Digital Signal Processing - Lab 2 Report

Shakleen Ishfar

08 July, 2019

1 First Task

1.1 Question

*Write MATLAB code that will take a sinusoid and quantize it using b bits. Use the sinusoid x(t) =sin(t) in the interval t=0:0.1:4*pi. Quantize x(t) using b = 4, 8, 12 and 16 bits. For each case of b, plot the original signal, quantized signal and the quantization error.*

1.2 Theory

1.3 MATLAB Code

```
function quantize(signal, bits)
    % Quantizes analog signal to create a digital signal.
    %
    % Parameters:
    % signal - The signal function. A sinusoid signal.
    % bits - The number of bits used for encoding.

    % The interval signal is going to run.
    interval = 0 : 0.1 : 4 * pi;

    % Number of levels that can exist for number of bits
    % used for encoding.
    levels = 2 ^ bits;

    % Value of a step is difference between min and max
    % signal value divided
    % by the total number of levels.
    stepValue = 2 / levels;

    % The total number of readings. This is the total
    % number of time instances where values are measured.
    readings = length(interval);
```

```
    % An array where the quantized results will be stored.
    % It will have the same number of readings as the
    % orignal analog signal.
    quantizedValues = zeros(1, readings);

    % Used to specify exactly which reading is being
    % quantized.
    index = 1;

    % For each value in the interval, the analog reading
    % is quantized.
    for t = interval
      % Analog reading of the signal for value t in
      % interval.
      analogReading = signal(t);

      % The value we get by dividing analog value by the
      % step value. Then rounding it off to get a rounded
      % value.
      tempValue = round(analogReading / stepValue);
      quantizedValue(index) = tempValue * stepValue;
      index += 1;
    endfor

    % Plotting the analog and quanized digital signal in
    % a graph.
    set(0, "defaultaxesfontname", "Helvetica");
    hold on;
    plot(signal(interval), 'r');
    plot(quantizedValue, 'b');
    plotTitle = sprintf('Quatizing signal with %d bits', bits);
    title(plotTitle);
    xlabel("Time");
    ylabel("Angel");
    legend("Analog Signal", "Discrete Signal");
    print -djpg ../Figures/Quantize.jpg
    hold off;
endfunction
```

1.4 Test Run

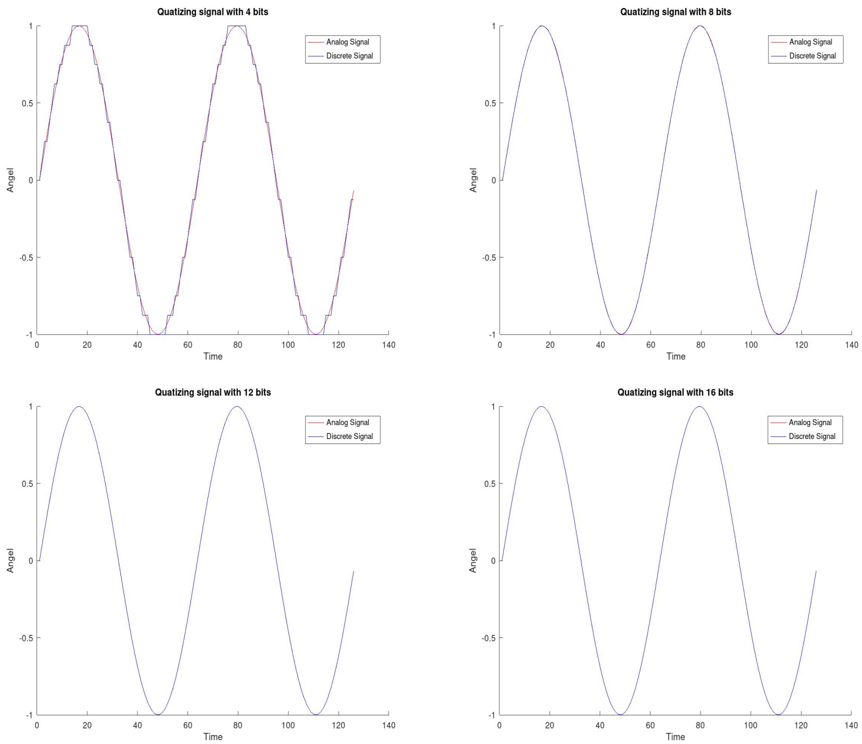
Running these commands

```
quantize(@(x)sin(x), 4);
quantize(@(x)sin(x), 8);
quantize(@(x)sin(x), 12);
```

```
quantize(@(x)sin(x), 16);
```

give the following plots. In each plot, the analog and digital signals are shown simultaneously.

Figure 1: Quantization of analog signals



2 Second Task

2.1 Question

Write MATLAB functions delta(n), unity(n) and unitramp(n) which will depict the elementary signals we read about in the class. Each of this functions, for a given value of n (n¿0), plots the corresponding signals in the range of –n to n.

2.2 Theory of Delta Signal

2.3 MATLAB Code of Delta Signal

```
function delta(n)
```

```
  % Creates a delta elementary signal.
  %
  % n - The number which will be used for the
  % range of the signal.

  % A sequence with numbers from -n to n will
  % have 2 * n + 1 number of elements.
  signal = zeros(1, 2 * n + 1);

  % The index n + 1 will be the origin or have
  % the 0 value.
  signal(n+1) = 1;
  time = -n : 1 : n;

  % Plotting signal value with respect to time.
  set(0, "defaultaxesfontname", "Helvetica");
  stem(time, signal);
  plotTitle = sprintf('Delta Signal in range (%d, -%d)', n, n);
  title(plotTitle);
  xlabel("Time");
  ylabel("Value");
  print -djpg ../Figures/Delta.jpg
endfunction
```

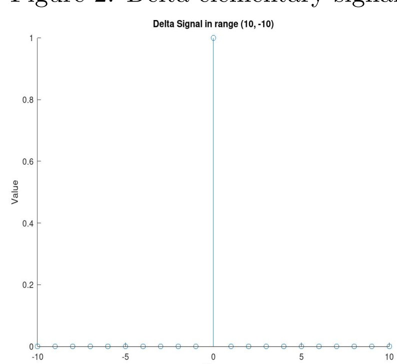
2.4 Test Run of Delta Signal

Running this command

```
delta(10);
```

gives the following plot:

Figure 2: Delta elementary signal



2.5 Theory of Unity Signal

2.6 MATLAB Code of Unity Signal

```
function unity(n)
  % Creates a unity elementary signal.
  %
  % n - The number which will be used for
  % the range of the signal.

  % A sequence with numbers from -n to n will
  % have 2 * n + 1 number of elements.
  signal = zeros(1, 2 * n + 1);

  % To make all positive numbers 1, the numbers
  % from n+1 to 2*n+1 will have to be set to 1.
  signal(n + 1 : 2 * n + 1) = 1;
  time = -n : 1 : n;

  % Plotting signal value with respect to time.
  set(0, "defaultaxesfontname", "Helvetica");
  stem(time, signal);
  plotTitle = sprintf('Unity Signal in range (%d, -%d)', n, n);
  title(plotTitle);
  xlabel("Time");
  ylabel("Value");
  print -djpg ../Figures/Unity.jpg
endfunction
```

2.7 Test Run of Unity Signal

Running this command

```
unity(10);
```

gives the following plot:

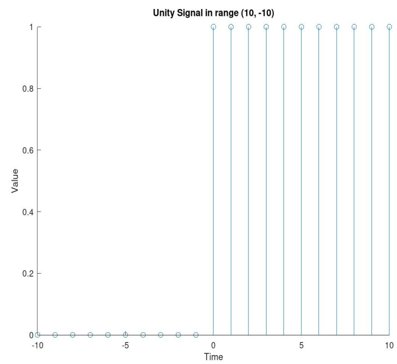
2.8 Theory of Unitramp Signal

2.9 MATLAB Code of Unitramp Signal

```
function uniramp(n)
  % Creates a uniramp elementary signal.
  %
  % n - The number which will be used for
  % the range of the signal.

  % A sequence with numbers from -n to n will
```

Figure 3: Unity elementary signal




```
    % have 2 * n + 1 number of elements.
    signal = zeros(1, 2 * n + 1);

    % Creating a row vector with values from 0
    % to 1.
    value = 0 : 1 : n;

    % All positive values will have values equal
    % to their indices.
    signal(n + 1 : 2 * n + 1) = value;
    time = -n : 1 : n;

    % Plotting signal value with respect to time.
    set(0, "defaultaxesfontname", "Helvetica");
    stem(time, signal);
    plotTitle = sprintf('Uniramp Signal in range (%d, -%d)', n, n);
    title(plotTitle);
    xlabel("Time");
    ylabel("Value");
    print -djpg ../Figures/Unitramp.jpg
endfunction
```

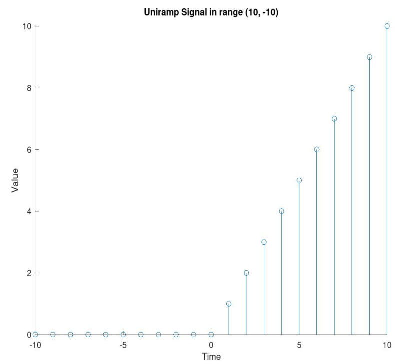
2.10 Test Run of Unitramp Signal

Running this command

`unitramp(10);`

gives the following plot:

Figure 4: Delta elementary signal



3 Third Task

3.1 Question

Write a MATLAB function that will take as input an arbitrary signal x(n) and divide it into Symmetric (even) and Antisymmetric (odd) parts and plots the three signals (original signal, even part and odd part) in the same plot.

3.2 Theory

3.3 MATLAB Code

```
function evenOddFunction(inputSignal)
  foldedSignal = flip(inputSignal);
  evenPart = (inputSignal + foldedSignal) / 2;
  oddPart = (inputSignal - foldedSignal) / 2;

  hold on;
  plot(inputSignal, 'r');
  plot(evenPart, 'g');
  plot(oddPart, 'b');
  title('Splitting a signal into even and odd parts');
  xlabel("Time");
  ylabel("Value");
  legend("Original Signal", "Even Part", "Odd Part");
  hold off;
endfunction

evenOddFunction([1 : 10] .^ 2);
```