Homework 12 – 5.2, 5.3, 5.4, 5.5

# Section 5.2

## Problem 3.

The actual solutions to the initial-value problems in Exercise 1 are given here. Compare the actual error at each step to the error bound.

Using Euler Method:

Using the solution:

The Error is given by:

## Problem 11.

Given the initial-value problem

With exact solution :

1. Approximate using Euler’s method with , , and .

Euler Method:

Using a program:

#include <iostream>

#include <fstream>

using namespace std;

double function(double, double);

double Euler(double, double,double);

int main()

{

// Create the file where the data will be stored.

ofstream outputFile;

outputFile.open("h = 0.2.txt");

double a = 0, b = 5; // t boundary.

double h = 0.2; // Mesh size.

double yn = 0; // initial value.

double s = (b - a) / h; // Number of steps.

double ti = a;

// Heading for the file.

outputFile << "Time" << " " << "Y(t)" << endl;

outputFile << ti << " " << yn << endl; // Inital points.

for (int i = 0; i < s; i++)

{

outputFile << ti + h << " " << Euler(yn, ti, h) << endl;

ti += h;

yn = Euler(yn, ti, h);

}

outputFile.close(); // Close the file.

return 0;

}

// Function of y and t.

double function(double y, double t)

{

double fxy = (-y) + t + 1;

return fxy;

}

// Euler's Method.

double Euler(double y, double t, double h)

{

double yi1 = y + h \* function(y, t);

return yi1;

}

1. Determine the optimal value of to use in computing , assuming and that eq. (5.14) is valid.

# Section 5.4

## Problem 1.

Use the Modified Euler method to approximate the solution to each of the following initial-value problems, and compare the results to the actual values.

Modified Euler’s Method:

Using the program for Euler Method and doing the following change:

// Modified Euler's Method.

double Euler(double y, double t, double h)

{

double yi1 = y + (h / 2) \* (function(y, t) + function(y + h \* function(y, t), t + h));

return yi1;

}

We get the following:

The actual solution is:

## Problem 13.

Repeat Exercise 1 using Runge-Kutta method of order four.

Runge-Kutta method of order 4:

Using a program:

#include <iostream>

#include <fstream>

using namespace std;

double function(double, double);

double K1(double, double, double);

double K2(double, double, double, double);

double K3(double, double, double, double);

double K4(double, double, double,double);

double RK4(double, double, double, double, double, double, double);

int main()

{

// Create the file where the data will be stored.

ofstream outputFile;

outputFile.open("Runge-Kutta, h = 0.5.txt");

double a = 0, b = 1; // t boundary.

double h = 0.5; // Mesh size.

double yn = 0; // initial value.

double k1, k2, k3, k4; // K values.

double s = (b - a) / h; // Number of steps.

double ti = a; // initial time.

// Heading for the file.

outputFile << "Time" << " " << "Y(t)" << endl;

outputFile << ti << " " << yn << endl; // Inital points.

for (int i = 0; i < s; i++)

{

// The K values.

k1 = K1(yn, ti, h);

k2 = K2(yn, ti, h, k1);

k3 = K3(yn, ti, h, k2);

k4 = K4(yn, ti, h, k3);

// Finding y(t\_(i+1)).

yn = RK4(yn, ti, h, k1, k2, k3, k4);

outputFile << ti + h << " " << yn << endl; // Write the information.

ti += h; // Increase the time.

}

outputFile.close(); // Close the file.

return 0;

}

// Function of y and t.

double function(double y, double t)

{

double fxy = t \* exp(3 \* t) - 2 \* y;

return fxy;

}

//K values.

double K1(double y, double t, double h)

{

double K1 = h \* function(y, t);

return K1;

}

double K2(double y, double t, double h, double K1)

{

double K2 = h \* function(y + (h / 2), y + (K1 / 2));

return K2;

}

double K3(double y, double t, double h, double K2)

{

double K3 = h \* function(y + (h / 2), y + (K2 / 2));

return K3;

}

double K4(double y, double t, double h, double K3)

{

double K4 = h \* function(y + K3, t + h);

return K4;

}

// Runge-Kutta Method.

double RK4(double y, double t, double h, double K1, double K2, double K3, double K4)

{

double r = (double) 1 / 6; // have to convert to double for precision purposes.

double yi1 = y + r \* (K1 + 2 \* K2 + 2 \* K3 + K4);

return yi1;

}

We get that the values are:

# Section 5.5

## Problem 3.

Use the Runge-Kutta-Fehlberg method with tolerance , , and to approximate the solution to the following initial-value problem. Compare the results to the actual value.

Using a program:

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

// All the functions that will be used.

double Y(double);

double function(double, double);

double K1(double, double, double);

double K2(double, double, double, double);

double K3(double, double, double, double, double);

double K4(double, double, double, double, double, double);

double K5(double, double, double, double, double, double, double);

double K6(double, double, double, double, double, double, double, double);

double RK4(double, double, double, double, double, double, double);

double RK5(double, double, double, double, double, double, double, double);

double q(double, double, double, double);

int main()

{

// Create the file where the data will be stored.

ofstream outputFile;

outputFile.open("Runge-Kutta-Fehlberg, h = 0.5.txt");

double a = 1, b = 3; // t boundary.

double h = 0.5; // Initial Mesh size.

double hmax = 0.5; // Mesh size cannot get bigger than this.

double hmin = 0.05; // Mesh size cannot get smaller than this.

double y = 0; // Initial value.

double yn = 0; // RK4 initial value.

double ynb = 0; // RK5 initial value.

double Q = 0; // Scaling.

double tol = pow(10, -6); // Tolerance.

double R = 0; // Error.

double k1, k2, k3, k4, k5, k6; // K values.

bool cont = true; // Loop will execute until the limit is reached.

double t = a; // initial time.

// Heading for the file.

outputFile << "Time" << " " << "w(t)" << " "<< "h" << " " << "y(t)" << endl;

outputFile << fixed << setprecision(6) << t << " " << y << " " << h << " " << Y(t) << endl; // Inital points.

while (cont)

{

// The K values.

k1 = K1(y, t, h);

k2 = K2(y, t, h, k1);

k3 = K3(y, t, h, k1, k2);

k4 = K4(y, t, h, k1, k2, k3);

k5 = K5(y, t, h, k1, k2, k3, k4);

k6 = K6(y, t, h, k1, k2, k3, k4, k5);

// Finding the values.

yn = RK4(y, t, h, k1, k3, k4, k5);

ynb = RK5(y, t, h, k1, k3, k4, k5, k6);

R = (1 / h) \* abs(ynb - yn); // Error.

//Cases.

if (R < tol)

{

t += h;

y = yn;

outputFile << fixed << setprecision(6) << t << " " << y << " " << h << " " << Y(t) << endl; // Inital points.

}

// The q value.

Q = q(tol, h, ynb, yn);

// Conditions on Q.

if (Q <= 0.1)

{

h \*= 0.1;

}

else if (Q >= 4)

{

h \*= 4;

}

else

{

h \*= Q;

}

if (h > hmax)

{

h = hmax;

}

// Changing conditions on t.

if (t >= b)

{

cont = false;

}

else if (t + h > b)

{

h = b - t;

}

else if (h < hmin)

{

cont = false;

cout << "Minimum h exceeded.";

}

}

outputFile.close(); // Close the file.

return 0;

}

// Solution.

double Y(double t)

{

double yt = t \* tan(log(t));

return yt;

}

// Function of y and t.

double function(double y, double t)

{

double fyt = 1 + (y / t) + pow((y / t), 2);

return fyt;

}

//K values.

double K1(double y, double t, double h)

{

double K1 = h \* function(y, t);

return K1;

}

double K2(double y, double t, double h, double K1)

{

double K2 = h \* function(y + (K1 / 4.), t + (h / 4.));

return K2;

}

double K3(double y, double t, double h, double K1, double K2)

{

double K3 = h \* function(y + (3. / 32.) \* K1 + (9. / 32.) \* K2, t + (3. / 8.) \* h);

return K3;

}

double K4(double y, double t, double h, double K1, double K2, double K3)

{

double K4 = h \* function(y + (1932. / 2197.) \* K1 - (7200. / 2197.) \* K2 + (7296. / 2197.) \* K3, t + (12. / 13.) \* h);

return K4;

}

double K5(double y, double t, double h, double K1, double K2, double K3, double K4)

{

double K5 = h \* function(y + (439. / 216.) \* K1 - 8 \* K2 + (3680. / 513.) \* K3 - (845. / 4104.) \* K4, t + h);

return K5;

}

double K6(double y, double t, double h, double K1, double K2, double K3, double K4, double K5)

{

double K6 = h \* function(y - (8. / 27.) \* K1 + 2. \* K2 - (3544. / 2565.) \* K3 + (1859. / 4104.) \* K4 - (11. / 40.) \* K5, t + (h / 2));

return K6;

}

// Runge-Kutta Method.

// Order 4.

double RK4(double y, double t, double h, double K1, double K3, double K4, double K5)

{

double yi1 = y + (25. / 216.) \* K1 + (1408. / 2565.) \* K3 + (2197. / 4104.) \* K4 - (1. / 5.) \* K5;

return yi1;

}

// Order 5.

double RK5(double y, double t, double h, double K1, double K3, double K4, double K5, double K6)

{

double yi1 = y + (16. / 135.) \* K1 + (6656. / 12825.) \* K3 + (28561. / 56430.) \* K4 - (9. / 50.) \* K5 + (2. / 55.) \* K6;

return yi1;

}

// The q value.

double q(double tol, double h, double yt, double y)

{

double q = pow((tol \* h) / (2 \* abs(yt - y)), (1. / 4.));

return q;

}

We get the following points:

Time w(t) h y(t)

1.000000 0.000000 0.500000 0.000000

1.145181 0.156202 0.145181 0.156202

1.282508 0.325864 0.137327 0.325863

1.422970 0.523858 0.140462 0.523858

1.548731 0.724267 0.125760 0.724267

1.666189 0.933176 0.117458 0.933176

1.778078 1.153622 0.111889 1.153621

1.885528 1.386957 0.107451 1.386956

1.989093 1.633924 0.103565 1.633923

2.089074 1.894989 0.099981 1.894987

2.185651 2.170475 0.096577 2.170473

2.278942 2.460619 0.093290 2.460617

2.369029 2.765603 0.090087 2.765601

2.455979 3.085570 0.086951 3.085568

2.539854 3.420639 0.083875 3.420636

2.620712 3.770908 0.080858 3.770905

2.698613 4.136460 0.077901 4.136457

2.773620 4.517370 0.075007 4.517366

2.845799 4.913702 0.072180 4.913698

2.915222 5.325515 0.069423 5.325509

2.981961 5.752860 0.066739 5.752854

3.000000 5.874106 0.018039 5.874100