Digital Signal Processing - Lab 2 Report

Shakleen Ishfar
ID: 160041029

18 July, 2019

Contents

1 First Task

1.1 Question: Task 1

*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: Quantization

An analog signal has no discrete number of values. This type of signal can take on any value within a specific range. However, this signal can't be processed efficiently due to this reason. Which is why for ease of storing and processing an analog signal is converted into a digital signal. The process of converting from analog to digital has 3 steps which are as follows:

1. **Sampling:** The process of taking finite sets of readings. Turns continuous signals to discrete signals.

2. **Quantization:** The process of taking only discrete values per reading. Turns analog signals to digital signals.

3. **Encoding:** Giving each quantized value a specific code for representation in digital format.

The steps of *quantization* are listed below:

1. Take the difference of min and max amplitude and save it as *range*.

$$range = max(amplitude) - min(amplitude)$$

2. To perform quantization we need to know how many discrete value to divide *range* into. Usually, this is a power of 2. So if we denote this as *levels* we can write

$$levels = 2^n$$

where n is the number of encoding bits we wish to use.

3. Next, we find the *step value* for quantization.

$$stepvalue = \frac{range}{levels}$$

4. This step value is the smallest step size in the quantization process. An analog reading will be converted to the closest discrete value which is a multiple of this step value in either the positive or negative direction.

1.3 MATLAB Code: quantize()

The code listed below demonstrates the quantization process.

```
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
```

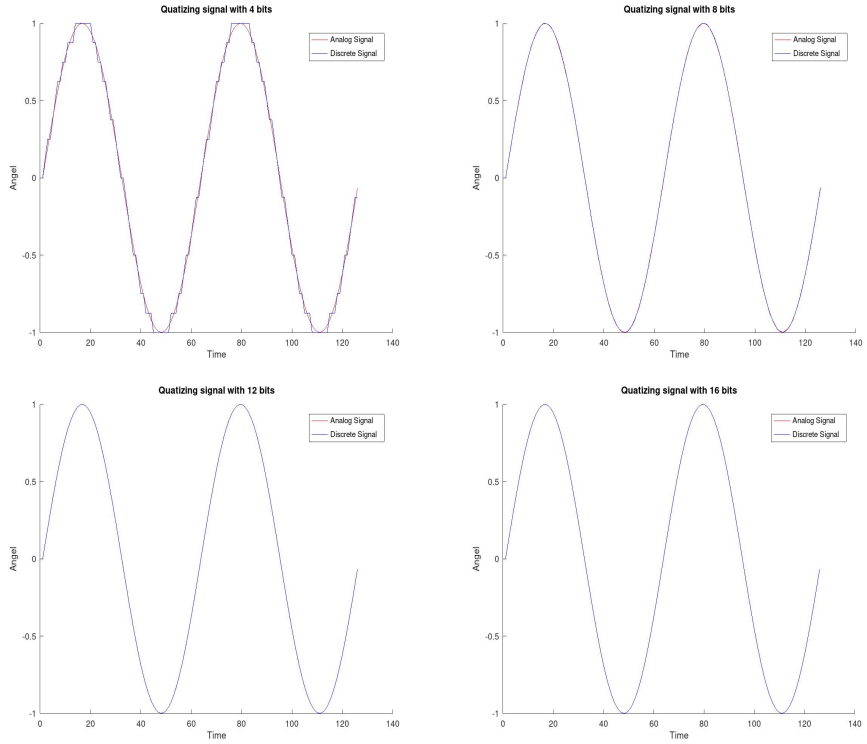
1.4 Test Run: quantize()

Running the function *quantize(signal, bits)* for sin wave as signal and different values of bits

```
% Running for values b = 4, 8, 12, 16
quantize(@(x)sin(x), 4);
quantize(@(x)sin(x), 8);
quantize(@(x)sin(x), 12);
quantize(@(x)sin(x), 16);
```

give the plots in *Figure 1.* In each plot, the analog and digital signals are shown simultaneously.

Figure 1: Quantization of analog signals



2 Second Task

2.1 Question: Task 2

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: Delta Signal

A *delta elementary signal* has a value 1 in the index position of 0 and has the value 0 everywhere else. In other words, if $n = 0$ then $x(n) = 1$. Otherwise, $x(n) = 0$.

2.3 MATLAB Function: delta()

The code listed below implements a function that plots a delta elementary signal when executed.

```
1 function delta(n)
```

```
  % Creates a delta elementary signal.
  %
  % Parameters:
  % 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
```

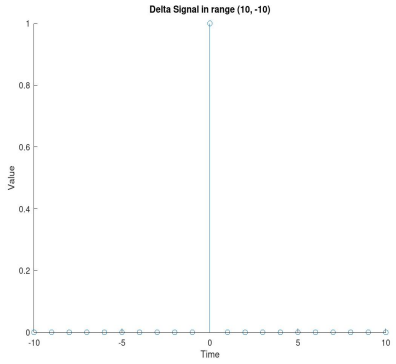
2.4 Test Run: delta()

Running the function *delta(n)* for n = 10

```
% Running for n = 10
delta(10);
```

gives the plot in *Figure 2*.

Figure 2: Delta Elementary Signal



2.5 Theory: Unity Signal

A *unity signal* has value 1 for all non-negative indices and has value of 0 for all negative indices. In other words, if $n \geq 0$ then $x(n) = 1$. Otherwise, $x(n) = 0$.

2.6 MATLAB Function: unity()

```
function unity(n)
  % Creates a unity elementary signal.
  %
  % Parameters:
  % 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: unity()

Running the function *unity(n)* for n = 10

```
% Running for n = 10.
unity(10);
```

gives the plot in *Figure 3*.

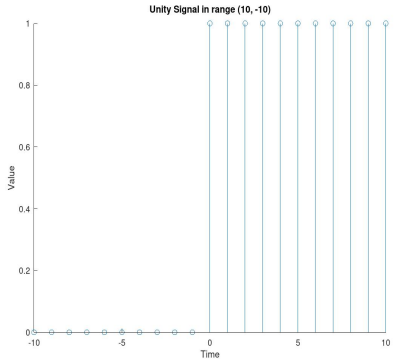
2.8 Theory: Unitramp Signal

A *unitramp signal* has value equal to the value of the indices in non-negative indices and has the value of 0 for all negative indices. In other words, if $n \geq 0$ then $x(n) = n$. Otherwise, $x(n) = 0$.

2.9 MATLAB Function: unitramp()

```
function unitramp(n)
  % Creates a unitramp elementary signal.
  %
  % n - The number which will be used for
```

Figure 3: Unity Elementary Signal




```
  % 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);

  % 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('Unitramp Signal in range (%d, -%d)', n, n);
  title(plotTitle);
  xlabel("Time");
  ylabel("Value");
  print -djpg ../Figures/Unitramp.jpg
endfunction
```

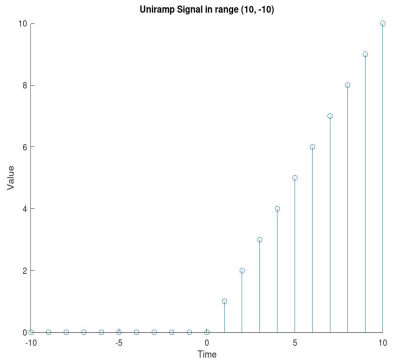
2.10 Test Run of unitramp Function

Running the function *unitramp(n)* for n = 10

```
% Running for n = 10.
unitramp(10);
```

gives the plot in *Figure 4*.

Figure 4: Unitramp elementary signal



3 Third Task

3.1 Question: Task 3

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: Even Odd Decomposition

Decomposing a signal into even and odd can be done using the following formulae:

$$even(n) = \frac{x(n) + x(N-n)}{2}$$

$$odd(n) = \frac{x(n) - x(N-n)}{2}$$

3.3 MATLAB Code: evenOddFunction()

The code listed below decomposes a signal into even and odd signal. The function also plots the signals in the same graph to give a visual representation of the signals.

```
function evenOddFunction(inputSignal)
    % Creates and displays even and odd signal of the
    % input signal.
    %
    % Parameters:
    % inputSignal - The signal function. A sinusoid signal.

    % The folded signal of the inputSignal.
    foldedSignal = flip(inputSignal);

    % The even signal of the inputSignal
```

```
    evenPart = (inputSignal + foldedSignal) / 2;

    % The odd signal of the inputSignal
    oddPart = (inputSignal - foldedSignal) / 2;

    % Plotting the even, odd and input signal side by side.
    set(0, "defaultaxesfontname", "Helvetica");
    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("Input Signal", "Even Part", "Odd Part");
    print -djpg ../Figures/Even_Odd_Signal.jpg
    hold off;
endfunction
```

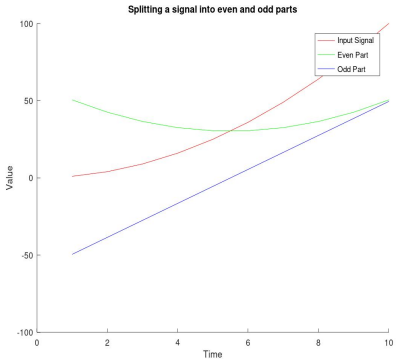
3.4 Test Run: evenOddFunction()

Running the function *evenOddFunction(inputSignal)* for the signal $1, 4, 9, ..., 100$

```
% Testing with a signal of [1 : 10] .^ 2
evenOddFunction([1 : 10].^2);
```

gives the plot in *Figure 5*.

Figure 5: Even Odd Decomposition of Signal



4 Fourth Task

4.1 Question: Task 4

Write a MATLAB function sigshift that takes a signal x(n) and a shift value k as inputs and returns the resulting signal y(n) = x(n-k).

4.2 Theory: Shifting Signals

Shifting a signal means getting the values earlier or later. Mathematically shifting is represented as $y(n) = x(n \pm k)$ Shifting is of two types which are:

1. Delayed: Shifting a signal by a negative amount. k is negative.
2. Early: Shifting a signal by a positive amount. k is positive.

4.3 MATLAB Code: sigshift()

The code listed below implements a function that shifts a signal by k units.

```
function [outputSignal] = sigshift(inputSignal, k)
  % Shifts signal by k bits resulting in outputSignal.
  % Here, output signal(n) = inputSignal(n-k).
  %
  % Parameters:
  % inputSignal - The signal function. A sinusoid signal.
  % k -  The amount to shift the signal.
  %
  % Returns:
  % outputSignal - The shifted signal.
  inputSize = size(inputSignal);
  outputSignal = zeros(1, inputSize(2));
  outputSignal(k + 1 : inputSize(2)) = inputSignal(1 : inputSize(2)
     - k);

  % Plotting the input and shifted signal in a graph.
  set(0, "defaultaxesfontname", "Helvetica");
  hold on;
  plot(inputSignal, 'r');
  plot(outputSignal, 'b');
  plotTitle = sprintf('Shifting signal %d bits', k);
  title(plotTitle);
  xlabel("Time");
  ylabel("Value");
  legend("Input Signal", "Shifted Signal");
  print -djpg ../Figures/Sigshift.jpg
  hold off;
endfunction
```

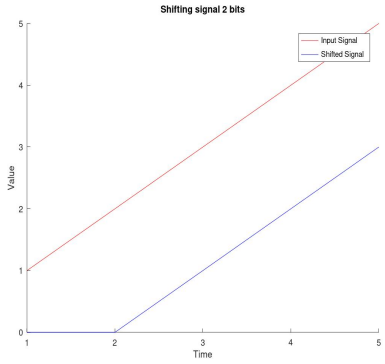
4.4 Test Run: sigshift()

Running the function sigshift(inputSignal, k) for the signal $1, 2, 3, 4, 5$ and k = 2

```
% Testing sigshift function
disp(sigshift([1 2 3 4 5], 2));
```

gives the result $0, 0, 1, 2, 3$ and shows the plot in *Figure 6*.

Figure 6: Shifting signals



5 Fifth Task

5.1 Question: Task 5

Write a MATLAB function sigfold that takes a signal x(n) and returns the resulting signal y(n) = x(-n). Verify the correctness of your function by taking suitable signals as input/output and plotting them.

5.2 Theory: Signal Folding

Folding a signal means reversing the axis the signal is on. Basically, shifting means $y(n) = x(-n)$. When a signal is folded the origin remains unchanged. However, the values on the positive part of the axis goes to the negative part of the axis and vise versa.

5.3 MATLAB Code: sigfold()

The code listed below implements a function that folds a signal.

```
function [outputSignal] = sigfold(inputSignal)
  % Folds inputSignal to produce outputSignal.
  % Here, outputSignal(n) = inputSignal(-n).
  %
  % Parameters:
  % inputSignal - The inputted signal that is to be folded.
  %
  % Returns:
  % outputSignal - The folded signal.

  % outputSignal is the folded signal of inputSignal.
  outputSignal = flip(inputSignal);

  % Plotting the input and output signal.
  set(0, "defaultaxesfontname", "Helvetica");
  hold on;
```

```
  plot(inputSignal, 'r');
  plot(outputSignal, 'b');
  title('Folding signal');
  xlabel("Time");
  ylabel("Angel");
  legend("Orignal Signal", "Folded Signal");
  print -djpg ../Figures/Sigfold.jpg
  hold off;
endfunction
```

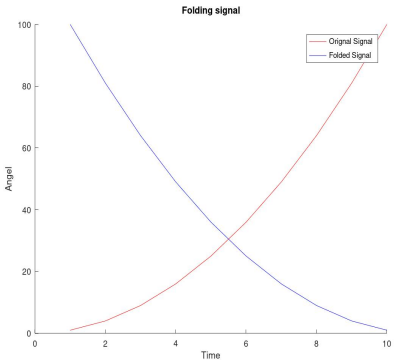
5.4 Test Run: sigfold()

Running the function sigfold(inputSignal) for the signal $1, 4, 9, ..., 100$

```
% Testing sigfold() function
disp(sigfold([1 : 10] .^ 2));
```

gives the output $100, 81, 64, ..., 9, 4, 1$ and shows the plot in *Figure 7*.

Figure 7: Folding signal



6 Sixth Task

6.1 Question: Task 6

Write a MATLAB function downsample that takes a signal x(n) and a value d (d is an integer, d¿1) and returns the down-sampled signal y(n) = x(dn). Verify the correctness by plotting suitable input/output signals.

6.2 Theory: Down Sampling

Down sampling a signal means taking a smaller subset of readings of the signal. Mathematically, $y(n) = x(dn)$ where d is the amount of down sampling applied.

6.3 MATLAB Code: downsample()

The code listed below implements a function that downsamples a signal.

```
function [outputSignal] = downSample(inputSignal, d)
  % Down sample inputSignal to produce the outputSignal.
  %
  % Parameters:
  % inputSignal - The signal to be down sampled.
  % d - The amount of down sampling to perform.
  %
  % Returns:
  % outputSignal - The down sampled signal.

  % outputSignal will have equal length of the inputSignal.
  outputSignal = zeros(1, length(inputSignal));
  index = [1 : length(inputSignal)];

  % When down sampling a signal only selective values will remain.
  % The indices who're divisible by d, will only be kept.
  outputSignal(mod(index, d) == 0) = inputSignal(mod(index, d) ==
    0);

  % Plotting the input and down sampled signal.
  set(0, "defaultaxesfontname", "Helvetica");
  hold on;
  subplot(2, 1, 1);
  stem(inputSignal, 'r');
  plotTitle = sprintf('Down sampling %d times', d);
  title(plotTitle);
  legend("Original Signal");
  subplot(2, 1, 2);
  stem(outputSignal, 'b');
  legend("Down Sampled Signal");
  print -djpg ../Figures/Down_Sample.jpg
  hold off;
endfunction
```

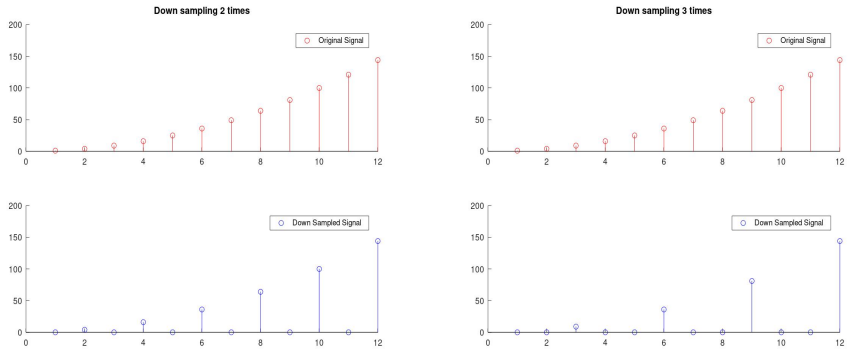
6.4 Test Run: downsample()

Running the function downsample(inputSignal, d) for the signal $1, 4, 9, ..., 144$ and d = 2, 3

```
% Testing downSample() function
disp(downSample([1 : 12] .^ 2, 2));
disp(downSample([1 : 12] .^ 2, 3));
```

gives the result $0, 4, 0, 16, 0, 36, 0, 64, 0, 100, 0, 144$ and shows the plot in *Figure 8*.

Figure 8: Down Sampling Signals



7 Seventh Task

7.1 Question: Task 7

Write a MATLAB function sigadd that takes two signals x1(n) and x2(n) as inputs and returns the resulting signal y(n) = x1(n)+x2(n).

7.2 Theory: Adding Signals

Adding two signals results in a signal where the resultant signal follows the follwing mathematical expression:

$$y(n) = x_1(n) + x_2(n)$$

7.3 MATLAB Code: sigadd()

The code listed below implements a function that adds two signal.

```
function [outputSignal] = sigadd(signal1, signal2)
  % Function for adding two signals. The function will add
  % signals after syncronizing their origins.
  %
  % Parameters:
  % signal1 - The first signal.
  % signal2 - The second signal.
  %
  % Returns:
  % outputSignal - The result of adding the two signals.

  % The total length of outputSignal will be the max of signal1
  % and signal2 length
  maxLength = max(length(signal1), length(signal2));

  % Initializing outputSignal to zeros.
  outputSignal = zeros(1, maxLength);

```

```
  % Adding the values of signal1 and signal2 and saving result
  % to outputSignal at index i.
  for i = 1 : maxLength
    value1 = 0;
    value2 = 0;

    if i <= length(signal1)
      value1 = signal1(i);
    endif

    if i <= length(signal2)
      value2 = signal2(i);
    endif

    outputSignal(i) = value1 + value2;
  endfor

  % Plotting the three signals.
  set(0, "defaultaxesfontname", "Helvetica");
  hold on;
  plot(signal1, 'r');
  plot(signal2, 'g');
  plot(outputSignal, 'b');
  title('Adding two signals');
  xlabel("Time");
  ylabel("Value");
  legend("Signal 1", "Signal 2", "Added Signal");
  print -djpg ../Figures/Sigadd.jpg
  hold off;
endfunction
```

7.4 Test Run: sigadd()

Running this command

```
% Testing sigadd() function
sigadd([1 : 2 : 10], [1 : 3 : 27]);
```

gives the plot in *Figure 9*.

8 Eighth Task

8.1 Question: Task 8

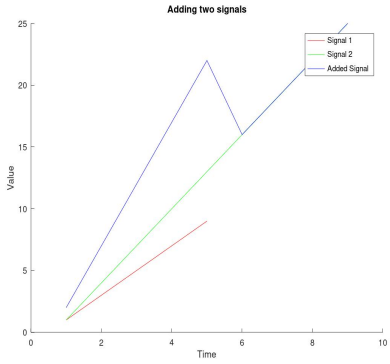
*Write a MATLAB function sigmult that takes two signals x1(n) and x2(n) as inputs and returns the resulting signal y(n) = x1(n)*x2(n).*

8.2 Theory: Multiplying Signals

Multiplying two signals results in a signal where the resultant signal follows the follwing mathematical expression:

$$y(n) = x_1(n) * x_2(n)$$

Figure 9: Adding signals



8.3 MATLAB Code: sigmult()

The code listed below implements a function that multiplies two signal.

```
function [outputSignal] = sigmult(signal1, signal2)
  % Function for multiplying two signals. The function will
  % multiply signals after syncronizing their origins.
  %
  % Parameters:
  % signal1 - The first signal.
  % signal2 - The second signal.
  %
  % Returns:
  % outputSignal - The result of multiplying the two signals.

  % The total length of outputSignal will be the max of signal1
  % and signal2 length
  maxLength = max(length(signal1), length(signal2));

  % Initializing outputSignal to zeros.
  outputSignal = zeros(1, maxLength);

  % Multiplying the values of signal1 and signal2 and saving result
  % to outputSignal at index i.
  for i = 1 : maxLength
    value1 = 0;
    value2 = 0;

    if i <= length(signal1)
      value1 = signal1(i);
    endif

    if i <= length(signal2)
      value2 = signal2(i);
    endif

    outputSignal(i) = value1 * value2;
  endfor

```

```
  % Plotting the three signals.
  set(0, "defaultaxesfontname", "Helvetica");
  hold on;
  plot(signal1, 'r');
  plot(signal2, 'g');
  plot(outputSignal, 'b');
  title('Multiplying two signals');
  xlabel("Time");
  ylabel("Value");
  legend("Signal 1", "Signal 2", "Added Signal");
  print -djpg ../Figures/Sigmult.jpg
  hold off;
endfunction
```

8.4 Test Run: sigmult()

Running this command

```
% Testing sigmult() function
sigmult([1 : 2 : 10], [1 : 3 : 27]);
```

gives the plot in *Figure 10*.

Figure 10: Multiplying signals