Homework 13 – 5.6

# Section 5.6

## Problem 3.

Use each of the Adams-Bashforth methods to approximate the solutions to the following initial-value problems. In each case use starting values obtained from the Runge-Kutta method of order four. Compare the results to the actual values.

### Adams-Bashforth Two-Step Explicit Method:

Using a Program:

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

long double f(long double, long double);

long double Y(long double);

long double AB2(long double, long double, long double, long double, long double);

int main()

{

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

ofstream outputFile;

outputFile.open("Adams-Bashforth Two-Step Explicit Method, h = 0.1.txt");

long double a = 1, b = 2; // t boundary.

long double h = 0.1; // Mesh size.

long double y0 = 0, y1 = 0;

long double yn = 0; // W(t) value.

long double t = a; // initial time.

//Initial values.

y0 = Y(t);

y1 = Y(t + h);

// Heading for the file.

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

outputFile << showpoint << setprecision(10) << t << " " << y0 << " " << Y(t) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + h << " " << y1 << " " << Y(t + h) << endl; // Inital points.

outputFile << endl;

while (t < b - h)

{

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

yn = AB2(t + h, t, y1, y0, h);

outputFile << showpoint << setprecision(10) << t + 2 \* h << " " << yn << " " << Y(t + 2 \* h) << endl; // Write the information.

t += h; // Increase the time.

// Change the initial values.

y0 = y1;

y1 = yn;

}

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

return 0;

}

// Y(t).

long double Y(long double t)

{

long double Yt = t / (1 + log(t));

return Yt;

}

// Function of y and t.

long double f(long double y, long double t)

{

long double fxy = (y / t) - (y / t) \* (y / t);

return fxy;

}

// Adams-Bashforth Two-Step Explicit Method

long double AB2(long double ti, long double tim1, long double wi, long double wim1, long double h)

{

long double W = wi + (h / 2) \* (3 \* f(wi, ti) - f(wim1, tim1));

return W;

}

We get the following values.

Time W(t) Y(t)

1.000000000 1.000000000 1.000000000

1.100000000 1.004281728 1.004281728

1.200000000 1.016198436 1.014952314

1.300000000 1.031682331 1.029813689

1.400000000 1.049766652 1.047533919

1.500000000 1.069714250 1.067262354

1.600000000 1.091020604 1.088432687

1.700000000 1.113329617 1.110655052

1.800000000 1.136384686 1.133653557

1.900000000 1.159997452 1.157228433

2.000000000 1.184027397 1.181232218

### Adams-Bashforth Three-Step Explicit Method:

Changing the following:

// Adams-Bashforth Three-Step Explicit Method

long double AB3(long double ti, long double tim1, long double tim2, long double wi, long double wim1, long double wim2, long double h)

{

long double W = wi + (h / 12) \* (23 \* f(wi, ti) - 16 \* f(wim1, tim1) + 5 \* f(wim2, tim2));

return W;

}

Time W(t) Y(t)

1.000000000 1.000000000 1.000000000

1.100000000 1.004281728 1.004281728

1.200000000 1.014952314 1.014952314

1.300000000 1.029358153 1.029813689

1.400000000 1.046873328 1.047533919

1.500000000 1.066479068 1.067262354

1.600000000 1.087583973 1.088432687

1.700000000 1.109769403 1.110655052

1.800000000 1.132746808 1.133653557

1.900000000 1.156309412 1.157228433

2.000000000 1.180305938 1.181232218

### Adams-Bashforth Four-Step Explicit Method:

Changing the following:

// Adams-Bashforth Four-Step Explicit Method

long double AB4(long double ti, long double tim1, long double tim2,long double tim3, long double wi, long double wim1, long double wim2, long double wim3, long double h)

{

long double W = wi + (h / 24) \* (55 \* f(wi, ti) - 59 \* f(wim1, tim1) + 37 \* f(wim2, tim2) - 9 \* f(wim3, tim3));

return W;

}

Time W(t) Y(t)

1.000000000 1.000000000 1.000000000

1.100000000 1.004281728 1.004281728

1.200000000 1.014952314 1.014952314

1.300000000 1.029813689 1.029813689

1.400000000 1.047728159 1.047533919

1.500000000 1.067536537 1.067262354

1.600000000 1.088756964 1.088432687

1.700000000 1.110999698 1.110655052

1.800000000 1.134009621 1.133653557

1.900000000 1.157590178 1.157228433

2.000000000 1.181596968 1.181232218

### Adams-Bashforth Five-Step Explicit Method:

Changing the following:

// Adams-Bashforth Five-Step Explicit Method

long double AB5(long double ti, long double tim1, long double tim2, long double tim3, long double tim4, long double wi, long double wim1, long double wim2, long double wim3, long double wim4, long double h)

{

long double W = wi + (h / 720) \* (1901 \* f(wi, ti) - 2774 \* f(wim1, tim1) + 2616 \* f(wim2, tim2) - 1274 \* f(wim3, tim3) + 251 \* f(wim4, tim4));

return W;

}

Time W(t) Y(t)

1.000000000 1.000000000 1.000000000

1.100000000 1.004281728 1.004281728

1.200000000 1.014952314 1.014952314

1.300000000 1.029813689 1.029813689

1.400000000 1.047533919 1.047533919

1.500000000 1.067169823 1.067262354

1.600000000 1.088304888 1.088432687

1.700000000 1.110503968 1.110655052

1.800000000 1.133497057 1.133653557

1.900000000 1.157067114 1.157228433

2.000000000 1.181069238 1.181232218

## Problem 9.

The initial-value problem

Has solution

Applying the three-step Adams-Moulton method to this problem is equivalent to finding the fixed point of

With , obtain by functional iteration for using exact starting values and . At each step use to initially approximate .

### Adams-Moulton Three-Step Implicit Method:

Using the Runge-Kutta Method to get , we get the following using a program:

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

long double f(long double, long double);

long double Y(long double);

// Runge-Kutta method.

long double K1(long double, long double, long double);

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

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

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

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

long double AM3(long double, long double, long double, long double, long double, long double, long double, long double, long double);

int main()

{

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

ofstream outputFile;

outputFile.open("Adams-Moulton Three-Step Implicit Method, h = 0.01.txt");

long double a = 0, b = 0.2; // t boundary.

long double h = 0.01; // Mesh size.

long double y0 = 0, y1 = 0, y2 = 0, y3 = 0;;

long double yn = 0; // W(t) value.

long double t = a; // initial time.

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

// Initial values.

y0 = Y(t);

y1 = Y(t + h);

y2 = Y(t + 2 \* h);

// Heading for the file.

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

outputFile << showpoint << setprecision(10) << t << " " << y0 << " " << Y(t) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + h << " " << y1 << " " << Y(t + h) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + 2 \* h << " " << y2 << " " << Y(t + 2 \* h) << endl; // Inital points.

outputFile << endl;

while (t < b - 2 \* h)

{

k1 = K1(y2, t + 3 \* h, h);

k2 = K2(y2, t + 3 \* h, h, k1);

k3 = K3(y2, t + 3 \* h, h, k2);

k4 = K4(y2, t + 3 \* h, h, k3);

y3 = RK4(y2, t + 3 \* h, h, k1, k2, k3, k4);

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

yn = AM3(t + 3 \* h, t + 2 \* h, t + h, t, y3, y2, y1, y0, h);

outputFile << showpoint << setprecision(10) << t + 3 \* h << " " << yn << " " << Y(t + 3 \* h) << endl; // Write the information.

t += h; // Increase the time.

// Change the initial values.

y0 = y1;

y1 = y2;

y2 = yn;

}

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

return 0;

}

// Y(t).

long double Y(long double t)

{

long double Yt = 1 - log(1 - exp(1) \* t);

return Yt;

}

// Function of y and t.

long double f(long double y, long double t)

{

long double fxy = exp(y);

return fxy;

}

// Adams-Bashforth Three-Step Explicit Method

long double AM3(long double tip1, long double ti, long double tim1, long double tim2, long double wip1, long double wi, long double wim1, long double wim2, long double h)

{

long double W = wi + (h / 24) \* (9 \* f(wip1, tip1) + 19 \* f(wi, ti) - 5 \* f(wim1, tim1) + f(wim2, tim2));

return W;

}

//K values.

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

{

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

return K1;

}

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

{

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

return K2;

}

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

{

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

return K3;

}

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

{

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

return K4;

}

// Runge-Kutta Method.

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

{

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

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

return yi1;

}

The values that we get are:

Time W(t) Y(t)

0.0000000000 1.000000000 1.000000000

0.01000000000 1.027559106 1.027559106

0.02000000000 1.055899293 1.055899293

0.03000000000 1.085064092 1.085066130

0.04000000000 1.115104924 1.115109296

0.05000000000 1.146076050 1.146083084

0.06000000000 1.178036920 1.178046997

0.07000000000 1.211052869 1.211066434

0.08000000000 1.245195924 1.245213494

0.09000000000 1.280545745 1.280567924

0.1000000000 1.317190746 1.317218246

0.1100000000 1.355229425 1.355263086

0.1200000000 1.394771956 1.394812775

0.1300000000 1.435942112 1.435991283

0.1400000000 1.478879601 1.478938557

0.1500000000 1.523742916 1.523813391

0.1600000000 1.570712867 1.570796977

0.1700000000 1.619996971 1.620097317

0.1800000000 1.671834987 1.671954792

0.1900000000 1.726505948 1.726649255

0.2000000000 1.784337243 1.784509169

## Problem 10.

Use the Milne-Simpson Predictor-Corrector method to approximate the solution to the initial-value problems in Exercise 3.

### Explicit Milne’s Method:

### Implicit Simpson’s Method:

Using a Program:

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

long double f(long double, long double);

long double Y(long double);

long double EMM(long double, long double, long double, long double, long double, long double, long double, long double);

long double ISM(long double, long double, long double, long double, long double, long double, long double);

int main()

{

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

ofstream outputFile;

outputFile.open("Milne-Simpson Predictor-Corrector method, h = 0.1.txt");

long double a = 1, b = 2; // t boundary.

long double h = 0.1; // Mesh size.

long double y0 = 0, y1 = 0, y2 = 0, y3 = 0, y4 = 0;

long double yn = 0; // W(t) value.

long double t = a; // initial time.

// Initial values.

y0 = Y(t);

y1 = Y(t + h);

y2 = Y(t + 2 \* h);

y3 = Y(t + 3 \* h);

// Heading for the file.

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

outputFile << showpoint << setprecision(10) << t << " " << y0 << " " << Y(t) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + h << " " << y1 << " " << Y(t + h) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + 2 \* h << " " << y2 << " " << Y(t + 2 \* h) << endl; // Inital points.

outputFile << showpoint << setprecision(10) << t + 3 \* h << " " << y3 << " " << Y(t + 3 \* h) << endl; // Inital points.

outputFile << endl;

while (t < b - 3 \* h)

{

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

y4 = EMM(t + 3 \* h, t + 2 \* h, t + h, y3, y2, y1, y0, h);

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

yn = ISM(t + 4 \* h, t + 3 \* h, t + 2 \* h, y4, y3, y2, h);

outputFile << showpoint << setprecision(10) << t + 4 \* h << " " << yn << " " << Y(t + 4 \* h) << endl; // Write the information.

t += h; // Increase the time.

// Change the initial values.

y0 = y1;

y1 = y2;

y2 = y3;

y3 = yn;

}

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

return 0;

}

// Y(t).

long double Y(long double t)

{

long double Yt = t / (1 + log(t));

return Yt;

}

// Function of y and t.

long double f(long double y, long double t)

{

long double fxy = (y / t) - (y / t) \* (y / t);

return fxy;

}

// Explicit Milne’s Method.

long double EMM(long double ti, long double tim1, long double tim2, long double wi, long double wim1, long double wim2, long double wim3, long double h)

{

long double W = wim3 + (4. \* h / 3.) \* (2 \* f(wi, ti) - f(wim1, tim1) + 2 \* f(wim2, tim2));

return W;

}

// Implicit Simpson's Method.

long double ISM(long double tip1, long double ti, long double tim1, long double wip1, long double wi, long double wim1, long double h)

{

long double W = wim1 + (h / 3) \* (f(wip1, tip1) + 4 \* f(wi, ti) + f(wim1, tim1));

return W;

}

We get that:

Time W(t) Y(t)

1.000000000 1.000000000 1.000000000

1.100000000 1.004281728 1.004281728

1.200000000 1.014952314 1.014952314

1.300000000 1.029813689 1.029813689

1.400000000 1.047529173 1.047533919

1.500000000 1.067260299 1.067262354

1.600000000 1.088426888 1.088432687

1.700000000 1.110652524 1.110655052

1.800000000 1.133647494 1.133653557

1.900000000 1.157225802 1.157228433

2.000000000 1.181226088 1.181232218

## Program 14.

Derive Simpson’s Method by applying Simpson’s rule to the integral