GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

MATH214 – NUMERICAL ANALYSIS

Alican Bayındır
200102002087

Project - 4

Preparation Date
23.12.2020
Upload Date
25.12.2020

Problem definition and formulas;

Transient response, which consists of natural and step responses, of an RL circuit with a DC voltage source V_S, an inductance L, and a resistance R, as shown in Figure 1, can be determined by solving the following ordinary differential equation (ODE) with the initial current i_0 as the initial condition:

$$V_S = L\frac{d}{dt}i(t) + Ri(t)\,, i(t_0) = i_0$$

Where i(t) is the current. In Figure 1, the switch is closed at $t = t_0$.

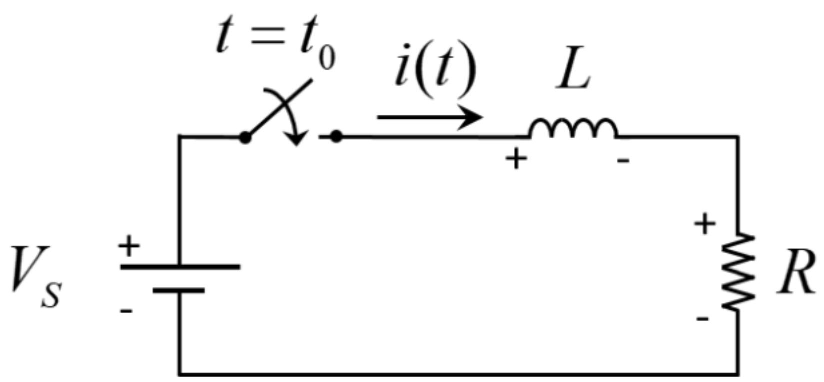


Figure 1 An RL circuit.

In this Project, we are asked to apply the following methods to calculate the current values numerically;

- Euler's method
- Modified Euler's method
- Midpoint method
- Runge-Kutta method (order four)

EULER'S METHOD;

Euler's Method, is a method to analyze a Differential equation, which uses the idea of local linearith or linear approximation, where we use small tangent lines over a short distance to approximate the solution to an initial-value problem.

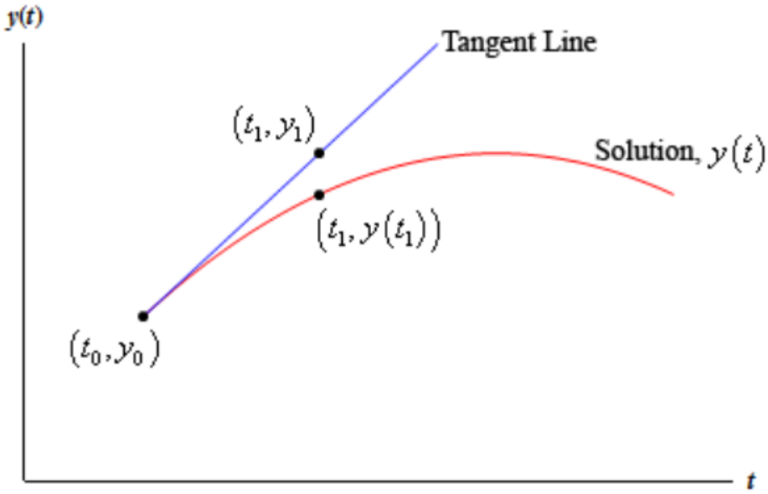


Figure 2 Tangent line and solution line's graph.

It is better how much ever t_1 is close to t_0 since the actual value of the solution at t_1 or y(t1). "y_1" can be calculated easily. Only thing needed to be done is to put t_1 in the equation of the tangent line. As a result of this, equation becomes;

$$W_{i+1} = W_i + f(t_i, y_i)(t_{i+1} - t_i)$$

The only thing needed here is W_i value of the equation. The other values can be found by using first value of the W.

MODIFIED EULER'S METHOD;

Euler's method can not give accurate results everytime since it is the most basic numerical analysis method to calculate numerical integration of ordinary diferential equations. The objective in numerical methods is, as always, finding the most accurate result with the minimum effort. For integrating the initial value problem the effort is usually measured by the number of times the function f(t, y) must be evaluated in stepping from a to b. As we will see, a simple improvement doubles the number of function evaluations per step, but yields a second order method.

$$W_{i+1} = W_i + \frac{h}{2}[f(t_i, W_i) + f\left(t_{i+1}, W_i + hf(t_i, W_i)\right)]$$

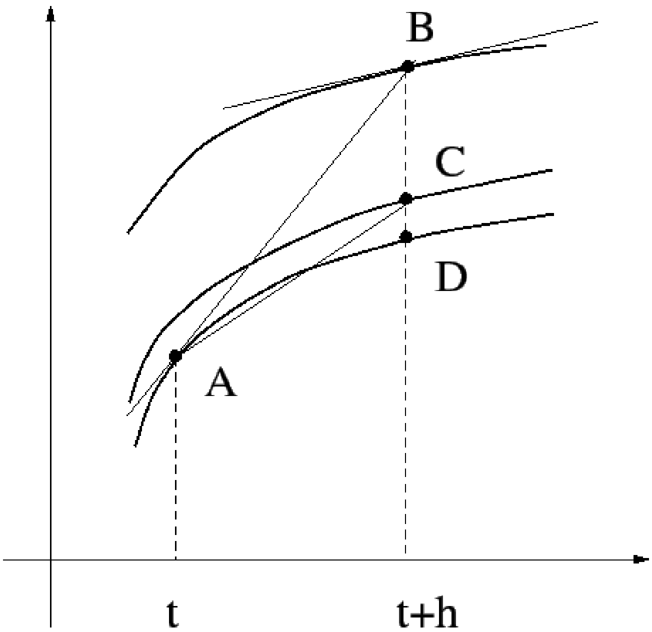


Figure 3 Graphical representation of the Euler and modified Euler method. (Point B is the result of Euler's Method and point C is the result of the Modified Euler's method)

It can be seen in the Figure 3 that the Modified Euler's method gives more accurate results than Euler's Method.

MIDPOINT METHOD;

Runge-Kutta method in second order is also called Midpoint Method. The Midpoint Method finds an approximate value of y for a given x. Only first-order ordinary differential equations can be solved by using The Midpoint method. The equation to calculate it is;

$$W_{i+1} = W_i + \ h[f(t_i + \frac{h}{2}, W_i + \frac{h}{2} f(t_i, W_i))$$

RUNGE-KUTTA METHOD (Fourth Order);

Runge–Kutta method is an effective and widely used method for solving the initial-value problems of differential equations. The Runge-Kutta method attempts to overcome the problem of the Euler's method, as far as the choice of a sufficiently small step size is concerned, to reach a reasonable accuracy in the problem resolution. Runge–Kutta method can be used to construct high order accurate numerical method by functions' self without needing the high order derivatives of functions. This method gives more accurate results than other methods and we can show its equation as follows;

$$k_1 = hf(t_i, w_i)$$

$$k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1$$

$$k_3 = hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2\right)$$

$$k_4 = hf(t_{i+1}, w_i + k_3)$$

$$W_{i+1} = W_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

We use one point's result to calculate the next point's result. In every step, the error rate we get from the results increases since we use previous step's result with its error. The error rate increases continuously in every step.

Codes, Inputs and Discussion;

First, Euler's method can be described in MATLAB platform as follows;

```
% Euler's method
% Step size = 0.05;
for k = 1:length(time_half)-1
    current_euler_half(k+1) = current_euler_half(k) + STEP_SIZE_1 * F(current_euler_half(k));
end

% Step size = 0.025;
for k = 1:length(time_quarter)-1
    current_euler_quarter(k+1) = current_euler_quarter(k) + STEP_SIZE_2 * F(current_euler_quarter(k));
end
```

Figure 4 Euler's Method on MATLAB Platform.

Where;

current_euler_half(1) = 0.1,

STEP_SIZE_1 (constant) = 0.05.

current_euler_quarter(1) = 0.1,

STEP_SIZE_2 (constant) = 0.025,

Length(time_half) = 13,

Length(time_quarter) = 26.

As a result of this part of the code is to get the results of Euler's method for step size 0.05 and 0.025. The plot of these two values can be seen in Figure 5.

The second method used to calcate the ODE is Modified Euler's method it can be described in MATLAB platform as seen in Figure 5;

```
% Modified Euler's Method
for k = 1:length(time_half)-1
    current_modieuler_half(k+1) = current_modieuler_half(k) + (STEP_SIZE_1 / 2) * (F(current_modieuler_half(k)) + STEP_SIZE_1 * F(current_modieuler_half(k)));
end

% Modified Euler's Method
for k = 1:length(time_quarter)-1
    current_modieuler_quarter(k+1) = current_modieuler_quarter(k) + (STEP_SIZE_2 / 2) * (F(current_modieuler_quarter(k)) + STEP_SIZE_2 * F(current_modieuler_quarter(k)));
end
```

Figure 5 Modified Euler's method on MATLAB platform.

Where all the variables are same as specified below the Figure 4.

The third method used in this Project is Midpoint Method. It can be seen in Figure 6.

```
% Midpoint Method
for k = 1:length(time_half)-1
    current_midpoint_half(k+1)= current_midpoint_half(k) + STEP_SIZE_1 * (F(current_midpoint_half(k)) + (STEP_SIZE_1 / 2) * F(current_midpoint_half(k)));
end

% Midpoint Method
for k = 1:length(time_quarter)-1
    current_midpoint_quarter(k+1)= current_midpoint_quarter(k) + STEP_SIZE_2 * (F(current_midpoint_quarter(k)) + (STEP_SIZE_2 / 2) * F(current_midpoint_quarter(k)));
end
```

Figure 6 Midpoint Method on MATLAB Platform.

The last but not the least one is Runge-Kutta Method it gives the most accurate results. The description of the method in MATLAB Platform can be seen in Figure 7 below.

```
% Runge Kutta Method Fourth Order
% Step size = 0.05;
for k = 1:length(time_half)-1
    runge_kutta_half1 = STEP_SIZE_1 * F(current_runge_kutta_half(k));
    runge_kutta_half2 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half1 / 2);
    runge_kutta_half3 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half2 / 2);
    runge_kutta_half4 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half3);
    current_runge_kutta_half(k + 1) = current_runge_kutta_half(k) + (runge_kutta_half1 + 2 * runge_kutta_half2 + 2 * runge_kutta_half3 + runge_kutta_half4) / 6;
end

% Step size = 0.025
for k = 1:length(time_quarter)-1
    runge_kutta_quarter1 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k));
    runge_kutta_quarter2 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) + runge_kutta_quarter1 / 2);
    runge_kutta_quarter3 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) + runge_kutta_quarter2 / 2);
    runge_kutta_quarter4 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) + runge_kutta_quarter3);
    current_runge_kutta_quarter(k + 1) = current_runge_kutta_quarter(k) + (runge_kutta_quarter1 + 2 * runge_kutta_quarter2 + 2 * runge_kutta_quarter3 + runge_kutta_quarter4) / 6;
end
```

Figure 7 Runge-Kutta Method on MATLAB Platform.

GEBZE TECHNICAL UNIVERSITY
ELECTRONIC ENGINEERING
2020-2021 FALL TERM
MATH 214 – NUMERICAL ANALYSIS – PROJECT 4

The results of each Method's when Δt = 0.05;

Iteration	Real Values	Euler's Method	Modi. Euler's Method	Midpoint Method	Runge-Kutta Method
1	0.1	0.1	0.1	0.1	0.1
2	0.5356	0.6398	0.3834	0.6533	0.4829
3	0.7466	0.7885	0.559	0.7957	0.669
4	0.8489	0.8295	0.6678	0.8324	0.7595
5	0.8985	0.8408	0.7352	0.8418	0.8035
6	0.9225	0.8439	0.777	0.8442	0.8249
7	0.9341	0.8447	0.8029	0.8449	0.8352
8	0.9398	0.845	0.8189	0.845	0.8403
9	0.9425	0.845	0.8289	0.8451	0.8427
10	0.9438	0.8451	0.835	0.8451	0.8439
11	0.9445	0.8451	0.8389	0.8451	0.8445
12	0.9448	0.8451	0.8412	0.8451	0.8448
13	0.9449	0.8451	0.8427	0.8451	0.8449

The results of each Method's when Δt = 0.025.

Iteration	Real value	Euler's Method	Modi. Euler's Method	Midpoint Method	Runge-Kutta Method
1	0.1	0.1	0.1	0.1	0.1
2	0.3568	0.3699	0.2383	0.3733	0.3264
3	0.5356	0.542	0.351	0.5463	0.484
4	0.66	0.6518	0.4427	0.6559	0.5937
5	0.7466	0.7218	0.5174	0.7253	0.6701
6	0.8069	0.7665	0.5782	0.7692	0.7232
7	0.8489	0.7949	0.6278	0.797	0.7603
8	0.8781	0.8131	0.6681	0.8147	0.786
9	0.8985	0.8247	0.701	0.8258	0.804
10	0.9126	0.8321	0.7277	0.8329	0.8165
11	0.9225	0.8368	0.7495	0.8373	0.8252
12	0.9294	0.8398	0.7672	0.8402	0.8312
13	0.9341	0.8417	0.7817	0.842	0.8354
14	0.9375	0.8429	0.7935	0.8431	0.8383
15	0.9398	0.8437	0.803	0.8438	0.8404
16	0.9414	0.8442	0.8108	0.8443	0.8418
17	0.9425	0.8445	0.8172	0.8446	0.8428
18	0.9433	0.8447	0.8224	0.8448	0.8435
19	0.9438	0.8448	0.8266	0.8449	0.844
20	0.9442	0.8449	0.83	0.8449	0.8443
21	0.9445	0.845	0.8328	0.845	0.8445
22	0.9447	0.845	0.8351	0.845	0.8447
23	0.9448	0.845	0.8369	0.845	0.8448
24	0.9449	0.845	0.8385	0.8451	0.8449

25	0.9449	0.8451	0.8397	0.8451	0.8449

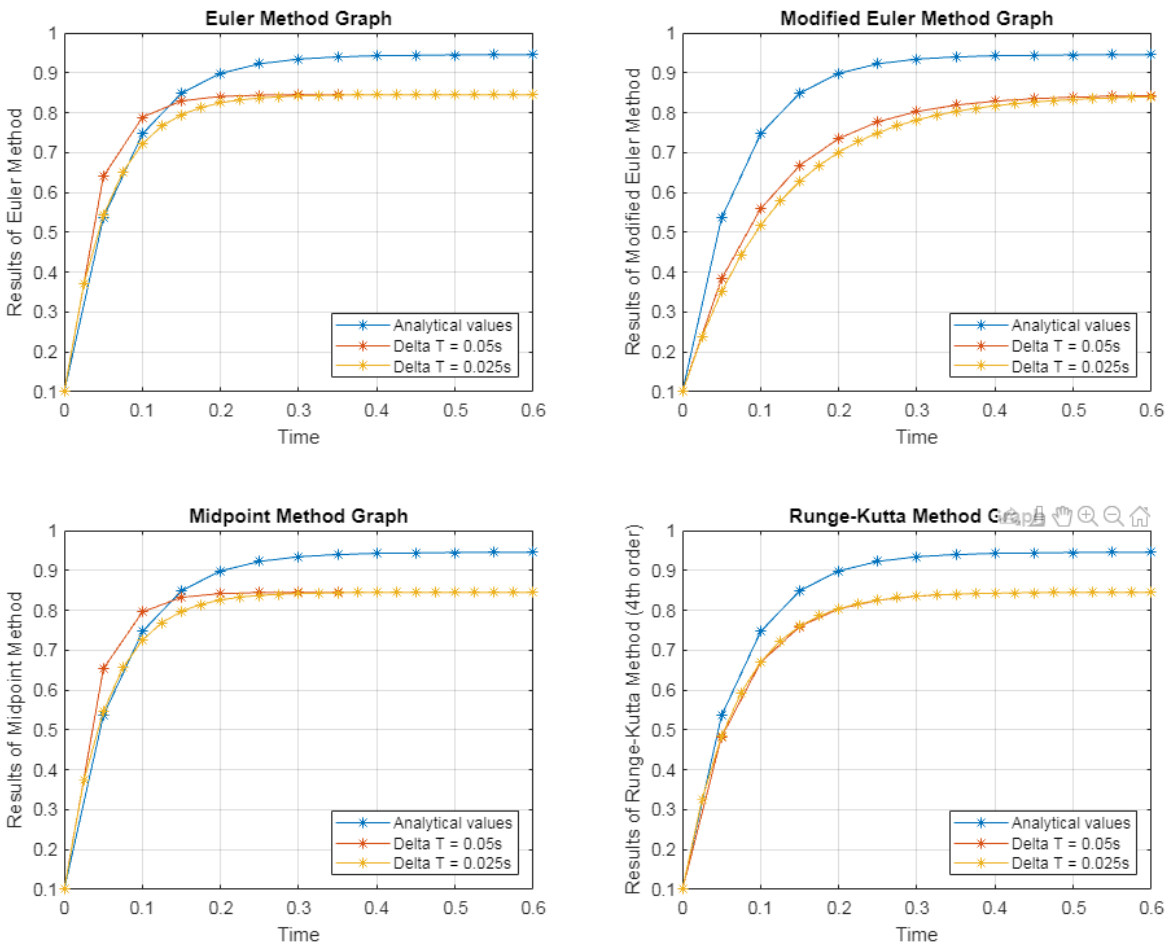
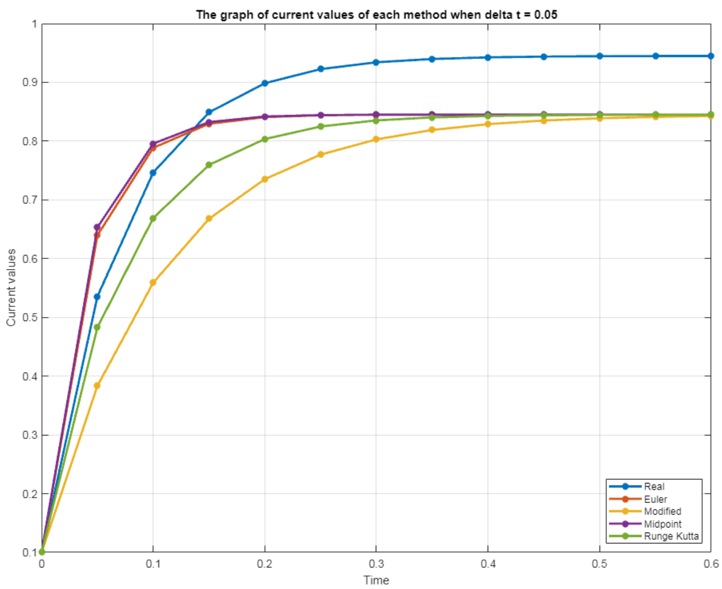


Figure 9 The comparison of all methods step size difference.



Here, as seen in Figure 10, all methods converges to real value until the time around 0.15 but then their error rates increases and creates more errors. It causes this deviation from the real value in the graph.

Figure 10 The comparison of all methods' results in same plot when delta t = 0.05.

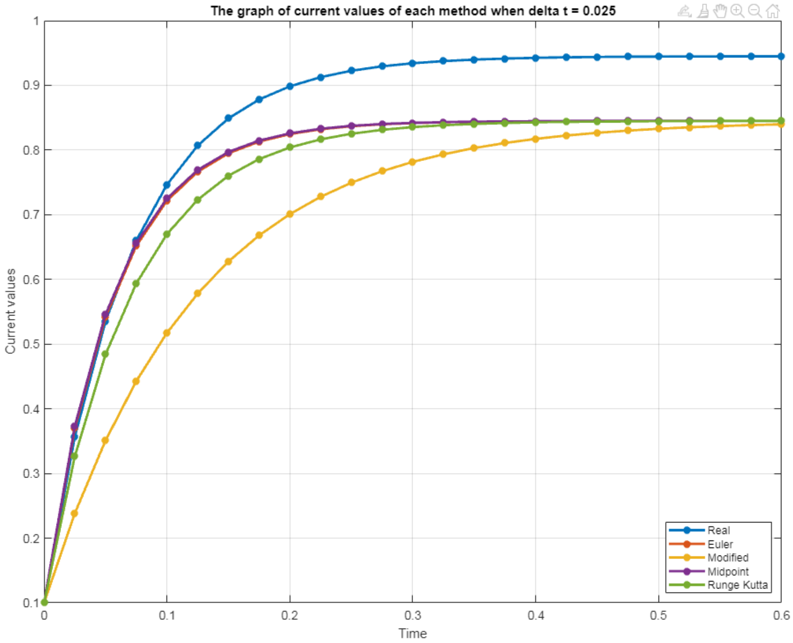


Figure 11 shows all methods converges to real value until the time around 0.1 but then their error rates increases and creates more errors. It causes this deviation from the real value in the graph.

Figure 11 The comparison of all methods' results in same plot when delta t = 0.025.

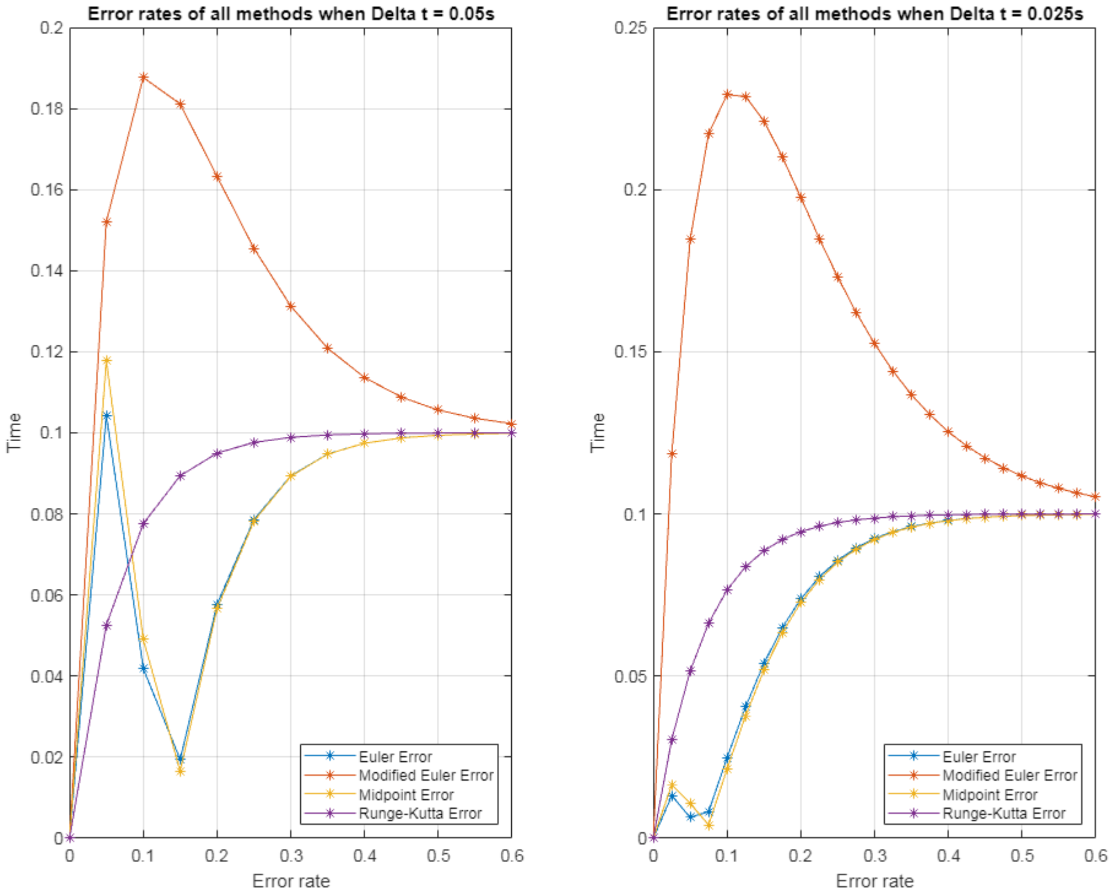


Figure 12 The comparison of error rates of all methods.

Error rates of all methods when delta t = 0.05s can be seen in table 3 below;

Iteration	Euler's Method	Modi. Euler's Method	Midpoint Method	Runge-Kutta Method
1	0	0	0	0
2	0.1042	0.1522	0.1177	0.0526
3	0.0419	0.1876	0.0491	0.0776
4	0.0194	0.1811	0.0166	0.0894
5	0.0577	0.1632	0.0567	0.095
6	0.0786	0.1455	0.0783	0.0976
7	0.0894	0.1312	0.0893	0.0989
8	0.0948	0.1208	0.0948	0.0995
9	0.0975	0.1136	0.0974	0.0998
10	0.0988	0.1088	0.0988	0.0999
11	0.0994	0.1056	0.0994	0.0999
12	0.0997	0.1036	0.0997	0.1
13	0.0999	0.1022	0.0999	0.1

Error rates of all methods when delta t = 0.025s can be seem in table 4 below;

Iteration	Euler's Method	Modi. Euler's Method	Midpoint Method	Runge-Kutta Method
1	0	0	0	0
2	0.0131	0.1185	0.0165	0.0304
3	0.0065	0.1846	0.0107	0.0516
4	0.0082	0.2173	0.0041	0.0663
5	0.0248	0.2292	0.0214	0.0766
6	0.0405	0.2287	0.0377	0.0837
7	0.054	0.2211	0.0519	0.0887
8	0.065	0.21	0.0635	0.0921
9	0.0738	0.1975	0.0727	0.0945
10	0.0806	0.1849	0.0798	0.0962
11	0.0857	0.173	0.0851	0.0973
12	0.0896	0.1621	0.0892	0.0982
13	0.0924	0.1524	0.0922	0.0987
14	0.0945	0.144	0.0943	0.0991
15	0.0961	0.1367	0.0959	0.0994
16	0.0972	0.1305	0.0971	0.0996
17	0.098	0.1253	0.0979	0.0997
18	0.0986	0.1209	0.0985	0.0998
19	0.099	0.1172	0.099	0.0999
20	0.0993	0.1142	0.0993	0.0999
21	0.0995	0.1117	0.0995	0.0999
22	0.0996	0.1096	0.0996	0.1
23	0.0997	0.1078	0.0997	0.1
24	0.0998	0.1064	0.0998	0.1
25	0.0999	0.1052	0.0999	0.1

Error rate should have been less in Runge-Kutta method but in this Project it went more than other methods. It is probably because of an error in my code but I could not find it exactly. Additionally, the methods is a bit failed as seen in Figure 11 and Figure 12. My aim was to find more accurate results by using numerical methods. However, this much difference might also be because of my code. The numerical methods are usually help us to converge the real values but here I am not sure If I did something wrong while finding real current value or while using the numerical methods on MATLAB. I would expect to find the results of Runge-Kutta method's more accurate than others and I would expect to find the Euler's method less accurate but I found it vice versa it is probably because of a bug in my code.

All codes that can be seen in APPENDIX part are written by me.

APPENDIX

```
%% Alican Bayındır 200102002087
% MATH 214 - Project 4
% 23.12.2020
close all; clear all; clc;


STEP_SIZE_1 = 0.05;
STEP_SIZE_2 = 0.025;
L = 0.98; R = 14.2; Vs = 12;
INITIAL_CURRENT = 0.1;


% To draw plot we need to specify the time values
time_half = 0:STEP_SIZE_1:0.6;
time_quarter = 0:STEP_SIZE_2:0.6;


% Some initial values that should be specified before starting methods
current_euler_half(1) = INITIAL_CURRENT;
current_euler_quarter(1) = INITIAL_CURRENT;


current_modieuler_half(1) = INITIAL_CURRENT;
current_modieuler_quarter(1) = INITIAL_CURRENT;


current_midpoint_half(1) = INITIAL_CURRENT;
current_midpoint_quarter(1) = INITIAL_CURRENT;


current_runge_kutta_half(1) = INITIAL_CURRENT;
current_runge_kutta_quarter(1) = INITIAL_CURRENT;


current_analytical_half(1) = INITIAL_CURRENT;
current_analytical_quarter(1) = INITIAL_CURRENT;


F = @(y) (Vs - R * y) / L;
current_equation = @(t) ((Vs*(1-exp((-R*t)/L))) / R) + INITIAL_CURRENT;


% Euler's method
% Step size = 0.05;
for k = 1:length(time_half)-1
current_euler_half(k+1) = current_euler_half(k) + STEP_SIZE_1 * F(current_euler_half(k));
end


% Step size = 0.025;
for k = 1:length(time_quarter)-1
current_euler_quarter(k+1) = current_euler_quarter(k) + STEP_SIZE_2 *
F(current_euler_quarter(k));
end


% Modified Euler's Method
for k = 1:length(time_half)-1
current_modieuler_half(k+1) = current_modieuler_half(k) + (STEP_SIZE_1 / 2) *
(F(current_modieuler_half(k)) + STEP_SIZE_1 * F(current_modieuler_half(k)));
```

```
end


% Modified Euler's Method
for k = 1:length(time_quarter)-1
current_modieuler_quarter(k+1) = current_modieuler_quarter(k) + (STEP_SIZE_2 / 2) *
(F(current_modieuler_quarter(k)) + STEP_SIZE_2 * F(current_modieuler_quarter(k)));
end


% Midpoint Method
for k = 1:length(time_half)-1
current_midpoint_half(k+1)= current_midpoint_half(k) + STEP_SIZE_1 *
(F(current_midpoint_half(k)) + (STEP_SIZE_1 / 2) * F(current_midpoint_half(k)));
end


% Midpoint Method
for k = 1:length(time_quarter)-1
current_midpoint_quarter(k+1)= current_midpoint_quarter(k) + STEP_SIZE_2 *
(F(current_midpoint_quarter(k)) + (STEP_SIZE_2 / 2) * F(current_midpoint_quarter(k)));
end


% Runge Kutta Method Fourth Order
% Step size = 0.05;
for k = 1:length(time_half)-1
    runge_kutta_half1 = STEP_SIZE_1 * F(current_runge_kutta_half(k));
    runge_kutta_half2 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half1 /
2);
    runge_kutta_half3 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half2 /
2);
    runge_kutta_half4 = STEP_SIZE_1 * F(current_runge_kutta_half(k) + runge_kutta_half3);
    current_runge_kutta_half(k + 1) = current_runge_kutta_half(k) + (runge_kutta_half1 + 2
* runge_kutta_half2 + 2 * runge_kutta_half3 + runge_kutta_half4) / 6;
end


% Step size = 0.025
for k = 1:length(time_quarter)-1
    runge_kutta_quarter1 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k));
    runge_kutta_quarter2 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) +
runge_kutta_quarter1 / 2);
    runge_kutta_quarter3 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) +
runge_kutta_quarter2 / 2);
    runge_kutta_quarter4 = STEP_SIZE_2 * F(current_runge_kutta_quarter(k) +
runge_kutta_quarter3);
    current_runge_kutta_quarter(k + 1) = current_runge_kutta_quarter(k) +
(runge_kutta_quarter1 + 2 * runge_kutta_quarter2 + 2 * runge_kutta_quarter3 +
runge_kutta_quarter4) / 6;
end


% Error Analysis
for i = 1:length(time_half)-1
    current_analytical_half(i+1) = current_equation(STEP_SIZE_1*i);
end

for i = 1:length(time_quarter)-1
    current_analytical_quarter(i+1) = current_equation(STEP_SIZE_2*i);
end
```

```
% Plots of the methods
figure(1);
plot(time_half,current_analytical_half, '-*', time_half, current_euler_half, '-*',
time_half, current_modieuler_half, '-*', time_half, current_midpoint_half, '-*', time_half,
current_runge_kutta_half, '-*', 'LineWidth', 2);
xlabel('Time'); ylabel('Current values'); grid on;
title('The graph of current values of each method when delta t = 0.05');
legend('Real','Euler', 'Modified', 'Midpoint', 'Runge Kutta', 'Location', 'southeast');


figure(2);
plot(time_quarter, current_analytical_quarter, '-*', time_quarter, current_euler_quarter,
'-*', time_quarter, current_modieuler_quarter, '-*', time_quarter,
current_midpoint_quarter, '-*', time_quarter, current_runge_kutta_quarter, '-*',
'LineWidth', 2);
xlabel('Time'); ylabel('Current values'); grid on;
title('The graph of current values of each method when delta t = 0.025');
legend('Real','Euler', 'Modified', 'Midpoint', 'Runge Kutta', 'Location', 'southeast');


figure(3);
subplot(2,2,1);
plot(time_half,current_analytical_half, '-*', time_half, current_euler_half, '-*',
time_quarter, current_euler_quarter, '-*');
grid on;
xlabel('Time'); ylabel('Results of Euler Method'); grid on;
title('Euler Method Graph');
legend('Analytical values', 'Delta T = 0.05s', 'Delta T = 0.025s', 'Location',
'southeast');


subplot(2,2,2);
plot(time_half,current_analytical_half, '-*', time_half, current_modieuler_half, '-*',
time_quarter, current_modieuler_quarter, '-*');
grid on;
xlabel('Time'); ylabel('Results of Modified Euler Method'); grid on;
title('Modified Euler Method Graph');
legend('Analytical values', 'Delta T = 0.05s', 'Delta T = 0.025s', 'Location',
'southeast');


subplot(2,2,3);
plot(time_half,current_analytical_half, '-*', time_half, current_midpoint_half, '-*',
time_quarter, current_midpoint_quarter, '-*');
grid on;
xlabel('Time'); ylabel('Results of Midpoint Method'); grid on;
title('Midpoint Method Graph');
legend('Analytical values', 'Delta T = 0.05s', 'Delta T = 0.025s', 'Location',
'southeast');


subplot(2,2,4);
plot(time_half,current_analytical_half, '-*', time_half, current_runge_kutta_half, '-*',
time_quarter, current_runge_kutta_quarter, '-*');
xlabel('Time'); ylabel('Results of Runge-Kutta Method (4th order)'); grid on;
title('Runge-Kutta Method Graph');
legend('Analytical values', 'Delta T = 0.05s', 'Delta T = 0.025s', 'Location',
'southeast');


error_euler_half = abs(current_analytical_half - current_euler_half);
error_modieuler_half=abs(current_analytical_half - current_modieuler_half);
error_midpoint_half=abs(current_analytical_half - current_midpoint_half);
```

```
error_rungekutta_half=abs(current_analytical_half - current_runge_kutta_half);


error_euler_quarter=abs(current_analytical_quarter - current_euler_quarter);
error_modieuler_quarter=abs(current_analytical_quarter - current_modieuler_quarter);
error_midpoint_quarter=abs(current_analytical_quarter- current_midpoint_quarter);
error_rungekutta_quarter=abs(current_analytical_quarter - current_runge_kutta_quarter);


figure(4);
subplot(1,2,1);
plot(time_half, error_euler_half, '-*', time_half, error_modieuler_half, '-*', time_half,
error_midpoint_half, '-*', time_half, error_rungekutta_half, '-*');
xlabel('Error rate'); ylabel('Time'); grid on;
title('Error rates of all methods when Delta t = 0.05s');
legend('Euler Error', 'Modified Euler Error', 'Midpoint Error', 'Runge-Kutta Error',
'Location', 'southeast');


subplot(1,2,2);
plot(time_quarter, error_euler_quarter, '-*', time_quarter, error_modieuler_quarter, '-*',
time_quarter, error_midpoint_quarter, '-*', time_quarter, error_rungekutta_quarter, '-*');
xlabel('Error rate'); ylabel('Time'); grid on;
title('Error rates of all methods when Delta t = 0.025s');
legend('Euler Error', 'Modified Euler Error', 'Midpoint Error', 'Runge-Kutta Error',
'Location', 'southeast');
```