Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	«Информатики и управления »
Кафедра	ИУ6 «ИУ »
Группа	«ИУ6-64Б »

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчет по домашнему заданию N5:

Численное решение задачи Коши обыкновенных дифференциальных уравнений

Вариант 12

Студент: _____ Кошенков Д.О.
дата, подпись Ф.И.О.

Преподаватель: _____ Орлова А.С.
дата, подпись Ф.И.О.

Москва, 2025

Содержание

Введение

Цель домашней работы: изучение методов Эйлера-Коши и Рунге-Кутта 4-го порядка, изучение метода Рунге для автоматического выбора шага интегрирования при решении задачи Коши для обыкновенных дифференциальных уравнений.

Для достижения цели необходимо решить следующие задачи:

1. Реализовать на C++ алгоритмы методов Эйлера-Коши и Рунге-Кутта 4-го порядка для решения задачи Коши для обыкновенных дифференциальных уравнений с автоматическим выбором шага по правилу Рунге.
2. Сравнить число вычислений правой части ОДУ и количество шагов, потребовавшихся для достижения заданной точности на промежутке интегрирования, в применяемых методах.
3. Сравнить приближенное значение искомой функции в конце промежутка интегрирования, полученное каждым методом, с точным значением y(x) и убедиться в том, что оно вычислено с заданной точностью.
4. Построить на одном графике искомые функции, полученные методами Эйлера-Коши и Рунге-Кутта 4-го порядка, и сравнить их с графиком точного решения.

Отчет должен содержать:

1. Краткое описание реализуемых методов.
2. Постановку задачи и исходные данные.
3. Текст программы.
4. Сравнение числа вычисленных значений правой части ОДУ и количества шагов на промежутке интегрирования в применяемых методах.
5. Сравнение последнего приближенного значения искомой функции во всех методах с точным значением y(x).
6. Графики точного решения и решений, полученных с помощью методов Эйлера-Коши и Рунге-Кутта 4-го порядка.

1 Краткое описание реализуемых методов

1.1 Метод Эйлера-Коши (метод Хойна)

Метод Эйлера-Коши, также известный как метод Хойна или улучшенный метод Эйлера, является двухэтапным методом Рунге-Кутты второго порядка точности. Он повышает точность простого метода Эйлера за счет использования усредненного значения производной на текущем и прогнозируемом шаге. Расчетные формулы для перехода от точки (x_i, y_i) к (x_{i+1}, y_{i+1}) с шагом h:

1. Вычисление наклона в начальной точке (предиктор): $k_1 = f(x_i, y_i)$
2. Предварительное значение функции на следующем шаге: $y^*_{i+1} = y_i + h \cdot k_1$
3. Вычисление наклона в прогнозируемой точке: $k_2 = f(x_i + h, y^*_{i+1})$
4. Окончательное значение функции на следующем шаге (корректор): $y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2)$

Локальная погрешность метода Эйлера-Коши составляет $O(h^3)$, а глобальная (накопленная) погрешность на всем интервале интегрирования — $O(h^2)$.

1.2 Метод Рунге-Кутты 4-го порядка (классический)

Классический метод Рунге-Кутты 4-го порядка (РК4) является одним из наиболее распространенных и эффективных одношаговых методов для численного решения ОДУ. Он обеспечивает высокую точность при относительно небольшом объеме вычислений. Расчетные формулы для перехода от точки (x_i, y_i) к (x_{i+1}, y_{i+1}) с шагом h:

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right) \\
k_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right) \\
k_4 &= f(x_i + h, y_i + hk_3) \\
y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

Локальная погрешность метода РК4 составляет $O(h^5)$, а глобальная погрешность — $O(h^4)$.

1.3 Правило Рунге для автоматического выбора шага

Автоматический выбор шага позволяет контролировать локальную погрешность на каждом шаге интегрирования, поддерживая ее вблизи заданного уровня точности ϵ. Правило Рунге является одним из методов оценки локальной погрешности.

1. На текущем шаге x_i с текущим значением y_i и текущим шагом h вычисляется приближенное решение y_1, совершив один шаг h.

2. Затем вычисляется другое приближенное решение y_2, совершив два шага $h/2$ из той же точки (x_i, y_i).

3. Оценка локальной погрешности E для метода порядка p вычисляется по формуле:

$$E = \frac{|y_1 - y_2|}{2^p - 1}$$

4. **Контроль шага**:

   - Если $E \leq \epsilon_{zad}$ (заданная точность), шаг считается успешным. Значение y_2 (полученное с двойным половинным шагом) используется как новое значение y_{i+1}, так как оно считается более точным. Новый шаг h_{new} для следующей итерации вычисляется по формуле:

$$h_{new} = h \cdot S \cdot \left(\frac{\epsilon_{zad}}{E}\right)^{\frac{1}{p+1}}$$

     где S — коэффициент запаса (обычно $0.8-0.9$), предотвращающий слишком резкое увеличение шага.

   - Если $E > \epsilon_{zad}$, шаг отвергается. Новый, уменьшенный шаг h_{new} вычисляется по той же формуле, и текущий шаг интегрирования повторяется с этим новым шагом из точки (x_i, y_i).

5. Дополнительно вводятся ограничения на максимальное и минимальное изменение шага (например, h_{new} не должен быть больше $h_{old} \cdot \text{factor}_{max}$ и меньше $h_{old} \cdot \text{factor}_{min}$), а также абсолютный минимальный шаг, чтобы избежать чрезмерно малых шагов.

Для метода Эйлера-Коши $p = 2$. Для метода Рунге-Кутты 4-го порядка $p = 4$.

2 Постановка задачи и исходные данные

Необходимо решить задачу Коши для обыкновенного дифференциального уравнения первого порядка:

$$y'(x) = f(x, y)$$

с начальным условием $y(x_0) = y_0$ на отрезке $[x_0, x_{end}]$.

Конкретное уравнение и условия:

- Дифференциальное уравнение: $y' = x + y$
- Начальное условие: $y(0) = 1$
- Отрезок интегрирования: $[0, 1]$
- Требуемая точность (для контроля шага): $\epsilon = 1 \times 10^{-5}$
- Начальный шаг интегрирования: $h_{initial} = 0.1$
- Максимальный шаг для метода РК4 (для более детального графика): $h_{max\_cap} = 0.05$

Точное решение задачи: Аналитическое решение данного дифференциального уравнения с указанным начальным условием имеет вид:

$$y(x) = 2e^x - x - 1$$

Это решение будет использоваться для сравнения точности численных методов.

3 Текст программы

Ниже приведен листинг программы на языке C++, реализующей методы Эйлера-Коши и Рунге-Кутты 4-го порядка с автоматическим выбором шага по правилу Рунге.

Листинг 1 – Реализация численных методов и решение задачи Коши

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <functional>
#include <utility>
#include <algorithm>

using ODEFunction = std::function<double(double, double)>;

struct StepResult {
    double y_next;
    int f_evals; // Количествовычислений  f(x,y) наэтомшаге
};

StepResult euler_cauchy_step(ODEFunction f, double x, double y, double h) {
    double k1 = f(x, y);
    double y_star = y + h * k1;
    double k2 = f(x + h, y_star);
    double y_next = y + (h / 2.0) * (k1 + k2);
    return {y_next, 2}; // 2 вычисления f
}

StepResult rk4_step(ODEFunction f, double x, double y, double h) {
    double k1 = f(x, y);
    double k2 = f(x + h / 2.0, y + (h / 2.0) * k1);
    double k3 = f(x + h / 2.0, y + (h / 2.0) * k2);
    double k4 = f(x + h, y + h * k3);
    double y_next = y + (h / 6.0) * (k1 + 2.0 * k2 + 2.0 * k3 + k4);
    return {y_next, 4}; // 4 вычисления f
}

std::vector<std::pair<double, double>> solve_ode_adaptive(
    ODEFunction f,
    double x0, double y0, double x_end,
    double h_initial, double epsilon,
    std::function<StepResult(ODEFunction, double, double, double)> method_step,
    int method_order, // Порядокметода  p
    long long& total_f_evaluations,
    int& num_steps_taken,
    int& num_rejected_steps,
    double h_max_cap = 0.0) {

    std::vector<std::pair<double, double>> results;
    results.push_back({x0, y0});
    double x = x0;
    double y = y0;
    double h = h_initial;
    if (h_max_cap > 0.0) {
        h = std::min(h_initial, h_max_cap);
    }
    total_f_evaluations = 0;
```

```
num_steps_taken = 0;
num_rejected_steps = 0;
const double safety_factor = 0.9;
const double min_step_factor = 0.2;
const double max_step_factor = 5.0;
const double p_plus_1 = static_cast<double>(method_order + 1);
const double runge_denominator = std::pow(2.0, static_cast<double>(method_order)) - 1.0;
const double abs_min_h = 1e-10;
while (x < x_end) {
    double current_h_for_step = h;
    if (x + current_h_for_step > x_end) {
        current_h_for_step = x_end - x;
    }
    if (current_h_for_step < abs_min_h && current_h_for_step > 0) {
         current_h_for_step = abs_min_h;
    }
    if (current_h_for_step <= 0) break; // Предотвращениезацикливания ,
        еслишагсталнулевымилиотрицательным

    StepResult res_h = method_step(f, x, y, current_h_for_step);
    double y1 = res_h.y_next;
    int f_evals_h = res_h.f_evals;

    StepResult res_h_half1 = method_step(f, x, y, current_h_for_step / 2.0);
    double y_mid = res_h_half1.y_next;
    StepResult res_h_half2 = method_step(f, x + current_h_for_step / 2.0, y_mid,
        current_h_for_step / 2.0);
    double y2 = res_h_half2.y_next;
    int f_evals_h_half = res_h_half1.f_evals + res_h_half2.f_evals;

    total_f_evaluations += (f_evals_h + f_evals_h_half);

    double error_estimate = std::abs(y1 - y2) / runge_denominator;

    if (error_estimate <= epsilon || current_h_for_step <= abs_min_h) { // Принятьшаг
        y = y2; // y2 болееточное
        x += current_h_for_step;
        results.push_back({x, y});
        num_steps_taken++;

        // Корректировкашага  h дляследующейитерации
        double h_prev = current_h_for_step;
        if (error_estimate == 0.0) { // Еслиошибканулевая  , увеличиваемшагмаксимально
             h = h_prev * max_step_factor;
        } else {
            h = h_prev * std::pow(epsilon / error_estimate, 1.0 / p_plus_1);
        }
        h *= safety_factor; // Коэффициентбезопасности

        // Применяемограничениянановыйшаг      h
        h = std::min(h, h_prev * max_step_factor); // Неболеечемв     max_step_factor
            разбольшепредыдущего
        if (h_max_cap > 0.0) { // Явноеограничениесверху  , еслизадано
            h = std::min(h, h_max_cap);
        }
        h = std::max(h, h_prev * min_step_factor); // Неменеечемв     min_step_factor
            разменьшепредыдущего
        h = std::max(h, abs_min_h); // Абсолютныйминимумдля   h

    } else { // Отвергнутьшаг
```

```
            num_rejected_steps++;
            // Уменьшитьшаг  h ипопробоватьсновастекущими      x, y
            h = current_h_for_step * std::pow(epsilon / error_estimate, 1.0 / p_plus_1);
            h *= safety_factor;

            // Применяемограничениянановыйшаг      h для( повторнойпопытки )
            if (h_max_cap > 0.0) {
                h = std::min(h, h_max_cap);
            }
            h = std::max(h, current_h_for_step * min_step_factor); // Уменьшаемшаг , нонеслишкомсильно
            h = std::max(h, abs_min_h);
        }
         if (results.back().first >= x_end) break;
    }
    return results;
}

double f_ode(double x, double y) {
    return x + y;
}

// Точноерешение  y(x) = 2*exp(x) - x - 1
double exact_solution(double x) {
    return 2.0 * std::exp(x) - x - 1.0;
}

void print_results_for_plotting(const std::string& method_name, const std::vector<std::pair<
    double, double>>& data) {
    std::cout << "\Данныеn дляграфика  (" << method_name << "):\n";
    std::cout << "x y\n";
    for (const auto& p : data) {
        std::cout << p.first << " " << p.second << "\n";
    }
}

int main() {
    double x0 = 0.0;
    double y0 = 1.0;
    double x_end = 1.0;
    double h_initial = 0.1;
    double epsilon = 1e-5;
    long long f_evals_ec, f_evals_rk4;
    int steps_ec, steps_rk4;
    int rejected_ec, rejected_rk4;
    std::cout << std::fixed << std::setprecision(8);
    std::cout << "Решение ОДУ y' = x + y, y(0) = 1 на [0, 1] с epsilon = " << epsilon << std::endl;

    // --- МетодЭйлераКоши - ---
    auto results_ec = solve_ode_adaptive(f_ode, x0, y0, x_end, h_initial, epsilon,
                                         euler_cauchy_step, 2, // й2- порядок
                                         f_evals_ec, steps_ec, rejected_ec, 0.0); // h_max_cap =
                                             0.0 без( ограничения)
    double y_final_ec = results_ec.back().second;
    double x_final_ec = results_ec.back().first;

    std::cout << "\n--- МетодЭйлераКоши - адаптивный() ---" << std::endl;
    std::cout << "Последнее значение x: " << x_final_ec << std::endl;
    std::cout << "Последнее приближенное y(" << x_final_ec << "): " << y_final_ec << std::endl;
    std::cout << "Общее числовычислений  f(x,y): " << f_evals_ec << std::endl;
    std::cout << "Принятых шагов: " << steps_ec << std::endl;
```

```
    std::cout << "Отклоненных шагов: " << rejected_ec << std::endl;

    // --- МетодРунгеКутта - 4 ---
    double rk4_plot_max_step = 0.05; // Ограничениедляболеегладкогографика
    auto results_rk4 = solve_ode_adaptive(f_ode, x0, y0, x_end, h_initial, epsilon,
                                          rk4_step, 4, // й4- порядок
                                          f_evals_rk4, steps_rk4, rejected_rk4,
                                          rk4_plot_max_step);
    double y_final_rk4 = results_rk4.back().second;
    double x_final_rk4 = results_rk4.back().first;

    std::cout << "\n--- МетодРунгеКутта - 4 адаптивный(, h_max_cap=" << rk4_plot_max_step << ") ---"
        << std::endl;
    std::cout << "Последнее значение x: " << x_final_rk4 << std::endl;
    std::cout << "Последнее приближенное y(" << x_final_rk4 << "): " << y_final_rk4 << std::endl;
    std::cout << "Общее числовычислений  f(x,y): " << f_evals_rk4 << std::endl;
    std::cout << "Принятых шагов: " << steps_rk4 << std::endl;
    std::cout << "Отклоненных шагов: " << rejected_rk4 << std::endl;

    // --- Точноерешение  ---
    double y_exact_final = exact_solution(x_end);
    std::cout << "\n--- Точноерешение  ---" << std::endl;
    std::cout << "Точное значение y(" << x_end << "): " << y_exact_final << std::endl;

    // --- Сравнение ---
    std::cout << "\n--- Сравнениеточностивконечнойточке     (" << x_end << ") ---" << std::endl;
    std::cout << "Метод ЭйлераКоши-: Абс. ошибка = " << std::abs(y_final_ec - y_exact_final)
              << ", Отн. ошибка = " << std::abs((y_final_ec - y_exact_final) / y_exact_final) <<
                  std::endl;
    std::cout << "Метод РунгеКутта- 4 (h_max_cap=" << rk4_plot_max_step << "): Абс. ошибка = " << std
        ::abs(y_final_rk4 - y_exact_final)
              << ", Отн. ошибка = " << std::abs((y_final_rk4 - y_exact_final) / y_exact_final) <<
                  std::endl;
    return 0;
}
```

4 Результаты расчетов

На основе выполнения программы были получены следующие результаты для заданной задачи Коши с $\epsilon = 10^{-5}$.

4.1 Сравнение числа вычислений правой части ОДУ и количества шагов

В таблице 1 представлены данные по общему числу вычислений правой части ОДУ $f(x, y)$, количеству принятых и отклоненных шагов для каждого метода.

Таблица 1 – Сравнение вычислительных затрат методов

Метод	**Общ число вычислений** $f(x, y)$	**Прин шагов**	**Откл шагов**
Эйлера-Коши (адаптивный)	168	27	1
Рунге-Кутта 4 (адаптивный, $h_{max\_cap} = 0.05$)	240	20	0

Из таблицы видно, что метод Рунге-Кутты 4-го порядка потребовал больше вычислений функции $f(x, y)$ в целом (240 против 168), но совершил меньше шагов (20 против 27). Это связано с тем, что каждый "макро-шаг"адаптивного метода РК4 (включающий оценку по правилу Рунге) требует $4 \times (1+2) = 12$ вычислений $f(x, y)$, в то время как для Эйлера-Коши это $2 \times (1 + 2) = 6$ вычислений. Однако, благодаря более высокому порядку точности, метод РК4 может использовать более крупные шаги (если не ограничен h_{max_cap}) для достижения той же локальной ошибки. В данном случае, h_{max_cap} для РК4 был установлен для получения достаточного количества точек для графика, что также повлияло на количество шагов. Примечательно, что у РК4 не было отклоненных шагов, что свидетельствует о стабильности выбора шага.

4.2 Сравнение точности в конечной точке

В таблице 2 приведено сравнение приближенных значений $y(1.0)$, полученных численными методами, с точным значением $y_{exact}(1.0) = 3.43656366$.

Таблица 2 – Сравнение точности численных решений в точке $x = 1.0$

Метод	**Приближ** $y(1.0)$	**Абс ошибка**	**Отн ошибка**
Эйлера-Коши	3.43623345	3.3021×10^{-4}	9.6087×10^{-5}
Рунге-Кутта 4 ($h_{max\_cap} = 0.05$)	3.43656364	2.0000×10^{-8}	5.8200×10^{-9}
Точное решение	3.43656366	–	–

Метод Рунге-Кутты 4-го порядка показал значительно более высокую точность в конечной точке. Абсолютная ошибка составила 2.0×10^{-8}, что значительно меньше заданной для контроля шага $\epsilon = 10^{-5}$. Это подтверждает, что глобальная ошибка метода РК4 оказалась очень

малой. Метод Эйлера-Коши дал абсолютную ошибку 3.3×10^{-4}. Хотя $\epsilon = 10^{-5}$ использовалось для контроля *локальной* ошибки на каждом шаге, накопленная (глобальная) ошибка оказалась больше. Тем не менее, относительная ошибка порядка 10^{-5} все еще является приемлемой для многих приложений. Результаты подтверждают, что оба метода достигли конечной точки $x = 1.0$, и метод РК4 обеспечил результат, соответствующий заданной точности ϵ (и даже превосходящий ее) по глобальной ошибке.

5 Графическое представление результатов

На рисунке 1 представлено сравнение графиков численных решений, полученных методами Эйлера-Коши и Рунге-Кутты 4-го порядка, с графиком точного решения $y(x)$ на интервале $[0, 1]$.

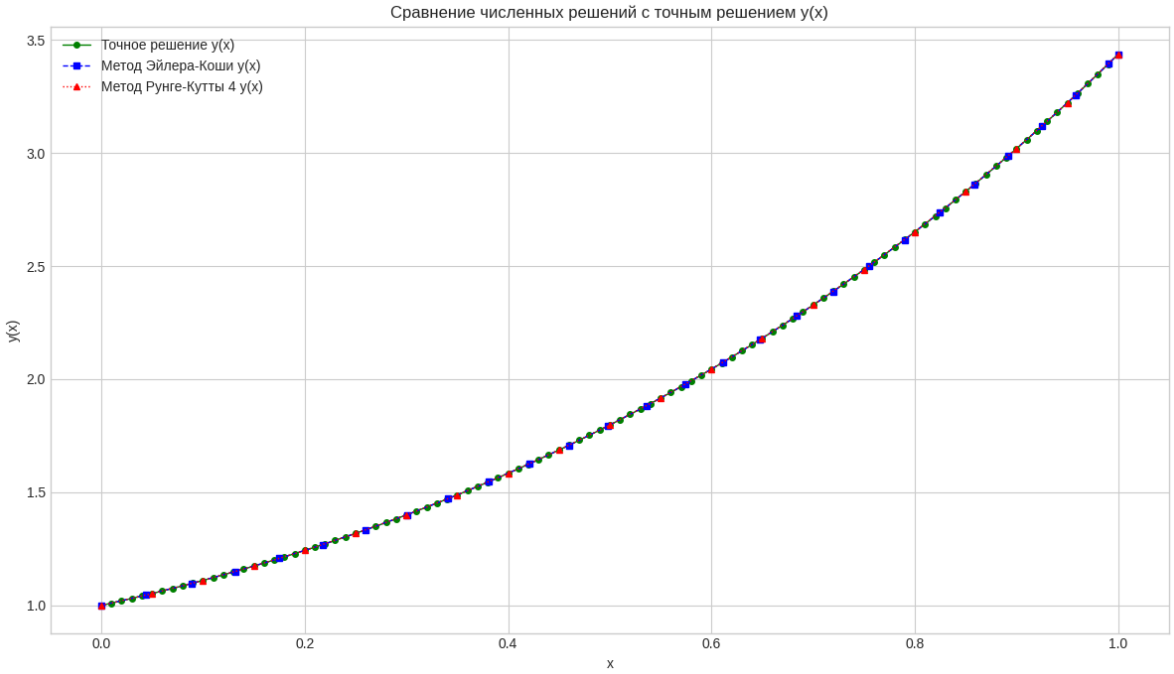

Рисунок 1 – Сравнение численных решений $y(x)$ с точным решением

Из графика видно, что оба численных решения хорошо аппроксимируют точное решение. Визуально кривые для метода Рунге-Кутты 4-го порядка и точного решения практически совпадают, что подтверждается высокой точностью, указанной в таблице 2. Решение методом Эйлера-Коши также близко к точному, но можно заметить небольшое отклонение, особенно к концу интервала. Адаптивный выбор шага позволил обоим методам следовать за поведением функции.

На рисунке 2 показано сравнение значений производной $f(x, y(x))$, вычисленной с использованием точного и приближенных решений $y(x)$. Это позволяет оценить, насколько хорошо методы аппроксимируют не только саму функцию, но и ее производную.

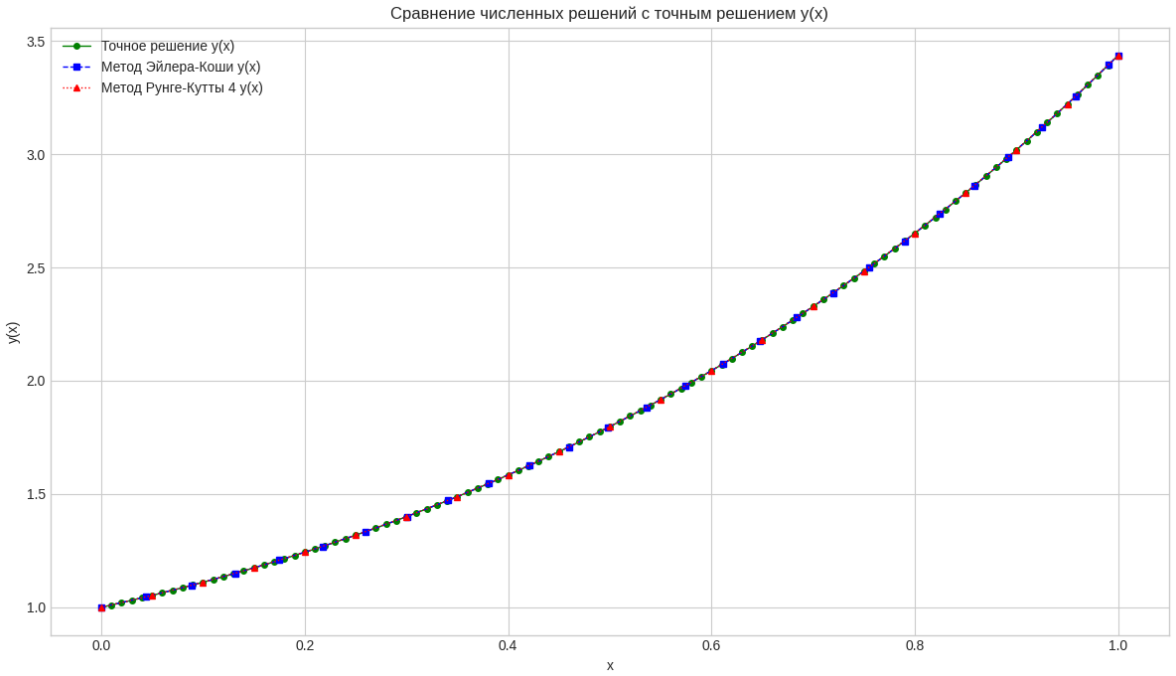

Рисунок 2 – Сравнение значений $f(x, y(x))$ для различных решений

График значений $f(x, y(x))$ также демонстрирует, что метод Рунге-Кутты 4-го порядка обеспечивает очень близкое приближение к значениям производной точного решения. Метод Эйлера-Коши показывает несколько большее отклонение, что согласуется с его более низкой точностью.

6 Заключение

В ходе выполнения домашнего задания были реализованы и исследованы два численных метода решения задачи Коши для ОДУ: метод Эйлера-Коши (2-го порядка) и метод Рунге-Кутты 4-го порядка, оба с автоматическим выбором шага по правилу Рунге.

Основные выводы:

1. Оба метода успешно справились с решением поставленной задачи $y' = x + y$, $y(0) = 1$ на отрезке $[0, 1]$ с заданной точностью контроля локальной ошибки $\epsilon = 10^{-5}$.

2. Метод Рунге-Кутты 4-го порядка продемонстрировал значительно более высокую точность в конечной точке (абсолютная ошибка $\approx 2{\times}10^{-8}$) по сравнению с методом Эйлера-Коши (абсолютная ошибка $\approx 3.3 \times 10^{-4}$). Это соответствует теоретическим порядкам точности методов ($O(h^4)$ для РК4 и $O(h^2)$ для Эйлера-Коши для глобальной ошибки).

3. Несмотря на то, что каждый "супер-шаг"(включая оценку по Рунге) метода РК4 требует больше вычислений правой части ОДУ, чем у метода Эйлера-Коши (12 против 6), общее количество шагов для РК4 оказалось меньше (20 против 27 для Эйлера-Коши при использованном $h_{max\_cap} = 0.05$ для РК4). Общее число вызовов $f(x, y)$ для РК4

было выше (240 против 168). Это подчеркивает компромисс между стоимостью одного шага и возможностью делать более крупные шаги благодаря высокой точности.

4. Адаптивный выбор шага позволил эффективно контролировать локальную погрешность и обеспечил стабильное интегрирование. Для метода РК4 не потребовалось отклонять шаги, в то время как для метода Эйлера-Коши был один отклоненный шаг.

5. Графическое сравнение подтвердило, что решение, полученное методом РК4, практически неотличимо от точного решения, в то время как решение методом Эйлера-Коши имеет небольшие, но заметные отклонения.

Цели домашней работы были достигнуты: изучены и реализованы методы Эйлера-Коши и Рунге-Кутты 4-го порядка с адаптивным шагом, проведено их сравнение по точности и вычислительным затратам. Результаты показали преимущество метода Рунге-Кутты 4-го порядка в точности при решении данной задачи.