ISTANBUL TECHNICAL UNIVERSITY

UCK 348E

Computer Applic. in Engineering

CRN : 15016

HomeWork 1

NAME : NESLİHAN GÜLSOY

STUDENT ID : 110160111

DEPARTMENT : AEROSPACE ENGINEERING

INSTRUCTOR : Bülent Tutkun

Deadline : 13.11.2021

Contents

1 Analysis of Ordinary Differential Equation

1.1 Givens :

$$\frac{dy}{dx} = 1 - x + 4y \qquad\qquad y(0) = 1 \tag{1}$$

Ordinary Differential Equation (ODE) is given above.

initial value of y is $y(0) = 1$.

Exact solution of ODE gives

$$y(x) = \frac{-3}{16} + \frac{1}{4}x + \frac{19}{16}e^{4x} \tag{2}$$

interval of x $0 \leq x \leq 1$

step sizes are $h = 0.1$ and $h = 0.05$.

Numerical computations of the above ODE will made by Modified-Euler, Fourth-Order Runge-Kutta and Third-Order Adams-Bashforth Method.

1.2 Steps :

Firstly x values are calculated with

$$x_{i+1} = h + x_i \tag{3}$$

Then true values of y are calculated with Eq. (2). After that proper equations are used for approximate values of y. Percent true error and percent relative error are calculated for each method. Results are given as table format at Section (2) and implemented code and graphics are given at Section (3.1).

1.2.1 Modified-Euler Method

Using Euler Method for first approximation of y_{i+1}

$$\begin{aligned} y^e_{i+1} &= y_i + f(x_i, y_i)h \\ y^m_{i+1} &= y_i + \frac{h}{2}f(x_i, y_i)f(x_{i+1}, y^e_{i+1}) \end{aligned} \tag{4}$$

1.2.2 4th Order Runge-Kutta Method

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h/2, y_i + k_1.h/2) \\ k_3 &= f(x_i + h/2, y_i + k_2.h/2) \\ k_4 &= f(x_i + h, y_i + k_3.h) \\ y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{5}$$

1.2.3 Third-Order Adams-Bashforth Method

Using 4th Order Runge-Kutta Method to obtain three starting values of y, (y_i, y_{i-1} and y_{i-2})

$$y_{i+1} = y_i + \frac{h}{12}[23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2})] \tag{6}$$

2 Results and Comments :

2.1 Step size $h = 0.1$:

Step	x	y_{True}	y_{ME}	y_{FoRK}	y_{ToAB}
0	0.0	1.000000	1.000000	1.000000	1.000000
1	0.1	1.609042	1.595000	1.608933	1.608933
2	0.2	2.505330	2.463600	2.505006	2.505006
3	0.3	3.830139	3.737128	3.829415	3.809080
4	0.4	5.794226	5.609949	5.792785	5.726527
5	0.5	8.712004	8.369725	8.709318	8.557855
6	0.6	13.052522	12.442193	13.047713	12.744577
7	0.7	19.515518	18.457446	19.507148	18.940651
8	0.8	29.144880	27.348020	29.130609	28.116018
9	0.9	43.497903	40.494070	43.473954	41.709047
10	1.0	64.897803	59.938223	64.858107	61.852549

Table 1: Results for h=0.1

	Modified-Euler		4th order Runge-Kutta		3rd order Adams-Bashforth	
Step	% | ε_{rel} |	% | ε_{true} |	% | ε_{rel} |	% | ε_{true} |	% | ε_{rel} |	% | ε_{true} |
0						
1	37.3041	0.8727	37.8470	0.0067	37.8470	0.0067
2	35.2573	1.6656	35.7713	0.0129	35.7713	0.0129
3	34.0777	2.4284	34.5851	0.0189	34.2359	0.5498
4	33.3839	3.1803	33.8934	0.0249	33.4836	1.1684
5	32.9733	3.9288	33.4875	0.0308	33.0846	1.7694
6	32.7311	4.6759	33.2502	0.0368	32.8510	2.3593
7	32.5898	5.4217	33.1132	0.0429	32.7131	2.9457
8	32.5090	6.1653	33.0356	0.0490	32.6339	3.5302
9	32.4641	6.9057	32.9930	0.0551	32.5901	4.1125
10	32.4403	7.6421	32.9707	0.0612	32.5670	4.6924

Table 2: Error-table for h=0.1

When Table (2), Table (4), Figure (1) and Figure (2) are examined; by looking absolute percent relative errors (%ε_{rel}), it can be seen that each methods converge to the true solution with increasing step number. According to absolute percent true error (%ε_{true}) values, the best results are obtain by 4th-Order Runge-Kutta Method - %0.0067 and %0.00025 for first estimations and %0.0612 and %0.0045 for last estimation- and using Modified-Euler Method gives the lowest accuracy -%0.87268 and %0.13061 for first estimations and %7.6421 and %2.2699 for last estimation. Even the 4th Order Runge-Kutta Method requires more computational effort, it can be used for solution of given ODE.

Accuracy improves with decreasing the step size for each methods. When %ε_{true} for each step size are compared at first and last estimations; percent error decreases by %85.0335 for ME, %96.1915 for 4th RK and %91.3060 for 3rd AB at first estimation and %70.2974 for ME, %92.6236 for 4th RK and %81.6301 for 3rd AB at last estimation. Therefore, for 4th OrderRunge-Kutta solution decreasing the step size decreases error at a faster rate than for other methods.

To obtain more improved accuracy, iterations can be made for each new estimation for Modified-Euler method and Adams-Moulton corrector formula can be applied for 3rd Order Adams-Bashforth method.

2.2 Step size $h = 0.05$:

Step	x	y_{True}	y_{ME}	y_{FoRK}	y_{ToAB}
0	0.00	1.000000	1.000000	1.000000	1.000000
1	0.05	1.275416	1.273750	1.275413	1.275413
2	0.10	1.609042	1.604975	1.609034	1.609034
3	0.15	2.013766	2.006320	2.013751	2.012803
4	0.20	2.505330	2.493210	2.505306	2.502837
5	0.25	3.102960	3.084466	3.102923	3.098346
6	0.30	3.830139	3.803048	3.830085	3.822606
7	0.35	4.715550	4.676969	4.715474	4.704032
8	0.40	5.794226	5.740402	5.794120	5.777329
9	0.45	7.108956	7.035041	7.108810	7.084863
10	0.50	8.712004	8.611750	8.711806	8.678359
11	0.55	10.667204	10.532585	10.666937	10.620961
12	0.60	13.052522	12.873253	13.052167	12.989755
13	0.65	15.963189	15.726119	15.962720	15.878852
14	0.70	19.515518	19.203865	19.514901	19.403140
15	0.75	23.851575	23.443966	23.850767	23.702880
16	0.80	29.144880	28.614138	29.143827	28.949301
17	0.85	35.607369	34.918999	35.606003	35.351443
18	0.90	43.497903	42.608178	43.496137	43.164506
19	0.95	53.132657	51.986228	53.130379	52.700040
20	1.00	64.897803	63.424698	64.894875	64.338391

Table 3: Results for h=0.05

	Modified-Euler		4th order Runge-Kutta		3rd order Adams-Bashforth	
Step	% | ε_{rel} |	% | ε_{true} |	% | ε_{rel} |	% | ε_{true} |	% | ε_{rel} |	% | ε_{true} |
0						
1	21.4917	0.1306	21.5940	0.0003	21.5940	0.0003
2	20.6374	0.2527	20.7343	0.0005	20.7343	0.0005
3	20.0040	0.3698	20.0977	0.0007	20.0601	0.0478
4	19.5287	0.4838	19.6205	0.0010	19.5791	0.0995
5	19.1688	0.5960	19.2598	0.0012	19.2202	0.1487
6	18.8949	0.7073	18.9855	0.0014	18.9468	0.1967
7	18.6856	0.8182	18.7762	0.0016	18.7377	0.2442
8	18.5254	0.9289	18.6162	0.0018	18.5777	0.2916
9	18.4027	1.0398	18.4938	0.0021	18.4553	0.3389
10	18.3088	1.1508	18.4003	0.0023	18.3617	0.3862
11	18.2371	1.2620	18.3289	0.0025	18.2903	0.4335
12	18.1824	1.3734	18.2746	0.0027	18.2359	0.4809
13	18.1409	1.4851	18.2334	0.0029	18.1946	0.5283
14	18.1096	1.5969	18.2024	0.0032	18.1635	0.5758
15	18.0861	1.7089	18.1791	0.0034	18.1402	0.6234
16	18.0686	1.8210	18.1619	0.0036	18.1228	0.6711
17	18.0557	1.9332	18.1491	0.0038	18.1100	0.7187
18	18.0463	2.0454	18.1399	0.0041	18.1007	0.7665
19	18.0395	2.1577	18.1332	0.0043	18.0940	0.8142
20	18.0347	2.2699	18.1285	0.0045	18.0893	0.8620

Table 4: Error-table for h=0.05

3 APPENDIX :

3.1 Implemented MatLab Code :

```
%%%%%% HOMEWORK 1 %%%%%%
clear; clc; close all;

%%Givens

func_ODE = @(x,y) 1 - x + 4 * y;    %ODE

y(1) = 1;    %Initial Value of y

%Interval of x
x(1) = 0;    %Initial Value for x
x_end = 1;   %Stop Value for x

func_true_y = @(x) -3/16 + x/4 + 19/16 * exp(4.*x); %True Func. of y

%% Step Size = 0.1
h = .1; %Step Size = 0.1

%Create x Points with Given Step Sie
for i = 1 : 1/h
    x(i+1) = h + x(i);
end

%True y Values
true_y = func_true_y(x);    %To Calculate True y Values Call func_true_y

%To Make Numerical Computation Call Defined Functions
[y_ME,rel_err_ME,true_err_ME] = ...
    ME(func_ODE,x,y,x_end,h,true_y);

[y_FORK,rel_err_FORK,true_err_FORK] = ...
    FORK(func_ODE,x,y,x_end,h,true_y);

[y_TOAB,rel_err_TOAB,true_err_TOAB] = ...
    TOAB(func_ODE,x,y,x_end,h,true_y);

% To Create Table for h = 0.1
Output.Step = (0:length(x)-1)';
Output.x = x';
Output.y = true_y';
Output.y_ME = y_ME';
Output.y_FORK =  y_FORK';
Output.y_TOAB = y_TOAB';

Output.rel_e_ME = rel_err_ME';
Output.rel_e_FORK = rel_err_FORK';
Output.rel_e_TOAB = rel_err_TOAB';
```

```
Output.true_e_ME = true_err_ME';
Output.true_e_FORK = true_err_FORK';
Output.true_e_TOAB = true_err_TOAB';

disp('   <strong> h = 0.1</strong>');
T = struct2table( Output );
disp(T) %Display Outputs

%To Create an Excel File to Form Proper Table
writetable(T,'Table.xls','Sheet',"h = 0.1")

%%% Plotting %%%%
%Function Plot
% Create figure
figure1 = figure('WindowState','maximized');

% Create axes
axes1 = axes('Parent',figure1);
hold(axes1,'on');

% Create fplot
fplot(func_true_y,[0 1],'DisplayName','True Function','Parent',axes1,...
    'MarkerSize',6);

% Create multiple lines using matrix input to plot
plot1 = plot(x,[y_ME; y_FORK; y_TOAB],'MarkerSize',24,'Marker','.',...
    'LineStyle','--',...
    'Parent',axes1);
set(plot1(1),'DisplayName','Modified-Euler');
set(plot1(2),'DisplayName','4th-Order RK');
set(plot1(3),'DisplayName','3rd-Order AB');

% Create ylabel
ylabel({'Y points'},'Interpreter','latex');

% Create xlabel
xlabel('X points','Interpreter','latex');

box(axes1,'on');
axis(axes1,'tight');
hold(axes1,'off');
% Set the remaining axes properties
set(axes1,'XGrid','on');
% Create legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.146055 0.75134 0.1219 0.1394],'LineWidth',0.8,...
    'Interpreter','latex',...
    'FontSize',12);
title(legend1,'h = 0.1');

saveas(gcf,'func1.png')
saveas(gcf,'func1.fig')
```

```
%ERROR PLOT

% Create figure
figure3 = figure('WindowState','maximized');

% Create axes
axes1 = axes('Parent',figure3);
hold(axes1,'on');

% Create multiple lines using matrix input to plot
plot1 = plot(x,[true_err_ME; true_err_FORK; true_err_TOAB],...
    'MarkerSize',24,'Marker','.',...
    'LineStyle','--',...
    'Parent',axes1);
set(plot1(1),'DisplayName','Modified-Euler');
set(plot1(2),'DisplayName','4th-Order RK');
set(plot1(3),'DisplayName','3rd-Order AB');

% Create ylabel
ylabel({'$\%\mid\varepsilon_{true}\mid$'},'Fontsize',15,...
    'Interpreter','latex');

% Create xlabel
xlabel('X points','Interpreter','latex');

box(axes1,'on');
axis(axes1,'tight');
hold(axes1,'off');
% Set the remaining axes properties
set(axes1,'XGrid','on');
% Create legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.146055 0.75134 0.1219 0.1394],...
    'LineWidth',0.8,...
    'Interpreter','latex',...
    'FontSize',12);
title(legend1,'h = 0.1');

saveas(gcf,'error1.png')
saveas(gcf,'error1.fig')

close all;
%% Step Size = 0.05
h = .05; %Step Size = 0.05

%Create x Points
for i = 1 : 1/h
    x(i+1) = h + x(i);
end

%True y Values
true_y = func_true_y(x);

[y_ME,rel_err_ME,true_err_ME,slope_ME] = ...
```

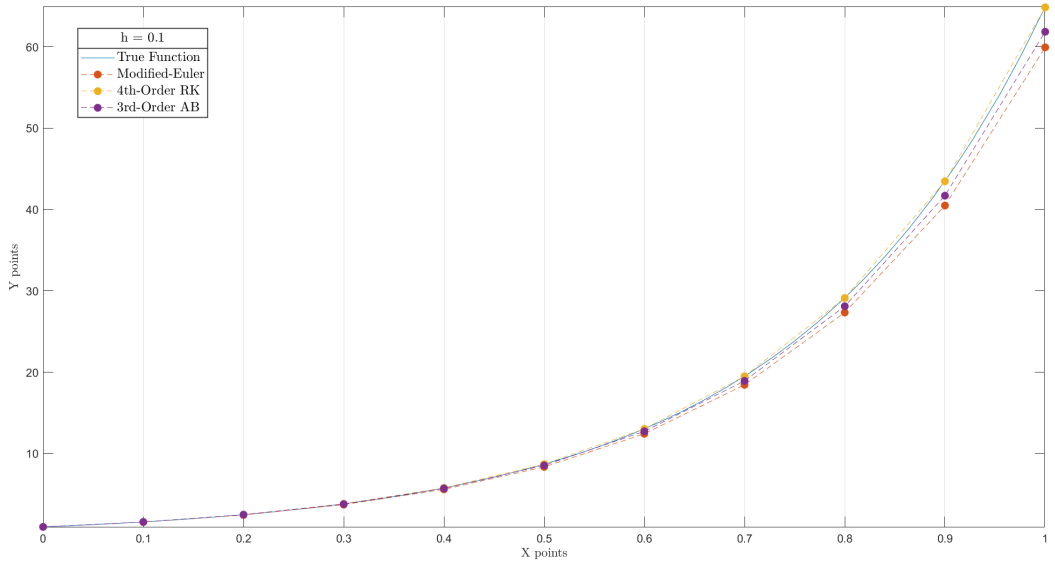
```
    ME(func_ODE,x,y,x_end,h,true_y);

[y_FORK,rel_err_FORK,true_err_FORK,k1,k2,k3,k4] = ...
    FORK(func_ODE,x,y,x_end,h,true_y);

[y_TOAB,rel_err_TOAB,true_err_TOAB] = ...
    TOAB(func_ODE,x,y,x_end,h,true_y);

%To Create Table for h = 0.05
Output.Step = (0:length(x)-1)';
Output.x = x';
Output.y = true_y';
Output.y_ME = y_ME';
Output.y_FORK =  y_FORK';
Output.y_TOAB = y_TOAB';

Output.rel_e_ME = rel_err_ME';
Output.rel_e_FORK = rel_err_FORK';
Output.rel_e_TOAB = rel_err_TOAB';

Output.true_e_ME = true_err_ME';
Output.true_e_FORK = true_err_FORK';
Output.true_e_TOAB = true_err_TOAB';

disp('   <strong> h = 0.05</strong>');
T = struct2table( Output );
disp(T) %Display Outputs

writetable(T,'Table.xls','Sheet',"h = 0.05")

%%%% PLOTTING
figure2 = figure('WindowState','maximized');

% Create axes
axes2 = axes('Parent',figure2);
hold(axes2,'on');

% Create fplot
fplot(func_true_y,[0 1],'DisplayName','True Function','Parent',axes2,...
    'MarkerSize',6);
% Create multiple lines using matrix input to plot
plot1 = plot(x,[y_ME; y_FORK; y_TOAB],'MarkerSize',24,'Marker','.',...
    'LineStyle','--',...
    'Parent',axes2);
set(plot1(1),'DisplayName','Modified-Euler');
set(plot1(2),'DisplayName','4th-Order RK');
set(plot1(3),'DisplayName','3rd-Order AB');

% Create ylabel
ylabel({'Y points'},'Interpreter','latex');

% Create xlabel
xlabel('X points','Interpreter','latex');

box(axes2,'on');
```

```
axis(axes2,'tight');
hold(axes2,'off');
% Set the remaining axes properties
set(axes2,'XGrid','on','XTick',(0:.05:1));
% Create legend
legend1 = legend(axes2,'show');
set(legend1,...
    'Position',[0.146055 0.75134 0.1219 0.1394],...
    'LineWidth',0.8,...
    'Interpreter','latex',...
    'FontSize',12);
title(legend1,'h = 0.05');

saveas(gcf,'func2.png')
saveas(gcf,'func2.fig')


%ERROR PLOT

figure4 = figure('WindowState','maximized');

% Create axes
axes2 = axes('Parent',figure4);
hold(axes2,'on');

% Create multiple lines using matrix input to plot
plot1 = plot(x,[true_err_ME; true_err_FORK; true_err_TOAB],...
    'MarkerSize',24,'Marker','.',...
    'LineStyle','--',...
    'Parent',axes2);
set(plot1(1),'DisplayName','Modified-Euler');
set(plot1(2),'DisplayName','4th-Order RK');
set(plot1(3),'DisplayName','3rd-Order AB');

% Create ylabel
ylabel({'$\%\mid\varepsilon_{true}\mid$'},'Fontsize',15,...
    'Interpreter','latex');
% Create xlabel
xlabel('X points','Interpreter','latex');

box(axes2,'on');
axis(axes2,'tight');
hold(axes2,'off');
% Set the remaining axes properties
set(axes2,'XGrid','on','XTick',(0:.05:1));
% Create legend
legend1 = legend(axes2,'show');
set(legend1,...
    'Position',[0.146055 0.75134 0.1219 0.1394],...
    'LineWidth',0.8,...
    'Interpreter','latex',...
    'FontSize',12);
title(legend1,'h = 0.05');

saveas(gcf,'error2.png')
```

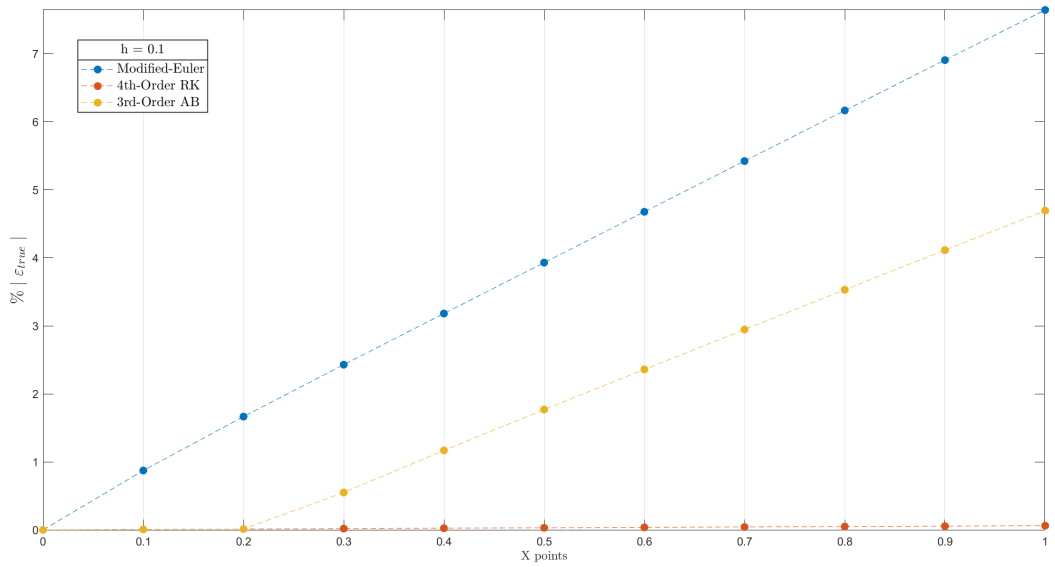
```
saveas(gcf,'error2.fig')

close all;

%% Functions for Methods

%For Modified-Euler Method
function [y,rel_err,true_err,slope] = ME(func,x,y,x_end,h,true_y)
%%Formulation for Moduler Euler Method
for i = 1 : x_end/h
    f(i) = func(x(i),y(i));
    y_e(i+1) = y(i) + h * f(i);   %First Appx. y_(i+1) with Euler Method
    %Corrector Equation
    y(i+1) = y(i) + h/2 * (f(i) + func(x(i+1),y_e(i+1)));
    %Relative Percent Error
    rel_err(i+1) = abs( (y(i+1)-y(i)) / (y(i+1)) ) *100;
end

%%True Percent Error Calculation
true_err = abs((true_y - y ) ./ (true_y)) * 100;
slope = func(x,y);
end

%For 4th Order Runge-Kutta Method
function [y,rel_err,true_err,k1,k2,k3,k4] = FORK(func,x,y,x_end,h,true_y)

for i = 1 : x_end/h
    k1(i) = func(x(i),y(i));
    k2(i) = func(x(i) + h/2, y(i)+k1(i)*h/2);
    k3(i) = func(x(i) + h/2, y(i)+k2(i)*h/2);
    k4(i) = func(x(i) + h, y(i)+k3(i)*h);
    y(i+1) = y(i) + h/6 * (k1(i) + 2*k2(i) + 2*k3(i) + k4(i));
    %Relative Percent Error
    rel_err(i+1) = abs( (y(i+1)-y(i)) / (y(i+1)) ) *100;
end

%%True Percent Error Calculation
true_err = abs((true_y - y ) ./ (true_y)) * 100;
%slope = func(x,y);

end

%For 3rd Order Adams-Bashforth Method
function [y,rel_err,true_err] = TOAB(func,x,y,x_end,h,true_y)
bo = 23/12; b1 = -16/12; b2 = 5/12; %Coef.
%To Calculate Initial Values with 4th Order RK Method
for i = 1 : 2
    k1(i) = func(x(i),y(i));
    k2(i) = func(x(i) + h/2, y(i)+k1(i)*h/2);
    k3(i) = func(x(i) + h/2, y(i)+k2(i)*h/2);
    k4(i) = func(x(i) + h, y(i)+k3(i)*h);
    y(i+1) = y(i) + h/6 * (k1(i) + 2*k2(i) + 2*k3(i) + k4(i));
end

for i = 3 : x_end/h
```

```
    %Using Predictor Formula
    y(i+1) = y(i) + h * (bo * func(x(i), y(i)) + b1 * func(x(i-1), ...
        y(i-1)) + b2 * func(x(i-2), y(i-2)) );
    %Relative Percent Error
    rel_err(i+1) = abs( (y(i+1)-y(i)) / (y(i+1)) ) *100;
end
%%True Error Calculation
true_err = abs((true_y - y ) ./ (true_y)) * 100;
end
```

3.2 Graphics :

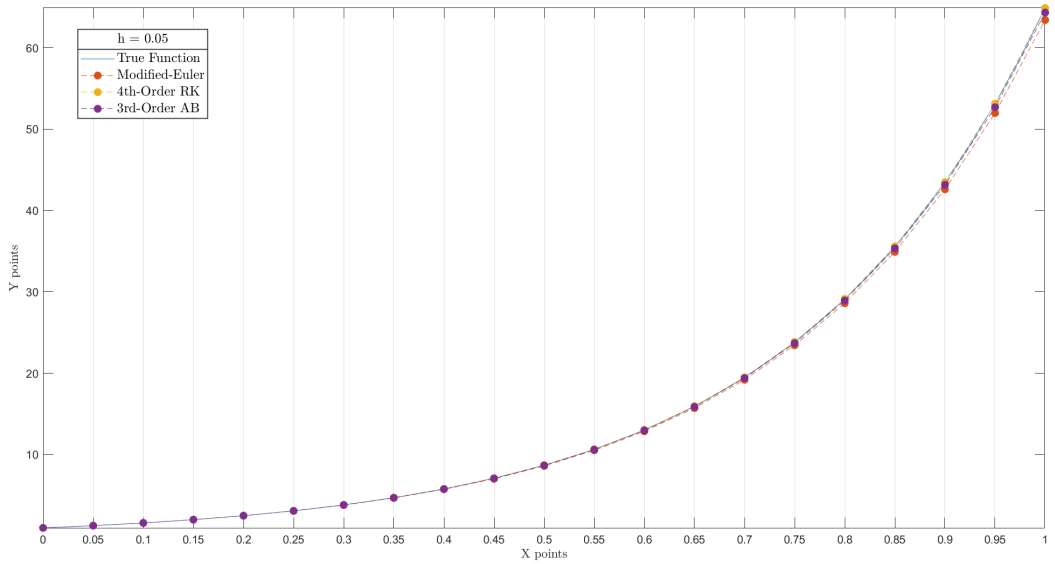


(a) Calculated and True Values of y

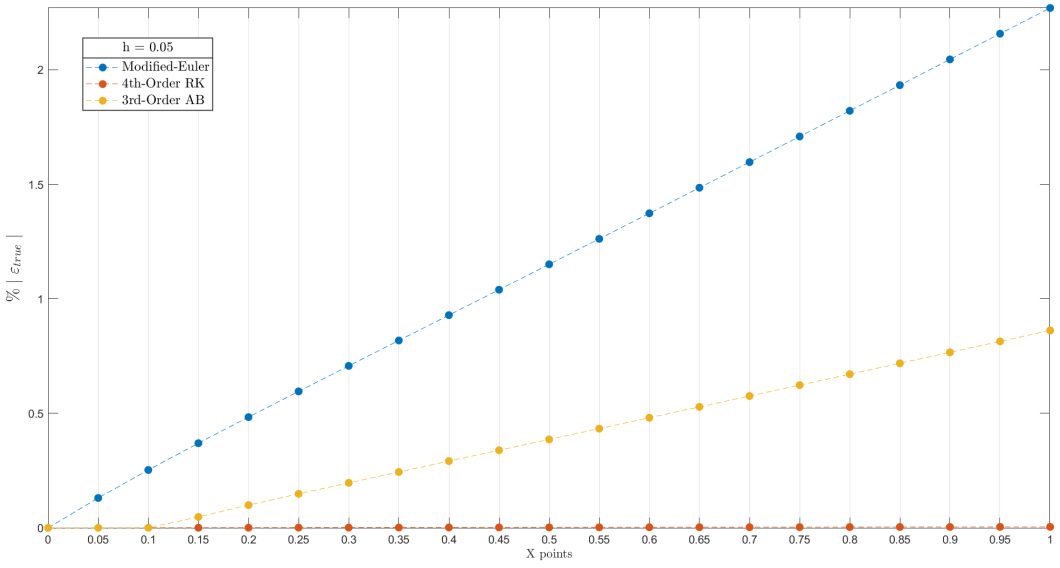


(b) Error Graphic

Figure 1: Graphics for h = 0.1



(a) Calculated and True Values of y



(b) Error Graphic

Figure 2: Graphics for h = 0.05