1. Analytical solution of the first order equation

Using:

Formulas:

Calculations:

Using partial fractions

Finally, using the inverse Laplace transform to obtain the solution

2. Full program codes

1) Implement Euler method

1. #include <iostream>
2. #include <math.h>
3. #include <fstream>
4. **using** **namespace** std;
6. **double** fun(**double** x, **double** y)
7. {
8. **return** (-2\*y+7\*pow(2.71,-2\*x));
9. }
11. **double** analytSol(**double** x)
12. {
13. **return** (7\*x-10)/(exp(2\*x));
14. }
16. **int** main()
17. {
18. ofstream X, Y, globalError, localError;
19. X.open ("dataGraphX.txt");
20. Y.open ("dataGraphY.txt");
21. globalError.open ("globalError.txt");
23. **double** h,maxv=5;
24. cout << "Enter value of h: ";
25. cin >> h;
27. **int** arrsize=(5/h);
28. **double** x[arrsize];
29. **double** y[arrsize];
30. **double** func[arrsize];
32. x[0] = 0;
33. //y[0] = -10;
34. func[0] = 27;
36. cout << "\ni\t\t" << "x\t\t" << "y" << endl << endl;
38. **for**(**int** i=0; i<=arrsize; i++) {
39. x[i] = x[0]+i\*h;
40. **if** (i == 0) {
41. y[i] = -10;
42. } **else** {
43. y[i] = y[i-1] + (h\*func[i-1]);
44. }
45. globalError << y[i] - analytSol(x[i]) << ",";
46. //X << x[i] << ",";
47. //Y << y[i] << ",";
49. // for calculating simple error for Y2
50. **if** (i%2 == 0) {
51. X << x[i] << ",";
52. Y << y[i] << ",";
53. } //
55. func[i] = fun(x[i],y[i]);
56. cout << i << "      " << x[i] << "      " << y[i] << "      " << endl;
57. }

60. **return** 0;
61. }

2) Implement 4th order Runge-Kutta method

1. #include <iostream>
2. #include <math.h>
3. #include <fstream>
4. **using** **namespace** std;
6. **double** fun(**double** a, **double** b, **double** c, **double** x, **double** y)
7. {
8. **return** (a\*y)+(b\*exp(c\*x));
9. }
11. **double** analytSol(**double** x)
12. {
13. **return** (7\*x-10)/(exp(2\*x));
14. }
16. **int** main()
17. {
18. ofstream X, Y, globalError, localError;
19. X.open ("dataGraphX.txt");
20. Y.open ("dataGraphY.txt");
21. globalError.open ("globalError.txt");
23. **double** a, b, c, d, e, h, sumdy;
24. cout << "Give parameters: ";
25. cin >> a; cin >> b; cin >> c; cin >> d; cin >> e; cin >> h;
26. **double** x0 = 0;
27. **double** y0 = d; // initial condition
28. **int** arrSize = (e/h)+1;
30. **double** x[arrSize][4];
31. **double** y[arrSize][4];
32. **double** k[arrSize][4];
33. **double** dy[arrSize][4];
35. cout << "\ni\t" << "x\t" << "y" << endl << endl;
37. **for** (**int** i = 0; i <= arrSize; i++) {
39. x[i][0] = x0;
40. y[i][0] = y0;
42. globalError << y[i][0] - analytSol(x[i][0]) << ",";
43. X << x[i][0] << ",";
44. Y << y[i][0] << ",";
46. /\* for calculating simple error for Y2
47. if (i%2 == 0) {
48. X << x[i][0] << ",";
49. Y << y[i][0] << ",";
50. } \*/
52. **if** (i == 0) {
53. k[i][0] = h\*fun(a, b, c, x[i][0], y[i][0]);
54. } **else** {
55. k[i][0] = k[i-1][3];
56. }
57. dy[i][0] = k[i][0];
59. cout << i << "\t" << x[i][0] << "\t" << y[i][0] << endl;
61. x[i][1] = x0 + (0.5)\*h;
62. y[i][1] = y0 + (0.5)\*k[i][0];
63. k[i][1] = h\*fun(a, b, c, x[i][1], y[i][1]);
64. dy[i][1] = 2\*k[i][1];
66. cout << i << "\t" << x[i][1] << "\t" << y[i][1] << endl;
68. x[i][2] = x0 + (0.5)\*h;
69. y[i][2] = y0 + (0.5)\*k[i][1];
70. k[i][2] = h\*fun(a, b, c, x[i][2], y[i][2]);
71. dy[i][2] = 2\*k[i][2];
73. cout << i << "\t" << x[i][2] << "\t" << y[i][2] << endl;
75. x[i][3] = x0 + h;
76. y[i][3] = y0 + k[i][2];
77. k[i][3] = h\*fun(a, b, c, x[i][3], y[i][3]);
78. dy[i][3] = k[i][3];
80. cout << i << "\t" << x[i][3] << "\t" << y[i][3] << endl;
82. x0 = x[i][3];
83. sumdy = ((dy[i][0] + dy[i][1] + dy[i][2] + dy[i][3]))/6;
84. y0 = y0 + sumdy; // sum + y0
86. cout << endl;
87. }
89. X.close();
90. Y.close();
91. globalError.close();
92. **return** 0;
93. }

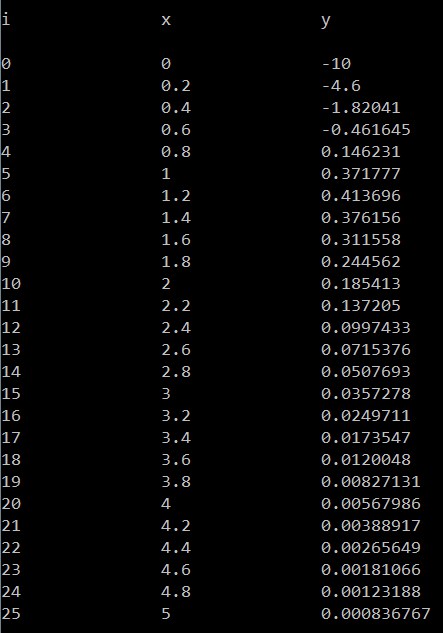
3. Programs’ outputs

1) Program outputs for Euler method

Working on parameters:



The program outputs



2) Program outputs for 4th order Runge-Kutta method

Working on parameters:

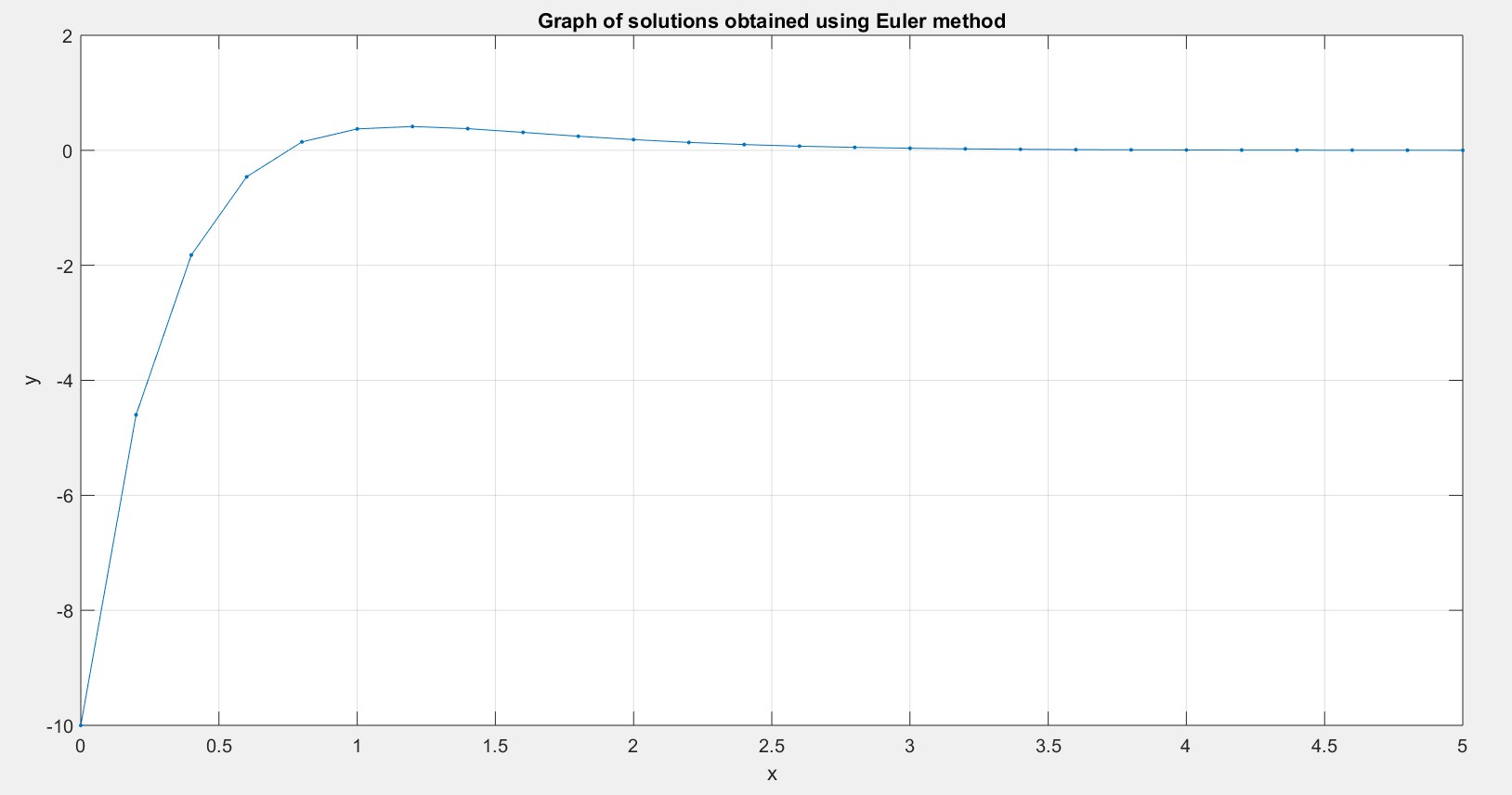


The program outputs

|  |  |  |
| --- | --- | --- |
|  |  |  |

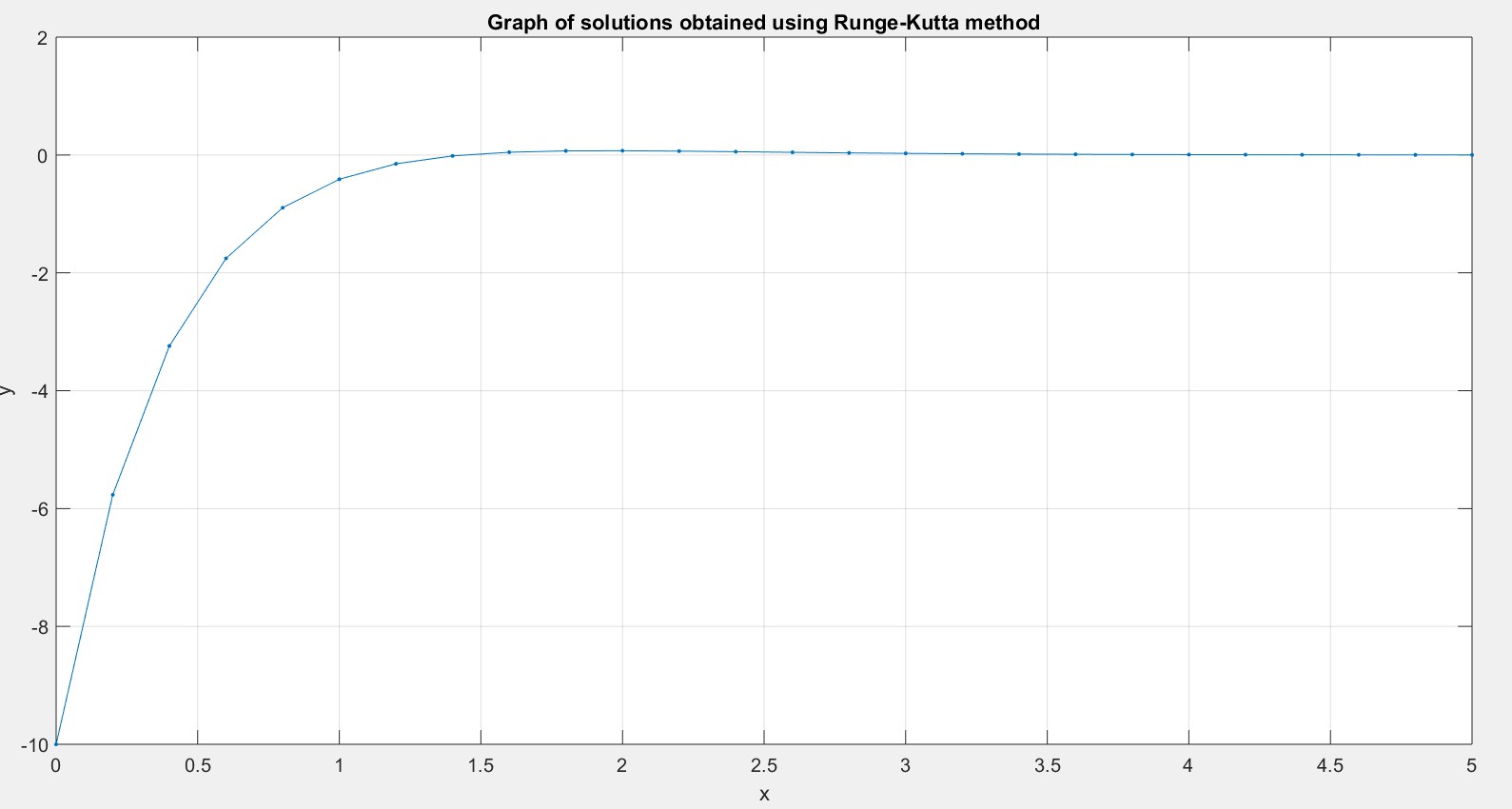
4. Graph solutions obtained using the programs

1) Using Euler method



h = 0.2

2) Using Runge-Kutta method



h = 0.2

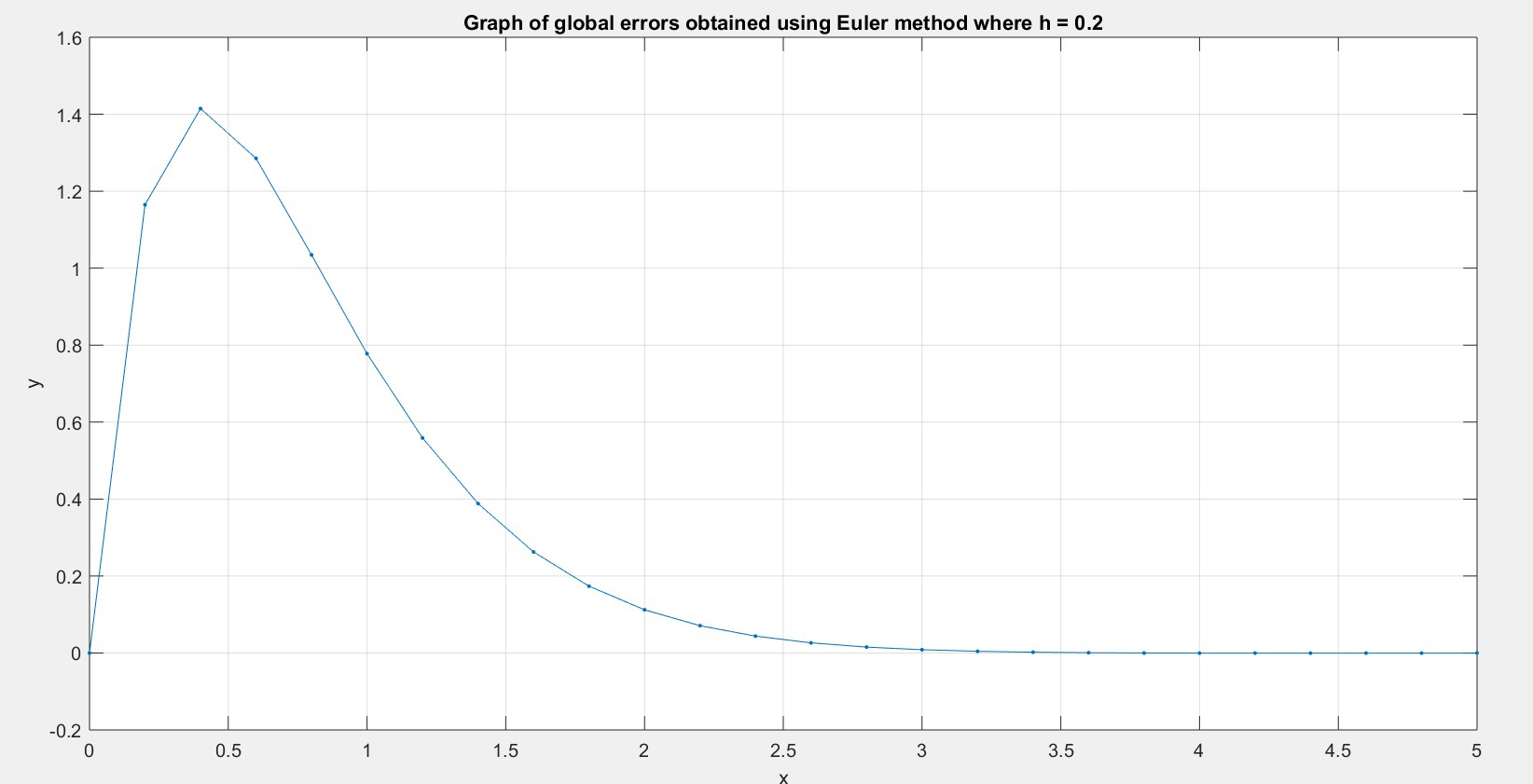
5. Analysis of both methods’ error

**1) Global error**

where,

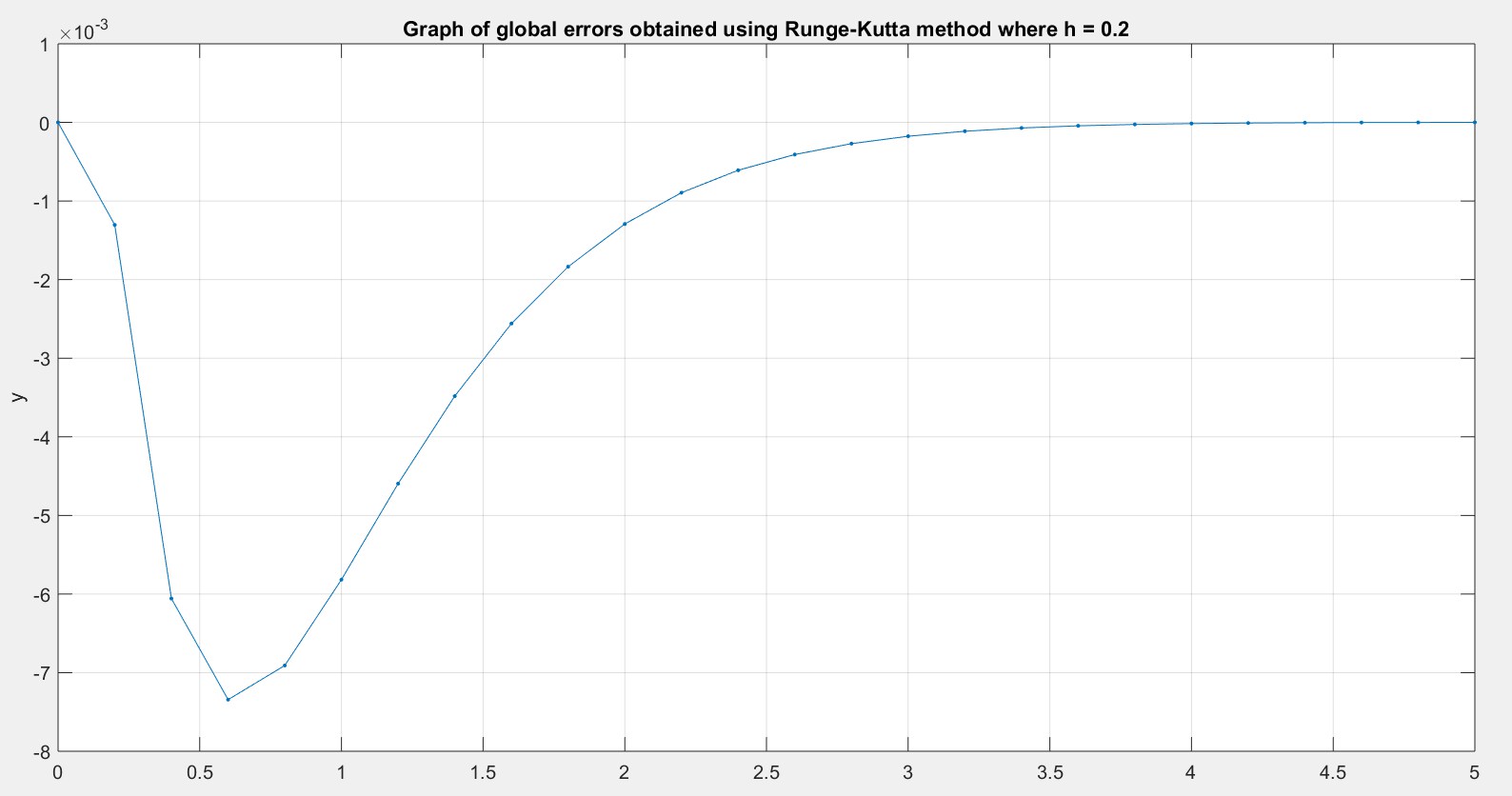
is a computed solution and is a true solution.

1. Using Euler method



h = 0.2

1. Using Runge-Kutta method



h = 0.2

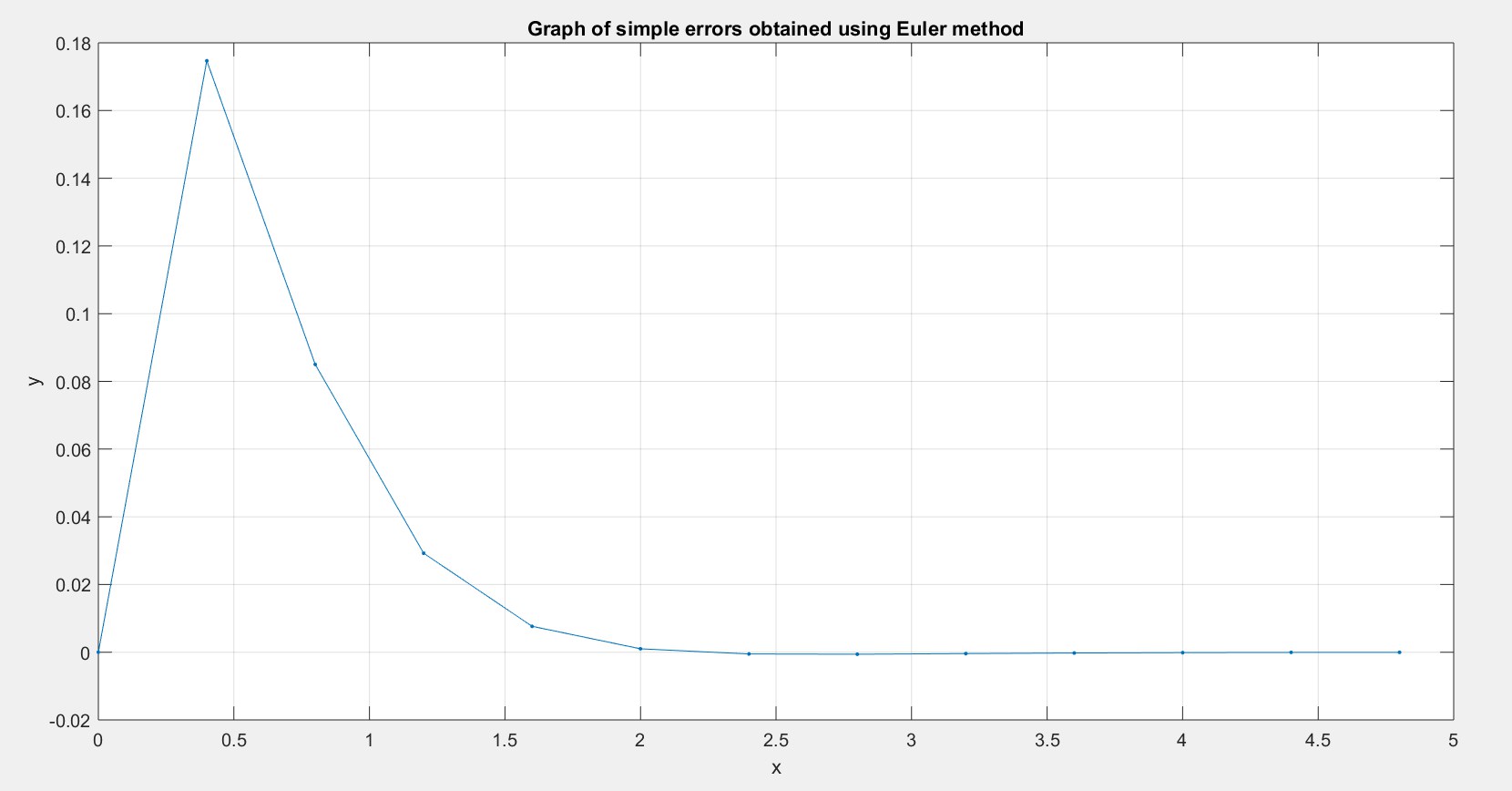
**2) Error obtained using simple calculations**

To estimate the error, there have to be chosen two intervals of length and

where

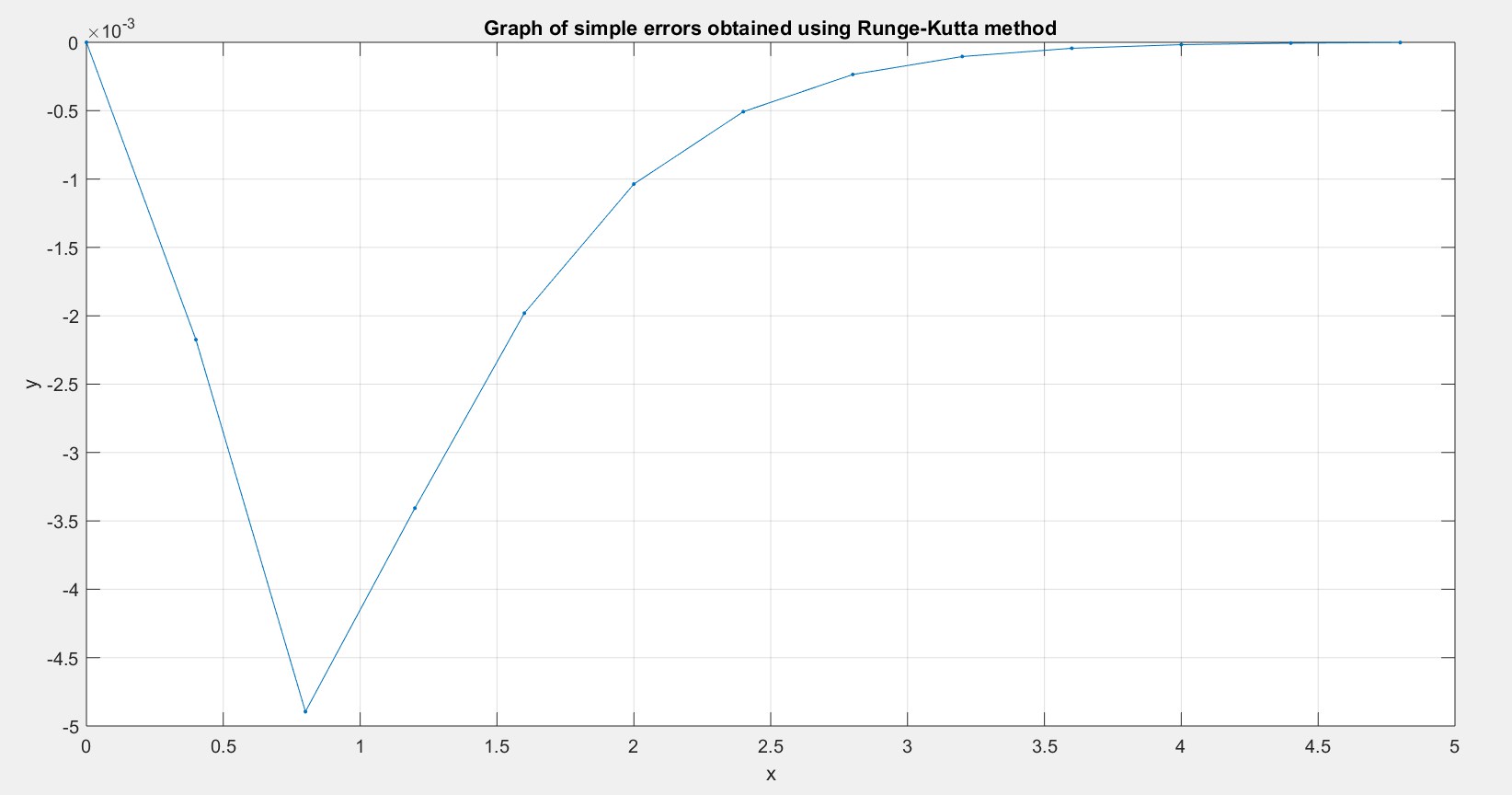
and from that

1. Using Euler method



h1 = 0.4, h2 = 0.2

1. Using Runge-Kutta method



h1 = 0.4, h2 = 0.2

**3) Error Term (using Taylor series)**

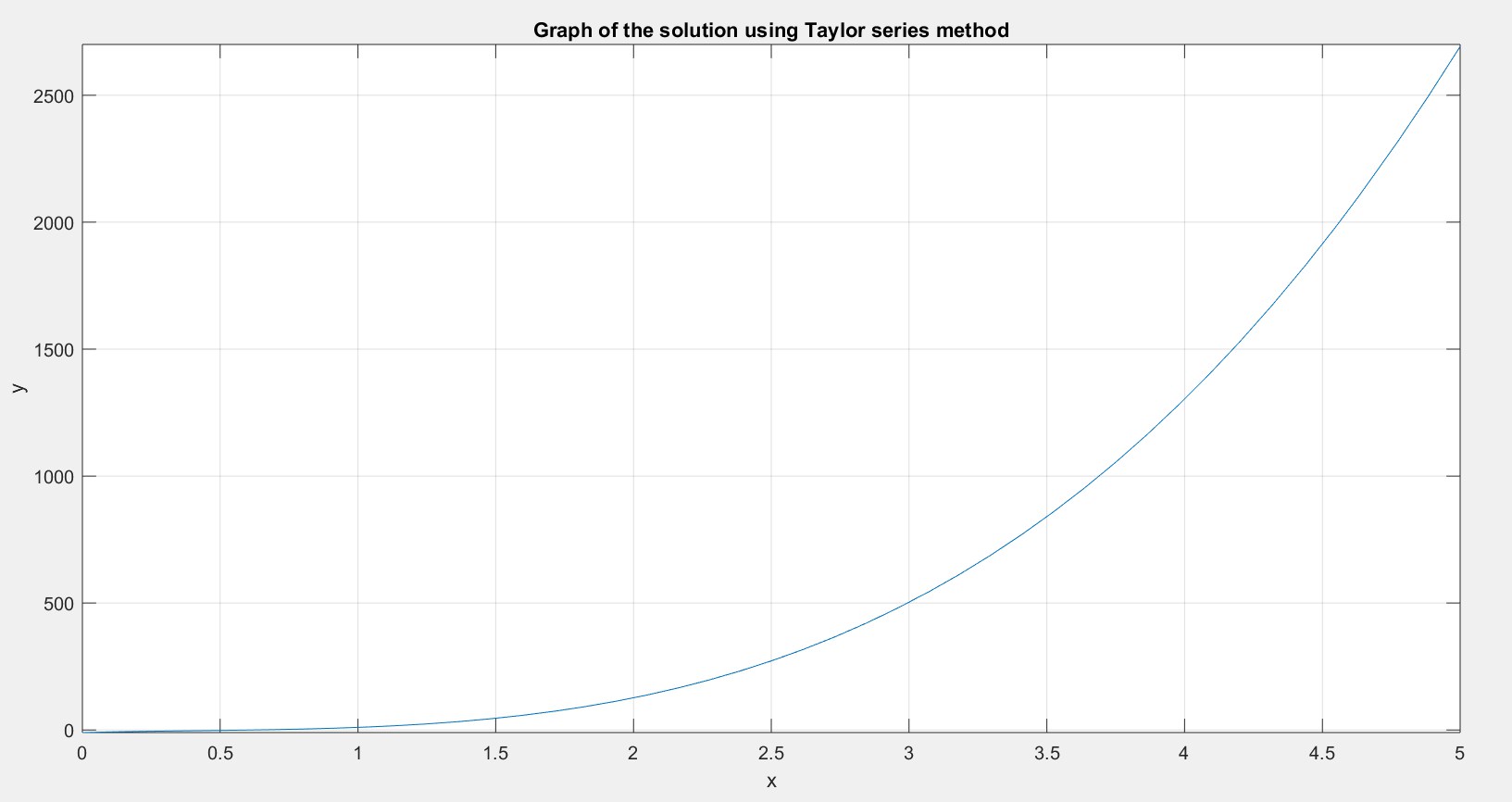
Using Runge-Kutta method

The coefficient of is

Estimating the error, for

6. Taylor series

Approximate solution of differential equation is of the form



7. Conclusions

The accurecies of the presented methods can be observed from the task dedicated to analysis of both methods’ error. From the obtained values we can conclude, that:

- The Runge-Kutta method is more accurate than the Euler method, giving very good approximation, for every chosens step,

- The Runge-Kutta method has a small error (of the magnitude of ), it seems that a step size doesn’t have any noticeable influence on the error,

We can conclude that Euler method has accuracy roughly equal to its step size.