МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский физико-технический институт
(национальный исследовательский университет)
Кафедра вычислительной математики

# Метод Рунге-Кутты решения задачи Коши для обыкновенных дифференциальных уравнений первого порядка

Лабораторная работа по курсу
вычислительная математика
Задание 5

Выполнил: студент группы Б04-856:
Крылов Александр

г. Долгопрудный
2020 год

# Содержание

# 1. Задача Коши

Рассмотрим задачу Коши на двумерном пространстве на открытом множестве $G$:

$$\begin{cases} \frac{dy}{dx} = f(x, y) \\ y(x_0) = y_0 \\ x \in [x_0, x_1] \end{cases} \tag{1}$$

Согласно теореме существования и единственности для любой точки $(x_0, y_0) \in G$ найдется решение $y = y(x)$ задачи, притом единственное. Нахождение общего решения дифферекнциального уравнения задачи (1) аналитически возможно лишь для некоторых частных случаев. Для нахождения чиленных решений используются методы Рунге-Кутты. В этой работе импользуется один из этих метода второго порядка: метод Эйлера с пересчетом.

# 2. Метод Эйлера с пересчетом

Простейшим неявным методом Рунге — Кутты является модифицированный метод Эйлера «с пересчётом». Он задаётся формулой:

$$y_{n+1} = y_n + h\frac{f(x_n, y_n) + f(x_{n+1}, y_{n+1})}{2}, \tag{2}$$

где $y_{n+1} = y_n + hf(x_n, y_n)$

Для его реализации на каждом шаге необходимы как минимум две итерации (и два вычисления функции).

Модифицированный метод Эйлера «с пересчётом» имеет второй порядок точности.

# 3. Достижение заданной точности

Получить приближенное представление о допущеннрй погрешности можно, если известен порядок точности используемого метода. Для этого решают задачу на сетке с шагом 2h и h, тогда:

$$y^{2h} = y_0 + C(2h)^k, y^h = y_0 + C(h)^k, \tag{3}$$

где k - порядок точности метода. Вычитая второе уравнение из первого, получим:

$$y^{2h} - y^h = ch^k(2^k - 1) \tag{4}$$

Тогда имеем оценку точности:

$$\mid y^h - y_0 \mid = \frac{\mid y^{2h} - y^h \mid}{(2^k - 1)} \leq \epsilon \tag{5}$$

Из оценки (номер уравнения точности) получим алгоритм достижения заданной точности. Если не выполняется условие (номер уравнения точности), то необходимо увеличить число шагов в 2 раза и снова проверить (номер уравнения точности). Так поступаем до тех пор, пока заданная точность не будет достигнута.

# 4. Реализация и результаты

Реализация метода Эйлера с пересчетом для решения задачи

$$\begin{cases} x(2x^2yln(y)+1)\frac{dy}{dx} = 2y \\ y(1) = 1 \\ x \in [1, 1.2] \end{cases} \tag{6}$$

была приведена реализация на языке программирования C++, код приведен в приложении. В ней сначала производится поиск такого количества шагов, которое будет удовлетворять заданной точности $\epsilon = 10^{-4}$. Затем производится расчет таблицы (x, y) на всех шагах выбранной сетки. Таблица сохраняется в отдельный файл. В конце происходит вывод значений функции на равномерной сетке с шагом h и 2h, с разностью между ними. Вывод также указан в приложении. Данная программа позволяет при соответствующих изменениях в коде может быть использавана для решения любой задачи Коши (1), и для любого другого метода Рунге-Кутты второго порядка.

# 5. Вывод

# 6. Приложения

## 6.1. Код программы

```
//Soft works for 2nd order Runge-Kutta methods
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdbool.h>
#define FTYPE long double

//Settings
//y' = DYX function
#define FUNC (2 * y / (x * ((2 * pow(x, 2) * y * log(y)) - 1)))
//y(X0) = Y0
#define X0_VALUE 1.0
#define Y0_VALUE 1
// BORDER_LEFT < X < BORDER_RIGHT
#define BORDER_LEFT 1.0
#define BORDER_RIGHT 1.2
#define EPSILON 1E-4


FTYPE func(FTYPE x, FTYPE y) {
return FUNC;
}
//Get f_1 and f_2
FTYPE* evaluateFunc(int funcNum, FTYPE xn, FTYPE yn, FTYPE MethodMatrix[3][3], FTYPE h)
FTYPE* f = (FTYPE*)malloc(sizeof(FTYPE) * 2);
for (int i = 0; i < 2; i++) {
```

```
f[i] = 0;
}
f[0] = func(xn, yn);
for (int i = 1; i < funcNum - 1; i++) {
FTYPE x = xn + MethodMatrix[i][0] * h;
FTYPE y = yn;
for (int j = 1; j < i + 2; j++) {
y += MethodMatrix[i][j] * f[j - 1] * h;
}
f[i] = func(x, y);
}
return f;
}
//Get y_n+1 from y_n
FTYPE evaluateY(FTYPE yn, FTYPE methodMatrix[3][3], FTYPE* functions, int length,
FTYPE h) {
FTYPE ynp1 = yn;
for (int i = 1; i < length; i++) {
FTYPE a = methodMatrix[length - 1][i];
ynp1 += functions[i - 1] * methodMatrix[length - 1][i] * h;
}
return ynp1;
}
//Reassign X0, Y0, borders and epsilon
FTYPE X0 = X0_VALUE;
FTYPE Y0 = Y0_VALUE;

FTYPE borders[2] = { BORDER_LEFT, BORDER_RIGHT };

FTYPE epsilon = EPSILON;

//Get h value
FTYPE evalGridStepH(int steps) {
return (borders[1] - borders[0]) / steps;
}
//check if our error is less than epsilon
bool checkIfErrorAcceptable(FTYPE delta, int method) {
FTYPE val = (fabsl(delta)) / (powl(2.0, method) - 1);
return val <= epsilon ? false : true;
}


int main(int argc, const char* argv[]) {
//function on h grid
FTYPE* f = NULL;
//function on 2h grid
FTYPE* fProxy = NULL;
//Butcher tableau
FTYPE methodMatrix[3][3] = {
{0, 0, 0},
```

```
{1, 1, 0},
{0.5, 0.5, 0}
};
//points in uniform grid
FTYPE calcPoints[11];
//y in uniform grid with h step
FTYPE valuesInPoints[11];
//y in uniform grid with 2h step
FTYPE valuesInPointsProxy[11];
//y(2h) - y(h)
FTYPE deltas[11];
//initial setup
for (int i = 0; i < 11; i++) {
calcPoints[i] = borders[0] + ((borders[1] - borders[0]) / 10) * i;
}
valuesInPoints[0] = Y0;
valuesInPointsProxy[0] = Y0;
int stepM = 3;
//counting appropriate number of steps
int steps = 5;
FTYPE maxDelta;
FTYPE h = 0.0;
FTYPE x = X0;
FTYPE xProxy = X0;
FTYPE value = Y0;
FTYPE valueProxy = Y0;
//finding approproate step
do {
x = X0;
xProxy = X0;
value = Y0;
valueProxy = Y0;
for (int i = 0; i <= steps; i++) {
h = evalGridStepH(steps);
x = h * i + X0;
xProxy = 2 * h * i + X0;
for (int j = 0; j < 11; j++) {
if (x == calcPoints[j]) {
valuesInPoints[j] = value;
}
if (xProxy == calcPoints[j]) {
valuesInPointsProxy[j] = valueProxy;
break;
}
}
//count values for output, will be changed if step is too small
f = evaluateFunc(stepM, x, value, methodMatrix, h);
fProxy = evaluateFunc(stepM, xProxy, valueProxy, methodMatrix, 2 * h);
value = evaluateY(value, methodMatrix, f, stepM, h);
valueProxy = evaluateY(valueProxy, methodMatrix, fProxy, stepM, 2 * h);
```

```
}
free(f);
free(fProxy);
for (int i = 0; i < 11; i++) {
deltas[i] = fabsl(valuesInPointsProxy[i] - valuesInPoints[i]);
}
maxDelta = deltas[0];
for (int i = 1; i < 11; i++) {
if (maxDelta < deltas[i]) {
maxDelta = deltas[i];
}
}
steps = steps * 2;
} while (checkIfErrorAcceptable(maxDelta, stepM));
FTYPE step = evalGridStepH(steps/2);
FILE* results = fopen( "results.txt", "w+");
FTYPE finalValue = Y0;
fprintf(results, "x = %Lf\t y = %Lf\n", X0, finalValue);
for (int i = 1; i <= steps/2; i++) {
f = evaluateFunc(stepM, step * i + X0, Y0, methodMatrix, step);
finalValue = evaluateY(finalValue, methodMatrix, f, stepM, step);
fprintf(results, "x = %Lf\t y = %Lf\n", X0 + i * step, finalValue);
free(f);
}
fclose(results);
printf("Calculated with %d steps\nAccuracy: %Lf\nh=%Lf\nDiff=%Le\n\n", steps / 2,
epsilon, h, maxDelta);
for (int i = 0; i < 11; i++) {
printf("%d) x=%Lf\n\ty(h)=%Lf\n\ty(2h)=%Lf\n\t--------------\n\ty(2h)-y(h)=%Lf\n\n",
i + 1, calcPoints[i], valuesInPoints[i], valuesInPointsProxy[i],
valuesInPointsProxy[i] - valuesInPoints[i]);
}
system("pause");
return 0;
}
```

## 6.2. Вывод консоли

```
    Calculated with 80 steps
Accuracy: 0.000100
h=0.002500
Diff=4.925484e-04

1) x=1.000000
        y(h)=1.000000
        y(2h)=1.000000
        --------------
        y(2h)-y(h)=0.000000

2) x=1.020000
```

```
        y(h)=0.980675
        y(2h)=0.980582
        --------------
        y(2h)-y(h)=-0.000093

3) x=1.040000
        y(h)=0.962779
        y(2h)=0.962609
        --------------
        y(2h)-y(h)=-0.000170

4) x=1.060000
        y(h)=0.946150
        y(2h)=0.945915
        --------------
        y(2h)-y(h)=-0.000235

5) x=1.080000
        y(h)=0.930653
        y(2h)=0.930362
        --------------
        y(2h)-y(h)=-0.000291

6) x=1.100000
        y(h)=0.916172
        y(2h)=0.915834
        --------------
        y(2h)-y(h)=-0.000338

7) x=1.120000
        y(h)=0.902608
        y(2h)=0.902230
        --------------
        y(2h)-y(h)=-0.000378

8) x=1.140000
        y(h)=0.889877
        y(2h)=0.889464
        --------------
        y(2h)-y(h)=-0.000413

9) x=1.160000
        y(h)=0.877903
        y(2h)=0.877459
        --------------
        y(2h)-y(h)=-0.000443

10) x=1.180000
        y(h)=0.866620
        y(2h)=0.866151
```

```
        --------------
        y(2h)-y(h)=-0.000470

11) x=1.200000
        y(h)=0.855972
        y(2h)=0.855479
        --------------
        y(2h)-y(h)=-0.000493
```