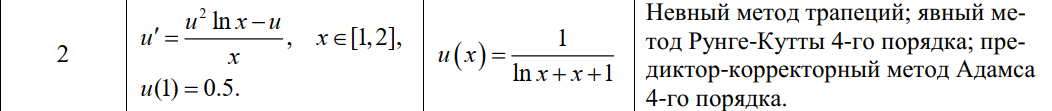
**Лабораторная работа №4**

**«Численные методы решения задачи Коши»**

Башев Ян 11 группа

Вариант 2



Простейшее ОДУ имеет вид:

Задача Коши заключается в нахождении решения (u = u(x)), удовлетворяющего уравнению выше и начальному условию:

**Неявный метод трапеций**

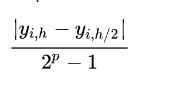
Начальное приближение метода Ньютона

Итерационный процесс

**Метод Рунге-Кутты 4-го порядка**

**Предиктор-корректорный метод Адамса 4-го порядка**

**Правило Рунге оценки погрешности**

Формула дает погрешность решения для шага . Под 𝑝 понимается порядок точности использованного численного метода.

Таблица

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *i* |  | Точное решение | Численное решение задачи Коши с шагом 0.1 | | |
| Метод 1 | Метод 2 | Метод 3 |
|  |  |  |  |
| 0 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| 1 | 1.1 | 0.455516 | 0.455207 | 0.455516 | 0.455516 |
| 2 | 1.2 | 0.419759 | 0.419279 | 0.419758 | 0.419758 |
| 3 | 1.3 | 0.390265 | 0.389688 | 0.390264 | 0.390264 |
| 4 | 1.4 | 0.365434 | 0.364803 | 0.365434 | 0.365423 |
| 5 | 1.5 | 0.344179 | 0.343518 | 0.344179 | 0.344162 |
| 6 | 1.6 | 0.325733 | 0.325056 | 0.325732 | 0.325713 |
| 7 | 1.7 | 0.309537 | 0.308855 | 0.309537 | 0.309517 |
| 8 | 1.8 | 0.295178 | 0.294495 | 0.295178 | 0.295157 |
| 9 | 1.9 | 0.282338 | 0.281659 | 0.282338 | 0.282317 |
| 10 | 2 | 0.270772 | 0.270099 | 0.270772 | 0.270751 |
|  | | | 0.00068258 | 1.60202e-07 | 2.10265e-05 |
| Оценка погрешности по правилу Рунге | | | 0.000681733 | 1.62337e-07 | - |

Выводы

В рамках лабораторной работы были рассмотрены следующие методы: неявный метод трапеций (порядок 2), метод Рунге-Кутты 4-го порядка, предиктор-корректорный метод Адамса 4-го порядка. Лучше всего проявил себя метод Рунге-Кутты, показав наилучшую точность. Хуже всего – неявный метод трапеций.

Исходный код

#include <iostream>

#include <vector>

#include <fstream>

const double a = 1;

const double b = 2;

const double h = 0.1;

const double u0 = 0.5;

const double EPS = 1e-7;

double f(double x, double u)

{

return (u \* u \* log(x) - u) / x;

}

double fproizv(double x, double u)

{

return (2. \* u \* log(x) - 1.) / x;

}

double solution(double x)

{

return 1. / (log(x) + x + 1);

}

using namespace std;

vector<double> simpTrapMethod(double h)

{

vector<double> y((b - a) / h + 1);

y[0] = u0;

for (int i = 1; i < y.size(); i++)

{

double xi = a + h \* (i - 1);

double ystart = 0;

double ynext = y[i - 1];

do

{

ystart = ynext;

ynext = ystart - (ystart - y[i - 1] - h / 2 \* (f(xi, y[i - 1]) + f(xi + h, ystart))) / (1. - h \* fproizv(xi + h, ystart) / 2);

} while (abs(ynext - ystart) > EPS);

y[i] = ynext;

y[i] = y[i - 1] + h \* (f(xi, y[i - 1]) + f(xi + h, y[i])) / 2;

}

return y;

}

vector<double> runge\_cute(double h)

{

vector<double> y((b - a) / h + 1);

y[0] = u0;

for (int i = 1; i < y.size(); i++)

{

double xi = a + h \* (i - 1);

double k1 = f(xi, y[i - 1]);

double k2 = f(xi + h / 2, y[i - 1] + h \* k1 / 2);

double k3 = f(xi + h / 2, y[i - 1] + h \* k2 / 2);

double k4 = f(xi + h, y[i - 1] + h \* k3);

y[i] = y[i - 1] + h \* (1. / 6 \* k1 + 2. / 6 \* k2 + 2. / 6 \* k3 + 1. / 6 \* k4);

}

return y;

}

vector<double> adams4()

{

vector<double> y((b - a) / h + 1);

y[0] = u0;

y[1] = runge\_cute(h)[1];

y[2] = runge\_cute(h)[2];

y[3] = runge\_cute(h)[3];

for (int i = 4; i < y.size(); i++)

{

double xi = a + h \* (i - 1);

double yR = y[i - 1] + h / 24 \* (55. \* f(xi, y[i - 1]) - 59. \* f(xi - h, y[i - 2]) +

37. \* f(xi - 2. \* h, y[i - 3]) - 9. \* f(xi - 3. \* h, y[i - 4]));

y[i] = y[i - 1] + h / 24 \* (9. \* f(xi + h, yR) + 19. \* f(xi, y[i - 1]) - 5. \* f(xi - h, y[i - 2]) + f(xi - 2. \* h, y[i - 3]));

}

return y;

}

vector<double> correctSolution()

{

vector<double> y((b - a) / h + 1);

for (int i = 0; i < y.size(); i++)

{

double xi = a + h \* i;

y[i] = solution(xi);

}

return y;

}

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(0);

cout.tie(0);

ofstream trapFile("trap.txt");

ofstream rungeFile("runge.txt");

ofstream adamsFile("adams.txt");

auto u = correctSolution();

auto trap = simpTrapMethod(h);

auto runge = runge\_cute(h);

auto adams = adams4();

auto trap2 = simpTrapMethod(h \* 2);

auto runge2 = runge\_cute(h \* 2);

double maxTrap = 0;

double maxRunge = 0;

double maxAdams = 0;

double rungeTrap = 0;

double rungeRunge = 0;

for (int i = 0; i < u.size(); i++)

{

cout<<i<<' '<< a + h \* i << ' ' << u[i] << ' ' << trap[i] << ' ' << runge[i] << ' ' << adams[i] << '\n';

maxTrap = max(abs(u[i] - trap[i]), maxTrap);

maxRunge = max(abs(u[i] - runge[i]), maxRunge);

maxAdams = max(abs(u[i] - adams[i]), maxAdams);

trapFile << a + h \* i << ' ' << trap[i] << '\n';

rungeFile << a + h \* i << ' ' << runge[i] << '\n';

adamsFile << a + h \* i << ' ' << adams[i] << '\n';

}

for (int i = 0; i < u.size() / 2 + (u.size() % 2); i++)

{

rungeTrap = max(rungeTrap, abs(trap2[i] - trap[2 \* i]) / 3);

rungeRunge = max(rungeRunge, abs(runge2[i] - runge[2 \* i]) / 15);

}

cout << "max(abs(u(xi) - yi)): " << maxTrap << ' ' << maxRunge << ' ' << maxAdams << '\n';

cout << "Runge rool: " << rungeTrap << ' ' << rungeRunge << " - \n";

}