Marwin B. Alejo 2020-20221 EE274_ProgEx01

Table of Contents

- Date Performed (d/m/y): 19/09/2020
- NOTE: for variable naming pattern, all cells that begin with A belongs to codes in section A and vis-a-vis for B to F.

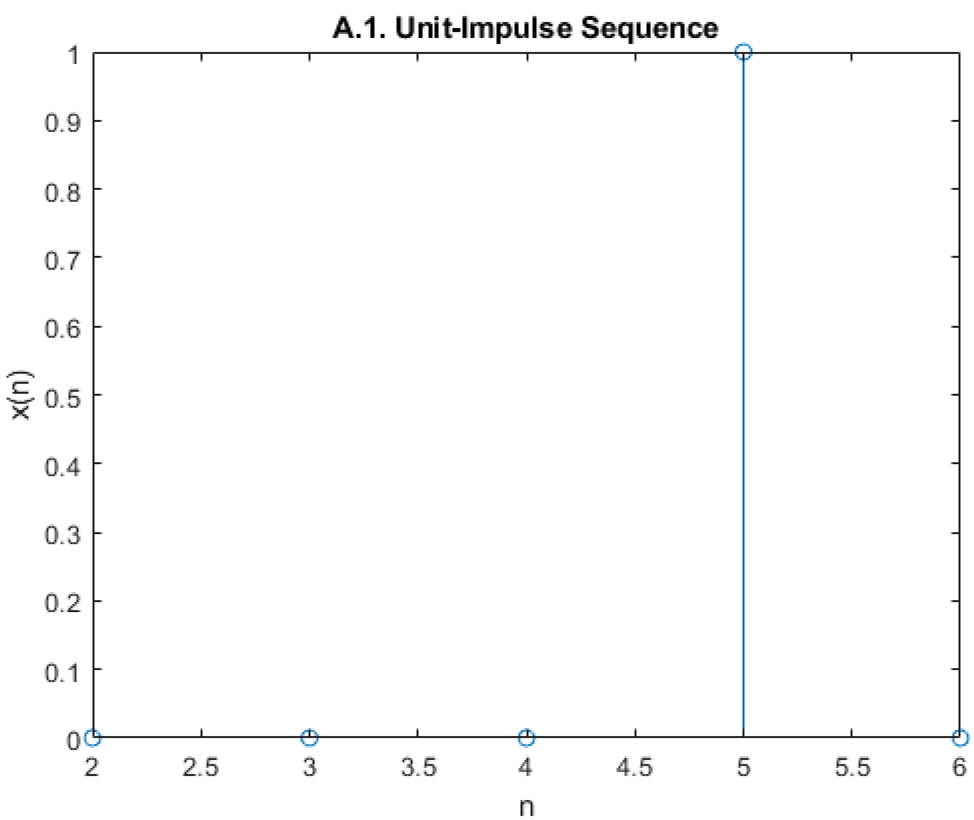
A. SIGNAL GENERATION

1. Unit-impulse sequence generation

```
%This function does generate unit-impulse sequence.
%     function [x,n]=impseq(n0,n1,n2)
```

```
%         n=[n1:n2]; x=[(n-n0)==0];
%         stem(n,x); title('Unit-Impulse Sequence');..
%            xlabel('time'); ylabel('amplitude');
%     end

[Ax_1,An_1]=impseq(5,2,6)
figure(1);
stem(An_1,Ax_1); title('A.1. Unit-Impulse Sequence'); xlabel('n');...
    ylabel('x(n)');
```

```
Ax_1 =

  1×5 logical array

   0   0   0   1   0


An_1 =

     2     3     4     5     6
```

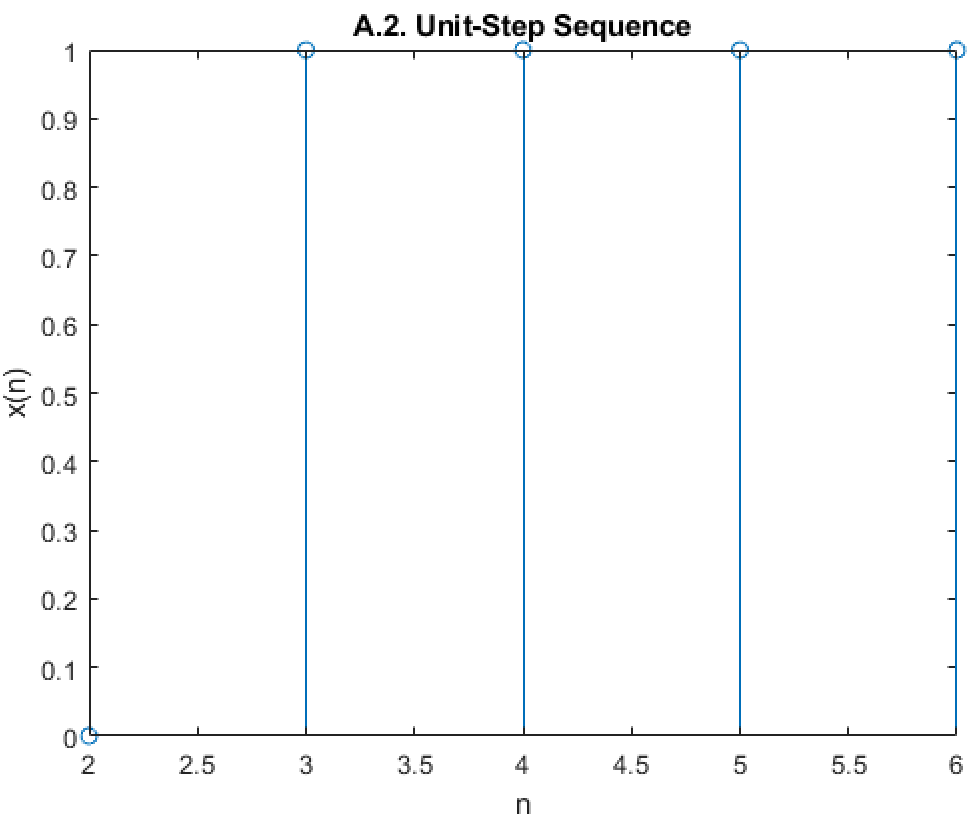


2. Unit-step sequence generation

```
% This function does generate unit-step sequence.
```

```
%     function [x,n]=stepseq(n0,n1,n2)
%         n=[n1:n2]; x=[(n-n0)>=0];
%     end
[Ax_2,An_2]=stepseq(3,2,6)
figure(2);
stem(An_2,Ax_2); title('A.2. Unit-Step Sequence'); xlabel('n');...
    ylabel('x(n)');
```

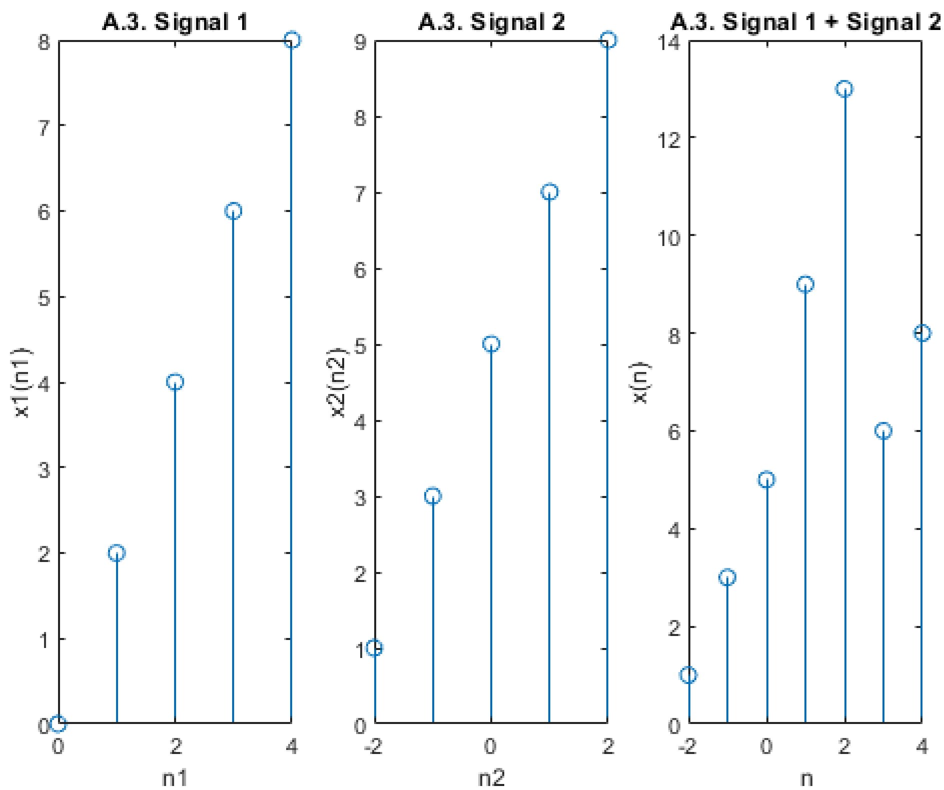
```
Ax_2 =

  1×5 logical array

   0   1   1   1   1


An_2 =

     2     3     4     5     6
```



3. Addition of two sequence

```
% This function does add two discrete time sequences.
%     function [y,n]=sigadd(x1,n1,x2,n2)
%         n=min(min(n1),min(n2)): max(max(n1),max(n2));
```

```
%           y1=zeros(1,length(n)); y2=y1;
%           y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;
%           y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;
%           y=y1+y2;
%       end

Ax1_3=[0,2,4,6,8];
An1_3=[0,1,2,3,4];
Ax2_3=[1,3,5,7,9];
An2_3=[-2,-1,0,1,2];
[Ay_3,An_3]=sigadd(Ax1_3,An1_3,Ax2_3,An2_3)
figure(3);
subplot(1,3,1); stem(An1_3,Ax1_3); title('A.3. Signal 1');...
    xlabel('n1');ylabel('x1(n1)');
subplot(1,3,2); stem(An2_3,Ax2_3); title('A.3. Signal 2');...
    xlabel('n2');ylabel('x2(n2)');
subplot(1,3,3); stem(An_3,Ay_3); title('A.3. Signal 1 + Signal 2');...
    xlabel('n');ylabel('x(n)');


Ay_3 =

     1     3     5     9    13     6     8


An_3 =

    -2    -1     0     1     2     3     4
```

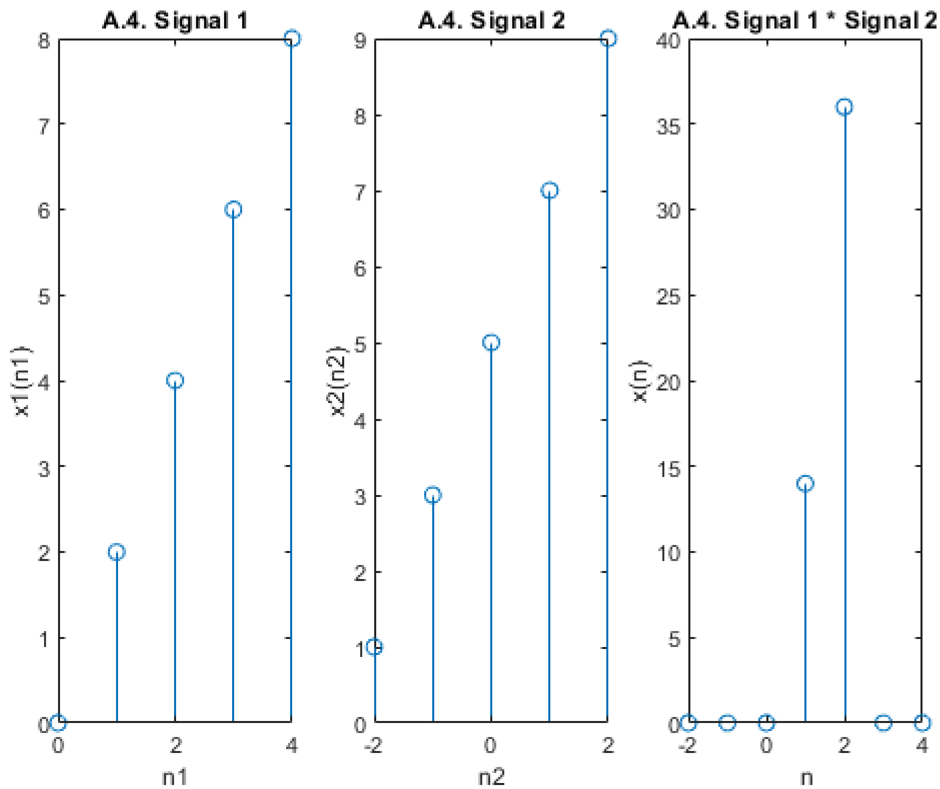


4. Multiplication of two sequences

```
% This function does multiply two discrete time sequences.
%     function [y,n]=sigadd(x1,n1,x2,n2)
%         n=min(min(n1),min(n2)): max(max(n1),max(n2));
%         y1=zeros(1,length(n)); y2=y1;
%         y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;
%         y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;
%         y=y1.*y2;
%     end

Ax1_4=[0,2,4,6,8];
An1_4=[0,1,2,3,4];
Ax2_4=[1,3,5,7,9];
An2_4=[-2,-1,0,1,2];
[Ay_4,An_4]=sigmult(Ax1_4,An1_4,Ax2_4,An2_4)
figure(4);
subplot(1,3,1); stem(An1_4,Ax1_4); title('A.4. Signal 1');...
    xlabel('n1');ylabel('x1(n1)');
subplot(1,3,2); stem(An2_4,Ax2_4); title('A.4. Signal 2');...
    xlabel('n2');ylabel('x2(n2)');
subplot(1,3,3); stem(An_4,Ay_4); title('A.4. Signal 1 * Signal 2');...
    xlabel('n');ylabel('x(n)');
```

Ay_4 =

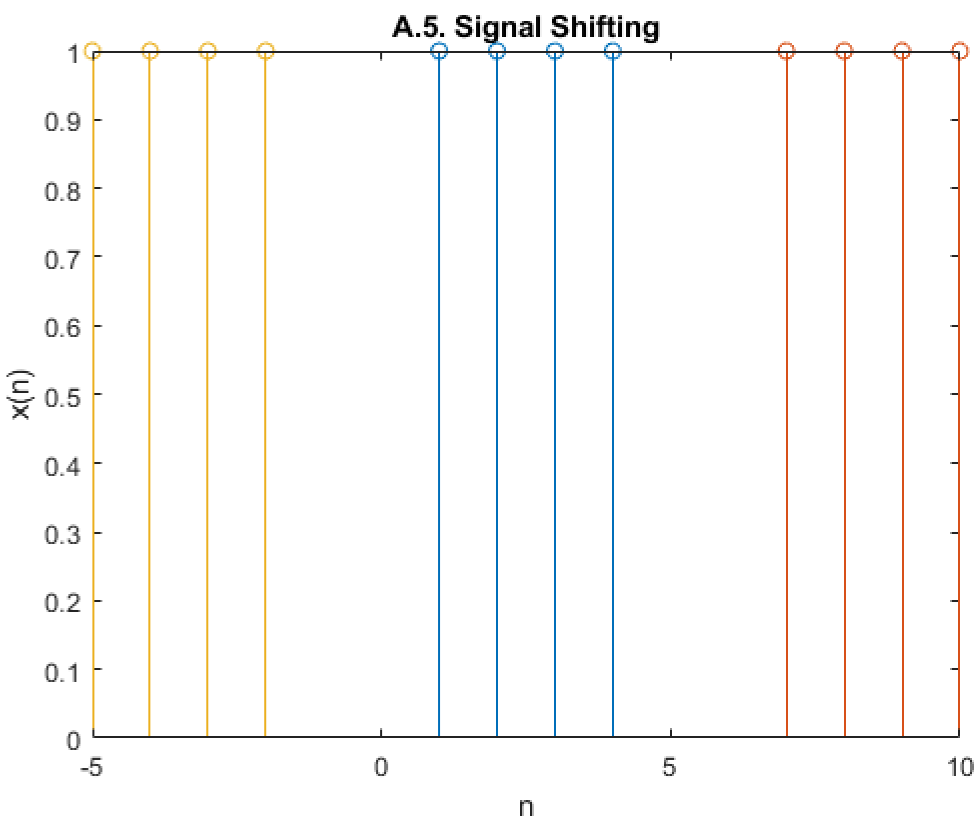
```
     0     0     0    14    36     0     0

An_4 =

    -2    -1     0     1     2     3     4
```



5. Shifting operation on a sequence

```
% This function that does  signal shifting of a sequence.
%     function [y,n]=sigshift(x,n,n0)
%         n=n+n0; y=x;
%     end

Ax_5=[1,1,1,1];
An_5=[1,2,3,4];
[Ay1_5,An2_5]=sigshift(Ax_5,An_5,6)
[Ay2_5,An3_5]=sigshift(Ax_5,An_5,-6)
figure(5);
stem(An_5,Ax_5),hold on,stem(An2_5,Ax_5),hold on,stem(An3_5,Ax_5),...
    hold off;title('A.5. Signal Shifting'); xlabel('n');
 ylabel('x(n)');
```

```
Ay1_5 =

     1     1     1     1


An2_5 =

     7     8     9    10


Ay2_5 =

     1     1     1     1


An3_5 =

    -5    -4    -3    -2
```

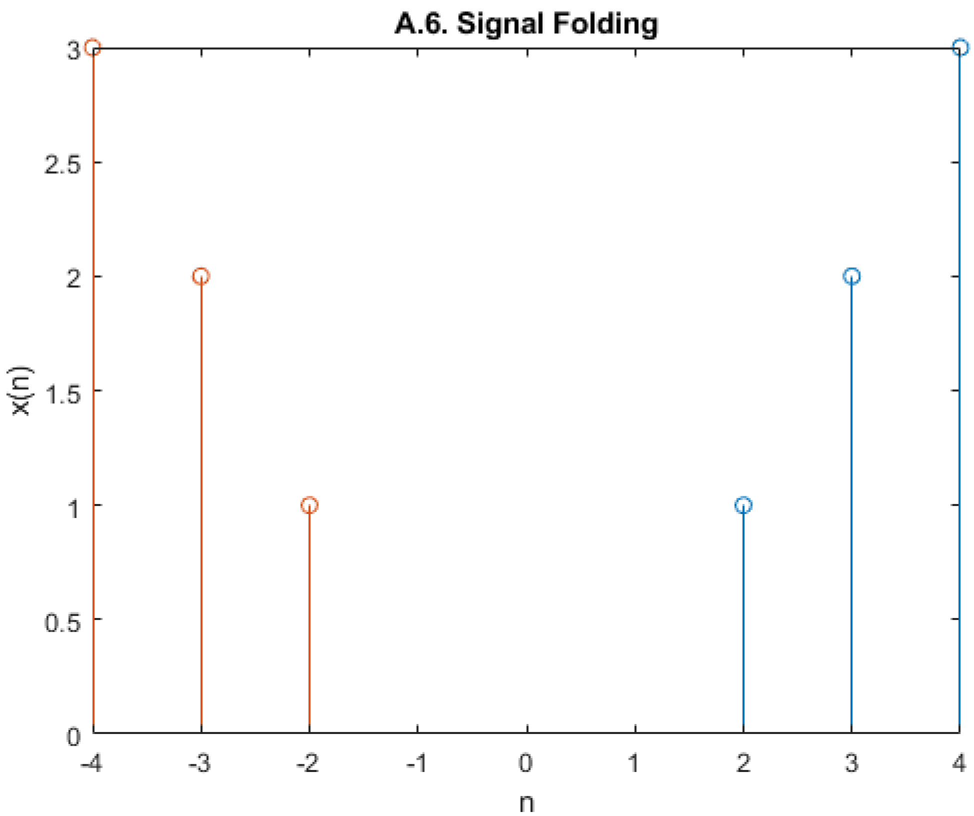


6. Folding operation on a sequence

```
% This function that does  signal folding of a sequence.
%     function [y,n]=sigfold(x,n)
%         y=fliplr(x); n=-fliplr(n);
%     end
```

```
Ax_6=[1,2,3];
An_6=[2,3,4];
[Ay_6,An2_6]=sigfold(Ax_6,An_6)
figure(6);
stem(An_6,Ax_6),hold on,stem(An2_6,Ay_6),hold off;...
    title('A.6. Signal Folding'); xlabel('n'); ylabel('x(n)');
```

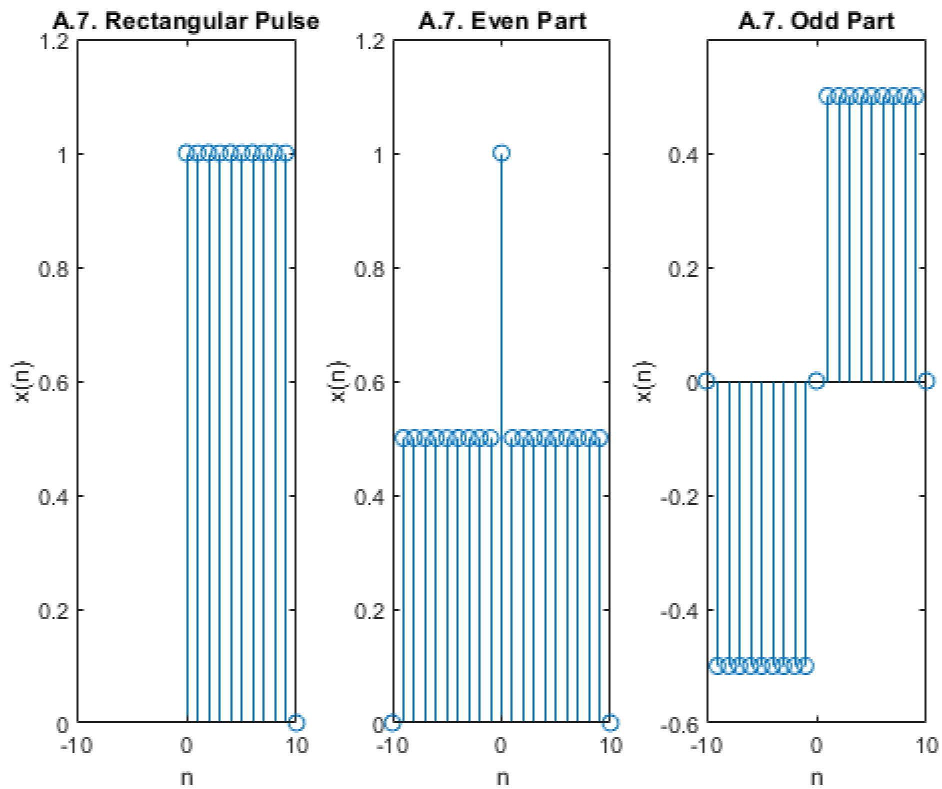
```
Ay_6 =

     3     2     1


An2_6 =

    -4    -3    -2
```



7. Decomposition or real sequence into even and odd components

```
% This function that does  real signal decomposition into even and odd
% of a sequence
%     function [xe, xo, m] = evenodd(x,n)
```

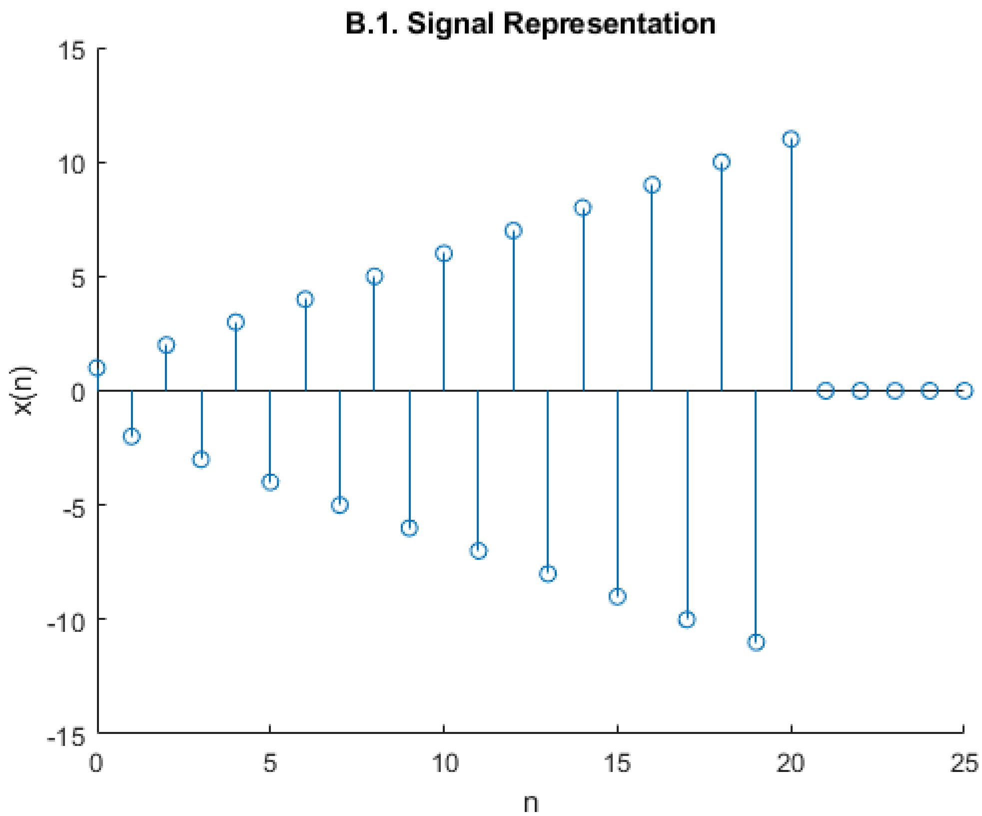
```
%     if any(imag(x) ~= 0)
%         error('x is not a real sequence')
%     end
%     m=-fliplr(n);
%     m1=min([m,n]);
%     m2=max([m,n]);
%     m=m1:m2;
%     nm=n(1)-m(1);
%     n1=1:length(n);
%     x1=zeros(1,length(m));
%     x1(n1+nm)=x;
%     x=x1;
%     xe=0.5*(x+fliplr(x));
%     xo=0.5*(x-fliplr(x));

An_7 =[0:10]; Ax_7=stepseq(0,0,10)-stepseq(10,0,10);
[Axe_7,Axo_7,Am_7]=evenodd(Ax_7,An_7);
figure(7);
subplot(1,3,1);stem(An_7,Ax_7);title('A.7. Rectangular Pulse');...
    xlabel('n');ylabel('x(n)');axis([-10,10,0,1.2]);
subplot(1,3,2);stem(Am_7,Axe_7);title('A.7. Even Part');...
    xlabel('n');ylabel('x(n)');axis([-10,10,0,1.2]);
subplot(1,3,3);stem(Am_7,Axo_7);title('A.7. Odd Part');...
    xlabel('n');ylabel('x(n)');axis([-10,10,-0.6,0.6]);
```



B. SIGNAL REPRESENTATION

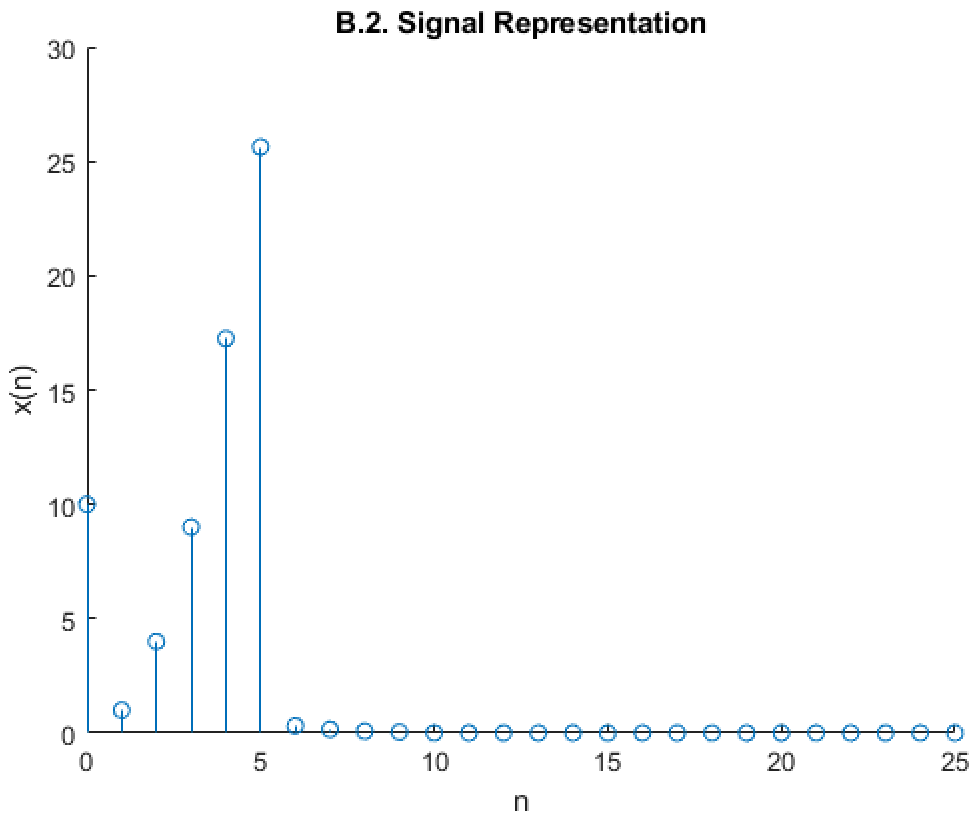
1. $x_1(n) = \sum_{m=0}^{10}(m+1)[\delta(n-2m) - \delta(n-2m-1)] \quad 0 \leq n \leq 25$

```
% The code below is the shorter equivalence of the code under.
% for k=1:1:10
%     stem(n_x1,((k+1)*impseq(2*k,0,25))), hold on, stem(n_x1,...
%        (k+1)*impseq((2.*k)-1,0,25)),hold on, stem(n_x1,...
%        ((k+1)*impseq(2*k,0,25))-((k+1)*impseq(((2.*k)-1),0,25)));
% end
% The following codes below are a direct code conversion of the above
% expresion without using any summation function.

Bn_x_1=[0:25];
Bx_1=(((0+1)*impseq(2*0,0,25))-((0+1)*impseq(((2.*0)-1),0,25)))+...
(((1+1)*impseq(2*1,0,25))-((1+1)*impseq(((2.*1)-1),0,25)))+...
(((2+1)*impseq(2*2,0,25))-((2+1)*impseq(((2.*2)-1),0,25)))+...
(((3+1)*impseq(2*3,0,25))-((3+1)*impseq(((2.*3)-1),0,25)))+...
(((4+1)*impseq(2*4,0,25))-((4+1)*impseq(((2.*4)-1),0,25)))+...
(((5+1)*impseq(2*5,0,25))-((5+1)*impseq(((2.*5)-1),0,25)))+...
(((6+1)*impseq(2*6,0,25))-((6+1)*impseq(((2.*6)-1),0,25)))+...
(((7+1)*impseq(2*7,0,25))-((7+1)*impseq(((2.*7)-1),0,25)))+...
(((8+1)*impseq(2*8,0,25))-((8+1)*impseq(((2.*8)-1),0,25)))+...
(((9+1)*impseq(2*9,0,25))-((9+1)*impseq(((2.*9)-1),0,25)))+...
(((10+1)*impseq(2*10,0,25))-((10+1)*impseq(((2.*10)-1),0,25)));
figure(8), hold on, stem(Bn_x_1,Bx_1), hold off;...
    title('B.1. Signal Representation');xlabel('n');...
    ylabel('x(n)');
```



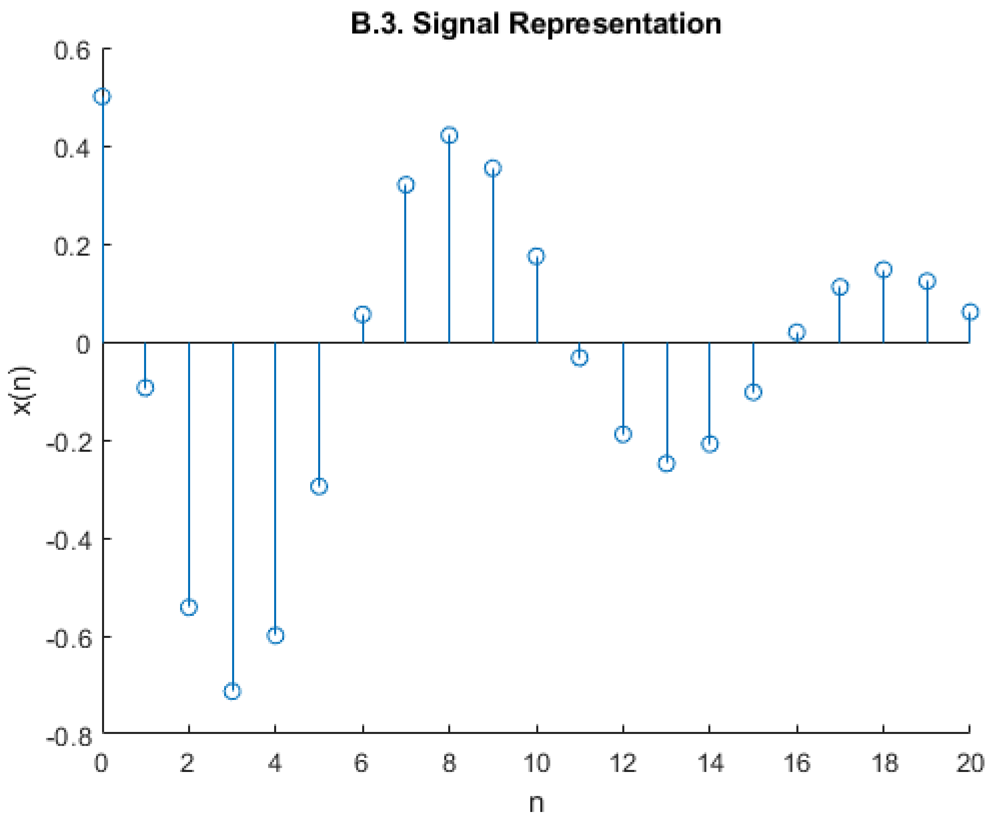
2. $x_2(n) = n^2[u(n+5) - u(n-6)] + 10\delta(n) + 20(0.5)^n[u(n-4) - u(n-10)] \quad 0 \le n \le 25$

```
Bn_x_2=[0:25]; %initialize n domain range of function 02.
Bx_2_1=Bn_x_2.^2;
Bx_2_2=(stepseq(-5,0,25))-(stepseq(6,0,25));
Bx_2_3=10*impseq(0,0,25);
Bx_2_4=20*(0.5.^Bn_x_2);
Bx_2_5=(stepseq(4,0,25))-(stepseq(10,0,25));
Bx_2_12=sigmult(Bx_2_1,Bn_x_2,Bx_2_2,Bn_x_2);
Bx_2_45=sigmult(Bx_2_4,Bn_x_2,Bx_2_5,Bn_x_2);
Bx_2_123=sigadd(Bx_2_12,Bn_x_2,Bx_2_3,Bn_x_2);
Bx_2=sigadd(Bx_2_123,Bn_x_2,Bx_2_45,Bn_x_2);
figure(9), hold on, stem(Bn_x_2,Bx_2),hold off;...
    title('B.2. Signal Representation');xlabel('n');ylabel('x(n)');
```



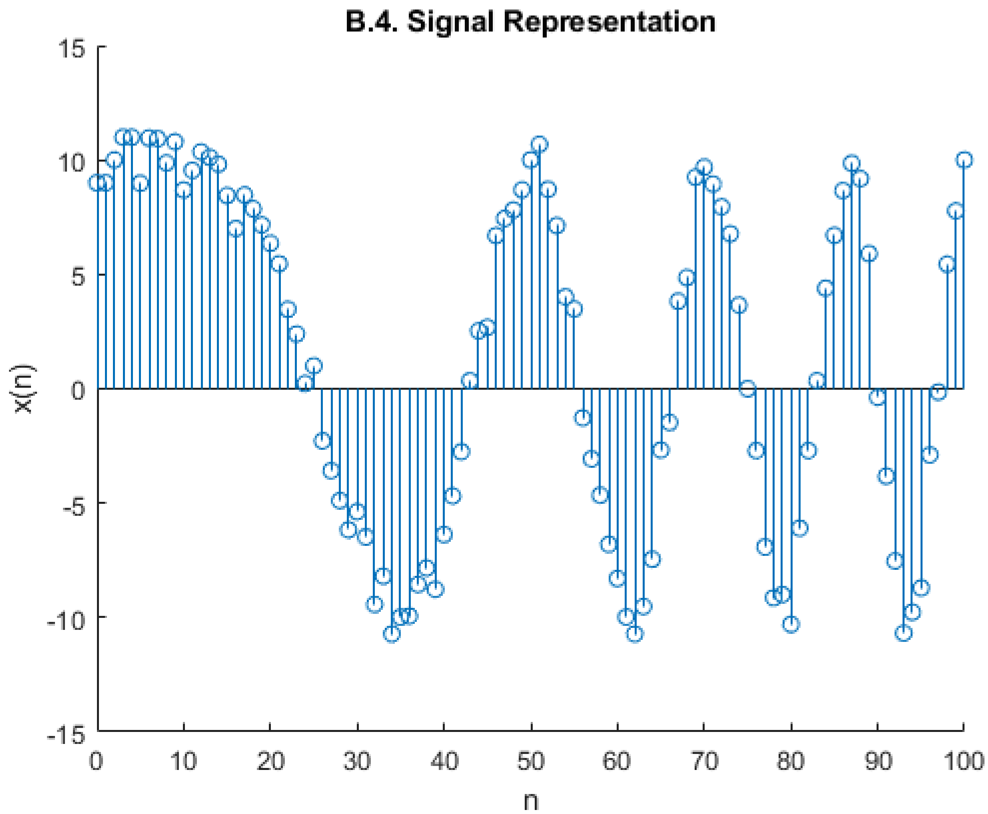
3. $x_3(n) = (0.9)^n \cos(0.2\pi n + \frac{\pi}{3}) \quad 0 \leq n \leq 20$

```
Bn_x_3=[0:20];
Bx_3_1=(0.9).^Bn_x_3;
Bx_3_theta=((0.2*pi*Bn_x_3)+(pi/3));
Bx_3_2=cos(Bx_3_theta);
Bx_3=sigmult(Bx_3_1,Bn_x_3,Bx_3_2,Bn_x_3);
figure(10), hold on, stem(Bn_x_3,Bx_3), hold off;...
    title('B.3. Signal Representation');xlabel('n');ylabel('x(n)');
```



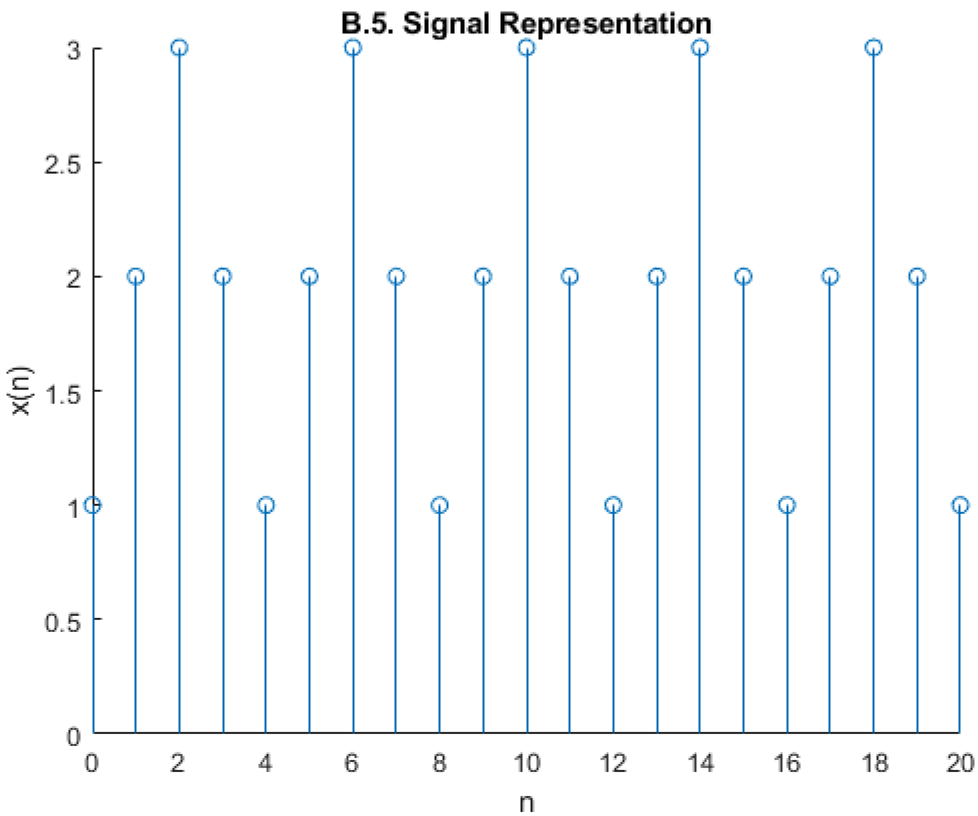
4. $x_4(n) = 10\cos(0.0008\pi n^2) + w(n) \quad 0 \le n \le 100$

```
Bn_x_4=[0:100];
Bx_4_theta=(0.0008*pi*Bn_x_4.^2);
Bx_4_1=10*cos(Bx_4_theta);
Bw_n=randi([-1 1],1,101);
Bx_4=sigadd(Bx_4_1,Bn_x_4,Bw_n,Bn_x_4);
figure(11), hold on, stem(Bn_x_4,Bx_4), hold off;...
    title('B.4. Signal Representation');xlabel('n');ylabel('x(n)');
```



5. $x_5(n) = ..., 1, 2, 3, 2, 1, 2, 3, 2, ... \quad 0 \leq n \leq 20$

```
Bx_5_t=(repmat([1 2 3 2],1,6)); Bn_x_5=[0:20];
B_arr_size=size(Bn_x_5);
Bx_5=Bx_5_t(B_arr_size(1):B_arr_size(2));
figure(12), hold on, stem(Bn_x_5,Bx_5), hold off;...
    title('B.5. Signal Representation');xlabel('n');ylabel('x(n)');
```



C. SAMPLING

1-2. Audio file and sample rate information import to workspace.

```
C_af = 'signal1.wav'; %loading signal1.wav to workspace as C_af
[C_y_af,C_fs_af]=audioread(C_af);...
    %init of audio and sample rate data as C_y_af and C_fs_af in
 workspace.
```

3. Audio Resampling

```
C_y1_af = upsample(C_y_af,2); %upsampling y by 2
C_y2_af = downsample(C_y_af,2); %downsampling y by 2
C_y3_af = upsample(C_y2_af,2); %upsampling y2 by 2
C_y4_af = upsample(C_y3_af,2); %upsampling y3 by 2
```

4.1. Is y3 the same as y?

```
whos C_y3_af;
whos C_y_af;
```

```
% By inspection, y3_audio and y_audio are the same in terms of their
 Fs.
```

```
  Name                  Size                  Bytes  Class     Attributes

  C_y3_af        396900x1                3175200  double

  Name                  Size                  Bytes  Class     Attributes

  C_y_af         396900x1                3175200  double
```

4.2. Is y4 the same as y1?

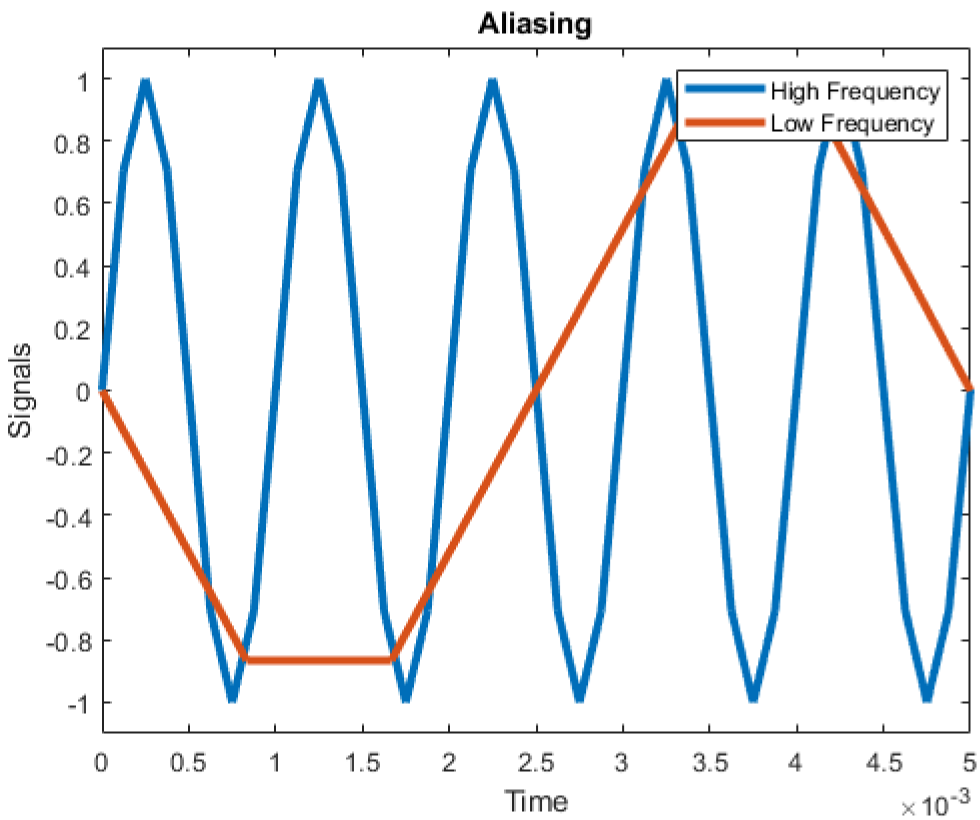
```
whos C_y4_af;
whos C_y1_af;
% By inspection, y4_audio and y1_audio are not the same in terms of
 their
% sample rate information and values');
```

```
  Name                  Size                  Bytes  Class     Attributes

  C_y4_af        793800x1                6350400  double

  Name                  Size                  Bytes  Class     Attributes

  C_y1_af        793800x1                6350400  double
```

D. ALIASING

1-2. Generating two 1kHz 2s sine signals with 8kHz and 1.2kHz fs.

```
D_fs_1 = 8000;
D_fs_2 = 1200;
[D_sw_1, D_t_1] = sin_uni(D_fs_1,1000,2);
[D_sw_2, D_t_2] = sin_uni(D_fs_2,1000,2);
figure(13);
plot(D_t_1,D_sw_1,D_t_2,D_sw_2,'LineWidth',3.0),
axis([0, 0.005, -1.1, 1.1])
legend('High Frequency','Low Frequency')
xlabel('Time')
ylabel('Signals')
title('Aliasing');
```



3. Listening to two signals one after another using soundsc(x,fs).

```
soundsc(D_sw_1,D_fs_1);
soundsc(D_sw_2,D_fs_2);
```

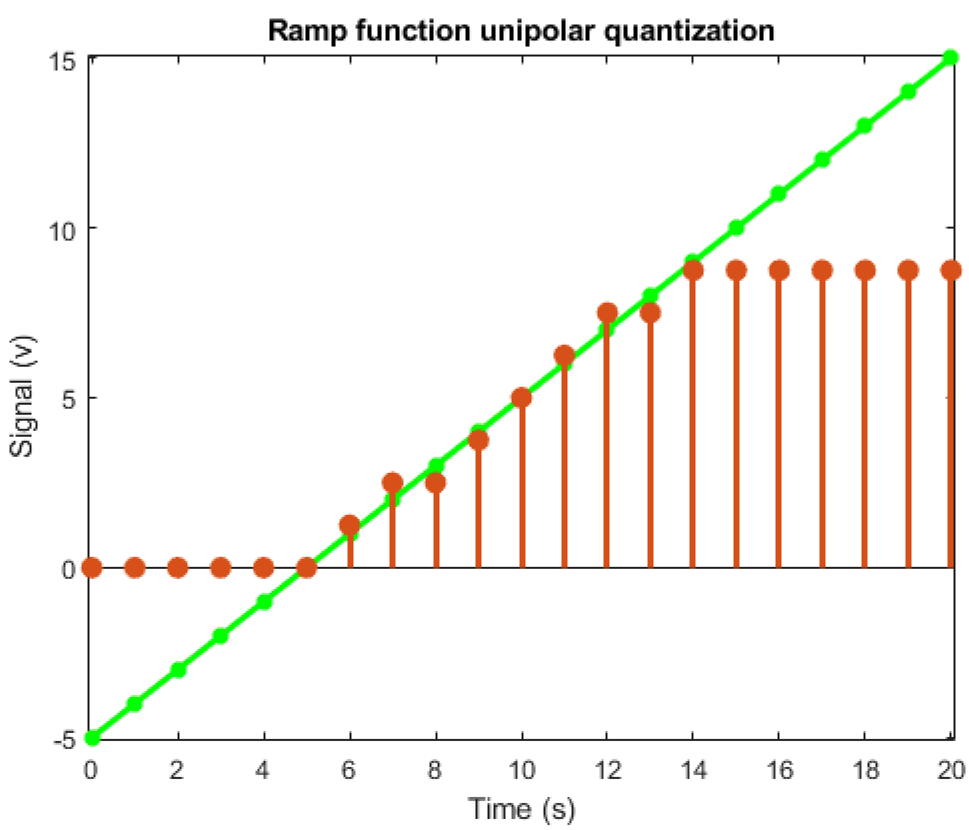
4. Compare the two sine signals. How does the sampling rate affect the digitized signal?

```
% There exist a little difference in the produced sound after sampling
% from the two sine signal. As per the sampling theorem, the frequency
% of the two sine signals should be twice more than the signal
% frequency. The maximum spectrum that can be heard from 8kHz sampling
% frequency should be 4kHz and 600Hz for 1.2kHz. Since the initial
% frequency is 1kHz, it cannot be heard with original 1.2kHz sampling
% rate thus, only the 600Hz is heard. Contrary to 8kHz frequency
% sampling rate, the sound is heard as original and no issues.
```

E. QUANTIZATION

```
%     function y = adc_uni(x, R, B)
```

```
%           level = [0:R/(2^B):R-R/(2^B)];
%           temp = [-Inf,(level(2:end)-R/(2^(B+1))),Inf];
%           y = zeros(1,length(x));
%           for i = 1:length(level)
%               y = y + (x >= temp(i)).*(x < temp(i+1)).*level(i);
%           end

E_R = 10;
E_B = 3;
E_x = -5:15;
E_y = adc_uni(E_x,E_R,E_B);
E_t = 0:length(E_x)-1;
figure(14)
plot(E_t,E_x,E_t,E_y)
plot(E_t,E_x,'g-*','LineWidth',2.2), hold on,...
    stem(E_t,E_y,'filled','LineWidth',2.2),...
    hold off; title('Ramp function unipolar quantization');
xlabel('Time (s)'); ylabel('Signal (v)')
axis([-0.1,20.1,-5.1,15.1])
```



F. AUDIO FILE FORMATS

1-4. Load music1.flac, quantize, sample, listen using soundsc(), compare, and SQNR computation.

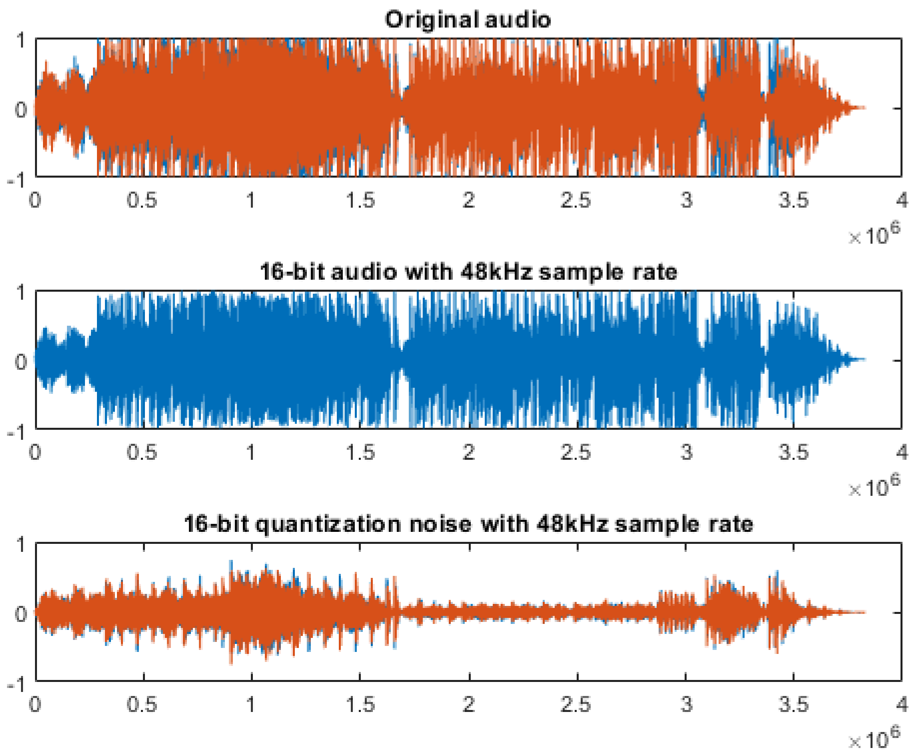
```
F_af = 'music1.flac'; %loading music1.flac to workspace as F_af
[F_y_af,F_fs_af]=audioread(F_af);  %init of audio and sample rate.

% Initialization of sampling rates
F_fs_1=48000; F_fs_2=16000; F_fs_3=8000;

% Initialization of bit rates
F_b_1=16; F_b_2=8; F_b_3=4;

% Sampling and Quantizing of orig audio data (F_y_af) with init
 config.
% For an efficient quantization noise generation, I created a function
% named quanoise() with the code below:
%         function [y,q,sqnr]=quanoise(y_af,bit)
%             y=double(uencode(y_af,bit));
%             y=y/max(y);
%             y=2*(y-0.5);
%             q=y_af-y; %quantization noise
%             sqnr=20*log10(norm(y_af)/norm(q));
%         end

% NOTE: UNCOMMENT ALL soundsc() lines below to listen to the
% quantized+sampled music1.flac.

[F_y_x1,F_y_x1qn,F_sqnr_x1]=quanoise(F_y_af,F_b_1); %Signal x1 16bps
 @48kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x1,F_fs_1); %16-bit audio with 48kHz sample rate
% The 16-bit audio at 48kHz sample rate is audible still but
% slower or at lower rates as compared to the original file.
% soundsc(F_y_x1qn,F_fs_1); %16-bit quantization noise with 48kHz Fs
% The 16-bit audio with a quantized noise at 48kHz sample
% rate moves the same pace as in 16-bit audio without quantized
% noise but inaudible.

figure(15);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x1);...
    title('16-bit audio with 48kHz sample rate');
subplot(3,1,3); plot(F_y_x1qn);...
    title('16-bit quantization noise with 48kHz sample rate');

[F_y_x2,F_y_x2qn,F_sqnr_x2]=quanoise(F_y_af,F_b_2); %Signal x2 8bps
 @48kHz
```

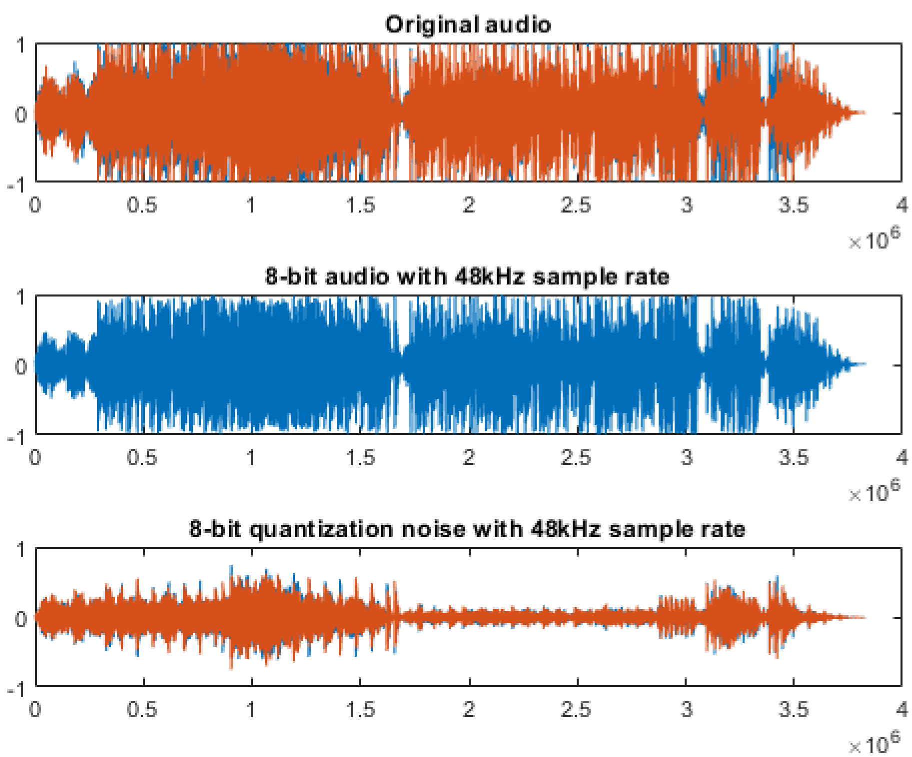
```
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x2,F_fs_1); %8-bit audio with 48kHz sample rate
% The 8-bit audio at 48kHz sample rate is audible still but
% slower or at lower rates than the original file and at
% 16-bit 48kHz config.
% soundsc(F_y_x2qn,F_fs_1); %8-bit quantization noise with 48kHz
% Having an 8-bit quantized noise in the file sampled at
% 48kHz moves like above result but inaudible.
figure(16);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x2);title('8-bit audio with 48kHz sample
 rate');
subplot(3,1,3); plot(F_y_x2qn);...
    title('8-bit quantization noise with 48kHz sample rate');

[F_y_x3,F_y_x3qn,F_sqnr_x3]=quanoise(F_y_af,F_b_3); %Signal x3 4bps
 @48kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x3,F_fs_1); %4-bit audio with 48kHz sample rate
% The original file samples at 48khz 4bps is audible but an
% addition of white noise.
% soundsc(F_y_x3qn,F_fs_1); %4-bit quantization noise with 48kHz
% The quantized 48kHz 4bps audio is inaudible with presence of white
 noise.
figure(17);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x3);...
    title('4-bit audio with 48kHz sample rate');
subplot(3,1,3); plot(F_y_x3qn);...
    title('4-bit quantization noise with 48kHz sample rate');

[F_y_x4,F_y_x4qn,F_sqnr_x4]=quanoise(F_y_af,F_b_1); %Signal x4 16bps
 @16kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x4,F_fs_2); %16-bit audio with 16kHz sample rate
% The audio sampled at 16kHz 16bps is audible at lower level
% going to inaudible and lower rate.
% soundsc(F_y_x4qn,F_fs_2); %16-bit quantization noise with 16kHz Fs
% The audio has tiny sound but inaudible to understand at human-level
figure(18);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x4);title('16-bit audio with 16kHz sample
 rate');
subplot(3,1,3); plot(F_y_x4qn);...
    title('16-bit quantization noise with 16kHz sample rate');

[F_y_x5,F_y_x5qn,F_sqnr_x5]=quanoise(F_y_af,F_b_2); %Signal x4 8bps
 @16kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x5,F_fs_2); %8-bit audio with 16kHz sample rate
% The audio has tiny sound but inaudible to understand at human-level
% soundsc(F_y_x5qn,F_fs_2); %8-bit quantization noise with 16kHz Fs
%The audio has tiny sound but inaudible to understand at
%human-level
```

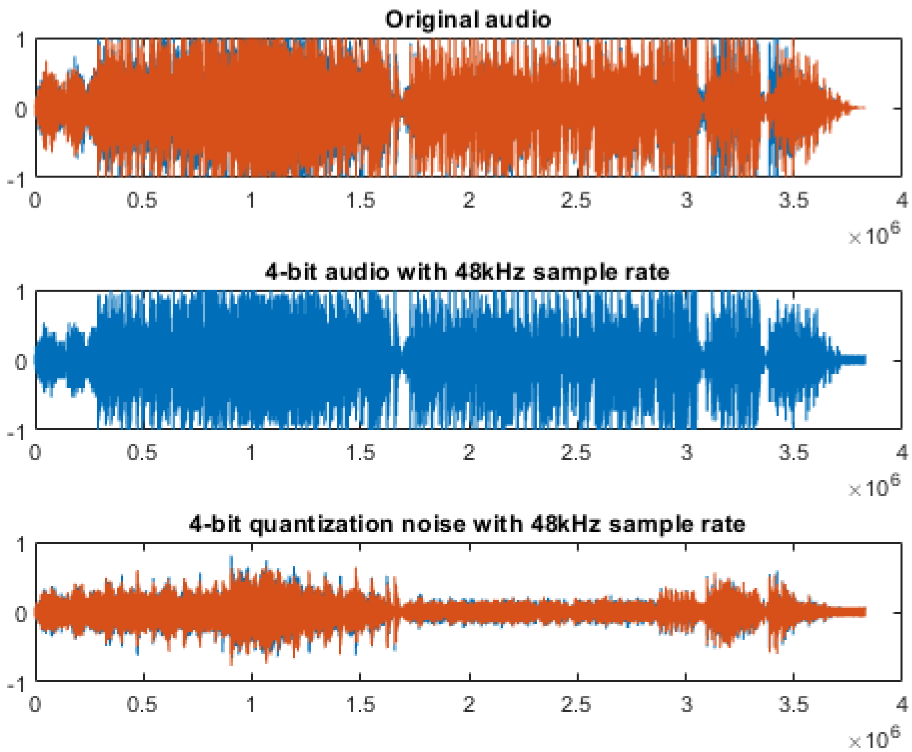
```
figure(19);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x5);title('8-bit audio with 16kHz sample
 rate');
subplot(3,1,3); plot(F_y_x5qn);...
    title('8-bit quantization noise with 16kHz sample rate');

[F_y_x6,F_y_x6qn,F_sqnr_x6]=quanoise(F_y_af,F_b_3); %Signal x6 4bps
 @16kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x6,F_fs_2); %4-bit audio with 16kHz sample rate
% The audio has tiny sound but inaudible to understand at human-level
% soundsc(F_y_x6qn,F_fs_2); %4-bit quantization noise with 16kHz Fs
% The audio has tiny sound but inaudible to understand at human-level
figure(20);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x1);title('4-bit audio with 16kHz sample
 rate');
subplot(3,1,3); plot(F_y_x1qn);...
    title('4-bit quantization noise with 16kHz sample rate');

[F_y_x7,F_y_x7qn,F_sqnr_x7]=quanoise(F_y_af,F_b_1); %Signal x7 16bps
 @8kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x7,F_fs_3); %16-bit audio with 8kHz sample rate
% The audio is inaudible except for quiet white noise at human-level
% soundsc(F_y_x7qn,F_fs_3); %16-bit quantization noise with 8kHz Fs
%The audio has tiny sound but inaudible to understand at human-level
figure(21);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x7);title('16-bit audio with 8kHz sample
 rate');
subplot(3,1,3); plot(F_y_x7qn);...
    title('16-bit quantization noise with 8kHz sample rate');

[F_y_x8,F_y_x8qn,F_sqnr_x8]=quanoise(F_y_af,F_b_2); %Signal x4 8bps
 @16kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x8,F_fs_3); %8-bit audio with 8kHz sample rate
% The audio is inaudible except for quite white noise at human-level
% soundsc(F_y_x8qn,F_fs_3); %8-bit quantization noise with 8kHz Fs
% The audio has tiny sound but inaudible to understand athuman-level
figure(22);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x8);title('8-bit audio with 8kHz sample
 rate');
subplot(3,1,3); plot(F_y_x8qn);...
    title('8-bit quantization noise with 8kHz sample rate');

[F_y_x9,F_y_x9qn,F_sqnr_x9]=quanoise(F_y_af,F_b_3); %Signal x6 4bps
 @16kHz
% soundsc(F_y_af,F_fs_af); %original audio
% soundsc(F_y_x9,F_fs_3); %4-bit audio with 8kHz sample rate
% The audio is inaudible except for tiny white noise at human-level
```

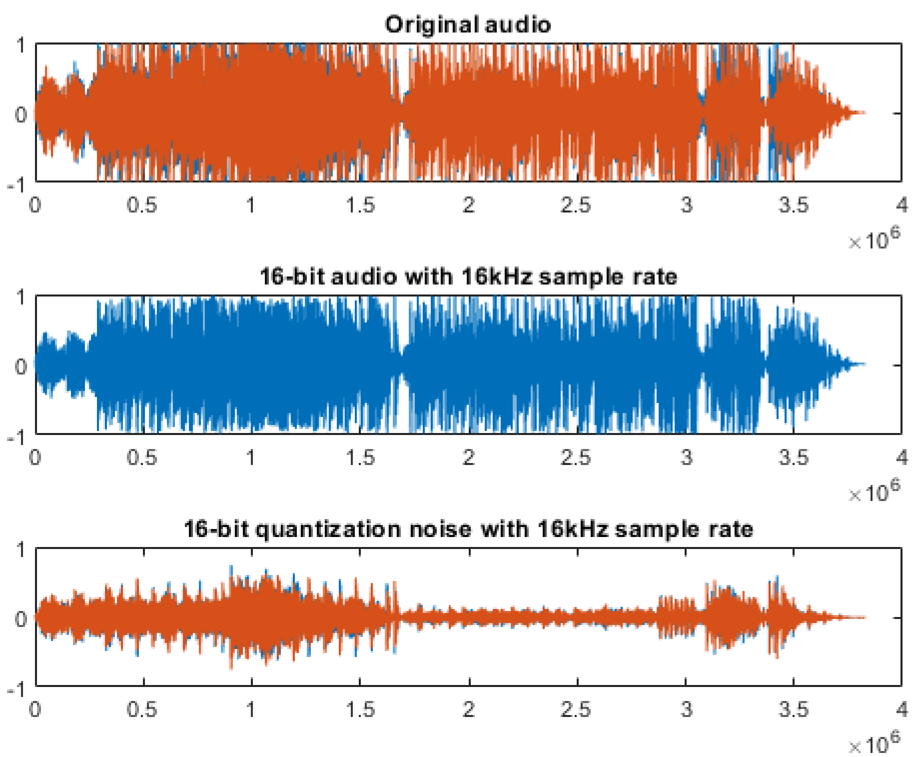
```
% soundsc(F_y_x9qn,F_fs_3); %4-bit quantization noise with 8kHz Fs
% The audio has tiny sound but inaudible to understand at human-level
figure(23);
subplot(3,1,1); plot(F_y_af);title('Original audio');
subplot(3,1,2); plot(F_y_x9);title('4-bit audio with 8kHz sample
 rate');
subplot(3,1,3); plot(F_y_x9qn);...
    title('4-bit quantization noise with 8kHz sample rate');

% The codes below print the SQNR result of each signal above.
% Computation of these SQNR may be seen in quanoise().
disp('Computed SQNR of the signals with respect to the original
 file.');
fprintf('SQNR of x1: %.6f dB\n',F_sqnr_x1);
fprintf('SQNR of x2: %.6f dB\n',F_sqnr_x2);
fprintf('SQNR of x3: %.6f dB\n',F_sqnr_x3);
fprintf('SQNR of x4: %.6f dB\n',F_sqnr_x4);
fprintf('SQNR of x5: %.6f dB\n',F_sqnr_x5);
fprintf('SQNR of x6: %.6f dB\n',F_sqnr_x6);
fprintf('SQNR of x7: %.6f dB\n',F_sqnr_x7);
fprintf('SQNR of x8: %.6f dB\n',F_sqnr_x8);
fprintf('SQNR of x9: %.6f dB\n',F_sqnr_x9);

% SQNR results sbove show that original audio signal is greater
% than the quantized noise signal thus, when mixed together,
% original may result into an audible output but beyond
% human-level of hearing perception.
```

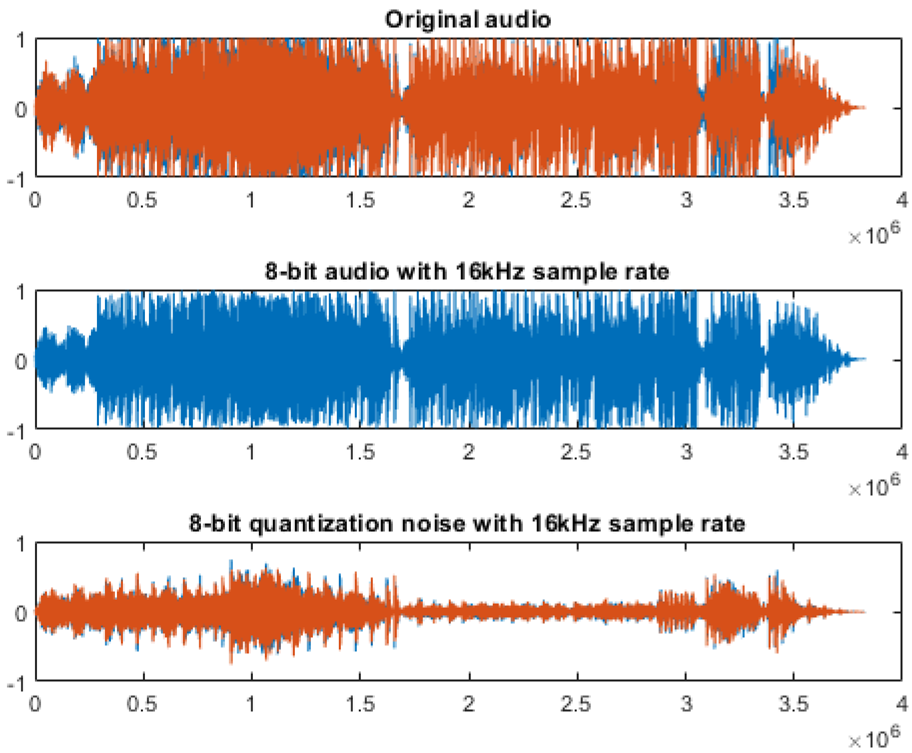
```
Computed SQNR of the signals with respect to the original file.
SQNR of x1: 5.100733 dB
SQNR of x2: 5.100732 dB
SQNR of x3: 5.100676 dB
SQNR of x4: 5.100733 dB
SQNR of x5: 5.100732 dB
SQNR of x6: 5.100676 dB
SQNR of x7: 5.100733 dB
SQNR of x8: 5.100732 dB
SQNR of x9: 5.100676 dB
```

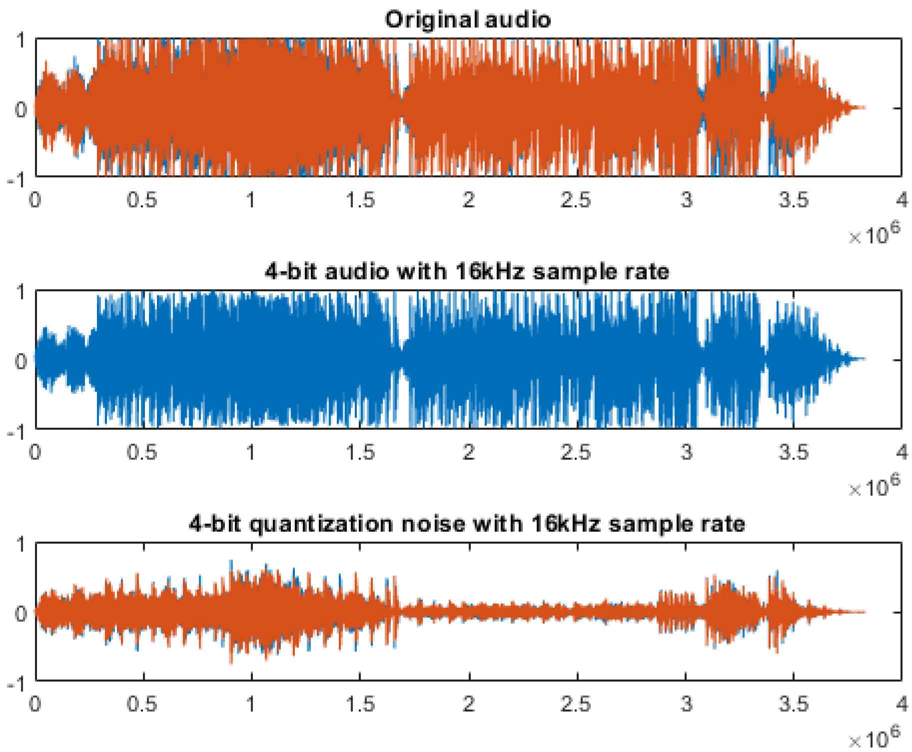

Original audio
16-bit audio with 48kHz sample rate
16-bit quantization noise with 48kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
×10^6

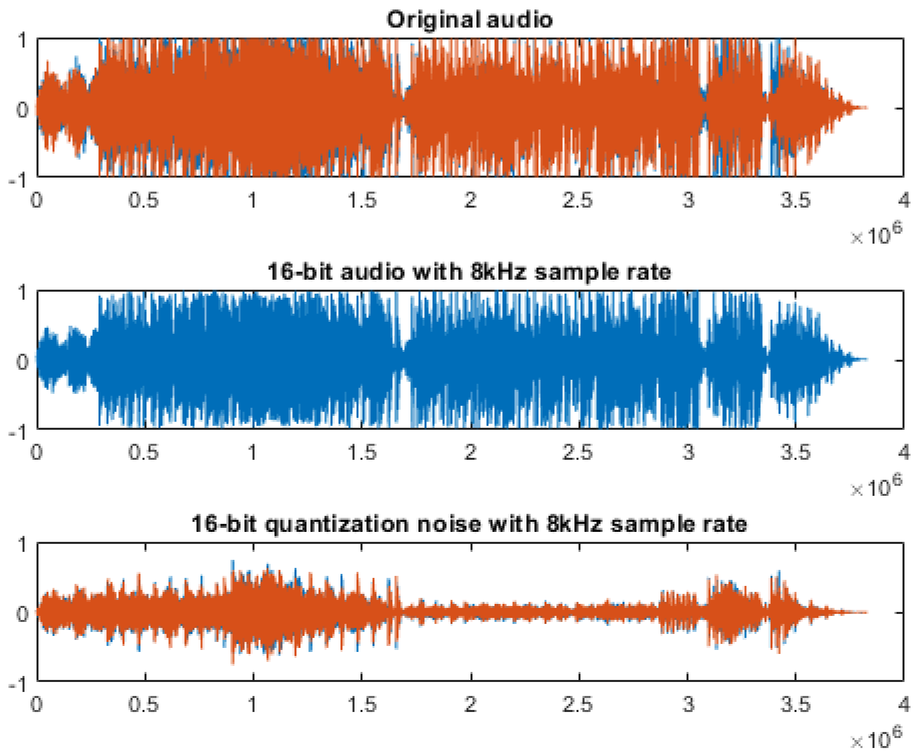

Original audio
8-bit audio with 48kHz sample rate
8-bit quantization noise with 48kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
×10^6

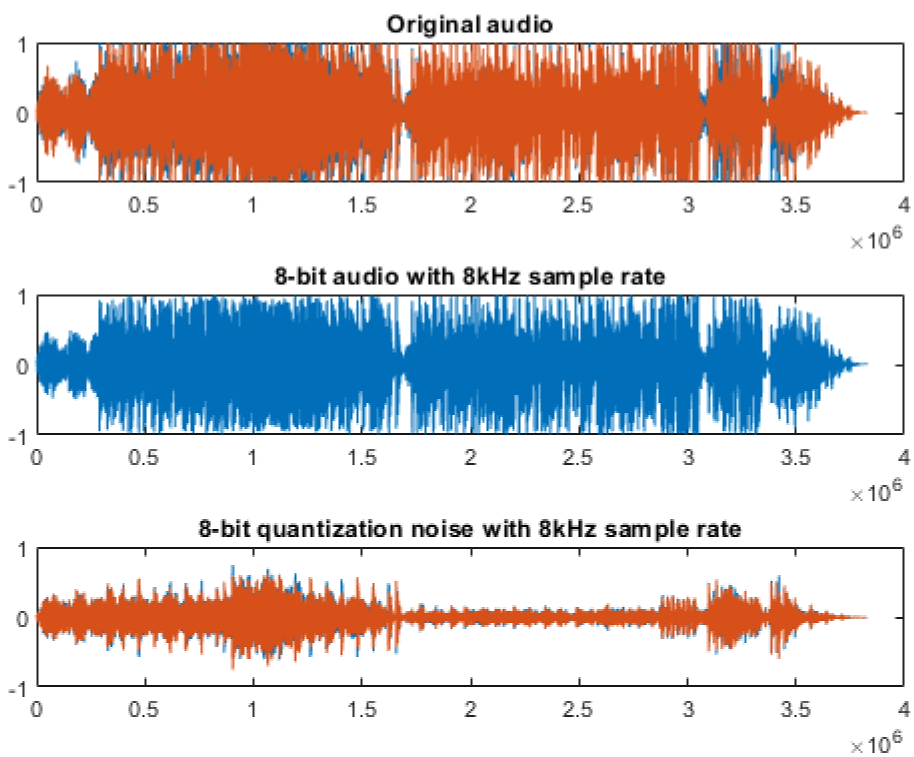

Original audio
4-bit audio with 48kHz sample rate
4-bit quantization noise with 48kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
$\times 10^6$

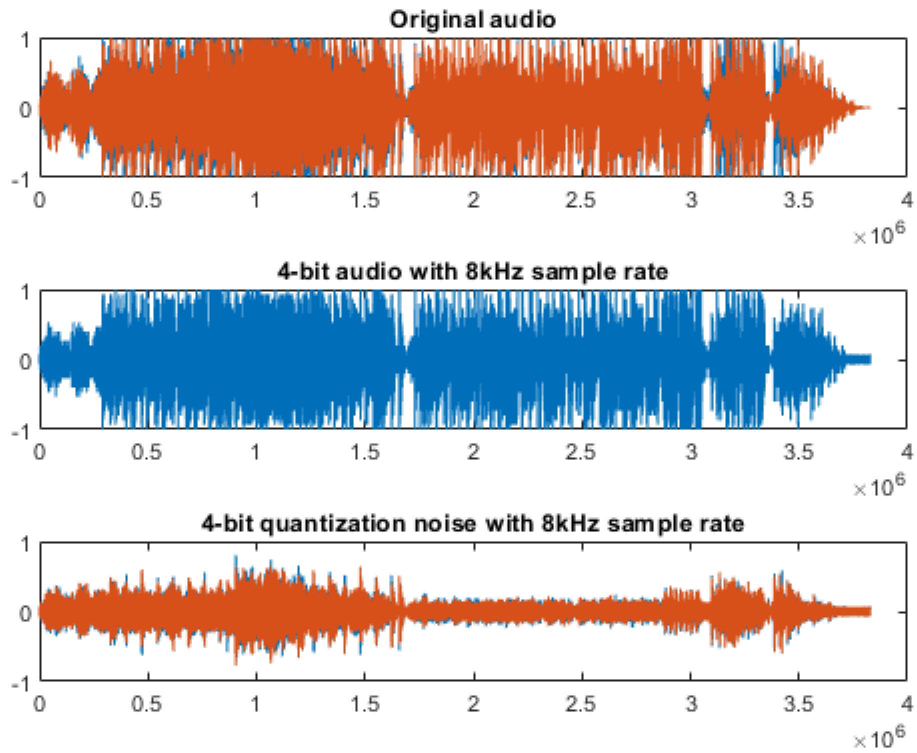

Original audio
16-bit audio with 16kHz sample rate
16-bit quantization noise with 16kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
$\times 10^6$


Original audio
8-bit audio with 16kHz sample rate
8-bit quantization noise with 16kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
$\times 10^6$


Original audio
4-bit audio with 16kHz sample rate
4-bit quantization noise with 16kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
$\times 10^6$


Original audio
16-bit audio with 8kHz sample rate
16-bit quantization noise with 8kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
×10^6


Original audio
8-bit audio with 8kHz sample rate
8-bit quantization noise with 8kHz sample rate
1
0
-1
0
0.5
1
1.5
2
2.5
3
3.5
4
×10^6



5. Half-sampling rate v. half-bit resolution.

```
% In theory, half-sampling rate produces a more audible effect than
% a half-bit resolution. The audio that will be produced in terms
% of loudness does not change but plays in lower rate. As for
% half-bit resolution, the rate of sound will not change except
% that it is noisier (or in laymans term, it is like listening to a
% music from a stereo
% that is connected to a microphone with a cloth on it. For an
% experiment consider playing the following codes below.

% UNCOMMENT EACH LINE!!!
% F_af = 'music1.flac'; %load the audio
%[F_y_af,F_fs_af]=audioread(F_af); %extract audio and sample rate data
% from the original audio
% audioinfo('music1.flac')%check audio stats esp. BPS and SampleRate
% consider halving BPS and SampleRate
% soundsc(F_y_af,48) %play audio data at Fs=48000, half of orig Fs
% [F_y_x1,F_y_x1qn,F_sqnr_x1]=quanoise(F_y_af,12); %quantize orig
% audio by 12-bits, half of orig bps: 24
% soundsc(F_y_x1qn,F_fs_af)
% Happy listening!
```

Published with MATLAB® R2020a