EE1103: Numerical Methods
Programming Assignment # 6

Amizhthni PRK, EE21B015
Collaborators:
Ankita Harsha Murthy, EE21B020
Anirudh B S, EE21B019

March 29, 2022

Contents

List of Figures

1 Problem 1

Consider the following differential equation

$$\frac{dy}{dt} = yt^3 - 1.5y$$

over the interval $t = 0$ to 2. It is given that $y(0) = 1$. Solve the following parts:

(a) Write the analytical solution for the differential equation.

(b) Numerically solve the same by applying the following methods. Use step size $h = 0.1$, 0.25 and 0.5.

- Euler's Method
- Heun's Predictor-Corrector Method
- Midpoint Method
- Fourth Order Runge Kutta Method

For each value of h, plot the results from across ALL methods along with the true value, on the same graph.

(c) Report the run time for each method.

(d) For each value of h, tabulate the absolute error of $y(2)$ for the different methods with respect to the true value. Infer the quality of the methods in terms of the error obtained. (Table should have h in the rows and the error from different methods as column entries; you can choose to use more values of h than estimated above, to prove your point)

1.1 Approach

In this problem, we first defined a function that calculates the value of $f(t, y) = dy/dt$ as well as a function that returns the true value of y for any particular value of t, which we obtained analytically. Then, we created 4 functions for each of the 4 methods mentioned- the Euler's, Heun's, Midpoint and Runge Kutta methods. We entered a variable step size h and estimated $y(t)$ over the interval $t = 0$ to 2. We hence solved the differential equation. This was done using simple `while` loops which terminated when t reached a value of 2. In each method, we calculated the run time as well as the absolute error in $y(2)$. Finally, the results were plotted.

1.2 Algorithm

The pseudocode for the problem is presented below.

Algorithm 1: Euler's method

$begin \leftarrow t_{initial}$
$h \leftarrow stepsize$
$y \leftarrow 1, t \leftarrow 0$
$f(y,t) \leftarrow y * t^3 - 1.5 * y$
for $t = 0$ *to* $2 + h$ **do**
  $answer \leftarrow e^{(\frac{t^4}{4}} - 1.5 * t)$
  $e \leftarrow y - answer$
  $y \leftarrow y + f(y,t) * h$
  $t \leftarrow t + h$
end
$end \leftarrow t_{final}$
$t_{spent} = end - begin$

Algorithm 2: Heun's method

$begin \leftarrow t_{initial}$
$h \leftarrow stepsize$
$y \leftarrow 1, t \leftarrow 0$
$f(y,t) \leftarrow y * t^3 - 1.5 * y$
for $t = 0$ *to* $2 + h$ **do**
  $answer \leftarrow e^{(\frac{t^4}{4}} - 1.5 * t)$
  $e \leftarrow y - answer$
  $f_1 \leftarrow f(y,t)$
  $y_{temp} \leftarrow y + f_1 * h$
  $f_2 \leftarrow f(y_{temp}, t + h)$
  $y \leftarrow y + \frac{h(f_1+f_2)}{2}$
  $t \leftarrow t + h$
end
$end \leftarrow t_{final}$
$t_{spent} = end - begin$

Algorithm 3: Midpoint method

$begin \leftarrow t_{initial}$
$h \leftarrow stepsize$
$y \leftarrow 1, t \leftarrow 0$
$f(y,t) \leftarrow y * t^3 - 1.5 * y$
for $t = 0$ *to* $2 + h$ **do**
  $answer \leftarrow e^{(\frac{t^4}{4}} - 1.5 * t)$
  $e \leftarrow y - answer$
  $f_1 \leftarrow f(y,t)$
  $y_{temp} \leftarrow y + \frac{f_1 * h}{2}$
  $f_2 \leftarrow f(y_{temp}, t + \frac{h}{2})$
  $y \leftarrow y + f_2 * h$
  $t \leftarrow t + h$
end
$end \leftarrow t_{final}$
$t_{spent} = end - begin$

Algorithm 4: Runge-Kutta method

$begin \leftarrow t_{initial}$
$h \leftarrow stepsize$
$y \leftarrow 1, t \leftarrow 0$
$f(y,t) \leftarrow y * t^3 - 1.5 * y$
for $t = 0$ *to* $2 + h$ **do**
  $answer \leftarrow e^{(\frac{t^4}{4}} - 1.5 * t)$
  $e \leftarrow y - answer$
  $k_1 \leftarrow f(y,t)$
  $k_2 \leftarrow f(y + \frac{k_1 * h}{2}, t + \frac{h}{2})$
  $k_3 \leftarrow f(y + \frac{k_2 * h}{2}, t + \frac{h}{2})$
  $k_4 \leftarrow f(y + k_3 * h, t + h$
  $y \leftarrow y + \frac{(k_1 + 2 * k_2 + 2 * k_3 + k_4) * h}{6})$
  $t \leftarrow t + h$
end
$end \leftarrow t_{final}$
$t_{spent} = end - begin$

1.3 Results

- The **analytical solution** for the differential equation, upon separating y and t and integrating, is
  $\mathbf{y}=e^{\frac{t^4}{4}-1.5t}$.

- The values of $y(t)$ as well as the true values for step sizes $h = 0.1, 0.25$ and 0.5 are plotted in figure 1, 2 and 3 respectively.

- The values of $y(t)$ as well as the true values for step size
  (i) $h = 0.1$ are given under table 4.
  (ii) $h = 0.25$ are given under table 5.
  (iii) $h = 0.5$ are given under table 6.

- The absolute error in $y(2)$ for the different methods, with respect to the true value, is tabulated in table 2. The same is plotted for arbitrarily small h values to analyse the relation between h and error under table 3.

- The run time for each method is tabulated in table 1Note that separate codes were written to obtain the run time.

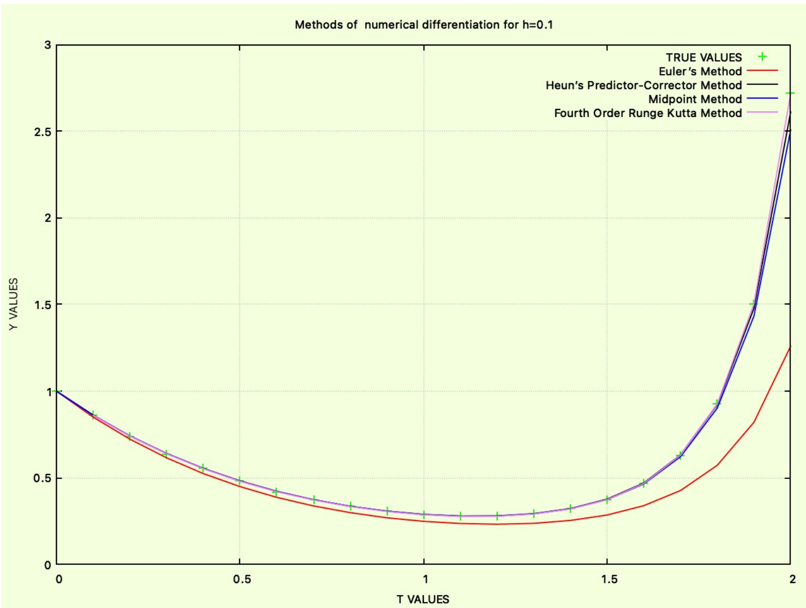

Figure 1: y(t) for $t = 0$ to 2 with a step size $h = 0.1$

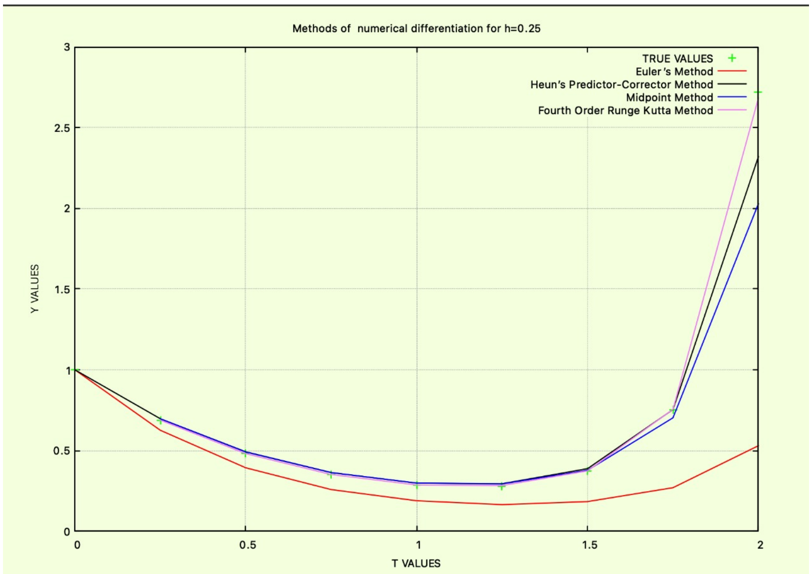

Figure 2: y(t) for $t = 0$ to 2 with a step size $h = 0.25$

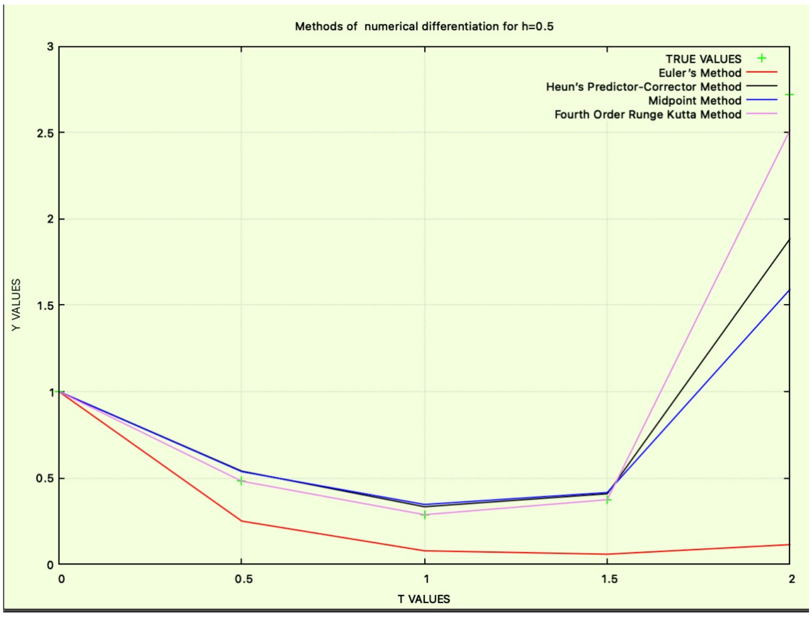

Figure 3: y(t) for $t = 0$ to 2 with a step size $h = 0.5$

Table 1: Runtime for various methods of solving for y

Method	**h=0.1**	**h=0.25**	**h=0.5**
Euler	0.000005	0.000005	0.000005
Heun	0.000004	0.000003	0.000004
Midpoint	0.000004	0.000004	0.000006
Runge Kutta	0.000004	0.000005	0.000005

Table 2: Absolute Error in $y(2)$

Method	**h=0.1**	**h=0.25**	**h=0.5**
Euler	1.458783	2.188587	2.604756
Heun	0.107443	0.397847	0.834097
Midpoint	0.211234	0.689259	1.126480
Runge Kutta	0.001732	0.03547	0.205209

Table 3: Absolute Error in $y(2)$ for smaller values of h

Method	$h = 10^{-6}$	**h=0.0001**	**h=0.001**	**h=0.01**	**h=0.7**
Euler	0.000025533042	0.002551	0.025347	0.237896	2.727786828459
Heun	0.000000000811	0.000000	0.000016	0.001502	2.382593395823
Midpoint	0.000000000828	0.000000	0.000033	0.003184	2.276292796170
Runge Kutta	0.000000000795	0.000000	0.000000	0.000000	2.397053780273

Table 4: y(t) as obtained in all methods and true method for h=0.1

t	**y true**	**Euler**	**Heun**	**Midpoint**	**Runge Kutta**
0.000000	1.000000	1.000000	1.000000	1.000000	1.00000
0.100000	0.860729	0.850000	0.861293	0.861262	0.860730
0.200000	0.741115	0.722585	0.742118	0.742024	0.741116
0.300000	0.638921	0.614775	0.640254	0.640097	0.638922
0.400000	0.552335	0.524219	0.553900	0.553696	0.552337
0.500000	0.479805	0.448941	0.481518	0.481288	0.479807
0.600000	0.419958	0.387212	0.421751	0.421515	0.419960
0.700000	0.371586	0.337494	0.373409	0.373180	0.371587
0.800000	0.333671	0.298446	0.335495	0.335274	0.333672
0.900000	0.305448	0.268959	0.307266	0.307040	0.305449
1.000000	0.286505	0.248222	0.288331	0.288069	0.286506
1.100000	0.276934	0.235811	0.278809	0.278453	0.276935
1.200000	0.277593	0.231826	0.279577	0.279030	0.277594
1.300000	0.290551	0.237112	0.292730	0.291817	0.290552
1.400000	0.319947	0.253638	0.322408	0.320819	0.319948
1.500000	0.373673	0.285191	0.376448	0.373592	0.373673
1.600000	0.466919	0.338664	0.469765	0.464464	0.466917
1.700000	0.630038	0.426582	0.631717	0.621476	0.630023
1.800000	0.927187	0.572174	0.923049	0.902257	0.927110
1.900000	1.503845	0.820040	1.477456	1.432618	1.503477
2.000000	2.718282	1.259499	2.610838	2.507047	2.716550

Table 5: y(t) as obtained in all methods and true method for h=0.25

t	y true	Euler	Heun	Midpoint	Runge Kutta
0.000000	1.000000	1.000000	1.000000	1.000000	1.000000
0.250000	0.687961	0.625000	0.696533	0.695709	0.688016
0.500000	0.479805	0.393066	0.492003	0.490696	0.479871
0.750000	0.3513767	0.257950	0.363927	0.363114	0.351429
1.000000	0.286505	0.188424	0.298268	0.297916	0.286543
1.250000	0.282339	0.164871	0.294408	0.292597	0.282374
1.500000	0.373673	0.183548	0.387902	0.377589	0.373684
1.750000	0.755577	0.269586	0.753668	0.702802	0.754580
2.000000	2.718282	0.529695	2.320435	2.029023	2.682811

Table 6: y(t) as obtained in all methods and true method for h=0.5

t	y true	Euler	Heun	Midpoint	Runge Kutta
0.000000	1.000000	1.000000	1.000000	1.000000	1.000000
0.500000	0.479805	0.250000	0.539062	0.536133	0.481096
1.000000	0.286505	0.078125	0.332703	0.346471	0.286932
1.500000	0.373673	0.058594	0.408081	0.415156	0.373752
2.000000	2.718282	0.113525	1.884185	1.591802	2.513072

1.4 Inferences

We deduce the following inferences in this experiment:

- Upon analysing the absolute errors in different methods and by visually comparing through graphs, the quality of the methods from highest to least is:
  Runge Kutta > Heun's Predictor Corrector Method > Midpoint > Euler's method.

- A direct observation is that the number of intermediate slopes considered is directly proportional to the accuracy of the concerned method. For example, the Runge Kutta fourth order method is rather complex and encompasses an analysis of 4 slopes to approximate the point desired. Similarly, the Heun's method comprises of analysing two slopes and hence it is the second best method

- The Runge Kutta method has an error of the order $\mathcal{O}(h^4)$ while Euler's method has just $\mathcal{O}(h)$.The Euler method is only **first order convergent**,and this is unacceptably poor, and requires a too small step size to achieve some serious accuracy.

- Heun's method, by contrast, is second order convergent, with an error$\mathcal{O}(h^2)$. Thus, the efficiency is dramatically improved. For example, for 100 times more accuracy, Euler method takes 100 times more smaller value of h while Heun's method takes up a value of h which is just ten times smaller to attain the same level of accuracy.

- From table 2, the error is the highest for Euler's method followed by Midpoint method. The difference between these two methods is stark. Following the Midpoint method, we have the Heun and the fourth-order Runge Kutta method.

- The Euler method is the most inefficient method of all since it has the highest absolute error value and its graph does not capture the intricate sloping of the graph to the degree of other alternative methods.

- On comparing errors for varying values of h, we notice that the **error increases with increasing sizes of intervals h**. Further, the difference between the errors as calculated from different methods increases with an increase in h. Thus, **smaller values of h are preferred.**

- Fourth order Runge Kutta method has the highest level of accuracy . This is evident from the fact that the graph lines of Runge Kutta and that of the true plot coincide completely. For better representation, we had to plot the true value points discretely (in green) instead of lines for the true values, as not doing so would have led to the Runge-Kutta curve and the true value curve being indistinguishable to the eye.

- The run time tabulated under table 1 includes the printing of all the values as ordered triplets. We get results in nanoseconds if printing the values is neglected. In our case, it is to the order of 10^{-5}. In most cases, the Euler's method takes the longest execution time. Also we do get inconsistent errors, for example for a greater value of h we get a higher execution time which is counter intuitive, we thus regard this to arise due to computational limitations and the state of the processor at the required moment.

- The results become inconsistent or meaningless for higher values of h.

1.5 Code

The code used for obtaining solutions to the Differential Equation is given below in listing 1 for Euler method,listing 2 for Heun's predictor corrector method,listing 3 for Midpoint method,listing 4 for 4^{th} order Runge Kutta method and listing 5 for plotting all methods along with the true value.
I have added comments to the Euler code for changing it to the Taylor's series example, which gives greater accuracy.
The code 5 has been commented out so as to facilitate the plotting for each value of h

```c
#include <stdio.h> //initialising libraries
#include <math.h> //initialising libraries
#include <time.h> //initialising libraries

/*double fact(double n) //extra factors for exact euler method
{
    int factorial=1;
    for (int i=1; i<=n;i++){
        factorial=factorial*i;
    }
    return factorial;
}*/
double f(double x, double y)
{
    return  y*pow(x,3)-1.5*y;  //returns values of function
}
/*
double f1(double x, double y) //extra factors for exact euler method
{
```

```
    return  3*y*pow(x,2);
}
double f2(double x, double y) //extra factors for exact euler method
{
    return  y*6*x;
}
double f3(double x, double y) //extra factors for exact euler method
{
    return  6*y;
}
*/
int main()
{
    double h;
    printf("BY IMPLICIT EULER'S METHOD\n");
    printf("Enter step size value h: "); //input of h value
    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0; //x low is the value analogius to x_i
    double y_low=1,y_high; //y low is the value analogius to y_i and y high is y_i+1
    printf("X\t\tY\n");
    printf("\n");
    printf("%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        y_high=y_low+h*f(x_low,y_low);// for higher orders +f1(x_low,y_low)*h*h/fact(2)+f2(x_low,y_low)*h*h*h/fact(3)+f3(x_low,y_low)*h*h*
        //this statement increments values of y to push it to the next highest element
        printf("%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high; // recursive condition
        x_low=x_low+h; //recursive condition

    }

    //to calculate the time of executIon
    clock_t begin = clock();//to calculate the start time
    clock_t end = clock(); //to calculate the end time
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC; //final answers expressed in seconds
    printf("EXECUTION TIME: %0.12lf",time_spent); //answer output for 12 digit precision
    return 0;
}
```

Listing 1: Code used for solving the Differential Equation using Euler's method

```
#include <stdio.h>//initialising libraries
#include <math.h>//initialising libraries
```

```
#include <time.h>//initialising libraries

double f(double x, double y)
{
    return  y*pow(x,3)-1.5*y;//returns values of function
}

int main()
{
    double h;
    printf("BY HEUN'S METHOD\n");
    printf("Enter step size value h: ");//input of h value
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0; //x low is the value analogius to x_i
    double y_low=1,y_high,k1,k2;//y low is the value analogius to y_i and y high is y_i+1,K1 IS THE INTERMEDIATE VALUE1,K2 IS THE INTERMEDIATE VALUE2
    printf("X\t\tY\n");
    printf("\n");
    printf("%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        k1=f(x_low,y_low);//INTERMEDIATE VALUE1
        k2=f(x_low+h,y_low+k1*h);//INTERMEDIATE VALUE2
        y_high=y_low+h*0.5*(k1+k2); // for higher orders
        printf("%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;// recursive condition
        x_low=x_low+h;// recursive condition


    }
    //to calculate the time of execution
    clock_t begin = clock();//to calculate the start time
    clock_t end = clock(); //to calculate the end time
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;//final answers expressed in seconds
    printf("EXECUTION TIME: %0.12lf",time_spent);//answer output for 12 digit
    return 0;
}
```

Listing 2: Code used for solving the Differential Equation using Heun's methd

```
#include <stdio.h>//initialising libraries
#include <math.h>//initialising libraries
#include <time.h>//initialising libraries


double f(double x, double y)
{
```

```
    return  y*pow(x,3)-1.5*y;//returns values of function
}



int main()
{
    double h;
    printf("BY MIDPOINT METHOD\n");
    printf("Enter step size value h: ");//input of h value
    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0;//x low is the value analogius to x_i
    double y_low=1,y_high,y_mid;//y low is the value analogius to y_i and y high is y_i+! and y mid is the midpoint value of y
    printf("X\t\tY\n");
    printf("\n");
    printf("%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){

        y_mid=y_low+h*0.5*f(x_low,y_low);//INTERMEDIATE VALUE1
        y_high=y_low+h*(f(x_low+h*0.5,y_mid)); // for higher orders
        printf("%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;// recursive condition
        x_low=x_low+h;// recursive condition

    }
    //to calculate the time of execution
    clock_t begin = clock();//to calculate the start time
    clock_t end = clock(); //to calculate the end time
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;//final answers expressed in seconds
    printf("EXECUTION TIME: %0.12lf",time_spent);//answer output for 12 digit
    return 0;
}
```

Listing 3: Code used for solving the Differential Equation using Midpoint method

```
#include <stdio.h>//initialising libraries
#include <math.h>//initialising libraries
#include <time.h>//initialising libraries


double f(double x, double y)
{
    return  y*pow(x,3)-1.5*y;//returns values of function
}

```

```

int main()
{
    double h;
    printf("BY RUNGE KUTTA 4TH ORDER METHOD\n");
    printf("Enter step size value h: ");//input of h value
    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0;//x low is the value analogius to x_i
    double y_low=1,y_high,k1,k2,k3,k4;//y low is the value analogius to y_i
    ↪ and y high is y_i+1,K1 IS THE INTERMEDIATE VALUE1,K2 IS THE
    ↪ INTERMEDIATE VALUE2,K3 IS THE INTERMEDIATE VALUE3,K4 IS THE
    ↪ INTERMEDIATE VALUE4.
    printf("X\t\tY\n");
    printf("\n");
    printf("%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        k1=f(x_low,y_low);//INTERMEDIATE VALUE1
        k2=f(x_low+h/2,y_low+h*k1/2);//INTERMEDIATE VALUE2
        k3=f(x_low+h/2,y_low+k2/2*h);//INTERMEDIATE VALUE3
        k4=f(x_low+h,y_low+k3*h);//INTERMEDIATE VALUE4
        y_high=y_low+h*(k1/6+k2/3+k3/3+k4/6);//higher orders
        printf("%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;// recursive condition
        x_low=x_low+h;// recursive condition

    }
    //to calculate the time of execution
    clock_t begin = clock();//to calculate the start time
    clock_t end = clock();//to calculate the end time
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;//final
    ↪ answer in seconds
    printf("EXECUTION TIME: %0.12lf",time_spent);//answer output for 12
    ↪ digits
    return 0;
}
```

Listing 4: Code used for solving the Differential Equation solving using Runge Kutta

```
#include <stdio.h>//inititalising libraries
#include <math.h>//inititalising libraries
#include <time.h>//inititalising libraries

double tru(double x)
{
    return pow(M_E,pow(x,4)/4-1.5*x);//true value plotting.
}
double f(double x, double y)
{
```

```
    return  y*pow(x,3)-1.5*y;
}
int main()
{   //we run the codes separately
    double h=0.1;

    printf("true values");
    double x_low=0;
    double y_low=1,y_high;
    FILE *fptr2;
    fptr2 = fopen("trueval.txt", "a");
    for (int i=1;i<=2/h+1;i++){

        fprintf(fptr2,"%lf \t%lf\n",x_low,tru(x_low));
        x_low=x_low+h;
    }
    fclose(fptr2);
    x_low=0;
    y_low=1;
    printf("BY IMPLICIT EULER'S METHOD\n");
    FILE *fptr;
    fptr = fopen("euler.txt", "a");
    fprintf(fptr,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        y_high=y_low+h*f(x_low,y_low);

        fprintf(fptr,"%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;
        x_low=x_low+h;

    }
    fclose(fptr);
    printf("error in y(2) value by euler is %lf",fabs(tru(2.0)-y_high));

    printf("BY HEUN'S METHOD\n");
    printf("\n");
    double x_low1=0;
    double y_low1=1,y_high1,k1,k2;
    FILE *fptr3;
    fptr3 = fopen("heun.txt", "a");
    fprintf(fptr3,"%lf \t%lf\n",x_low1,y_low1);
    for (int i=1;i<=2/h;i++){
        k1=f(x_low1,y_low1);
        k2=f(x_low1+h,y_low1+k1*h);
        y_high1=y_low1+h*0.5*(k1+k2);
        fprintf(fptr3,"%lf \t%lf\n",x_low1+h,y_high1);
        y_low1=y_high1;
        x_low1=x_low1+h;}
    fclose(fptr3);
    printf("error in y(2) value by heun is %lf",fabs(tru(2.0)-y_high1));
```

```
printf("BY MIDPOINT METHOD\n");
double x_low2=0;
double y_low2=1,y_high2,y_mid;
FILE *fptr4;
fptr4 = fopen("mp.txt", "a");
//fprintf(fptr4,"%lf \t%lf\n",x_low,y_low);
for (int i=1;i<=2/h;i++){

    y_mid=y_low2+h*0.5*f(x_low2,y_low2);
    y_high2=y_low2+h*(f(x_low2+h*0.5,y_mid));
    fprintf(fptr4,"%lf \t%lf\n",x_low2+h,y_high2);
    y_low2=y_high2;
    x_low2=x_low2+h;}
fclose(fptr4);
printf("error in y(2) value by midpoint is %lf",fabs(tru(2.0)-y_high2));
 printf("BY RUNGE KUTTA 4TH ORDER METHOD\n");
 double x_low3=0;
double y_low3=1,y_high3,k3,k4;
FILE *fptr5;
fptr5= fopen("rk4.txt", "a");
//fprintf(fptr5,"%lf \t%lf\n",x_low,y_low);
for (int i=1;i<=2/h;i++){
k1=f(x_low3,y_low3);
k2=f(x_low3+h/2,y_low3+h*k1/2);
k3=f(x_low3+h/2,y_low3+k2/2*h);
k4=f(x_low3+h,y_low3+k3*h);
 y_high3=y_low3+h*(k1/6+k2/3+k3/3+k4/6);
fprintf(fptr5,"%lf \t%lf\n",x_low3+h,y_high3);
y_low3=y_high3;
x_low3=x_low3+h;}
fclose(fptr5);
printf("error in y(2) value by runge kutta is %lf",fabs(tru(2.0)-y_high3));
FILE *pipek = popen("gnuplot --persist", "w");
fprintf(pipek, "set title \"Methods of  numerical differentiation for h=0.1\"\n");//GIVING TITLE TO THE GNUPLOT
fprintf(pipek, "set xlabel \"T VALUES \"\n");//GIVING X AXIS TITLE TO THE GNUPLOT
fprintf(pipek, "set ylabel \"Y VALUES\"\n");//GIVING Y AXIS TITLE TO THE GNUPLOT
fprintf(pipek, "set xrange [0:2]\n");//SETTING THE RANGE OF VALUES FOR X AXIS
fprintf(pipek, "set yrange [0:3]\n");//SETTING THE RANGE OF VALUES FOR Y AXIS
fprintf(pipek, "set grid\n");//TO MAKE THE GRID LINES VISIBLE FOR BETTER CLARITY
```

```
    fprintf(pipek, "plot 'trueval.txt' title 'TRUE VALUES' lt rgb \"green\" , 'euler.txt' with line title 'Euler’s Method' lt rgb \"red\" ,'heun.txt' with line title 'Heun’s Predictor-Corrector Method' lt rgb \"black\" , 'mp.txt' with line title 'Midpoint Method' lt rgb \"blue\", 'rk4.txt'  with line title 'Fourth Order Runge Kutta Method' lt rgb \"violet\" \n");
    /*double h;
    h=0.5;

    printf("true values");
    double x_low=0;
    double y_low=1,y_high;
    FILE *fptr2;
    fptr2 = fopen("trueval.txt", "a");
    for (int i=1;i<=2/h+1;i++){

        fprintf(fptr2,"%lf \t%lf\n",x_low,tru(x_low));
        x_low=x_low+h;
    }
    fclose(fptr2);

     double x_low=0;
     double y_low=1,y_high;
    printf("BY IMPLICIT EULER'S METHOD\n");
    FILE *fptr;
    fptr = fopen("euler.txt", "a");
    fprintf(fptr,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        y_high=y_low+h*f(x_low,y_low);
        fprintf(fptr,"%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;
        x_low=x_low+h;

    }
    fclose(fptr);
     printf("error in y(2) value by euler is %lf",fabs(tru(2.0)-y_high3));
    printf("BY HEUN'S METHOD\n");
    printf("\n");
    double x_low1=0;
    double y_low1=1,y_high1,k1,k2;
    FILE *fptr3;
    fptr3 = fopen("heun.txt", "a");
    fprintf(fptr3,"%lf \t%lf\n",x_low1,y_low1);
    for (int i=1;i<=2/h;i++){
        k1=f(x_low1,y_low1);
        k2=f(x_low1+h,y_low1+k1*h);
        y_high1=y_low1+h*0.5*(k1+k2);
        fprintf(fptr3,"%lf \t%lf\n",x_low1+h,y_high1);
        y_low1=y_high1;
        x_low1=x_low1+h;}
```

```
    fclose(fptr3);
     printf("error in y(2) value by heun is %lf",fabs(tru(2.0)-y_high3));
    printf("BY MIDPOINT METHOD\n");
    double x_low2=0;
    double y_low2=1,y_high2,y_mid;
    FILE *fptr4;
    fptr4 = fopen("mp.txt", "a");
   // fprintf(fptr4,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){

        y_mid=y_low2+h*0.5*f(x_low2,y_low2);
        y_high2=y_low2+h*(f(x_low2+h*0.5,y_mid));
        fprintf(fptr4,"%lf \t%lf\n",x_low2+h,y_high2);
        y_low2=y_high2;
        x_low2=x_low2+h;}
    fclose(fptr4);
     printf("error in y(2) value by midpoint is %lf",fabs(tru(2.0)-y_high3));
     printf("BY RUNGE KUTTA 4TH ORDER METHOD\n");
     double x_low3=0;
    double y_low3=1,y_high3,k3,k4;
    FILE *fptr5;
    fptr5= fopen("rk4.txt", "a");
    //fprintf(fptr5,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
    k1=f(x_low3,y_low3);
    k2=f(x_low3+h/2,y_low3+h*k1/2);
    k3=f(x_low3+h/2,y_low3+k2/2*h);
    k4=f(x_low3+h,y_low3+k3*h);
     y_high3=y_low3+h*(k1/6+k2/3+k3/3+k4/6);
    fprintf(fptr5,"%lf \t%lf\n",x_low3+h,y_high3);
    y_low3=y_high3;
    x_low3=x_low3+h;}
     printf("error in y(2) value by runge kutta is %lf",fabs(tru(2.0)-y_high3));
    FILE *pipek = popen("gnuplot --persist", "w");
    fprintf(pipek, "set title \"Methods of  numerical differentiation for h=0.5\"\n");//GIVING TITLE TO THE GNUPLOT
    fprintf(pipek, "set xlabel \"T VALUES \"\n");//GIVING X AXIS TITLE TO THE GNUPLOT
    fprintf(pipek, "set ylabel \"Y VALUES\"\n");//GIVING Y AXIS TITLE TO THE GNUPLOT
    fprintf(pipek, "set xrange [0:2]\n");//SETTING THE RANGE OF VALUES FOR X AXIS
    fprintf(pipek, "set yrange [0:3]\n");//SETTING THE RANGE OF VALUES FOR Y AXIS
    fprintf(pipek, "set grid\n");//TO MAKE THE GRID LINES VISIBLE FOR BETTER CLARITY
```

```c
    fprintf(pipek, "plot 'trueval.txt' title 'TRUE VALUES' lt rgb \"green\" , 'euler.txt' with line title 'Euler's Method' lt rgb \"red\" ,'heun.txt' with line title 'Heun's Predictor-Corrector Method' lt rgb \"black\" , 'mp.txt' with line title 'Midpoint Method' lt rgb \"blue\", 'rk4.txt'  with line title 'Fourth Order Runge Kutta Method' lt rgb \"violet\" \n");


    double h;
    h=0.25;

    printf("true values");
    double x_low=0;
    double y_low=1,y_high;
    FILE *fptr2;
    fptr2 = fopen("trueval.txt", "a");
    for (int i=1;i<=2/h+1;i++){

        fprintf(fptr2,"%lf \t%lf\n",x_low,tru(x_low));
        x_low=x_low+h;
    }
    fclose(fptr2);

    printf("BY IMPLICIT EULER'S METHOD\n");
    FILE *fptr;
     double x_low=0;
    double y_low=1,y_high;
    fptr = fopen("euler.txt", "a");
    fprintf(fptr,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        y_high=y_low+h*f(x_low,y_low);
        fprintf(fptr,"%lf \t%lf\n",x_low+h,y_high);
        y_low=y_high;
        x_low=x_low+h;

    }
    fclose(fptr);

   printf("BY HEUN'S METHOD\n");
    printf("\n");
    double x_low1=0;
    double y_low1=1,y_high1,k1,k2;
    FILE *fptr3;
    fptr3 = fopen("heun.txt", "a");
    fprintf(fptr3,"%lf \t%lf\n",x_low1,y_low1);
    for (int i=1;i<=2/h;i++){
        k1=f(x_low1,y_low1);
        k2=f(x_low1+h,y_low1+k1*h);
        y_high1=y_low1+h*0.5*(k1+k2);
        fprintf(fptr3,"%lf \t%lf\n",x_low1+h,y_high1);
```

```
        y_low1=y_high1;
        x_low1=x_low1+h;}
    fclose(fptr3);

    printf("BY MIDPOINT METHOD\n");
    double x_low2=0;
    double y_low2=1,y_high2,y_mid;
    FILE *fptr4;
    fptr4 = fopen("mp.txt", "a");
    //fprintf(fptr4,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){

        y_mid=y_low2+h*0.5*f(x_low2,y_low2);
        y_high2=y_low2+h*(f(x_low2+h*0.5,y_mid));
        fprintf(fptr4,"%lf \t%lf\n",x_low2+h,y_high2);
        y_low2=y_high2;
        x_low2=x_low2+h;}
    fclose(fptr4);
     printf("BY RUNGE KUTTA 4TH ORDER METHOD\n");
     double x_low3=0;
    double y_low3=1,y_high3,k3,k4;
    FILE *fptr5;
    fptr5= fopen("rk4.txt", "a");
    //fprintf(fptr5,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
    k1=f(x_low3,y_low3);
    k2=f(x_low3+h/2,y_low3+h*k1/2);
    k3=f(x_low3+h/2,y_low3+k2/2*h);
    k4=f(x_low3+h,y_low3+k3*h);
     y_high3=y_low3+h*(k1/6+k2/3+k3/3+k4/6);
    fprintf(fptr5,"%lf \t%lf\n",x_low3+h,y_high3);
    y_low3=y_high3;
    x_low3=x_low3+h;}

    FILE *pipek = popen("gnuplot --persist", "w");
    fprintf(pipek, "set title \"Methods of  numerical differentiation for h=0.25\"\n");//GIVING TITLE TO THE GNUPLOT
    fprintf(pipek, "set xlabel \"T VALUES \"\n");//GIVING X AXIS TITLE TO THE GNUPLOT
    fprintf(pipek, "set ylabel \"Y VALUES\"\n");//GIVING Y AXIS TITLE TO THE GNUPLOT
    fprintf(pipek, "set xrange [0:2]\n");//SETTING THE RANGE OF VALUES FOR X AXIS
    fprintf(pipek, "set yrange [0:3]\n");//SETTING THE RANGE OF VALUES FOR Y AXIS
    fprintf(pipek, "set grid\n");//TO MAKE THE GRID LINES VISIBLE FOR BETTER CLARITY
```

```
    fprintf(pipek, "plot 'trueval.txt' title 'TRUE VALUES' lt rgb \"green\" , 'euler.txt' with line title 'Euler's Method' lt rgb \"red\" ,'heun.txt' with line title 'Heun's Predictor-Corrector Method' lt rgb \"black\" , 'mp.txt' with line title 'Midpoint Method' lt rgb \"blue\", 'rk4.txt'  with line title 'Fourth Order Runge Kutta Method' lt rgb \"violet\" \n");

    */

return 0;
}
```

Listing 5: Code used for Plotting all values along with true values

1.6 Contributions

Approach, code, results, inferences, LaTeX, Graphs: Me(AMIZHTHNI)
Complete efforts were put in by me.
Ankita helped me with algorithms.

2 Problem 2

Solve

$$\frac{dy}{dt} = -100,000y + 99,999e^{-t}$$

over the interval from $t = 0$ to 2 using the following methods. Note that $y(0) = 0$.

(a) Explicit Euler method, after estimating the step size required to maintain stability.

(b) Implicit Euler method with a step size of 0.1.

2.1 Approach

In this problem, we were given $f(t, y) = dy/dt$ and we calculated $y(t)$ for different t values between 0 and 2, using the Euler methods.
First, we carried out the explicit Euler's method with a step size $h = 10^{-6}$. Then we performed recursion for the implicit Euler's method with a step size $h = 0.1$.
We estimated the value of $y(2)$ using these methods. Simple `while` loops were used in each function.
Finally, we called the above functions in the main() function.

The formula used in Euler's Explicit method is

$$y_{i+1} = y_i + [-100000y_i + 99999e^{-t_i}] * h$$

The formula used in Euler's Implicit method is

$$y_{i+1} = \frac{y_i + 99999e^{-t_{i+1}} * h}{1 + 100000h}$$

2.2 Algorithm

The pseudocode snippets for the Euler's Implicit and Explicit Methods are given below.

Algorithm 5: Explicit Euler Method

$h \leftarrow 10^{-6}$
$y \leftarrow 0$,$t \leftarrow 0$
for $t = 0$ *to* 2 **do**
  $y \leftarrow y + ((-100000 * y + 99999 * e^{-t}) * h)$
  $t \leftarrow t + h$
end
return $y(2)$

Algorithm 6: Implicit Euler Method

$h \leftarrow 0.1$
$y \leftarrow 0$,$t \leftarrow 0$
for $t = 0$ *to* 2 **do**
  $y \leftarrow \frac{99999*h*e^{-t-h}+y}{1+100000*h}$
  $t \leftarrow t + h$
end
return $y(2)$

2.3 Results

- The values of $y(t)$ for different values of t, as obtained via the **Implicit Euler's Method**, are tabulated in table 7. The step size h is taken to be **0.1**.

- The solution for the given differential equation is given by: $-e^{-100000t} + e^{-t}$ (obtained via Wolfram-Alpha).

- We use the above function to calculate the exact true solutions of the given differential equation at all points in the interval $[0, 2]$ with a step size of 0.1 to a precision of 12 digits. This is done to compare the accuracy of both the methods.

Table 7: y(t) as obtained via Implicit Euler's Method

t	y
0.000000	0.000000
0.100000	0.904738
0.200000	0.818731
0.300000	0.740819
0.400000	0.670320
0.500000	0.606531
0.600000	0.548812
0.700000	0.496586
0.800000	0.449329
0.900000	0.406570
1.000000	0.367880
1.100000	0.332871
1.200000	0.301194
1.300000	0.272532
1.400000	0.246597
1.500000	0.223130
1.600000	0.201897
1.700000	0.182684
1.800000	0.165299
1.900000	0.149569
2.000000	0.135335

- We show the true values of $y(t)$ thus generated in . This code was written after obtaining a formula for y via Wolfram-Alpha.

- For the **Explicit Euler's Method**, by the formula, the step value has to be lesser than $\frac{2}{a}$ to ensure stability, where a is the coefficient of the exponential in the differential equation.
  Thus, $h_{max}=\frac{2}{100000}$=0.000002. In my code, I take $h = 10^{-6} = 0.000001$.

- The values of $y(t)$ for different values of t, as obtained via the **Explicit Euler's Method**, are tabulated in table 7. The step size h is taken to be 10^{-6}.

Table 8: y(t) as obtained via Explicit Euler's Method

t	**y**
0.000000	0.000000
0.000001	0.099999000000
0.000002	0.189998000001
0.000003	0.270997000003
0.000004	0.343896000006
0.000005	0.409505000010
0.000006	0.468553000016
0.000007	0.521696100022
0.000008	0.569524790029
0.000009	0.612570511037
0.000010	0.651311559947
0.000011	0.686178403967
0.000012	0.717558463587
0.000013	0.745800417248
0.000014	0.771218075545
0.000015	0.794093868014
.	.
.	.
1.009994	0.364221164888
1.009995	0.364220800667
1.009996	0.364220436446
1.009997	0.364220072226
1.009998	0.364219708006
1.009999	0.364219343786
1.010000	0.364218979567
1.010001	0.364218615348
1.010002	0.364218251130
1.010003	0.364217886912
1.010004	0.364217522694
1.010005	0.364217158477
1.010006	0.364216794260
1.010007	0.364216430043
.	.
.	.
1.999989	0.135336771942
1.999990	0.135336636606
1.999991	0.135336501269
1.999992	0.135336365933
1.999993	0.135336230596
1.999994	0.135336095260
1.999995	0.135335959924
1.999996	0.135335824588
1.999997	0.135335689252
1.999998	0.135335553917
1.999999	0.135335418581
2.000000	0.135335283246

```
Enter step size value h: 0.1
h=0.100000

X                   Y

0.000000            0.000000000000
0.100000            0.904837418036
0.200000            0.818730753078
0.300000            0.740818220682
0.400000            0.670320046036
0.500000            0.606530659713
0.600000            0.548811636094
0.700000            0.496585303791
0.800000            0.449328964117
0.900000            0.406569659741
1.000000            0.367879441171
1.100000            0.332871083698
1.200000            0.301194211912
1.300000            0.272531793034
1.400000            0.246596963942
1.500000            0.223130160148
1.600000            0.201896517995
1.700000            0.182683524053
1.800000            0.165298888222
1.900000            0.149568619223
2.000000            0.135335283237
```

Figure 4: True values of y(t)

2.4 Inferences

- The plots for the methods are given below.
- Both the methods are very accurate and give almost the same answers.

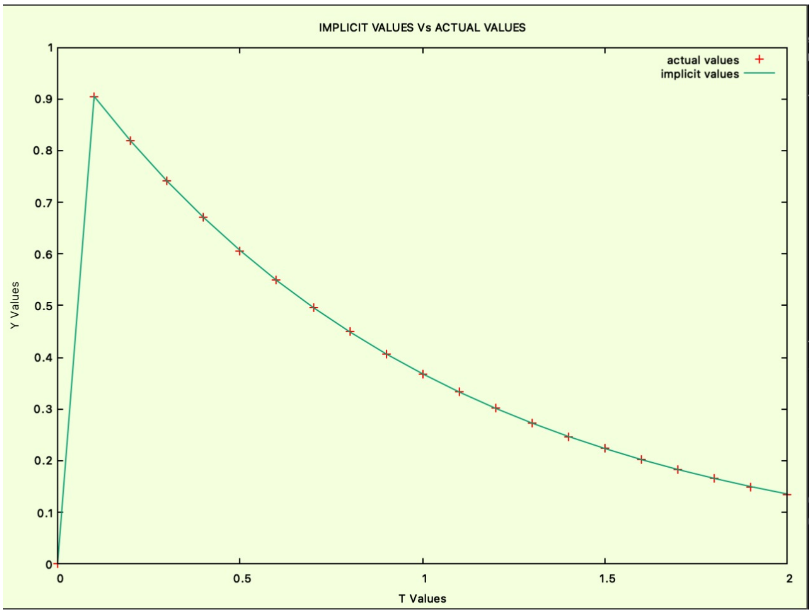

Figure 5: Plot for Implicit Euler's Method

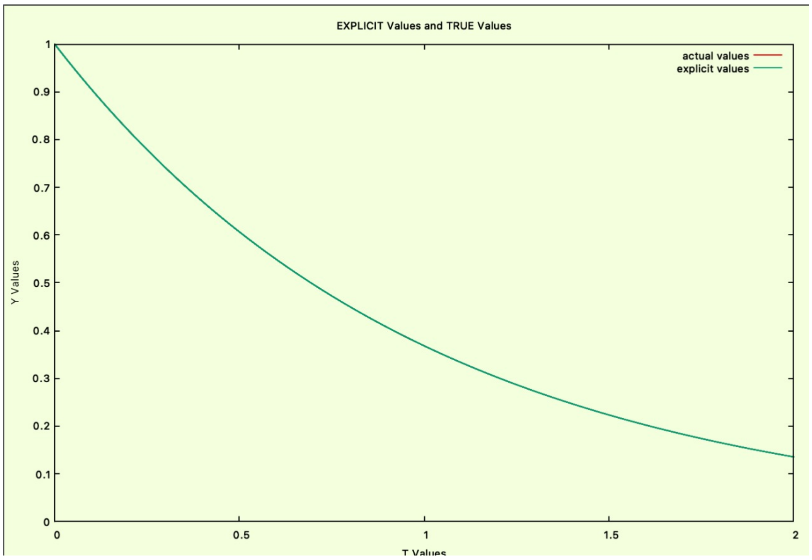

Figure 6: Plot for Explicit Euler's Method

- To understand better, we compare the value of y(2) generated by both the methods and find the absolute error with respect to the true value.
  In the case of **Explicit Euler**, $y(2) = 0.135335283246$ and in case of **Implicit Euler**, we get, $y(2) = 0.135335353218$. However, the true value at $y = 2$ is 0.135335283237.

- The errors in case of Explicit Method and Implicit Method are 9×10^{-12} and 69981×10^{-12} respectively. Evidently, the Explicit Method gives an answer that is exact for four decimal places greater than that of the Implicit method.

- In a nutshell, the Explicit method gave a more accurate value compared to the implicit Euler method. However, this was at the cost of power and extra computation memory and space. This is clearly evident from the time it takes owing to the high increase in the number of sub intervals h.

- However, on the other hand, increasing the number of step values is not the ideal way of computing the exact solution. We must make sure that h is large enough, but is also lesser than the threshold stability value.
- For h> 2×10^{-5} for the explicit method, the solution becomes erratic and unstable.

2.5 Code

Code for **2a** and **2b** are given under Listings 6 and 7 respectively.

```

#include <stdio.h>
#include <math.h>
#include <time.h>
/*
double fact(double n) //FOR EXTRA PRECISION WE CAN UNCOMMENT THIS
{
    int factorial=1;
    for (int i=1; i<=n;i++){
        factorial=factorial*i;
    }
    return factorial;
}
*/
double f(double x, double y)
{
    return  -100000*y + 99999*pow(M_E,-x); //RETURNS F(X) VALUE
}
/*
double f1(double x, double y)//FOR EXTRA PRECISION WE CAN UNCOMMENT THIS
{
    return  3*y*pow(x,2);
}
double f2(double x, double y)//FOR EXTRA PRECISION WE CAN UNCOMMENT THIS
{
    return  y*6*x;
}
double f3(double x, double y)//FOR EXTRA PRECISION WE CAN UNCOMMENT THIS
{
    return  6*y;
}
*/
int main()
{
    double h;
    printf("BY EXPLICIT EULER'S METHOD\n");
```

```
    printf("Enter step size value h: "); //THAT VALUE OF H FOR WHICH THE ANSWER IS STABLE,LESS THAN 0.00002

    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0;//EQUIVALENT TO X_I
    double y_low=0,y_high;//EQUIVALENT TO Y_I AND Y_I+1
    FILE *fptr2;
    fptr2 = fopen("eee.txt", "a");//OPENING FILE TO INPUT VALUES AS ORDERED PAIRS
    printf("X\t\tY\n");
    printf("\n");
    fprintf(fptr2,"%lf \t%lf\n",x_low,y_low);
    for (int i=1;i<=2/h;i++){
        y_high=y_low+h*f(x_low,y_low);// for higher orders +f1(x_low,y_low)*h*h/fact(2)+f2(x_low,y_low)*h*h*h/fact(3)+f3(x_low,y_low)*h*h*

        fprintf(fptr2,"%lf \t%0.12lf\n",x_low+h,y_high);//TO WRITE INTO THE FILE ORDERED PAIRS OF X AND Y AS OBTAINED FROM EXPLICIT CODE
        y_low=y_high;//RECURSIVE RELATION
        x_low=x_low+h; //RECURSIVE RELATION

    }
    fclose(fptr2);

    clock_t begin = clock();//START TIME
    clock_t end = clock();//END TIME
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;//TIME OF EXECUTION IN SECONDS
    printf("EXECUTION TIME: %lf",time_spent);
    return 0;
}
```

Listing 6: Code snippet to perform Explicit Euler method

```
#include <stdio.h>      //initialising libraries
#include <math.h>       //for power function
#include <time.h>       //for calculating time complexity to improve the efficiency of algorithm


double f(double x, double y)
{
    return  -100000*y + 99999*pow(M_E,-x); //to  return function values as and when necessary
}

int main()
{
```

```c
    double h;
    printf("BY IMPLICIT EULER'S METHOD\n");
    printf("Enter step size value h: ");// the given value of h is 0.1
    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    FILE *fptr2;
    fptr2 = fopen("iii.txt", "a");
    double x_low=0;//Lower limit of operation of x
    double y_low=0,y_high;//Lower and higher limist of operation of y
    printf("X\t\tY\n");
    printf("\n");
    fprintf(fptr2,"%lf \t%lf\n",x_low,y_low);
    x_low=x_low+h;
    for (int i=1;i<=2/h;i++){
        y_high=(y_low+h*99999*pow(M_E,-x_low))/(1+100000*h); //the implicit formula as given from chapra's text page 754
        fprintf(fptr2,"%lf \t%0.12lf\n",x_low,y_high); //we print  the values of each intermediate (x,y)
        y_low=y_high; //recursive relations
        x_low=x_low+h;

    }
    fclose(fptr2);
    //to calculate the time of executuon
    clock_t begin = clock(); //to calculate the time  taken by code start time

    clock_t end = clock();//end time
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;//time of execution
    printf("EXECUTION TIME: %lf",time_spent);//answer output for 12 digit
    return 0;
}
```

Listing 7: Code snippet to perform Implicit Euler method

```c
#include <stdio.h> //initialising libraries
#include <math.h>//initialising libraries
#include <time.h>//initialising libraries


double f(double x)
{
    return -pow(M_E,-100000*x)+pow(M_E,-x);  //returns true values of function
}

int main()
{
```

```
    double h;
    FILE *fptr2;
    fptr2 = fopen("ttt.txt", "a");

    printf("Enter step size value h: "); //input of h value
    scanf("%lf", &h);
    printf("h=%lf\n",h);
    printf("\n");
    double x_low=0;
    printf("X\t\tY\n");
    printf("\n");
    for (int i=1;i<=2/h+1;i++){
        fprintf(fptr2,"%lf \t%0.12lf\n",x_low,f(x_low));//for plotting as many outputs as required.
        x_low=x_low+h;// recursive condition
    }
    fclose(fptr2);
    return 0;
}
```

Listing 8: Code snippet to calculate the true values for equation obtained by solving manually

2.6 Contributions

Approach, algorithm, code, results, inferences, LaTeX, Graphs: Me(AMIZHTHNI)
Complete efforts were put in by me.