ORIGINAL PAPER

NUMERICAL COMPUTATION OF INITIAL VALUE PROBLEM BY VARIOUS TECHNIQUES

AZIZUL HASAN[1]

Manuscript received: 11.07.2017; Accepted paper: 22.09.2017;
Published online: 30.03.2018.

Abstract. *The purpose of this paper is, to study the scientific computation of initial value problem and to show the details of implementing a few steps of Euler's method, as well as how to use built-in functions available in MATLAB(2009a)[2]. In the first part, we use Euler methods to introduce the basic ideas associated with initial value problems (IVP). In the second part, we use the Runge-Kutta method presented together with the built-in MATLAB solver Ode45 and Ode23 .The implementations that we develop in this paper are designed to build intuition and are the first step from textbook formula on ODE to production software and also we have studied accuracy and efficiency of various methods.*

Keywords: *Initial value problem, Euler's methods, Runge Kutta method, ode45, ode23, Matlab, Scientific computation, accuracy and efficiency.*

1. INTRODUCTION

In the field of Engineering and Science, we come across physical and natural phenomena which, when represented by mathematical models happen to be differential equations. For example, simple harmonic motion, equation of motion, deflection of a beam, etc. are represented by differential equations. Hence solution of differential equation is a necessity in such studied. There are number of differential equations which we studied in calculus to get closed form solutions. But all differential equations do not possess closed form solutions or finite form solutions. Even they possess closed form solutions; we do not know the method of getting it. In such situations depending upon the need of the hour, we go in for numerical solutions of differential equations. In researches, especially after the advent of computer, the numerical solutions of the differential equations have becomes easy for manipulation. The dynamic behavior of systems is an important subject. A mechanical system involves displacements, velocities, and accelerations. An electric or electronic system involves voltages, currents, and time derivatives of these quantities. An equation that involves one or more derivatives of the unknown function is called an ordinary differential equation, abbreviated as ODE. The order of the equation is determined by the order of the highest derivative. The problems of solving an ode are classified into initial-value problems (IVP) and boundary value problems (BVP), depending on how the conditions at the endpoints of the domain are specified. All the conditions of an initial-value problem are specified at the initial point. On the other hand, the problem becomes a boundary-value problem if the conditions are needed for both initial and final points. The ode in the time domain are initial-value problems, so all the conditions are specified at the initial time, such as t = 0 or x = 0. For

[1] Jazan University, Faculty of Science, Department of Mathematics, 45142 Jazan, Saudi Arabia.
E-mail: azizulhasan.math@gmail.com

notations, we use t or x as an independent variable. Some literatures use t as time for independent variable. It is important to note that our focus here is on the practical use of numerical methods in order to solve some typical problems, not to present any consistent theoretical background. Today there are numerous methods that produce numerical approximations to the solution of differential equations. There are many excellent and exhaustive texts on these subjects that may be consulted [3-9].

1.1. DEFINITION AND NOTATION

An ordinary differential equation is a relation between a function, its derivative, and the variable upon which they depend. The most general form of an ordinary differential equation is given by

$$\emptyset\left(x, y', y'', \dots\dots\dots\dots\dots, y^{(m)} \right) = 0 \qquad \text{(i)}$$

where m represents the highest order derivative, and y and its derivative are function of x. A linear differential equation of order m can be expressed in the form

$$\sum_{p=0}^{m} \emptyset_p\,(x) y^{(p)}(x) = r(x) \qquad \text{(ii)}$$

in which $\emptyset_p(x)$ are known function. If the general nonlinear differential equation (i) of order m can be written as $y^{(m)} = F(x, y, y', \dots\dots, y^{(m-1)}$

Then this equation is called a canonical representation of the differential equation (i).

1.2. INITIAL VALUE PROBLEM

A general solution of an ordinary differentia equation such as (i) is a relation between y, x and m arbitrary constant, which satisfy the equation, but which contain no derivatives. The solution may be an explicit form

$$y = w(x, c_1, c_2, \dots\dots\dots c_m) \qquad \text{(iii)}$$

The m arbitrary constant $c_1, c_2, \dots\dots\dots c_m$ can be determined by prescribing m conditions of the form

$$y^{(k)}(x_0) = \gamma_k, \qquad k = 0,1,2,3 \dots\dots\dots, m-1\,, \qquad \text{(iv)}$$

at one point $x = x_0$ which are called initial conditions. The point x_0 is called initial point. The differential equation (i) together with the initial conditions (IV) is called an m[th] order initial value problem [1].

Theorem: We assume that f(x, y) satisfies the following conditions:

(i) f(x, y) is a real function

(ii) f(x, y) is defined and continuous in the strip $x \in [x_0\,, b], y \in (-\infty, \infty)$

(iii) then there exist a constant L such that for any $x \in [x_0\,, b]$ and any $y_1\ and\ y_2$ $|f(x, y_1) - f(x, y_2)\,| \le L|y_1 - y_2|$ where L is called Lipschitz constant. Then for any y_o, the initial value problems, $\frac{dy}{dx} = f(x, y), \quad y(x_0) = \gamma_0,$ has a unique solution y(x) for $x \in [x_0\,, b]$ (2010) [1].

2. MATERIALS AND METHODS

Numerical methods are commonly used for solving mathematical problems that are formulated in science and engineering where it is difficult or even impossible to obtain exact solutions [10, 11]. Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can give an approximate solution to (almost) any equation. Literal implementation of this procedure results in Euler's method, which is, however, not recommended for any practical use. There are other methods more sophisticated than Euler's. Among them, there are three major types of practical numerical methods for solving initial value problems for ODEs: (i) Runge-Kutta methods, (ii) Burlirsch-Stoer method, and (iii) predictor-corrector methods. Now, we are interested to talk about Euler's methods, Runge Kutta method and built in function. We shall first explain the principle involved in the various methods. Integrating the differential equation $y' = f(x, y)$ on the interval $[x_j, x_{j+1}]$, we get

$$\int_{x_j}^{x_{j+1}} \frac{dy}{dx} . dx = \int_{x_j}^{x_{j+1}} f(x, y) dx \tag{1}$$

By the mean value theorem of integral calculus, we obtain

$$y(x_{j+1}) = y(x_j) + hf\left(x_j + \theta h , y(x_j + \theta h)\right) \quad , \quad 0 < \theta < 1 \tag{2}$$

Any value of $\theta \in [0,1]$ produces a numerical method. Consider the following cases.

(i) Case $\theta = 0$, when $\theta = 0$, we obtain the approximation

$$y_{j+1} = y_j + hf(x_j , y_j) \tag{3}$$

which is Euler Method

(ii) Case $\theta = 1$, when $\theta = 1$, we obtain the approximation

$$y(x_{j+1}) = y(x_j) + hf\left(x_j + h , y(x_j + h)\right)$$

Now we have the numerical method as

$$y_{j+1} = y_j + hf_{j+1} \tag{4}$$

which is Euler backward method.

(iii) Case $\theta = 1/2$, when $\theta = 1/2$, we obtain the approximation

$$y(x_{j+1}) = y(x_j) + hf\left(x_j + \frac{h}{2}, y\left(x_j + \frac{h}{2}\right)\right) \tag{5}$$

However $x_j + h/2$ is not a nodal point .If we approximate the $y(x_j + h/2)$ in (5), by Euler method with spacing $h/2$ we get

$$y\left(x_j + \frac{h}{2}\right) = y_j + \frac{h}{2} f_j$$

Then we have the approximation

$$(y_{j+1}) = (y_j) + hf\left(x_j + \tfrac{h}{2}, \left(y_j + \tfrac{h}{2} f_j\right)\right) \quad (6)$$

If we set $k_1 = hf_j$ and

$$k_2 = hf\left(x_j + \frac{h}{2}, \left(y_j + \frac{k_1}{2}\right)\right)$$

Then the (6) can be written as

$$(y_{j+1}) = (y_j) + k_2 \quad (7)$$

Alternatively, if we use the approximation

$$y'\left(x_j + \tfrac{h}{2}\right) \approx \tfrac{1}{2}\left[y'(x_j) + y'(x_j + h)\right]$$

and the Euler method we obtain

$$y'\left(x_j + \tfrac{h}{2}\right) \approx \tfrac{1}{2}\left[f(x_j, y_j) + f(x_{j+1}, y_j + hf_j)\right]$$

thus equation (5) may be approximated by

$$y_{j+1} = y_j + \tfrac{h}{2}[f(x_j, y_j) + f(x_{j+1}, y_j + hf_j) \quad (8)$$

which is called Euler modified method.(2010)[1]

RUNGE KUTTA METHOD

There are many variants of runge kutta methods but the most widely used one is the following given below $y' = f(x, y), \quad y(x_j) = y_j, where\ x_j = x_0 + jh$. We will compute in term of

$$k_1 = hf(x_j, y_j)$$
$$k_2 = hf(x_j + \tfrac{h}{2}, y_j + \tfrac{k_1}{2})$$
$$k_3 = hf(x_j + \tfrac{h}{2}, y_j + \tfrac{k_2}{2})$$
$$k_4 = hf(x_j + h, y_j + k_3)$$
$$y_{j+1} = y_j + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

Runge kutta method is a fourth order method, meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4. Note that the above formula is valid for both scalar and vector valued function.

MATLAB PROGRAMS

Write all programs in M-file and save it as euler.m, euler_backward.m, euler_modified.m, rk4.m. [14]

```
function [x,y] = euler(h, x0, y0, interval_length, func)
nsteps = floor(interval_length/h) + 1;
x = zeros(nsteps,1);
y = zeros(nsteps,1);
x(1) = x0;
y(1) = y0;
for i=2:nsteps
   y(i) = y(i-1) + h* func(x(i-1), y(i-1));
   x(i) = x(i-1) + h;
end
```

```
function [x,y]=euler_backward(f,xinit,yinit,xfinal,n)
% Euler approximation for ODE initial value problem
% Calculation of h from xinit, xfinal, and n
h=(xfinal-xinit)/n;
% Initialization of x and y as column vectors
x=[xinit zeros(1,n)];
y=[yinit zeros(1,n)];
% Calculation of x and y
for i=1:n
x(i+1)=x(i)+h;
ynew=y(i)+h*(f(x(i),y(i)));
y(i+1)=y(i)+h*f(x(i+1),ynew);
end
```

```
function [x,y]=euler_modified(f, xinit, yinit, xfinal,n)
% Euler approximation for ODE initial value problem
% Calculation of h from xinit, xfinal, and n
h=(xfinal-xinit)/n;
% Initialization of x and y as column vectors
x=[xinit zeros(1,n)]; y=[yinit  zeros(1,n)];
% Calculation of x and y
for i=1:n
x(i+1)=x(i)+h;
ynew=y(i)+h*f(x(i),y(i));
y(i+1)=y(i)+(h/2)*(f(x(i),y(i))+f(x(i+1), ynew));
end
```

```
 function [x,y] = rk4(h, x0, y0, interval_length,func)
h=0.1;                                  % step size
x = 0:h:1;                              % Calculates upto y(10)
y = zeros(1,length(x));                 % Calculation of loop
y(1) = 1;                               % initial condition
F = @(t,r) 2.*(-t)*r^2;                 % change the function as you desire
for i=1:(length(x)-1)                   % calculation loop
   k_1 = F(x(i),y(i));
```

```
   k_2 = F(x(i)+0.5*h,y(i)+0.5*h*k_1);
   k_3 = F((x(i)+0.5*h),(y(i)+0.5*h*k_2));
   k_4 = F((x(i)+h),(y(i)+k_3*h));
y(i+1) = y(i) + (1/6)*(k_1+2*k_2+2*k_3+k_4)*h;
end
```

USING BUILT IN FUNCTION

Matlab has a several different functions (built-ins) for the numerical solution of ordinary differential equations. In this section we will present two of them. We will also give an example how to use it, instead of writing our own matlab code. These solvers can be used with the following syntax:

[x, y] = solver (@ odefun, tspan, y0)

Solver is the solver as you are using, such as name, ode45, ode23 or odefun is the function that defines the derivatives so odefun defines y' as a function of the independent parameter like as x or t as well as y , tspan a vector that specifies the interval of the solution [to, tf]. y0 is the initial value of y. [x, y] is the output, which is the solution of the ordinary differential equation.

USING ode45

ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dorman-Prince pair [12]. That means the numerical solver ode45 combines a fourth order method and a fifth order method, both of which are similar to the classical fourth order Runge-Kutta (RK) method discussed above. The modified RK varies the step size, choosing the step size at each step in an attempt to achieve the desired accuracy. Therefore, the solver ode45 is suitable for a wide variety of initial value problems in practical applications. In general, ode45 is the best function to apply as a ‘’first try" for most problems. It is important to note that in MATLAB 7.8 (2009a) version, it is preferred to have ode fun in the form of a function handle. For example, it is recommended to use the following syntax,

ode45 (@xdot, tspan, y0)**, or** ode45 ('xdot', tspan, y0)

USING ode23

This is also based on Runge-Kutta formula. That means the numerical solver ode23 combines second order and third order Runge–Kutta method.

[x, y]=de23 (f, xvalues, y0)

Note the use of @xdot and 'xdot'. Use function handles to pass any function that defines quantities the MATLAB solver will compute, in particular for simple functions. On the other hand, it is also important to remember that complicated differential equations should be written an M-file instead of using inline command or function handle. Here, we use the same data as defined in the first part for Euler's methods. The initial conditions and the time steps are the same as before. The integration proceeds by steps, taken to the values specified in tspan. Note that the step size does not have to be uniform. For more information on numerical methods, we refer to Sham pine [13].

3. ANALYSIS OF CONVERGENCE OF METHODS

The numerical solutions y_i will contain errors. We shall be concerned with the effect of these errors on the solutions, and ask what happens as we try to get a more accurate solution, by taking more grid points. A method is convergent if, as more grid points are taken or step size is decreased, the numerical solution converges to the exact solutions, in the absence of round –off errors.

4. NUMERICAL COMPUTATION AND COMPARATIVE DISCUSSION

In this section, we employ the different techniques, obtained in this paper to solve the ordinary differential equations with initial value problems and compare them. We use the stopping criteria up to fifteen decimal places. We have $E_{i+1} = y(x_{i+1}) - y_{i+1}$, $i = 0,1, \dots \dots \dots, N-1$ for computer programs. All programs are written in Matlab 2009a [2]. Let us consider the initial value problem $\frac{dy}{dx} = -2xy^2$, $y(0) = 1$ over the interval [0, 1].

The exact solution for this problem is $y = \frac{1}{1+x^2}$

Table 1.Numerical and exact solutions of various techniques at different grid points, when h=0.1

Values of x_i	Numerical Solution of Euler method	Numerical Solution of Euler backward	Numerical Solution of Euler modified method	Exact Solution $y(x_i)$
0.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000
0.100000000000000	1.000000000000000	0.980000000000000	0.990000000000000	0.990099009900990
0.200000000000000	0.980000000000000	0.943075149309440	0.961365554431920	0.961538461538461
0.300000000000000	0.941584000000000	0.893661826601068	0.917245807332359	0.917431192660550
0.400000000000000	0.888389174256640	0.836439201775134	0.861954319809959	0.862068965517241
0.500000000000000	0.825250348261728	0.775526053442303	0.800034025054427	0.800000000000000
0.600000000000000	0.757146534531118	0.714113486880054	0.735527018675442	0.735294117647059
0.700000000000000	0.688354029560820	0.654431124078703	0.671587054764514	0.671140939597315
0.800000000000000	0.622017651759054	0.597887627676893	0.610398784831307	0.609756097560976
0.900000000000000	0.560112698303078	0.545264921454263	0.553289088518793	0.552486187845304
1.000000000000000	0.503641976039014	0.496901617530450	0.500918575857537	0.500000000000000

Table 2.Numerical and exact solutions of various techniques at different grid points when h =0.1

Values of x_i	Exact Solution $y(x_i)$	Numerical Solution of RK Forth order	Numerical Solution of $y(x_i)$ using Ode-45	Numerical Solution of $y(x_i)$ Using Ode 23
0.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000
0.100000000000000	0.990099009900990	0.990098924950166	0.990099010281578	0.990099625000000
0.200000000000000	0.961538461538461	0.961538143658087	0.961538461579960	0.961542619151974
0.300000000000000	0.917431192660550	0.917430597519571	0.917431191784696	0.917441957273036
0.400000000000000	0.862068965517241	0.862068183488285	0.862068963452397	0.862086986706123
0.500000000000000	0.800000000000000	0.799999209018538	0.799999996783799	0.800022805971222
0.600000000000000	0.735294117647059	0.735293500279022	0.735294113852340	0.735317375528039
0.700000000000000	0.671140939597315	0.671140619517870	0.671140936374488	0.671160344247257
0.800000000000000	0.609756097560976	0.609756119188268	0.609756096232322	0.609768677623216
0.900000000000000	0.552486187845304	0.552486529856882	0.552486189373518	0.552490683631081
1.000000000000000	0.500000000000000	0.500000602210524	0.500000004711942	0.499996585223659

Table 3. Estimation of errors analysis by various techniques at grid points when h =0.1

Error in Euler method	Error in Euler backward method	Error in Euler modified method	Error in Runge Kutta method	Error in ODE-23 method	Error in ODE 45 method
0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000
0.009900990099010	0.010099009900990	0.000099009900990	0.000000084950 8239	0.000000615099010	0.000000000380588
0.018461538461539	0.018463312229021	0.000172907106541	0.000000317880 3740	0.000004157613513	0.000000000041499
0.024152807339450	0.023769366059482	0.000185385328191	0.000000595140 9789	0.000010764612486	0.000000000875854
0.026320208739399	0.025629763742107	0.000114645707282	0.000000782028 9560	0.000018021188882	0.000000002064844
0.025250348261728	0.024473946557697	0.000034025054427	0.000000790981 4620	0.000022805971222	0.000000003216201
0.021852416884059	0.021180630767005	0.000232901028383	0.000000617368 0370	0.000023257880980	0.000000003794719
0.017213089963505	0.016709815518612	0.000446115167199	0.000000320079 4451	0.000019404649942	0.000000003222827
0.012261554198078	0.011868469884083	0.000642687270331	0.000000021627 2920	0.000012580062240	0.000000001328654
0.007626510457774	0.007221266391041	0.000802900673489	0.000000342011 5779	0.000004495785777	0.000000001528214
0.003641976039014	0.003098382469550	0.000918575857537	0.000000602210 5240	0.000003414776341	0.000000004711942
Average absolute error= 0.015152858222142	Average absolute error= 0.014773996683599	Average absolute error= 0.000331741190397	Average absolute error= 0.000000040675 2679	Average absolute error= 0.000010865240036	Average absolute error= 0.000000001924122

Table 3 shows that the computational data of accuracy of various techniques at different grid point. And also shows that the average absolute errors of the methods. It is observed that the error data are found in the following order between [0, 1] when h=0.1

$$0.015152858222142 > 0.014773996683599 > 0.000331741190397 > 0.000010865240036 > 0.0000000406752679 > 0.000000001924122$$

Thus the orders of the accuracy of methods near to exact solution are in the following manner:

$$\text{Ode45} > \textit{Runge Kutta method} > \textit{Ode23} > \textit{Euler modified} > \textit{Euler backward} > \textit{Euler}$$

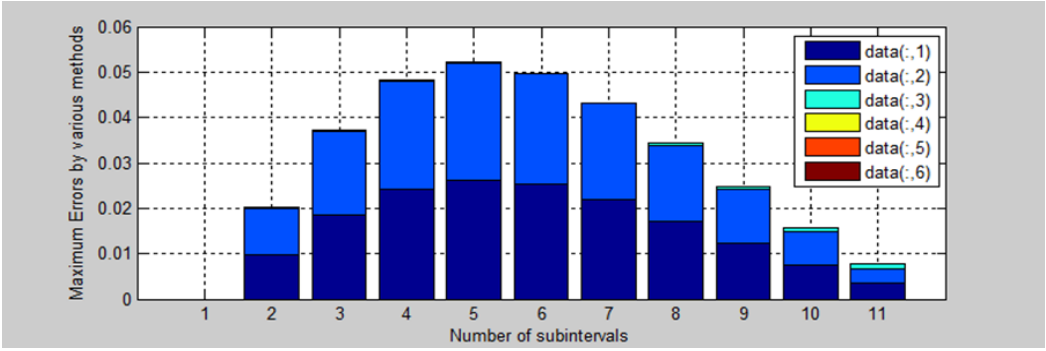

Figure 1. Comparison of errors among the techniques versus x.

According to Fig.1, the calculated results using the built-in function (ode45) give a very good result when compared and also we see that if we connect the maximum error points of some methods then we get a curve. Which is looks like a downward parabola.

Table 4. Numerical and exact solutions of various techniques at different grid points when h=0.02,

Values of x_i	Numerical Solution of Euler Method	Numerical Solution of Euler backward Method	Numerical Solution of Euler modified Method	Exact Solution $y(x_i)$
0	1.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000
0.020000000000000	1.000000000000000	0.999200000000000	0.999600000000000	0.999600159936026
0.040000000000000	0.999200000000000	0.997605111816187	0.998402237761433	0.998402555910543
0.060000000000000	0.997602558976000	0.995224212382756	0.996412442275457	0.996412913511359
0.080000000000000	0.995214052898379	0.992069827336063	0.993640085115513	0.993640699523052
0.100000000000000	0.992044609662902	0.988157973270470	0.990098268253272	0.990099009900990
0.120000000000000	0.988107999632658	0.983507962269229	0.985803570485189	0.985804416403785
0.140000000000000	0.983421484021755	0.978142173031229	0.980775855800542	0.980776775205963
0.160000000000000	0.978005624256436	0.972085793482648	0.975038047770964	0.975039001560062
0.180000000000000	0.971884056249542	0.965366540168724	0.968615874652894	0.968616815187912
0.200000000000000	0.965083234194239	0.958014359959828	0.961537590352404	0.961538461538461
0.220000000000000	0.957632149002856	0.950061119679929	0.953833676697604	0.953834414345670
0.240000000000000	0.949562026874183	0.941540289183325	0.945536532595976	0.945537065052950
0.260000000000000	0.940906013662521	0.932486623181613	0.936680155627804	0.936680404645935
0.280000000000000	0.931698850746440	0.922935846776348	0.927299821453883	0.927299703264095
0.300000000000000	0.921976547963439	0.912924349205827	0.917431766112471	0.917431192660550
0.320000000000000	0.911776058903504	0.902488889790665	0.907112875867257	0.907111756168360
0.340000000000000	0.901134963459157	0.891666319486775	0.896380388767864	0.896378630333453
0.360000000000000	0.890091162034944	0.880493320849307	0.885271611521121	0.885269121813031
0.380000000000000	0.878682585249993	0.869006168599092	0.873823654668854	0.873820342537574
0.400000000000000	0.866946922348545	0.857240512383376	0.862073188449164	0.862068965517241
0.420000000000000	0.854921370889831	0.845231182751575	0.850056221103739	0.850051003060184
0.440000000000000	0.842642409643041	0.833012020837558	0.837807900801784	0.837801608579088
0.460000000000000	0.830145595985730	0.820615731762442	0.825362341796566	0.825354902608122
0.480000000000000	0.817465388511895	0.808073761352588	0.812752474924953	0.812743823146944
0.500000000000000	0.804634995012729	0.795416195410181	0.800009922112014	0.800000000000000
0.520000000000000	0.791686245508747	0.782671680479358	0.787164894156822	0.787153652392947
0.540000000000000	0.778649489593130	0.769867364818186	0.774246110754311	0.774233508826262
0.560000000000000	0.765553516996027	0.757028858112828	0.761280741451143	0.761266747868453
0.580000000000000	0.752425499998603	0.744180208350635	0.748294366038670	0.748278958395690
0.600000000000000	0.739290956111886	0.731343894198237	0.735310952749278	0.735294117647059
0.620000000000000	0.726173729284954	0.718540831203229	0.722352852538449	0.722334585379948
0.640000000000000	0.713095987814385	0.705790390147351	0.709440807697869	0.709421112372304
0.660000000000000	0.700078237085758	0.693110425919176	0.696593973048371	0.696572861521315
0.680000000000000	0.687139345281473	0.680517315339109	0.683829947998975	0.683807439824945
0.700000000000000	0.674296580229992	0.668026002453491	0.671164817823343	0.671140939597315
0.720000000000000	0.661565655642916	0.655650049912605	0.658613202591552	0.658587987355111
0.740000000000000	0.648960785081200	0.643401695154926	0.646188312297659	0.646161798914448
0.760000000000000	0.636494742104233	0.631291910233018	0.633902006837026	0.633874239350913
0.780000000000000	0.624178925179753	0.619330464231723	0.621764859607446	0.621735886595374
0.800000000000000	0.612023426063830	0.607525987343982	0.609786223630921	0.609756097560976
0.820000000000000	0.600037100494201	0.595886035781397	0.597974299215531	0.597943075819182
0.840000000000000	0.588227640173602	0.584417156803912	0.586336202296440	0.586303939962477
0.860000000000000	0.576601645149684	0.573124953254307	0.574878032709966	0.574844791906185
0.880000000000000	0.565164695822372	0.562014147077756	0.563604941763140	0.563570784490532
0.900000000000000	0.553921423926551	0.551088641393856	0.552521198562373	0.552486187845304
0.920000000000000	0.542875581946698	0.540351580768033	0.541630254657983	0.541594454072790
0.940000000000000	0.532030110519656	0.529805409400945	0.530934806646149	0.530898279889573
0.960000000000000	0.521387203472072	0.519451927018621	0.520436856446178	0.520399666944213
0.980000000000000	0.510948370219806	0.509292342302756	0.510137769038970	0.510099979596001
1.000000000000000	0.500714495328220	0.499327323750286	0.500038327512522	0.500000000000000

Table 5. Numerical and exact solutions of various techniques at different grid points when h =0.02

Values of x_i	Numerical Solution of RK4	Numerical Solution of $y(x_i)$ using ode45	Numerical Solution of $y(x_i)$ using ode 23	Exact Solution $y(x_i)$
0	1.000000000000000	1.000000000000000	1.000000000000000	1.000000000000000
0.020000000000000	0.999600159930688	0.999600159936049	0.999600159976000	0.999600159936026
0.040000000000000	0.998402555889244	0.998402522372726	0.998400273516781	0.998402555910543
0.060000000000000	0.996412913463756	0.996412879948895	0.996407945460614	0.996412913511359
0.080000000000000	0.993640699439335	0.993640711952511	0.993636091638163	0.993640699523052
0.100000000000000	0.990099009772114	0.990099037725034	0.990097627880086	0.990099009900990
0.120000000000000	0.985804416221694	0.985804416661433	0.985805470017046	0.985804416403785
0.140000000000000	0.980776774963775	0.980776712851776	0.980776635645589	0.980776775205963
0.160000000000000	0.975039001252236	0.975038948248061	0.975037975951441	0.975039001560062
0.180000000000000	0.968616814810373	0.968616850808976	0.968617157149040	0.968616815187912
0.200000000000000	0.961538461088691	0.961538517467479	0.961541845452820	0.961538461538461
0.220000000000000	0.953834413822753	0.953834414130793	0.953839707077218	0.953834414345670
0.240000000000000	0.945537064457590	0.945537002227058	0.945542472155865	0.945537065052950
0.260000000000000	0.936680403980421	0.936680357200526	0.936686970989374	0.936680404645935
0.280000000000000	0.927299702532237	0.927299748411198	0.927308520042654	0.927299703264095
0.300000000000000	0.917431191867583	0.917431252816287	0.917442435780614	0.917431192660550
0.320000000000000	0.907111755320808	0.907111754970220	0.907124034668161	0.907111756168360
0.340000000000000	0.896378629438982	0.896378587308326	0.896391860757806	0.896378630333453
0.360000000000000	0.885269120880274	0.885269095116424	0.885284707364703	0.885269121813031
0.380000000000000	0.873820341575943	0.873820381963871	0.873838264847721	0.873820342537574
0.400000000000000	0.862068964536735	0.862069010141739	0.862088223565730	0.862068965517241
0.420000000000000	0.850051002071189	0.850051000662816	0.850070273877600	0.850051003060184
0.440000000000000	0.837801607592188	0.837801589450798	0.837821948270782	0.837801608579088
0.460000000000000	0.825354901633908	0.825354895422538	0.825377571740036	0.825354902608122
0.480000000000000	0.812743822195846	0.812743846344375	0.812768023405191	0.812743823146944
0.500000000000000	0.799999999082119	0.800000020946025	0.800024182386079	0.800000000000000
0.520000000000000	0.787153651517920	0.787153648920578	0.787176927802530	0.787153652392947
0.540000000000000	0.774233508003137	0.774233506416054	0.774257451691547	0.774233508826262
0.560000000000000	0.761266747105587	0.761266750006942	0.761292120079632	0.761266747868453
0.580000000000000	0.748278957700667	0.748278962166164	0.748304573070864	0.748278958395690
0.600000000000000	0.735294117026630	0.735294117256753	0.735318450769320	0.735294117647059
0.620000000000000	0.722334584839991	0.722334581531862	0.722357393279078	0.722334585379948
0.640000000000000	0.709421111917799	0.709421117235563	0.709444033895727	0.709421112372304
0.660000000000000	0.696572861156340	0.696572863361376	0.696596042064582	0.696572861521315
0.680000000000000	0.683807439552686	0.683807427232481	0.683829612114311	0.683807439824945
0.700000000000000	0.671140939420088	0.671140925543085	0.671160938373580	0.671140939597315
0.720000000000000	0.658587987274397	0.658587984358418	0.658606215171057	0.658587987355111
0.740000000000000	0.646161798930938	0.646161804818529	0.646179708070434	0.646161798914448
0.760000000000000	0.633874239464556	0.633874236398228	0.633891428709184	0.633874239350913
0.780000000000000	0.621735886805437	0.621735863962138	0.621751190526640	0.621735886595374
0.800000000000000	0.609756097866101	0.609756078066863	0.609768806962136	0.609756097560976
0.820000000000000	0.597943076217451	0.597943074960989	0.597954091455005	0.597943075819182
0.840000000000000	0.586303940451474	0.586303944653023	0.586314429892699	0.586303939962477
0.860000000000000	0.574844792483062	0.574844784296657	0.574853968321657	0.574844791906185
0.880000000000000	0.563570785152067	0.563570758381673	0.563577660419692	0.563570784490532
0.900000000000000	0.552486188587964	0.552486169070433	0.552490459864621	0.552486187845304
0.920000000000000	0.541594454892786	0.541594456197878	0.541597320334257	0.541594454072790
0.940000000000000	0.530898280782915	0.530898282094853	0.530900775609700	0.530898279889573
0.960000000000000	0.520399667906760	0.520399662169587	0.520401174321899	0.520399666944213
0.980000000000000	0.510099980623505	0.510099974187796	0.510100192423444	0.510099979596001
1.000000000000000	0.500000001088152	0.500000002662208	0.499999505866926	0.500000000000000

Table.6. Estimation of errors by various techniques at different grid points with h =0.02

Error in Euler method	Error in Euler backward method	Error in Euler modified method	Error in Runge Kutta method	Error in ode-23 method	Error in ODE 45 method
0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000	0.0000000000000000
0.0003998400639740	0.0004001599360260	0.0000001599360260	0.0000000000053381	0.0000000000399740	0.0000000000000023
0.0007974440894560	0.0007974440943560	0.0000003181491100	0.0000000000212991	0.0000022823937621	0.0000000335378017
0.0011896454646041	0.0011887011286029	0.0000004712359002	0.0000000000476029	0.0000049680507450	0.0000000335624064
0.0015733533753327	0.0015708721869890	0.0000006144075039	0.0000000000837170	0.0000046078848890	0.0000000124294059
0.0019455997619112	0.0019410366305200	0.0000007416477718	0.0000000001288760	0.0000013820209039	0.0000000278240044
0.0023035832288273	0.0022964541345560	0.0000008459185960	0.0000000001820910	0.0000010536132610	0.0000000002576048
0.0026447088157792	0.0026346021747340	0.0000009194054021	0.0000000002421879	0.0000001395603739	0.0000000623541087
0.0029666226963774	0.0029532080774140	0.0000009537890998	0.0000000003078260	0.0000010256086209	0.0000000533120101
0.0032672410616130	0.0032502750191880	0.0000009405350118	0.0000000003775390	0.0000003419611280	0.0000000356210064
0.0035447726557778	0.0035241015786330	0.0000008711860057	0.0000000004497700	0.0000033839143589	0.0000000559290018
0.0037977346571786	0.0037732946657409	0.0000007376480666	0.0000000005229169	0.0000052927315480	0.0000000002148777
0.0040249618212333	0.0039967758696250	0.0000005324569774	0.0000000005953600	0.0000054071029150	0.0000000628258992
0.0042256090165186	0.0041937814643220	0.0000002490181131	0.0000000006655141	0.0000065663434390	0.0000000474454009
0.0043991474823245	0.0043638564877471	0.0000001181897888	0.0000000007318580	0.0000088167785589	0.0000000451471003
0.0045453553028289	0.0045068434547230	0.0000005734519021	0.0000000007929669	0.0000112431200641	0.0000000601557037
0.0046643027351444	0.0046228663776951	0.0000011196988997	0.0000000008475520	0.0000122784998010	0.0000000011981440
0.0047563331257004	0.0047123108466780	0.0000017584344111	0.0000000008944709	0.0000132304243530	0.0000000430251227
0.0048220402219113	0.0047758009637240	0.0000024897080090	0.0000000009327570	0.0000155855516720	0.0000000266966007
0.0048622427124119	0.0048141739384821	0.0000033121312080	0.0000000009616310	0.0000179223101470	0.0000000394262997
0.0048779568313004	0.0048284531338650	0.0000042229319023	0.0000000009805060	0.0000192580484890	0.0000000446244998
0.0048703678296747	0.0048198203086090	0.0000052180435055	0.0000000009889950	0.0000192708174160	0.0000000023973068
0.0048408010639053	0.0047895877415300	0.0000062922226096	0.0000000009869000	0.0000203396916940	0.0000000191282090
0.0047906933776008	0.0047391708456801	0.0000074391884044	0.0000000009742140	0.0000226691319140	0.0000000071855084
0.0047215653649551	0.0046700617943560	0.0000086517780009	0.0000000009510981	0.0000242002582470	0.0000000231974031
0.0046349950127229	0.0045838045898190	0.0000099221120014	0.0000000009178810	0.0000241823860789	0.0000000209460025
0.0045325931158000	0.0044819719135890	0.0000112417638075	0.0000000008750269	0.0000232754095830	0.0000000034723069
0.0044159807668068	0.0043661440080760	0.0000126019280049	0.0000000008231250	0.0000239428652851	0.0000000024102008
0.0042867691275074	0.0042378897556250	0.0000139935826090	0.0000000007628661	0.0000253722111789	0.0000000021384089
0.0041465416029113	0.0040987500450550	0.0000154076429080	0.0000000006950230	0.0000256146751739	0.0000000037704074

0.0039968384648 27	0.0039502234488 220	0.0000168351022 19	0.0000000006204 290	0.0000243331222 610	0.0000000003903 06
0.0038391439050 06	0.0037937541767 190	0.0000182671585 01	0.0000000005399 570	0.0000228078991 300	0.0000000038480 86
0.0036748754420 81	0.0036307222249 530	0.0000196953255 65	0.0000000004545 050	0.0000229215234 230	0.0000000048632 59
0.0035053755644 43	0.0034624356021 390	0.0000211115270 56	0.0000000003649 751	0.0000231805432 670	0.0000000018400 61
0.0033319054565 28	0.0032901244858 360	0.0000225081740 30	0.0000000002722 590	0.0000221722893 661	0.0000000125924 64
0.0031556406326 77	0.0031149371438 241	0.0000238782260 28	0.0000000001772 270	0.0000199987762 650	0.0000000140542 30
0.0029776682878 05	0.0029379374425 060	0.0000252152364 41	0.0000000000807 140	0.0000182278159 460	0.0000000029966 93
0.0027989861667 52	0.0027601037595 221	0.0000265133832 11	0.0000000000164 899	0.0000179091559 860	0.0000000059040 81
0.0026205027533 20	0.0025823291178 950	0.0000277674861 13	0.0000000001136 430	0.0000171893582 710	0.0000000029526 85
0.0024430385843 79	0.0024054223636 510	0.0000289730120 72	0.0000000002100 630	0.0000153039312 659	0.0000000226332 36
0.0022673285028 54	0.0022301102169 940	0.0000301260699 45	0.0000000003051 249	0.0000127094011 599	0.0000000194941 13
0.0020940246750 19	0.0020570400377 851	0.0000312233963 49	0.0000000003982 690	0.0000110156358 230	0.0000000008581 93
0.0019237002111 25	0.0018867831585 649	0.0000322623339 63	0.0000000004889 971	0.0000104899302 220	0.0000000046905 46
0.0017568532434 99	0.0017198386518 780	0.0000332408037 81	0.0000000005768 770	0.0000091764154 720	0.0000000076095 28
0.0015939113318 40	0.0015566374127 760	0.0000341572726 08	0.0000000006615 350	0.0000068759291 600	0.0000000261088 59
0.0014352360812 47	0.0013975464514 480	0.0000350107170 69	0.0000000007426 599	0.0000042720193 170	0.0000000187748 71
0.0012811278739 08	0.0012428733047 570	0.0000358005851 93	0.0000000008199 960	0.0000028662614 669	0.0000000021250 88
0.0011318306300 83	0.0010928704886 281	0.0000365267565 76	0.0000000008933 420	0.0000024957201 269	0.0000000022052 80
0.0009875365278 59	0.0009477399255 921	0.0000371895019 65	0.0000000009625 469	0.0000015073776 860	0.0000000047746 26
0.0008483906238 05	0.0008076372932 450	0.0000377894429 69	0.0000000010275 041	0.0000002128274 430	0.0000000054082 05
0.0007144953282 20	0.0006726762497 140	0.0000383275125 22	0.0000000010881 520	0.0000004941330 740	0.0000000026622 08
Average absolute error= 0.0030436708366 63	Average absolute error= 0.0030092540416 316	Average absolute error= 0.0000142183758 13	Average absolute error= 0.0000000056004 3	Average absolute error= 0.0000119571268 96	Average absolute error= 0.0000000183729 46

Table 6 shows that the computational data of accuracy of various techniques at different grid point, and also shows that the average absolute errors of the methods. It is observed that the error data are found in the following order between [0, 1] when h=0.02

$$\mathbf{0.003043670836663} > 0.0030092540416316 > 0.000014218375813 > 0.000011957126896 > 0.000000018372946 > 0.00000000560043$$

Thus the orders of the accuracy of methods are near to exact solution in the following manner as.

$$Runge\ Kutta\ method > Ode45 > Ode23 > Eulers\ modified\ method > Euler\ backward\ methos > Euler\ method$$

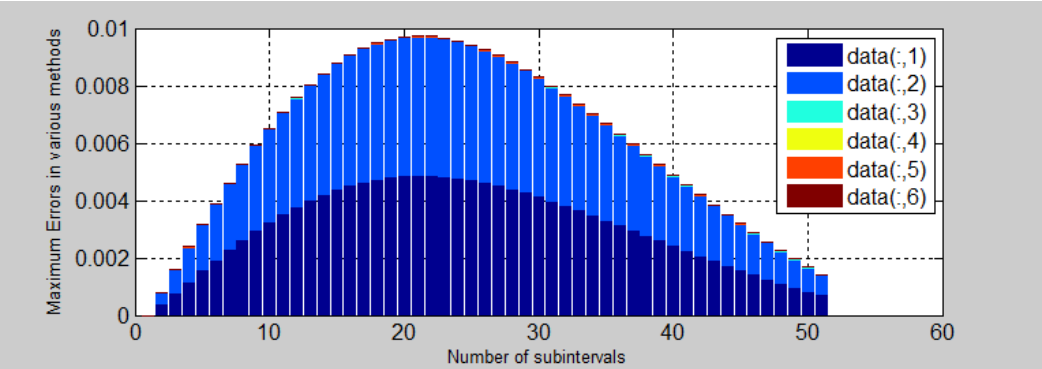

Figure 2. Comparison of errors among the techniques versus x.

From the Fig. 2 we see that if we connect the maximum error point of some methods then we get a curve. This curve shows that the graph of downward parabola.

***Executing time*.**

The execution time of a given task is defined as the time spent by the system executing that task, including the time spent, executing run time or system services on its behalf.

Table 7. Execution time and functions evaluations

Methods	When h = 0.1 (cputime)	Function Evaluations	When h = 0.02 (cputime)	Function Evaluations
Euler method	31.855404	11	32.554400	50
Euler backward method	26.800971	22	27.331375	100
Euler modified method	27.924178	33	28.3609818	150
Runge Kutta method	47.221502	44	49.2807158	200
ode-23 method	30.045793	31	32.807010	34
ODE 45 method	30.560596	61	33.384214	67

From Table 7 we see that if we increase the number of intervals, function evaluation increases but the cputime is increases slowly. Hence based on our results and discussion including accuracy and efficiency, Ode23 converges quickly but the accuracy is not good. So we now conclude that ode45 is most efficient overall. Hence the accuracy of initial value problem by various techniquess has been found in the following order.

$$Ode45 \geq Runge\ Kutta\ method > Ode23 > Eulers\ modified\ method > Euler\ backward\ methos > Euler\ method$$

It is evident that the ode45 is better approximation near to exact solution for solving the initial value problem as compared to other methods.

5. CONCLUSION

The Euler methods are simple methods of solving first-order ODE, particularly suitable for quick programming because of their great simplicity, although their accuracy is not high. First, the modified Euler method is more accurate than the Euler method. Second, it is more stable. The above plots show the results obtained from different algorithms. Consequently, we can see that ode45 and Runge-Kutta technique is even more accurate at

large step size (h = 0.1) than Euler techniques at small step size (h = 0:02). Now ode 45 is more accurate than RK4 when h=0.1and less accurate when h=0.02. But we see that ode45 is more effective with all parameters such as running time factor and function evaluations.One can easily adopt these MATLAB codes as needed for a different type of problem. In the examples, the accuracy of the numerical results could be ascertained directly by a comparison with the solution obtained analytically. Of course, usually the analytical solution is not always available to compare. Hence from the numerical computations we see that the accuracy and efficiency of Ode45 is most effective.

REFRENCES

[1] Jain, M.K., Iyenger, S.R.K., Jain, R.K., *Numerical methods for scientific and engineering computation*, New Age International Publishers, India. 2010.
[2] The Math Works Inc., *MATLAB: 7.8.*, The Math Works Inc., 2009a.
[3] Edwards, C.H., Penny, D.E., *Differential Equations and Boundary Value Problems: Computing and Modeling*, Prentice Hall, 2000.
[4] Boyce, W.E., Di Prima, R.C., *Elementary Differential Equations and Boundary Value Problems.* John Wiley and Sons, 2001.
[5] Coombes, K.R., Hunt, B.R., Lipsman, R.L., Osborn, J.E., Stuck G.J., *Differential Equations with MATLAB*, John Wiley and Sons, 2000.
[6] Van Loan, C.F., *Introduction to Scientific Computing.* Prentice Hall, 1997.
[7] Nakamura, S., *Numerical Analysis with MATLAB*, Prentice Hall, 2002.
[8] Moler, C.B., *Numerical Computing with MATLAB*, Siam, 2011.
[9] Chapra, S.C., *Applied Numerical methods with matlab for engineers and scientist*, 2012.
[10] Dormand, J.R., Prince, P.J., *J.Comp.Appl. Math*, **6**, 19, 1980.
[11] Miron-Alexe, V., Vasile, I., *Journal of Science and Arts*, **4**(41), 853, 2017.
[12] Kamal, S., Samia, B., *Journal of Science and Arts*, **2**(39), 257, 2017.
[13] Shampine, L.F., *Numerical Solution of Ordinary Equations*, Chapman and Hall, 1994.