Министерство цифрового развития связи и массовых коммуникаций Российский Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

“Сибирский государственный университет телекоммуникаций и информатики” (СибГУТИ)

Кафедра прикладной математики и кибернетики

Курсовая работа

по дисциплине «Вычислительная математика»

Вариант 8

Выполнил:

Студент группы ИП-316

Иванов А.Д.

Проверил:

Ассистент кафедры ПМиК

Истомина А. С.

Новосибирск, 2025

Содержание

[1. Постановка задачи 3](#_Toc198209707)

[2.Теоретические сведения 4](#_Toc198209708)

[3.Описание алгоритма 6](#_Toc198209709)

[4.Результат работы программы 10](#_Toc198209710)

[5.Листинг 11](#_Toc198209711)

# Постановка задачи

Решить дифференциальное уравнение на интервале [0,1] методами Рунге кутта 2 и 4 порядка с точностью стартовый шаг h=0,1

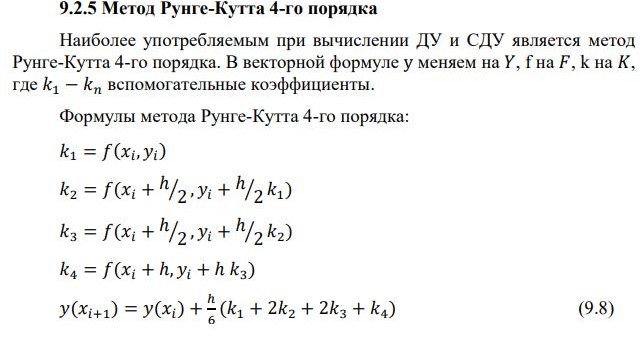
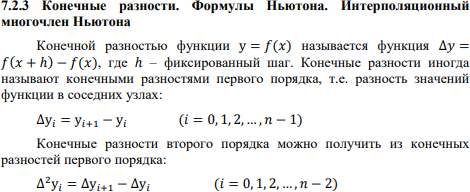
+

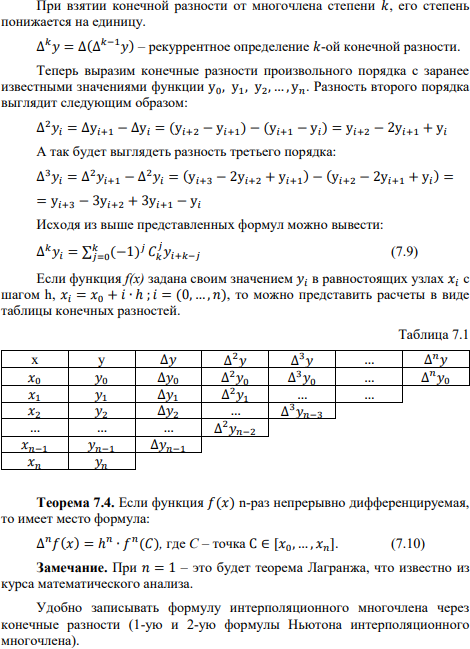
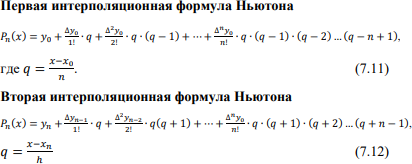
y(0) = 1

y(1) = 2,718

Про интерполировать найденное решение с помощью интерполяционного многочлена ньютона по узлам интерполяции 0,0; 0,2; 0,4; 0,6; 0,8; 1,0.

# 2.Теоретические сведения





# 

# 3.Описание алгоритма

1. **Начальные констаны**

const double EPSILON = 1e-6;

const double h\_start = 0.1;

const double X\_START = 0.0;

const double X\_END = 1.0;

1. **Структура State:**

struct State {

double y, dy;

};

Структура для хранения текущего состояния системы:

y — значение функции в точке x.

dy — значение первой производной функции в точке x.

1. **Функция f:**

State f(double x, State s) {

    double ddy = exp(2 \* x) + (s.y + s.dy) / 2;

    return {s.dy, ddy};

}

Вычисляет правую часть дифференциального уравнения второго порядка

**Возвращает:**  
Структуру State, где: y — первая производная (dy/dx), dy — вторая производная (d^2y/dx^2).

1. **Функция rk2\_step:**

State rk2\_step(double x, State s, double h) {

    State k1 = f(x, s);

    State k2 = f(x + h, {s.y + h \* k1.y, s.dy + h \* k1.dy});

    return {s.y + h \* (k1.y + k2.y) / 2,

            s.dy + h \* (k1.dy + k2.dy) / 2};

}

Реализует один шаг метода Рунге-Кутты 2-го порядка (модифицированный метод Эйлера) для решения системы ОДУ.

**Возвращает:** Новое состояние после шага.

1. **Функция rk4\_step:**

State rk4\_step(double x, State s, double h) {

    State k1 = f(x, s);

    State k2 = f(x + h / 2, {s.y + h \* k1.y / 2, s.dy + h \* k1.dy / 2});

    State k3 = f(x + h / 2, {s.y + h \* k2.y / 2, s.dy + h \* k2.dy / 2});

    State k4 = f(x + h, {s.y + h \* k3.y, s.dy + h \* k3.dy});

    return {s.y + h \* (k1.y + 2 \* k2.y + 2 \* k3.y + k4.y) / 6,

            s.dy + h \* (k1.dy + 2 \* k2.dy + 2 \* k3.dy + k4.dy) / 6};

}

Реализует один шаг метода Рунге-Кутты 4-го порядка для системы ОДУ.  
**Возвращает:**  
Более точное новое состояние по сравнению с rk2\_step.

1. **Функция solve\_rk2:**

std::vector<std::pair<double, double>> solve\_rk2() {

    std::vector<std::pair<double, double>> result;

    double x = X\_START;

    State s = {1.0, 0.0};

    double h = h\_start;

    while (x <= X\_END) {

        result.push\_back({x, s.y});

        s = rk2\_step(x, s, h);

        x += h;

    }

    return result;

}

Решает ОДУ на интервале [X\_START, X\_END] методом РК2.  
**Возвращает:**  
Вектор пар (x, y(x)) — решение в узлах сетки.  
**Начальные условия:**

y(0) = 1.0,

dy/dx(0) = 0.0.

1. **Функция solve\_rk4:**

std::vector<std::pair<double, double>> solve\_rk4() {

    std::vector<std::pair<double, double>> result;

    double x = X\_START;

    State s = {1.0, 0.0};

    double h = h\_start;

    while (x <= X\_END) {

        result.push\_back({x, s.y});

        s = rk4\_step(x, s, h);

        x += h;

    }

    return result;

}  
То же, что solve\_rk2, но с методом РК4 для большей точности.

1. **Функция newton\_interpolation:**

double newton\_interpolation(const std::vector<std::pair<double, double>>& points, double x) {

int n = points.size();

    std::vector<double> coef(n);

    for (int i = 0; i < n; i++) {

        coef[i] = points[i].second;

    }

    for (int j = 1; j < n; j++) {

        for (int i = n - 1; i >= j; i--) {

            coef[i] = (coef[i] - coef[i - 1]) / (points[i].first - points[i - j].first);

        }

    }

    double result = coef[n - 1];

    for (int i = n - 2; i >= 0; i--) {

        result = result \* (x - points[i].first) + coef[i];

    }

    return result;

}

Вычисляет значение интерполяционного многочлена Ньютона в точке x.  
**Возвращает:**  
Интерполированное значение y(x).

1. **Функция main:**

int main() {

std::vector<std::pair<double, double>> solution\_rk2 = solve\_rk2();

    std::vector<std::pair<double, double>> solution\_rk4 = solve\_rk4();

    std::vector<std::pair<double, double>> interp\_points\_rk2 = {{0.0, 0.0}, {0.2, 0.0}, {0.4, 0.0}, {0.6, 0.0}, {0.8, 0.0}, {1.0, 0.0}};

    std::vector<std::pair<double, double>> interp\_points\_rk4 = interp\_points\_rk2;

    for (auto& p : interp\_points\_rk2) {

        for (const auto& s : solution\_rk2) {

            if (fabs(s.first - p.first) < EPSILON) {

                p.second = s.second;

                break;

            }

        }

    }

    for (auto& p : interp\_points\_rk4) {

        for (const auto& s : solution\_rk4) {

            if (fabs(s.first - p.first) < EPSILON) {

                p.second = s.second;

                break;

            }

        }

    }

    std::cout << "y(1) по RK4: " << solution\_rk4.back().second << "\n";

    std::cout << std::fixed << std::setprecision(6);

    std::cout << "y(1) по RK2: " << solution\_rk2.back().second << "\n";

    std::cout << std::fixed << std::setprecision(6);

    std::cout << "x\t\ty\_RK2\t\ty\_RK4\n";

    for (double x = 0.0; x <= 1.0; x += 0.1) {

        double y\_interp\_rk2 = newton\_interpolation(interp\_points\_rk2, x);

        double y\_interp\_rk4 = newton\_interpolation(interp\_points\_rk4, x);

        std::cout << x << "\t" << y\_interp\_rk2 << "\t" << y\_interp\_rk4 << "\n";

    }

    return 0;

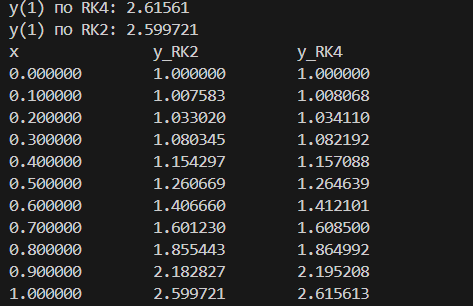
}

Решает ОДУ двумя методами.

Формирует таблицу значений для интерполяции.

Выводит результаты в виде таблицы (x, y\_RK2, y\_RK4) для x ∈ [0, 1] с шагом 0.1. Выводит результаты решения методами Рунге-Кутта 2 и 4 порядка

# 4.Результат работы программы



# 5.Листинг

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

const double EPSILON = 1e-6;

const double h\_start = 0.1;

const double X\_START = 0.0;

const double X\_END = 1.0;

struct State {

    double y, dy;

};

State f(double x, State s) {

    double ddy = exp(2 \* x) + (s.y + s.dy) / 2;

    return {s.dy, ddy};

}

State rk2\_step(double x, State s, double h) {

    State k1 = f(x, s);

    State k2 = f(x + h, {s.y + h \* k1.y, s.dy + h \* k1.dy});

    return {s.y + h \* (k1.y + k2.y) / 2,

            s.dy + h \* (k1.dy + k2.dy) / 2};

}

State rk4\_step(double x, State s, double h) {

    State k1 = f(x, s);

    State k2 = f(x + h / 2, {s.y + h \* k1.y / 2, s.dy + h \* k1.dy / 2});

    State k3 = f(x + h / 2, {s.y + h \* k2.y / 2, s.dy + h \* k2.dy / 2});

    State k4 = f(x + h, {s.y + h \* k3.y, s.dy + h \* k3.dy});

    return {s.y + h \* (k1.y + 2 \* k2.y + 2 \* k3.y + k4.y) / 6,

            s.dy + h \* (k1.dy + 2 \* k2.dy + 2 \* k3.dy + k4.dy) / 6};

}

std::vector<std::pair<double, double>> solve\_rk2() {

    std::vector<std::pair<double, double>> result;

    double x = X\_START;

    State s = {1.0, 0.0};

    double h = h\_start;

    while (x <= X\_END) {

        result.push\_back({x, s.y});

        s = rk2\_step(x, s, h);

        x += h;

    }

    return result;

}

std::vector<std::pair<double, double>> solve\_rk4() {

    std::vector<std::pair<double, double>> result;

    double x = X\_START;

    State s = {1.0, 0.0};

    double h = h\_start;

    while (x <= X\_END) {

        result.push\_back({x, s.y});

        s = rk4\_step(x, s, h);

        x += h;

    }

    return result;

}

double newton\_interpolation(const std::vector<std::pair<double, double>>& points, double x) {

    int n = points.size();

    std::vector<double> coef(n);

    for (int i = 0; i < n; i++) {

        coef[i] = points[i].second;

    }

    for (int j = 1; j < n; j++) {

        for (int i = n - 1; i >= j; i--) {

            coef[i] = (coef[i] - coef[i - 1]) / (points[i].first - points[i - j].first);

        }

    }

    double result = coef[n - 1];

    for (int i = n - 2; i >= 0; i--) {

        result = result \* (x - points[i].first) + coef[i];

    }

    return result;

}

int main() {

    std::vector<std::pair<double, double>> solution\_rk2 = solve\_rk2();

    std::vector<std::pair<double, double>> solution\_rk4 = solve\_rk4();

    std::vector<std::pair<double, double>> interp\_points\_rk2 = {{0.0, 0.0}, {0.2, 0.0}, {0.4, 0.0}, {0.6, 0.0}, {0.8, 0.0}, {1.0, 0.0}};

    std::vector<std::pair<double, double>> interp\_points\_rk4 = interp\_points\_rk2;

    for (auto& p : interp\_points\_rk2) {

        for (const auto& s : solution\_rk2) {

            if (fabs(s.first - p.first) < EPSILON) {

                p.second = s.second;

                break;

            }

        }

    }

    for (auto& p : interp\_points\_rk4) {

        for (const auto& s : solution\_rk4) {

            if (fabs(s.first - p.first) < EPSILON) {

                p.second = s.second;

                break;

            }

        }

    }

    std::cout << "y(1) по RK4: " << solution\_rk4.back().second << "\n";

    std::cout << std::fixed << std::setprecision(6);

    std::cout << "y(1) по RK2: " << solution\_rk2.back().second << "\n";

    std::cout << std::fixed << std::setprecision(6);

    std::cout << "x\t\ty\_RK2\t\ty\_RK4\n";

    for (double x = 0.0; x <= 1.0; x += 0.1) {

        double y\_interp\_rk2 = newton\_interpolation(interp\_points\_rk2, x);

        double y\_interp\_rk4 = newton\_interpolation(interp\_points\_rk4, x);

        std::cout << x << "\t" << y\_interp\_rk2 << "\t" << y\_interp\_rk4 << "\n";

    }

    return 0;

}