**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное автономное образовательное учреждение**
**высшего образования**
**«Национальный исследовательский университет ИТМО»**
**Факультет программной инженерии и компьютерной техники**

**Вариант №11**
**Лабораторная работа №6**
по дисциплине
**Вычислительная математика**

Выполнил Студент группы P3112
**Пархоменко Кирилл Александрович**
Преподаватель:
**Наумова Надежда Александровна**

г. Санкт-Петербург
2024г.

# Содержание



# 1 Задание

## 1.1 Порядок выполнения работы

1. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции;
2. Пользователь выбирает ОДУ вида $y' = f(x, y)$ (не менее трех уравнений), из тех, которые предлагает программа;
3. Предусмотреть ввод исходных данных с клавиатуры: начальные условия $y_0 = y(x_0)$, интервал дифференцирования $[x_0, x_n]$, шаг $h$, точность $\varepsilon$;
4. Для исследования использовать одношаговые методы и многошаговые методы (см. табл.1);
5. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе;
6. Для оценки точности одношаговых методов использовать правило Рунге;
7. Для оценки точности многошаговых методов использовать точное решение задачи: $\varepsilon = \max_{0 \le i \le n} |y_{i\text{точн}} - y_i|$
8. Построить графики точного решения и полученного приближенного решения (разными цветами);
9. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.
10. Проанализировать результаты работы программы.

## 1.2 Вариант

### 1.2.1 Методы для реализации в программе:

1. Метод Эйлера Усовершенствованный,
2. Адамса,
3. Метод Рунге-Кутта 4-го порядка.

## 1.3 Цель работы

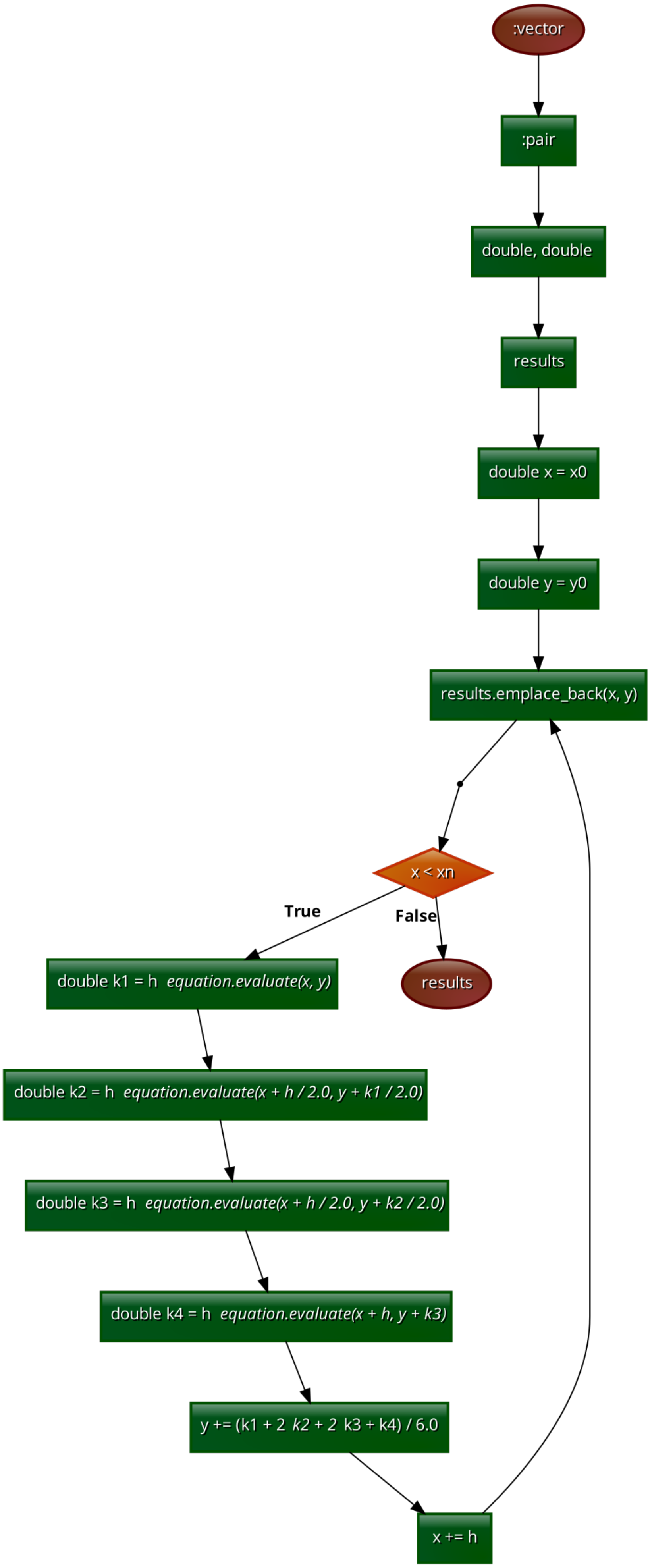
Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами.

# 2 Выполнение

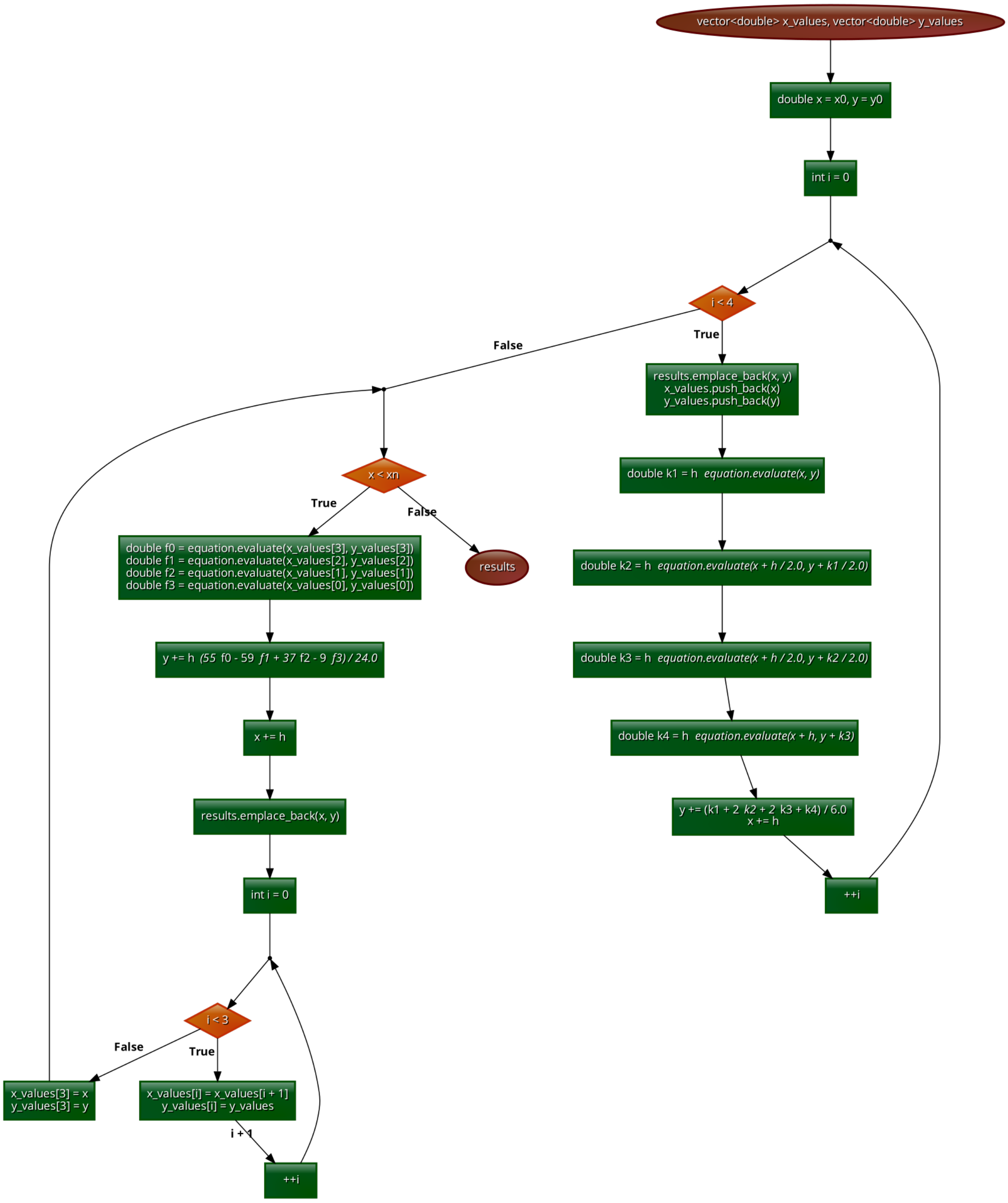
# 3 Программная реализация

## 3.1 Код и диаграммы

```cpp
std::vector<std::pair<double, double>>
DifferentialEquationCalculator::rungeKutta4() const {
  std::vector<std::pair<double, double>> results;
  double x = x0;
  double y = y0;
  results.emplace_back(x, y);

  while (x < xn) {
    double k1 = h * equation.evaluate(x, y);
    double k2 = h * equation.evaluate(x + h / 2.0, y + k1 / 2.0);
    double k3 = h * equation.evaluate(x + h / 2.0, y + k2 / 2.0);
    double k4 = h * equation.evaluate(x + h, y + k3);

    y += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
    x += h;

    results.emplace_back(x, y);
  }

  return results;
}
```

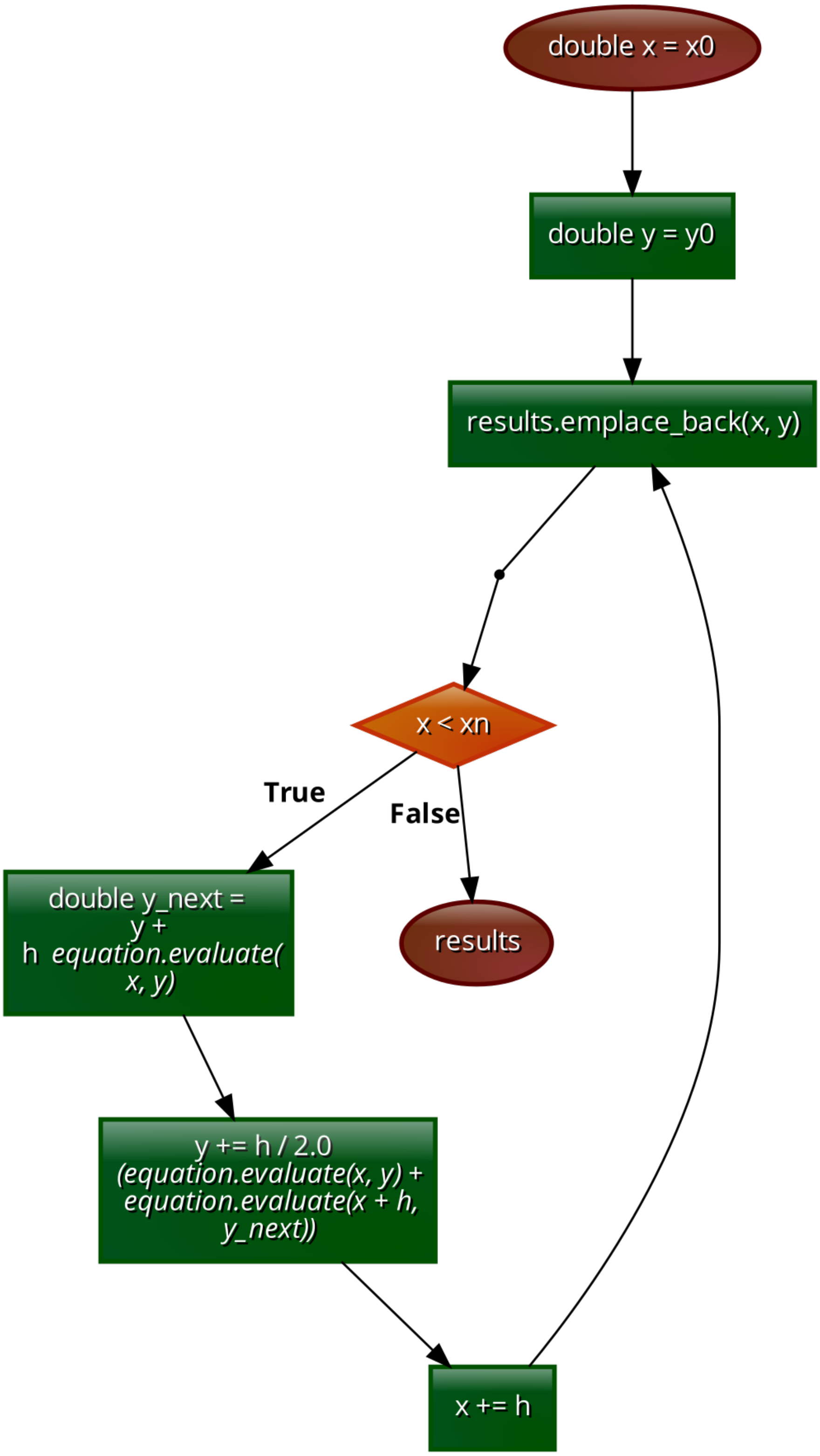

:vector
:pair
double, double
results
double x = x0
double y = y0
results.emplace_back(x, y)
x < xn
True
False
double k1 = h  equation.evaluate(x, y)
results
double k2 = h  equation.evaluate(x + h / 2.0, y + k1 / 2.0)
double k3 = h  equation.evaluate(x + h / 2.0, y + k2 / 2.0)
double k4 = h  equation.evaluate(x + h, y + k3)
y += (k1 + 2  k2 + 2  k3 + k4) / 6.0
x += h

```cpp
std::vector<std::pair<double, double>>
DifferentialEquationCalculator::adams() const {
  std::vector<std::pair<double, double>> results;
  std::vector<double> x_values;
  std::vector<double> y_values;
  double x = x0;
  double y = y0;

  // Initializing with Runge-Kutta to get the first few values
  for (int i = 0; i < 4; ++i) {
    results.emplace_back(x, y);
    x_values.push_back(x);
    y_values.push_back(y);

    double k1 = h * equation.evaluate(x, y);
    double k2 = h * equation.evaluate(x + h / 2.0, y + k1 / 2.0);
    double k3 = h * equation.evaluate(x + h / 2.0, y + k2 / 2.0);
    double k4 = h * equation.evaluate(x + h, y + k3);

    y += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
    x += h;
  }

  // Adams-Bashforth method
  while (x < xn) {
    double f0 = equation.evaluate(x_values[3], y_values[3]);
    double f1 = equation.evaluate(x_values[2], y_values[2]);
    double f2 = equation.evaluate(x_values[1], y_values[1]);
    double f3 = equation.evaluate(x_values[0], y_values[0]);

    y += h * (55 * f0 - 59 * f1 + 37 * f2 - 9 * f3) / 24.0;
    x += h;

    results.emplace_back(x, y);

    // Shift the values for the next iteration
    for (int i = 0; i < 3; ++i) {
      x_values[i] = x_values[i + 1];
      y_values[i] = y_values[i + 1];
    }
    x_values[3] = x;
    y_values[3] = y;
  }

  return results;
}
```



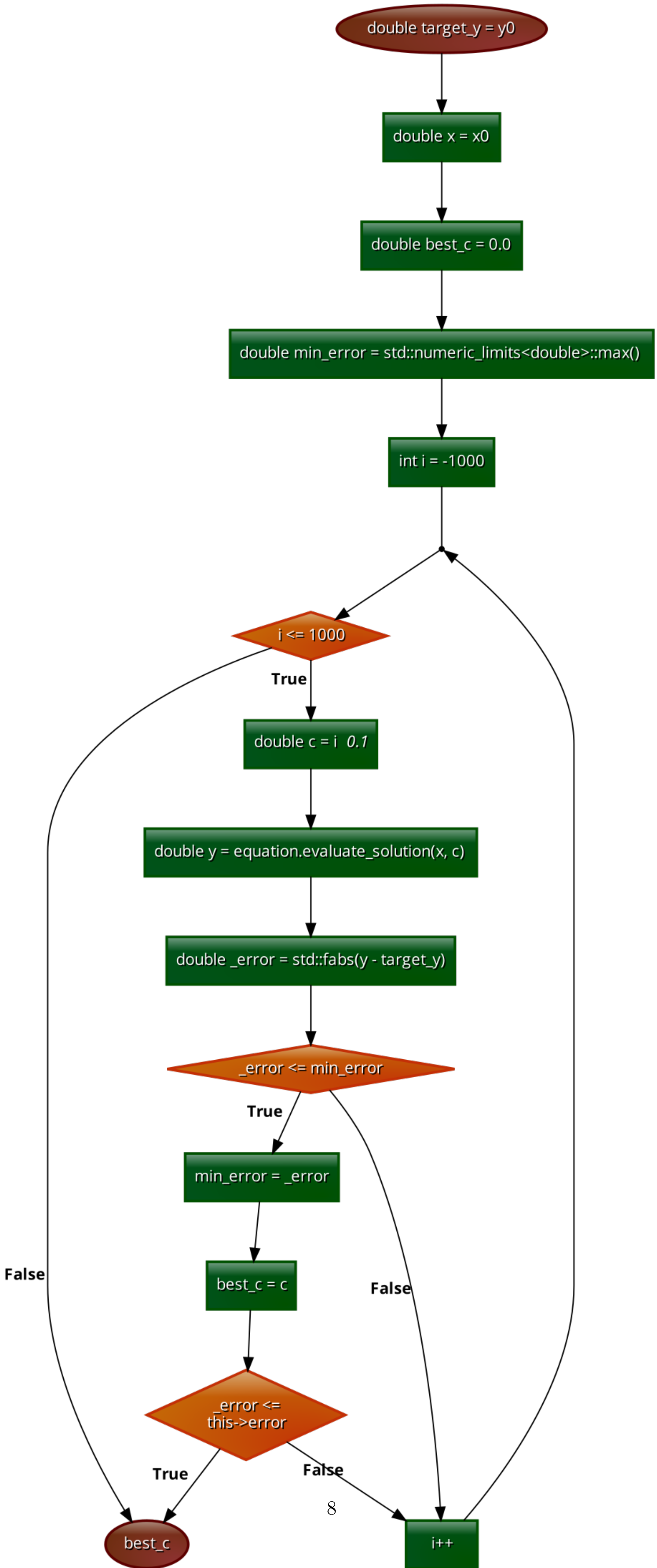

```cpp
std::vector<std::pair<double, double>>
DifferentialEquationCalculator::extendedEuler() const {
  std::vector<std::pair<double, double>> results;
  double x = x0;
  double y = y0;
  results.emplace_back(x, y);

  while (x < xn) {
    double y_next =
        y +
        h * equation.evaluate(
                x, y); // Euler's method to find the approximate next value of y
    y += h / 2.0 *
         (equation.evaluate(x, y) +
          equation.evaluate(x + h,
                            y_next)); // Correcting using the average of slopes
```

```
    x += h;

    results.emplace_back(x, y);
  }

  return results;
}
```




```
double DifferentialEquationCalculator::findConstantC() const {
```

```cpp
  double target_y = y0;
  double x = x0;
  double best_c = 0.0;
  double min_error = std::numeric_limits<double>::max();

  // Iterate over a range of possible C values
  for (int i = -1000; i <= 1000; i++) {
    double c = i * 0.1; // Adjust range and step as needed
    double y = equation.evaluate_solution(x, c);
    double _error = std::fabs(y - target_y);

    if (_error <= min_error) {
      min_error = _error;
      best_c = c;

      if (_error <=
          this->error) { // Utilize the given tolerance to possibly exit early
        break;
      }
    }
  }

  return best_c;
}
```


double target_y = y0
double x = x0
double best_c = 0.0
double min_error = std::numeric_limits<double>::max()
int i = -1000
i <= 1000
True
False
double c = i  0.1
double y = equation.evaluate_solution(x, c)
double _error = std::fabs(y - target_y)
_error <= min_error
True
False
min_error = _error
best_c = c
_error <=
this->error
True
False
best_c
i++

### 3.2 Ссылка на Github с кодом

GitHub

## 4 Заключение

При работе были изучены метод интерполяции по точкам.

## Список литературы

[1] Слайды с лекций (2023). // Кафедра информатики и вычислительной техники – Малышева Татьяна Алексеевна, к.т.н., доцент.