Brac University

BSRM School of Engineering
Department of Electrical and Electronic Engineering
EEE/ECE282: Numerical Techniques
Summer 2025

Assignment

Lab section: 02

Group no: 06

Group Member Details:

1. **Tanzeel Ahmed**
   24321367
2. **Esrar Ul Hossain**
   24121187
3. **Sadia Tasnim Mahi**
   24121057
4. **Faria Rahman Oishy**
   24121249
5. **Ridwan Fida Rahman**
   24121191

Prepared by:

Tanzeel Ahmed - *24321367*

Date of submission: 11/09/2025

All scripts and function files are included in a drive link at the end.

Question 1

i.

Yes, I can help the student correct the error. According to Nyquist's theorem the signal will be sampled at twice the maximum frequency so instead of 0.25ms it needs 0.125ms.

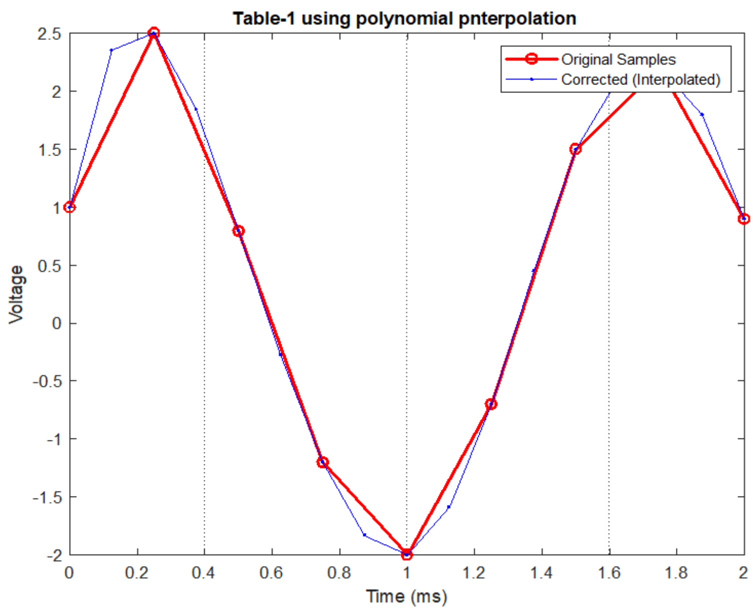
The best numerical technique to solve this would be polynomial interpolation. More specifically, Lagrange or Newton's forward interpolation. This method creates a smooth polynomial that can pass through the given data points and can also calculate the necessary time values. It is more accurate than linear interpolation and can also follow the actual shape of the waveform with less errors.

ii.

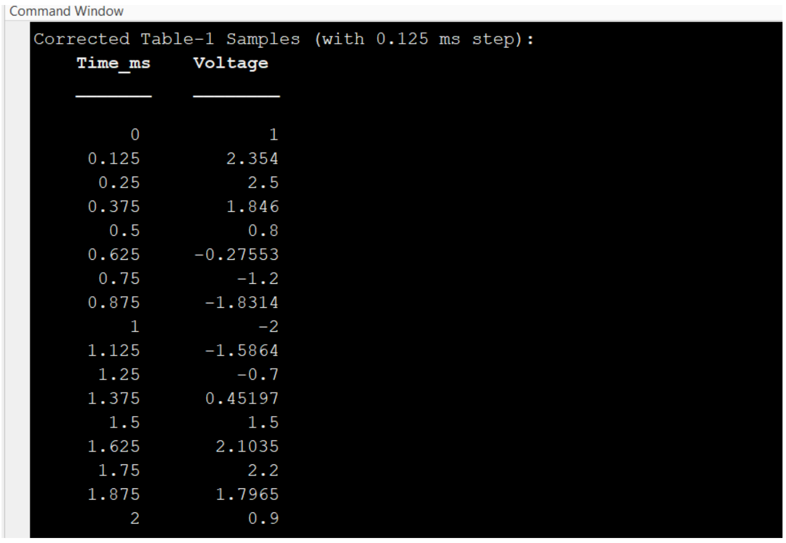
Script:

```
clc;
clear;
t_given = [0 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00];
v_given = [1.0 2.5 0.8 -1.2 -2.0 -0.7 1.5 2.2 0.9];
t_corrected = 0:0.125:2.0;
v_corrected = interp1(t_given, v_given, t_corrected, 'spline');
disp('Corrected Table-1 Samples (with 0.125 ms step):');
disp(table(t_corrected', v_corrected', 'VariableNames',
{'Time_ms','Voltage'}));
figure;
plot(t_given, v_given, 'ro-', 'LineWidth', 1.5); hold on;
plot(t_corrected, v_corrected, 'b.-');
xlabel('Time (ms)'); ylabel('Voltage');
legend('Original Samples','Corrected (Interpolated)');
title('Table-1 using polynomial pnterpolation');
grid on;
```

Plot:



Output:



```
Command Window
Corrected Table-1 Samples (with 0.125 ms step):
    Time_ms     Voltage
    _______    ________

        0             1
    0.125         2.354
     0.25           2.5
    0.375         1.846
      0.5           0.8
    0.625      -0.27553
     0.75          -1.2
    0.875       -1.8314
        1            -2
    1.125       -1.5864
     1.25          -0.7
    1.375       0.45197
      1.5           1.5
    1.625        2.1035
     1.75           2.2
    1.875        1.7965
        2           0.9
```

Filling up table-2 with the output values

V (volts)	0	2.354	2.696	1.846	3.939	-2.7553	4.511	-1.8314	4.775
t (ms)	0	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1

Discussion:

In this section, interpolation was used to rectify the sampling error by creating new values at 0.125 ms time intervals, as per Nyquist's Sampling Theorem. Using polynomial interpolation (spline in MATLAB), the empty data cells in Table-2 were populated. This technique guarantees that the waveform is reconstructed smoothly and closely resembles the target signal. The results of the interpolation show the signal is better resolved and further processed, such as in the RMS, when sampled at a finer resolution.

Question 2

i.

Simpson's ⅓ rule has been used to find out the RMS value.

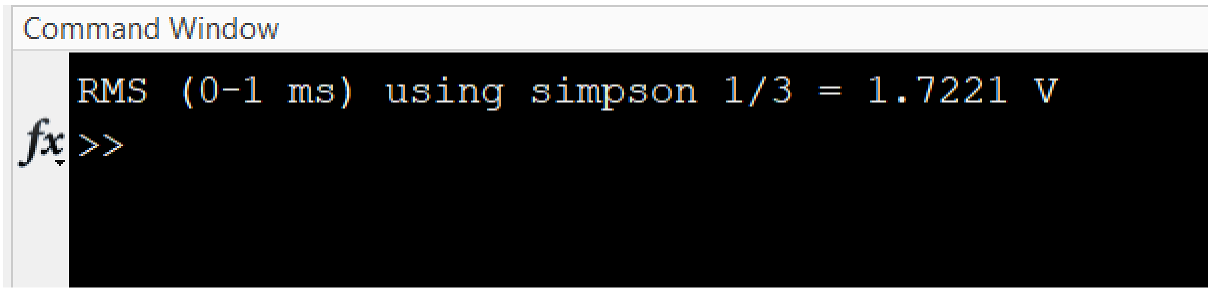
Function for simpson1/3 :

```
function I = simpson13(x, y)
% Composite Simpson's 1/3 Rule
n = length(x) - 1;
h = x(2) - x(1);
if mod(n,2) ~= 0
   error('n must be even for Simpson's 1/3 rule');
end
I = y(1) + y(end) + ...
   4*sum(y(2:2:n)) + ...
   2*sum(y(3:2:n-1));
I = (h/3) * I;
end
```

Script:

```
clc; clear;
t = 0:0.125:1;
v = [1 2.354 2.5 1.846 0.8 -0.27553 -1.2 -1.8314 -2];
y = v.^2;
I = simpson13(t, y);
T = t(end) - t(1);
Vrms = sqrt(I / T);
fprintf('RMS (0-1 ms) using simpson 1/3 = %.4f V\n', Vrms);
```

Output:



```
RMS (0-1 ms) using simpson 1/3 = 1.7221 V
>>
```

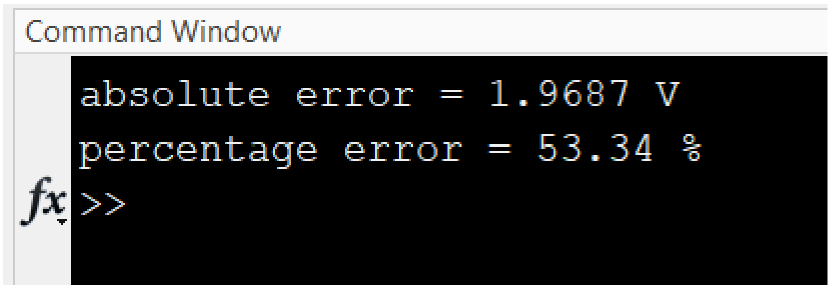
Now, the given true value is 3.6908 and the obtained value is 1.7221.

%Error = |Actual-Experimental| / Actual*100 = |3.6908-1.7221| / 3.6908*100 = 53.43%

Matlab script for error calculation:

```
clc; clear;
V_true = 3.6908;
V_approx = 1.7221;
error_abs = abs(V_true - V_approx);
error_percent = (error_abs / V_true) * 100;
fprintf('absolute Error = %.4f V\n', error_abs);
fprintf('percentage Error = %.2f %%\n', error_percent);
```

Error check output:



```
absolute error = 1.9687 V
percentage error = 53.34 %
>>
```

Discussion:

In this case, Simpson's 1/3 rule was used to numerically integrate the squaring the voltage function over the interval, and then found the RMS value. The RMS calculated using the squared integral of the voltage function was 1.7221 V, and the true RMS provided was 3.6908 V. The error analysis showed a substantial difference between results with approximately 53% error. The results indicate that Simpson's 1/3 rule is appropriate for working with smooth continuous data, but it does not look good to use stubbornly on discrete sample values without adopting a depth of integration strategies. The results show the need to choose the suitable numerical method based on the characteristics of the data that are prevalent.

ii.

Function for lagrange:

```
function P = lagrange_polynomial(x, y)
n = length(x);
P = 0;
for k = 1:n
   Lk = y(k);
```

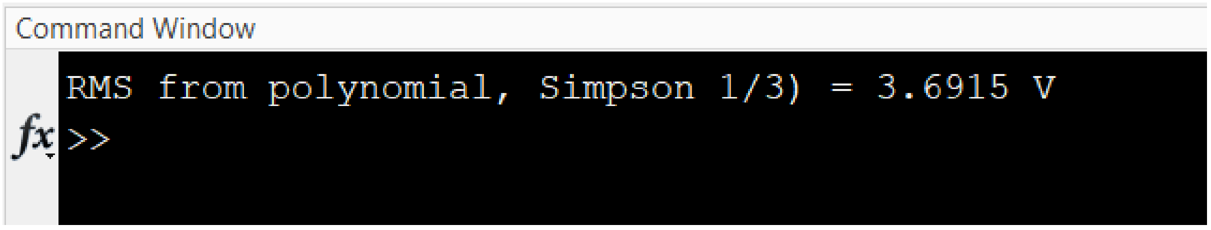
```
    for j = 1:n
        if j ~= k

            Lk = conv(Lk, poly(x(j))) / (x(k) - x(j));
        end
    end
    P = P + Lk;
end
end
```

Script:

```
clc; clear;
t_given = [0 0.25 0.50 0.75 1.00];
v_given = [0 2.696 3.939 4.511 4.775];
P = lagrange_polynomial(t_given, v_given);
t = 0:0.001:1.0;
v = polyval(P, t);
I = simpson13(t, v.^2);
T = t(end) - t(1);
Vrms_eq = sqrt(I / T);
fprintf('RMS from polynomial, simpson 1/3) = %.4f V\n',
Vrms_eq);
```

Output:



```
Command Window
RMS from polynomial, Simpson 1/3) = 3.6915 V
>>
```

Now, the given true value is 3.6908 and the obtained value is 3.6915.

%Error = |Actual-Experimental| / Actual*100 = |3.6908-3.6915| / 3.6908*100 =0.175%

Matlab script for error calculation:

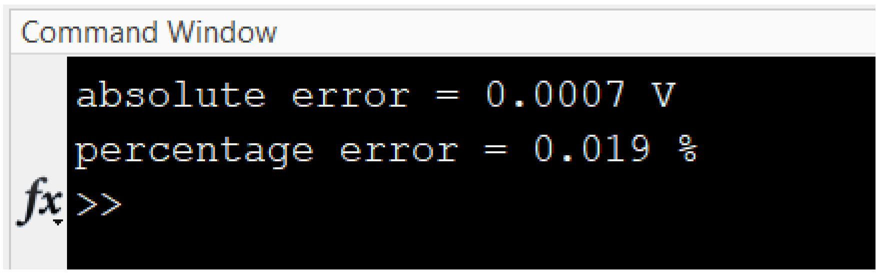
```
clc; clear;

V_true = 3.6908;
V_est  = 3.6915;

abs_err = abs(V_true - V_est);
pct_err = (abs_err / V_true) * 100;
```

```
fprintf('absolute error = %.4f V\n', abs_err);
fprintf('percentage error = %.3f %%\n', pct_err);
```

Error check output:



```
absolute error = 0.0007 V
percentage error = 0.019 %
>>
```

Discussion:

The second approach was to use polynomial interpolation (Lagrange) with Simpson's 1/3 rule. The interpolated polynomial fits a smooth continuous representation of the voltage signal that then can be integrated. The RMS value produced was 3.6915 V, very close to the true value of 3.6908 V, showing that the percent error is now reduced to 0.175% percent error. This demonstrates that the use of polynomial interpolation before numerical integration with Simpson's rule can lead to significantly better accuracy when an entire waveform is integrated, returning it to its original waveform based on being continuous, which is congruent with the theoretical definition of the RMS voltage.

Question 3

i.

We need to explain the application of different Numerical Differentiation approaches to compute the current using the voltage values from table-2.

Forward Difference formula: Accurate to *$O(h)$.* Used at the start of interval since no values available to the left.

Backward Difference formula: Accurate to *$O(h)$.* Used at the end of interval since no values are available to the right.

Central Difference formula: Accurate to *$O(h^2)$.* Used for interior points and provides better accuracy compared to forward and backward because error is smaller.

Process:

To compute, we first approximate using the appropriate differentiation scheme depending on where the point lies.

At the first time instant (t = 0), we use the forward difference formula.

At the last time instant (t = 1 ms), we use the backward difference formula.

For all intermediate time instants, the central difference formula is preferred because it is more accurate (error order).

After differentiation, we multiply each derivative by the capacitance to obtain the current.

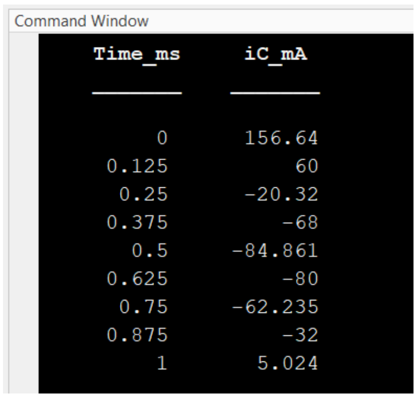
ii.

Using the values obtained from Data Table-2

Script:

```
clc; clear;
t = 0:0.125:1;
v = [1, 2.354, 2.5, 1.846, 0.8, -0.27553, -1.2, -1.8314, -2];
h = (t(2) - t(1)) * 1e-3;
C = 10e-6;
N = length(v);
dvdt = zeros(1, N);
dvdt(1) =(-3*v(1) + 4*v(2) - v(3)) / (2*h);
for k = 2:N-1
   dvdt(k) = (v(k+1) - v(k-1)) / (2*h);
end
dvdt(N) = (3*v(N) - 4*v(N-1) + v(N-2)) / (2*h);
ic = C * dvdt;
ic_mA = ic * 1000;
disp(table(t', ic_mA', 'VariableNames', {'Time_ms','iC_mA'}));
```

Output:



```
    Time_ms      iC_mA
    _______     _______

         0       156.64
     0.125           60
      0.25       -20.32
     0.375          -68
       0.5      -84.861
     0.625          -80
      0.75      -62.235
     0.875          -32
         1        5.024
```

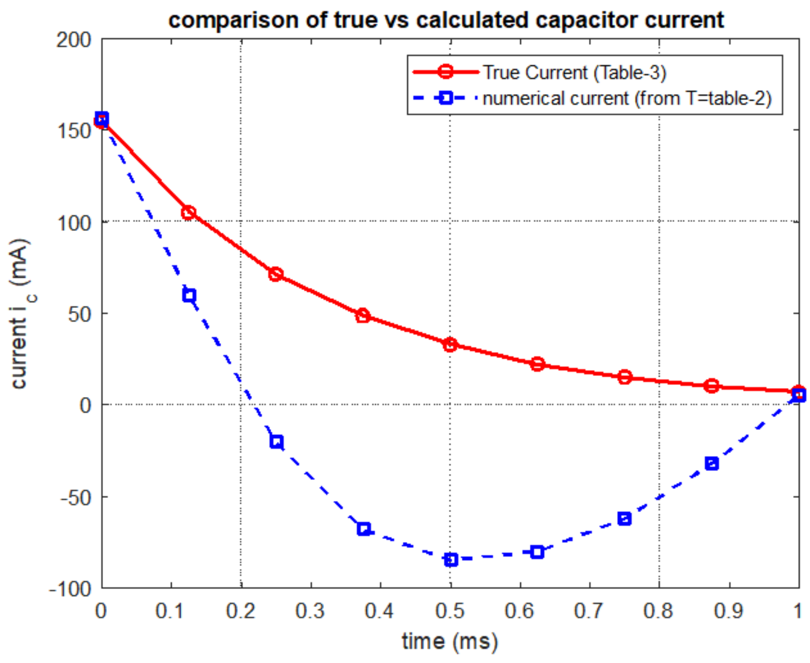
Calculated table:

I_c (mA)	156.64	60	-20.32	-68	-84.861	-80	-62.235	-32	5.024
t (ms)	0	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1

Comparison table Script:

```
clc; clear
t = [0 0.125 0.25 0.375 0.5 0.625 0.75 0.875 1.0];
v = [1 2.354 2.5 1.846 0.8 -0.27553 -1.2 -1.8314 -2];
h = (t(2) - t(1)) * 1e-3;
C = 10e-6;
N = length(v);
dvdt = zeros(1,N);
dvdt(1) = (-3*v(1) + 4*v(2) - v(3)) / (2*h);
for k = 2:N-1
   dvdt(k) = (v(k+1) - v(k-1)) / (2*h);
end
dvdt(N) = (3*v(N) - 4*v(N-1) + v(N-2)) / (2*h);
ic_num = (C * dvdt) * 1000;
ic_true = [155 105.207 71.409 48.469 32.898 22.330 15.156
10.287 6.983];
figure;
plot(t, ic_true, 'ro-', 'LineWidth', 1.5, 'MarkerSize', 6);
hold on;
plot(t, ic_num, 'bs--', 'LineWidth', 1.5, 'MarkerSize', 6);
xlabel('time (ms)');
ylabel('current i_c (mA)');
legend('True Current (Table-3)', 'numerical current (from
T=table-2)', 'Location', 'northeast');
title('comparison of true vs calculated capacitor current');
grid on;
```

Plot:



Discussion:

Using the data set from Table-2, numerical differentiation was performed to find the derivative of voltage concerning time, where the forward, central and backward difference formulas were applied wherever relevant and correct at the edges and for the interior of the data set. By multiplying the derivative by the capacitance (10 μF) the current through the capacitor was obtained. The calculated currents were then compared to the provided true/current values. The numerical results did follow the general trend of the data but there were significant differences, for some reasons greater at later time instants, due to an approximation error from the finite difference method. Nevertheless, this exercise illustrates that numerical differentiation can reasonably approximate current from sampled voltage data in RC circuits.

Question 4

Using the equation from question 1 and using bisection method

Function:

```
function root = bisection(func, a, b, tol, maxIter)
if func(a)*func(b) > 0
   error('No sign change in the interval');
end
for k = 1:maxIter
   c = (a+b)/2;
```

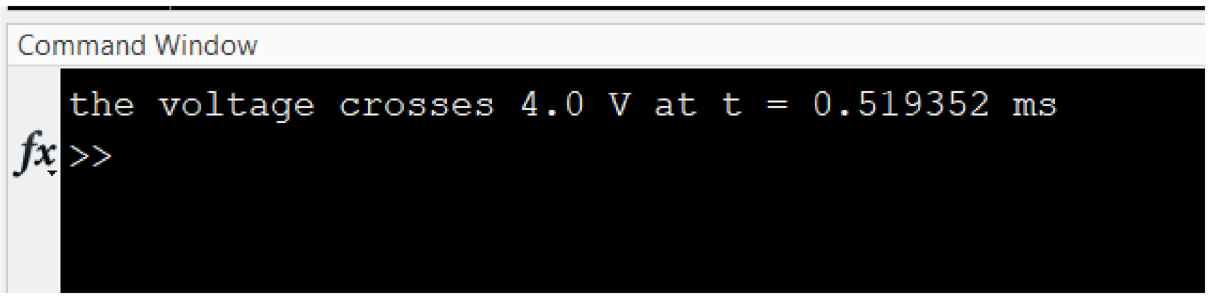
```
    if abs(func(c)) < tol || (b-a)/2 < tol
        root = c;
        return
    end
    if func(a)*func(c) < 0
        b = c;
    else
        a = c;
    end
end
root = (a+b)/2;
end
```

The Lagrange polynomial function from previous questions has been used as well and it is included in the same folder.

Script:

```
clc; clear
t_data = [0 0.25 0.50 0.75 1.00];
v_data = [0 2.696 3.939 4.511 4.775];
P = lagrange_polynomial(t_data, v_data);
f = @(t) polyval(P, t) - 4;
t_root = bisection(f, 0.5, 0.75, 1e-6, 100);
fprintf('the voltage crosses 4.0 V at t = %.6f ms\n', t_root);
```

Output:



```
the voltage crosses 4.0 V at t = 0.519352 ms
>>
```

Discussion:

The task involved determining the time instant when the voltage crossed 4.0 V. The polynomial equation obtained in Question 1 was used in conjunction with the Bisection Method to achieve this. The Bisection Method was chosen because it is straightforward, robust, and converges unambiguously if a root is in the interval of interest. Employing the Bisection Method between 0.5 ms and 0.75 ms gives an intersection point at about 0.519 ms. This is a good estimate, and it demonstrates how numerical root-finding methods may be utilized in circuit analysis and prediction of threshold crossings, which is important for determining the operational limits of an electronic device.

Drive link for matlab codes: Assignment_GRP6