**Assignment - 9**

**Ashutosh Jha**

**11811**

***Ques. 1:*** Write a code for Runge Kutta 2nd order, Runge Kutta 4th order and Euler's method for the given equation:

= 1 + *{1 <= x <= 6}*

Where step size, *h = 0.25*

& exact solution: *y(x) = x (1 + ln x)*

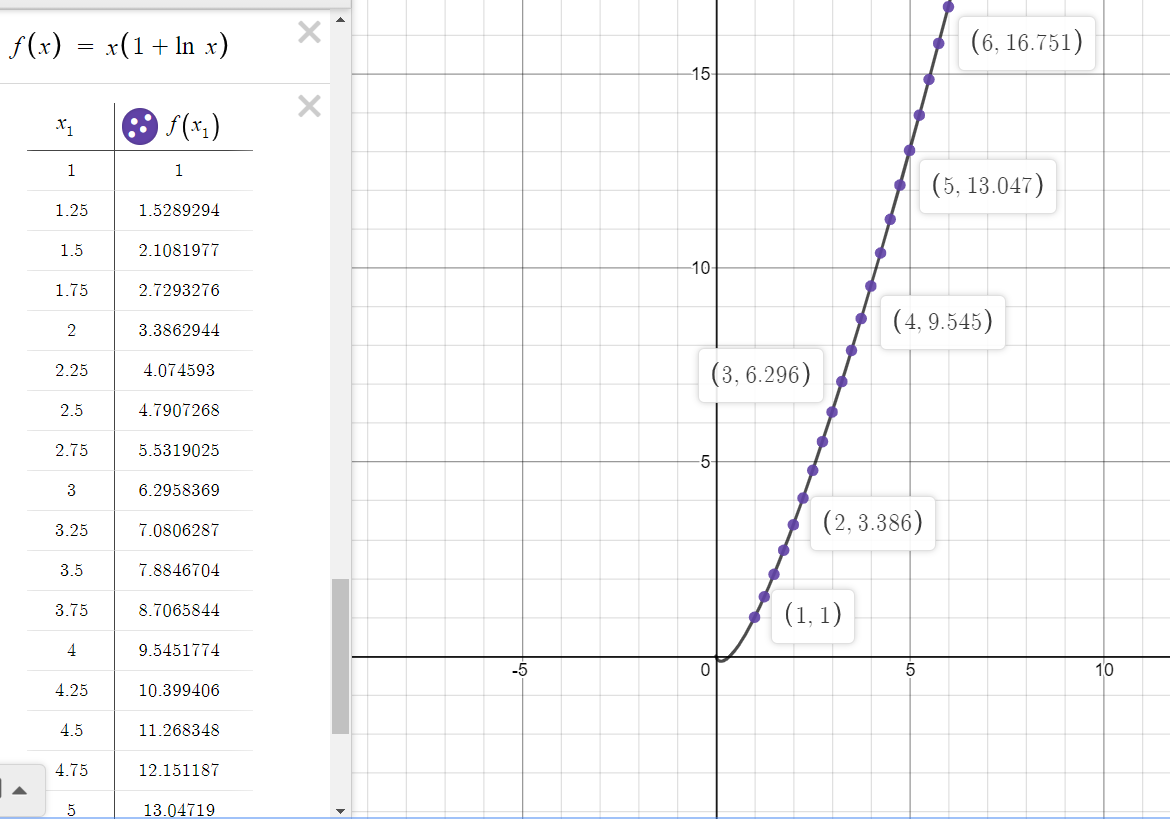
***Sol. 1:***

**Observation Table:**

| ***x*** | ***y (Runge Kutta 2)*** | ***y (Runge Kutta 4)*** | ***y (Euler’s)*** | ***y (Exact Value)*** |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 |
| 1.25 | 1.251259 | 1.528909 | 1.5 | 1.5289294 |
| 1.5 | 1.517761 | 2.108165 | 2.05 | 2.1081977 |
| 1.75 | 1.775651 | 2.729285 | 2.64167 | 2.7293276 |
| 2 | 2.033039 | 3.386243 | 3.26905 | 3.3862944 |
| 2.25 | 2.290006 | 4.074533 | 3.92768 | 4.074593 |
| 2.5 | 2.546614 | 4.79066 | 4.61409 | 4.7907268 |
| 2.75 | 2.802913 | 5.531828 | 5.3255 | 5.5319025 |
| 3 | 3.058942 | 6.295755 | 6.05963 | 6.2958369 |
| 3.25 | 3.314731 | 7.08054 | 6.8146 | 7.0806287 |
| 3.5 | 3.570307 | 7.884575 | 7.5888 | 7.8846704 |
| 3.75 | 3.825692 | 8.706482 | 8.38086 | 8.7065844 |
| 4 | 4.080903 | 9.545068 | 9.18958 | 9.5451774 |
| 4.25 | 4.335957 | 10.399289 | 10.0139 | 10.399406 |
| 4.5 | 4.590866 | 11.268225 | 10.853 | 11.268348 |
| 4.75 | 4.845642 | 12.151056 | 11.7059 | 12.151187 |
| 5 | 5.100296 | 13.047052 | 12.572 | 13.04719 |
| 5.25 | 5.354836 | 13.955553 | 13.4506 | 13.955697 |
| 5.5 | 5.60927 | 14.875963 | 14.3411 | 14.876115 |
| 5.75 | 5.863605 | 15.807741 | 15.243 | 15.807899 |
| 6 | 6.117848 | 16.750393 | 16.1557 | 16.750557 |

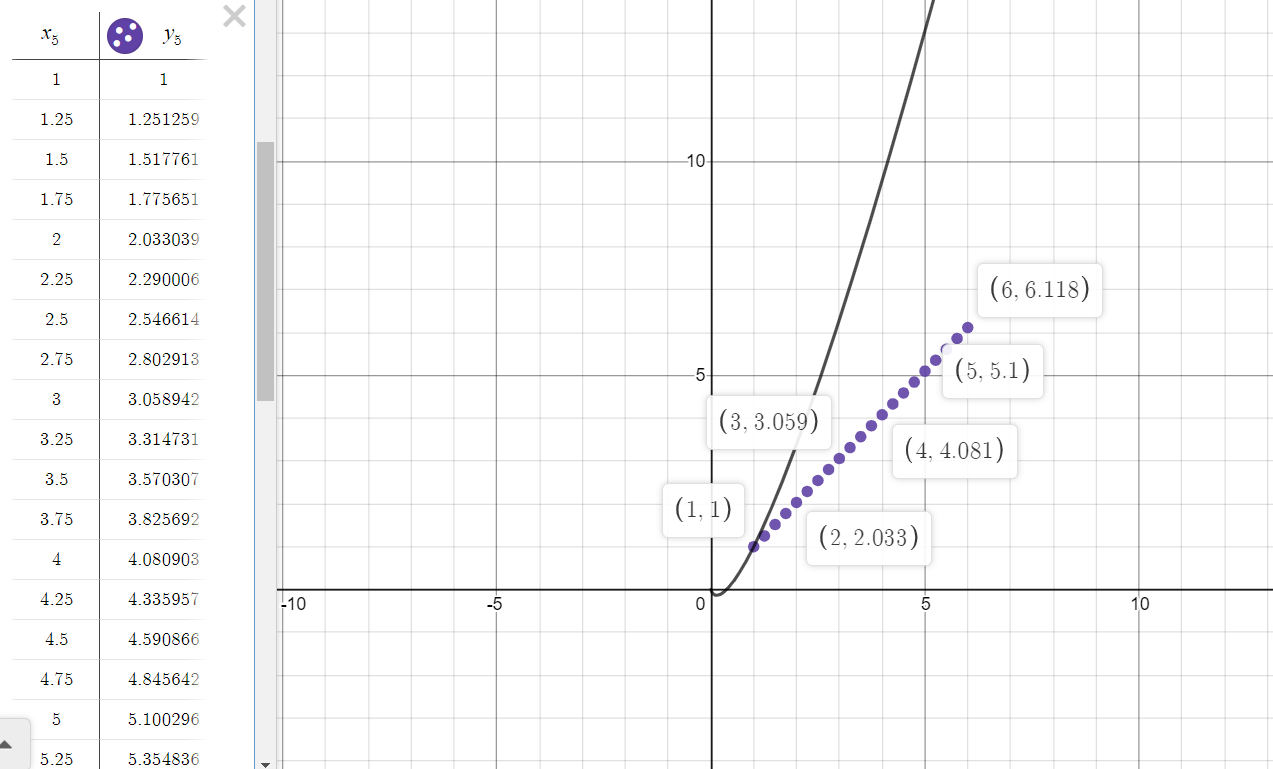
**Graph for Exact Values:**

This graph shows the exact values of the function in the interval [1, 6]. The violet dots depict these points and the black line shows our function

****

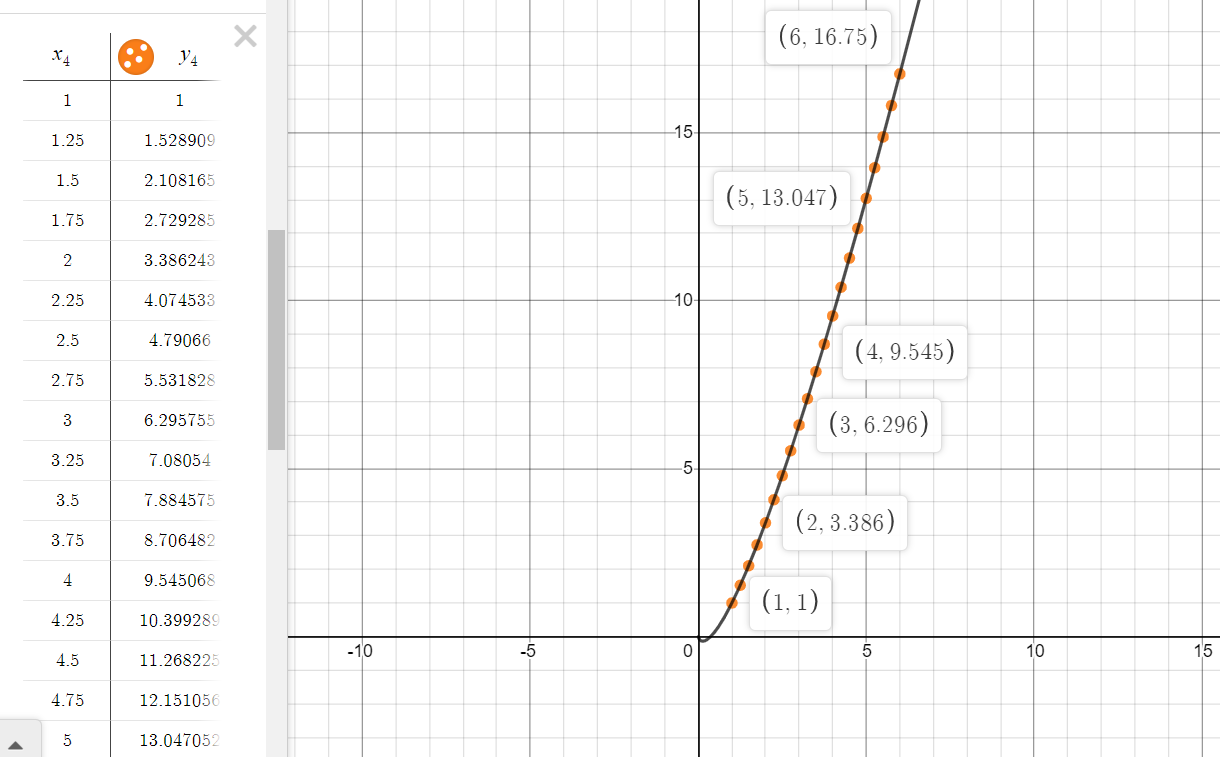
**Graph for Runge Kutta - 2nd Order:**

The violet dots show the value of y for a given x using Runge Kutta 2nd order. Black line denotes our function

****

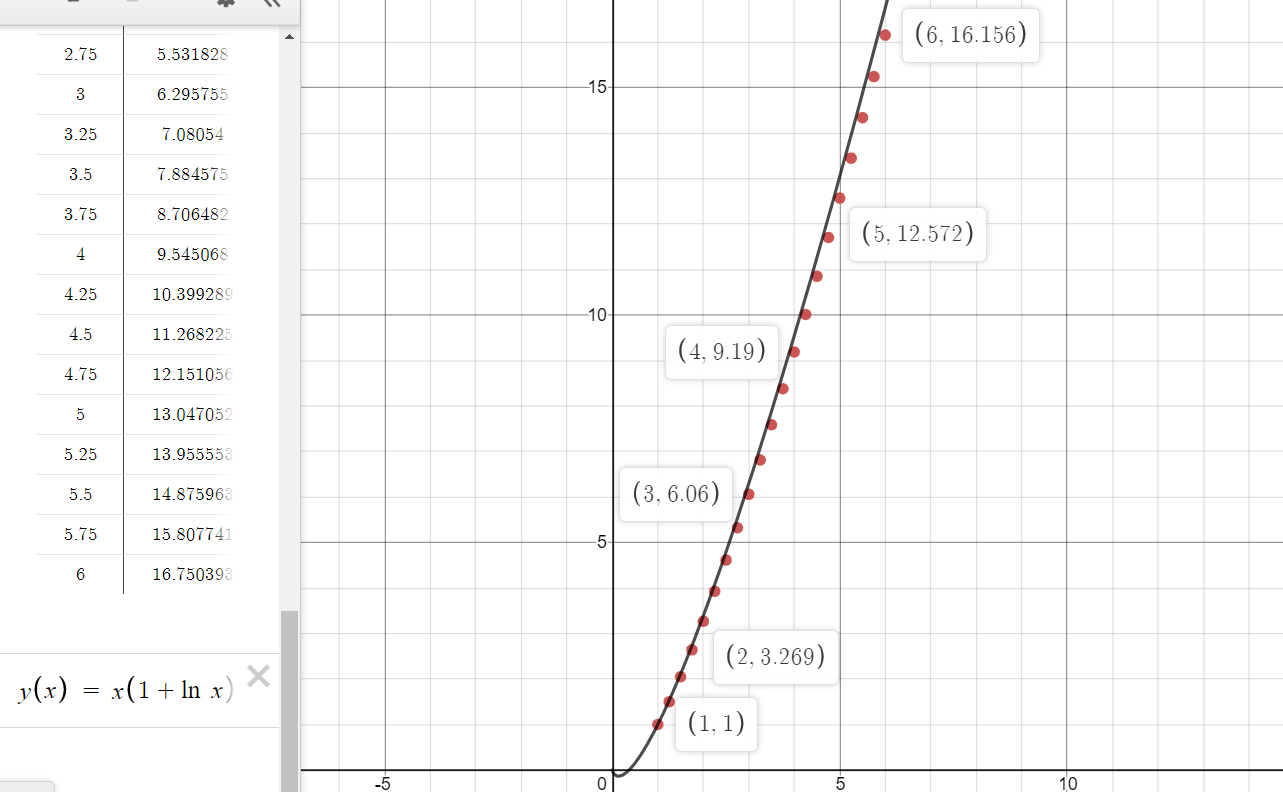
**Graph for Runge Kutta - 4th Order**

The orange dots show the value of y for a given x using Runge Kutta 4th order. Black line denotes our function

****

**Graph for Euler’s Method:**

The red dots show the value of y for a given x using Euler’s method. Black line denotes our function

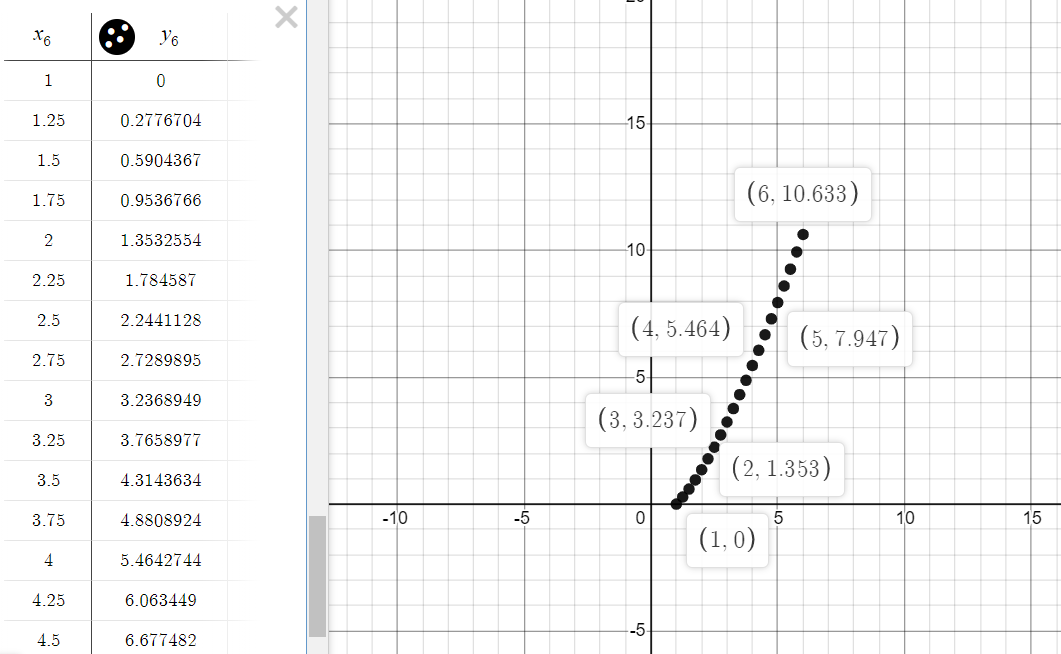
****

**Error Table:**

| ***x*** | ***Runge Kutta 2*** | ***Runge Kutta 4*** | ***Euler’s*** |
| --- | --- | --- | --- |
| 1 | 0 | 0 | 0 |
| 1.25 | 0.2776704 | 0.0000204 | 0.0289294 |
| 1.5 | 0.5904367 | 0.0000327 | 0.0581977 |
| 1.75 | 0.9536766 | 0.0000426 | 0.0876576 |
| 2 | 1.3532554 | 0.0000514 | 0.1172444 |
| 2.25 | 1.784587 | 0.00006 | 0.146913 |
| 2.5 | 2.2441128 | 0.0000668 | 0.1766368 |
| 2.75 | 2.7289895 | 0.0000745 | 0.2064025 |
| 3 | 3.2368949 | 0.0000819 | 0.2362069 |
| 3.25 | 3.7658977 | 0.0000887 | 0.2660287 |
| 3.5 | 4.3143634 | 0.0000954 | 0.2958704 |
| 3.75 | 4.8808924 | 0.0001024 | 0.3257244 |
| 4 | 5.4642744 | 0.0001094 | 0.3555974 |
| 4.25 | 6.063449 | 0.000117 | 0.385506 |
| 4.5 | 6.677482 | 0.000123 | 0.415348 |
| 4.75 | 7.305545 | 0.000131 | 0.445287 |
| 5 | 7.946894 | 0.000138 | 0.47519 |
| 5.25 | 8.600861 | 0.000144 | 0.505097 |
| 5.5 | 9.266845 | 0.000152 | 0.535015 |
| 5.75 | 9.944294 | 0.000158 | 0.564899 |
| 6 | 10.632709 | 0.000164 | 0.594857 |

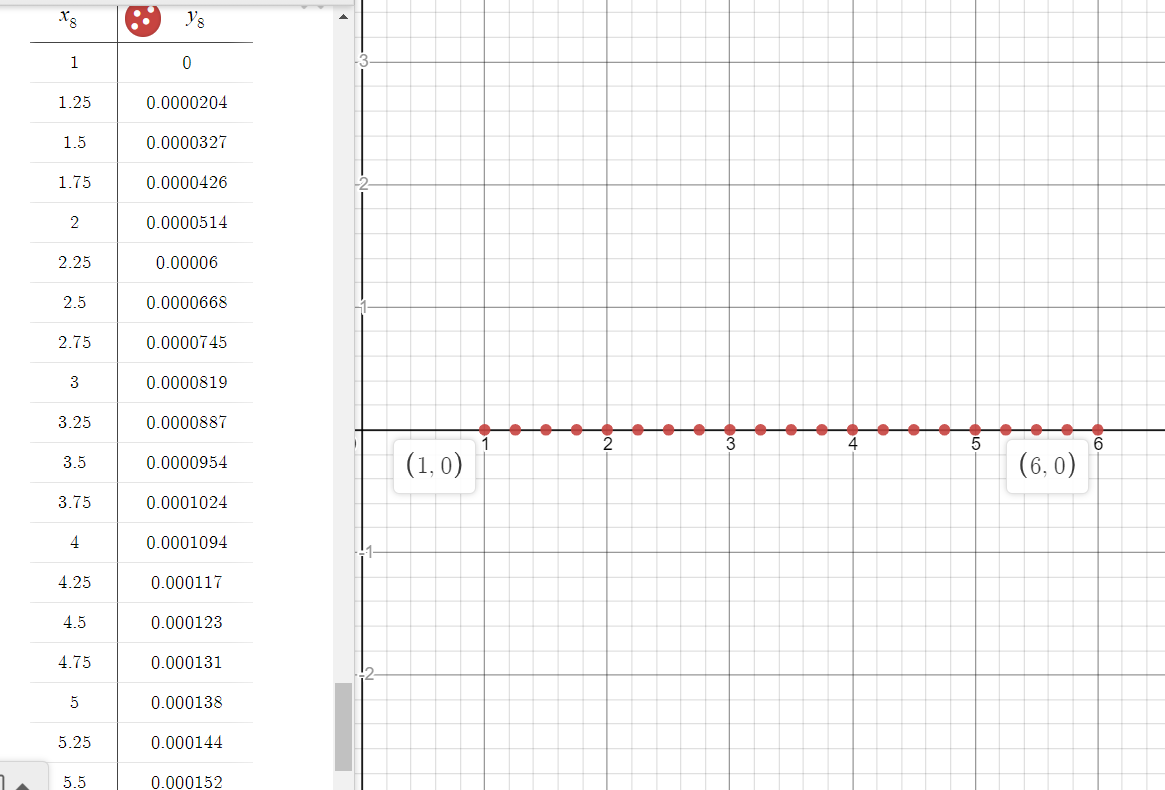
**Error Graph for Runge Kutta 2nd Order:**

The black dots denote the error for a given value of x using Runge Kutta 2nd Order. The error is calculated using Actual Value - Estimated Value

****

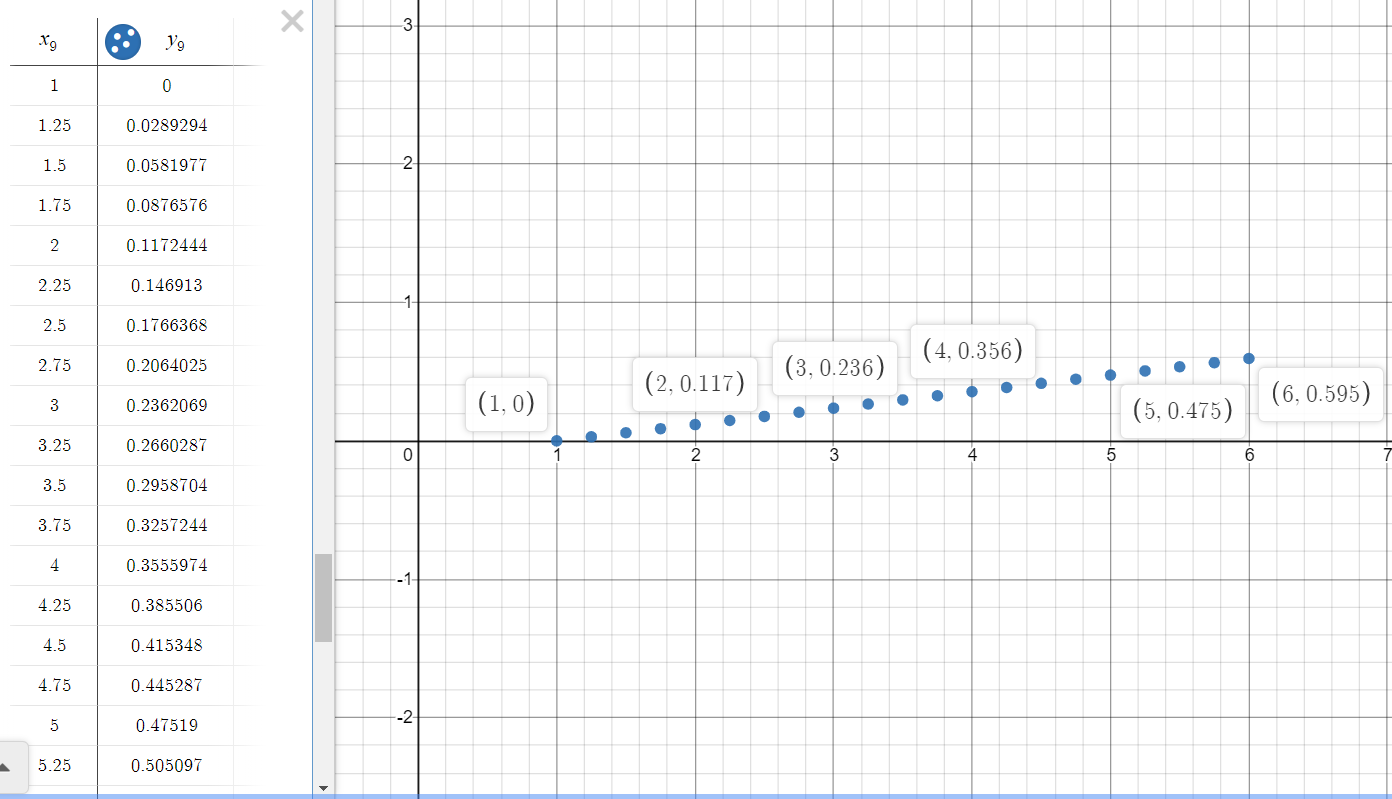
**Error Graph for Runge Kutta 4th Order:**

The red dots denote the error for a given value of x using Runge Kutta 4th Order. The error is calculated using Actual Value - Estimated Value

****

**Error Graph for Euler’s method :**

The blue dots denote the error for a given value of x using Euler’s method. The error is calculated using Actual Value - Estimated Value

****

**Conclusion :**

For *Runge Kutta 4th order*, the error reduces in the order *O(h5)* while for the *Runge Kutta 2nd order* the error reduces in the order *O(h3)* and for *Euler’s method* the error reduces in the order *O(h2)*. It is evident from the error graphs that *Runge Kutta 4th order* is best as compared to Euler’s and *Runge Kutta 2nd Order*. For *Euler’s method*, if the step size is large then the numerical solution comes inaccurate and for small steps, accuracy of the numerical solution comes much higher than before (If we take step size as *0.5* instead of *0.25*, then error increases significantly).

**Appendix:**

**Code for Runge Kutta 4th Order (Using C)*:***

#include<stdio.h>

float dydx(float x, float y)

{

**return**(1 + y/x);

}

float rungeKutta(float x0, float y0, float x, float h)

{

int n = (int)((x - x0) / h);

float k1, k2, k3, k4;

float y = y0;

**for** (int i=1; i<=n; i++)

{

k1 = h\*dydx(x0, y);

k2 = h\*dydx(x0 + 0.5\*h, y + 0.5\*k1);

k3 = h\*dydx(x0 + 0.5\*h, y + 0.5\*k2);

k4 = h\*dydx(x0 + h, y + k3);

y = y + (1.0/6.0)\*(k1 + 2\*k2 + 2\*k3 + k4);;

x0 = x0 + h;

}

**return** y;

}

int main()

{

**for**(int i = 1; i <= 6; i++) {

printf("**\n** x : %d y : %f", i,

rungeKutta(1, 1, i, 0.25));

}

**return** 0;

}

**Code for Runge Kutta 2nd Order (Using C)*:***

#include <stdio.h>

float dydx(float x, float y)

{

**return** (1 + y/x);

}

float rungeKutta(float x0, float y0,

float x, float h)

{

int n = (int)((x - x0) / h);

float k1, k2;

float y = y0;

**for** (int i = 1; i <= n; i++) {

k1 = h \* dydx(x0, y);

k2 = h \* dydx(x0 + 0.5 \* h,

y + 0.5 \* k1);

y = y + (1.0 / 6.0) \* (k1 + 2 \* k2);

x0 = x0 + h;

}

**return** y;

}

int main()

{

**for**(int i = 1; i <= 6; i++) {

printf("**\n**x : %d y : %f", i,

rungeKutta(1, 1, i, 0.25));

}

**return** 0;

}

**Code for Euler’s Method (Using C)*:***

#include <iostream>

using namespace std;

float func(float x, float y)

{

**return** (1 + y/x);

}

void euler(float x0, float y, float h, float x)

{

float temp = -0;

**while** (x0 < x) {

temp = y;

y = y + h \* func(x0, y);

x0 = x0 + h;

}

cout << "x : "

<< x << " y : " << y << endl;

}

int main()

{

**for**(int i =1; i <= 6; i++) {

euler(1, 1, 0.25, i);

}

**return** 0;

}