

Department of Electrical and Computer Engineering


Course Number	ELE532
Course Title	Signals and Systems I
Semester/Year	F2022

Instructor	Dr. Dimitri Androutsos
------------	------------------------

**ASSIGNMENT No. 1**

Assignment Title	Working with Matlab, Visualization of Signals
------------------	-----------------------------------------------

Submission Date	2022-09-18
Due Date	2022-10-03

Student Name	Danilo Zelenovic
Student ID	501032542
Signature*	

*\*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: [www.ryerson.ca/senate/current/pol60.pdf](http://www.ryerson.ca/senate/current/pol60.pdf).*

# Table of Contents

Part A..... Page 3

A.1... Page 3

A.2... Page 5

A.2... Page 6

Part B..... Page 6

B.1... Page 6

B.2... Page 7

B.3... Page 9

B.4... Page 11

B.5... Page 13

Part C..... Page 14

C.1... Page 14

C.2... Page 15

C.3... Page 16

C.4... Page 17

Part D..... Page 17

D.1... Page 17

D.2... Page 20

D.3... Page 22

## A. Anonymous functions and plotting continuous functions

### A.1

```
1 f = @(t) exp(-t).*cos(2*pi*t);  
2 t = (-2:2);  
3 plot(t,f(t));  
4 xlabel('t');  
5 ylabel('f(t)');  
6 grid;
```

Code A1a. Matlab code for Figure 1.46.

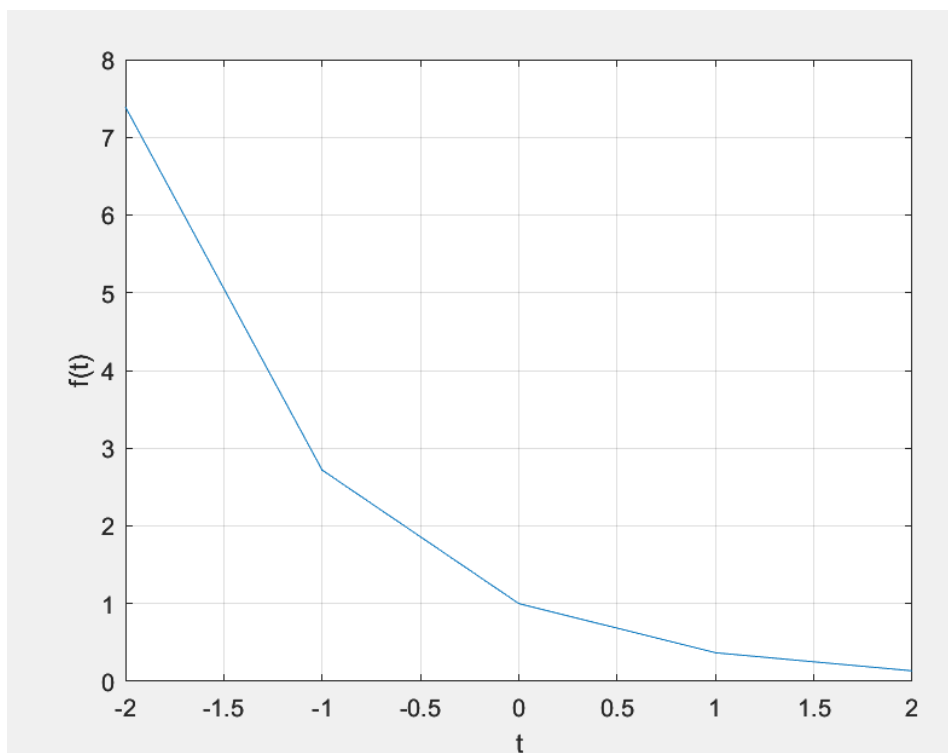


Figure 1.46. Generated graph.

```
1 f = @(t) exp(-t).*cos(2*pi*t);  
2 t = (-2:0.01:2);  
3 plot(t,f(t));  
4 xlabel('t');  
5 ylabel('f(t)');  
6 grid;
```

Code A1b. Matlab code required to generate Figure 1.47.

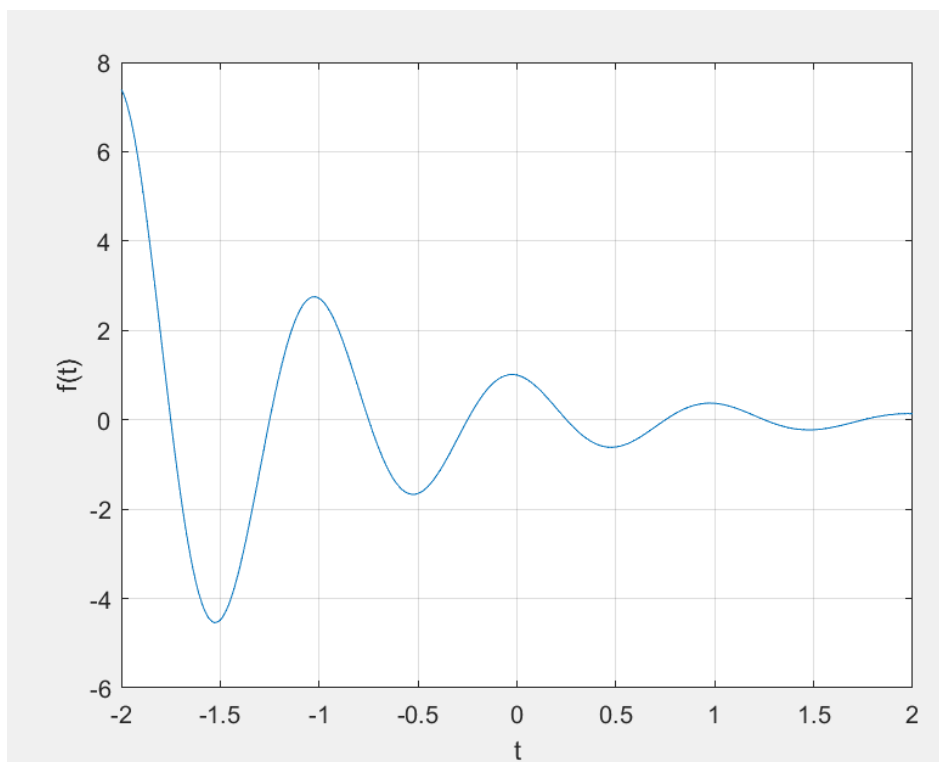


Figure 1.47. Generated graph.

## A.2

```
1 f = @(t) exp(-t);  
2 t = [-2:2];  
3 plot(t,f(t));  
4 xlabel('t');  
5 ylabel('f(t)');  
6 grid;
```

Code A2. Code for plotting Figure A.2.

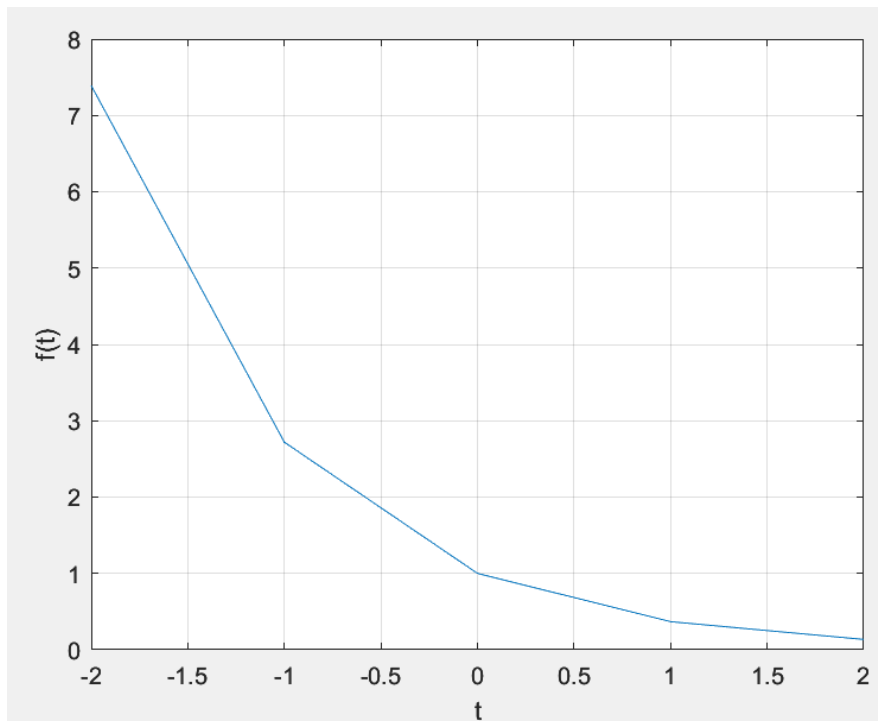


Figure A.2. Graph of  $e^{-t}$ .

## A.3

Figure 1.46 and Figure A.2 are identical. Because of the bounds and increments, both graphs ended up identical.

## B. Time shifting and time scaling

### B.1

```
1  p = @(t) 1.0.*((t>=0)&(t<1));  
2  t = (-1:0.01:2);  
3  plot(t,p(t));  
4  xlabel('t');  
5  ylabel('p(t) = u(t)-u(t-1)');  
6  axis([-1 2 -.1 1.1]);  
7  grid;
```

Code B1. Code for  $p(t)$ .

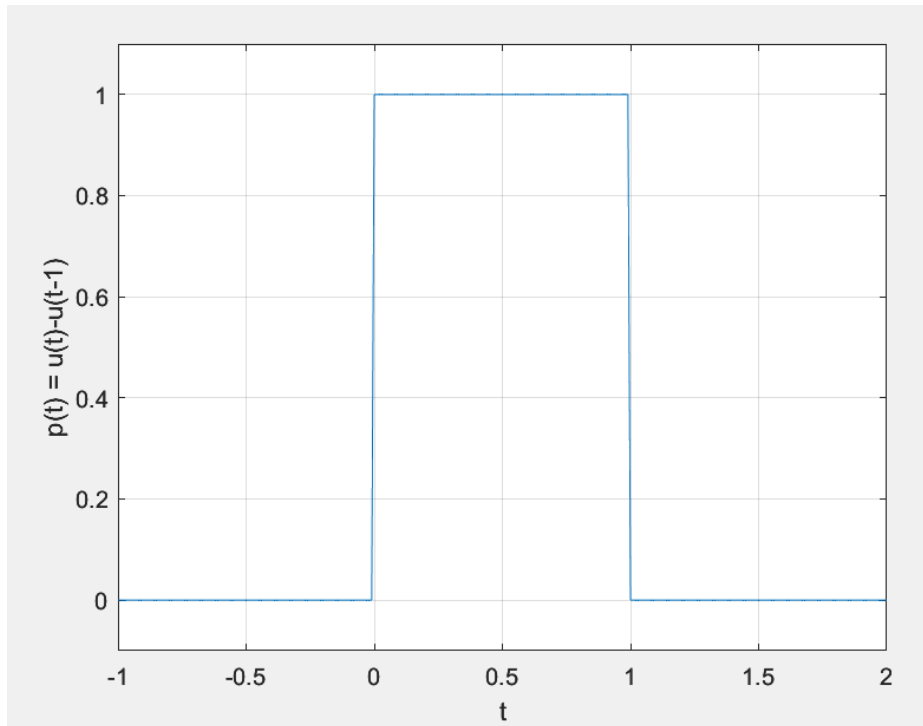


Figure 1.50. Generated graph of page 129.

## B.2

```

1      u = @(t) 1.0.*(t>=0);
2      p = @(t) u(t)-u(t-1);
3      r = @(t) t.*p(t);
4      t = (-1:0.01:2); %changed the t intervals to make the gap lower and look better
5      plot(t,r(t));
6      xlabel('t');
7      ylabel('r(t) = t*p(t)');
8      axis([-1 2 -.1 1.1]);
9      grid;

```

Code B2a. Code for  $r(t)$ , using  $p(t)$ .

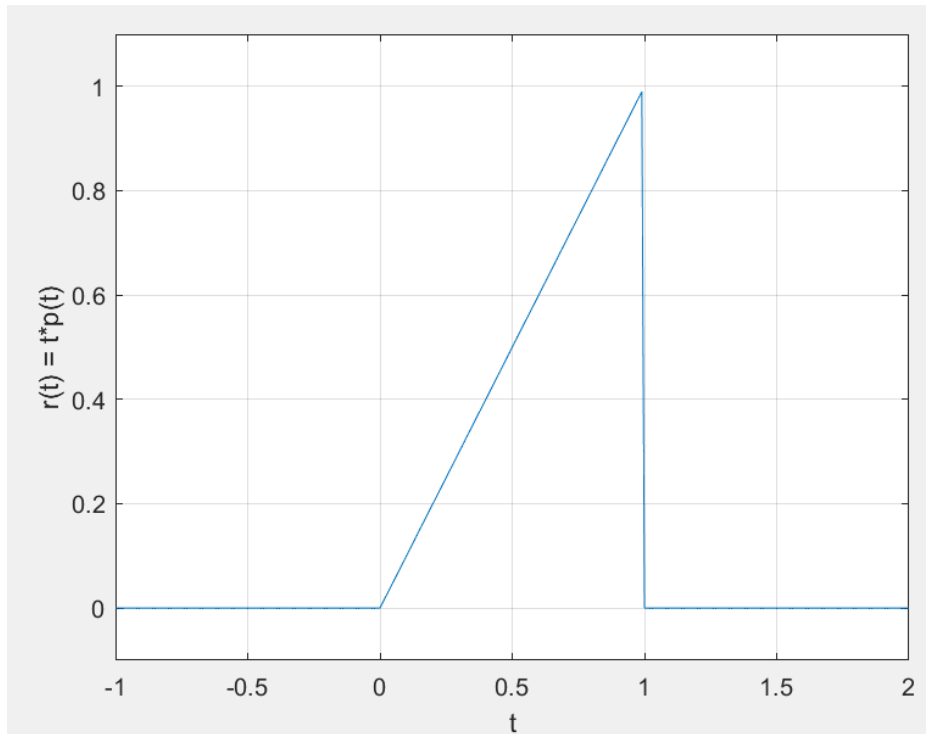


Figure B.2a.  $r(t)$  graph generated by code above.

```

1    u = @(t) 1.0.*(t>=0);
2    p = @(t) u(t)-u(t-1);
3    r = @(t) t.*p(t);
4    n = @(t) r(t) + r(-t + 2);
5    t = (-1:0.01:2); %changed the t intervals to make the gap lower and look better
6    plot(t,n(t));
7    xlabel('t');
8    ylabel('n(t) = r(t) + r(-t +2)');
9    axis([-1 2 -.1 1.1]);
10   grid;

```

Code B2b. Code to plot  $n(t)$ .



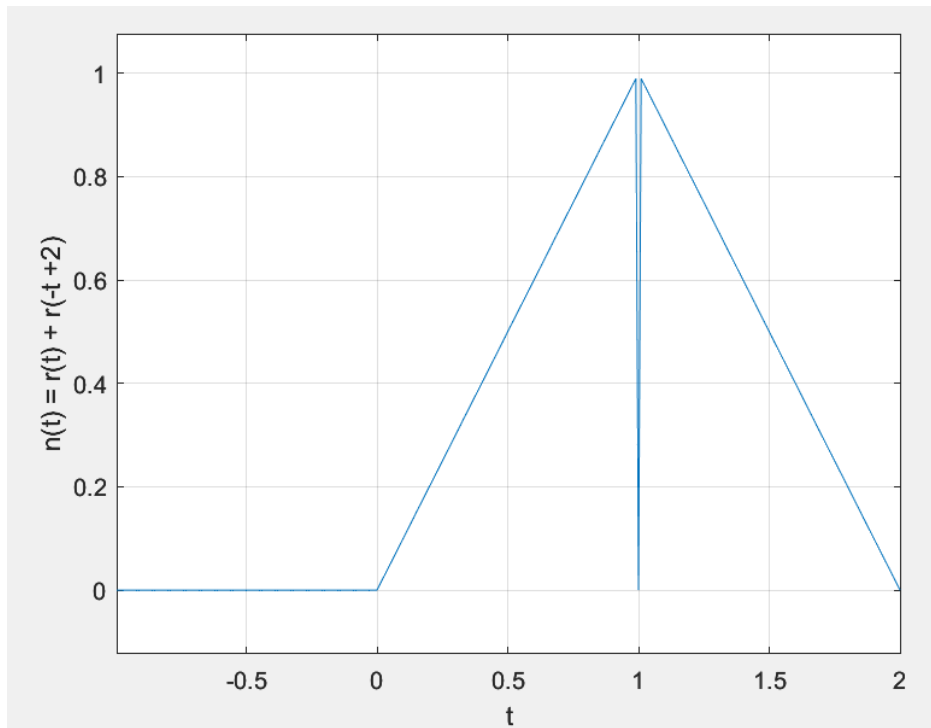


Figure B.2b. Generated graph of  $n(t)$ .

### B.3

```

1  p = @(t) 1.0.*((t>=0)&(t<1));
2  r = @(t) t.*p(t);
3  n = @(t) r(t) + r(-t + 2);
4  n1 = @(t) n(0.5.*t);
5  t = (-1:0.001:5); %changed the t intervals to make the gap lower and look better
6  plot(t,n1(t));
7  xlabel('t');
8  ylabel('n1(t) = n((1/2)t)');
9  axis([-1 5 -.1 1.1]);
10 grid;

```

Code B3a.

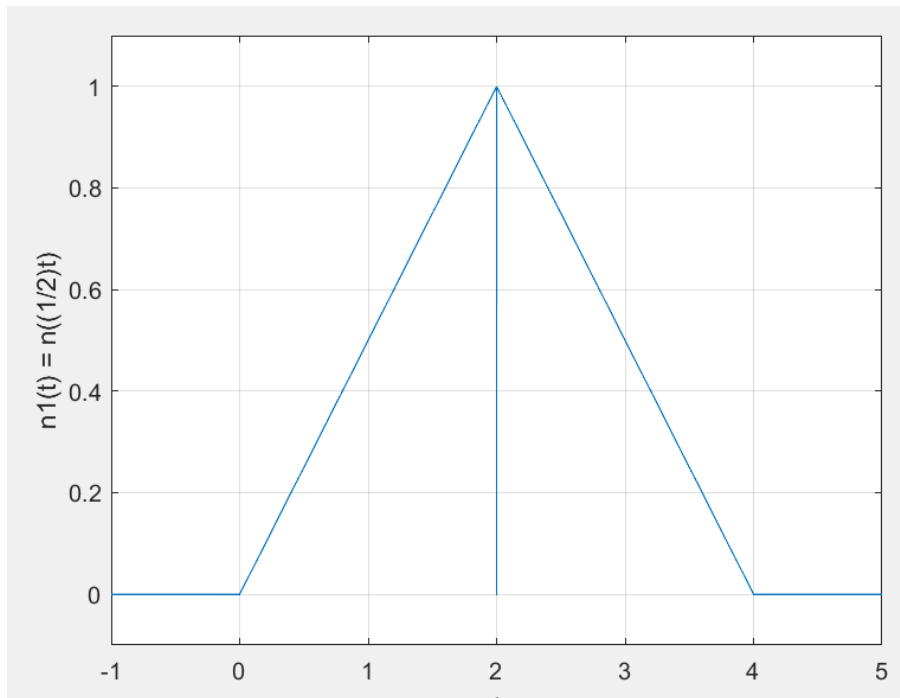


Figure n1. Full-scale graph of above code, over entire required axis.

```

1  p = @(t) 1.0.*((t>=0)&(t<1));
2  r = @(t) t.*p(t);
3  n = @(t) r(t) + r(-t + 2);
4  n1 = @(t) n((1/2).*t);
5  n2 = @(t) n1(t + (1/2));
6  t = (-1:0.001:5); %changed the t intervals to make the gap lower and look better
7  plot(t,n2(t));
8  xlabel('t');
9  ylabel('n2(t) = n1(t + (1/2))');
10 axis([-1 5 -.1 1.1]);
11 grid;

```

Code B3b.

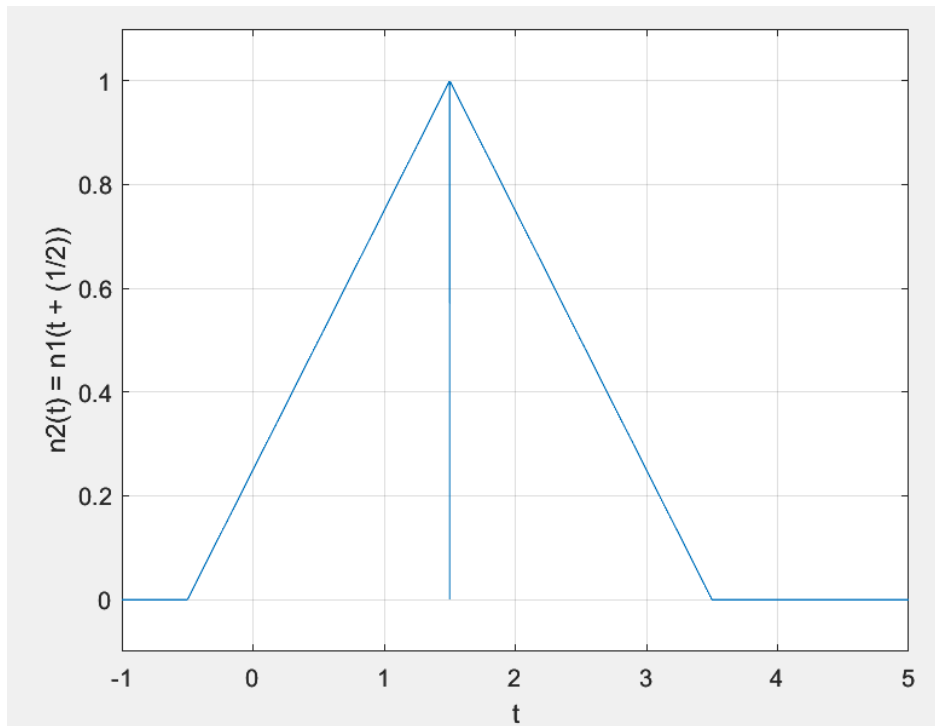


Figure n2. Full-scale graph of above code, over entire required axis.

## B.4

```

1      p = @(t) 1.0.*((t>=0)&(t<1));
2      r = @(t) t.*p(t);
3      n = @(t) r(t) + r(-t + 2);
4      n3 = @(t) n(t + (1/4));
5      t = (-1:0.001:5); %changed the t intervals to make the gap lower and look better
6      plot(t,n3(t));
7      xlabel('t');
8      ylabel('n3(t) = n(t + (1/4))');
9      axis([-1 5 -.1 1.1]);
10     grid;

```

Code B4a. Code for plot n3.

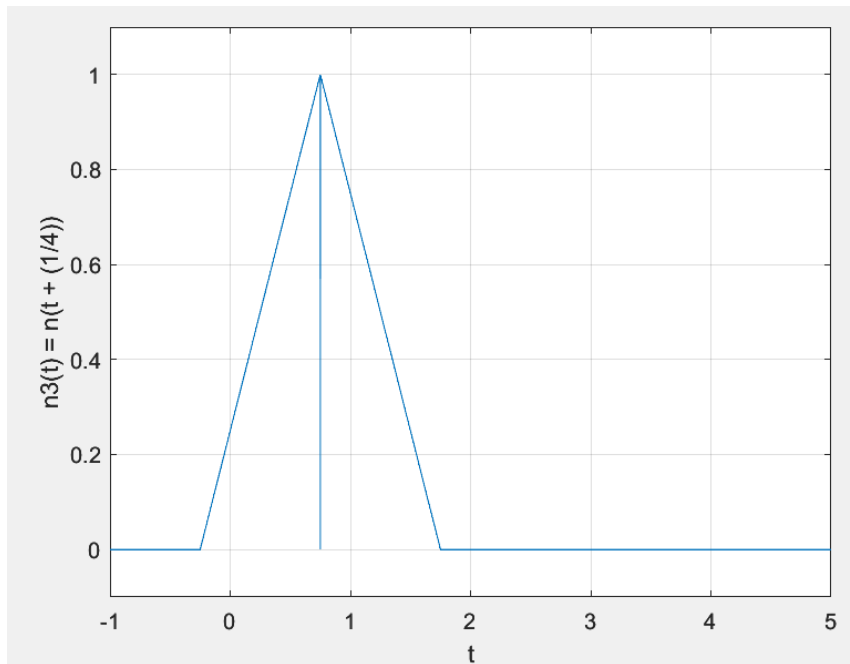


Figure n3. Figure ranging from -1 to 5.

```

1  p = @(t) 1.0.*((t>=0)&(t<1));
2  r = @(t) t.*p(t);
3  n = @(t) r(t) + r(-t + 2);
4  n3 = @(t) n(t + (1/4));
5  n4 = @(t) n3((1/2)*t);
6  t = (-1:0.001:5); %changed the t intervals to make the gap lower and look better
7  plot(t,n4(t));
8  xlabel('t');
9  ylabel('n4(t) = n3((1/2)*t)');
10 axis([-1 5 -.1 1.1]);
11 grid;

```

Code B4b.

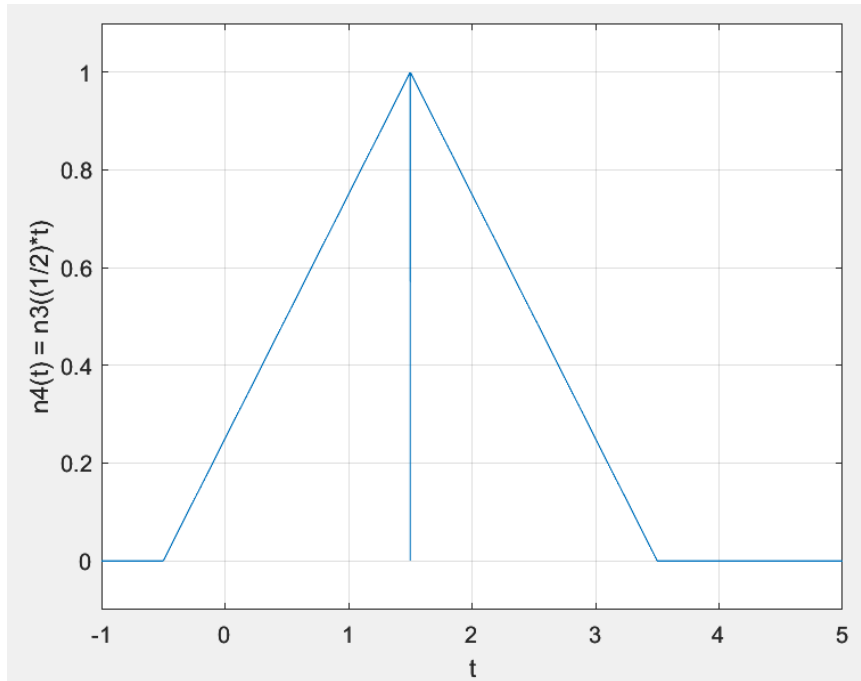


Figure n4. Full figure of n4 over required x-axis.

## B.5

In both of the graphs, the beginning and peak of the positive slope have identical x and y coordinates, meaning the slope must also be identical. Both graphs also have the same ending y-value (the value of y that corresponds to the defined end of the time boundary), which is (2,0.75). By going over each graph point by point, we notice that all points align. This is enough for us to conclude that both graphs are identical.

## C. Visualizing operations on the independent variable and algorithm vectorization

### C.1

```
1 g = @(t) f(t).*u(t);  
2 u = @(t) 1.0.*(t>=0);  
3 f = @(t) exp(-2*t).*cos(4*pi*t);  
4 t = (-2:0.01:2);  
5 axis([-2 2 -0.1 1.1]);  
6 plot(t,g(t));  
7 xlabel('t');  
8 ylabel('g(t) = f(t)u(t)');  
9 grid;
```

Code C1.

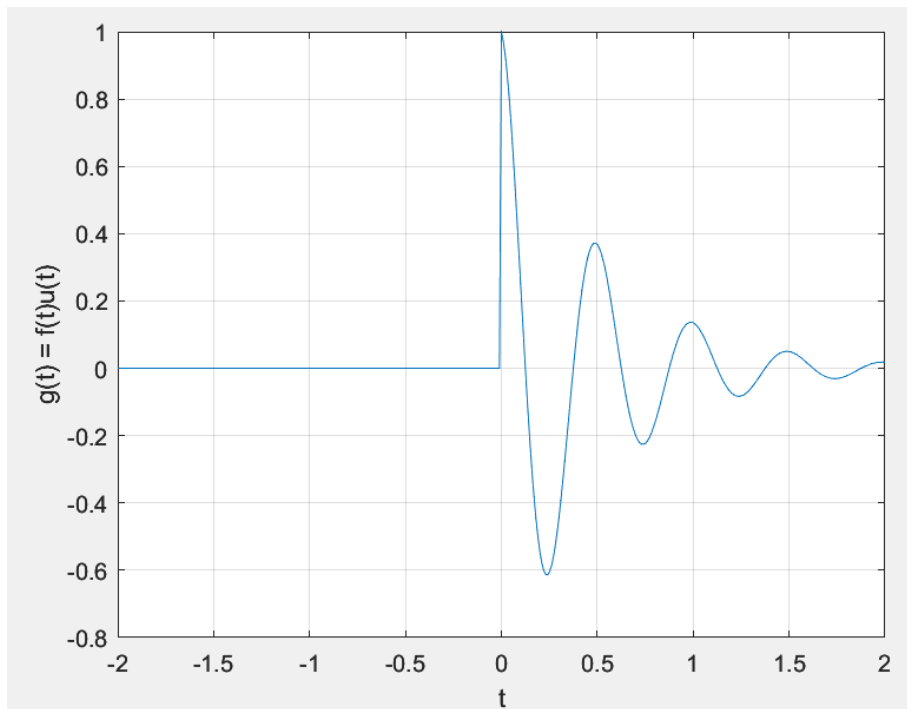


Figure g1. Graph of our altered version of textbook's  $g(t)$ .

## C.2

```
1 f = @(t) exp(-2*t).*cos(4*pi*t);  
2 t = (-2:0.01:2);  
3 u = @(t) 1.0.*(t>=0);  
4 axis([-2 2 -0.1 1.1]);  
5 g = @(t) f(t).*u(t);  
6 s = @(t) g(t+1);  
7 t = ([0:0.01:4]);  
8 plot(t,s(t));  
9 xlabel('t');  
10 ylabel('s(t) = g(t+1)');  
11 grid;
```

Code C2.

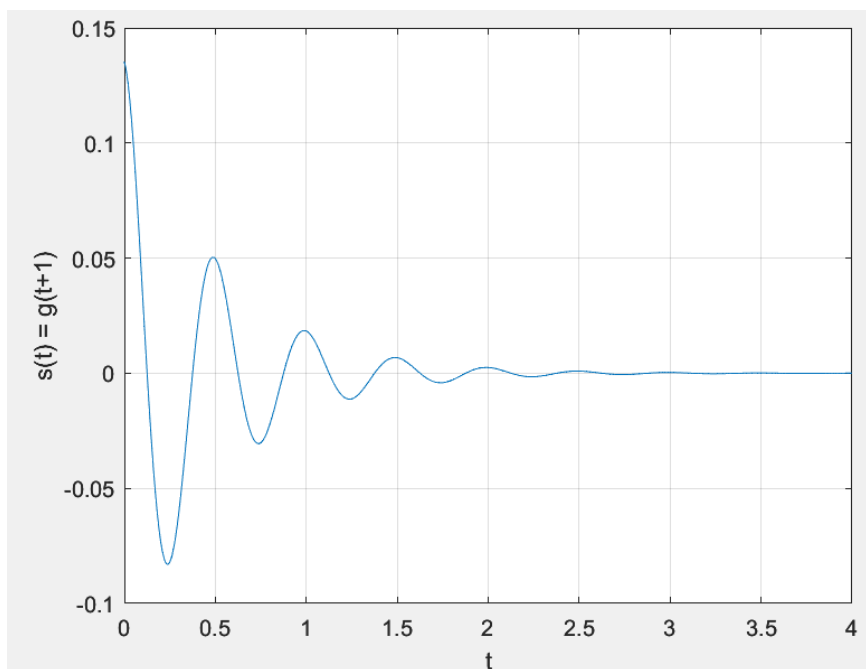


Figure s1. Generated graph of  $s(t)$ , using our previous  $g(t)$  code.

### C.3

```
1  u = @(t) 1.0.*(t>=0);
2  t = ([0:0.01:4]);
3  numel(s(t))
4  size(s(t))
5
6  for alpha = 1:2:7
7      s = @(t) exp(-2).*exp(-alpha.*t).*cos(4*pi*t).*u(t);
8      plot(t,s(t));
9      xlabel('t');
10     ylabel('s(t)')
11
12     hold on;
13 end
14
15 legend('alpha = 1', 'alpha = 3', 'alpha = 5', 'alpha = 7');
16
17 hold off;
```

Code C3. Also used in part C.4.

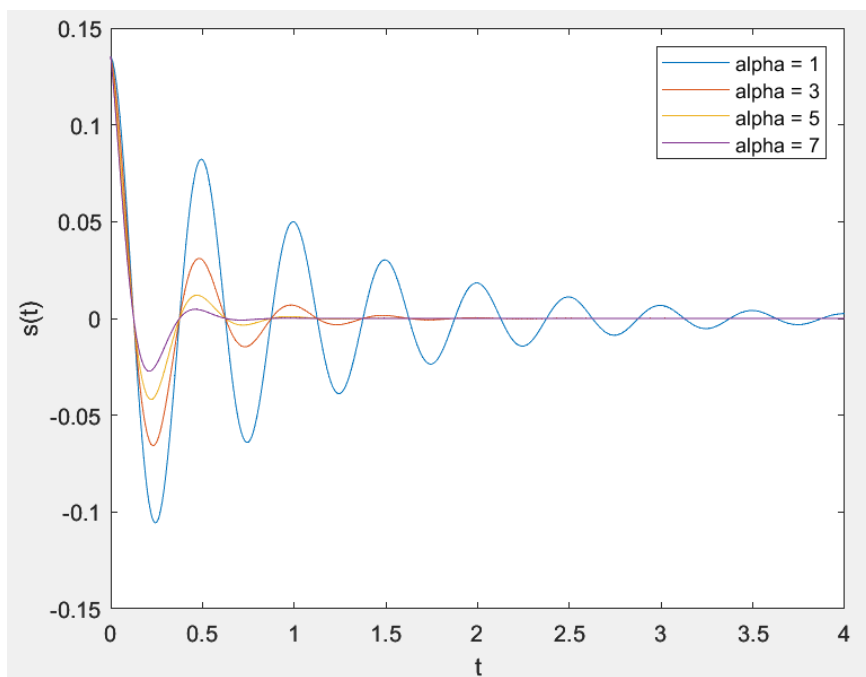


Figure s2. All value sets of alpha, on one graph, with legend.



## C.4

The size of matrix generated by  $s(t)$  was found to be  $1 \times 401$ , for a total of 401 elements. The code used was integrated within Code C3. Image showing this is below:

```
ans =  
  
    1    401
```

## D. Array indexing

### D.1

```
1  A = [0.5377 -1.3077 -1.3499 -0.2050;  
2      1.8339 -0.4336 3.0349 -0.1241;  
3      -2.2588 0.3426 0.7254 1.4897;  
4      0.8622 3.5784 -0.0631 1.4090;  
5      0.3188 2.7694 0.7147 1.4172]  
6  
7  d1 = A(:) %displays all values in a s  
8  d2 = A([2 4 7]) %displays matrix valu  
9  d3 = [A >= 0.2] %displays logical arr  
10 d4 = A([A >= 0.2]) %lists all element  
11 A([A >= 0.2]) = 0 % replaces all elem
```

Code D1.

a)

This function displays/lists all elements of matrix A in a single column, in order, from top left being the first, and going downwards, for each column until the bottom right value.

```
d1 =  
  
    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)

This displays all matrix values in positions 2, 4 and 7, with 1 being the upper-left most value, going downwards and right.

```
d2 =  
  
    1.8339    0.8622   -0.4336
```

c)

This function is used to create a 5x4 (or whatever given matrix size we have) logical array consisting of only 0's and 1's. In this case, we defined the function to make it a value of 1 wherever the element value was greater or equal to 0.2 and place a 0 everywhere else.

```
d3 =
```

```
5×4 logical array
```

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

d)

This command now lists all of the elements in a similar way as part a), however, now it only takes the elements which have a value that is greater than or equal to 0.2, and discards any values less than 0.2.

```
d4 =
```

```
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)

This function now searches through our matrix A, and replaces any element that has a value greater than or equal to 0.2 with 0, and keeps all other values as they were, and then displays the new matrix.

A =

```
      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
```

## D.2

a)

```
1  load('ELE532_Lab1_Data.mat')
2  rs = size(B,1);
3  cs = size(B,2);
4
5  for i = 1:1:rs
6      for j=1:1:cs
7          if(abs(B(i,j))<0.01)
8              B(i,j) = 0
9          end
10     end
11 end
```

Code D2a.

b)

```
1  load('ELE532_Lab1_Data.mat')
2  B([abs(B) < 0.01]) = 0
```

Code D2b. Same job as Code D2a, but with matlab indexing features.

c)

i)

```

1      tic
2      load('ELE532_Lab1_Data.mat');
3      rs = size(B,1);
4      cs = size(B,2);
5
6      for i = 1:1:rs
7          for j=1:1:cs
8              if(abs(B(i,j))<0.01)
9                  B(i,j) = 0;
10             end
11         end
12     end
13     toc

```

Code D2c. Same as code D2a, but with tic and toc implemented to track the time required to complete.

```

>> ProblemD_2partC_A
Elapsed time is 0.006620 seconds.

```

ii)

```

1      tic
2      load('ELE532_Lab1_Data.mat');
3      B([abs(B) < 0.01]) = 0;
4      toc

```

Code D2d. Same as code D2b, but with tic and toc implements to track the time required to complete.

```

>> ProblemD_2partC_B
Elapsed time is 0.006392 seconds.

```

NOTE: The time varies slightly each time the code is executed, so noted values are approximate. However, we can see that while extremely close, the second implementation appears to be slightly faster.

### D.3

```
1  load("ELE532_Lab1_Data.mat");
2
3  noise = x_audio;
4  rows = size(noise,1);
5  columns = size(noise,2);
6
7  zs = 0;
8
9  for r = 1:rows
10     for c = 1:columns
11         if(abs(noise(r,c)) == 0)
12             zs = zs + 1;
13         end
14     end
15 end
16
17 zs
18
19 sound(noise,8000)
```

Code D4. Code with all specifications accommodated.

The number of elements set to 0 is 58, as calculated by the program:

zs =

58