A Study on Numerical Solution of Initial Value Problem by Using Euler's Method, Runge-Kutta 2nd Order, Runge-Kutta 4th Order, and Runge-Kutta Fehlberg Method with MATLAB

Mst. Sharmin Banu[1], Irfan Raju[2], Ummay Habiba Mabsuma Zaman[3]

Abstract--- In this paper, Euler method, Runge-Kutta second order, Runge-Kutta fourth order, and Runge-Kutta Fehlberg methods are presented to solve the initial value problem of ordinary differential equation. These methods are effective enough to reach accuracy and well established for solving practical problems. To verify the accuracy, in comparison with the exact solution of our computed approximate solution, a standard initial value problem is taken as an example. Also, the accuracy and convergence are compared among these methods with different step sizes by using MATLAB.

Keywords: Euler method, Runge-Kutta second order, Runge-Kutta fourth order, and Runge-Kutta Fehlberg methods, Initial Value Problem, MATLAB

[1] Department of Mathematics, Jashore University of Science and Technology, Jashore, Bangladesh +8801722928744 E-mail: msbsharmin_math@just.edu.bd
[2] Department of Arts and Sciences, Bangladesh Army University of Science and Technology, Saidpur, Bangladesh +8801743810588 E-mail: irfan@baust.edu.bd
[3] Department of Mathematics, Jashore University of Science and Technology, Jashore, Bangladesh +8801720349933 E-mail: mabsumazaman@gmail.com

1. Introduction

Many real life problems related to science and engineering can be modeled using differential equations (DEs). Most of the time model equations that form using DEs get complicated and therefore it becomes very difficult to find its exact solution, though there are several exact methods to solve DEs. In this case, researcher fined their faith in numerical techniques which are able to find approximate solutions to the complicated model equations. One of the oldest numerical methods to solve the initial value differential equation is Euler Method named after Leonhard Euler in 1768. After then several methods have been developed in this regard and still, researchers are trying to develop new methods or trying to improve the accuracy of the existing method to find a better solution. However, Runge-Kutta method is also a popular method to solve ordinary differential equations. There are many Runge-Kutta methods [1]-[10]. In this paper we have studied Euler Method, Runge-Kutta 2nd order method, Runge-Kutta 4th order method, and Runge-Kutta Fehlberg Method to solve selected initial value ordinary differential equation (ODE) with the help of MATLAB software. The rest of the paper is organized as follows: Section 2 deals with the brief discussion of the methodology, in section 3 formulation of the problem and computation are presented, discussion of the result is illustrated in section 4, and the conclusion is presented in section 5.

2. Brief discussion of methodology

In this study we have studied Euler Method, Runge-Kutta 2nd order method, Runge-Kutta 4th order method and Runge-Kutta Fehlberg Method. Details about these methods can be found in any standard numerical book [11]-[16]. A short note about the studied method to achieve the approximated numerical solution of the Initial Value Problem $y' = f(x, y), y(x_0) = y_0$ is given in the following subsections.

2.1 Euler's Method

One of the oldest numerical methods to solve ordinary differential equation is Euler Method. The method was originated by Leonhard Euler in 1768. The general formula of approximation is

$$y_{n+1}(x) = y_n(x) + hf(x_n, y_n), n = 0, 1, 2, 3, \dots\dots\dots$$

2.2 Runge-Kutta 2nd order method

Runge-Kutta 2nd order method is the very begging method of Runge-Kutta family. Only ODE can be solved by using the Runge-Kutta 2nd order method. The general formula of the Runge-Kutta 2nd order method is as follows:

$y_{n+1} = y_n + h\frac{1}{2}(k_1 + k_2)$, for $n = 0, 1, 2, 3, \dots\dots\dots\dots$

where

$k_1 = hf(x_n, y_n)$

$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$.

2.3 Runge-Kutta 4th order method

Runge-Kutta 4th order method is one of the most favorite ones among the Runge-Kutta family. It is a good choice for a common purpose because it is quite accurate, stable, and easy to code. The derivation of Runge-Kutta 4th order method can be found in any standard numerical textbook. Due to the paucity of space, we only write here the general formula of it.

$y_{n+1}(x + h) = y_n(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$,

where $n = 1, 2, 3 \dots\dots$

Also,

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$

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

$$k_4 = hf(x_n + h, y_n + k_1)$$

2.4 Runge-Kutta Fehlberg Method

Runge-Kutta-Fehlberg's method is an embedded method from the Runge-Kutta family. Erwin Fehlberg introduced a six-stage RK method which requires six function evaluations per step. If a proper step size is being used then at each step the method approximate two different solutions and compare it, if the approximate solutions are close enough then the method accept the solution. If it is not so, then a new step size is being chosen for further computation [17]. The general form of RK45 method is as follows:

$y_{n+1} = y_n + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$, where $n = 1, 2, 3 \dots\dots$

Also,

$$k_1 = hf(x_n, y_n),$$

$$k_2 = hf\left(x_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right),$$

$$k_3 = hf\left(x_n + \frac{3}{8}h, y_n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),$$

$$k_4 = hf\left(x_n + \frac{12}{13}h, y_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),$$

$$k_5 = hf\left(x_n + h, y_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),$$

$$k_6 = hf\left(x_n + \frac{1}{2}h, y_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 - \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)$$

3. Problem formulation and computation

In this section, we discus about the operation procedure of the programing language MATLAB with the methods mention earlier in solving the initial valued problem. MATLAB is a high-performance language that uses for technical computing (like applied mathematics such as education, research, and industry) and stands for matrix laboratory that is built up around vector and matrices [18]. The language, tools, and built-in math functions enable you to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java [19]. To find the solutions of ODE we have taken an initial valued problem. The following equation is considered as our model equation:

EQ: We consider the initial value problem

$y'(x) = 0.1y + (1 - 0.1)\cos x - (1 + 0.1)\sin x,$

where $y(0) = 1,\ 0 \leq x \leq 10.$

The exact solution of this problem is given by $y(x) = \sin x + \cos x.$

The results are computed with the initial step sizes h=0.75, 0.6, and 0.3 within the geometrical domain $0 \leq x \leq 10$.

3.1 Solution of the selected problem using Euler's Method

3.1.1 MATLAB implementation procedure for Euler method:

Input:
Starting point a, end point b, initial condition α, step size h
Output:

```
    Step: 1 setting initial condition
            $x = a$
            $y = \alpha$
Step: 2 While ($b > x$) do step 3-4
            Step: 3
            $y = y + h * f(x, y)$
            Step: 4 if $x + h > b$
            Set $h = b - x$
Step: 5 Stop.
```

3.1.2 Relative description:

Table: 3.1.1 to **Table: 3.1.3** contains the approximated solution, exact solution, and the relative absolute errors for the Euler's method with step size h=0.75, 0.6, and 0.3 respectively. And the relative figures are displayed at the **Fig. 3.1.1** to **Fig. 3.1.2** respectively. Where the line curve is the approximated solution and the circle represents the exact solution.

Table 3.1.1: Comparison between Euler approximated and exact solution with step size h=0.75

x	Euler	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.750000	1.750000	1.413328	0.336672
1.500000	1.812788	1.068232	0.744556
2.250000	1.173561	0.149900	1.023662
3.000000	0.195651	-0.848872	1.044523
3.750000	-0.574344	-1.392121	0.817776
4.500000	-0.699760	-1.188326	0.488566
5.250000	-0.088066	-0.346849	0.258783
6.000000	0.959607	0.680755	0.278853
6.750000	1.910211	1.343050	0.567160
7.500000	2.284969	1.284635	1.000334
8.250000	1.916471	0.536856	1.379615
9.000000	1.038678	-0.499012	1.537690
9.750000	0.161568	-1.267099	1.428667
10.000000	-0.202327	-1.355233	1.152905

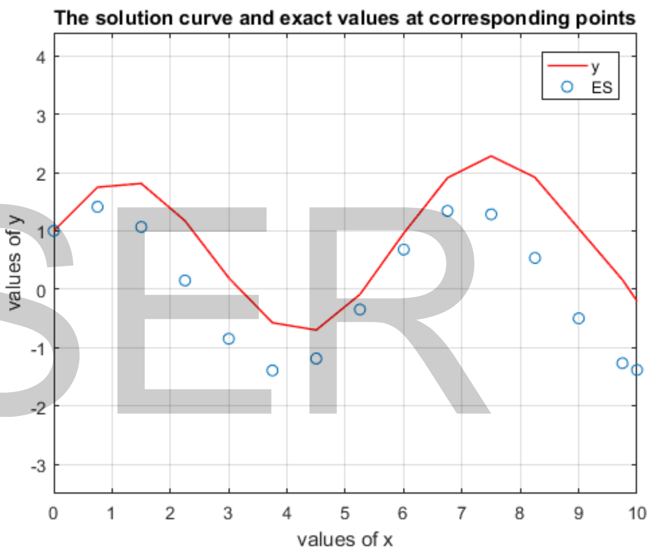

Fig. 3.1.1: Graphical representation of approximated solution with Euler Method (h=0.75)

Table 3.1.2: Comparison between Euler approximated and exact solution with step size h=0.6

x	Euler	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.600000	1.600000	1.389978	0.210022
1.200000	1.769017	1.294397	0.474620
1.800000	1.455686	0.746646	0.709040
2.400000	0.777598	-0.061931	0.839529
3.000000	-0.019744	-0.848872	0.829128
3.600000	-0.648664	-1.339279	0.690615
4.200000	-0.879770	-1.361837	0.482067
4.800000	-0.622057	-0.908666	0.286609
5.400000	0.045338	-0.138072	0.183409
6.000000	0.900817	0.680755	0.220062
6.600000	1.657772	1.261774	0.395998
7.200000	2.064747	1.402019	0.662727
7.800000	1.993320	1.052499	0.940821
8.400000	1.483017	0.335310	1.147706
9.000000	0.727547	-0.499012	1.226558
9.600000	0.007191	-1.159015	1.166205
10.000000	-0.409053	-1.414140	1.005087

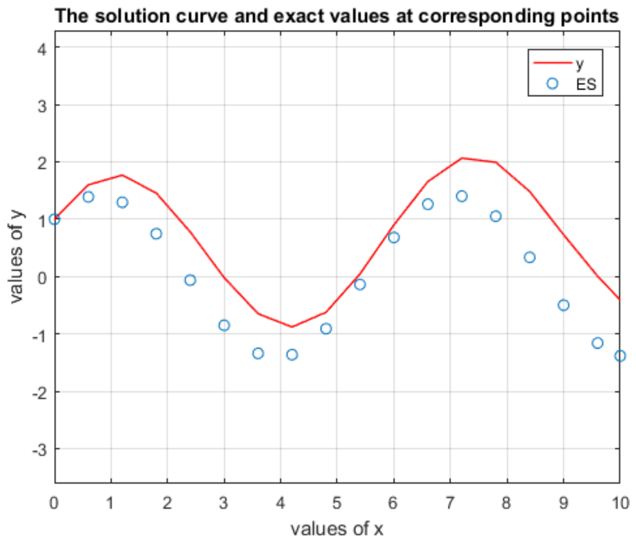

Fig. 3.1.2: Graphical representation of approximated solution with Euler Method (h=0.6)

Table 3.1.3: Comparison between Euler approximated and exact solution with step size h=0.3

x	Euler	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.300000	1.300000	1.250857	0.049143
0.600000	1.499419	1.389978	0.109441
0.900000	1.580910	1.404937	0.175973
1.200000	1.537674	1.294397	0.243278
1.500000	1.374068	1.068232	0.305836
1.800000	1.105216	0.746646	0.358571
2.100000	0.755658	0.358363	0.397295
2.400000	0.357161	-0.061931	0.419091
2.700000	-0.054124	-0.476692	0.422569
3.000000	-0.440882	-0.848872	0.407990
3.300000	-0.767976	-1.145225	0.377249
3.600000	-1.005579	-1.339279	0.333700
3.900000	-1.131840	-1.413698	0.281859
4.200000	-1.134834	-1.361837	0.227003
4.500000	-1.013629	-1.188326	0.174697
4.800000	-0.778368	-0.908666	0.130298
5.100000	-0.449360	-0.547837	0.098477
5.400000	-0.055268	-0.138072	0.082804
5.700000	0.369454	0.284027	0.085426
6.000000	0.787636	0.680755	0.106881
6.300000	1.162718	1.016673	0.146045
6.600000	1.462013	1.261774	0.200239
6.900000	1.659627	1.394165	0.265462
7.200000	1.738777	1.402019	0.336758
7.500000	1.693285	1.284635	0.408649
7.800000	1.528135	1.052499	0.475636
8.100000	1.259027	0.726346	0.532682
8.400000	0.910978	0.335310	0.575667
8.700000	0.516081	-0.085677	0.601759
9.000000	0.110649	-0.499012	0.609661
9.300000	-0.268035	-0.867771	0.599736
9.600000	-0.585047	-1.159015	0.573967
9.900000	-0.810937	-1.346727	0.535790
10.000000	-0.924359	-1.414140	0.489781

3.2 Solution of the selected problem using Runge-Kutta 2nd order method

3.2.1 MATLAB implementation procedure for RK2 method:

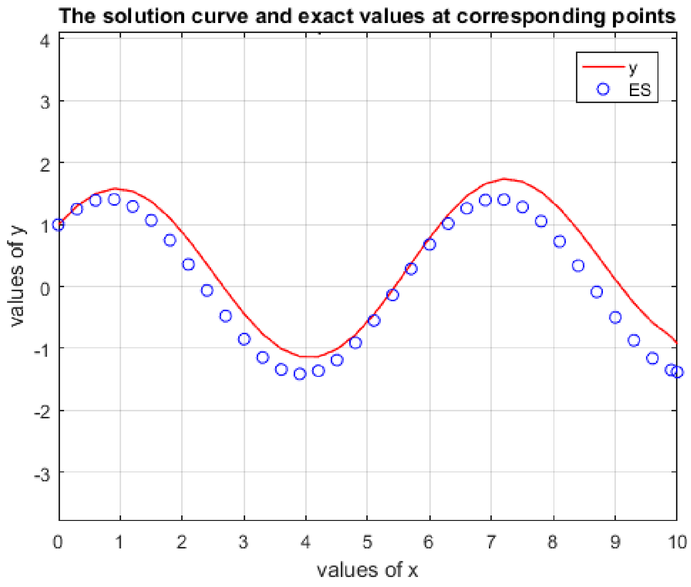

Fig. 3.1.3: Graphical representation of approximated solution with Euler Method (h=0.3)

<u>Input:</u>
Starting point a, end point b, initial condition α, step size h
<u>Output:</u>

```
Step: 1 setting initial condition
    $x = a$
    $y = \alpha$
Step: 2 While $(b > x)$ do step 3-5
    Step: 3
    $k1 = h * f(x, y);$
    $k2 = h * f(x + h, y + k1);$
    Step: 4
    $y = y + (1/2) * (k1 + k2)$
    Step: 5 if $x + h > b$
    Set $h = b - x$
Step: 6 Stop.
```

3.2.2 Relative description:

The approximated solution, exact solution, and the relative absolute errors for the RK2 method with step size h=0.75, 0.6, and 0.3 are displayed from **Table: 3.2.1** to **Table: 3.2.3** respectively. And **Fig. 3.2.1** to **Fig. 3.2.2** represents the corresponding graphical representation relative to those step sizes respectively. Where the line curve is the approximated solution and the circle represents the exact solution as before.

Table 3.2.1: Comparison between RK2 approximated and exact solution with step size h=0.75

x	Exact	RK2	Error
0.000000	1.000000	1.000000	0.000000
0.750000	1.413328	1.406394	0.006934
1.500000	1.068232	1.091438	0.023206
2.250000	0.149900	0.226739	0.076840
3.000000	-0.848872	-0.720888	0.127984
3.750000	-1.392121	-1.239912	0.152209
4.500000	-1.188326	-1.048562	0.139764
5.250000	-0.346849	-0.246020	0.100829
6.000000	0.680755	0.740828	0.060073
6.750000	1.343050	1.386485	0.043435
7.500000	1.284635	1.348863	0.064228
8.250000	0.536856	0.652876	0.116020
9.000000	-0.499012	-0.322899	0.176113
9.750000	-1.267099	-1.049349	0.217750
10.000000	-1.355233	-1.130728	0.224505

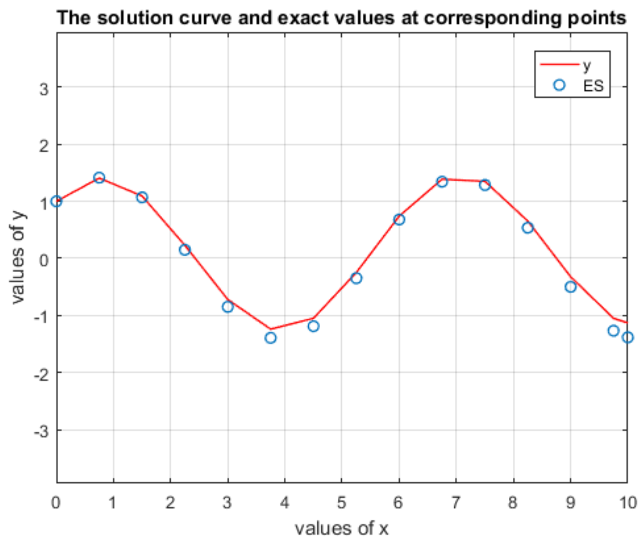

Fig. 3.2.1: Graphical representation of approximated solution with RK2 Method (h=0.75)

Table 3.2.2: Comparison between RK2 approximated and exact solution with step size h=0.6

x	RK2	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.600000	1.384509	1.389978	0.005469
1.200000	1.299034	1.294397	0.004637
1.800000	0.774280	0.746646	0.027634
2.400000	-0.005546	-0.061931	0.056384
3.000000	-0.767076	-0.848872	0.081797
3.600000	-1.243274	-1.339279	0.096005
4.200000	-1.266717	-1.361837	0.095120
4.800000	-0.828076	-0.908666	0.080589
5.400000	-0.079372	-0.138072	0.058700
6.000000	0.719138	0.680755	0.038384
6.600000	1.289876	1.261774	0.028102
7.200000	1.434914	1.402019	0.032895
7.800000	1.105125	1.052499	0.052626
8.400000	0.417348	0.335310	0.082038
9.000000	-0.386423	-0.499012	0.112588
9.600000	-1.023567	-1.159015	0.135448
10.000000	-1.269554	-1.414140	0.144586

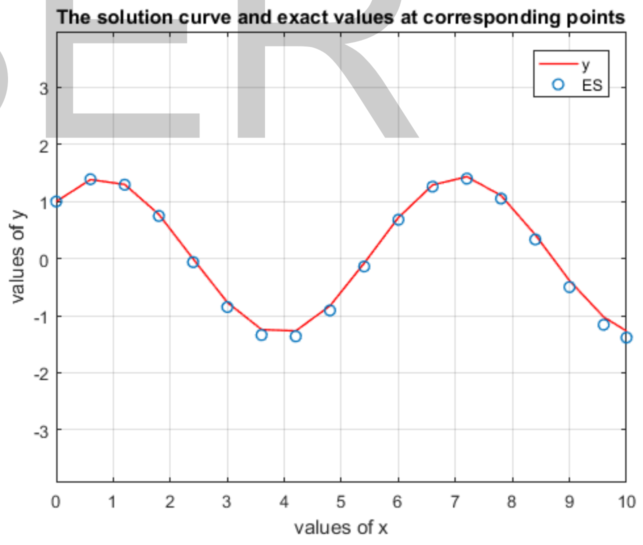

Fig. 3.2.2: Graphical representation of approximated solution with RK2 Method (h=0.6)

Table 3.2.3: Comparison between RK2 approximated and exact solution with step size h=0.3

x	RK2	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.300000	1.249710	1.250857	0.001147
0.600000	1.388633	1.389978	0.001345
0.900000	1.404388	1.404937	0.000549
1.200000	1.295592	1.294397	0.001195
1.500000	1.071991	1.068232	0.003759
1.800000	0.753588	0.746646	0.006942
2.100000	0.368853	0.358363	0.010490
2.400000	-0.047816	-0.061931	0.014114
2.700000	-0.459169	-0.476692	0.017523
3.000000	-0.828429	-0.848872	0.020443
3.300000	-1.122579	-1.145225	0.022647
3.600000	-1.315308	-1.339279	0.023971
3.900000	-1.389366	-1.413698	0.024332
4.200000	-1.338103	-1.361837	0.023734
4.500000	-1.166059	-1.188326	0.022267
4.800000	-0.888566	-0.908666	0.020099
5.100000	-0.530372	-0.547837	0.017465
5.400000	-0.123432	-0.138072	0.014640
5.700000	0.295944	0.284027	0.011917
6.000000	0.690338	0.680755	0.009583
6.300000	1.024564	1.016673	0.007891
6.600000	1.268812	1.261774	0.007038
6.900000	1.401311	1.394165	0.007146
7.200000	1.410274	1.402019	0.008254
7.500000	1.294950	1.284635	0.010314
7.800000	1.065692	1.052499	0.013193
8.100000	0.743032	0.726346	0.016686
8.400000	0.355847	0.335310	0.020537
8.700000	-0.061221	-0.085677	0.024456
9.000000	-0.470858	-0.499012	0.028153
9.300000	-0.836414	-0.867771	0.031357
9.600000	-1.125171	-1.159015	0.033843
9.900000	-1.311274	-1.346727	0.035453
10.000000	-1.378033	-1.414140	0.036108

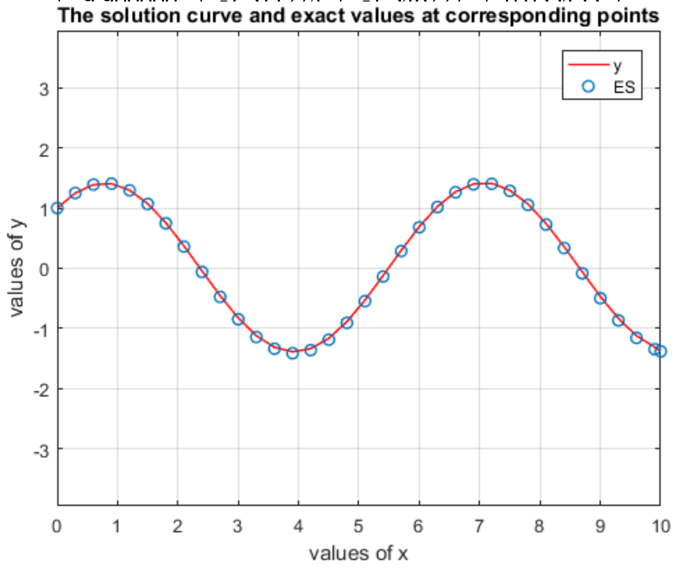

Fig. 3.2.3: Graphical representation of approximated solution with RK2 Method (h=0.3)

3.3 Solution of the selected problem using Runge-Kutta 4th order method

3.3.1 MATLAB implementation procedure for RK4 method:

Input:
Starting point a, end point b, initial condition α, step size h

Output:

```
Step: 1 setting initial condition
        Set $n = (b - a)/h$
        $x = a$
        $y = \alpha$
Step: 2 While $(b > x)$ do step 3-5
        Step: 3
        $k1 = h * f(x(i), y(i));$
        $k2 = h * f(x(i) + h/2, y(i) + k1/2);$
        $k3 = h * f(x(i) + h/2, y(i) + k2/2);$
        $k4 = h * f(x(i) + h, y(i) + k3);$
        Step: 4
        $y = y + (1/6) * (k1 + 2 * k2 + 2 * k3 + k4)$
        Step: 5 if $(x + h > b)$
        Set $h = b - x$
Step: 6 Stop.
```

3.3.2 Relative description:

Here the **Table: 3.3.1** to **Table: 3.3.3** contains the approximation of RK4 method with step size h=0.75, 0.6, and 0.3 respectively, where the approximated solution, exact solution, and the relative absolute errors are presented. And the relative figures are displayed at the **Fig. 3.3.1** to **Fig. 3.3.2** respectively. Again the line curve is the approximated solution and the circle represents the exact solution.

Table 3.3.1: Comparison between RK4 approximated and exact solution with step size h=0.75

x	RK4	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.750000	1.413319	1.413328	0.000009
1.500000	1.068134	1.068232	0.000099
2.250000	0.149671	0.149900	0.000228
3.000000	-0.849207	-0.848872	0.000335
3.750000	-1.392490	-1.392121	0.000369
4.500000	-1.188647	-1.188326	0.000321
5.250000	-0.347075	-0.346849	0.000226
6.000000	0.680611	0.680755	0.000144
6.750000	1.342921	1.343050	0.000129
7.500000	1.284433	1.284635	0.000202
8.250000	0.536522	0.536856	0.000335
9.000000	-0.499481	-0.499012	0.000469
9.750000	-1.267647	-1.267099	0.000548
10.000000	-1.355776	-1.355233	0.000544

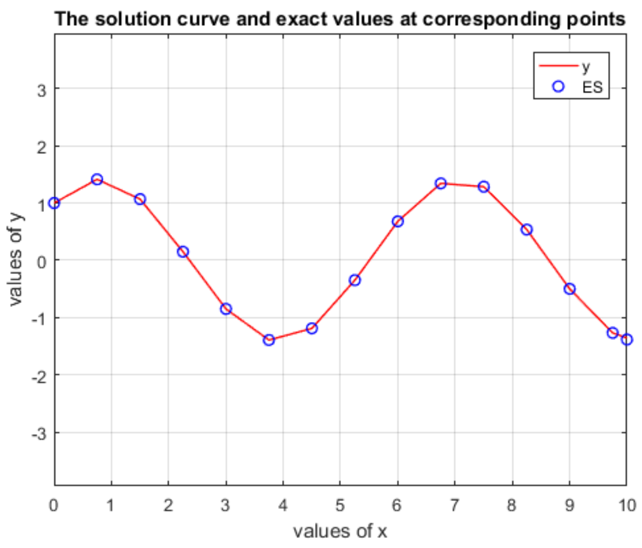

Fig. 3.3.1: Graphical representation of approximated solution with RK4 Method (h=0.75)

Table 3.3.2: Comparison between RK4 approximated and exact solution with step size h=0.6

x	RK4	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.600000	1.389978	1.389978	0.000000
1.200000	1.294375	1.294397	0.000022
1.800000	0.746585	0.746646	0.000061
2.400000	-0.062034	-0.061931	0.000103
3.000000	-0.849009	-0.848872	0.000136
3.600000	-1.339429	-1.339279	0.000150
4.200000	-1.361979	-1.361837	0.000142
4.800000	-0.908782	-0.908666	0.000116
5.400000	-0.138156	-0.138072	0.000084
6.000000	0.680696	0.680755	0.000059
6.600000	1.261723	1.261774	0.000051
7.200000	1.401953	1.402019	0.000066
7.800000	1.052397	1.052499	0.000102
8.400000	0.335162	0.335310	0.000148
9.000000	-0.499203	-0.499012	0.000191
9.600000	-1.159234	-1.159015	0.000219
10.000000	-1.414366	-1.414140	0.000226

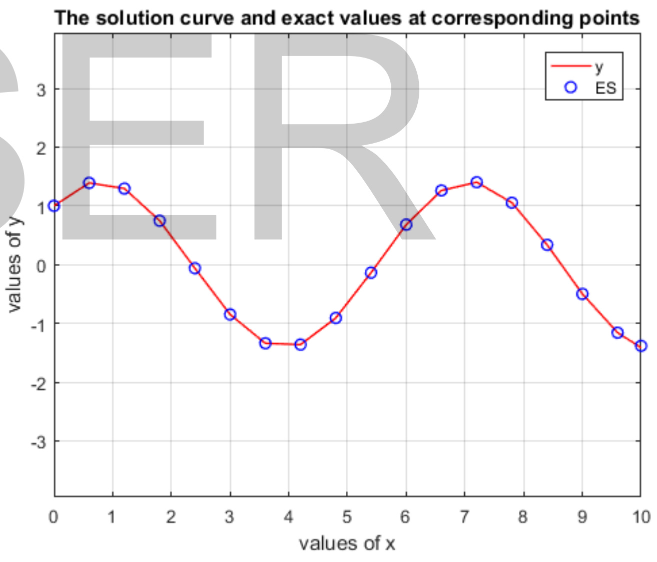

Fig. 3.3.2: Graphical representation of approximated solution with RK4 Method (h=0.6)

Table 3.3.3: Comparison between RK4 approximated and exact solution with step size h=0.3

x	RK4	Exact	Error
0.000000	1.000000	1.000000	0.000000
0.300000	1.250857	1.250857	0.000000
0.600000	1.389978	1.389978	0.000000
0.900000	1.404936	1.404937	0.000001
1.200000	1.294395	1.294397	0.000001
1.500000	1.068230	1.068232	0.000002
1.800000	0.746642	0.746646	0.000004
2.100000	0.358358	0.358363	0.000005
2.400000	-0.061937	-0.061931	0.000006
2.700000	-0.476700	-0.476692	0.000008
3.000000	-0.848881	-0.848872	0.000008
3.300000	-1.145235	-1.145225	0.000009
3.600000	-1.339288	-1.339279	0.000009
3.900000	-1.413708	-1.413698	0.000009
4.200000	-1.361845	-1.361837	0.000009
4.500000	-1.188334	-1.188326	0.000008
4.800000	-0.908673	-0.908666	0.000007
5.100000	-0.547843	-0.547837	0.000006
5.400000	-0.138077	-0.138072	0.000005
5.700000	0.284023	0.284027	0.000004
6.000000	0.680751	0.680755	0.000004
6.300000	1.016669	1.016673	0.000003
6.600000	1.261771	1.261774	0.000003
6.900000	1.394161	1.394165	0.000003
7.200000	1.402015	1.402019	0.000004
7.500000	1.284630	1.284635	0.000005
7.800000	1.052492	1.052499	0.000006
8.100000	0.726338	0.726346	0.000008
8.400000	0.335301	0.335310	0.000009
8.700000	-0.085688	-0.085677	0.000011
9.000000	-0.499024	-0.499012	0.000012
9.300000	-0.867784	-0.867771	0.000013
9.600000	-1.159028	-1.159015	0.000014
9.900000	-1.346741	-1.346727	0.000014
10.000000	-1.175279	-1.175266	0.000013

3.4 Solution of the selected problem using Runge-Kutta Fehlberg Method

3.4.1 MATLAB implementation procedure for RK45 method:

<u>Input:</u>

Starting point a, end point b, initial condition α, step size h

<u>Output:</u>

```
    Step: 1 setting initial condition
        $x = a$
        $y = \alpha$
```

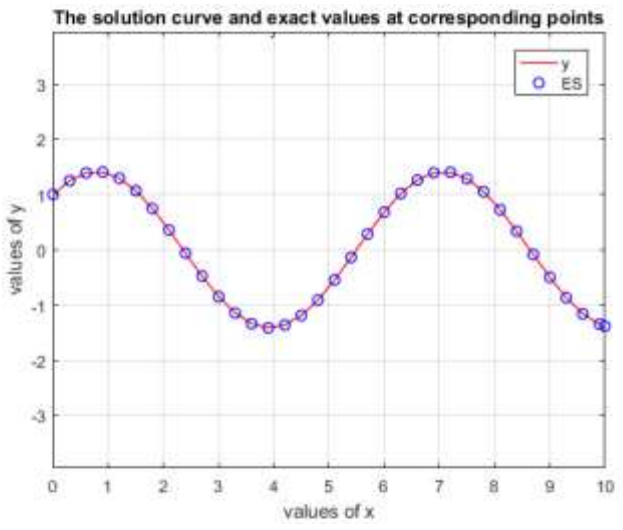

Fig. 3.3.3: Graphical representation of approximated solution with RK4 Method (h=0.3)

```
Step: 2 While $(b > x)$ do step 3-7
Step: 3
        $k1 = h * f(x, y);$
        $k2 = h * f(x + h/4, y + k1/4);$
        $k3 = h * f(x + 3 * h/8, y + 3 * k1/32 + 9 * k2/32);$
        $k4 = h * f(x + 12 * h/13, y + 1932 * k1/2197 - 7200 * k2/2197 + 7296 * k3/2197);$
        $k5 = h * f(x + h, y + 439 * k1/216 - 8 * k2 + 3680 * k3/513 - 845 * k4/4104);$
        $k6 = h * f(x + h/2, y - 8 * k1/27 + 2 * k2 - 3544 * k3/2565 + 1859 * k4/4104 - 11 * k5/40);$

        Step: 4
        $error = abs(k1/360 - 128 * k3/4275 - 2197 * k4/75240.0 + k5/50 + 2 * k6/55)/h;$
        $ynew = y(i) + 25 * k1/216 + 1408 * k3/2565 + 2197 * k4/4104 - k5/5;$
        Step: 5 If (error < tolerance)
        $y = ynew;$
        $\delta = 0.84 * (tolerance/r)^(1/4)$
        $h = \delta * h;$
        Step: 6
        if $(h < hmin)$
        $h = hmin$
        else if $(h > hmax)$
        $h = hmax$
        Step: 7
        if $(x + h > b)$
        $h = b - x$
        $x = x + h;$
Step: 8 Stop.
```

3.3.2 Relative description:

The results of our remaining method of RK45 are contained in **Table: 3.4.1** to **Table: 3.4.3**, where the approximated solution, exact solution, the relative absolute errors, and also the variable step size are presented with the initial step size h=0.75, 0.6, and 0.3 respectively. And the relative figures are displayed in **Fig. 3.4.1** to **Fig. 3.4.2** respectively. And to the previous, the line curve represents the approximated solution and the circle represents the exact solution.

Table 3.4.1: Comparison between RK45 approximated and exact solution with step size h=0.75

x	RKF45	Exact	Step Size	Error
0.000000	1.000000	1.000000	0.750000	0.000000
0.750000	1.413333	1.413328	0.750000	0.000006
1.417406	1.141058	1.141048	0.667406	0.000010
2.127314	0.320883	0.320867	0.709908	0.000016
2.770328	-0.569056	-0.569075	0.643013	0.000019
3.433401	-1.245391	-1.245410	0.663073	0.000019
4.244424	-1.343550	-1.343559	0.811023	0.000009
4.900699	-0.795117	-0.795123	0.656275	0.000006
5.562244	0.091097	0.091093	0.661546	0.000004
6.204693	0.918513	0.918509	0.642449	0.000003
6.907399	1.395889	1.395882	0.702706	0.000007
7.600413	1.218896	1.218884	0.693013	0.000013
8.336521	0.421810	0.421789	0.736108	0.000020
8.982712	-0.476039	-0.476062	0.646191	0.000023
9.639583	-1.190151	-1.190175	0.656871	0.000024
10.00000	-1.383068	-1.383093	0.360417	0.000025

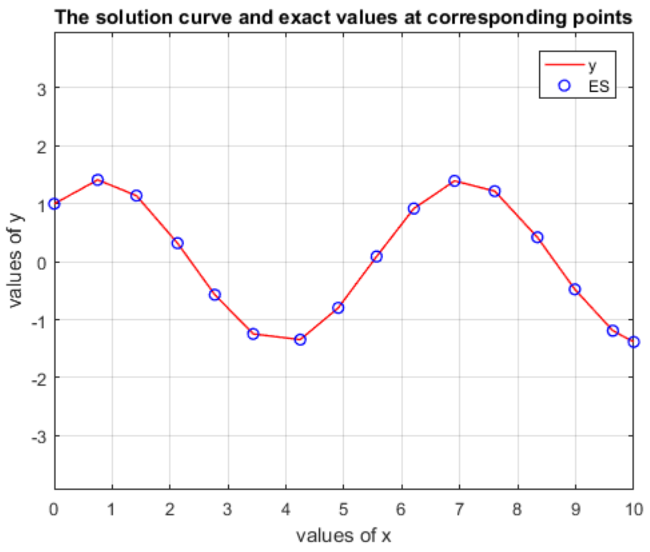

Fig. 3.4.1: Graphical representation of approximated solution with RK45 order Method (h=0.75)

Table 3.4.2: Comparison between RK45 approximated and exact solution with step size h=0.6

x	RKF45	Error	Step Size	Error
0.000000	1.000000	1.000000	0.600000	0.000000
0.600000	1.389979	1.389978	0.600000	0.000001
1.295652	1.234078	1.234072	0.695652	0.000007
2.037966	0.442500	0.442486	0.742313	0.000014
2.685005	-0.456658	-0.456675	0.647040	0.000017
3.340739	-1.178051	-1.178069	0.655734	0.000017
4.114388	-1.389444	-1.389454	0.773648	0.000011
4.778601	-0.931638	-0.931646	0.664213	0.000007
5.453537	-0.062555	-0.062559	0.674936	0.000004
6.094267	0.794415	0.794412	0.640730	0.000004
6.778986	1.355330	1.355324	0.684719	0.000006
7.517214	1.274279	1.274266	0.738228	0.000014
8.285605	0.489965	0.489942	0.768391	0.000023
8.935284	-0.412364	-0.412391	0.649679	0.000027
9.588597	-1.149671	-1.149699	0.653313	0.000028
10.000000	-1.383063	-1.383093	0.411403	0.000029

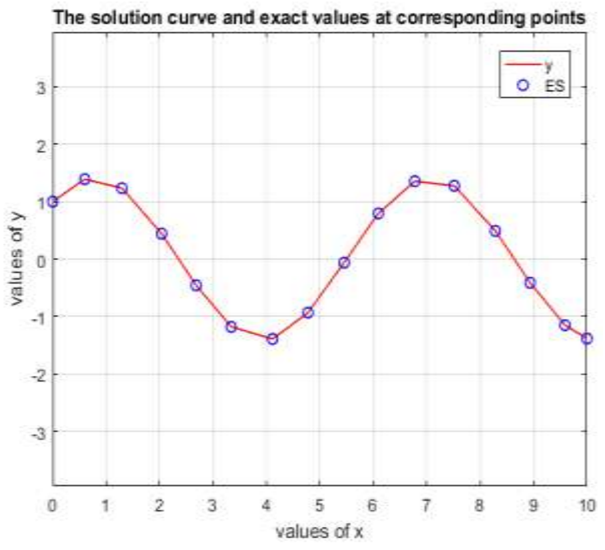

Fig. 3.4.2: Graphical representation of approximated solution with RK45 order

Table 3.4.3: Comparison between RK45 approximated and exact solution with step size h=0.3

x	RK45	Exact	Step Size	Error
0.000000	1.000000	1.000000	0.300000	0.000000
0.300000	1.250857	1.250857	0.300000	0.000000
0.900000	1.404939	1.404937	0.600000	0.000002
1.500000	1.068237	1.068232	0.600000	0.000005
2.100000	0.358370	0.358363	0.600000	0.000007
2.700000	-0.476684	-0.476692	0.600000	0.000009
3.300000	-1.145216	-1.145225	0.600000	0.000009
3.900000	-1.413691	-1.413698	0.600000	0.000008
4.500000	-1.188320	-1.188326	0.600000	0.000006
5.100000	-0.547833	-0.547837	0.600000	0.000004
5.700000	0.284030	0.284027	0.600000	0.000003
6.300000	1.016675	1.016673	0.600000	0.000003
6.900000	1.394169	1.394165	0.600000	0.000004
7.500000	1.284642	1.284635	0.600000	0.000007
8.100000	0.726355	0.726346	0.600000	0.000010
8.700000	-0.085665	-0.085677	0.600000	0.000012
9.300000	-0.867758	-0.867771	0.600000	0.000013
9.900000	-1.346714	-1.346727	0.600000	0.000013
10.00000	-1.383079	-1.383093	0.100000	0.000013

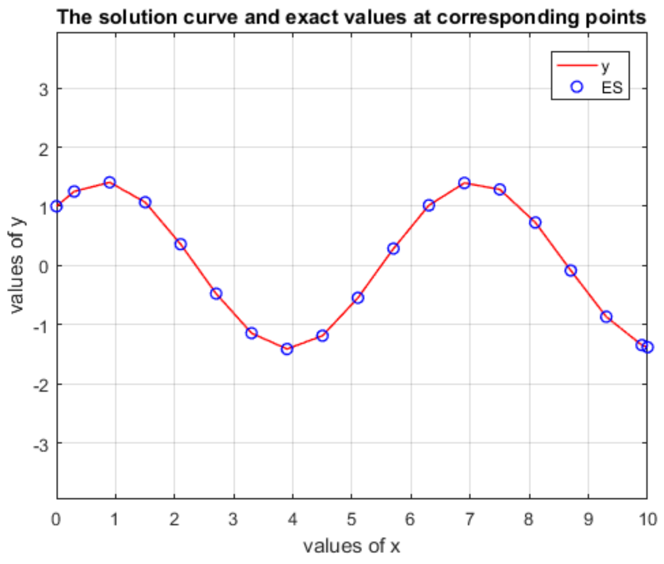

Fig. 3.4.3: Graphical representation of approximated solution with RK45 order Method (h=0.3)

4. Results and discussions

The tables and graphs displayed above are our computed results and comparison with the exact solution. The maximum absolute error is also calculated at each point of approximation with conducted step sizes (0.75, 0.6, 0.3). The bar diagrams **Fig: 4.1-4.3** display the error at each step of our discussed methods for different step sizes respectively and we can easily see that as much as the step size is small the error is small and convergence is faster. But we can examine the difference in the case of the RK45 method. Since the step size could be increased or decreased as required within the MATLAB program and we can choose our suitable tolerance limit, so the accuracy is not much dependent on the initial step size. It is observed from the result and graph that the solution with the RK45 method is so much close to the exact solution in comparison to the Euler method, RK2 method, and RK4 method.

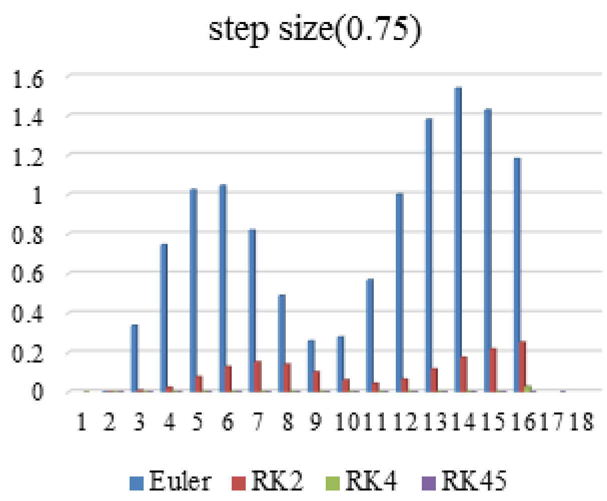

Fig. 4.1: The error of Euler, RK2, RK4, and RK45 methods with step size h=0.75

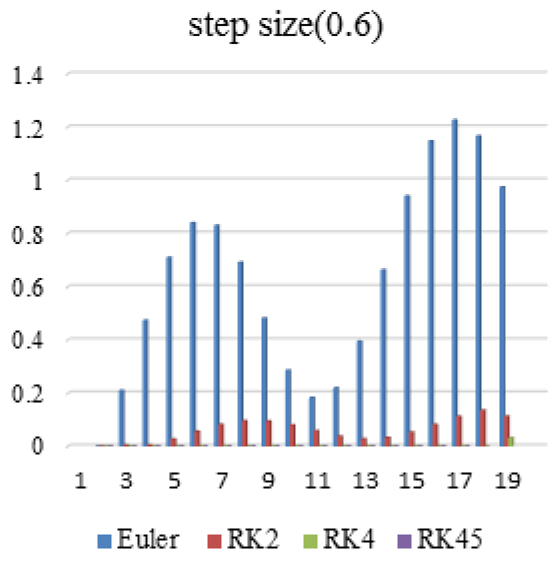

Fig. 4.2: The error of Euler, RK2, RK4, and RK45 methods with step size h=0.6

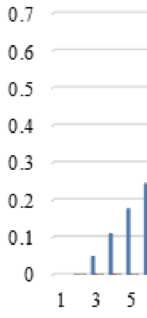

Fig.

5. Conclusion

In this study, the Euler method, Runge-Kutta 2nd order, Runge-Kutta 4th order, and Runge-Kutta Fehlberg methods are used to solve the initial value problem of an ordinary differential equation. Our obtained results from all methods are in good agreement with the actual solution, accuracy depends, and convergence with a small step size. Though The RK2 method gives better accuracy than the Euler method and the result of the RK4 method gives better than RK2, the accuracy of RK45 (which is in our choice) is better than all of these methods. Thus, to solve the initial valued ordinary differential equation, the RK45 method will be the better choice and easily adapted with MATLAB.

Acknowledgement

We would like to express our gratitude to our dear students Md. Imran Hossen and Md. Biddut Rana for their effortless support in completing this paper. We would also like to thanks our colleagues for their advice and support.

Conflict of interest: The authors declare that there is no conflict of interest.

References

[1] P. Gowri , S. Priyadharsini, T. Maheswari (2017). A Case Study on Runge Kutta 4th Order Differential Equations and Its Application. Imperial Journal of Interdisciplinary Research Vol-3, Issue-2.

[2] O.Y. Ababneh, R. Ahmad, & E. S. Ismail, (2009). New multi-step Runge-Kutta method. *Applied Mathematical Sciences*, *3*(45), 2255-2262.

[3] R. A. Ademiluyi, & P. O. Babatola, (2001). Semi implicit rational Runge-Kutta formulas of approximation of stiff initial value problems in ODEs. *Journal of Mathematical Science and Education*, 3, 1-25.

[4] S. A. Agam, Y. A. Yahaya (2014). A highly efficient implicit Runge-Kutta method for first order ordinary differential equations. African Journal of Mathematics and computer science research.7(5), 55-60.

[5] G.C. Paul, A. I. M. Ismail, A. Rahman, M. F. Karim and A. Hoque, (2016). Development of tide-surge interaction model for the coastal region of Bangladesh, *Estuaries and Coasts*, 39(6), 1582–1599.

[6] G. C. Paul, A. I. M. Ismail and M. F. Karim, (2014). Implementation of method of lines to predict water levels due to a storm along the coastal region of Bangladesh, *Journal of Oceanography*, 70(3), 199–210.

[7] S. Senthilkumar and G. C. Paul, (2012). Application of new RKAHeM(4.4) technique to analyze the structure of initial extrasolar giant protoplanets, *Earth Science Informatics*, 5(1), 23–31.

[8] G.C. Paul and S. SenthilKumar, (2015). Exploration on initial structures of extrasolar protoplanets via new explicit RKAHeM(4.4) method, *The Egyptian Journal of Remote Sensing and Space Science*, 18(1), 1–8.

[9] E. Fehlberg, Neue genauere Runge-Kutta-Formeln für Differentialgleichungen*n*-ter Ordnung. Z. Angew. Math. Mech.**40**, 449–455 (1960)

[10] J. C. Butcher, Coefficients for the study of Runge Kutta processes. J. Aust. Math. Soc.**3**, 185–201 (1963).

[11] R. L. Burden & D. J. Faires (1985). Numerical analysis.

[12] E. Balagurusamy (2006) Numerical Methods. Tata McGraw-Hill, New Delhi.

[13] S. S. Sastry (2000) Introductory Methods of Numerical Analysis. Prentice-Hall, India.

[14] C. F. Gerald & P O Wheatley (2002) Applied Numerical Analysis. Pearson Education, India.

[15] J. H. Mathews (2005) Numerical Methods for Mathematics, Science and Engineering. Prentice-Hall, India.

[16] S. M. E. Ali, (2006) A Text Book of Numerical Methods with Computer Programming. Beauty Publication, Khulna.

[17] J. H. Mathews and K. K. Fink, (2004) Numerical Methods Using Matlab 4th Edition. Prentice-Hall Inc. Upper Saddle River, New Jersey, USA

[18] B. Panjaitan, (2016). Designing a Learning Media using Matlab for Matrix and Its Operations.

[19] C. Lopez, (2014). *MATLAB optimization techniques*. Apress.