МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Лабораторна робота №7

з курсу «Чисельні методи»

Виконав: студент 3 курсу

групи КА-23

Деундяк О.В.

Прийняла: Кузнєцова Н. В.

Київ – 2014р.

**Умова**

Знайти розв’язок задачі Коші для диференційного рівняння:

**Аналітичний розв’язок**

**Текст програми**

**main.cpp**

// y' = (2x + y - 3) / (x - 1)

// y(2) = 1

// answer: y = 2(x-1)log(x-1) + 1

#include "RKM.h"

#include "Adams.h"

#include <iostream>

#include <fstream>

#include <functional>

#include <vector>

int main()

{

std::function<double(double)> answer = [](double x)->double{return 2 \* (x - 1)\*log(x - 1) + 1; };

std::function<double(double, double)> deriv = [](double x, double y)->double{ return (2 \* x + y - 3) / (x - 1); };

std::ofstream output("output.txt");

// Runge-Kutta

{

int n = 10;

double x = 2, y = 1;

std::vector<double> res = RKM::Result(deriv, 2, 1);

double h = 1. / abs(n);

output << "RUNGE-KUTTA h = " << h << std::endl << std::endl;

for (double y : res)

{

output << x << '\t' << y << '\t' << answer(x) << '\t' << abs(y - answer(x)) << std::endl;

x += h;

}

}

output << std::endl;

// Adams

{

int n = 10;

double x = 2, y = 1;

std::vector<double> res = Adams::Result(deriv, x, y, n);

double h = 1./abs(n);

output << "ADAMS h = " << h << std::endl << std::endl;

for (double y : res)

{

output << x << '\t' << y << '\t' << answer(x) << '\t' << abs(y - answer(x)) << std::endl;

x += h;

}

}

}

**RKM.h**

#pragma once

#include <functional>

#include <vector>

class RKM

{

struct RKMResult

{

double h;

std::function<double(double, double)> deriv;

double x0, y0;

double operator()(double x);

};

public:

static std::function<double(double)> Solution(const std::function<double(double, double)>& func, double x0, double y0, int n = 10);

static std::vector<double> Result(const std::function<double(double, double)>& deriv, double x0, double y0, int n = 10);

};

**RKM.cpp**

#include "RKM.h"

std::function<double(double)> RKM::Solution(const std::function<double(double, double)>& func, double x0, double y0, int n /\*=10\*/)

{

RKMResult result;

result.h = 1./abs(n);

result.x0 = x0;

result.y0 = y0;

result.deriv = func;

return result;

}

double RKM::RKMResult::operator()(double xr)

{

if (xr > x0)

h = abs(h);

else

h = -abs(h);

double k1, k2, k3, k4;

double x = x0, y = y0;

while (abs(x - xr) > abs(h / 2))

{

k1 = deriv(x, y);

k2 = deriv(x + h / 2, y + k1 \* h / 2);

k3 = deriv(x + h / 2, y + k2 \* h / 2);

k4 = deriv(x + h, y + k3 \* h);

y += (k1 + 2\*k2 + 2\*k3 + k4) \* h/6;

x += h;

}

return y;

}

std::vector<double> RKM::Result(const std::function<double(double, double)>& deriv, double x0, double y0, int n /\*= 10\*/)

{

std::vector<double> result{ y0 };

double h = 1./abs(n);

double k1, k2, k3, k4;

double x = x0, y = y0;

while (abs(x - x0) < 1)

{

k1 = deriv(x, y);

k2 = deriv(x + h / 2, y + k1 \* h / 2);

k3 = deriv(x + h / 2, y + k2 \* h / 2);

k4 = deriv(x + h, y + k3 \* h);

y += (k1 + 2 \* k2 + 2 \* k3 + k4) \* h / 6;

x += h;

result.push\_back(y);

}

return result;

}

**Adams.h**

#pragma once

#include <functional>

#include <vector>

class Adams

{

public:

static std::vector<double> Result(const std::function<double(double, double)>& deriv, double x0, double y0, int n = 10);

};

**Adams.cpp**

#include "Adams.h"

#include "RKM.h"

std::vector<double> Adams::Result(const std::function<double(double, double)>& deriv, double x0, double y0, int n /\*= 10\*/)

{

double h = 1./abs(n);

double f[4];

std::vector<double> y;

// Getting initial values using Runge-Kutta method

double x = x0;

std::function<double(double)> rk = RKM::Solution(deriv, x0, y0, n);

for (size\_t i = 0; i < 4; i++)

{

y.push\_back(rk(x));

x += h;

}

for (int i = 4; i <= n; i++)

{

y.push\_back(y[i-1] + h/24\*(55\*deriv(x - h, y[i-1]) - 59\*deriv(x - 2\*h, y[i-2]) + 37\*deriv(x - 3\*h, y[i-3]) - 9\*deriv(x - 4\*h, y[i-4])));

x += h;

}

return y;

}

**Результати роботи програми**

RUNGE-KUTTA h = 0.1

2 1 1 0

2.1 1.20968 1.20968 5.43031e-007

2.2 1.43757 1.43757 9.70056e-007

2.3 1.68215 1.68215 1.32161e-006

2.4 1.94212 1.94212 1.62237e-006

2.5 2.21639 2.2164 1.88793e-006

2.6 2.50401 2.50401 2.12846e-006

2.7 2.80413 2.80414 2.35083e-006

2.8 3.11603 3.11603 2.55974e-006

2.9 3.43904 3.43904 2.75851e-006

3 3.77259 3.77259 2.94953e-006

ADAMS h = 0.1

2 1 1 0

2.1 1.20968 1.20968 5.43031e-007

2.2 1.43757 1.43757 9.70056e-007

2.3 1.68215 1.68215 1.32161e-006

2.4 1.94214 1.94212 1.95543e-005

2.5 2.21643 2.2164 3.80655e-005

2.6 2.50406 2.50401 5.15547e-005

2.7 2.8042 2.80414 6.33901e-005

2.8 3.11611 3.11603 7.39506e-005

2.9 3.43913 3.43904 8.33415e-005

3 3.77268 3.77259 9.19056e-005

**Висновок**

На графіку різниці між аналітичним розв’язком та обчисленими значенням не помітно. h = 0.1

Графік похибок при h = 0.1

По графіку видно, що метод Рунге-Кутта більш точний.

Бачимо, що чим більше розбиттів тим точніше діють методи.