Lab Report: *06*

Title: *Differential Pulse Code Modulation (DPCM) and Delta Modulation*

Course Title: *Data and Telecommunication Laboratory*

Course Code: *CSE-260*

2nd Year 2nd Semester Examination 2023

Submitted to :-

Sarnali Basak
Associate Professor

Dr. Md. Imdadul Islam
Professor

Dr. Md. Abul Kalam Azad
Professor

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Class Roll	Name
371	Md. Ahad Siddiki

Date of Submission: 05/01/2025

Differential Pulse Code Modulation (DPCM) and Delta Modulation

Objective:

- To study the principles of Differential Pulse Code Modulation (DPCM) and Delta Modulation.
- To encode and recover signals using these modulation techniques.
- To analyze the signal quality and error characteristics in both methods.

Theory:

Differential Pulse Code Modulation (DPCM)

- DPCM reduces the bit requirements by encoding the difference between consecutive samples rather than the absolute values.
- It maintains signal quality (SQR) comparable to PCM while requiring fewer bits.

Delta Modulation (DM)

- A special case of DPCM where the difference between consecutive samples is encoded using a single bit.
- Simplicity in design makes it efficient but more prone to granular noise and slope overload.

Equipment Used:

- MATLAB R2024a software for simulation and analysis.
- MATLAB functions such as dpcmenco and dpcmdeco for encoding and decoding signals.

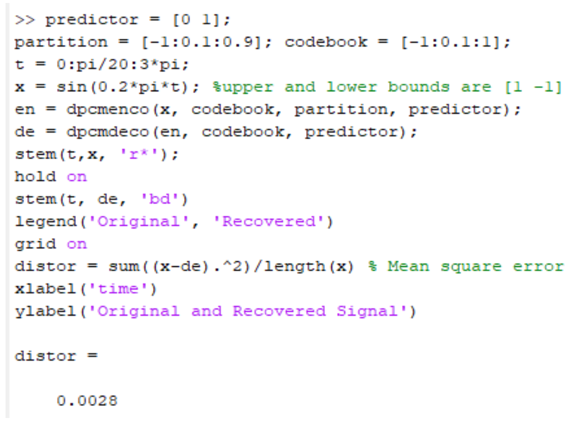
Experimental Procedure:

1. **Sampling of Analog Signals**
   - Generate a sinusoidal wave using MATLAB for simulation.
   - Implement DPCM and DM to encode and recover the wave.
2. **Parameters for DPCM**
   - Codebook: Defines the quantized values.
   - Partition: Sets the boundaries for quantization.
   - Predictor: Uses the previous sample to predict the current value.
3. **Delta Modulation Implementation**
   - Use a step size k for encoding the difference between consecutive samples.
   - Recover the signal using the initial sample and cumulative differences.
4. **Analysis**
   - Plot and compare the original and recovered signals.
   - Calculate and analyze the mean square error (MSE) for signal quality.

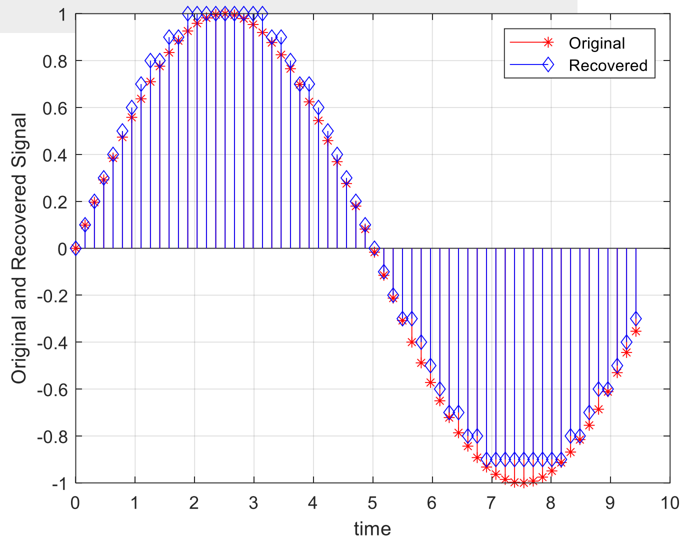
Q1. Differential Pulse Code Modulation (DPCM)

Code & Output:

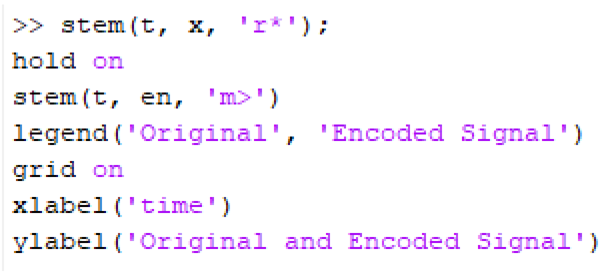
```
predictor = [0 1];
partition = [-1:0.1:0.9]; codebook = [-1:0.1:1];
t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); %upper and lower bounds are [1 -1]
en = dpcmenco(x, codebook, partition, predictor);
de = dpcmdeco(en, codebook, predictor);
stem(t,x, 'r*');
hold on
stem(t, de, 'bd')
legend('Original', 'Recovered')
grid on
distor = sum((x-de).^2)/length(x) % Mean square error
xlabel('time')
ylabel('Original and Recovered Signal')
```



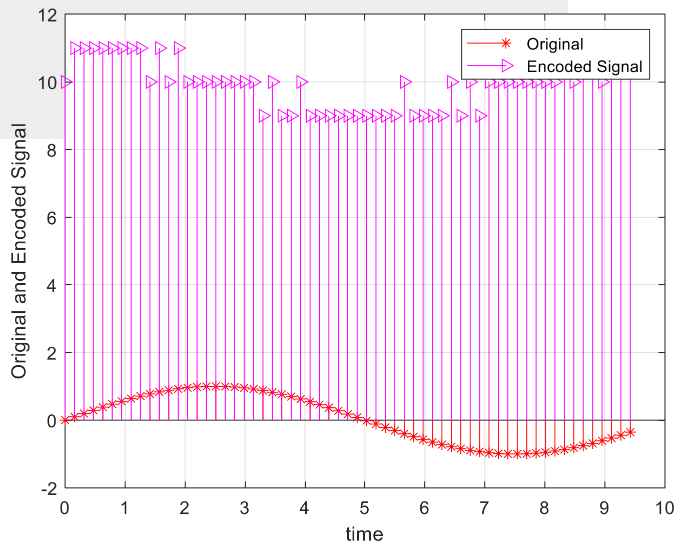
```
>> predictor = [0 1];
partition = [-1:0.1:0.9]; codebook = [-1:0.1:1];
t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); %upper and lower bounds are [1 -1]
en = dpcmenco(x, codebook, partition, predictor);
de = dpcmdeco(en, codebook, predictor);
stem(t,x, 'r*');
hold on
stem(t, de, 'bd')
legend('Original', 'Recovered')
grid on
distor = sum((x-de).^2)/length(x) % Mean square error
xlabel('time')
ylabel('Original and Recovered Signal')

distor =

    0.0028
```



```
% Original and Encoded Signal
stem(t, x, 'r*');
hold on
stem(t, en, 'm>')
legend('Original', 'Encoded Signal')
grid on
xlabel('time')
ylabel('Original and Encoded Signal')
```



```
>> stem(t, x, 'r*');
hold on
stem(t, en, 'm>')
legend('Original', 'Encoded Signal')
grid on
xlabel('time')
ylabel('Original and Encoded Signal')
```



Observations:

- **Original and Recovered Signal:**

  - **Visualization:**
    - The red stars (r*) represent the original sinusoidal signal (x).
    - The blue diamonds (bd) represent the recovered signal (de).
  - **Observation:**
    - The recovered signal closely follows the shape of the original signal, indicating the effectiveness of the DPCM encoding and decoding process.
    - Small deviations between the original and recovered signals are due to quantization errors introduced during encoding.
    - The mean square error (MSE), calculated as `distor`, provides a quantitative measure of this deviation. A lower MSE indicates better signal recovery.

- **Original and Encoded Signal:**

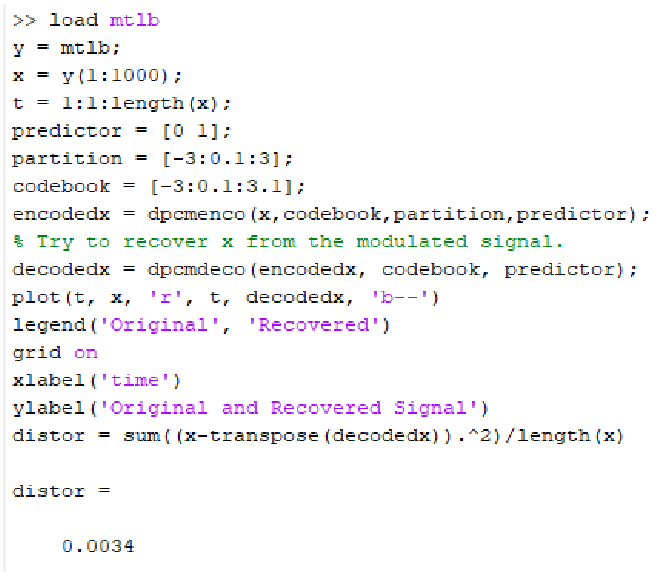
  - **Visualization:**
    - The red stars (r*) represent the original sinusoidal signal (x).
    - The magenta triangles (m>) represent the encoded signal (en).
  - **Observation:**
    - The encoded signal exhibits a step-like pattern compared to the smooth sinusoidal original signal. This reflects the discrete nature of the DPCM encoding process, which quantizes the signal into finite levels based on the `partition` and `codebook`.
    - The magnitude and shape of the encoded signal vary in response to changes in the original signal but with reduced resolution due to quantization.

This implementation demonstrates the power of DPCM in balancing compression efficiency and signal fidelity.

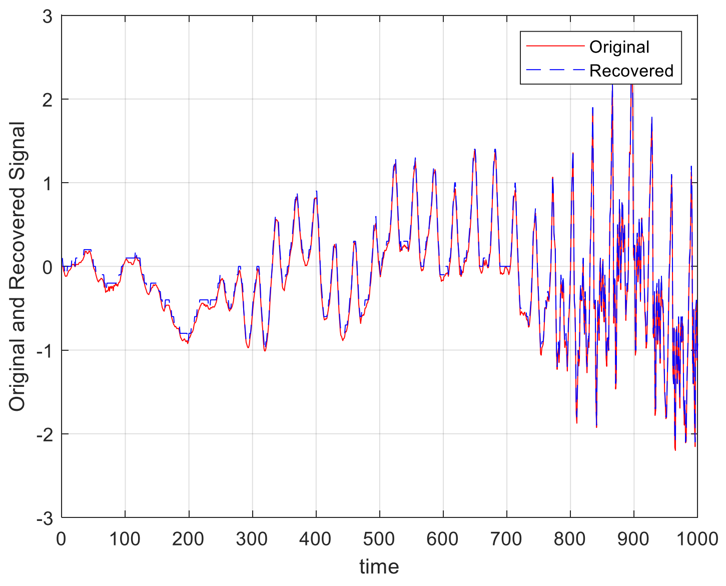
Q2. Encode a sinusoidal signal based on DPCM and recover it using previous sample as the predictor.

Code & Output:

```
load mtlb
y = mtlb;
x = y(1:1000);
t = 1:1:length(x);
predictor = [0 1];
partition = [-3:0.1:3];
codebook = [-3:0.1:3.1];
encodedx = dpcmenco(x,codebook,partition,predictor);
% Try to recover x from the modulated signal.
decodedx = dpcmdeco(encodedx, codebook, predictor);
plot(t, x, 'r', t, decodedx, 'b--')
legend('Original', 'Recovered')
grid on
xlabel('time')
ylabel('Original and Recovered Signal')
distor = sum((x-transpose(decodedx)).^2)/length(x)
```



```
>> load mtlb
y = mtlb;
x = y(1:1000);
t = 1:1:length(x);
predictor = [0 1];
partition = [-3:0.1:3];
codebook = [-3:0.1:3.1];
encodedx = dpcmenco(x,codebook,partition,predictor);
% Try to recover x from the modulated signal.
decodedx = dpcmdeco(encodedx, codebook, predictor);
plot(t, x, 'r', t, decodedx, 'b--')
legend('Original', 'Recovered')
grid on
xlabel('time')
ylabel('Original and Recovered Signal')
distor = sum((x-transpose(decodedx)).^2)/length(x)

distor =

    0.0034
```



Observations:

- **Visual Comparison**:
    - The red solid line represents the **original signal**, showing the actual data from the sampled audio segment.
    - The blue dashed line represents the **recovered signal** obtained after encoding and decoding using DPCM.
- **Signal Alignment**:
    - The recovered signal closely follows the shape and trends of the original signal, indicating that the encoding and decoding process effectively preserved the signal's key features.
- **Deviations**:
    - Minor differences between the original and recovered signals are visible, especially around rapid changes in the signal (e.g., steep slopes or sharp transitions).
    - These deviations are due to **quantization errors**, which occur when continuous differences are mapped to discrete levels.
- **Overall Fidelity**:
    - The recovered signal is smooth and retains the overall structure of the original signal.
    - The deviations are relatively small, suggesting that the encoding process is well-suited for the chosen quantization levels and predictor.

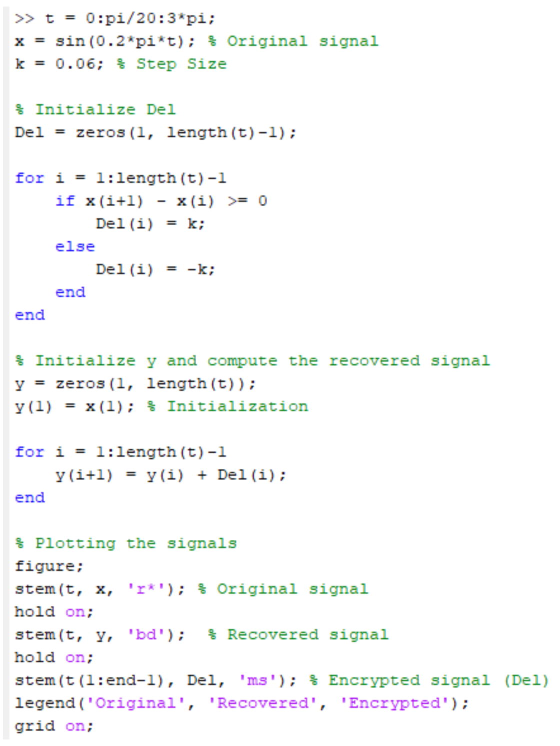
The graph shows that DPCM provides a good balance between compression and signal fidelity. While there are minor errors due to quantization, the recovered signal accurately approximates the original, demonstrating the method's effectiveness.

Delta Modultaion

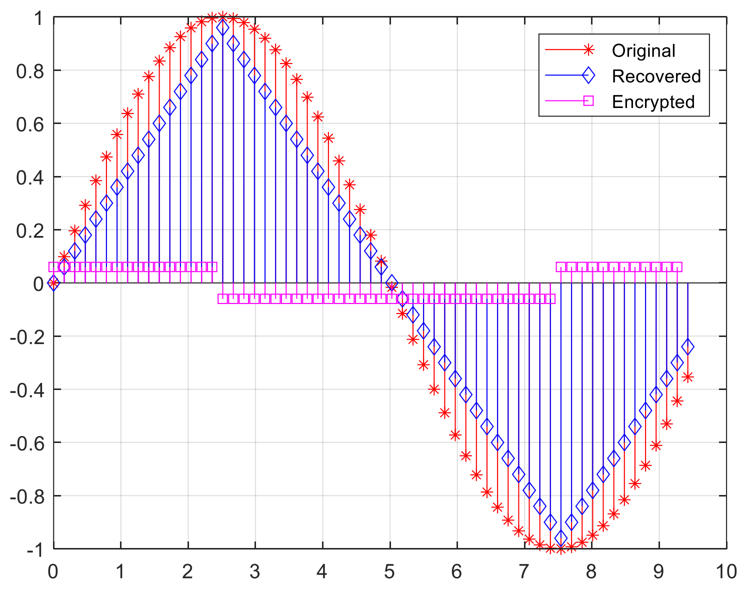
Q3. Signal Recovery and Encryption Using Step-wise Adjustment Based on Signal Differences

Code & Output:

```
t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); % Original signal
k=0.06; % Step Size
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
                Del(i)=-k;
end
end
y(1)=x(1); % Initialisation
for i=1:length(t)-1
y(i+1) = y(i)+Del(i);
end
stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
hold on
stem(0:pi/20:3*pi-pi/20, Del, 'ms')
legend('Original', 'Recovered', 'Encrypted')
grid on
```



```
>> t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); % Original signal
k = 0.06; % Step Size

% Initialize Del
Del = zeros(1, length(t)-1);

for i = 1:length(t)-1
    if x(i+1) - x(i) >= 0
        Del(i) = k;
    else
        Del(i) = -k;
    end
end

% Initialize y and compute the recovered signal
y = zeros(1, length(t));
y(1) = x(1); % Initialization

for i = 1:length(t)-1
    y(i+1) = y(i) + Del(i);
end

% Plotting the signals
figure;
stem(t, x, 'r*'); % Original signal
hold on;
stem(t, y, 'bd');  % Recovered signal
hold on;
stem(t(1:end-1), Del, 'ms'); % Encrypted signal (Del)
legend('Original', 'Recovered', 'Encrypted');
grid on;
```



Observations:

- **Original Signal (Red Dots)**:
    - The sinusoidal wave **(`x`)** is smooth and continuous, representing the unaltered signal before any modulation.
    - This curve serves as the reference to evaluate the performance of Delta Modulation.

- **Recovered Signal (Blue Diamonds)**:
    - The recovered signal **(`y`)** closely follows the original sinusoidal wave but appears as a step-like approximation due to the incremental nature of Delta Modulation.
    - The accuracy of recovery depends on the step size **(`k`)**. Here, the chosen step **size (`k = 0.06`)** seems appropriate for capturing the general shape of the wave.

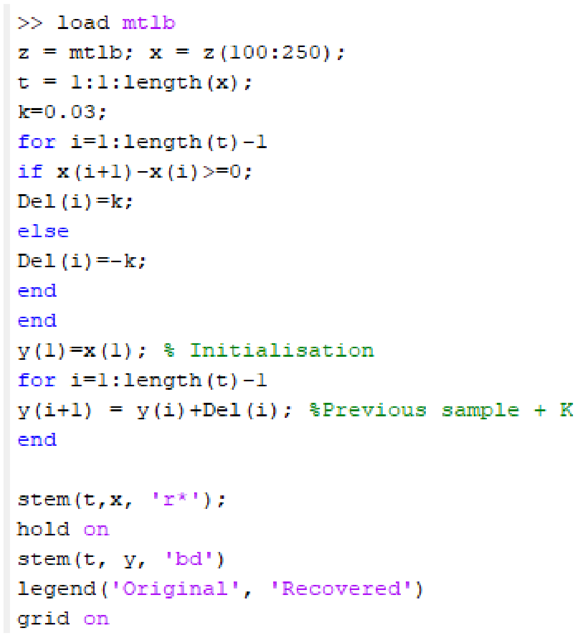
- **Delta Signal (Magenta Squares)**:
    - The Delta **(`Del`)** values represent the step-by-step differences between consecutive samples.
    - Positive and negative deltas correspond to the upward and downward movements in the sinusoidal wave.

- **Overall Analysis**:
    - The recovered signal aligns well with the original signal, demonstrating the effectiveness of Delta Modulation for this step size.
    - Slight deviations in the recovered signal from the original indicate minor granular noise, a common artifact in Delta Modulation.
    - If the step size were too large, the recovered signal might overshoot, leading to slope overload distortion. Conversely, a smaller step size would increase granular noise, reducing accuracy.

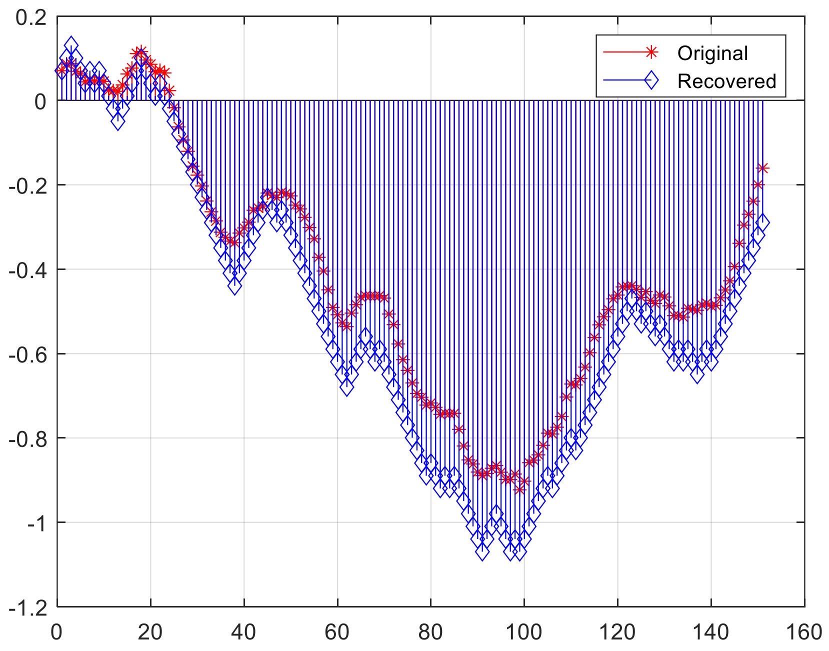
Q4. Signal Recovery Using Step-wise Adjustment Based on Signal Differences

Code & output:

```
load mtlb
z = mtlb; x = z(100:250);
t = 1:1:length(x);
k=0.03;
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
Del(i)=-k;
end
end
y(1)=x(1); % Initialisation
for i=1:length(t)-1
y(i+1) = y(i)+Del(i); %Previous sample + K
end

stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
legend('Original', 'Recovered')
grid on
```



```
>> load mtlb
z = mtlb; x = z(100:250);
t = 1:1:length(x);
k=0.03;
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
Del(i)=-k;
end
end
y(1)=x(1); % Initialisation
for i=1:length(t)-1
y(i+1) = y(i)+Del(i); %Previous sample + K
end

stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
legend('Original', 'Recovered')
grid on
```



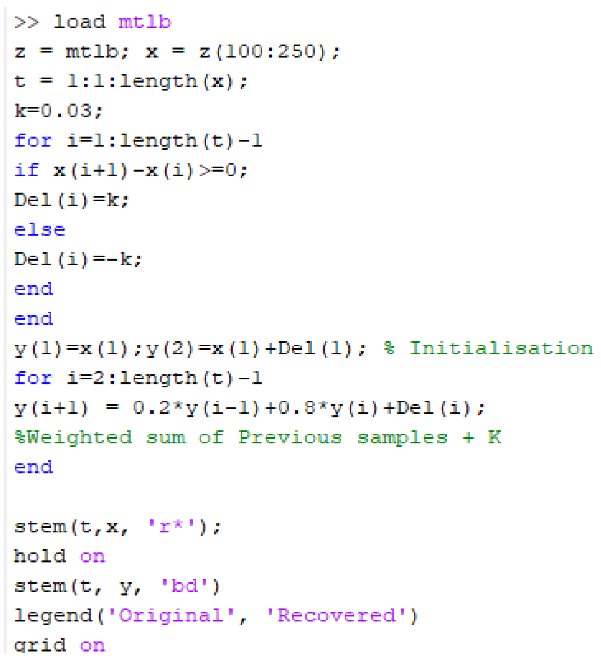
Observations:

- **Original Signal (Red Stars)**:
    - The red star markers represent the original signal, `x`. It displays a sinusoidal wave-like pattern, where the values oscillate between positive and negative.
    - The signal seems to have a smooth, continuous variation over time, which is characteristic of a typical sine wave.
- **Recovered Signal (Blue Diamonds)**:
    - The blue diamond markers represent the recovered signal, `y`. This signal is an approximation of the original one, but it differs slightly due to the step-wise adjustment applied.
    - The recovered signal has a stepped appearance, indicating that it has been modified based on the changes between consecutive samples in the original signal.
    - While the overall shape of the recovered signal is similar to the original, there are noticeable jumps at each point where the recovery process adjusted the signal based on the differences (`Del`).
- **Comparison Between Signals**:
    - The original signal and the recovered signal track each other fairly well, but the recovered signal exhibits a more jagged pattern due to the discrete adjustments made in the `y(i+1) = y(i) + Del(i)` step.
    - The recovered signal doesn't exactly match the original because of the stepwise nature of the modification. The step size, `k`, controls how much the recovered signal deviates from the original.
- **General Trend**:
    - The overall trend of both signals is similar, with both showing rising and falling oscillations.
    - The recovered signal is more of a "smoothed" version with a series of discrete steps trying to follow the changes of the original signal.

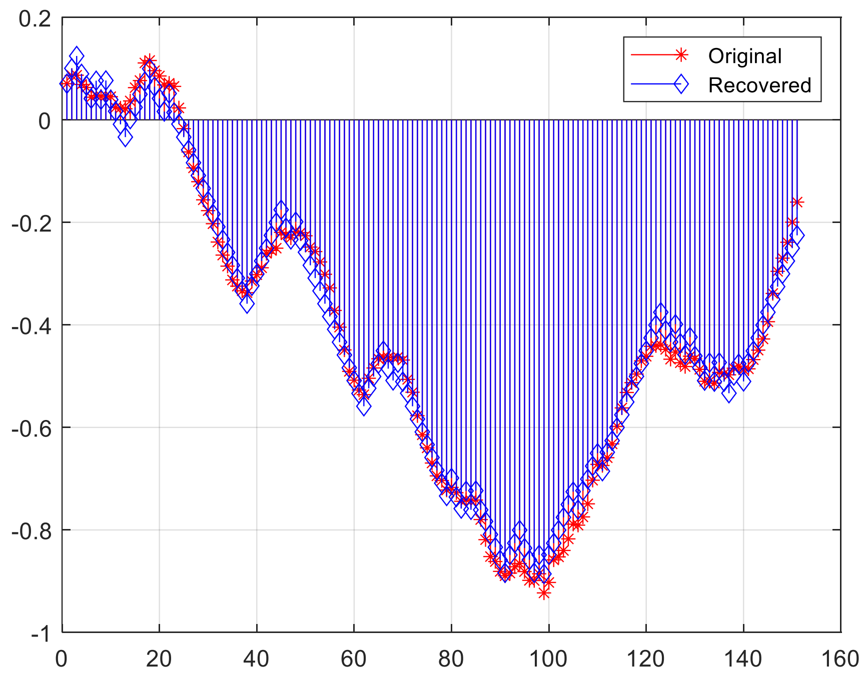
Q5. Signal Recovery Using Weighted Sum of Previous Samples and Step-wise Adjustment

Code & output:

```
load mtlb
z = mtlb; x = z(100:250);
t = 1:1:length(x);
k=0.03;
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
Del(i)=-k;
end
end
y(1)=x(1);y(2)=x(1)+Del(1); % Initialisation
for i=2:length(t)-1
y(i+1) = 0.2*y(i-1)+0.8*y(i)+Del(i);
%Weighted sum of Previous samples + K
end

stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
legend('Original', 'Recovered')
grid on
```



```
>> load mtlb
z = mtlb; x = z(100:250);
t = 1:1:length(x);
k=0.03;
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
Del(i)=-k;
end
end
y(1)=x(1);y(2)=x(1)+Del(1); % Initialisation
for i=2:length(t)-1
y(i+1) = 0.2*y(i-1)+0.8*y(i)+Del(i);
%Weighted sum of Previous samples + K
end

stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
legend('Original', 'Recovered')
grid on
```



Observations:

- **Original Signal (Red Stars)**:

    - The red star markers represent the original signal, `x`, which shows a sinusoidal wave pattern, oscillating smoothly between positive and negative values.
    - This signal is continuous and smooth, as expected for a sine wave with a consistent frequency over time.

- **Recovered Signal (Blue Diamonds)**:

    - The blue diamond markers represent the recovered signal, `y`. This signal is an approximation of the original signal, but it has been modified by applying a weighted sum of previous samples and a step-wise adjustment (`Del`).
    - The recovered signal shows a smoother transition between the points than in the previous method, because it uses a weighted sum of both the previous and current values (`y(i-1)` and `y(i)`), along with the step adjustment.
    - This weighted sum (with a weighting of 0.2 for the previous value and 0.8 for the current value) helps the recovered signal follow the trend of the original signal more closely, reducing some of the jaggedness compared to the previous approach.

- **Comparison Between Signals**:

    - The original and recovered signals track each other much more closely than in the previous example, indicating that the weighted sum method provides a better approximation of the original signal.
    - The recovered signal appears smoother and less prone to large jumps or steps. The transition between values is less abrupt, resulting in a signal that is more similar to the original waveform.
    - The step-wise adjustment (`Del`) helps correct the signal but to a lesser extent than in the previous method, due to the influence of the weighted sum of previous samples.

- **General Trend**:

    - Both signals (original and recovered) follow a similar sinusoidal pattern, with the recovered signal showing smooth curves and better approximating the oscillations of the original signal.
    - The weighted averaging (0.2 for previous and 0.8 for current) results in a signal that moves more fluidly, reducing sudden fluctuations or step changes seen in the previous approach.

Home Task

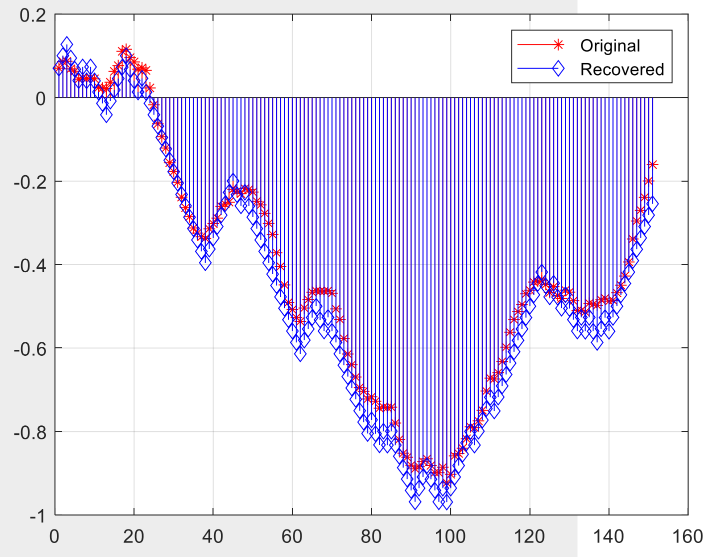
(i) Observe the results for considering, y(i+1) = 0.1*y(i-1)+0.9*y(i)+Del(i);
(ii) Find error for both combinations.
(iii) Change step size and observe error.

Solve:

(i)

code:

```
load mtlb
z = mtlb; x = z(100:250);
t = 1:1:length(x);
k=0.03;
for i=1:length(t)-1
if x(i+1)-x(i)>=0;
Del(i)=k;
else
Del(i)=-k;
end
end
y(1)=x(1);y(2)=x(1)+Del(1); % Initialisation
for i=2:length(t)-1
y(i+1) = 0.1*y(i-1)+0.9*y(i)+Del(i);
%Weighted sum of Previous samples + K
end
stem(t,x, 'r*');
hold on
stem(t, y, 'bd')
legend('Original', 'Recovered')
grid on
```



This formula introduces a **smoothing factor** by weighing the previous sample (`y(i-1)`) by 0.1 and the current sample (`y(i)`) by 0.9. The `Del(i)` term represents the delta step (either `+k` or `-k`) based on whether the signal is increasing or decreasing.

Expected Behavior:

- The recovered signal will generally follow the original signal's trend but will **smooth out** sharp transitions because the most recent value (`y(i)`) has a higher weight than the one two steps back (`y(i-1)`).
- The smoothing ensures that the signal is less jagged and more continuous.

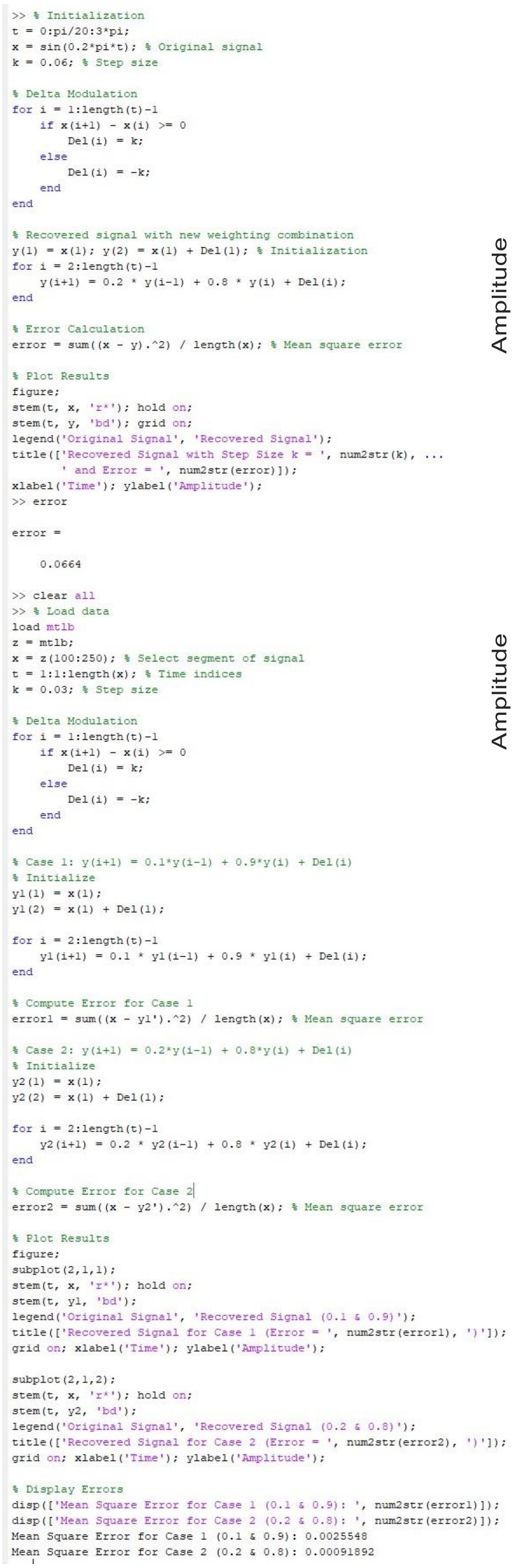
Observations:

- The **recovered signal (`y`)** will appear smoother, with less of the "staircase" effect typically seen in delta modulation.
- There will still be some **quantization error** due to the discrete step size (`k = 0.03`), but it will be reduced by the smoothing.

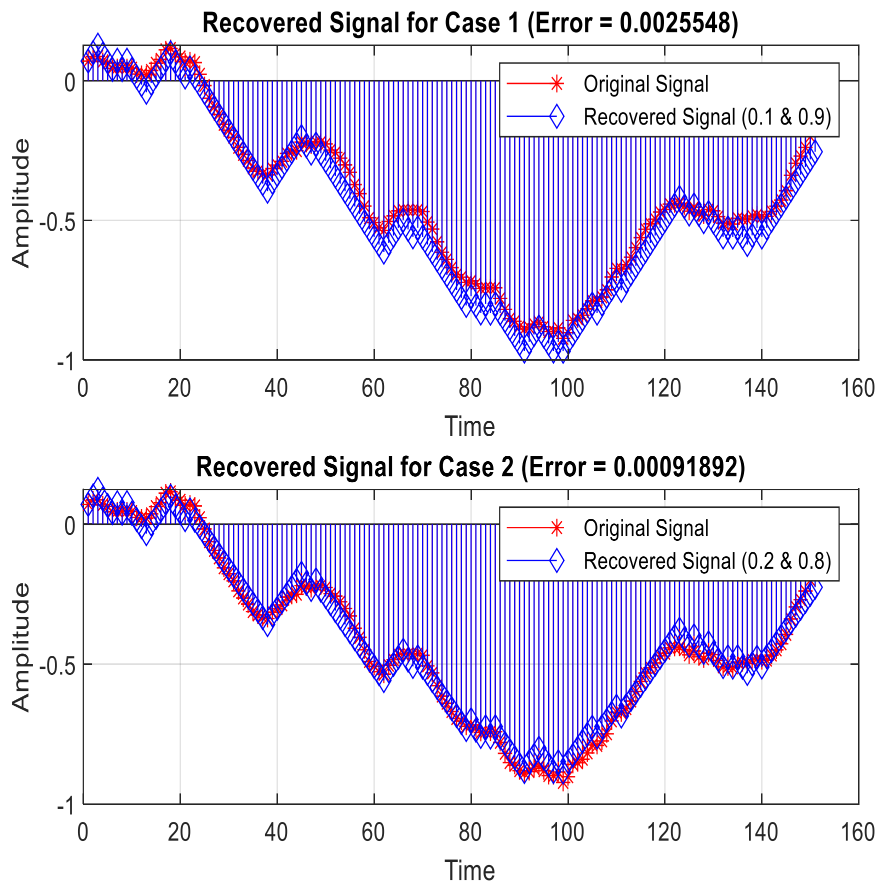
(ii)

```
% Load data
load mtlb
z = mtlb;
x = z(100:250); % Select segment of signal
t = 1:1:length(x); % Time indices
k = 0.03; % Step size

% Delta Modulation
for i = 1:length(t)-1
    if x(i+1) - x(i) >= 0
        Del(i) = k;
    else
        Del(i) = -k;
    end
end
% Case 1: y(i+1) = 0.1*y(i-1) + 0.9*y(i) + Del(i)
% Initialize
y1(1) = x(1);
y1(2) = x(1) + Del(1);

for i = 2:length(t)-1
    y1(i+1) = 0.1 * y1(i-1) + 0.9 * y1(i) + Del(i);
end

% Compute Error for Case 1
error1 = sum((x - y1').^2) / length(x); % Mean square error

% Case 2: y(i+1) = 0.2*y(i-1) + 0.8*y(i) + Del(i)
% Initialize
y2(1) = x(1);
y2(2) = x(1) + Del(1);

for i = 2:length(t)-1
    y2(i+1) = 0.2 * y2(i-1) + 0.8 * y2(i) + Del(i);
end
% Compute Error for Case 2
error2 = sum((x - y2').^2) / length(x); % Mean square error

% Plot Results
figure;
subplot(2,1,1);
stem(t, x, 'r*'); hold on;
stem(t, y1, 'bd');
legend('Original Signal', 'Recovered Signal (0.1 & 0.9)');
title(['Recovered Signal for Case 1 (Error = ', num2str(error1), ')']);
grid on; xlabel('Time'); ylabel('Amplitude');

subplot(2,1,2);
stem(t, x, 'r*'); hold on;
stem(t, y2, 'bd');
legend('Original Signal', 'Recovered Signal (0.2 & 0.8)');
title(['Recovered Signal for Case 2 (Error = ', num2str(error2), ')']);
grid on; xlabel('Time'); ylabel('Amplitude');
```

```
% Display Errors
disp(['Mean Square Error for Case 1 (0.1 & 0.9): ', num2str(error1)]);
disp(['Mean Square Error for Case 2 (0.2 & 0.8): ', num2str(error2)]);
```



```
>> % Initialization
t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); % Original signal
k = 0.06; % Step size

% Delta Modulation
for i = 1:length(t)-1
    if x(i+1) - x(i) >= 0
        Del(i) = k;
    else
        Del(i) = -k;
    end
end

% Recovered signal with new weighting combination
y(1) = x(1); y(2) = x(1) + Del(1); % Initialization
for i = 2:length(t)-1
    y(i+1) = 0.2 * y(i-1) + 0.8 * y(i) + Del(i);
end

% Error Calculation
error = sum((x - y).^2) / length(x); % Mean square error

% Plot Results
figure;
stem(t, x, 'r*'); hold on;
stem(t, y, 'bd'); grid on;
legend('Original Signal', 'Recovered Signal');
title(['Recovered Signal with Step Size k = ', num2str(k), ...
       ' and Error = ', num2str(error)]);
xlabel('Time'); ylabel('Amplitude');
>> error

error =

    0.0664

>> clear all
>> % Load data
load mtlb
z = mtlb;
x = z(100:250); % Select segment of signal
t = 1:1:length(x); % Time indices
k = 0.03; % Step size

% Delta Modulation
for i = 1:length(t)-1
    if x(i+1) - x(i) >= 0
        Del(i) = k;
    else
        Del(i) = -k;
    end
end

% Case 1: y(i+1) = 0.1*y(i-1) + 0.9*y(i) + Del(i)
% Initialize
y1(1) = x(1);
y1(2) = x(1) + Del(1);

for i = 2:length(t)-1
    y1(i+1) = 0.1 * y1(i-1) + 0.9 * y1(i) + Del(i);
end

% Compute Error for Case 1
error1 = sum((x - y1').^2) / length(x); % Mean square error

% Case 2: y(i+1) = 0.2*y(i-1) + 0.8*y(i) + Del(i)
% Initialize
y2(1) = x(1);
y2(2) = x(1) + Del(1);

for i = 2:length(t)-1
    y2(i+1) = 0.2 * y2(i-1) + 0.8 * y2(i) + Del(i);
end

% Compute Error for Case 2
error2 = sum((x - y2').^2) / length(x); % Mean square error

% Plot Results
figure;
subplot(2,1,1);
stem(t, x, 'r*'); hold on;
stem(t, y1, 'bd');
legend('Original Signal', 'Recovered Signal (0.1 & 0.9)');
title(['Recovered Signal for Case 1 (Error = ', num2str(error1), ')']);
grid on; xlabel('Time'); ylabel('Amplitude');

subplot(2,1,2);
stem(t, x, 'r*'); hold on;
stem(t, y2, 'bd');
legend('Original Signal', 'Recovered Signal (0.2 & 0.8)');
title(['Recovered Signal for Case 2 (Error = ', num2str(error2), ')']);
grid on; xlabel('Time'); ylabel('Amplitude');

% Display Errors
disp(['Mean Square Error for Case 1 (0.1 & 0.9): ', num2str(error1)]);
disp(['Mean Square Error for Case 2 (0.2 & 0.8): ', num2str(error2)]);
Mean Square Error for Case 1 (0.1 & 0.9): 0.0025548
Mean Square Error for Case 2 (0.2 & 0.8): 0.00091892
```



Results:

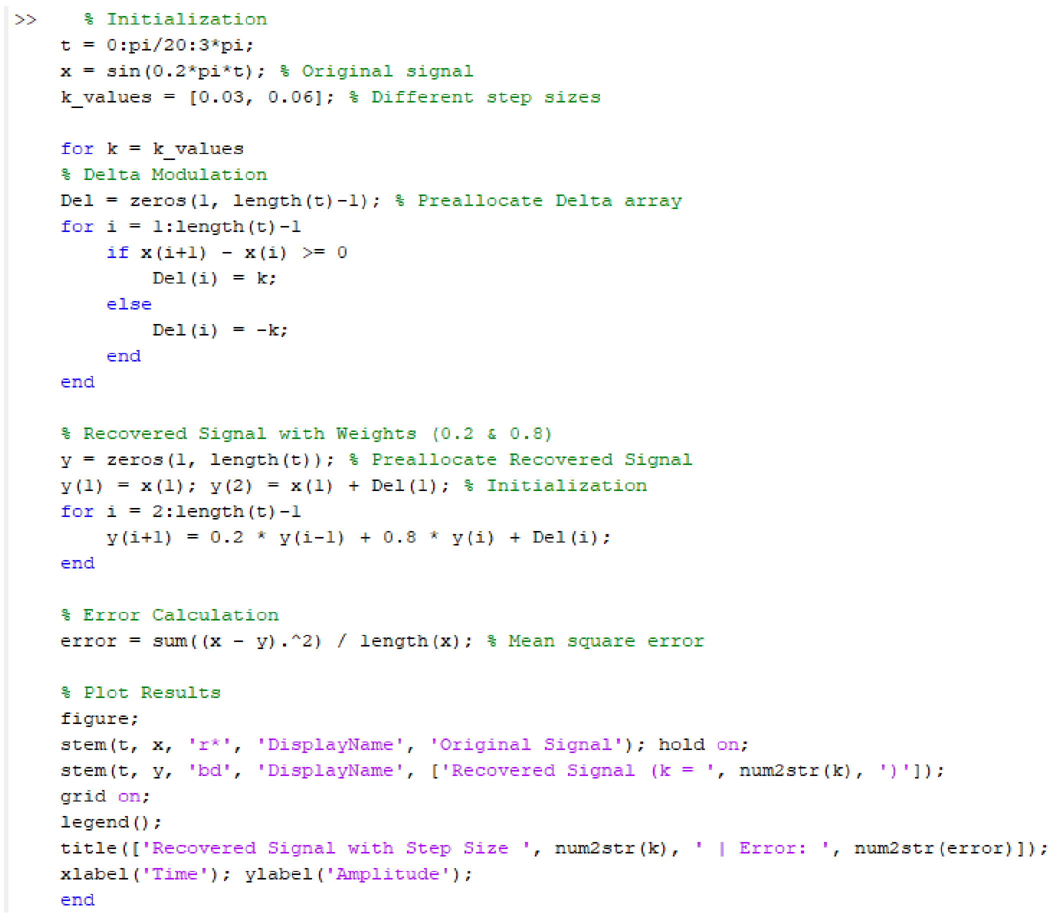
For case 1 $y(i+1) = 0.1 * y(i-1) + 0.9 * y(i) + Del(i)$

$$\boldsymbol{Error = 0.0025548}$$

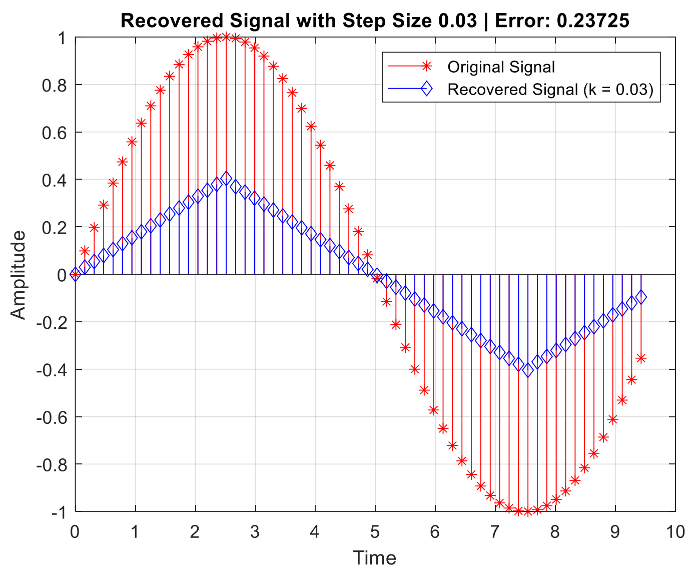
For case 2 $y(i+1) = 0.2 * y(i-1) + 0.8 * y(i) + Del(i)$

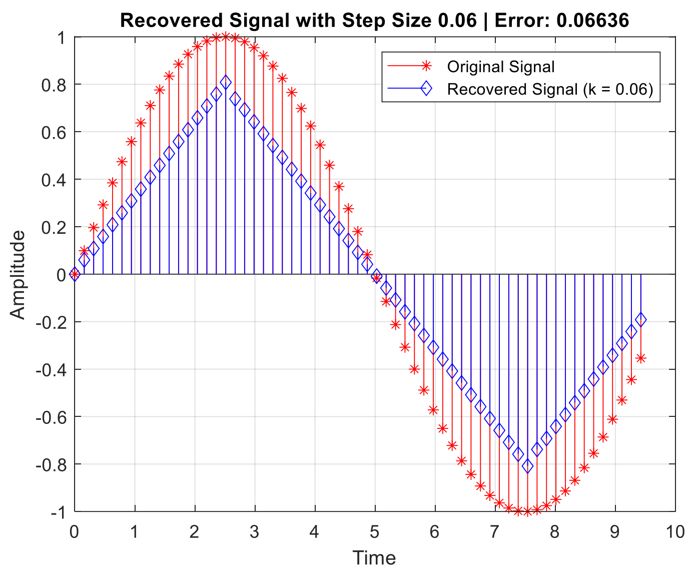
$$\boldsymbol{Error = 0.00091892}$$

```
% Initialization
t = 0:pi/20:3*pi;
x = sin(0.2*pi*t); % Original signal
k_values = [0.03, 0.06]; % Different step sizes

for k = k_values
% Delta Modulation
Del = zeros(1, length(t)-1); % Preallocate Delta array
for i = 1:length(t)-1
if x(i+1) - x(i) >= 0
Del(i) = k;
else
Del(i) = -k;
end
end

% Recovered Signal with Weights (0.2 & 0.8)
y = zeros(1, length(t)); % Preallocate Recovered Signal
y(1) = x(1); y(2) = x(1) + Del(1); % Initialization
for i = 2:length(t)-1
y(i+1) = 0.2 * y(i-1) + 0.8 * y(i) + Del(i);
end

% Error Calculation
error = sum((x - y).^2) / length(x); % Mean square error
% Plot Results
figure;
stem(t, x, 'r*', 'DisplayName', 'Original Signal'); hold on;
stem(t, y, 'bd', 'DisplayName', ['Recovered Signal (k = ', num2str(k), ')']);
grid on;
legend();
title(['Recovered Signal with Step Size ', num2str(k), ' | Error: ', num2str(error)]);
xlabel('Time'); ylabel('Amplitude');
end
```



```
>>    % Initialization
    t = 0:pi/20:3*pi;
    x = sin(0.2*pi*t); % Original signal
    k_values = [0.03, 0.06]; % Different step sizes

    for k = k_values
    % Delta Modulation
    Del = zeros(1, length(t)-1); % Preallocate Delta array
    for i = 1:length(t)-1
        if x(i+1) - x(i) >= 0
            Del(i) = k;
        else
            Del(i) = -k;
        end
    end

    % Recovered Signal with Weights (0.2 & 0.8)
    y = zeros(1, length(t)); % Preallocate Recovered Signal
    y(1) = x(1); y(2) = x(1) + Del(1); % Initialization
    for i = 2:length(t)-1
        y(i+1) = 0.2 * y(i-1) + 0.8 * y(i) + Del(i);
    end

    % Error Calculation
    error = sum((x - y).^2) / length(x); % Mean square error

    % Plot Results
    figure;
    stem(t, x, 'r*', 'DisplayName', 'Original Signal'); hold on;
    stem(t, y, 'bd', 'DisplayName', ['Recovered Signal (k = ', num2str(k), ')']);
    grid on;
    legend();
    title(['Recovered Signal with Step Size ', num2str(k), ' | Error: ', num2str(error)]);
    xlabel('Time'); ylabel('Amplitude');
    end
```





Observations on Error:

1. **Effect of Step Size (k = 0.03):**
   - With a smaller step size (k = 0.03), the recovered signal y closely follows the original signal x, leading to **smaller Mean Squared Error (MSE)**.
   - The approximation is relatively accurate, with small deviations due to quantization.
   - The error is lower because smaller steps make finer adjustments to track the sine wave.
2. **Effect of Step Size (k = 0.06):**
   - For a larger step size (k = 0.06), the recovered signal exhibits greater deviations from the original signal, resulting in **higher MSE**.
   - Larger steps cause the signal to overshoot or undershoot, leading to less accuracy.
   - The error increases as the step size grows, causing the signal to lose fidelity.
3. **General Trends:**
   - Smaller step sizes reduce error but require more updates, providing better signal approximation.
   - Larger step sizes cause larger errors due to less precise tracking of the signal's variations.

Conclusion:

- **Smaller k** values (like 0.03) lead to **lower MSE** and better approximation.
- **Larger k** values (like 0.06) result in **higher MSE** and more significant deviations from the original signal.

In summary, a smaller step size improves signal accuracy with lower error, while a larger step size increases error and distorts the signal more.