БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

Отчет по лабораторной работе №1

«Методы Рунге-Кутте и Адамса»

Студент: Шелег Владислава Михайловна

3 курс 9 группа

Преподаватель: Полещук Максим Александрович

МИНСК

2017

**Задание 1**

* Методом Рунге-Кутты четвёртого порядка точности RK4 найти приближенное решение задачи Коши дифференциального уравнения:

*; ;*

*; .*

Локальную погрешность на шаге ограничить величиной , величину шага определять с помощью правила Рунге, число итераций применения правила ограничить небольшим числом. Первоначально положить длину шага равной .

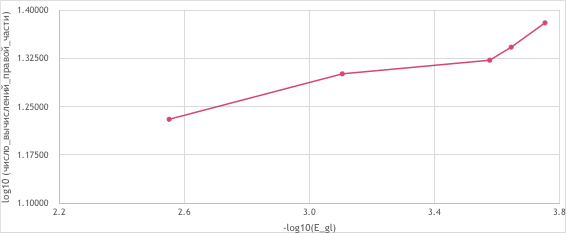
Некоторые дополнительные условия*:*

Результат.

*.*

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **h\_n** | **y\_n** | **|y\_n - u(x\_n) |** |
| 0 | 0.4 | 2.71828 | 0 |
| 1 | 0.180561 | 2.71828 | 0 |
| 2 | 0.198391 | 2.712 | 0.000509262 |
| 3 | 0.0549731 | 2.712 | 0.000509262 |
| 4 | 0.0826484 | 2.63356 | 0.000538349 |
| 5 | 0.0580544 | 2.63356 | 0.000538349 |
| 6 | 0.0610985 | 1.89215 | 0.000505567 |
| 7 | 0.0453134 | 1.89215 | 0.000505567 |
| 8 | 0.0574201 | 0.59655 | 0.000903308 |
| 9 | 0.0208966 | 0.59655 | 0.000903308 |
| 10 | 0.0265819 | 0.368809 | 0.000524819 |
| 11 | 0.0165427 | 0.368809 | 0.000524819 |
| 12 | 0.0213892 | 0.582972 | 0.000723362 |
| 13 | 0.0111209 | 0.582972 | 0.000723362 |
| 14 | 0.0120144 | 1.34604 | 0.00139797 |
| 15 | 0.000523388 | 2.69206 | 0.00224757 |
| 16 | 0 | 2.70921 | 0.00226164 |

* Построить график зависимости от значений , выполнив серию запусков алгоритма метода Рунге-Кутты с ограничением локальной погрешности на шаге величинами , , .

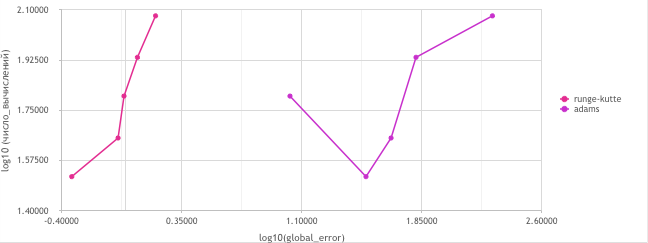
****

**Задание 2.**

* Интерполяционным методом Адамса четвёртого порядка точности найти приближённое решение задачи Коши дифференциального уравнения из задания 1 на сетке узлов с фиксированным шагом , вычислив стартовые значения с использованием метода Рунге-Кутты. В качестве предиктора для интерполяционного метода Адамса применить один шаг экстраполяционного метода Адамса четвёртого порядка точности.

Результат.

|  |  |  |  |
| --- | --- | --- | --- |
| n | y\_n | n | y\_n |
| 0 | 2,71828 | 12 | 2.71823 |
| 1 | 2,71828 | 13 | 2.71802 |
| 2 | 2,71828 | 14 | 2.71716 |
| 3 | 2,71828 | 15 | 2.71386 |
| 4 | 2,71828 | 16 | 2.70233 |
| 5 | 2,71828 | 17 | 2.66539 |
| 6 | 2,71828 | 18 | 2.55763 |
| 7 | 2,71828 | 19 | 2.28249 |
| 8 | 2,71828 | 20 | 1.72805 |
| 9 | 2,71828 | 21 | 0.99376 |
| 10 | 2,71828 | 22 | 0.447357 |
| 11 | 2,71827 | 23 | 0.387682 |



* Построить графики зависимости от значений , выполнив серию запусков алгоритмов метода Рунге-Кутты четвёртого порядка точности RK4 без адаптивного выбора шага по правилу Рунге (фиксировать длину шага) и интерполяционного метода Адамса четвёртого порядка точности для длин шагов , . Представить два графика на одном рисунке. Сравнить эффективность методов по полученным графикам.

**Листинг программы**

**Файл rungekutte.h:**

#ifndef MCHA\_1\_RUNGEKUTTE\_H

#define MCHA\_1\_RUNGEKUTTE\_H

#include <list>

#include "conditions.h"

namespace rungekutte {

float err\_coeff = (float) (1 / 15.0); // == (1 / 2^p - 1) && (p == 4)

float rungekutte\_next\_y(float x\_prev, float y\_prev, float h) {

float k1 = derivative(x\_prev, y\_prev);

float k2 = derivative(x\_prev + h / 2.0f, y\_prev + h \* k1 / 2.0f);

float k3 = derivative(x\_prev + h / 2.0f, y\_prev + h \* k2 / 2.0f);

float k4 = derivative(x\_prev + h, y\_prev + h \* k3);

float y\_next = y\_prev + (k1 + 2.0f \* k2 + 2.0f \* k3 + k4) \* h / 6.0f;

return y\_next;

}

std::list<float> adaptive\_runge\_step(float x, float y, float h, float eps) {

// auxiliary function to avoid duplication

float y2 = rungekutte\_next\_y(x + h,

rungekutte\_next\_y(x, y, h),

h);

float y2\_double\_step = rungekutte\_next\_y(x, y, 2 \* h);

float err = fabsf(y2 - y2\_double\_step) \* err\_coeff;

float delta = powf(eps / err, 1 / (p + 1));

float h\_new = runge\_alpha \* delta \* h;

if (delta < 1) {

h = h\_new;

} else {

x = x + h \* 2;

y = y2;

h = std::min(h\_new, (end - x) / 2.0f);

}

return {x, y, h}; //it's just a list

}

method\_res adaptive\_runge\_solve(float x0, float y0, float h0, float eps) {

std::vector<step\_res> log;

float global\_error = -1.0f;

float x = x0;

float y = y0;

float h = h0;

int n;

for (n = 0; x < end; ++n) {

auto step = rungekutte::adaptive\_runge\_step(x, y, h, eps);

x = \*step.begin();

y = \*std::next(step.begin(), 1);

h = \*std::next(step.begin(), 2);

float local\_error = fabsf(y - solution(x));

global\_error = std::max(global\_error, local\_error);

log.emplace\_back(n, h, y, local\_error);

}

if (log.size() > 2 \* log\_count) { // left only 20 first and 20 last elements

log.erase(log.begin() + log\_count, log.end() - log\_count);

}

return method\_res(log, n, global\_error);

}

std::vector<std::pair<float, float>> get\_rungekutte\_graph(float x0, float y0, float h0) {

std::vector<std::pair<float, float >> graph;

for (int q = 0; q < q\_max; ++q) {

float curr\_eps = get\_eps(q);

float x = x0;

float y = y0;

float h = h0;

float global\_error = -1;

int n;

for (n = 0; x < end; ++n) {

auto step = rungekutte::adaptive\_runge\_step(x, y, h, curr\_eps);

x = \*step.begin();

y = \*std::next(step.begin(), 1);

h = \*std::next(step.begin(), 2);

float u = solution(x);

global\_error = std::max(global\_error, fabsf(y - u));

}

graph.emplace\_back(log10(n), -log10f(global\_error));

}

return graph;

};

float get\_error\_fix\_step\_rk(float x0, float y0, float h) {

float global\_error = -1.0f;

float x = x0;

float y = y0;

while (x < end) {

y = rungekutte\_next\_y(x, y, h);

global\_error = std::max(global\_error, fabsf(y - solution(x)));

x += h;

}

return global\_error;

}

}

#endif //MCHA\_1\_RUNGEKUTTE\_H

**Файл adams.h:**

#ifndef MCHA\_1\_ADAMS\_H

#define MCHA\_1\_ADAMS\_H

#include "conditions.h"

#include "rungekutte.h"

namespace adams {

method\_res adams\_solve(float x0, float y0, float h) {

std::vector<step\_res> log;

float global\_error = -1.0f;

int steps\_count = (int) ((end - begin) / h);

std::vector<float> x(steps\_count);

std::vector<float> y(steps\_count);

std::vector<float> derivs(steps\_count);

for (int i = 0; i < steps\_count; ++i) {

x[i] = x0 + i \* h;

}

// initialization with rungekutte

y[0] = y0;

y[1] = rungekutte::rungekutte\_next\_y(x[0], y[0], h);

y[2] = rungekutte::rungekutte\_next\_y(x[1], y[1], h);

y[3] = rungekutte::rungekutte\_next\_y(x[2], y[2], h);

for (int i = 0; i < 4; ++i) {

derivs[i] = derivative(x[i], y[i]);

float local\_error = fabsf(y[i] - solution(x[i]));

log.emplace\_back(i, h, y[i], local\_error);

global\_error = std::max(global\_error, local\_error);

}

for (int i = 4; i < steps\_count; ++i) {

y[i] = y[i - 1] + h \* (55.0f \* derivs[i - 1] / 24.0f - 59.0f \* derivs[i - 2] / 24.0f + 37.0f \* derivs[i - 3] / 24.0f -

3.0f \* derivs[i - 4] / 8.0f);

derivs[i] = derivative(x[i], y[i]);

y[i] = y[i - 1] + h \* (3.0f \* derivs[i] / 8.0f + 19.0f \* derivs[i - 1] / 24.0f -

5.0f \* derivs[i - 2] / 24.0f + 1.0f \* derivs[i - 3] / 24.0f);

float local\_error = fabsf(y[i] - solution(x[i]));

log.emplace\_back(i, h, y[i], local\_error);

global\_error = std::max(global\_error, local\_error);

}

if (log.size() > 2 \* log\_count) { // left only 20 first and 20 last elements

log.erase(log.begin() + log\_count, log.end() - log\_count);

}

return method\_res(log, steps\_count, global\_error);

}

}

#endif //MCHA\_1\_ADAMS\_H

**Файл conditions.h:**

#ifndef MCHA\_1\_CONDITIONS\_H

#define MCHA\_1\_CONDITIONS\_H

#include <sstream>

float g = 9.0f;

float s = 17.0f;

int log\_count = 20;

float begin = 0.0f;

float end = 1.2f;

float p = 4.0f;

int q\_max = 5;

float runge\_alpha = 0.8f;

float derivative(float x, float u) {

float tmp = g + 0.05f \* s;

return -(tmp \* powf(x, tmp - 1.0f) \* u \* sinf(powf(x, tmp)));

}

float solution(float x) {

return expf(cosf(powf(x, g + 0.05f \* s)));

}

float get\_h(int q) {

return powf(5.0f, (float) q / q\_max) \* 0.01f;

}

float get\_eps(int q) {

return powf(10.0f, (float) q / q\_max - 4.0f);

}

float x0 = begin;

float y0\_ = solution(x0); float eps = 0.001f;

float h0 = 0.2f;

float adams\_fix\_h = 0.05f;

struct step\_res {

int n;

float h;

float y;

float error;

step\_res(int n, float h, float y, float error) : n(n), h(h), y(y), error(error) {}

std::string to\_string\_rk() {

std::stringstream res;

res << "step\_" << n << ": h = " << h << ", y = " << y <<

", err = " << error;

return res.str();

}

std::string to\_string\_adams() {

std::stringstream res;

res << "step\_" << n << ", y = " << y ;

return res.str();

}

};

struct method\_res {

std::vector<step\_res> log;

int n\_count;

float global\_error;

method\_res(std::vector<step\_res> log, int n\_count, float global\_error) :

log(std::move(log)),

n\_count(n\_count),

global\_error(global\_error) {}

};

#endif //MCHA\_1\_CONDITIONS\_H

**Файл main.cpp:**

#include <cmath>

#include <iostream>

#include <vector>

#include <map>

#include <fstream>

#include "conditions.h"

#include "rungekutte.h"

#include "adams.h"

std::ofstream fout("out.txt");

void task1();

void task2();

std::map<float, std::pair<float, float>> get\_rk\_vs\_adams\_graph(float x0, float y0);

int main() {

std::cout << "task1:" << std::endl;

fout << "task1:" << std::endl;

task1();

std::cout << "\n\ntask2:" << std::endl;

fout << "\n\ntask2:" << std::endl;

task2();

fout.close();

return 0;

}

void task1() {

auto res = rungekutte::adaptive\_runge\_solve(x0, y0\_, h0, eps);

std::cout << "N = " << res.n\_count << ", E\_gl = " << res.global\_error << std::endl;

fout << "N = " << res.n\_count << ", E\_gl = " << res.global\_error << std::endl;

for (auto l : res.log) {

std::cout << l.to\_string\_rk() << std::endl;

fout << l.to\_string\_rk() << std::endl;

}

auto graph = rungekutte::get\_rungekutte\_graph(x0, y0\_, h0);

for (auto p : graph) {

std::cout << p.first << " -> " << p.second << std::endl;

fout << p.first << "; " << p.second << std::endl;

}

}

void task2() {

auto adams\_res = adams::adams\_solve(x0, y0\_, adams\_fix\_h);

std::cout << "N = " << adams\_res.n\_count << ", E\_gl = " << adams\_res.global\_error << std::endl;

fout << "N = " << adams\_res.n\_count << ", E\_gl = " << adams\_res.global\_error << std::endl;

for (auto l : adams\_res.log) {

std::cout << l.to\_string\_adams() << std::endl;

fout << l.to\_string\_adams() << std::endl;

}

auto res = get\_rk\_vs\_adams\_graph(x0, y0\_);

std::cout << "graph: log10(n\_count) -> runge vs adams (-log10(global\_err)):" << std::endl;

fout << "graph: log10(n\_count) -> runge vs adams (-log10(global\_err)):" << std::endl;

for (auto r : res) {

std::cout << r.first << " -> " << r.second.first << " vs " << r.second.second << std::endl;

fout << r.first << "; " << r.second.first << "; " << r.second.second << std::endl;

}

}

std::map<float, std::pair<float, float>> get\_rk\_vs\_adams\_graph(float x0, float y0) {

std::map<float, std::pair<float, float>> res;

for (int q = 0; q < q\_max; ++q) {

float curr\_h = get\_h(q);

float adams\_error = adams::adams\_solve(x0, y0, curr\_h).global\_error;

float runge\_error = rungekutte::get\_error\_fix\_step\_rk(x0, y0, curr\_h);

int steps\_count = (int) ((end - begin) / curr\_h);

auto p = std::make\_pair(-log10f(runge\_error), -log10f(adams\_error));

res.insert(std::make\_pair(log10f(steps\_count), p));

}

return res;

}