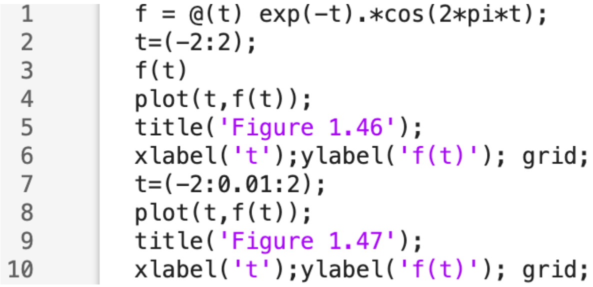
Lab 1: Working with MATLAB Functions, Visualization of Signals, and Signals Properties

A. Anonymous functions and plotting continuous functions

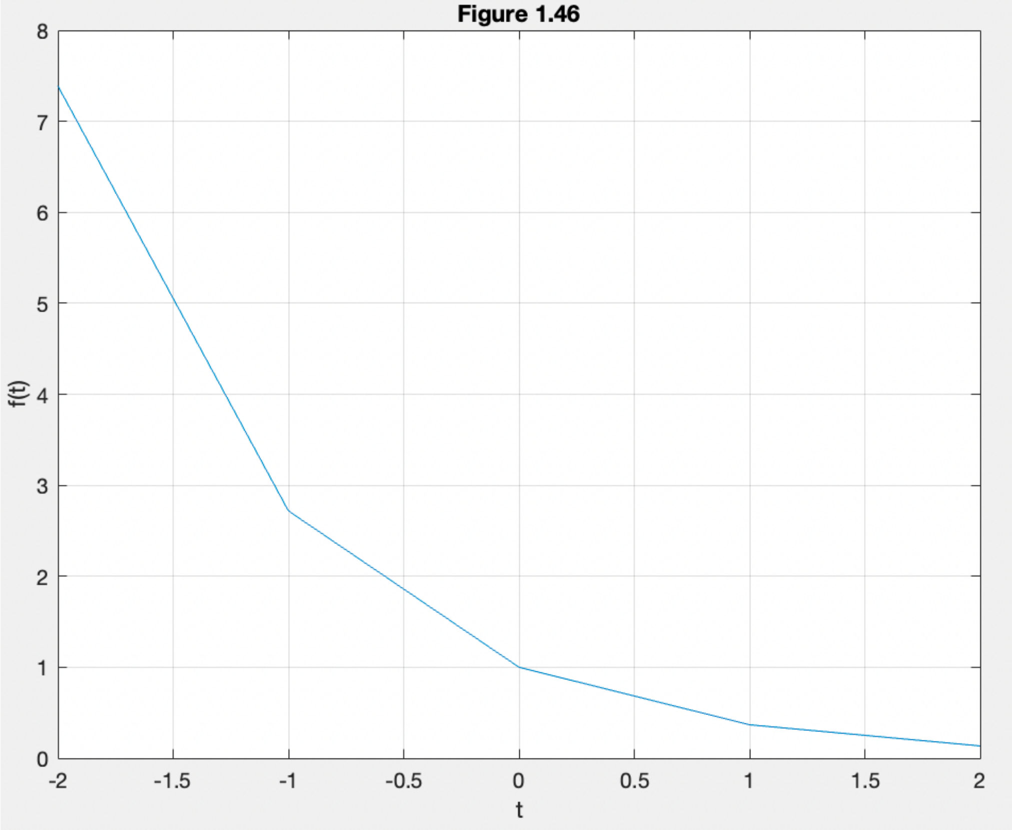
Problem A.1: Generate and plot the graphs as shown in Figures 1.46 and 1.47.

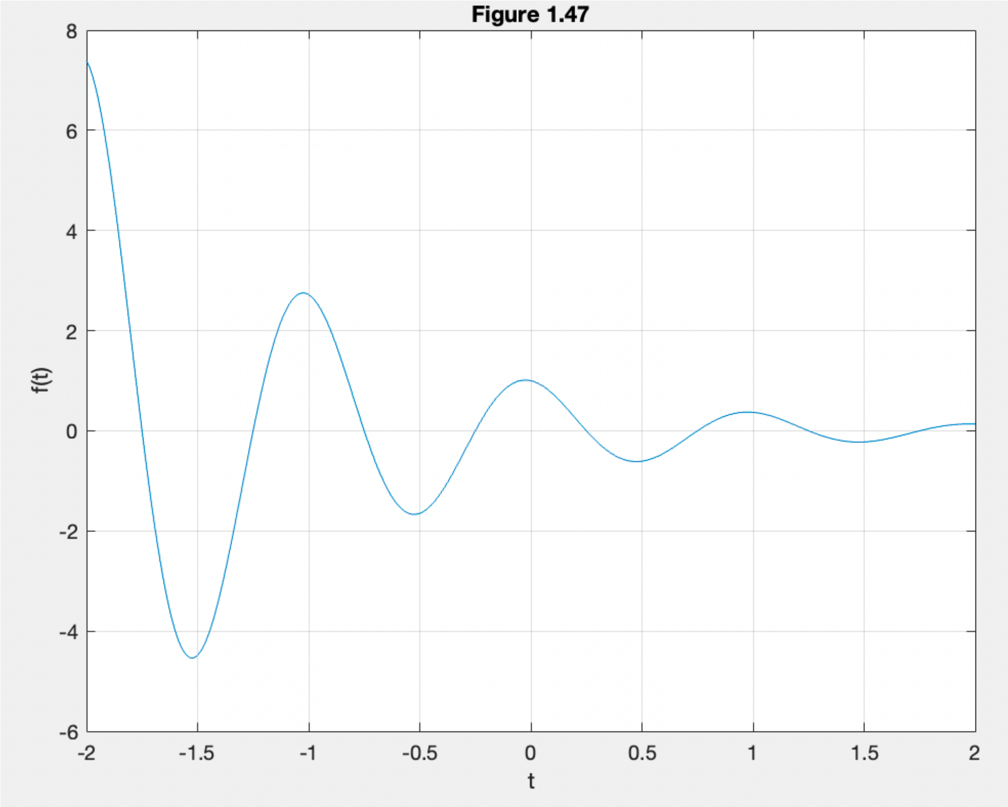
Code:



```
f = @(t) exp(-t).*cos(2*pi*t);
t=(-2:2);
f(t)
plot(t,f(t));
title('Figure 1.46');
xlabel('t');ylabel('f(t)'); grid;
t=(-2:0.01:2);
plot(t,f(t));
title('Figure 1.47');
xlabel('t');ylabel('f(t)'); grid;
```

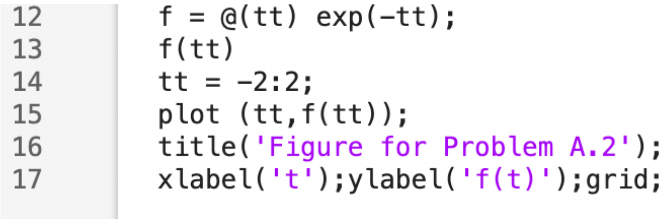
Plots:





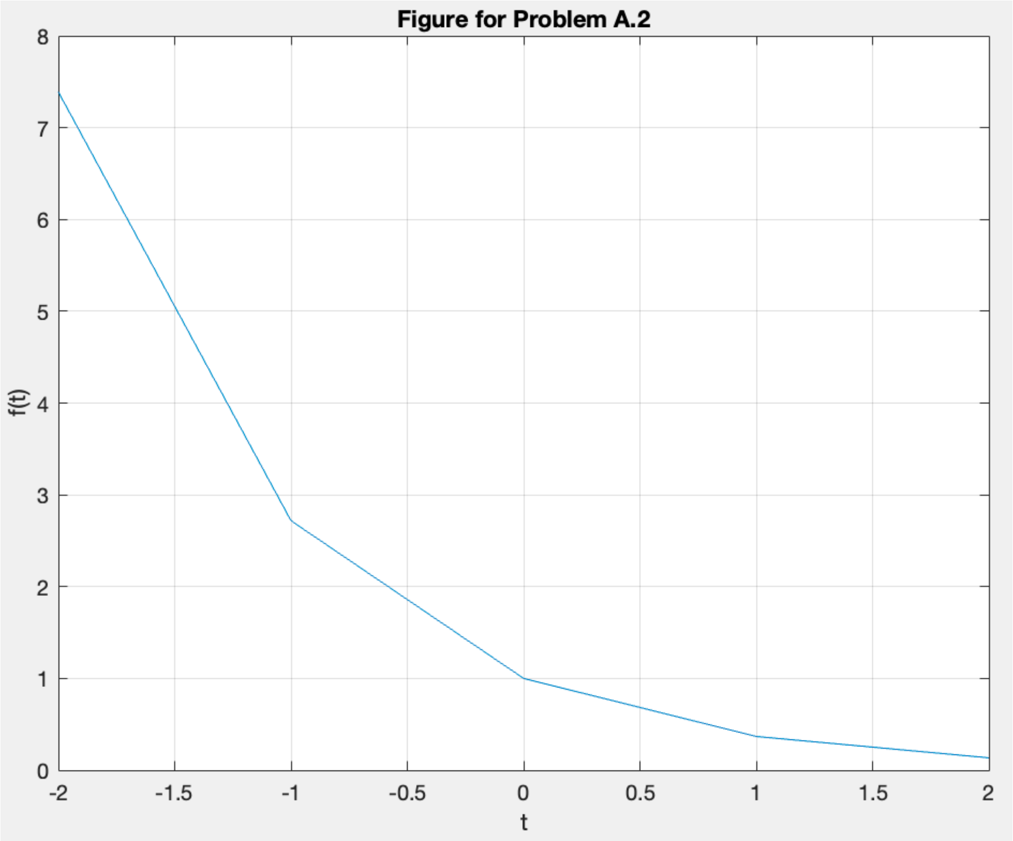
Problem A.2: Plot the function e^{-t} for t taking on integer values contained in $-2 \leq t \leq 2$.

Code:



```
f = @(tt) exp(-tt);
f(tt)
tt = -2:2;
plot (tt,f(tt));
title('Figure for Problem A.2');
xlabel('t');ylabel('f(t)');grid;
```

Plot:



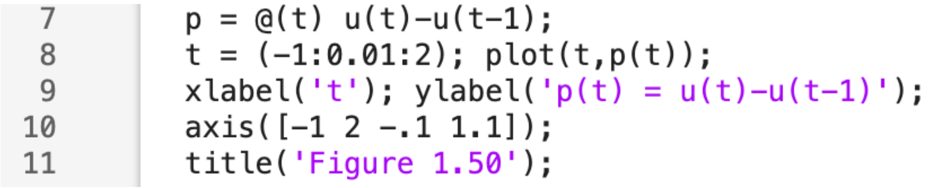
Problem A.3: Compare the results of Problem A.2 with Figure 1.46 in Problem A.1.

The Figure for Problem A.2 and Figure 1.46 are different functions but generate the same plot, this is the case because in Figure 1.46 there is a multiplication factor of cos(2πt) which results to 1 in all time intervals specified. The cosine multiplication factor for t = -2, -1, 0, 1, 2 are cos(-4π), cos(-2π), cos(0), cos(2π) and cos(4π) respectively. All these values equal to 1 so basically the function plotted is e^{-t} * 1 which is the same as the function in Problem A.2.

B. Time shifting and time scaling

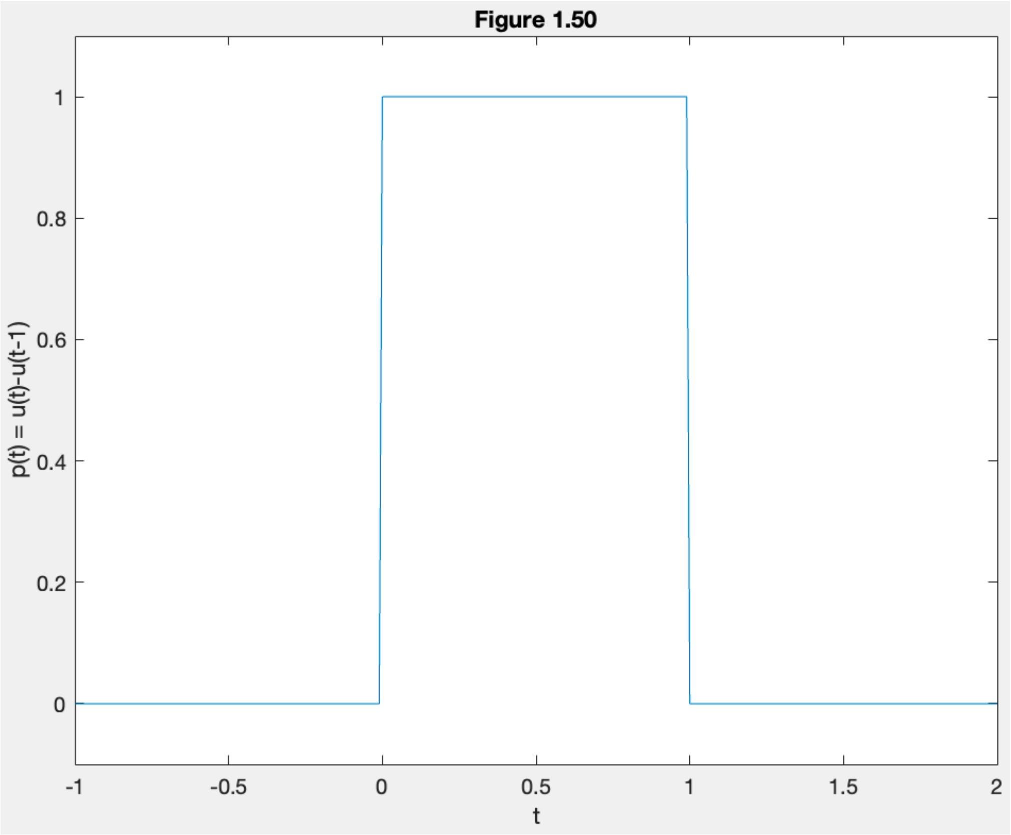
Problem B.1: Generate and plot p(t) as shown in Figure 1.50 on page 129.

Code:



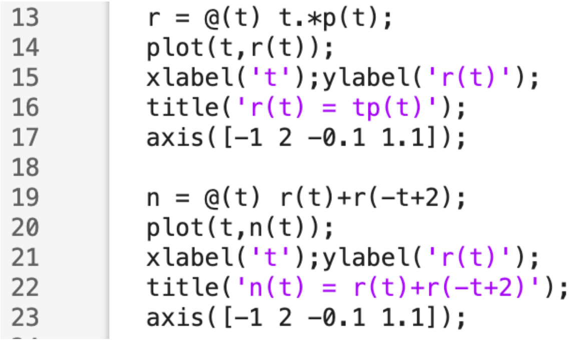
```
p = @(t) u(t)-u(t-1);
t = (-1:0.01:2); plot(t,p(t));
xlabel('t'); ylabel('p(t) = u(t)-u(t-1)');
axis([-1 2 -.1 1.1]);
title('Figure 1.50');
```

Plot:



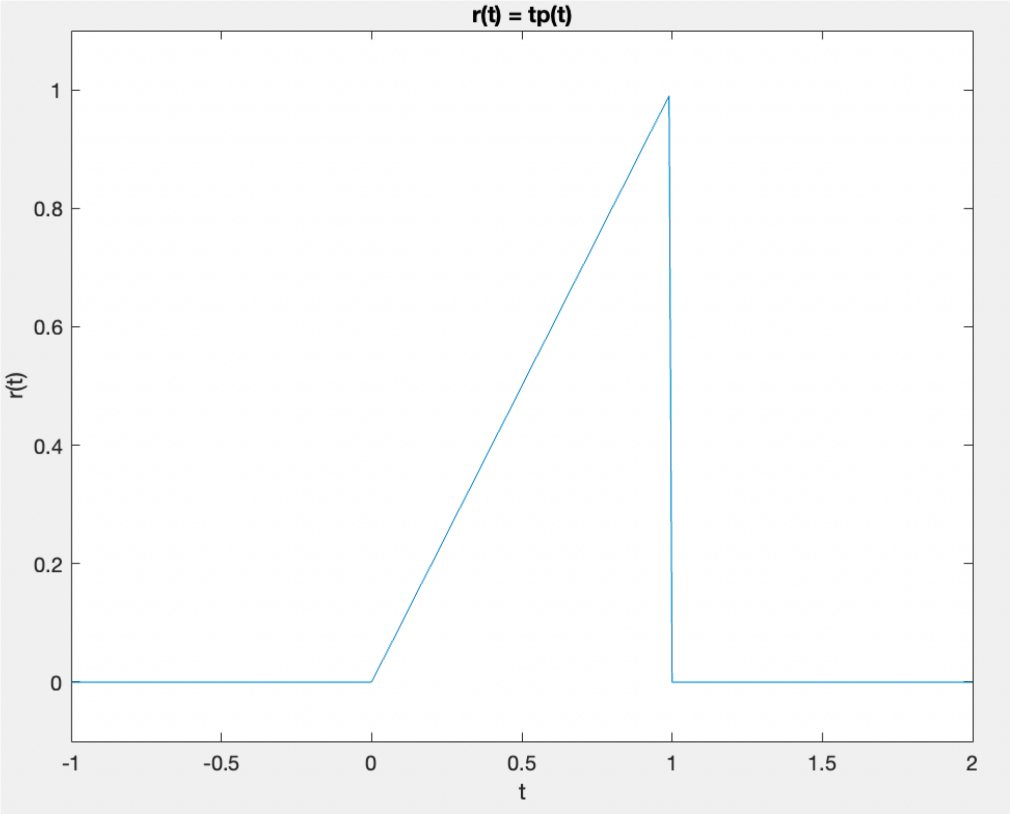
Problem B.2: Use p(t) defined in Problem B.1 to generate and plot functions $r(t) = tp(t)\ and\ n(t) = r(t) + r(-t+2)$.

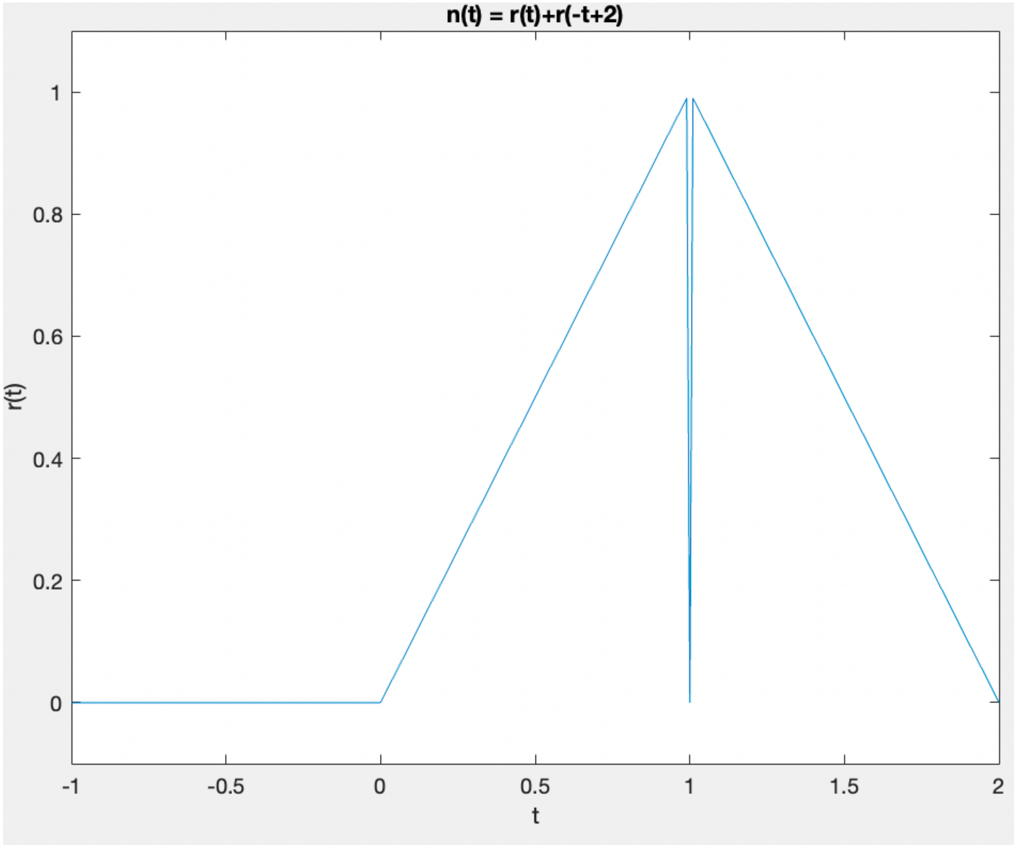
Code:



```
r = @(t) t.*p(t);
plot(t,r(t));
xlabel('t');ylabel('r(t)');
title('r(t) = tp(t)');
axis([-1 2 -0.1 1.1]);

n = @(t) r(t)+r(-t+2);
plot(t,n(t));
xlabel('t');ylabel('r(t)');
title('n(t) = r(t)+r(-t+2)');
axis([-1 2 -0.1 1.1]);
```

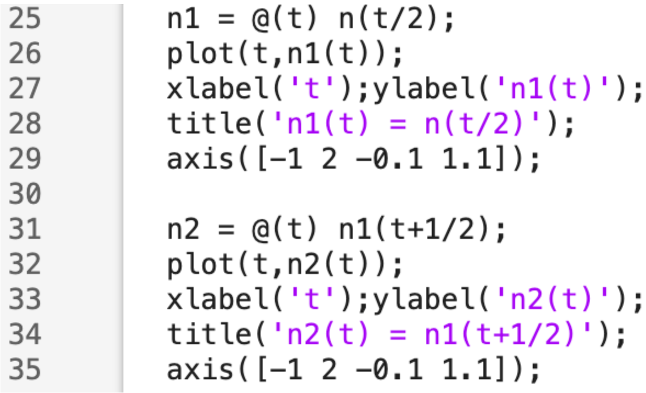
Plots:





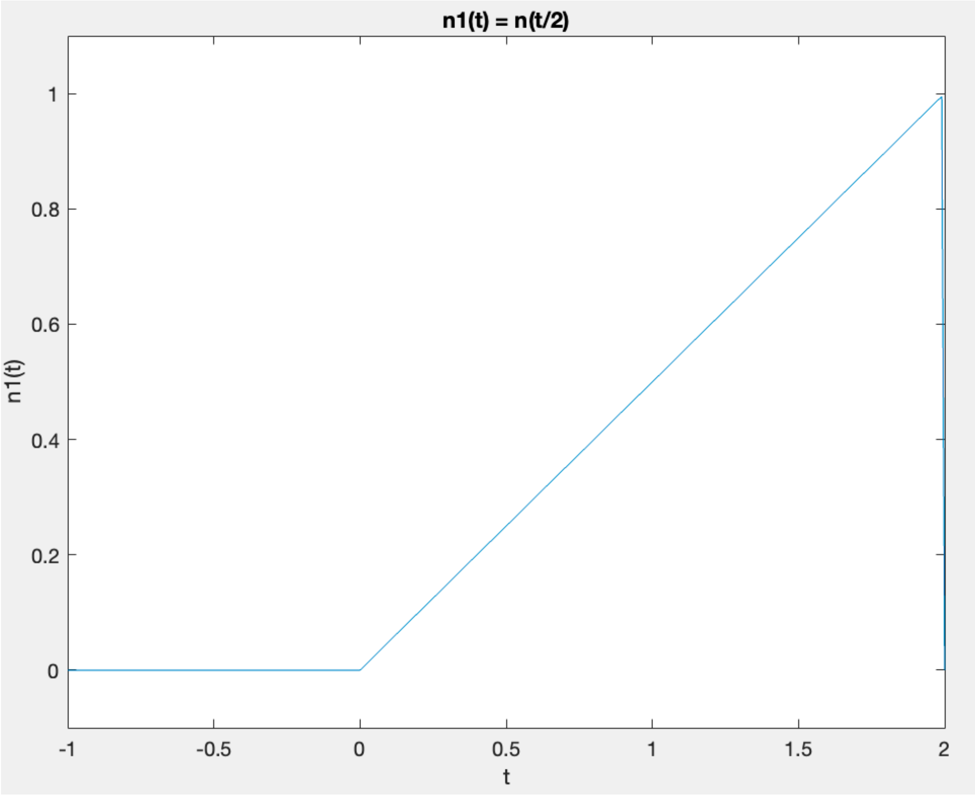
Problem B.3: Plot the following two signals: $n1(t) = n\left(\frac{1}{2}t\right), n2(t) = n1\left(t + \frac{1}{2}\right)$.

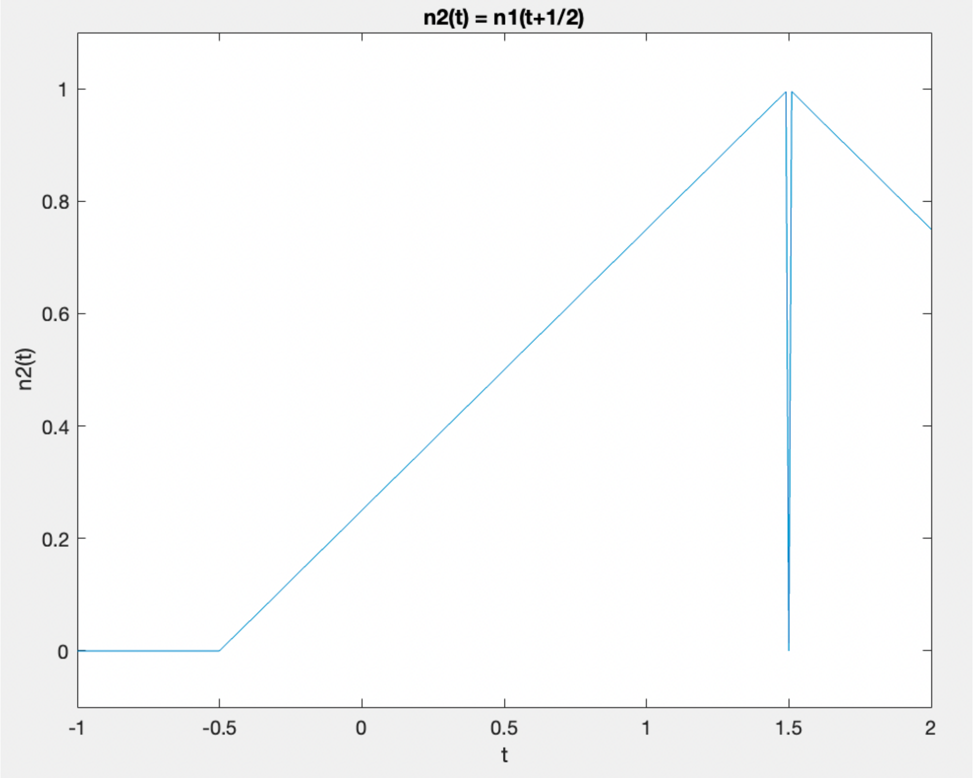
Code:



```
n1 = @(t) n(t/2);
plot(t,n1(t));
xlabel('t');ylabel('n1(t)');
title('n1(t) = n(t/2)');
axis([-1 2 -0.1 1.1]);

n2 = @(t) n1(t+1/2);
plot(t,n2(t));
xlabel('t');ylabel('n2(t)');
title('n2(t) = n1(t+1/2)');
axis([-1 2 -0.1 1.1]);
```

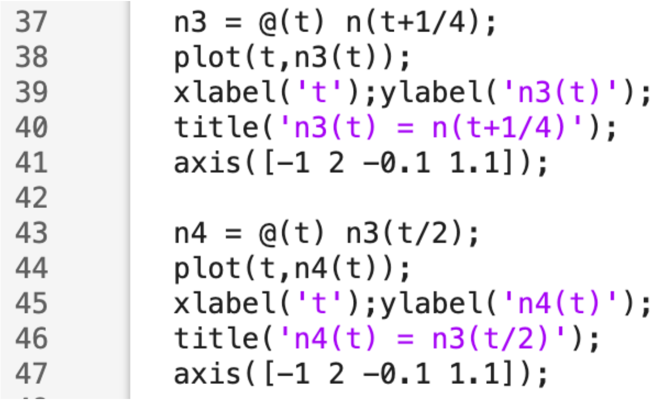
Plots:


n1(t) = n(t/2)
n1(t)
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2
t


n2(t) = n1(t+1/2)
n2(t)
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1
1.5
2
t

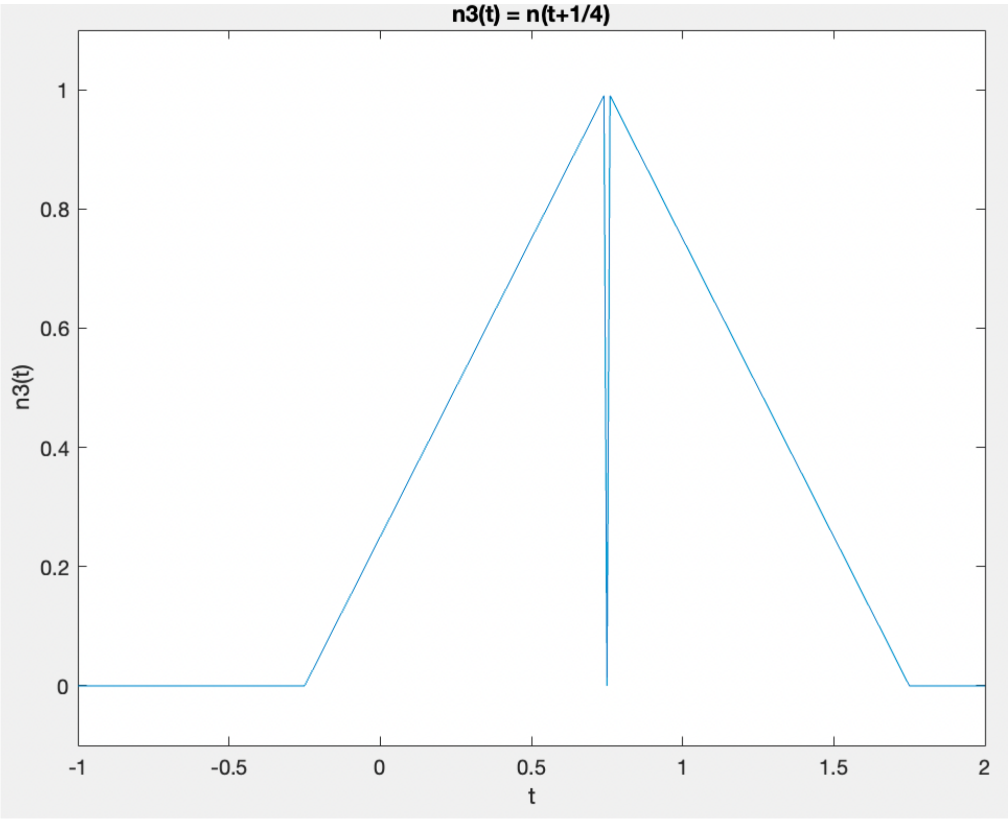
Problem B.4: Plot the following two signals: $n3(t) = n\left(t + \frac{1}{4}\right), n4(t) = n3\left(\frac{1}{2}t\right)$.

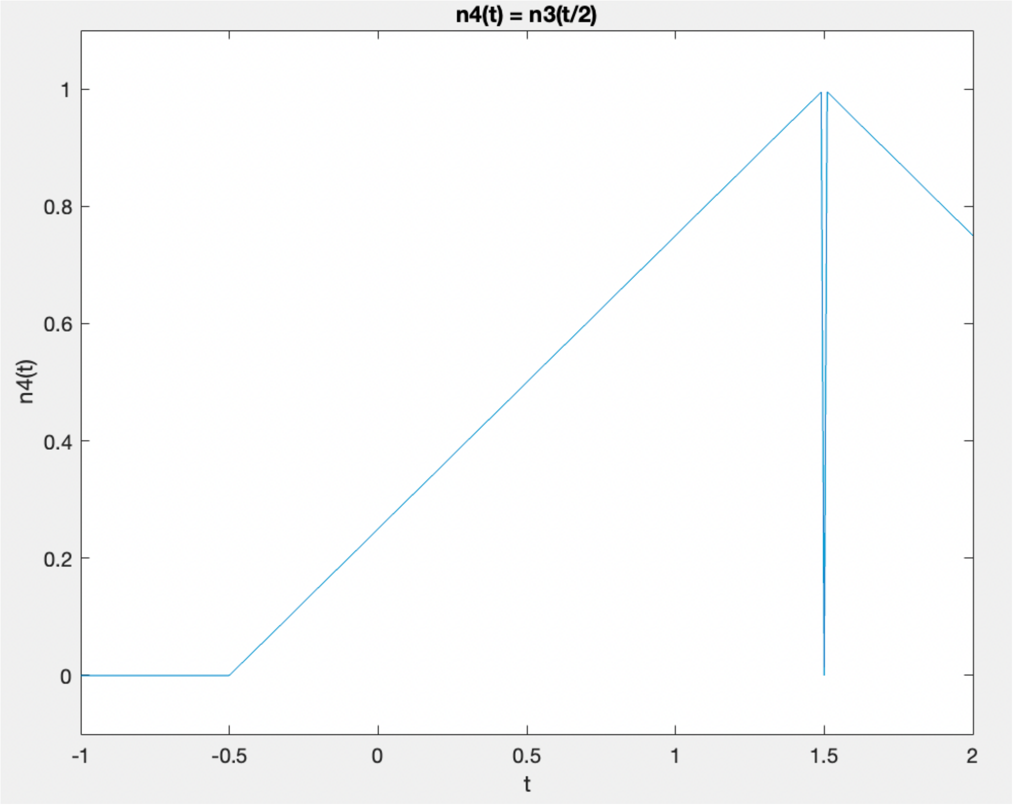
Code:



```
n3 = @(t) n(t+1/4);
plot(t,n3(t));
xlabel('t');ylabel('n3(t)');
title('n3(t) = n(t+1/4)');
axis([-1 2 -0.1 1.1]);

n4 = @(t) n3(t/2);
plot(t,n4(t));
xlabel('t');ylabel('n4(t)');
title('n4(t) = n3(t/2)');
axis([-1 2 -0.1 1.1]);
```

Plots:



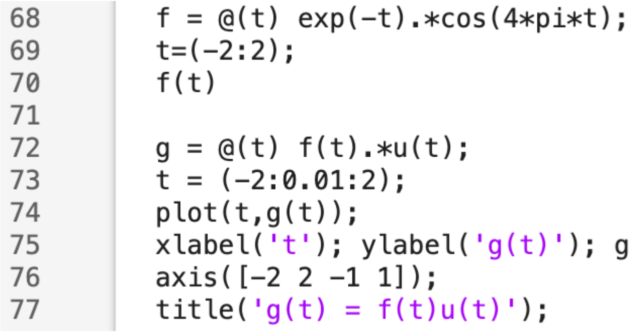


Problem B.5: Compare n4(t) and n2(t); explain any observed differences and/or similarities.

C. Visualization operations on the independent variable and algorithm vectorization.

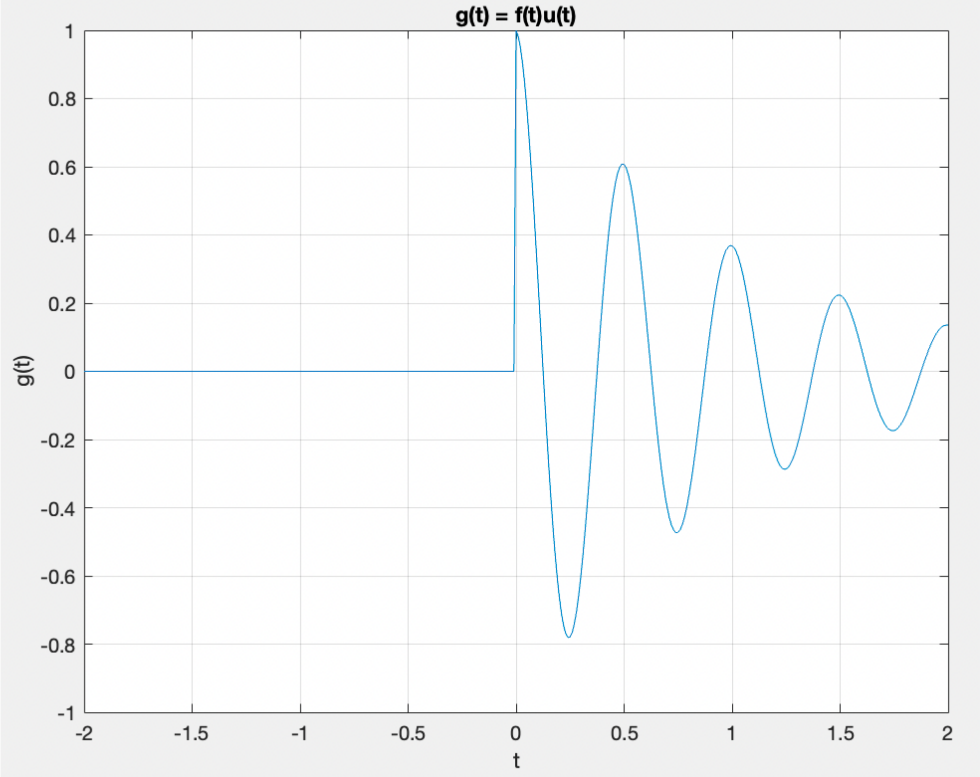
Problem C.1: Follow the steps in Section 1.11-3, but instead generate g(t) = f(t)u(t) where $f(t) = e^{-2t}\cos(4\pi t)$.

Code:



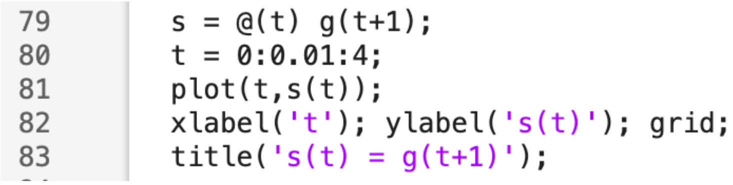
```
f = @(t) exp(-t).*cos(4*pi*t);
t=(-2:2);
f(t)

g = @(t) f(t).*u(t);
t = (-2:0.01:2);
plot(t,g(t));
xlabel('t'); ylabel('g(t)'); grid;
axis([-2 2 -1 1]);
title('g(t) = f(t)u(t)');
```

Plot:



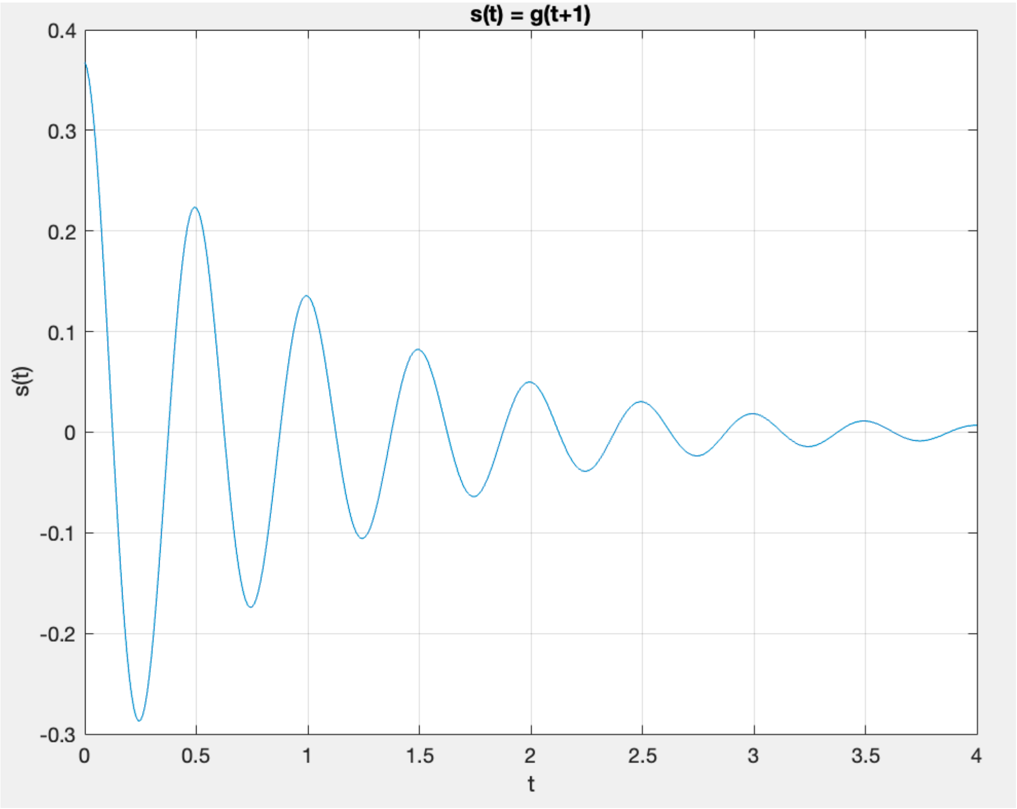
Problem C.2: Using g(t) as described in Problem C.1, generate and plot s(t) = g(t+1) for t = [0:0.01:4].

Code:



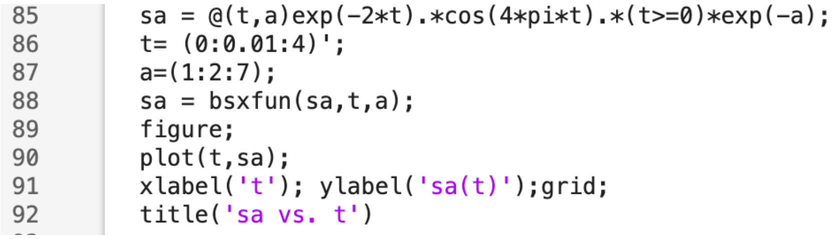
```
s = @(t) g(t+1);
t = 0:0.01:4;
plot(t,s(t));
xlabel('t'); ylabel('s(t)'); grid;
title('s(t) = g(t+1)');
```

Plot:



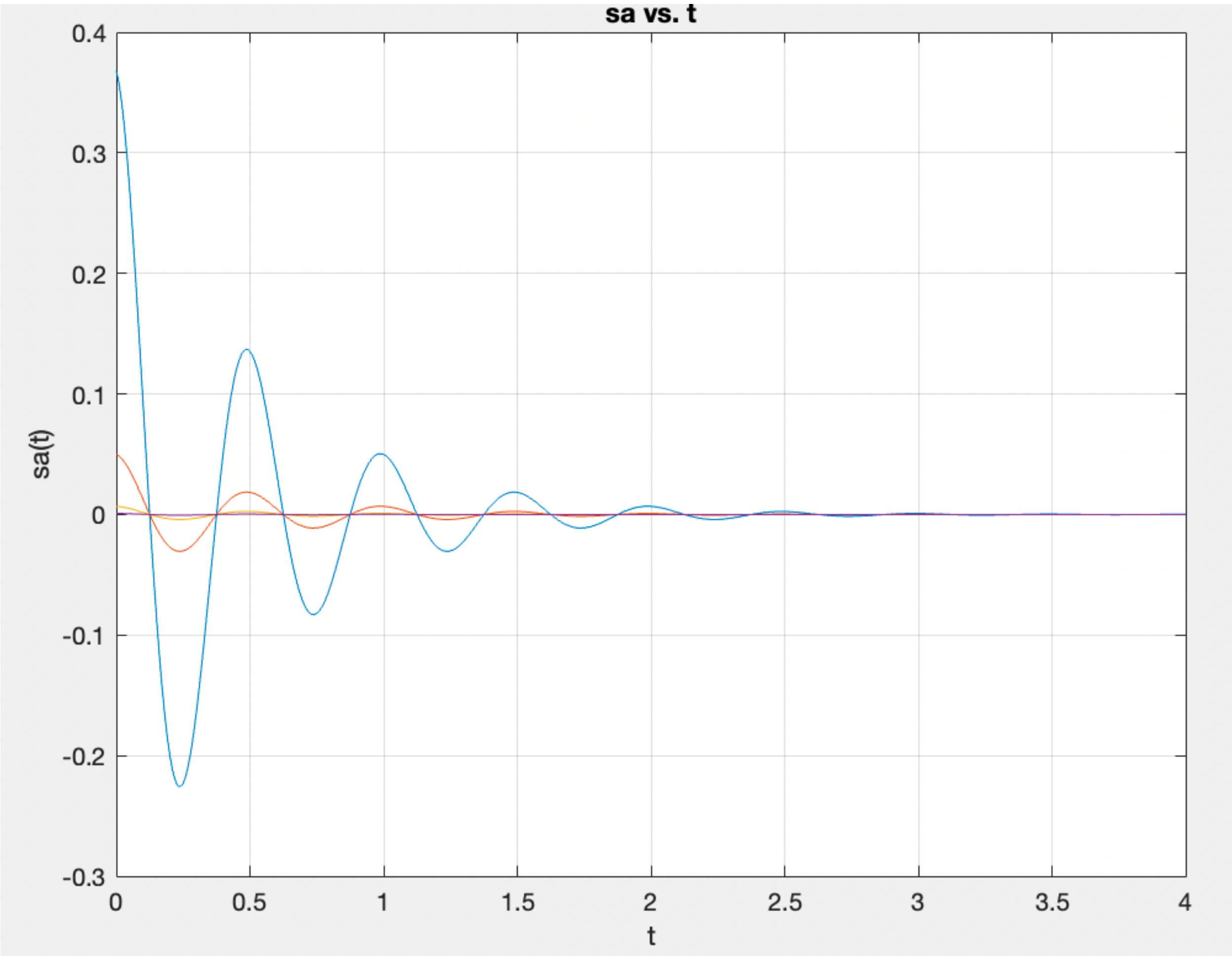
Problem C.3: Plot $s\alpha(t) = e^{-2}e^{-\alpha t}\cos(4\pi t)\,u(t)\;for\;\alpha \in \{1,3,5,7\}$ in one figure for t = [0:0.01:4]. For this plot you can use the for command for a loop structure. Also try to use matrix and vector operations to generate and plot the desired functions by following the steps of Section B.7-6, page 49.

Code:



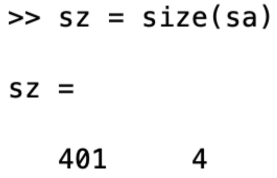
```
sa = @(t,a)exp(-2*t).*cos(4*pi*t).*(t>=0)*exp(-a);
t= (0:0.01:4)';
a=(1:2:7);
sa = bsxfun(sa,t,a);
figure;
plot(t,sa);
xlabel('t'); ylabel('sa(t)');grid;
title('sa vs. t')
```

Plot:



Problem C.4: Determine the size of the matrix s(t) generated in Problem C.3.

The size of the matrix in Problem C.3 is a 401 x 4 matrix, this is found inputting: sz = size(sa) into the command window and it returns the dimensions of sa.



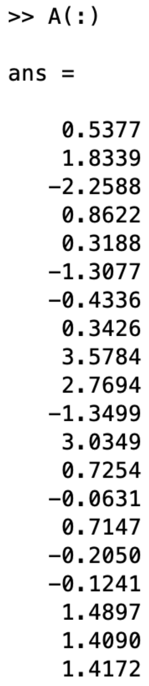
```
>> sz = size(sa)

sz =

   401     4
```

D. Array Indexing

Problem D.1: Let A by a 5 x 4 matrix array with real valued elements (refer to matrix A in lab manual). For matrix A in Equation (1) implement the following operations:

(a) A (:)

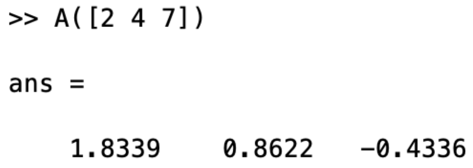
This operation just lists all the elements in matrix A into one column.



```
>> A(:)

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) A ([2 4 7]

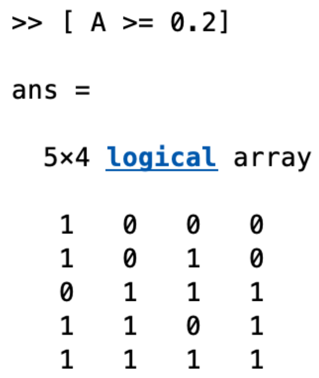
This operation creates a matrix with 3 elements in position 2, 4 and 7 of matrix A, the positions go from the first column down → second column and down…all the way to the last column and down.



```
>> A([2 4 7])

ans =

    1.8339    0.8622   -0.4336
```

(c) [A >= 0.2]

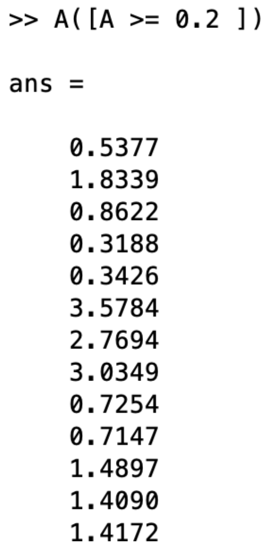
This operation creates a logical array, in the array the value is 1 if the element in the matrix is >= 0.2 and 0 if the element in the array is < 0.2.



```
>> [ A >= 0.2]

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) A ([A >= 0.2])

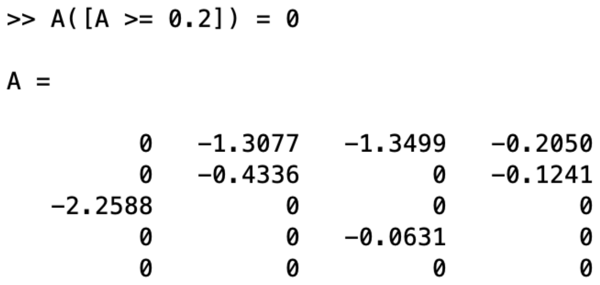
This operation lists all the elements in matrix A that are >= 0.2 into one column.



```
>> A([A >= 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) A ([A >= 0.2]) = 0

This operation finds all the values in matrix A that are >= 0 and set those elements to equal to 0. The elements < 0.2 remain the same while the other elements are 0 in the resulting matrix.

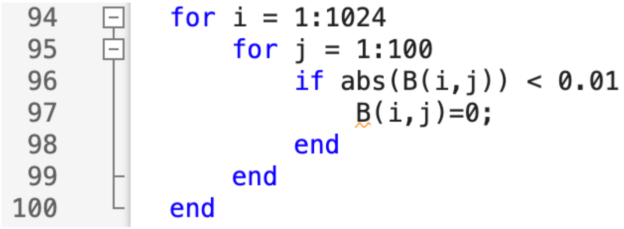


```
>> A([A >= 0.2]) = 0

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
```

Problem D.2: Let B be a 1024 x 100 data matrix representing 100 blocks of non-overlapping 1024-element input samples from a particular data source.

(a) Write a simple MATLAB program using two nested loops that will set all elements of the data matrix B with magnitude values below 0.01 to zero: B(i, j) = 0, if |B(i, j)| < 0.01, where B(i, j) is element of data matrix B in i-th row and j-th column.

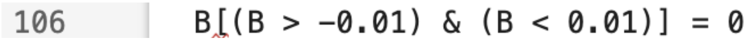
Code:



```
for i = 1:1024
    for j = 1:100
        if abs(B(i,j)) < 0.01
            B(i,j)=0;
        end
    end
end
```

(b) Repeat part (a) using MATLAB’s indexing features as described in Problem D.1.

Code:



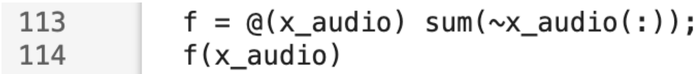
```
B[(B > -0.01) & (B < 0.01)] = 0
```

(c) Use the MATLAB commands tic and toc to compare the execution time of the code you wrote in parts (a) and (b).

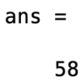
The execution time of the code written in part (a) is 0.000611 seconds and the execution time of the code written in part (b) is 0.076975 seconds. The execution time of the nested for loop is faster than using MATLAB’s indexing features.

Problem D.3: Let x_audio be a 20,000 sample-long row vector representing 2.5 sec of an audio signal sampled at 8 kHz. A simple data compression algorithm can be implemented by setting all elements of data array x_audio with magnitude values below a threshold to zero.

Code:



```
f = @(x_audio) sum(~x_audio(:));
f(x_audio)
```



```
ans =

    58
```