Ryerson University

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

Course Title:	ELE
Course Number:	532
Semester/Year (e.g.F2016)	F 2021

Instructor:	Fei Yuan

Assignment/Lab Number:	1
Assignment/Lab Title:	Working with MATLAB, Visualization of Signals

Submission Date :	Oct 3rd /2021
Due Date:	Oct 3rd /2021

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Youssef	Kirolos	500968175	01	K. Y
Miceli	Julian	500984182	01	J. M

*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

Objective

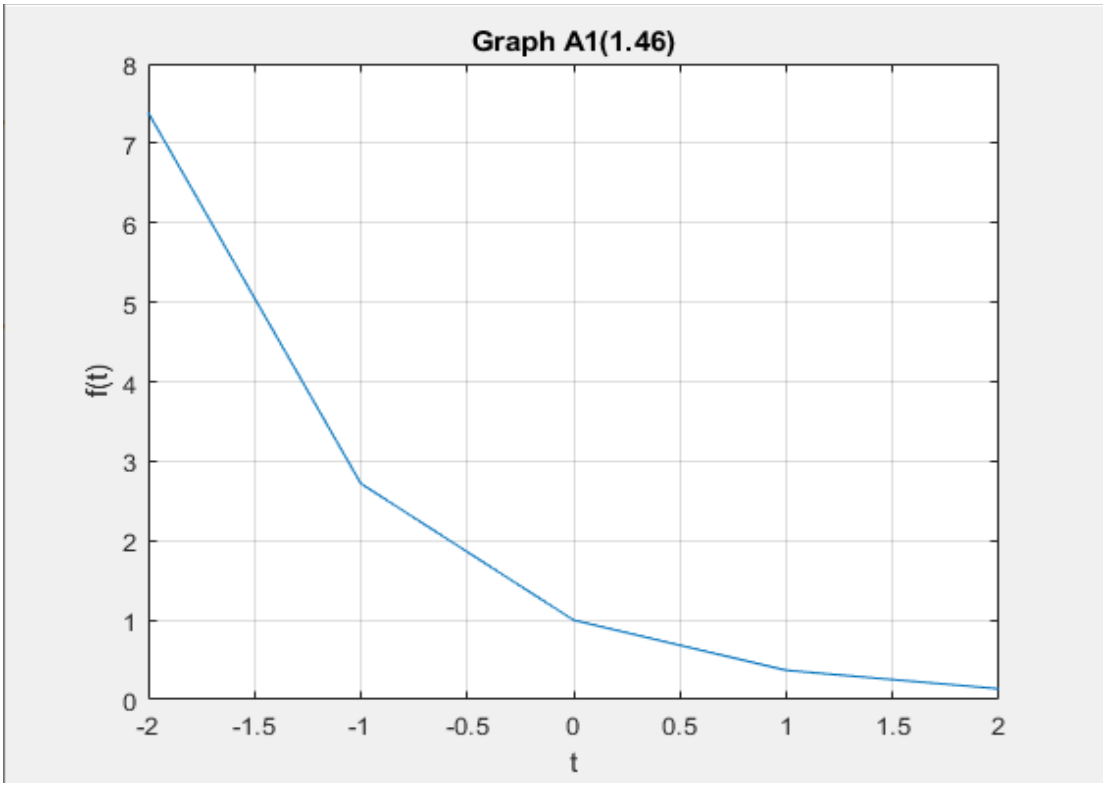
The objective of this lab is to refamiliarize ourselves with MATLAB, visualize different signal waveforms, and explore how compression algorithms work.

Experiment

A1:

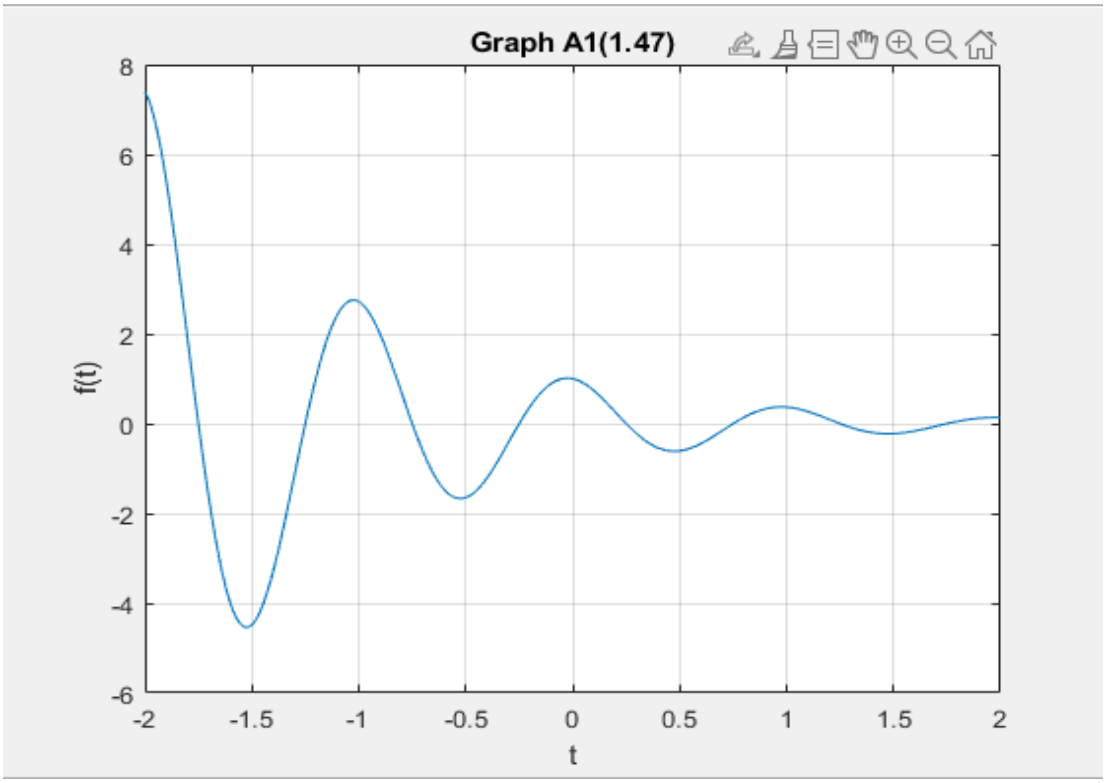
Matlab Code used to plot Graph A1(1.46):

```
f = @(t) exp(-t).*cos(2*pi*t);
t = (-2:2);
plot(t, f(t));
xlabel('t'); ylabel('f(t)'); grid; title("Graph A1(1.46)");
```



Matlab Code used to plot Graph A1(1.47):

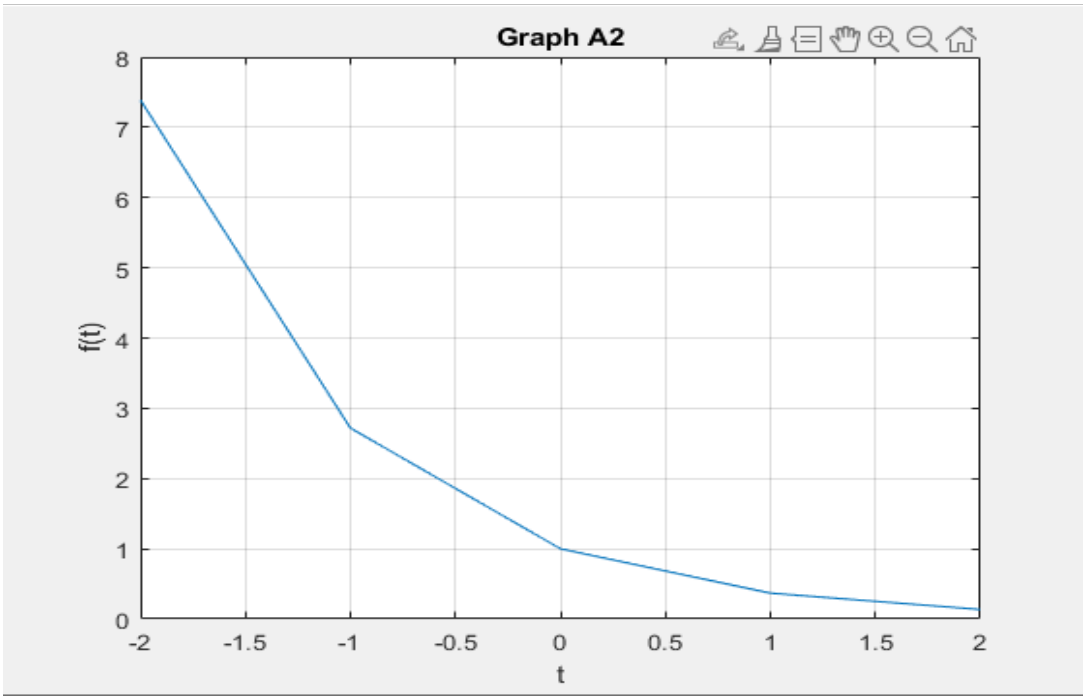
```
f = @(t) exp(-t).*cos(2*pi*t);
t = (-2:0.01:2);
plot(t, f(t));
xlabel('t'); ylabel('f(t)'); grid; title("Graph A1(1.47)");
```



A2:

Matlab Code used to plot Graph A2:

```
f = @(t) exp(-t);
t = (-2:2);
figure; plot(t, f(t));
xlabel('t'); ylabel('f(t)'); grid; title("Graph A2");
```



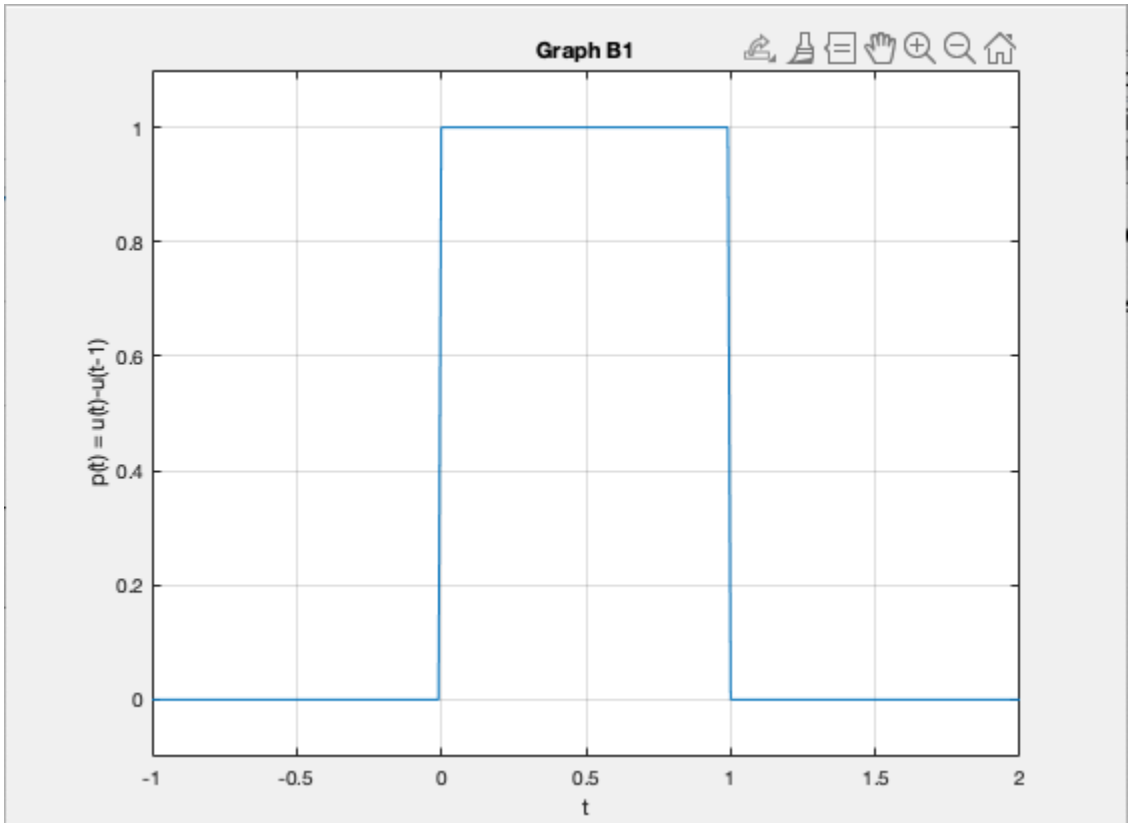
A3:

After comparing Graph 1.46 from A1 to Graph A2, it was observed that both the graphs are identical. Both graphs were observed to decrease exponentially, and follow the same nature in differentiating IROC's at specific ranges of t.

B1:

Matlab Code used to plot Graph B1 (1.50):

```
p = @(t) 1.0.*((t>=0)&(t<1));
t = (-1:0.01:2); plot(t,p(t));
xlabel('t'); ylabel('p(t) = u(t)-u(t-1)');
grid; title ("Graph B1");
axis([-1 2 -.1 1.1]);
```



B2:

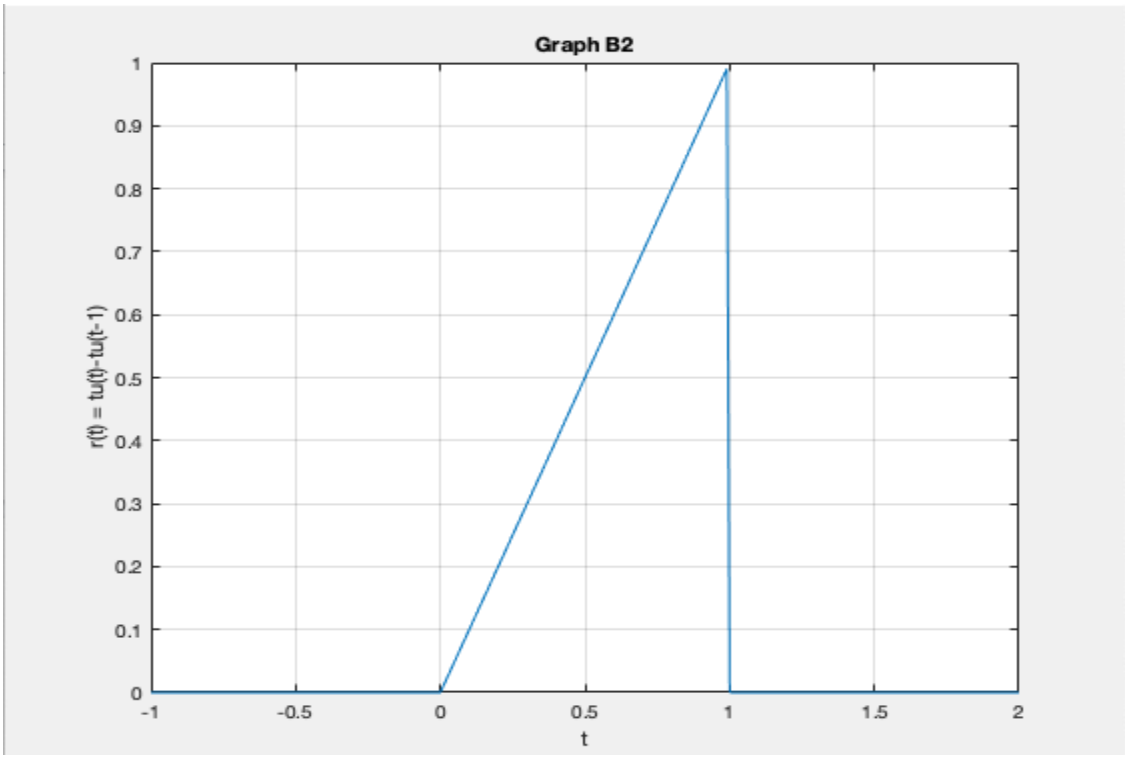
Matlab Code used to plot Graph B2:

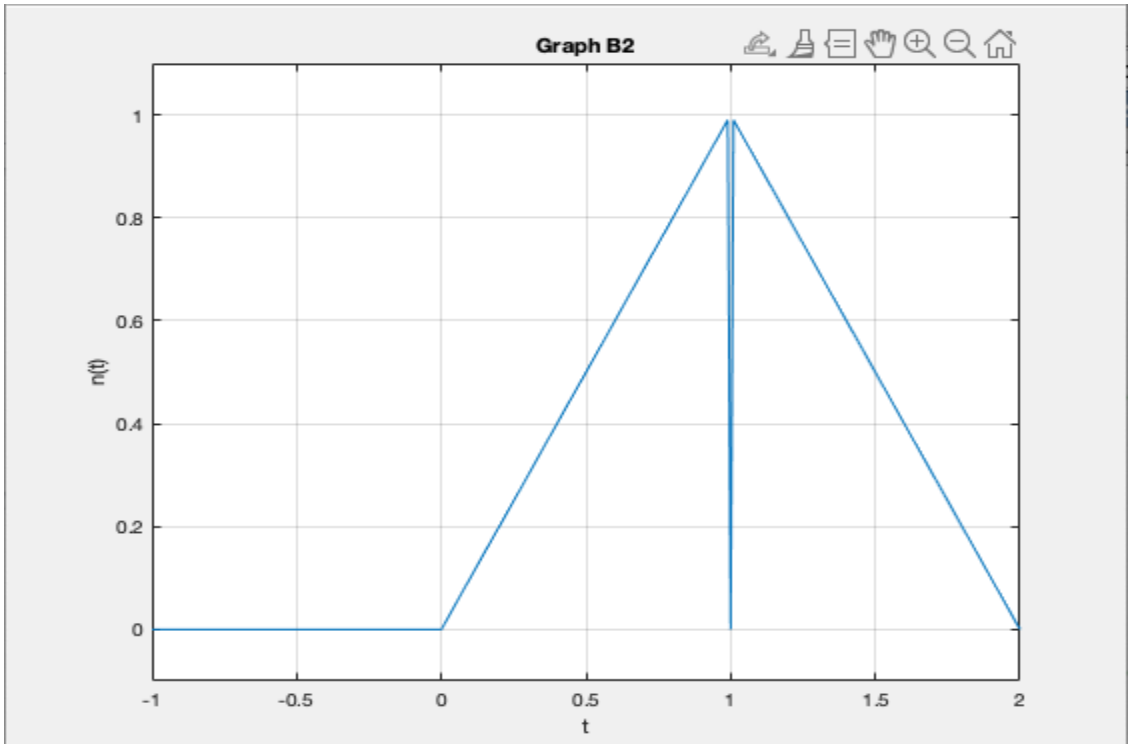
```
r = @(t) t*1.0.*((t>=0)&(t<1));
t = (-1:0.01:2);
plot(t,r(t));
xlabel('t'); ylabel('r(t) = tu(t)-tu(t-1)');
grid; title ("Graph B2"); figure;
axis([-1 2 -.1 1.1]);

% hold on

n = @(t) r(t)+ r(-t+2);
t = (-1:0.01:2);
plot(t,n(t));
xlabel('t'); ylabel('n(t)');
axis([-1 2 -.1 1.1]);
grid; title ("Graph B2");

% hold off
```




Graph B2
1
0.8
0.6
0.4
0.2
0
n(t)
-1
-0.5
0
0.5
1
1.5
2
t

B3:

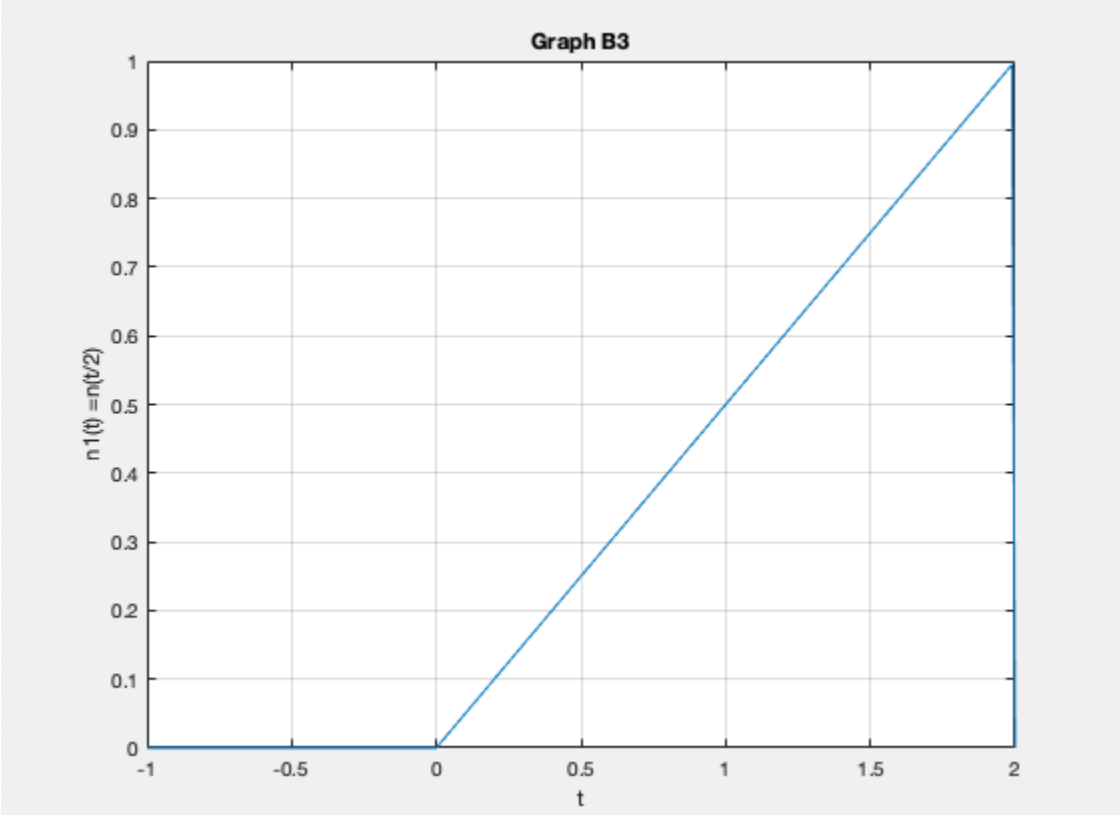
Matlab Code used to plot Graph B3:

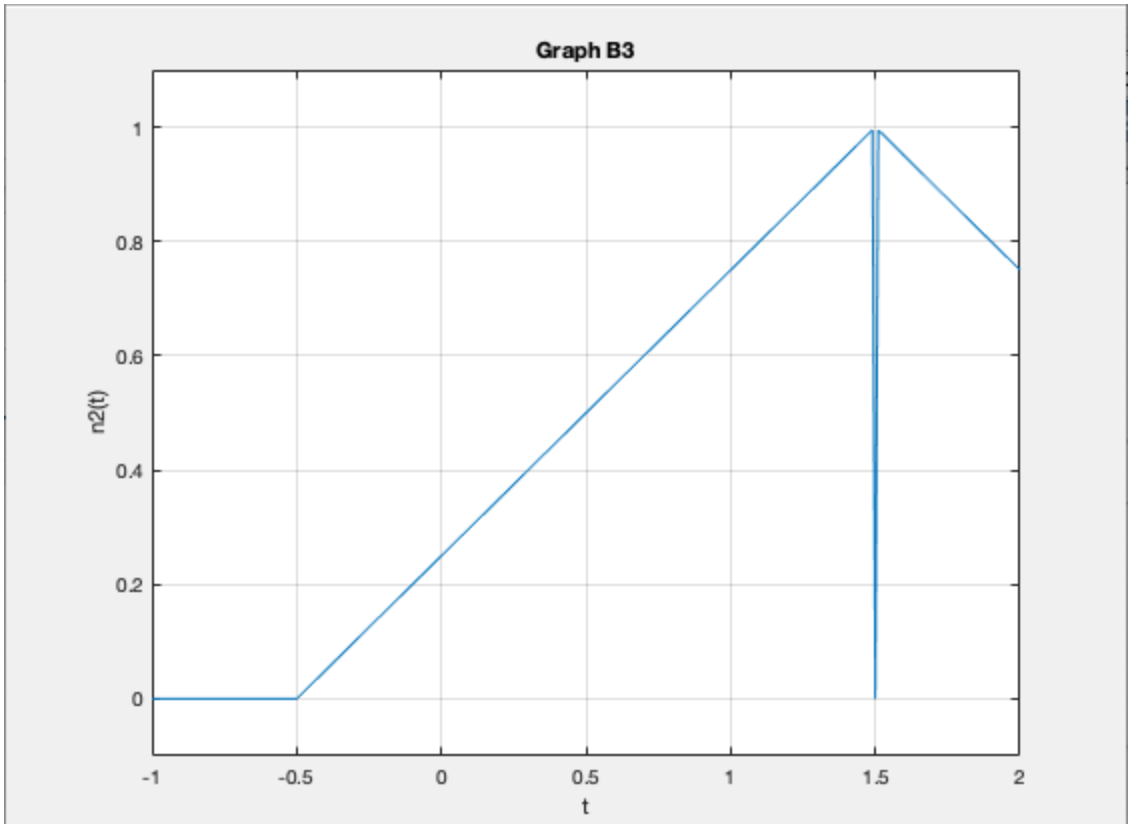
```
n1 = @(t) n(t/2);
t = (-1:0.01:2);
plot(t,n1(t));
xlabel('t'); ylabel('n1(t) =n(t/2)');
grid; title ("Graph B2"); figure;
axis([-1 2 -.1 1.1]);

% hold on

n2 = @(t) n1(t+(1/2));
t = (-1:0.01:2);
plot(t,n2(t));
xlabel('t'); ylabel('n2(t)');
axis([-1 2 -.1 1.1]);
grid; title ("Graph B3");

% hold off
```




Graph B3
n2(t)
t
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2

B4:

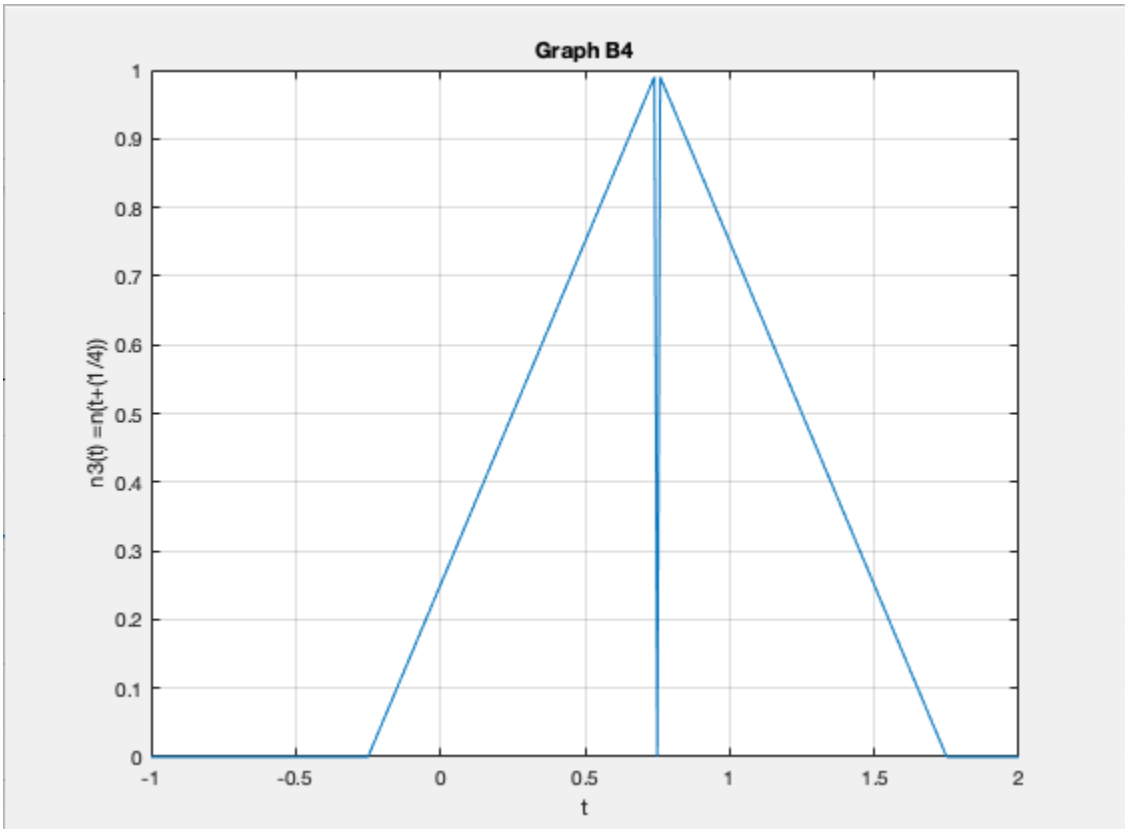
Matlab Code used to plot Graph B4:

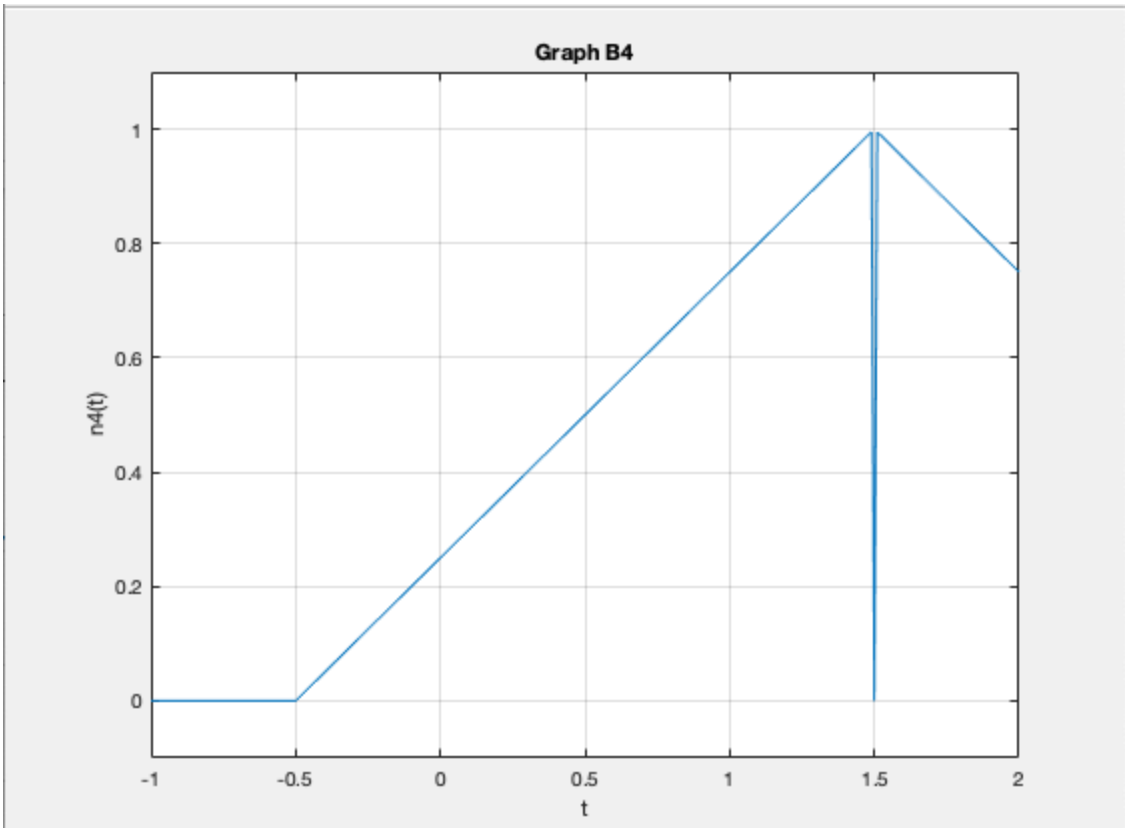
```
n3 = @(t) n(t+(1/4));
t = (-1:0.01:2);
plot(t,n3(t));
xlabel('t'); ylabel('n3(t) =n(t+(1/4))');
grid; title ("Graph B4"); figure;
axis([-1 2 -.1 1.1]);

% hold on

n4 = @(t) n3(t*(1/2));
t = (-1:0.01:2);
plot(t,n4(t));
xlabel('t'); ylabel('n4(t)');
axis([-1 2 -.1 1.1]);
grid; title ("Graph B4");

% hold off
```





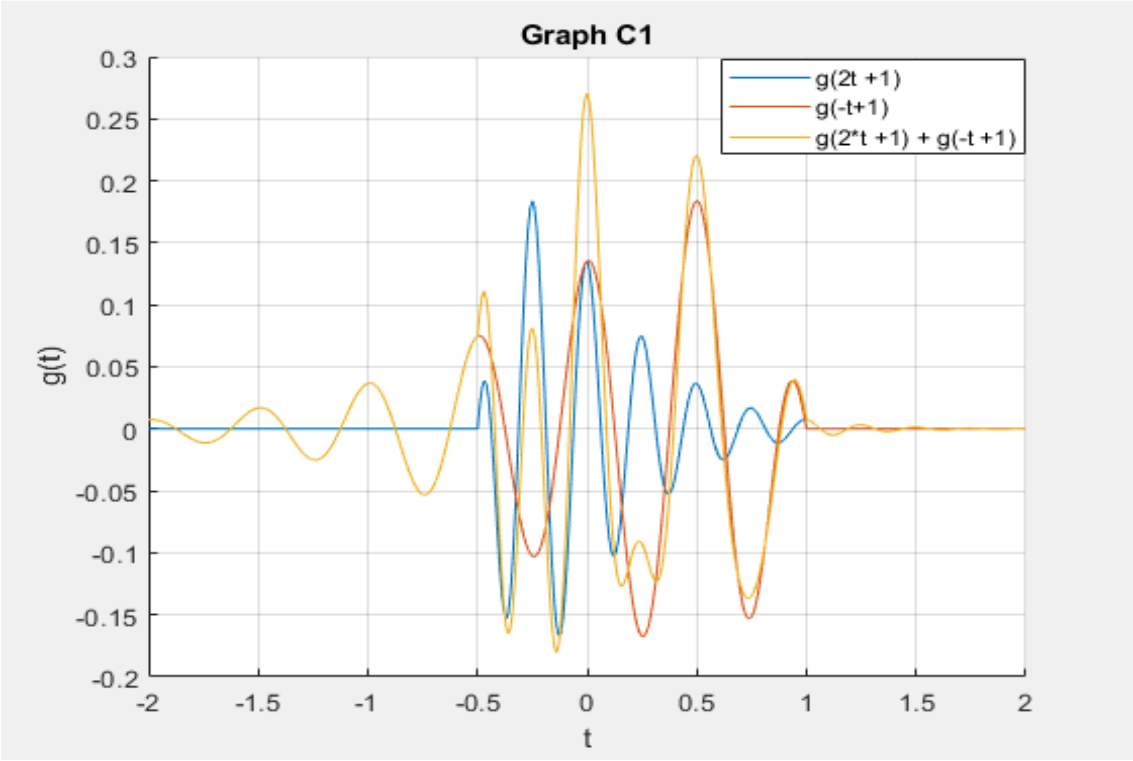
B5:

The graphs of n2(t) and n4(t) are identical because they are 2 functions with the same values. This is because both function expressed with regards to n(t) results in, n2(t)=n((½)t+(½)) and n4(t)=n((½)(t+(¼))) which are equivalent in terms of transformation of n

C1:

Matlab Code used to plot Graph C1:

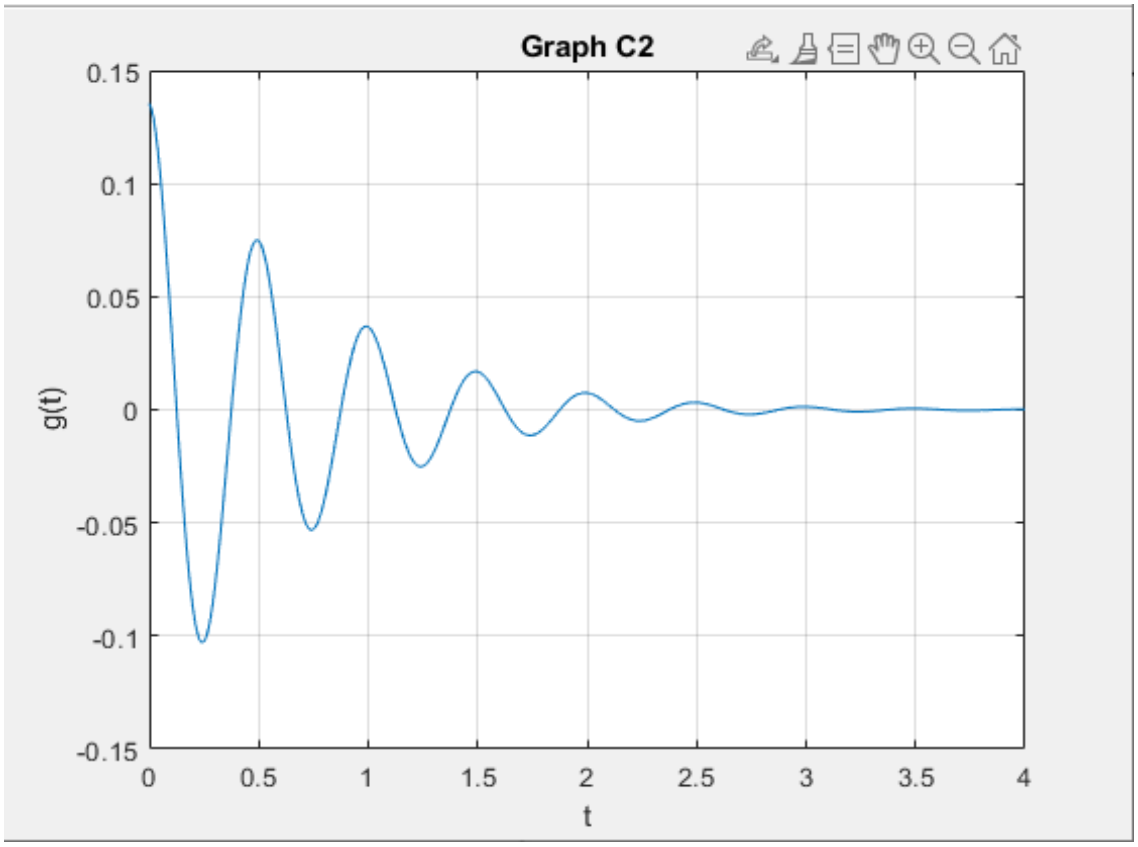
```
f = @(t) exp(-2*t).*cos(4*pi*t);
u = @(t) t*1.0.*((t>=0));
g = @(t) f(t).*u(t);
t = (-2:0.01:2);
hold on
plot(t, g(2*t+1));
plot(t, g(-t + 1));
plot(t, g(2*t +1) + g(-t +1));
xlabel('t'); ylabel('g(t)'); grid; title("Graph C1");
legend("g(2t +1)", "g(-t+1)", "g(2*t +1) + g(-t +1)");
hold off
```



C2:

Matlab Code used to plot Graph C2:

```
1 -    f = @(t) exp(-2*t).*cos(4*pi*t);
2 -    g = @(t) f(t).*u(t);
3 -    t = (0:0.01:4);
4 -    figure; plot(t, g(t + 1));
5 -    xlabel('t'); ylabel('g(t)'); grid; title("Graph C2");
```



C3:

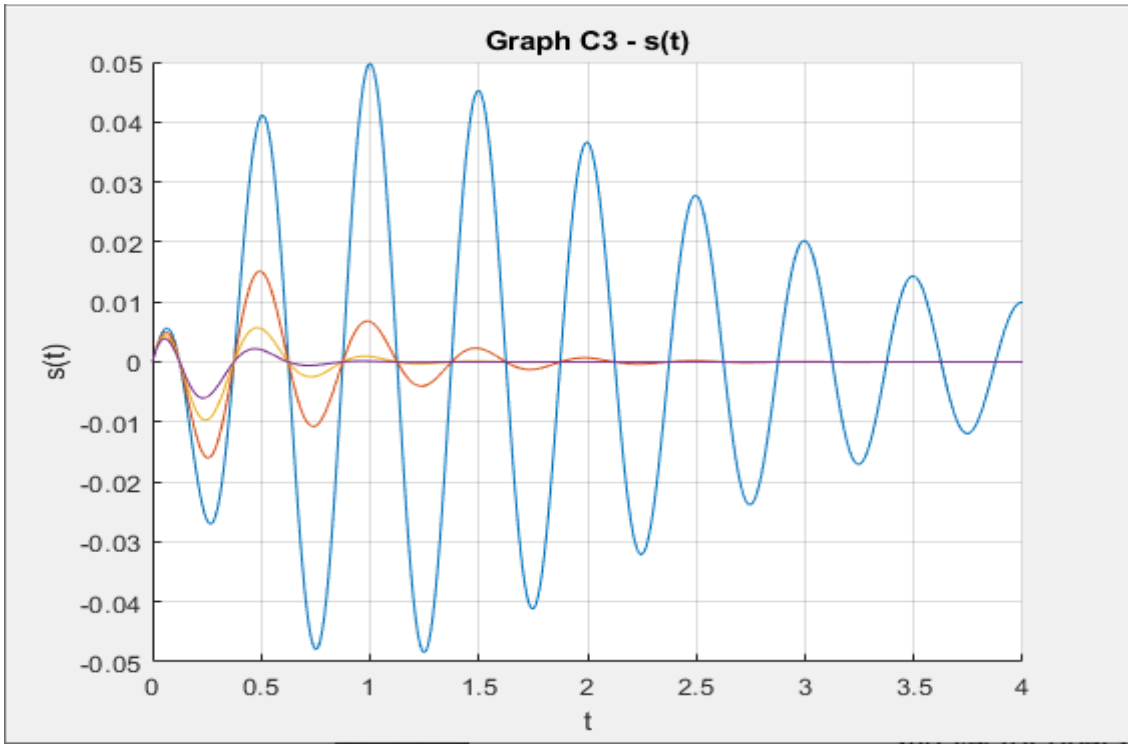
Matlab Code used to plot **Graph C3 - s(t)** and **Graph C3 - H(t)**:

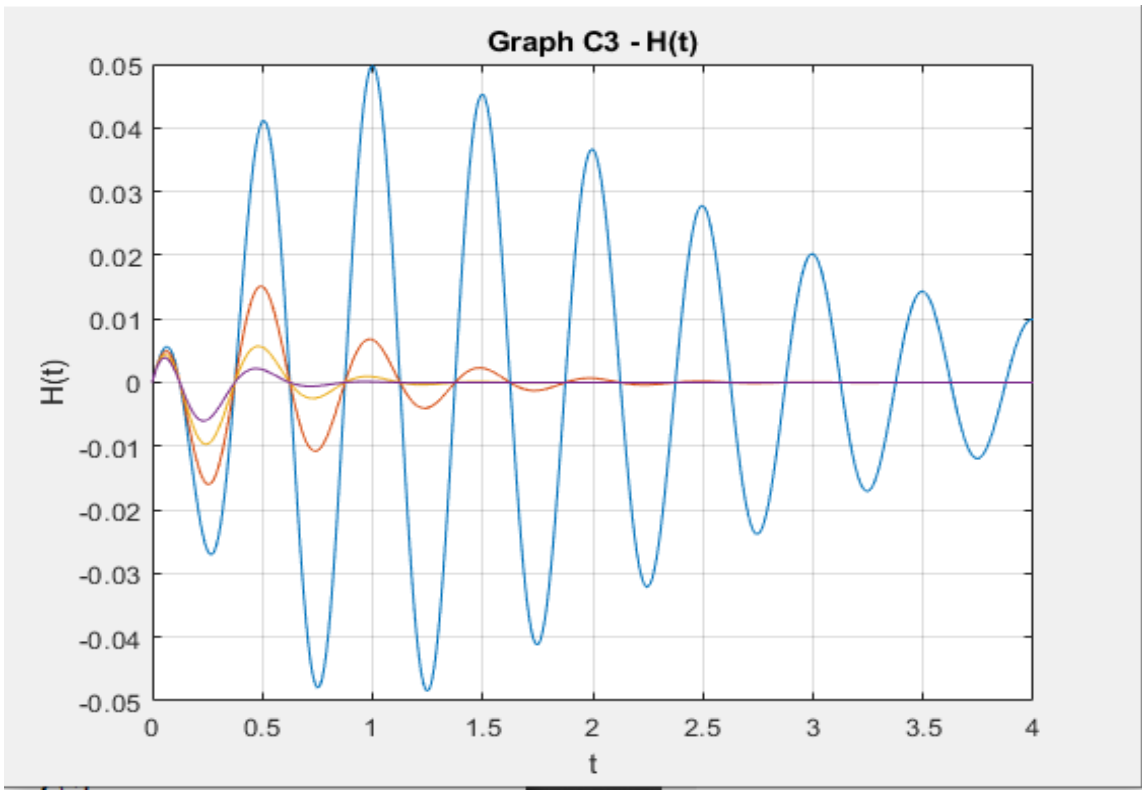
```
f = @(t,a) exp(-2).*exp(-a*t).*cos(4*pi*t);
s = @(t,a) f(t,a).*u(t);
t = (0:0.01:4);
xlabel('t'); ylabel('s(t)'); grid; title("Graph C3 - s(t)");

for a=1:2:7
    hold on
    plot(t, s(t,a));
    hold off

end

alpha = (1:2:7);
t = (0:0.01:4)';
T = t*ones(1,4);
H = exp(-2).*exp(-T*diag(alpha)).*cos(4*pi*T).*u(t);
figure; plot(t, H); xlabel('t'); ylabel('H(t)'); grid; title("Graph C3 - H(t)");
[i, j] = size(H);
fprintf("The size of matrix is %d by %d", i, j);
```


Graph C3 - s(t)
s(t)
t
0.05
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
0
0.5
1
1.5
2
2.5
3
3.5
4


Graph C3 - H(t)
H(t)
t
0.05
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
0
0.5
1
1.5
2
2.5
3
3.5
4

C4:

The dimensions of the matrix produced by the **H(t)** in **C3** is 401 by 4. Therefore the length of the matrix is 1604.

D1:

Matlab code used for problem D1:

```
1 -     D = A;
2
3 -     D(:)
4 -     D([2 4 7])
5 -     ([D >= 0.2])
6 -     D([ D >= 0.2])
7 -     D([ D >= 0.2]) = 0
```

(a) The following operation, D(:), displays the matrix vertically starting from the leftmost column.

```
ans =

    0.5377
    1.8339
   -2.2588
    0.8622
    0.3188
   -1.3077
   -0.4336
    0.3426
    3.5784
    2.7694
   -1.3499
    3.0349
    0.7254
   -0.0631
    0.7147
   -0.2050
   -0.1241
    1.4897
    1.4090
    1.4172
```

(b) The following operation, D([2 4 7}), displays the contents at the following indexes: (1, 2), (1, 5), (2, 2).

```
ans =

    1.8339    0.8622   -0.4336
```

(c) The following operation, ([D >= 0.2]), displays the logical matrix of D. It assigns the digit 1 if the value at an index satisfies D >= 0.2, otherwise it assigns the digit 0.

```
ans =

  5×4 logical array

   1   0   0   0
   1   0   1   0
   0   1   1   1
   1   1   0   1
   1   1   1   1
```

(d) The following operation, D([D >= 0.2]), displays the contents of any index in a list format that satisfies the inequality D >= 0.2.

```
ans =

    0.5377
    1.8339
    0.8622
    0.3188
    0.3426
    3.5784
    2.7694
    3.0349
    0.7254
    0.7147
    1.4897
    1.4090
    1.4172
```

(e) The following operation, D([D >= 0.2]) = 0, sets any index that satisfies D >= 0.2 to 0, and displays the contents afterwards in a matrix format.

```
D =

         0   -1.3077   -1.3499   -0.2050
         0   -0.4336         0   -0.1241
   -2.2588         0         0         0
         0         0   -0.0631         0
         0         0         0         0
```

D2:

```
1  -    C=B;
2  -     tic
3  -   for i=1:1:1024
4  -       for j=1:1:100
5  -       if abs(C(i,j))<0.01
6  -           C(i,j)=0;
7  -       end
8  -       end
9  -   end
10 -     toc
11
12     %  Elapsed t=ime is 0.001421 seconds.
13
14 -   C=B;
15 -   tic
16 -   C(abs(C)<0.01)=0;
17 -   toc
18
19     % Elapsed time is 0.000875 seconds.
```

```
>> ProblemD2
Elapsed time is 0.001460 seconds.
Elapsed time is 0.000934 seconds.
```

Set B to a variable C so that the value of the raw data does not change from one iteration to the next. The task of setting any value in the matrix B with magnitude <0.01 to 0 is performed twice, once with 2 nested for loops and once using matlab's built-in indexing feature and their execution times are measured using the tic toc function. The results are conclusive and show that

using the second method, the execution time is about 55-65% the execution time of the first method meaning it is much more efficient at executing the task.

D3:

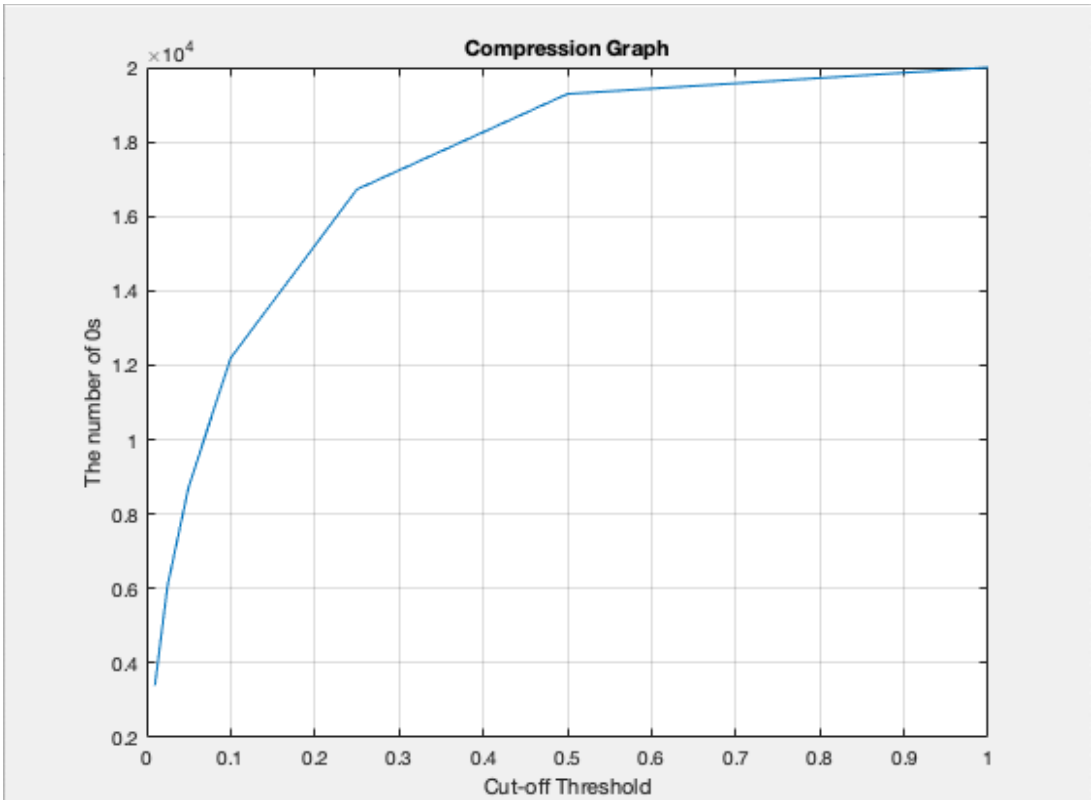
```
count=0;
y_audio=x_audio;
y_audio(abs(y_audio)<0.01)=0;
for i=1:1:20000
    if y_audio(i)==0
    count=count+1;
    end
end
sound(y_audio,8000)
disp(count);
```

```
>> ProblemD3
        3399
```

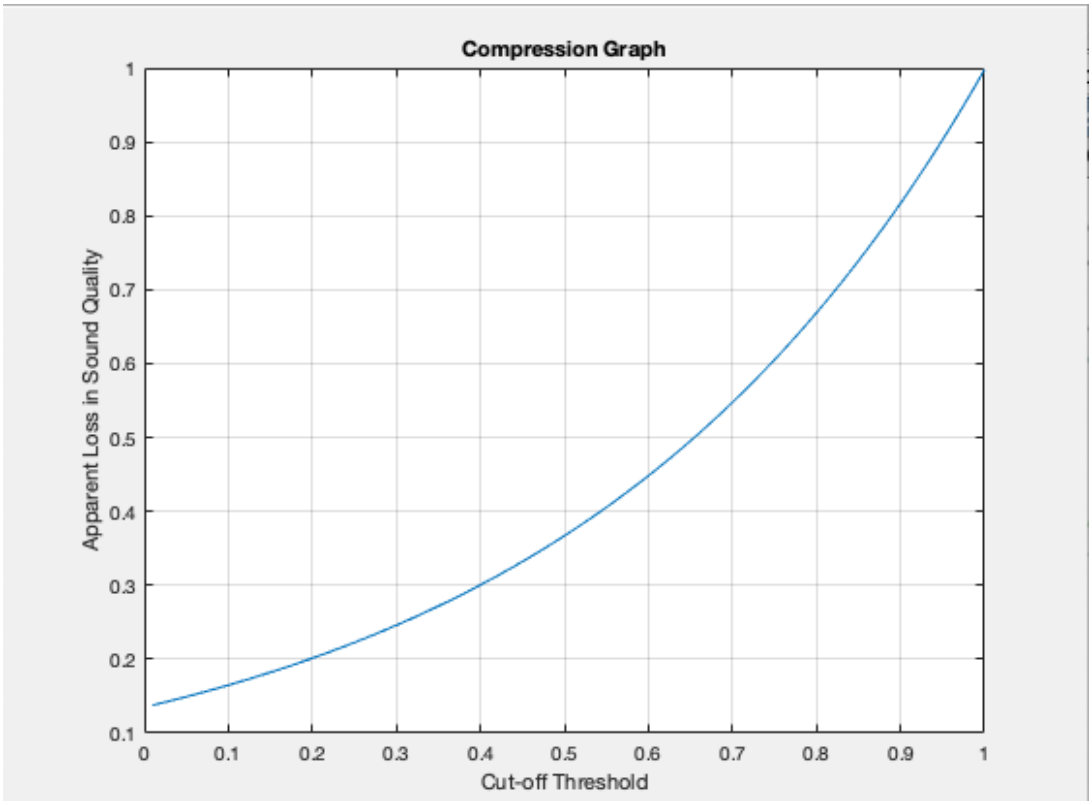
Set x_audio to a variable y_audio so that the value of the raw data does not change from one execution to the next and reset the count. Then use matlab's built-in indexing feature to set any value in the matrix with magnitude <0.01 to 0. Sift through the matrix using a for loop and check each element if it has a value of zero, if that condition is met then increment count. Finally play the sound and display count.

Conclusion

We created a simple audio compression algorithm that saves memory space at the cost of sound quality. This tradeoff can be optimized because the values falling under the threshold (that end up as 0) as a function of the cut-off threshold looks like a logarithmic function displayed below. (Created with real values from D3)



Yet the apparent loss in sound quality looks like an exponential function displayed below



Therefore, we can optimize the compression while minimizing the apparent loss in sound quality by picking a low cut-off threshold (somewhere around the 0.1 range) that would yield the majority of the file size savings (60% size reduction) for the least amount of apparent loss in sound quality (around 15% or so).