.

In this part, I implemented **fsCoeffs** function which founds Fourier coefficients of the signal x within the range of –k to k. I changed the integral to the sum with the sampling period . You can see the following code:

function [fsCoeffs] = FSAnalysis(x, k)

N = length(x);

fsCoeffs = zeros(2\*k + 1, 1);

for n = 1:N

for m = -k:k

fsCoeffs(m+k+1) = fsCoeffs(m+k+1) + (1/N) \* x(n) \* exp(-1i \* m \* 2\*pi/N \* (n-1));

end

end

end

Part 1.2:

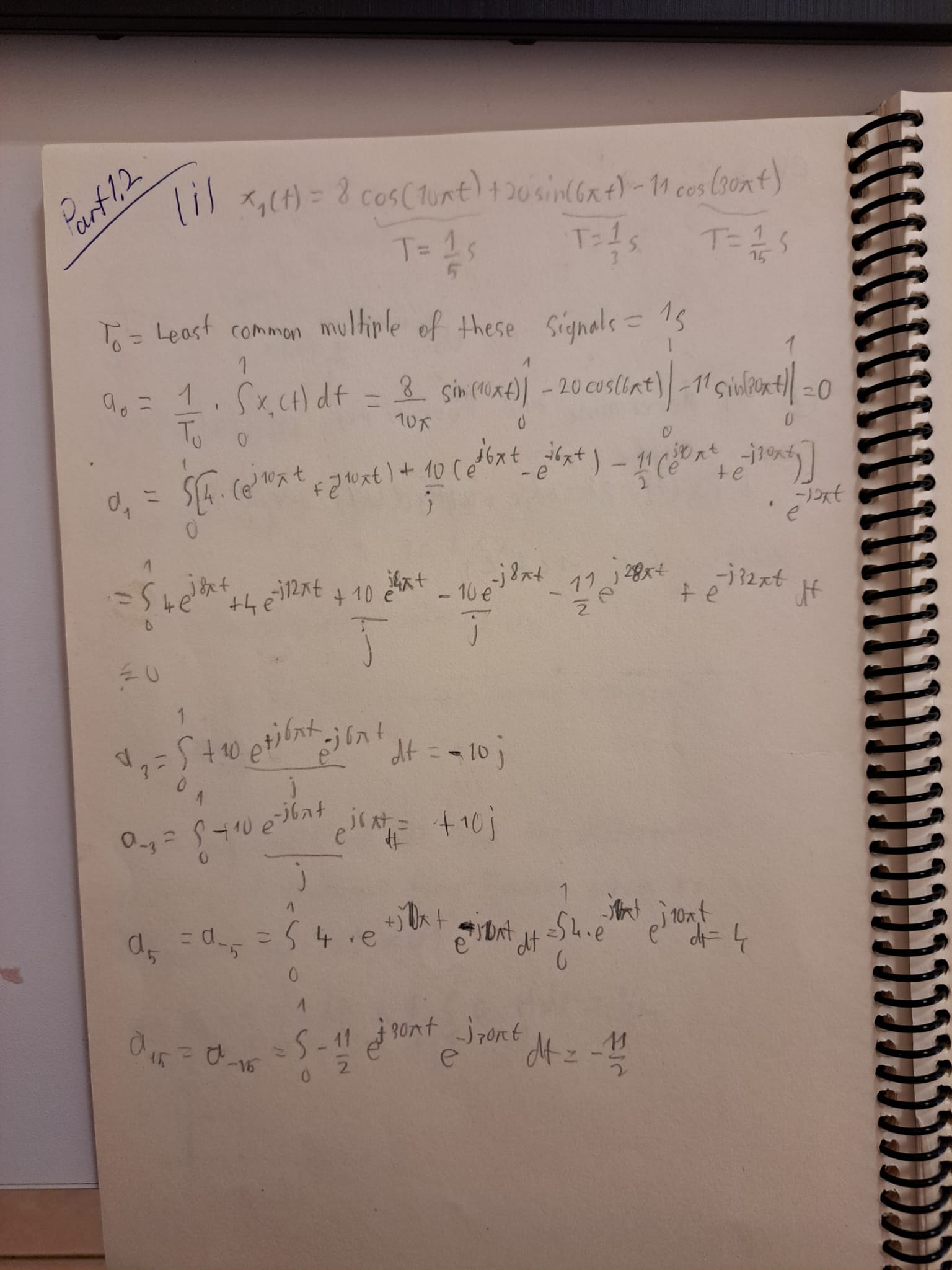


Fig. 1: Fourier series coefficients of

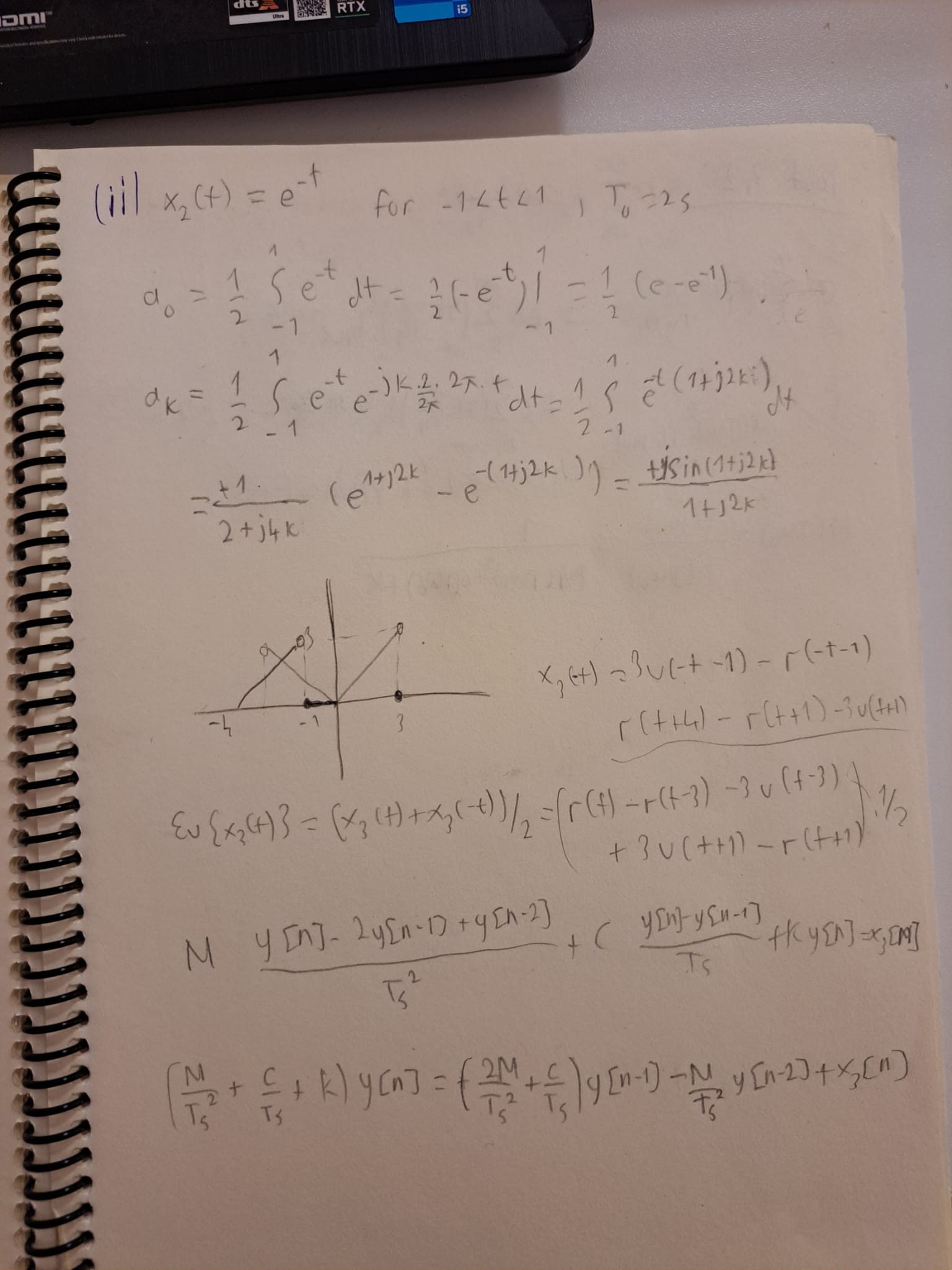


Fig. 2: Fourier series coefficients of

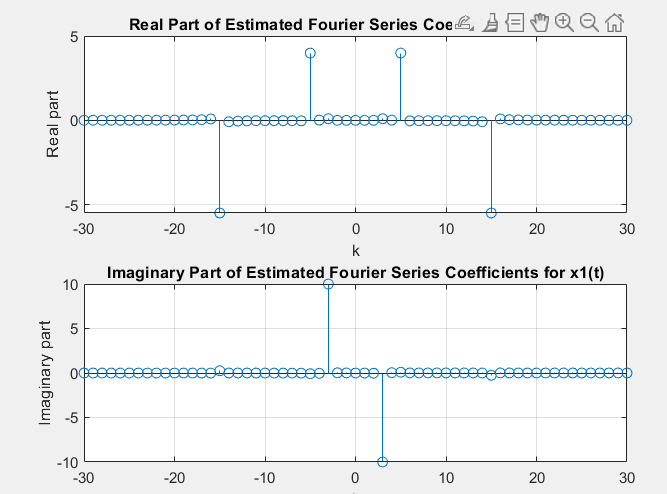


Fig. 3: Fourier series coefficients of

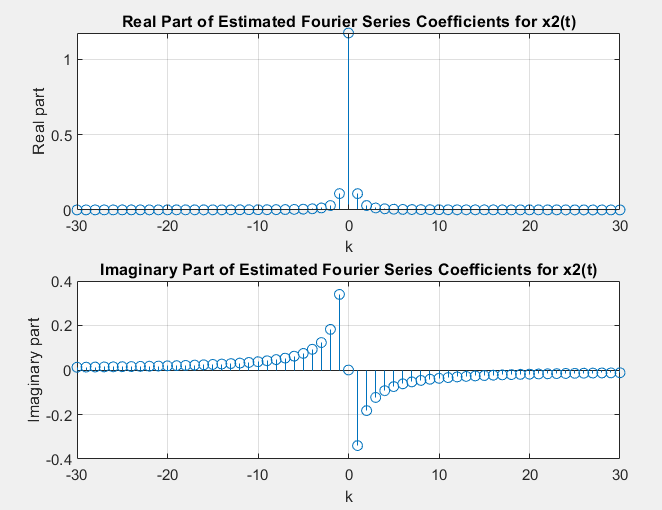


Fig. 4: Fourier series coefficients of

As it is seen from the figures coefficients are nearly equal to each other. However, the change of integral to sampling sum method gives small error rate. For example, in calculation while matlab result is 3.988 which has 0.3% error. This error rate can be reduced by shortening the Ts (sampling period).

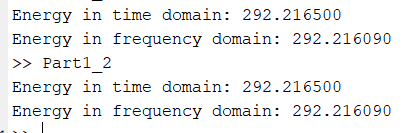


Fig. 5: Energy calculations

Parsevals relation as follows:

In Fig. 5, you can see this equation approximately holds computationally as well. Corresponding code part as follows:

% Define the signals

t1 = linspace(0, 1, 1000); % Time vector for x1(t)

t2 = linspace(-1, 1, 2000); % Time vector for x2(t)

Ts = 0.001; % Sampling period

% Signal x1(t)

x1\_t = 8\*cos(10\*pi\*t1) + 20\*sin(6\*pi\*t1) - 11\*cos(30\*pi\*t1);

% Signal x2(t)

x2\_t = exp(-t2);

% a) Fourier series coefficients for x1(t)

% Coefficients calculation manually

a1\_1 = 8/2; % Coefficient for 10\*pi\*t

a1\_2 = 20/2i; % Coefficient for 6\*pi\*t

a1\_3 = -11/2; % Coefficient for 30\*pi\*t

% b) Estimating Fourier series coefficients using FSAnalysis

% For x1(t)

fsCoeffs\_x1 = FSAnalysis(x1\_t, 30);

% For x2(t)

fsCoeffs\_x2 = FSAnalysis(x2\_t, 30);

% Plotting real and imaginary parts of estimated coefficients for x1(t)

figure;

subplot(2, 1, 1);

stem(-30:30, real(fsCoeffs\_x1));

title('Real Part of Estimated Fourier Series Coefficients for x1(t)');

xlabel('k');

ylabel('Real part');

grid on;

subplot(2, 1, 2);

stem(-30:30, imag(fsCoeffs\_x1));

title('Imaginary Part of Estimated Fourier Series Coefficients for x1(t)');

xlabel('k');

ylabel('Imaginary part');

grid on;

figure;

subplot(2, 1, 1);

stem(-30:30, real(fsCoeffs\_x2));

title('Real Part of Estimated Fourier Series Coefficients for x2(t)');

xlabel('k');

ylabel('Real part');

grid on;

subplot(2, 1, 2);

stem(-30:30, imag(fsCoeffs\_x2));

title('Imaginary Part of Estimated Fourier Series Coefficients for x2(t)');

xlabel('k');

ylabel('Imaginary part');

grid on;

energy\_time\_domain = sum(abs(x1\_t).^2) \* Ts; % Multiply by Ts for discrete case

energy\_freq\_domain = sum(abs(fsCoeffs\_x1).^2);

fprintf('Energy in time domain: %f\n', energy\_time\_domain);

fprintf('Energy in frequency domain: %f\n', energy\_freq\_domain);

* Part 2:

In this part, I handled and its different versions, such as shifted version, time-reversed version etc. is given as follows:

I found the fourier series coefficients in Fig. 6.

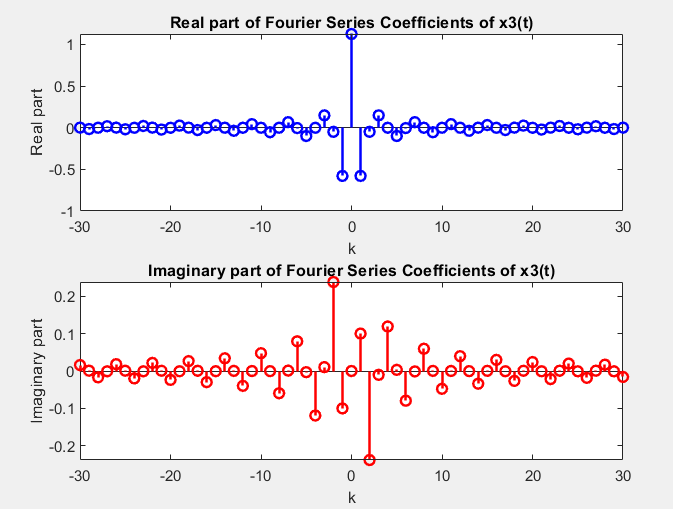


Fig. 6: Fourier series coefficients of

The other signals have same coefficients with some changes. signals coefficients are equal to negative indexed ’s coefficients since it is the time reversed version of it, i.e. . In Fig. 7, fourier series coefficients of is plotted which is consistent with the expected result.

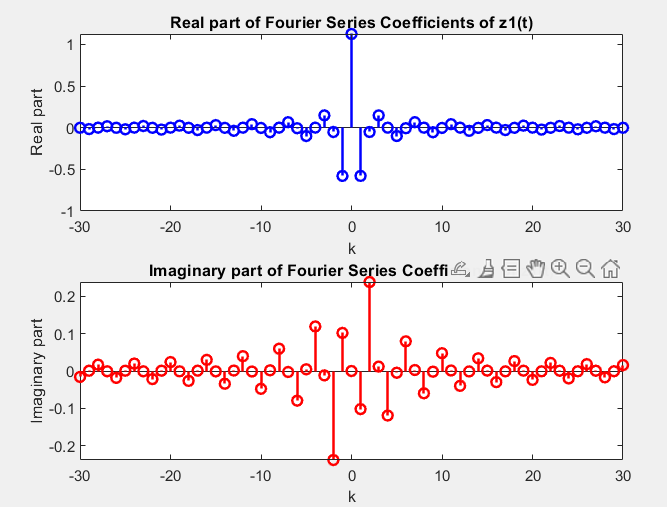


Fig. 7: Fourier series coefficients of

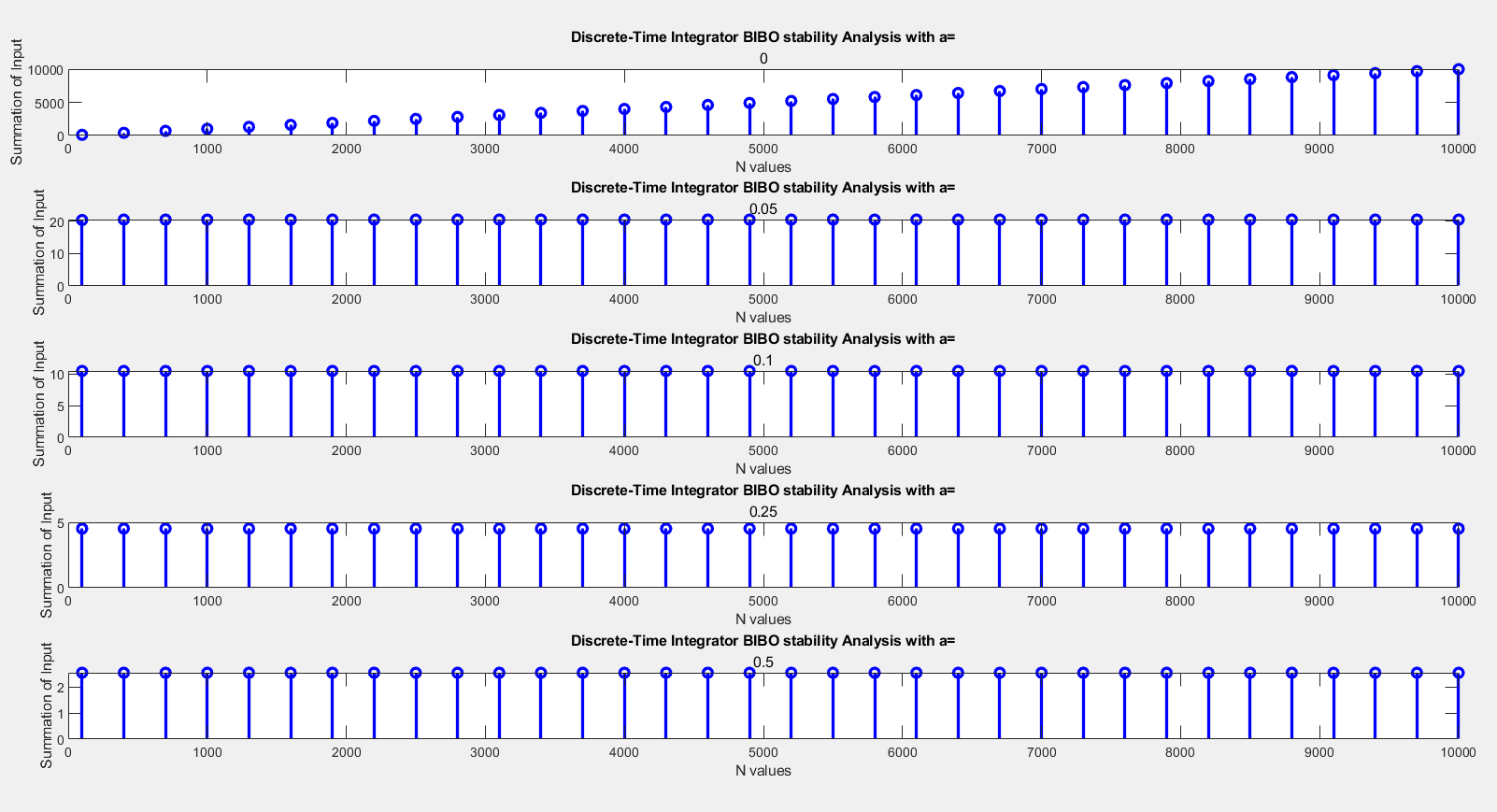


Fig. 3: BIBO stability results between 0 and N

Fig. 2 represents the BIBO stability of the sum of the exponential function with impulse response with negative n-values included and Fig. 3 represents the BIBO stability of the sum of the exponential function with impulse response without negative n-values. As expected, exponentially decayed Integrator has BIBO stability while Ideal Integrator does not satisfy this property. Results are compatible with Part 1.

Corresponding codes presented below:

function [sum\_array] = sumElements(h, N\_range)

ind=(length(h)+1)/2;

sum\_array = zeros(size(N\_range));

for i = 1:length(N\_range)

N = N\_range(i);

sliced\_arr = h(ind-N:ind+N);

sum\_array(i) = sum(abs(sliced\_arr)) ;

end

N\_range = 100:300:10000;

a\_val = [0,0.05,0.1,0.25,0.5];

output\_array = zeros(length(a\_val),length(N\_range));

n\_arr = -N\_range(length(N\_range)):N\_range(length(N\_range));

figure

for i = 1:length(a\_val)

imp\_resp = exp(-a\_val(i)\*n\_arr);

x = sumElements(imp\_resp,N\_range);

subplot(5,1,i)

stem(N\_range, x, 'b', 'LineWidth', 2);

xlabel('N values');

ylabel('Summation of Input');

title('Discrete-Time Integrator BIBO stability Analysis with a=',num2str(a\_val(i)));

end

x = sumElements(exp(-a\_val(3)\*n\_arr),N\_range);

* Part 3:

In this part, I investigate the previous systems for the given inputs and . I computed their ideal integration and stored them in and . Then I calculate the following to observe the differences between the ideal integrator and imperfect integrator (those with exponentially decay):

The following plots show the results:

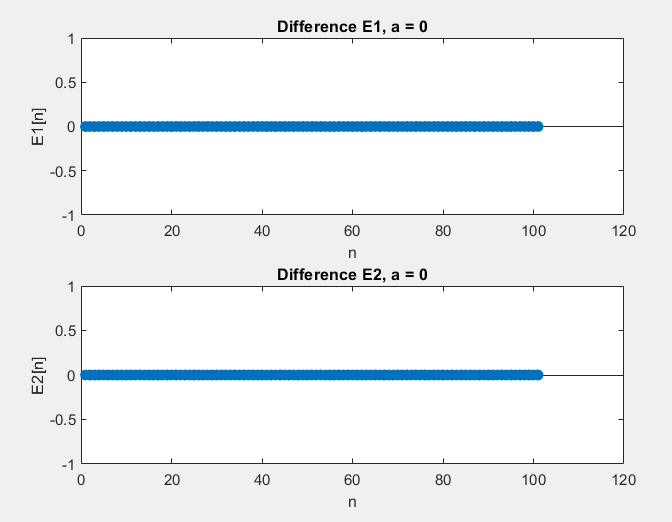
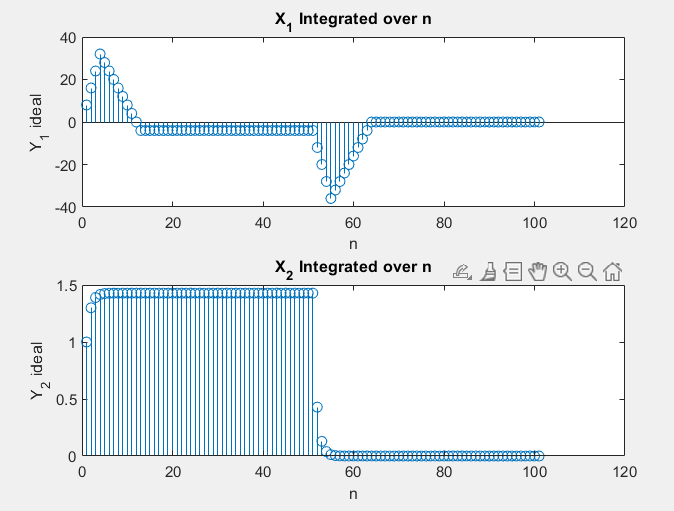


Fig. 4: and Fig. 5: and ,

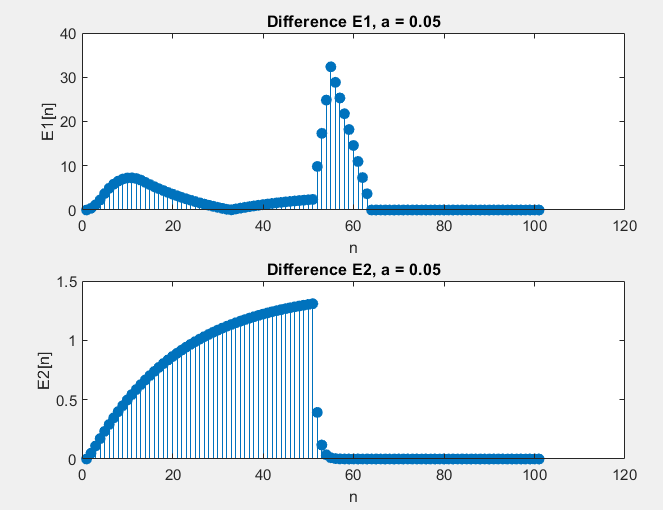
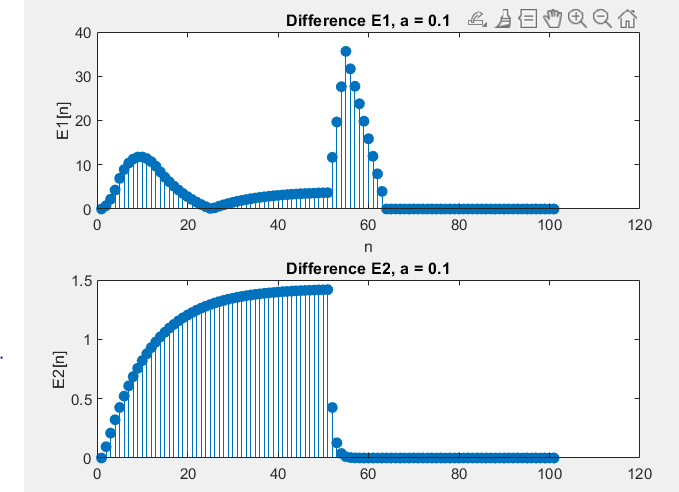
 

Fig. 6: and , Fig. 7: and ,

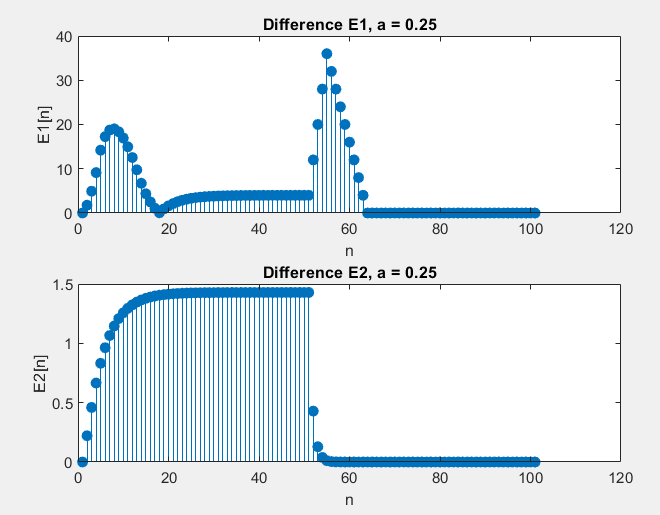
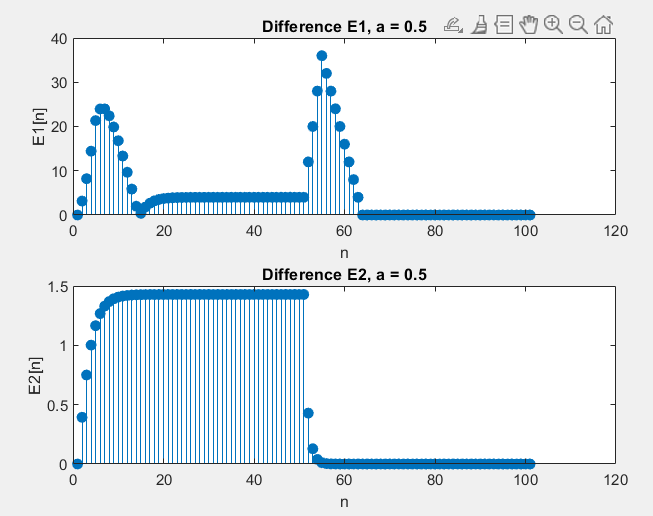
 

Fig. 8: and , Fig. 9: and ,

As seen from the figures, increased exponential decay rate reduces the sum of integration and ideal are affected slightly from the subtraction operation.

Following lines contain corresponding codes:

n = 0:50;

x1 = 8\*(double(n>=0)-double(n>=4)) - 4\*(double(n>=4)-double(n>=13));

x2 = 0.3.^n .\* double(n>=0);

a\_values = [0, 0.05, 0.1, 0.25, 0.5];

y1\_ideal = conv(x1,double(n>=0));

y2\_ideal = conv(x2,double(n>=0));

n\_new = 1:length(y1\_ideal);

figure;

subplot(2,1,1)

stem(n\_new,y1\_ideal)

title("X\_1 Integrated over n");

xlabel('n');

ylabel('Y\_1 ideal');

subplot(2,1,2)

stem(n\_new,y2\_ideal)

title("X\_2 Integrated over n");

xlabel('n');

ylabel('Y\_2 ideal');

y1 = zeros(length(a\_values), 2\*length(n)-1);

y2 = zeros(length(a\_values), 2\*length(n)-1);

for i = 1:length(a\_values)

a = a\_values(i);

exp\_current = exp(-a\*n); % Modify impulse response for current 'a'

y1(i, :) = conv(x1, exp\_current);

y2(i, :) = conv(x2, exp\_current);

end

E1 = zeros(size(y1));

E2 = zeros(size(y2));

for i = 1:length(a\_values)

E1(i, :) = abs(y1\_ideal - y1(i, :));

E2(i, :) = abs(y2\_ideal - y2(i, :));

end

for i = 1:length(a\_values)

figure;

subplot(2, 1, 1);

stem(n\_new, E1(i, :), 'filled');

title(['Difference E1, a = ' num2str(a\_values(i))]);

xlabel('n');

ylabel('E1[n]');

subplot(2, 1, 2);

stem(n\_new, E2(i, :), 'filled');

title(['Difference E2, a = ' num2str(a\_values(i))]);

xlabel('n');

ylabel('E2[n]');

end

* Part 4:

1. **First and Second Order Differentiation:**

We are given the following equation for the first order derivative:

For the second order approximation, we can use the same approximation as well:

The system is casual since does not contain any future input values. The system has memory since the second order derivative approximation depends on the previous values of the input. Also system has BIBO stability as well since there isn’t any way to obtain unbounded output in this system with three bounded inputs. The system has a finite impulse response (FIR) because its output is only determined by a finite number of previous input values like BIBO stability analysis. The impulse response is this system is:

Remaining terms are zero since they do not contain the term. Following Matlab figure verifies the obtained results.

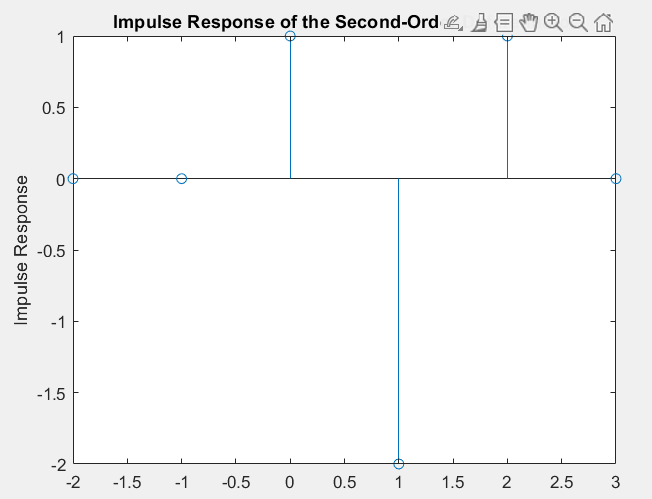


Fig. 10: Impulse response of the system

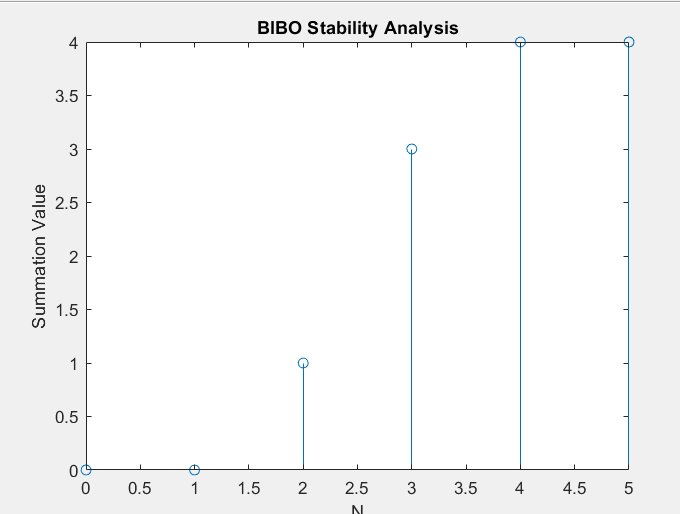


Fig. 11: BIBO stability analysis plot

To investigate BIBO stability of the system in Matlab, I used **sumElements()** function that is written in Part 2. Obtained plot that is presented in Fig. 11 proves that the system has BIBO stability. With a finite impulse it gives a finite output.

After that, I tested the system with the signals and that are given in Part 3. Their second order derivative approximations are as in Fig. 12.

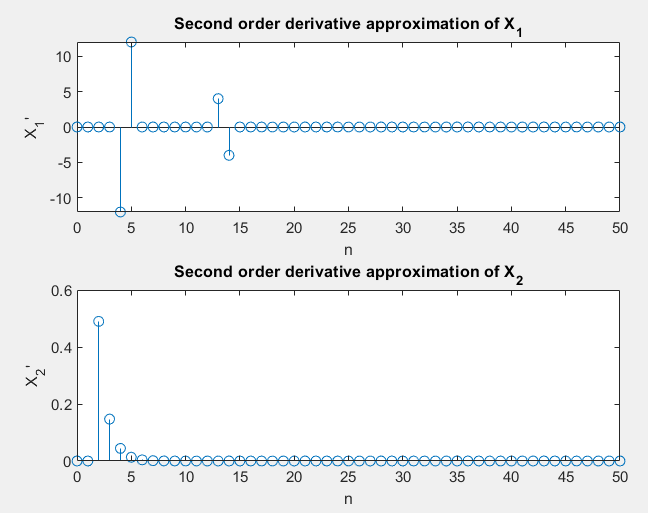


Fig. 12: Second order derivative approximations of and

1. **Invertibility of Second Order Difference:**

The original system is linear and time-invariant. Hence, it satisfies the necessary rules for the invertibility. The following figure shows the derivation of the inverse system and inverse system itself.

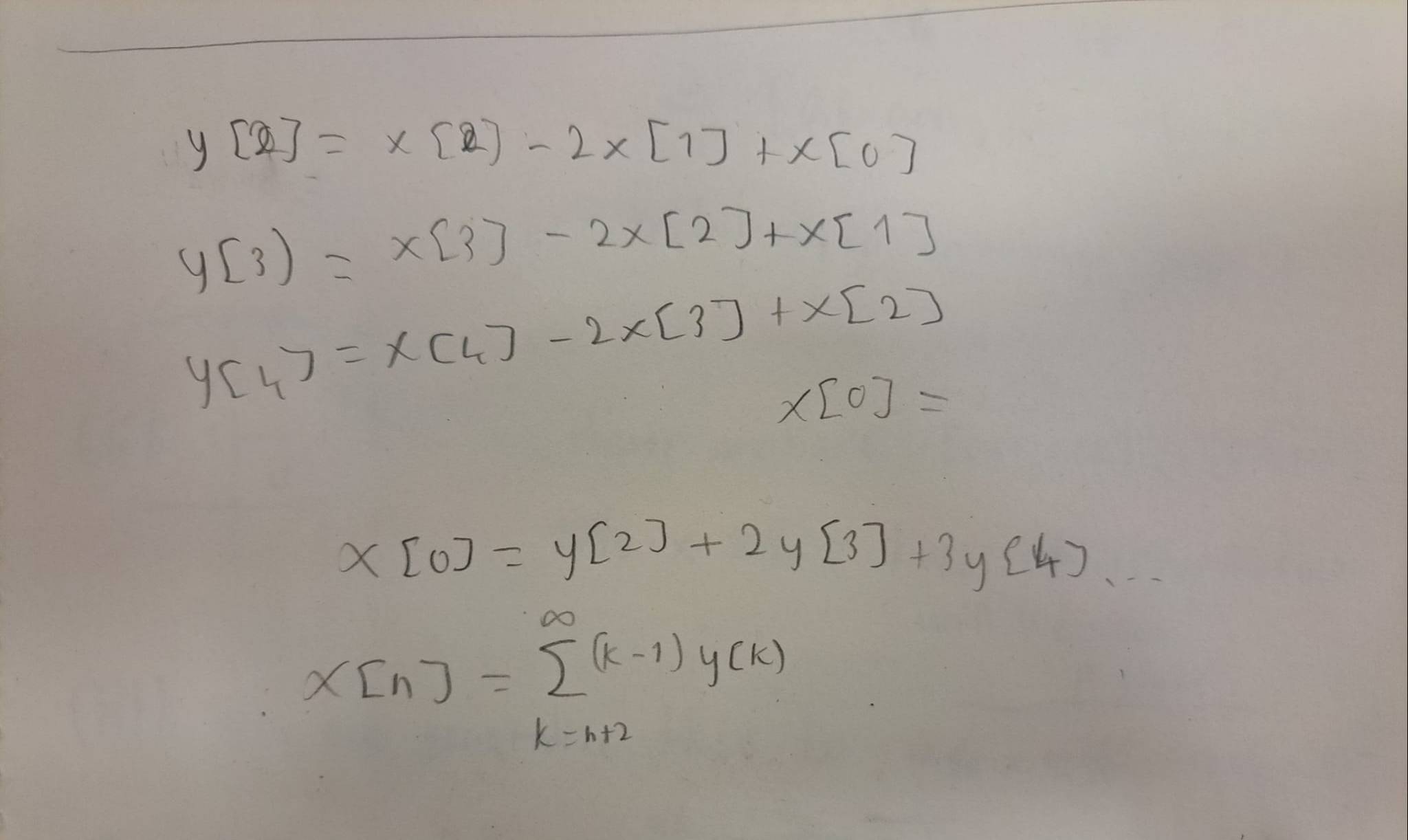


Fig. 13: Inverse system of the second-order difference equation system

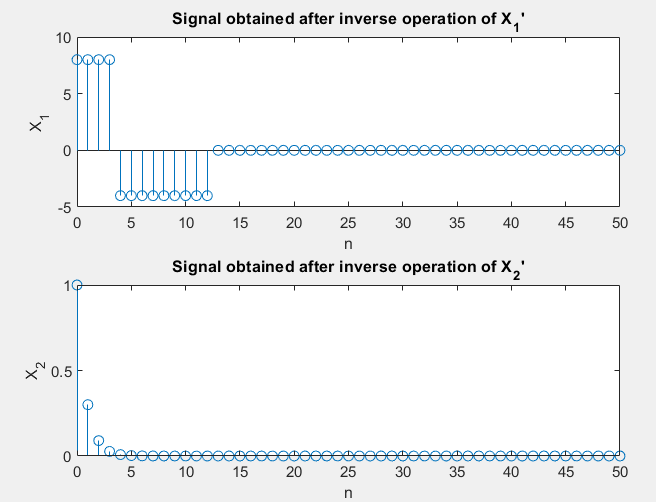


Fig. 14: Original signal obtained from the second order approximations

The system is not casual since the current term depends on the future input values. However, it doesn’t have memory since past terms do not appear in the current term’s expression.

I couldn’t recognize what is this system. However, its dependence on future terms seems interesting. Following two figures (Fig.15 and Fig. 16) shows the convolution of these two systems and obtained impulse response. Hence, I can say that this is the inverse system.

One can use the following codes for testing the procedure and results:

function [h\_ret] = Inverse\_2ndOrd(h)

array = (0:length(h)-1)-1;

h\_inter = h.\*array;

h\_ret = zeros(1,length(h\_inter));

for i = 3:length(h\_ret)

h\_ret(i-2) = sum(h\_inter(i:length(h\_inter)));

array = array - 1;

h\_inter = h.\*array;

end

end

h = [zeros(1,2),1, zeros(1, 50)];

hsec\_order = zeros(size(h));

for n = 3:length(h)

hsec\_order(n) = h(n) - 2\*h(n-1) + h(n-2);

end

% Plot impulse response

stem(-2:3, hsec\_order(1:6));

xlabel('n');

ylabel('Impulse Response');

title('Impulse Response of the Second-Order Differentiator');

% Investigate BIBO stability

figure;

N\_range = 0:5;

sum\_array = sumElements([zeros(1,length(hsec\_order)-1),hsec\_order], N\_range);

stem(N\_range, sum\_array);

xlabel('N');

ylabel('Summation Value');

title('BIBO Stability Analysis');

n = 0:50;

x1 = 8\*(double(n>=0)-double(n>=4)) - 4\*(double(n>=4)-double(n>=13));

x2 = 0.3.^n .\* double(n>=0);

x1sec\_order = zeros(size(x1));

x2sec\_order = zeros(size(x2));

for n = 3:length(x1)

x1sec\_order(n) = x1(n) - 2\*x1(n-1) + x1(n-2);

x2sec\_order(n) = x2(n) - 2\*x2(n-1) + x2(n-2);

end

figure

subplot(2,1,1)

stem(0:length(x1sec\_order)-1, x1sec\_order);

xlabel('n');

ylabel('X\_1''');

title('Second order derivative approximation of X\_1');

subplot(2,1,2)

stem(0:length(x2sec\_order)-1, x2sec\_order);

xlabel('n');

ylabel('X\_2''');

title('Second order derivative approximation of X\_2');

x1\_inversed = Inverse\_2ndOrd(x1sec\_order);

figure;

subplot(2,1,1)

stem(0:length(x1\_inversed)-1, x1\_inversed);

xlabel('n');

ylabel('X\_1');

title('Signal obtained after inverse operation of X\_1''');

x2\_inversed = Inverse\_2ndOrd(x2sec\_order);

subplot(2,1,2)

stem(0:length(x2\_inversed)-1, x2\_inversed);

xlabel('n');

ylabel('X\_2');

title('Signal obtained after inverse operation of X\_2''');

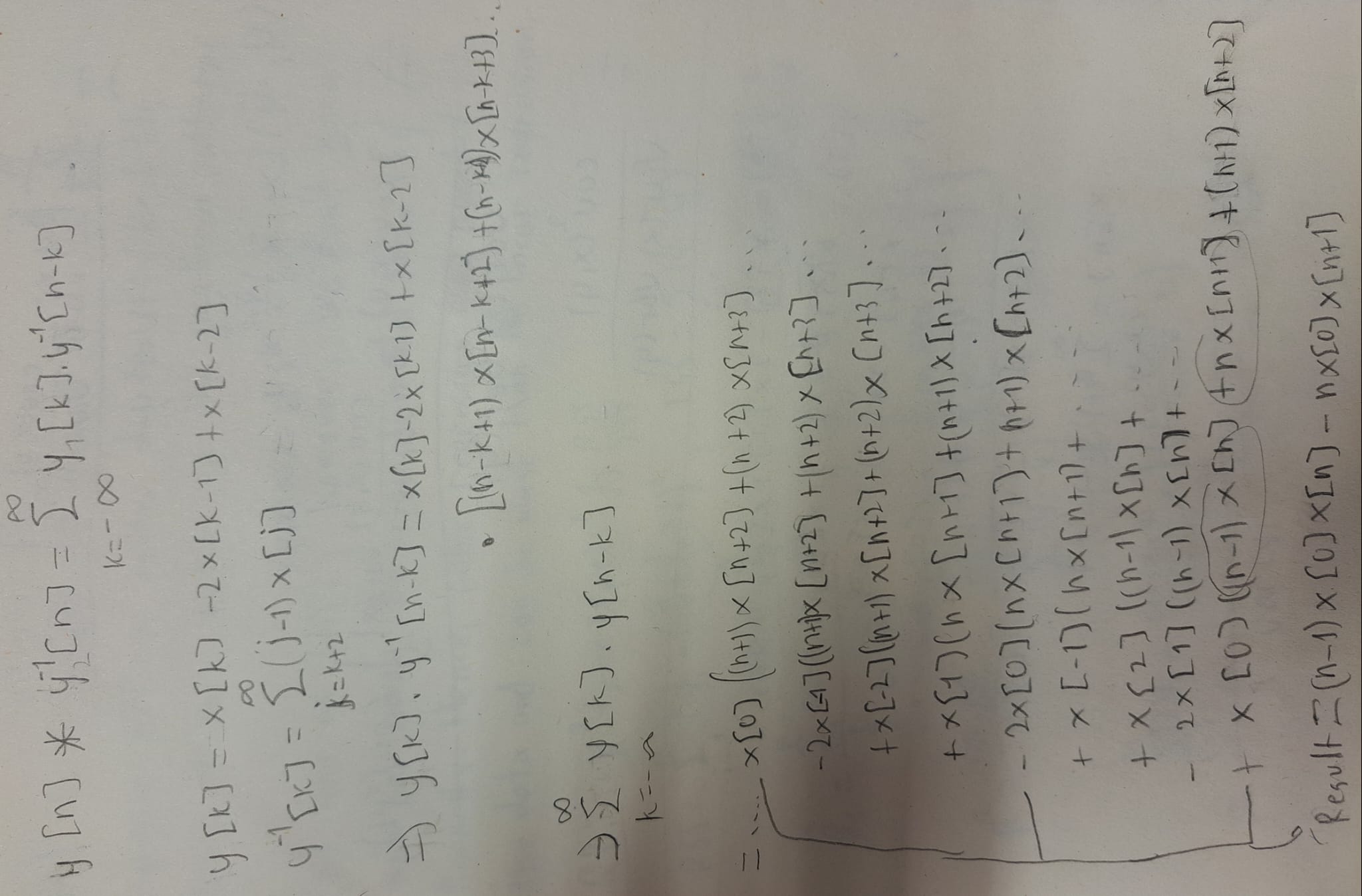


Fig. 15: Hand derivation of convolution

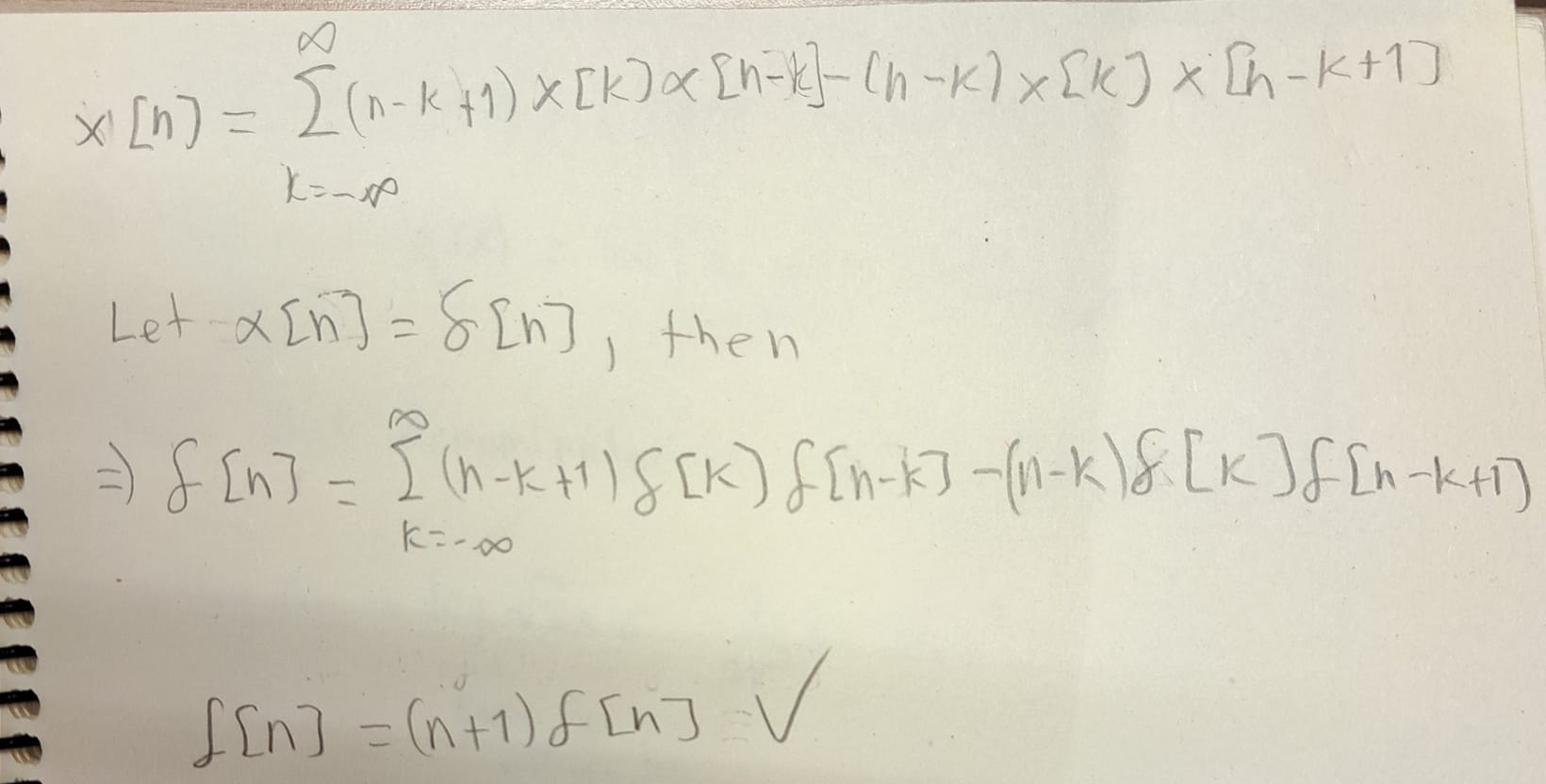


Fig. 16: Hand derivation of convolution