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 №4
по дисциплине
Вычислительная математика

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

г. Санкт-Петербург
2024г.

Содержание

1 Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

2 Задание

2.1 Программна реализация

Для исследования использовать

1. линейную функцию,
2. полиномиальную функцию 2-й степени,
3. полиномиальную функцию 3-й степени,
4. экспоненциальную функцию,
5. логарифмическую функцию,
6. степенную функцию.

2.2 Вычислительная реализация

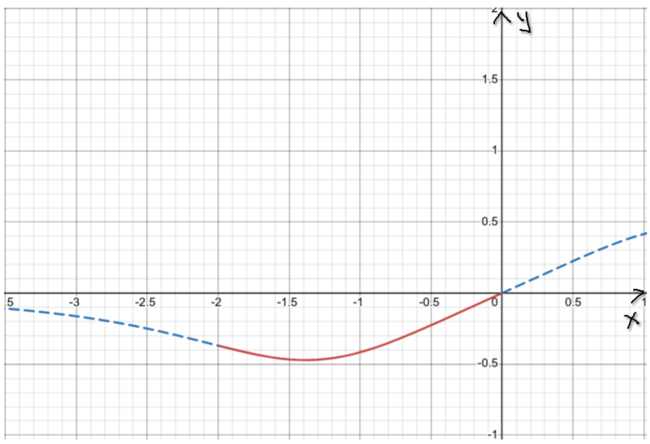
1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
4. Выбрать наилучшее приближение;
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;
6. Привести в отчете подробные вычисления.

2.3 Вариант

Исходная функция:

$$y = \frac{5x}{x^4+11},\ x \in [-2;0]\ h = 0.2$$

График функции:



x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.37	-0.42	-0.46	-0.47	-0.46	-0.42	-0.35	-0.27	-0.18	-0.09	0.0

Таблица 1: Трассировка

3 Линейная аппроксимация

Рассмотрим в качестве эмпирической формулы линейную функцию:

$$\phi(a,x,b) = ax + b$$

Сумма квадратов отклонений запишется следующим образом:

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n}(\phi(x_i) - y_i)^2 = \sum_{i=1}^{n}(ax_i^2 + b - y_i)^2 \quad -> \quad min$$

Для нахождения a и b необходимо найти минимум функции S. Необходимое условие существования минимума для функции S:

$$\begin{cases} \frac{\partial S}{\partial a} = 0 \\ \frac{\partial S}{\partial b} = 0 \end{cases} => \begin{cases} 2\sum_{i=1}^{n}(ax_i + b - y_i)x_i = 0 \\ 2\sum_{i=1}^{n}(ax_i + b - y_i) = 0 \end{cases} => \begin{cases} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i \end{cases}$$

Проведём подсчёты:

$$\sum_{i=1}^{n} x_i = (-2.0) + (-1.8) + (-1.6) + (-1.4) + (-1.2) + (-1.0) + (-0.8) + (-0.6) + (-0.4) + (-0.2) + 0.0 = -11.0$$

$$\sum_{i=1}^{n} x_i^2 = (-2.0)^2 + (-1.8)^2 + (-1.6)^2 + (-1.4)^2 + (-1.2)^2 + (-1.0)^2 + (-0.8)^2 + (-0.6)^2 + (-0.4)^2 + (-0.2)^2 + 0.0^2 = 15.4$$

$$\sum_{i=1}^{n} y_i = (-0.37) + (-0.42) + (-0.46) + (-0.47) + (-0.46) + (-0.42) + (-0.35) + (-0.27) + (-0.18) + (-0.09) + (-0.0) = -3.484$$

$$\sum_{i=1}^{n} x_i y_i = (-2.0 \times -0.37) + (-1.8 \times -0.42) + (-1.6 \times -0.46) + (-1.4 \times -0.47) + (-1.2 \times -0.46) + (-1.0 \times -0.42) +$$

$$+ (-0.8 \times -0.35) + (-0.6 \times -0.27) + (-0.4 \times -0.18) + (-0.2 \times -0.09) + (-0.0 \times -0.0) = 4.384$$

Получим систему:

$$\begin{cases} 15.4a + (-11.0)b = 4.384 \\ -11.0a + 11b = -3.484 \end{cases}$$

из которой находим:

$$\Delta = 15.4 \cdot 11 - 11^2 = 48.4$$

$$\Delta_1 = 4.384 \cdot 11 - 11 \cdot 3.484 = 9.9$$

$$\Delta_2 = 15.4 \cdot (-3.484) - (-11.0) \cdot 4.384 = -5.4296$$

Подставим найденные значения:

$$a = \frac{9.9}{48.4} \approx 0.205; \quad b = \frac{-5.4296}{48.4} \approx -0.112$$

Тогда аппроксимирующая функция будет иметь вид:

$$\phi(x) = 0.205x - 0.112$$

Дополним таблицу:

x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.37	-0.42	-0.46	-0.47	-0.46	-0.42	-0.35	-0.27	-0.18	-0.09	0.0
$\phi(x)$	-0.52	-0.48	-0.44	-0.40	-0.36	-0.32	-0.28	-0.24	-0.19	-0.15	-0.11
$\lvert y-\phi\rvert$	0.15	0.06	0.02	0.07	0.10	0.10	0.07	0.03	0.01	0.06	0.11
$\lvert y-\phi\rvert^2$	0.02	0.00	0.00	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01

Таблица 2: Трассировка

Среднеквадратические отклонения по формуле:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(\phi(x_i) - y_i)^2}{n}} = 0.1516$$

4 Квадратичная аппроксимация

Рассмотрим в качестве эмпирической формулы квадратичную функцию:

$$\phi(x, a_0, a_1, a_2) = a_0 + a_1 x + a_2 x^2$$

Сумма квадратов отклонений записывается следующим образом:

$$S = S(a_0, a_1, a_2) = \sum_{i=1}^{n}(a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2 \to \min$$

Приравниваем к нулю частные производные S по неизвестным параметрам, получаем систему линейных уравнений:

$$\begin{cases} \frac{\partial S}{\partial a_0} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i) = 0 \\ \frac{\partial S}{\partial a_1} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i)x_i = 0 \\ \frac{\partial S}{\partial a_2} = 2\sum(a_2 x_i^2 + a_1 x_i + a_0 - y_i)x_i^2 = 0 \end{cases}$$

$$\begin{cases} a_0 n + a_1 \sum x_i + a_2 \sum x_i^2 = \sum y_i \\ a_0 \sum x_i + a_1 \sum x_i^2 + a_2 \sum x_i^3 = \sum x_i y_i \\ a_0 \sum x_i^2 + a_1 \sum x_i^3 + a_2 \sum x_i^4 = \sum x_i^2 y_i \end{cases}$$

$$\sum_{i=1}^{n} x_i = -11.0 \quad \sum_{i=1}^{n} x_i^2 = 15.4 \quad \sum_{i=1}^{n} y_i = -3.484 \quad \sum_{i=1}^{n} x_i y_i = 4.384$$

$$\sum_{i=1}^{n} x_i^3 = -24.2 \quad \sum_{i=1}^{n} x_i^4 = 40.5328 \quad \sum_{i=1}^{n} x_i^2 y_i = -6.3607$$

Подставим значения:

$$\begin{cases} 11 \cdot a_0 + (-11.0) \cdot a_1 + 15.4 \cdot a_2 = -3.484 \\ -11.0 \cdot a_0 + 15.4 \cdot a_1 + (-24.2) \cdot a_2 = 4.384 \\ 15.4 \cdot a_0 + (-24.2) \cdot a_1 + 40.5328 \cdot a_2 = -6.3607 \end{cases}$$

Решение системы уравнений:

$$\begin{cases} a_0 = 0.026 \\ a_1 = 0.66 \\ a_2 = 0.231 \end{cases}$$

Получим формулу для квадратичной аппроксимации

$$\phi(x) = 0.026 + 0.66 \cdot x + 0.231 \cdot x^2$$

x	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y	-0.37	-0.42	-0.46	-0.47	-0.46	-0.42	-0.35	-0.27	-0.18	-0.09	0.0
$\phi(x)$	-0.37	-0.41	-0.44	-0.45	-0.43	-0.40	-0.35	-0.29	-0.20	-0.10	0.03
$\|y-\phi\|$	0.00	0.01	0.02	0.03	0.03	0.01	0.00	0.02	0.02	0.01	0.03
$\|y-\phi\|^2$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Таблица 3: Трассировка

Среднеквадратические отклонения по формуле:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(\phi(x_i) - y_i)^2}{n}} = 0.011$$

5 Построения результатов

Исходная:

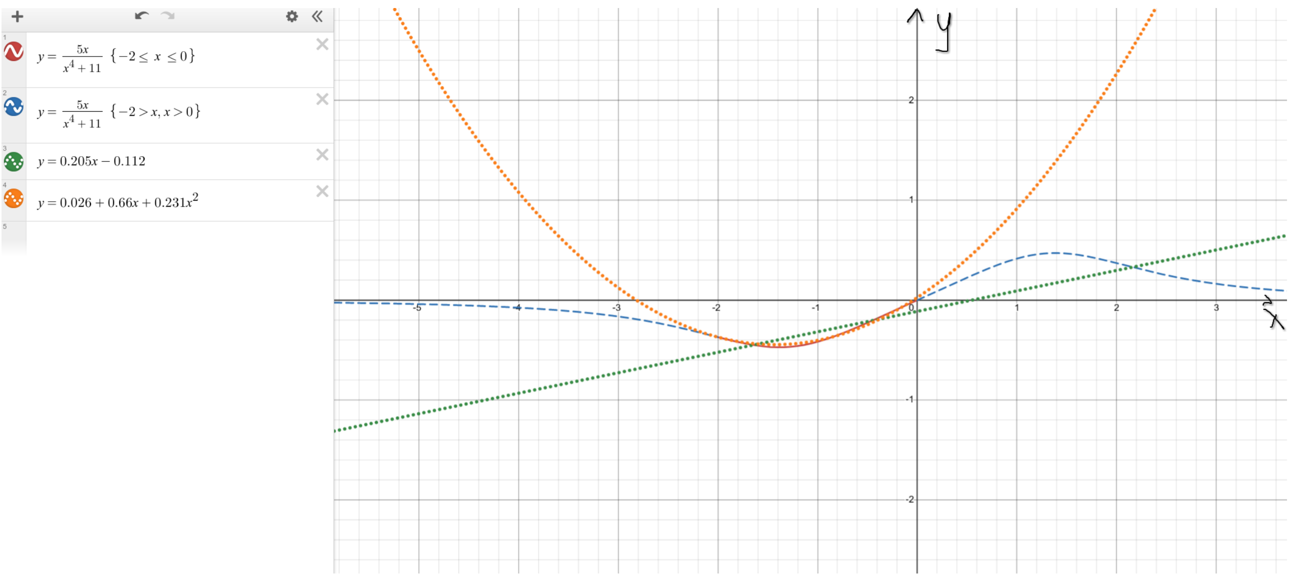
$$y = \frac{5x}{x^4 + 11}$$

Линейная:

$$\zeta(x) = 0.205x - 0.112$$

Квадратичная:

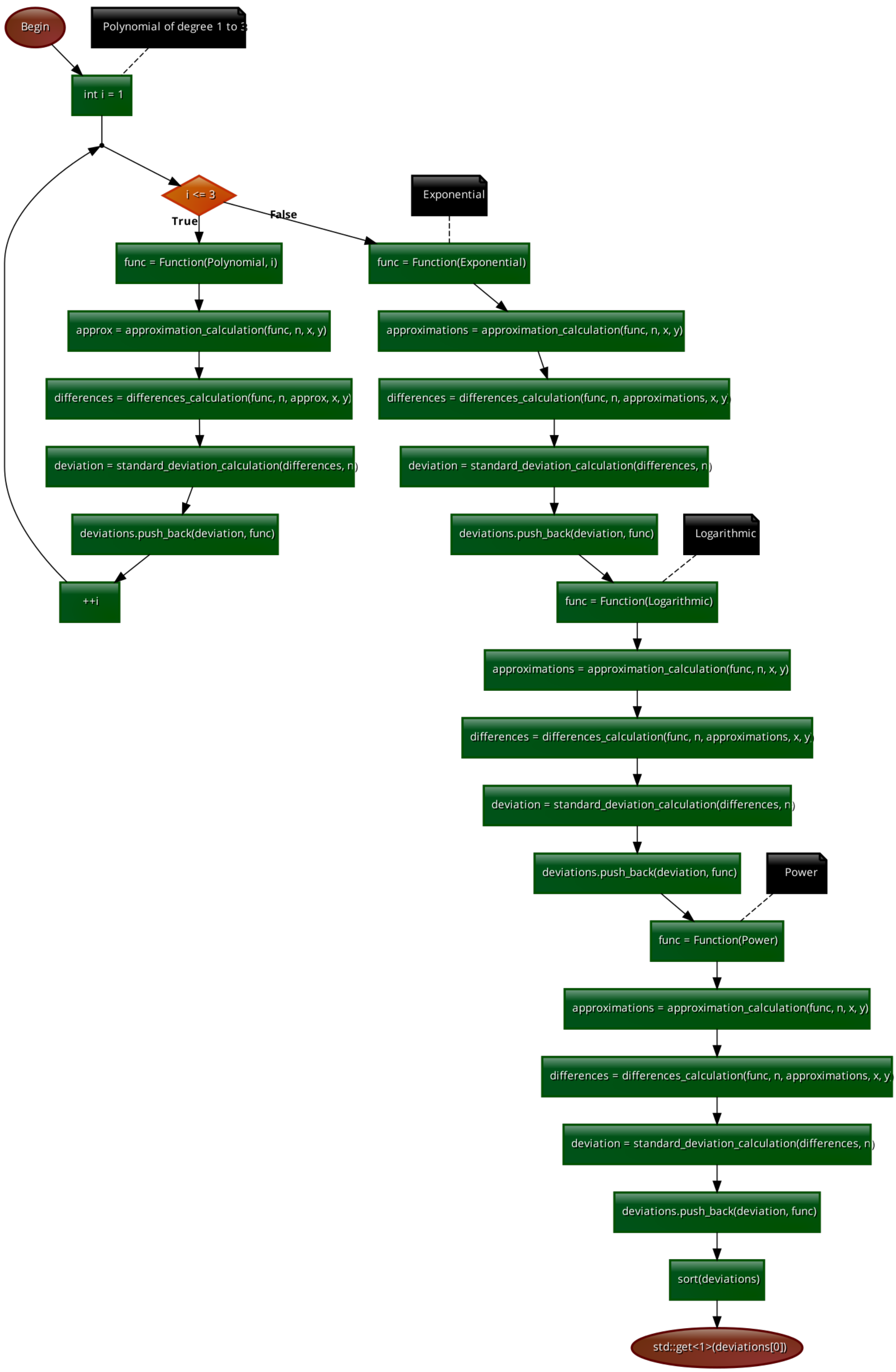
$$\aleph(x) = 0.026 + 0.66 \cdot x + 0.231 \cdot x^2$$



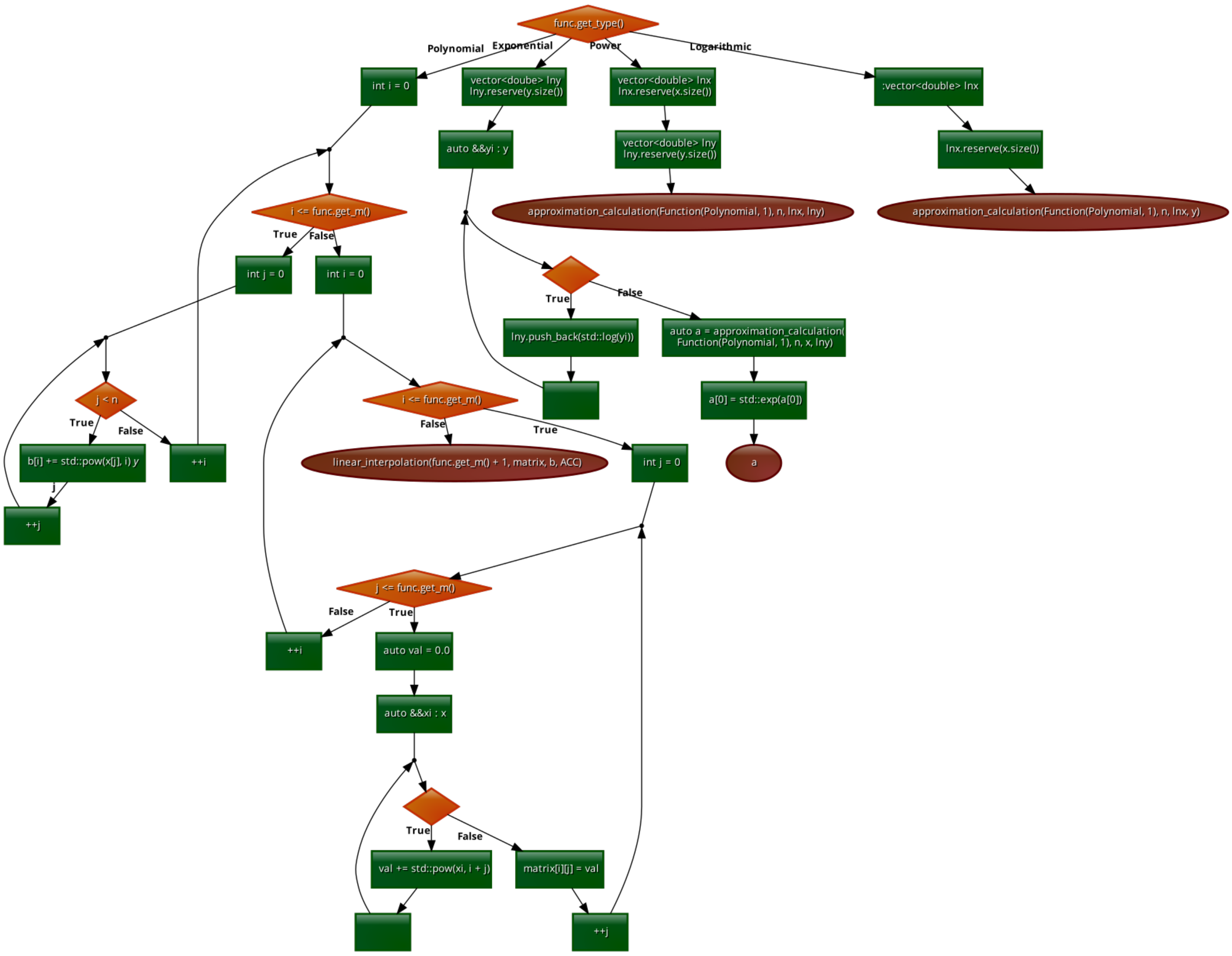
6 Программная реализация

6.1 Код и диаграммы

```cpp
static Function find_best_function(int n, std::vector<double> const &x,
                            std::vector<double> const &y) {
   std::vector<std::pair<double, Function>> deviations;

   // Polynomial of degree 1 to 3
   for (int i = 1; i <= 3; ++i) {
    auto func = Function(Function::Type::Polynomial, i);
    auto approx = approximation_calculation(func, n, x, y);
    auto differences = differences_calculation(func, n, approx, x, y);
    auto deviation = standard_deviation_calculation(differences, n);
    deviations.push_back({deviation, func});
   }

   // Exponential
   {
    auto func = Function(Function::Type::Exponential);
    auto approximations = approximation_calculation(func, n, x, y);
    auto differences = differences_calculation(func, n, approximations, x, y);
    auto deviation = standard_deviation_calculation(differences, n);
    deviations.push_back({deviation, func});
   }

   // Logarithmic
   {
    auto func = Function(Function::Type::Logarithmic);
    auto approximations = approximation_calculation(func, n, x, y);
    auto differences = differences_calculation(func, n, approximations, x, y);
    auto deviation = standard_deviation_calculation(differences, n);
    deviations.push_back({deviation, func});
   }

   // Power
   {
    auto func = Function(Function::Type::Power);
    auto approximations = approximation_calculation(func, n, x, y);
    auto differences = differences_calculation(func, n, approximations, x, y);
    auto deviation = standard_deviation_calculation(differences, n);
    deviations.push_back({deviation, func});
   }

   std::sort(deviations.begin(), deviations.end(),
          [](const auto &a, const auto &b) {
            return std::get<0>(a) < std::get<0>(b);
          });
   return std::get<1>(deviations[0]);
 }
```

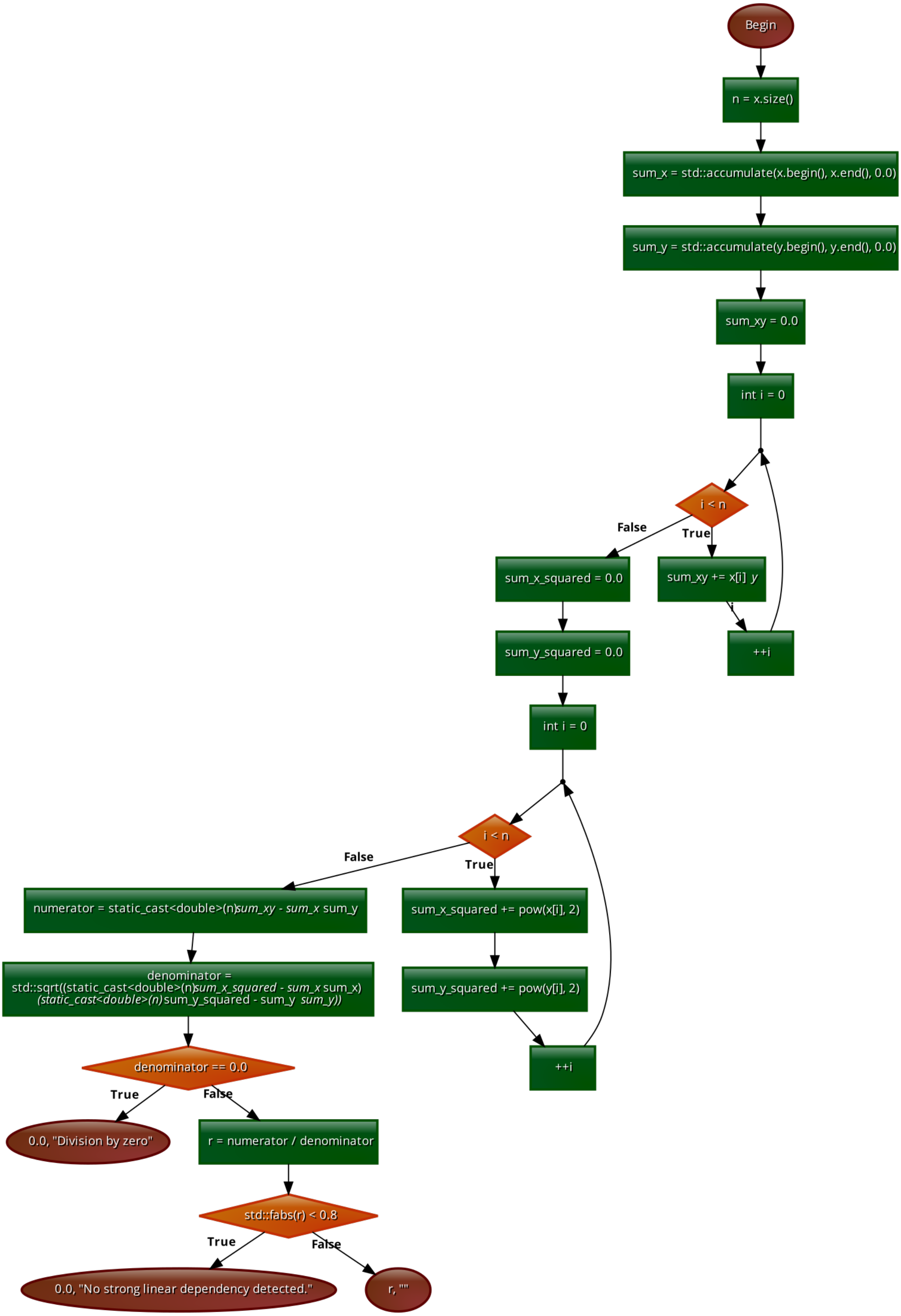

Begin
Polynomial of degree 1 to 3
int i = 1
i <= 3
True
False
func = Function(Polynomial, i)
approx = approximation_calculation(func, n, x, y)
differences = differences_calculation(func, n, approx, x, y)
deviation = standard_deviation_calculation(differences, n)
deviations.push_back(deviation, func)
++i
Exponential
func = Function(Exponential)
approximations = approximation_calculation(func, n, x, y)
differences = differences_calculation(func, n, approximations, x, y)
deviation = standard_deviation_calculation(differences, n)
deviations.push_back(deviation, func)
Logarithmic
func = Function(Logarithmic)
approximations = approximation_calculation(func, n, x, y)
differences = differences_calculation(func, n, approximations, x, y)
deviation = standard_deviation_calculation(differences, n)
deviations.push_back(deviation, func)
Power
func = Function(Power)
approximations = approximation_calculation(func, n, x, y)
differences = differences_calculation(func, n, approximations, x, y)
deviation = standard_deviation_calculation(differences, n)
deviations.push_back(deviation, func)
sort(deviations)
std::get<1>(deviations[0])

```
static std::vector<double>
  approximation_calculation(Function func, int n, std::vector<double> const &x,
                            std::vector<double> const &y) {
    switch (func.get_type()) {
    case Function::Type::Polynomial: {
      std::vector<double> b(func.get_m() + 1, 0.0);
      std::vector<std::vector<double>> matrix(
          func.get_m() + 1, std::vector<double>(func.get_m() + 1, 0.0));
      for (int i = 0; i <= func.get_m(); ++i) {
        for (int j = 0; j < n; ++j) {
          b[i] += std::pow(x[j], i) * y[j];
        }
      }
      for (int i = 0; i <= func.get_m(); ++i) {
        for (int j = 0; j <= func.get_m(); ++j) {
          auto val = 0.0;
          for (auto &&xi : x) {
            val += std::pow(xi, i + j);
          }
          matrix[i][j] = val;
        }
      }
      return linear_interpolation(func.get_m() + 1, matrix, b, ACC);
    }
    case Function::Type::Exponential: {
      std::vector<double> lny;
      lny.reserve(y.size());
      for (auto &&yi : y) {
        lny.push_back(std::log(yi));
      }
      auto a = approximation_calculation(
          Function(Function::Type::Polynomial, 1), n, x, lny);
      a[0] = std::exp(a[0]);
      return a;
    }
    case Function::Type::Power: {
      std::vector<double> lnx;
      lnx.reserve(x.size());
      std::vector<double> lny;
      lny.reserve(y.size());
      return approximation_calculation(Function(Function::Type::Polynomial, 1),
                                       n, lnx, lny);
    }

    case Function::Type::Logarithmic: {
      std::vector<double> lnx;
      lnx.reserve(x.size());
      return approximation_calculation(Function(Function::Type::Polynomial, 1),
                                       n, lnx, y);
    }
    }
    return {};
  }
```

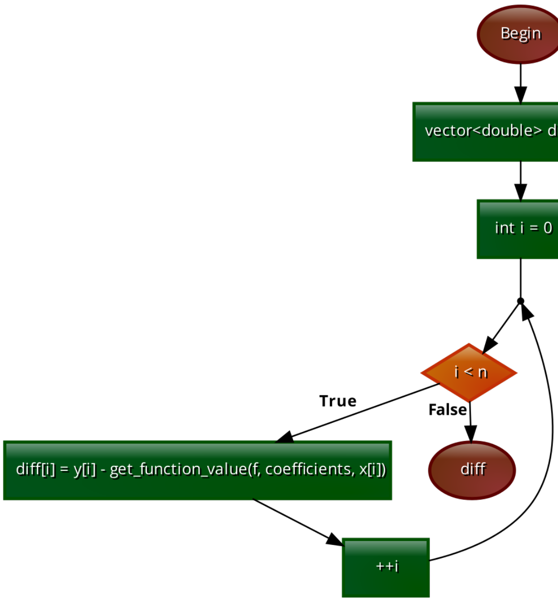



```
std::pair<double, std::string> calculate_pearson_correlation() {
    auto n = this->x.size();
    auto sum_x = std::accumulate(x.begin(), x.end(), 0.0);
    auto sum_y = std::accumulate(y.begin(), y.end(), 0.0);
    auto sum_xy = 0.0;
    for (int i = 0; i < n; ++i) {
      sum_xy += x[i] * y[i];
    }
    auto sum_x_squared = 0.0;
    auto sum_y_squared = 0.0;
    for (int i = 0; i < n; ++i) {
      sum_x_squared += pow(x[i], 2);
      sum_y_squared += pow(y[i], 2);
    }

    auto numerator = static_cast<double>(n) * sum_xy - sum_x * sum_y;
    auto denominator =
        std::sqrt((static_cast<double>(n) * sum_x_squared - sum_x * sum_x) *
                  (static_cast<double>(n) * sum_y_squared - sum_y * sum_y));
    if (denominator == 0.0) {
      return {0.0, "Division by zero"};
    }
    auto r = numerator / denominator;
    if (std::fabs(r) < 0.8) {
      return {0.0, "No strong linear dependency detected."};
    }

    return {r, ""};
```

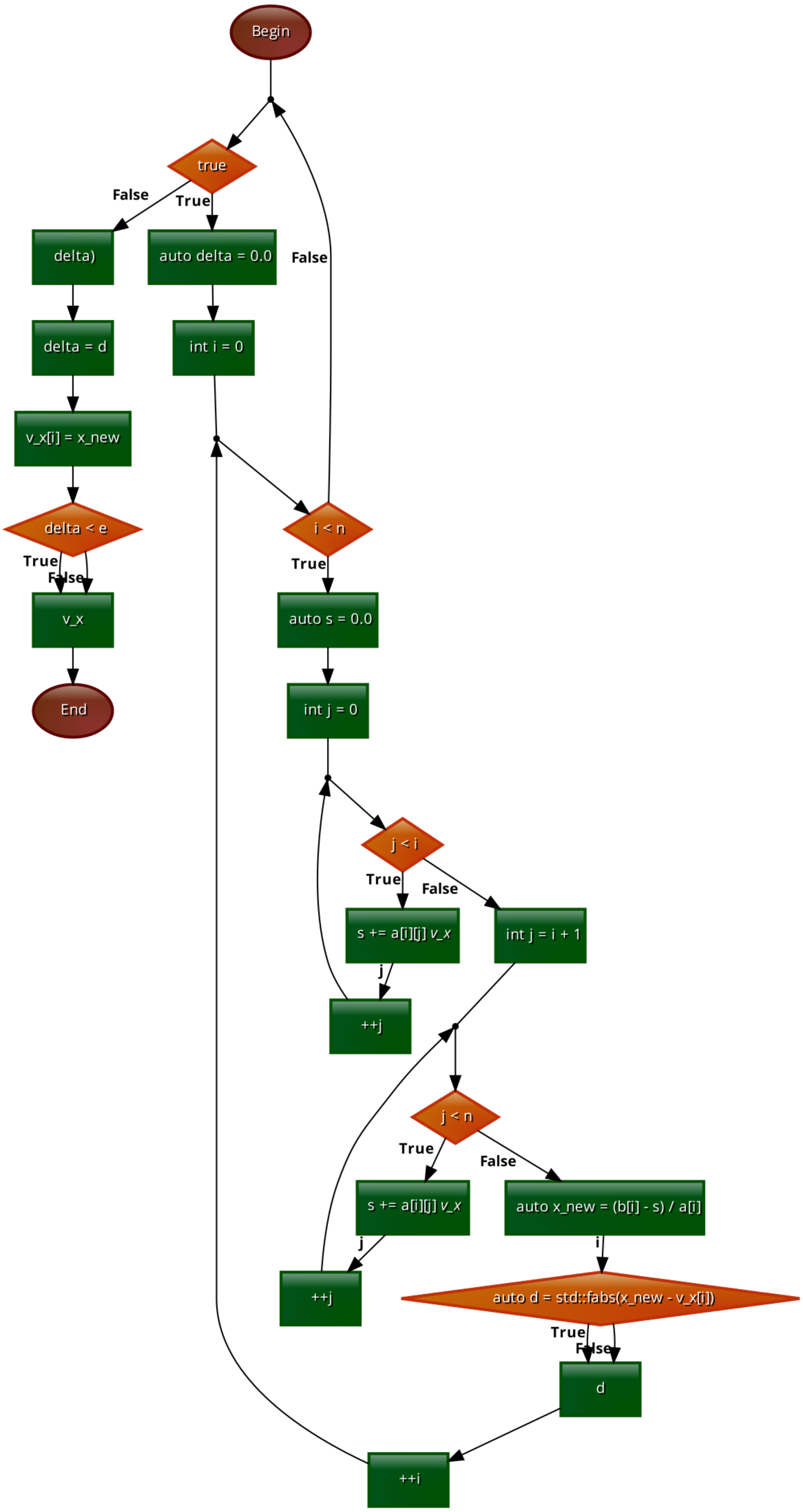
```
}
```


Begin
n = x.size()
sum_x = std::accumulate(x.begin(), x.end(), 0.0)
sum_y = std::accumulate(y.begin(), y.end(), 0.0)
sum_xy = 0.0
int i = 0
i < n
False
True
sum_x_squared = 0.0
sum_xy += x[i] y
i
++i
sum_y_squared = 0.0
int i = 0
i < n
False
True
numerator = static_cast<double>(n)sum_xy - sum_x sum_y
sum_x_squared += pow(x[i], 2)
denominator =
std::sqrt((static_cast<double>(n)sum_x_squared - sum_x sum_x)
(static_cast<double>(n)sum_y_squared - sum_y sum_y))
sum_y_squared += pow(y[i], 2)
denominator == 0.0
++i
True
False
0.0, "Division by zero"
r = numerator / denominator
std::fabs(r) < 0.8
True
False
0.0, "No strong linear dependency detected."
r, ""

```
static std::vector<double> differences_calculation(
    Function f, int n, std::vector<double> const &coefficients,
    std::vector<double> const &x, std::vector<double> const &y) {
  std::vector<double> diff(n, 0.0);

  for (int i = 0; i < n; ++i) {
    diff[i] = y[i] - get_function_value(f, coefficients, x[i]);
  }

  return diff;
}
```

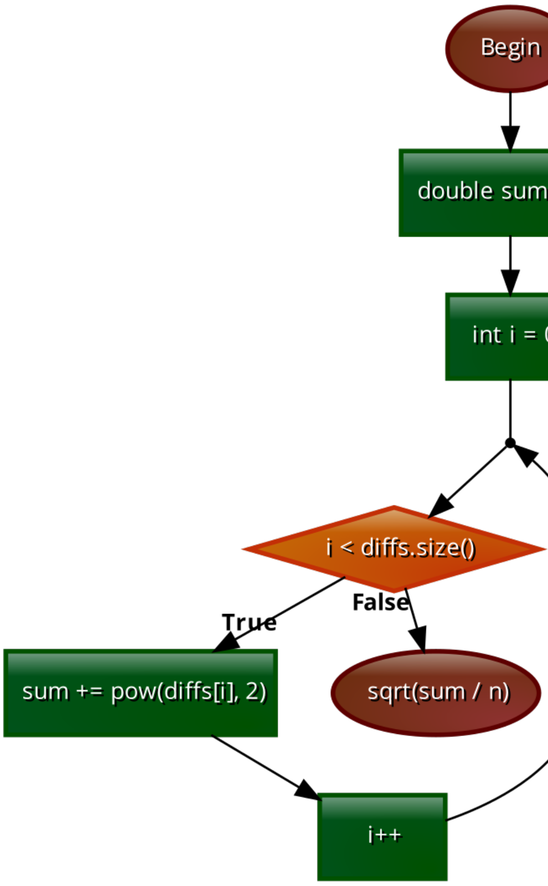



```
static std::vector<double>
  linear_interpolation(int n, std::vector<std::vector<double>> &a,
                       std::vector<double> const &b, double e) {
  std::vector<double> v_x(n, 0.0);
  while (true) {
    auto delta = 0.0;
    for (int i = 0; i < n; ++i) {
      auto s = 0.0;
      for (int j = 0; j < i; ++j) {
        s += a[i][j] * v_x[j];
      }
      for (int j = i + 1; j < n; ++j) {
        s += a[i][j] * v_x[j];
      }
      auto x_new = (b[i] - s) / a[i][i];
      if (auto d = std::fabs(x_new - v_x[i]); d > delta) {
        delta = d;
      }
      v_x[i] = x_new;
    }
    if (delta < e) {
      break;
    }
  }
  return v_x;
}
```




```cpp
static double standard_deviation_calculation(std::vector<double> const &diffs,
                                              int n) {
  double sum = 0.0;
  for (auto &&diff : diffs) {
    sum += diff * diff;
  }
  return std::sqrt(sum / n);
```

```
}
```



6.2 Ссылка на Github с кодом

GitHub

7 Заключение

При работе были изучены метод аппроксимации различными функциями, написано приложение для автоматизации подсчётов. Изучено поведение аппроксимации различных функций.

Список литературы

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