Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Signal and Systems II
Course Number:	ELE 632
Semester/Year (e.g.F2016)	Winter 2021
Instructor:	Dr. Dimitri Androustos (TA : Mahdi Shamsi)

Assignment/Lab Number:	01
Assignment/Lab Title:	Time-Domain Analysis of Discrete-Time Systems - Part 1

Submission Date:	Jan. 31, 2021
Due Date:	Jan. 31, 2021 (according to D2l)

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
ALAM	ABRAR	500725366	11	Abrar Alam
BHATT	SAGAR	500837713	10	Sagar

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Table of Contents

Abstract

In Part A, the goal was to highlight both similarities and differences between continuous and discrete time signal transformation operations. Continuous and discrete time signals are similar in cases of time shifting, and time reversal operations, but can have quite different characteristics when it comes to certain types of scaling such as time expansion. Besides, this part illustrates that to eliminate signal loss and errors, it is better to time expand the continuous time signal first, then sample that signal in discrete time domain.

Part B focuses on translating a real-world financial problem into a first order difference equation. Just like continuous time differential equations, zero-input response, zero-state responses, and total system response were calculated using an initial condition given.

In Part C, the goal was to design a causal N-point maximum filter. “Causal” implies that the filter looks at the present and past points, but not future points. “N-point” implies that a number N is supplied, which is the number of points that need to be looked at. “Maximum” implies that the maximum needs to be found among these values. Putting all this together, the goal was to design a filter that looks at N points - the present point and N-1 points directly prior to it - and find the maximum of these points, and assign it to N. This was done using a MATLAB function that

In Part D, the energy is the sum of squares of the absolute values of each point, and for a finite-length vector the power is energy divided by the length of the period of the vector. The following are the equations for the energy and power:

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

$$P_x = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{-N}^{N} |x[n]|^2$$

Since the vector has a finite length, the limit does not affect the value of the power and hence it is equal to the energy divided by the length of the vector.

Part A

Part 1

In this part, time shifting, and time reversal operations on discrete time signals have been examined. No difference between the applications of these transformations continuous signals, and discrete signals were observed. **Figure 1** and **figure 2** illustrate these results below.

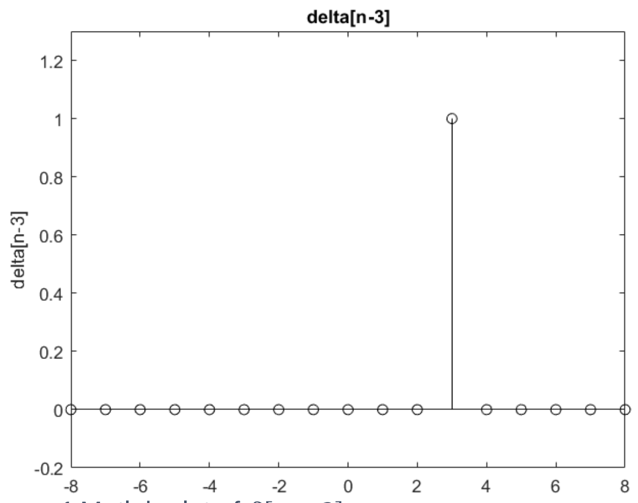


*Figure 1:*Matlab plot of $\delta[n-3]$

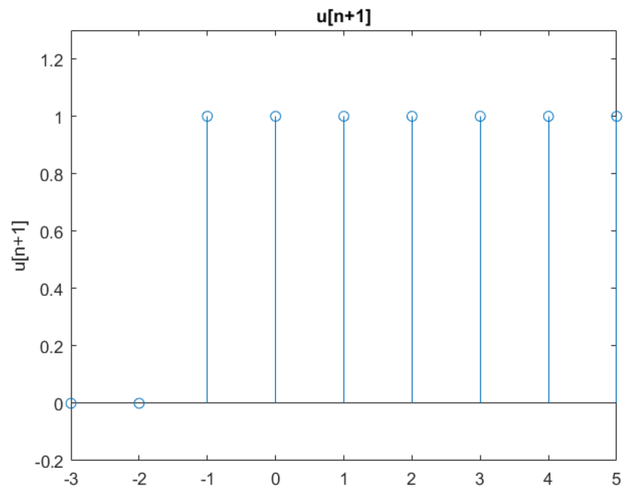


Figure 2 Matlab plot of u[n+1]

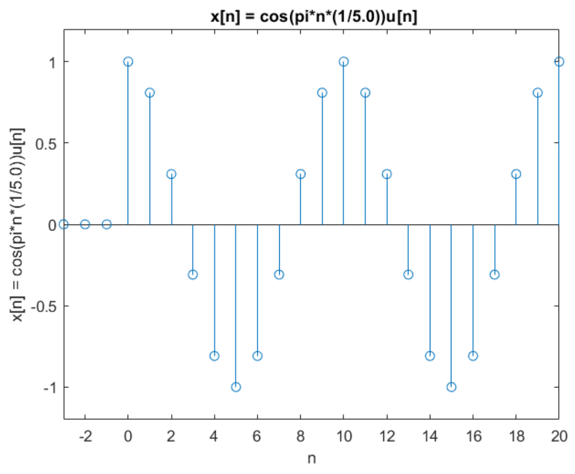


Figure 3 Matlab plot of $x[n] = \ cos\left(\frac{\pi n}{5}\right) u[n]$

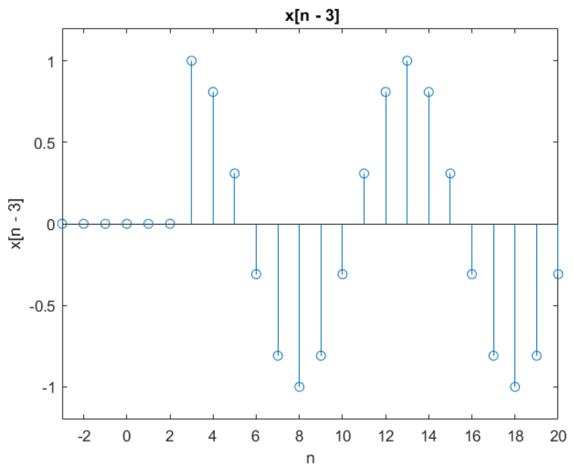


Figure 4 plot of $x_1[n] = x[n-3]$

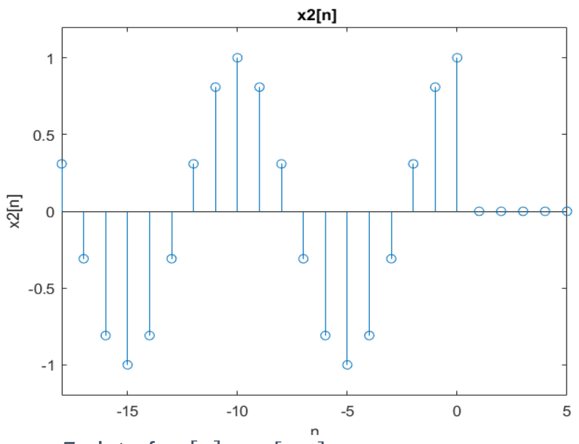


Figure 5 plot of $x_2[n] = x[-n]$

In $x_1[n]$, time shifting property has been applied, while in $x_2[n]$, time reversal property has been applied, these results have been shown in **figure 4** and **5**, respectively.

Part 2

In this part, the effects of scaling on a discrete time signal $y[n] = 5e^{-\frac{n}{8}}(u[n] - u[n-10])$ has been examined, and the results were plotted in **Figure 6** to **8** (in the time interval [-10, 70]).

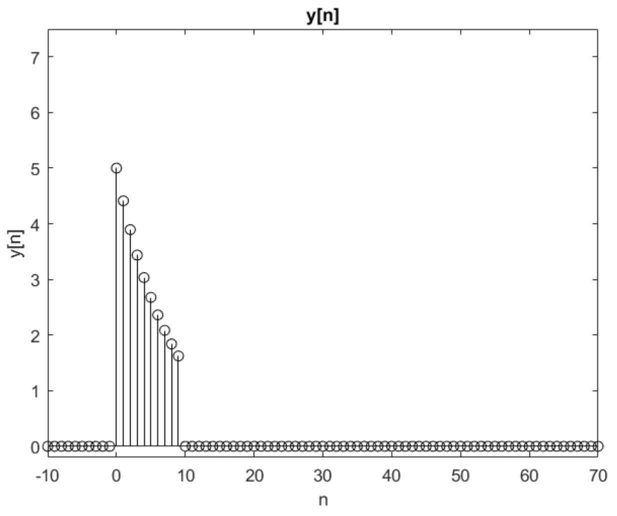


Figure 6 $Plot\ of\ y[n] = 5e^{-\frac{n}{8}}(u[n] - u[n-10])$

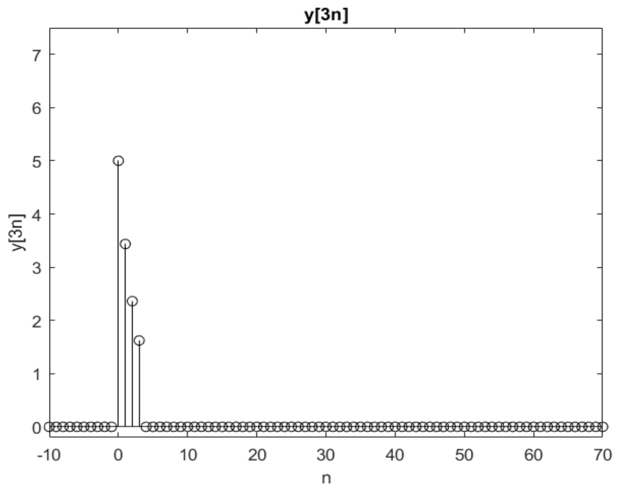


Figure 7 $Plot\ of\ y_1[n] = y[3n]$

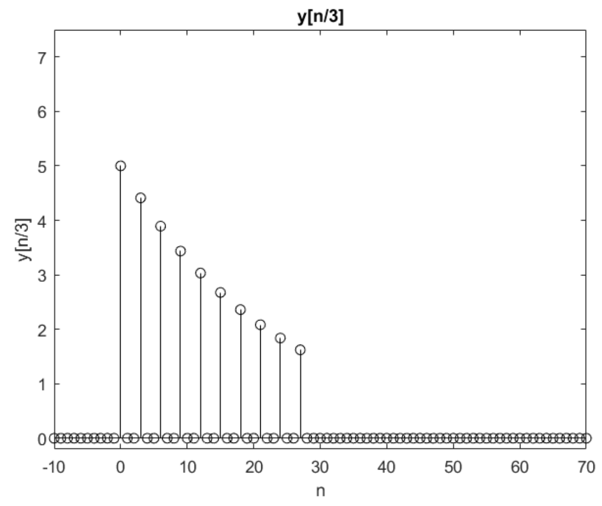


Figure 8: $y_2[n] = y\left[\frac{n}{3}\right]$. Please note the padded zeros in corresponding ns which are not divisible by 3.

From the plot of $y_1[n] = y[3n]$ in **figure 7**, we see that application of time compression will cause the sampling to occur in n = 3K, where K ∈ $Integer$. This will result in the decimation of every (3-1) = 2 samples of the original signal $y[n]$. On the other hand, in **figure 8**, we see that the application of time expansion $y_2[n] = y\left[\frac{n}{3}\right]$ will result in added zeros in whenever n/3 is not an integer. This makes logical sense because, we are up sampling from a discrete time signal, not a continuous time signal.

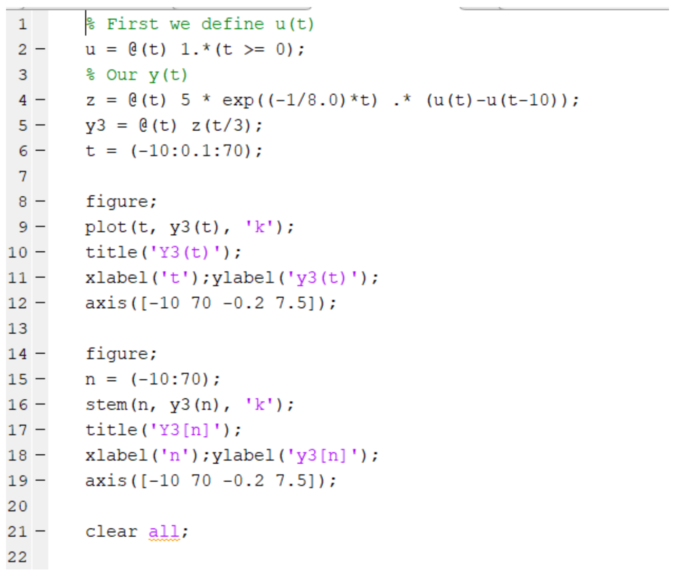
Part 3

In this part, we demonstrate that the sampling of a continuous signal and applying a linear transformation after will not always result in the same signal as if first applying the transformation and then executing sampling. For this we consider the continuous time signal $z(t)$

$$z(t) = 5e^{-\frac{t}{8}}\big(u(t) - u(t-10)\big), t = -10{:}\,0.1{:}\,70$$

Section 1

First we find $y_3(t) = z\left(\frac{t}{3}\right)$, and discretize this signal as $y_3[n]$ and then plot $y_3[n]$. **Figure 9** shows the Matlab code used for this part, and **Figure 10**, shows the plot of $y_3[n]$.



```
% First we define u(t)
u = @(t) 1.*(t >= 0);
% Our y(t)
z = @(t) 5 * exp((-1/8.0)*t) .* (u(t)-u(t-10));
y3 = @(t) z(t/3);
t = (-10:0.1:70);

figure;
plot(t, y3(t), 'k');
title('Y3(t)');
xlabel('t');ylabel('y3(t)');
axis([-10 70 -0.2 7.5]);

figure;
n = (-10:70);
stem(n, y3(n), 'k');
title('Y3[n]');
xlabel('n');ylabel('y3[n]');
axis([-10 70 -0.2 7.5]);

clear all;
```

Figure 9: Matlab code that discretizes $y_3[n]$ from $y_3(t) = z\left(\frac{t}{3}\right)$.

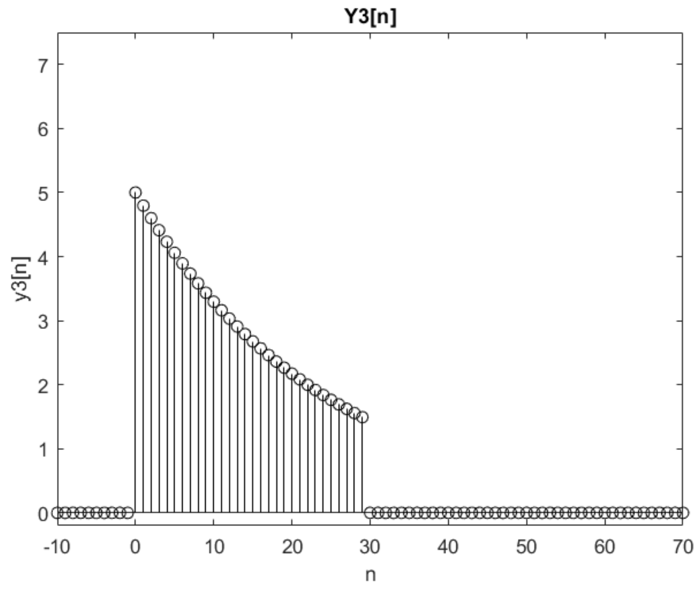


Figure 10: $y_3[n]$ discretized from $y_3(t) = z\left(\frac{t}{3}\right)$. Please note the absence of padded zeros in contrast with Figure 8

Section 2

Examination of **figure 10** reveals that this plot does not have any artificially inserted zeros in n which are not divisible by 3, which is to be expected because we sampled this signal from a continuous time signal $y_3(t) = z\left(\frac{t}{3}\right)$, and thus we had all the information on the values of $y_3[n]$

when n/3 is not an integer, unlike $y_2[n] = y\left[\frac{n}{3}\right]$ in **figure 8**. So $y_3[n]$ results in better up-sampling compared to $y_2[n]$.

Part B

Part 1

Here, to model the given scenario as a discrete time signal, we assume the customer's account balance at a given month (January to December, January being 1, February 2, and so on) is $y[n]$. However, we have $y[0] = \$2000$, as it's the customer's balance at the beginning of new year, that is before depositing any amount on January. So, $y[0] = \$2000$ is the initial condition. The amount the customer deposits per month is input, that is, $x[n]$. Here the interest rate, r is 2%. Now, the relationship between output $y[n]$, and input $x[n]$ is as follows:

$$y[n] = (1 + 0.02)y[n-1] + x[n], \text{ with } y[0] = \$2000$$

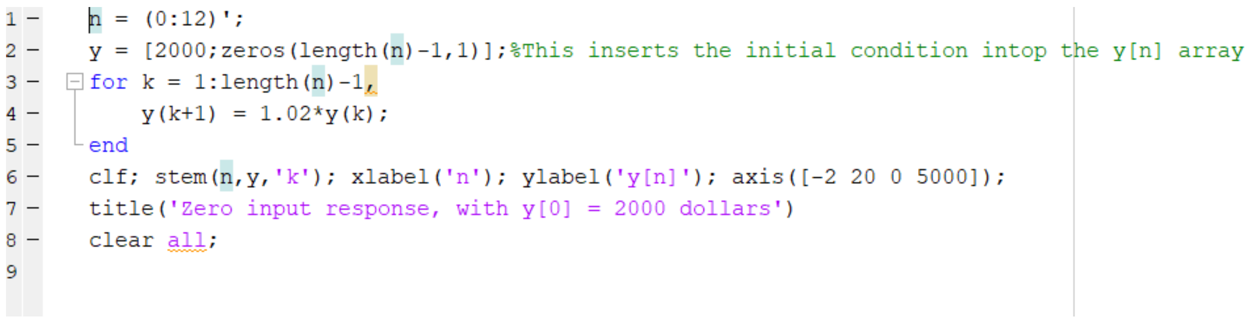
Part 2

Here we look for zero-input response, thus we set input $x[n]$ to be zero and look for system response with the initial condition specified in part1 above. So, our zero-input response equation will be:

$$y[n] = (1 + 0.02)y[n-1], \text{ with } y[0] = \$2000$$

The Matlab code of the recursive function that calculates and plots the zero input response is presented

in **figure 11**, and the resulting plot is presented in **figure 12**.



```
n = (0:12)';
y = [2000;zeros(length(n)-1,1)];%This inserts the initial condition intop the y[n] array
for k = 1:length(n)-1,
    y(k+1) = 1.02*y(k);
end
clf; stem(n,y,'k'); xlabel('n'); ylabel('y[n]'); axis([-2 20 0 5000]);
title('Zero input response, with y[0] = 2000 dollars')
clear all;
```

Figure 11 Matlab recursive function that calculates zero-input response.

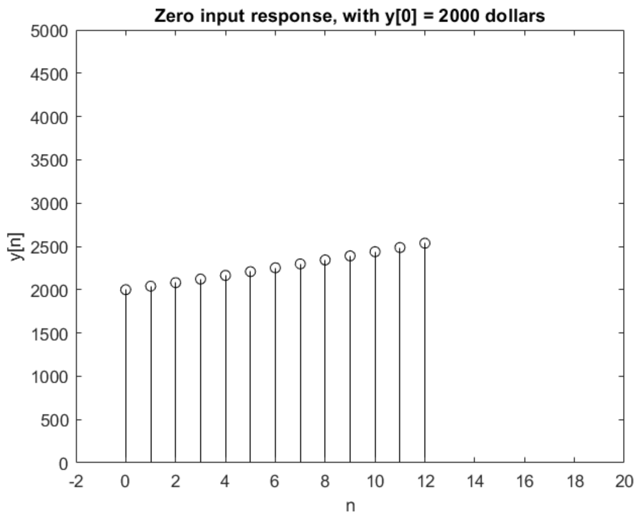
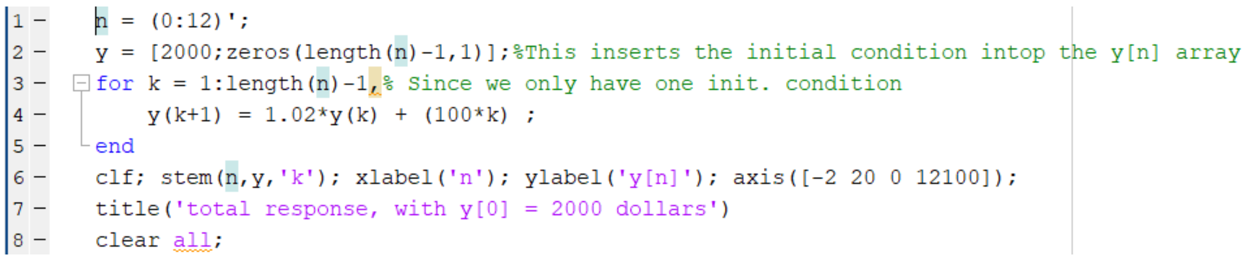


Figure 12 Matlab plot of the zero-input response.

Part 3

Now we calculate the total response of the system, with the input $x[n] = 100n$ dollars per month, where January is n = 1, February is n = 2, and so on. The matlab code, and the resulting plot is presented in **figure 13**, and **figure 14** respectively.



```
n = (0:12)';
y = [2000;zeros(length(n)-1,1)];%This inserts the initial condition intop the y[n] array
for k = 1:length(n)-1,% Since we only have one init. condition
    y(k+1) = 1.02*y(k) + (100*k) ;
end
clf; stem(n,y,'k'); xlabel('n'); ylabel('y[n]'); axis([-2 20 0 12100]);
title('total response, with y[0] = 2000 dollars')
clear all;
```

Figure 13 Matlab recursive function that calculates the total response.

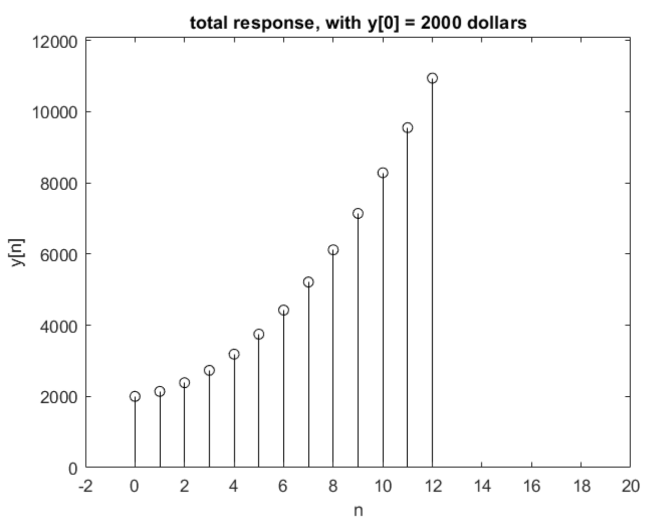


Figure 14 Matlab plot of the total response.

Part C

Part 1

The following is the MATLAB code for a maximum filter.

```
function [y] = partCpart1(x, N)
%Maximum Filtering
%   Finds the maximum of the signal among N points
%   (x[n]...x[n-(N-1)] and assigns it to output y[n]

x_extended = [zeros(1,N-1) x];

M = size(x, 2);

for n = M+N-1:-1:N
      x_extended(1,n) = max(x_extended(1,n-(N-1):n));
end

y = x_extended(1,N:M+N-1);

end
```

Part 2

Below is the code for creating the graphs.

```
% First we define u[n]
u = @(n) 1.*(n >= 0);
```

```
% Now define delta[n]
delta = @(n) u(n) - u(n - 1);
% Now we define x[n]
x = @(n) (cos(pi*n/5) + delta(n-20) + delta(n-35));

n = (0:45);
x_n = x(n);

figure(1);
stem(n, x_n);
title("Part C: Unmodified x[n] = cos(\pin/5)) + \delta(n-20) + \delta(n-35)
over 0≤n≤45");
xlabel("n"); ylabel("x[n]");
axis([-0.5 46 -1.5 2.5]);

y1 = partCpart1(x_n, 4);
y2 = partCpart1(x_n, 8);
y3 = partCpart1(x_n, 12);

figure(2);
stem(n, y1);
title("Part C: y_1[n] for x[n] with N = 4");
xlabel("n"); ylabel("y_1[n]");
axis([-1 46 -1.5 2.5]);

figure(3);
stem(n, y2);
title("Part C: y_2[n] for x[n] with N = 8");
xlabel("n"); ylabel("y_2[n]");
axis([-1 46 -1.5 2.5]);

figure(4)
stem(n, y3);
title("Part C: y_3[n] for x[n] with N = 12");
xlabel("n"); ylabel("y_3[n]");
axis([-1 46 -1.5 2.5]);
```

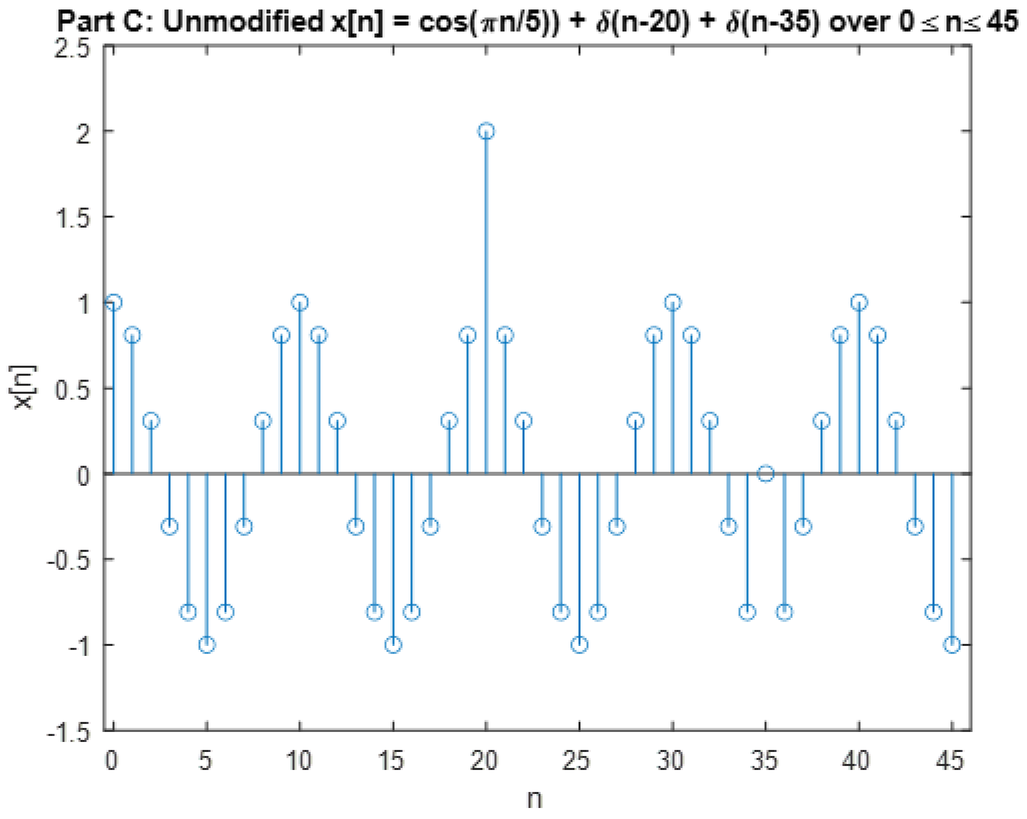


Figure 15 Part C: Unmodified x[n] = cos(πn/5)) + δ(n-20) + δ(n-35) over 0≤n≤45

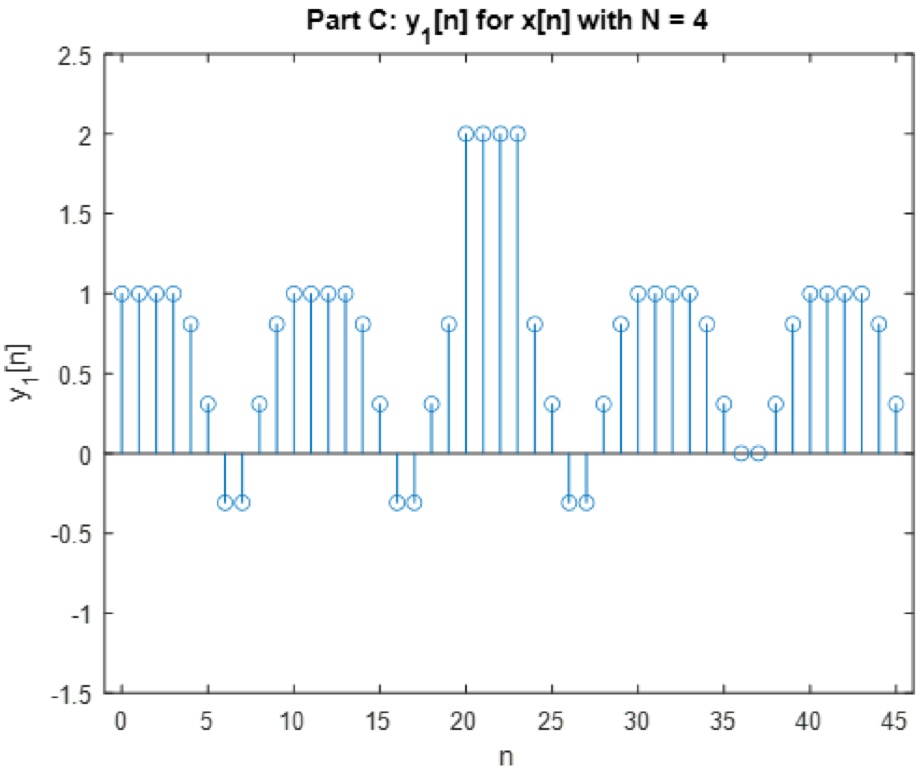


Figure 16 Part C: y_1[n] for x[n] with N = 4

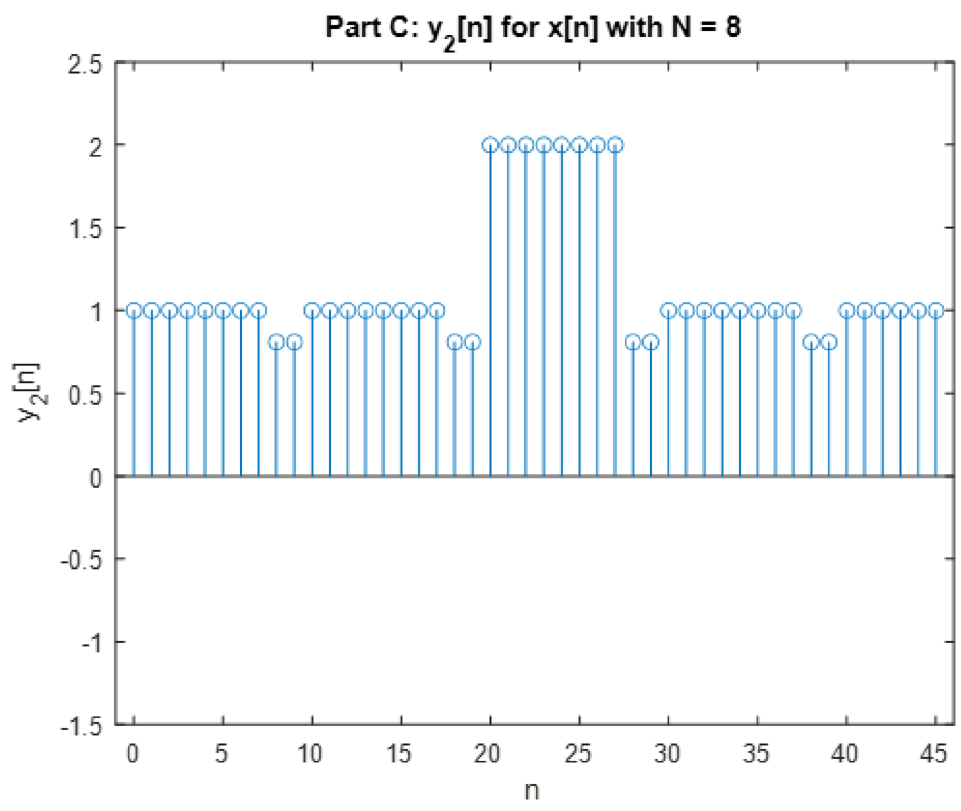


Figure 17 Part C: $y_2[n]$ for x[n] with N = 8

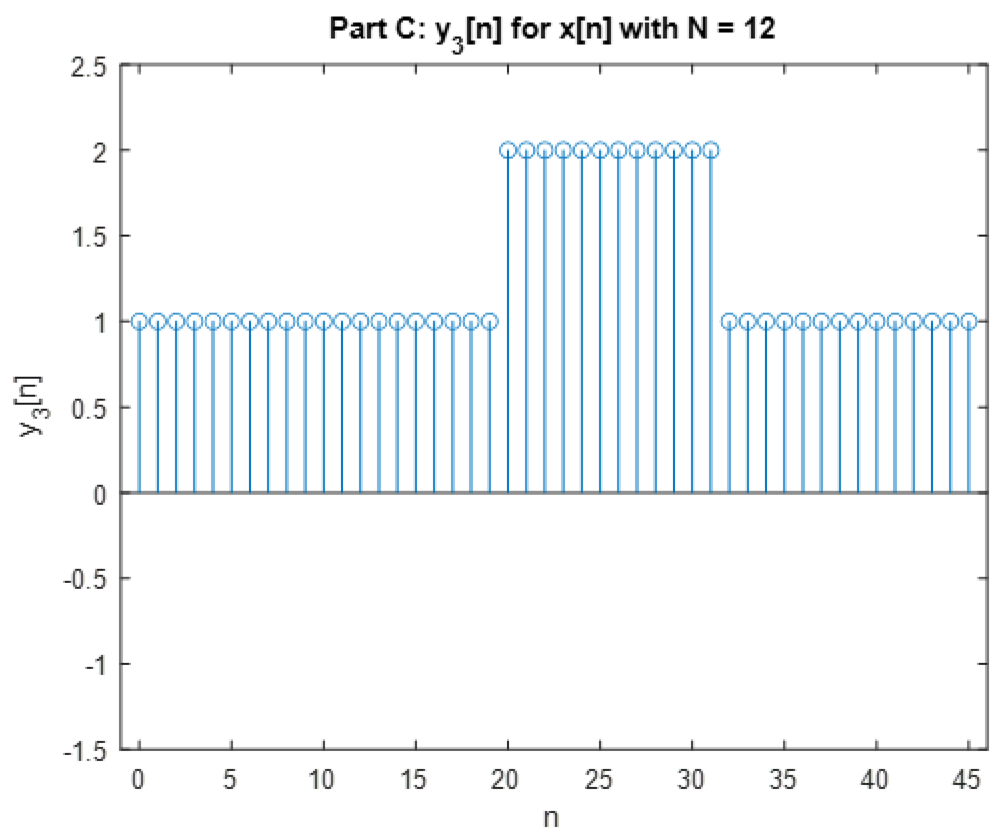


Figure 18 Part C: $y_3[n]$ for x[n] with N = 12

Part 3

As the value of N increases, each x[n] is more likely to be greater in magnitude because it is being compared to more values prior to it. Hence, the average magnitude increases, more values are filtered out, and there are fewer differences among values.

Part D

Part 1

Below is the code for outputting the energy and power of a finite-length vector:

```
function [E, P] = partDpart1(x)
%Maximum Filtering
%   Calculates power and energy of finite length vector x

E = sum(x.^2);

P = E/size(x,2);

end
```

Part 2

Below is the code for calculating the energy and power of the given graph:

```
x_n = linspace(-9,9,7);


% plot for verifying that the function matches the figure
n = linspace(-3,3,7);
stem(n,x_n);
xlabel('n'); ylabel('x[n]');


[E, P] = partDpart1(x_n);
```

The calculated values for Energy and Power are:
Energy = 252
Power = 36

This is the plot generated:

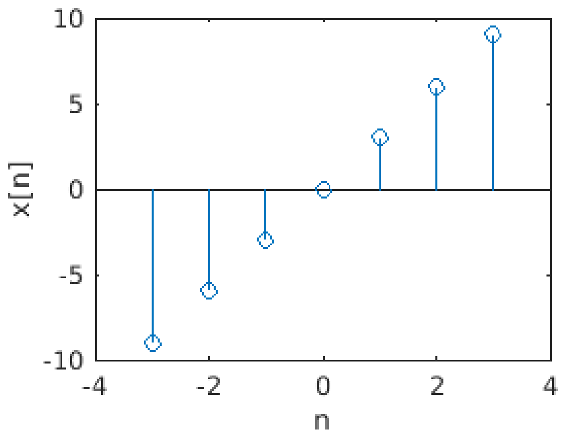


Figure 19 Part D part 2: plot of the figure