МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное бюджетное образовательное государственное
учреждение высшего образования

**«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»**

КАФЕДРА «ВЫСШАЯ МАТЕМАТИКА»

КУРСОВАЯ РАБОТА

по дисциплине «Численные методы» на тему
«Решение уравнений, систем линейных уравнений, интегралов, задачи Коши с помощью численных методов. »

Выполнил:
Учебная группа 2бПМ
Акилин.Я.А
Руководитель курсового проекта:
Старший преподаватель
Шильников Евгений Владимирович
Подпись _____

Курсовой проект защищен
с оценкой «_____»
«___» _____ 2023 г.

Москва 2023

Содержание

1 Введение

В современном мире численные методы играют важную роль в решении различных математических задач. Они позволяют нам приближенно находить решения, которые не всегда возможно получить аналитически. В данной курсовой работе мы рассмотрим применение численных методов для решения нескольких классических задач.

Одной из таких задач является решение нелинейных уравнений. Нелинейные уравнения широко применяются в различных областях науки и техники, и их точное решение может быть сложной задачей. Мы рассмотрим графический метод определения корней уравнения, а также итерационные методы, такие как метод деления отрезка пополам, метод простой итерации и метод Ньютона, которые позволят нам найти численные решения с заданной точностью.

Другой важной задачей, которую мы рассмотрим, является решение систем линейных уравнений (СЛАУ). СЛАУ возникают во многих областях науки и инженерии, и их решение является ключевым этапом в анализе и моделировании различных процессов. Мы изучим различные методы решения СЛАУ, которые позволят нам найти численные решения и рассмотреть их свойства.

Для вычисления интегралов мы рассмотрим численные методы, такие как метод трапеций и метод Симпсона. Эти методы позволяют приближенно вычислить значения интегралов путем разбиения области интегрирования на под отрезки и аппроксимации значения функции на этих подотрезках. Кроме того, мы также оценим погрешности этих вычислений, что позволит нам оценить точность полученных результатов.

Другим важным аспектом, который мы рассмотрим, является использование интерполяционных многочленов. Интерполяция позволяет нам приближенно находить значения функции на основе ее известных значений в некоторых точках. Мы будем использовать многочлены различных степеней для нахождения значений функции, заданной таблично. Это позволит нам провести анализ и сравнение различных интерполяционных подходов.

Наконец, мы обратимся к решению задач Коши, которые возникают в области дифференциальных уравнений. Задачи Коши описывают развитие системы дифференциальных уравнений с заданными начальными условиями. Мы рассмотрим методы Эйлера, модифицированный метод Эйлера и метод Рунге-Кутты 4-го порядка для численного решения задач Коши. Кроме того, мы оценим погрешность решений, что поможет нам сделать выводы о точности и стабильности этих методов.

В данной работе мы реализуем описанные методы на языке программиро-

вания Java. Благодаря численным методам, которые мы изучим, мы сможем приближенно решать различные математические задачи и получать результаты с требуемой точностью.

2 Описание реализации проекта

В данной главе мы рассмотрим структуру проекта и описание реализации численных методов для решения различных задач. Проект разделен на несколько папок, каждая из которых содержит реализацию методов для решения определенных задач.

2.1 Уравнение

В папке "Уравнения"содержатся файлы и классы, связанные с численными методами для решения нелинейных уравнений. Рассмотрим каждый метод подробнее:

2.1.1 Метод деления отрезка пополам

Здесь представлен код реализации метода деления отрезка пополам для решения нелинейных уравнений. Ниже приведен фрагмент кода, в котором происходит вычисление корня уравнения с использованием данного метода:

```
public static double bisection(String functionString, double EPS, double[] limit
   Scanner scanner = new Scanner(System.in);

   double leftBoundary = limits[0];
   double rightBoundary = limits[1];

   double valueLeftBoundFunction = evaluate(functionString, leftBoundary);
   double valueRightBoundFunction = evaluate(functionString, rightBoundary);

   if (valueLeftBoundFunction * valueRightBoundFunction > 0) {
       System.out.println("На заданном интервале нет корней!");
       return Double.NaN; // Возвращаем NaN, чтобы показать, что корень не найде
   }

   double x0 = (leftBoundary + rightBoundary) / 2;

   while (Math.abs(rightBoundary - leftBoundary) >= EPS) {
       double valueX0Function = evaluate(functionString, x0);

       if (valueRightBoundFunction * valueX0Function >= 0) {
```

```
            rightBoundary = x0;
        } else {
            leftBoundary = x0;
        }

        x0 = (leftBoundary + rightBoundary) / 2;
    }

    return x0;
}
```

2.1.2 Метод простой итерации

Здесь представлен представлен код реализации метода простой итерации для решения нелинейных уравнений. Ниже приведен фрагмент кода, демонстрирующий использование данного метода:

```
    public static double iteration(String functionString, double EPS, double[]
      int i = 1;
      double x = evaluate(functionString, limits[0]);
      double x0 = evaluate(functionString, x);

      while (true) {
          if (Math.abs(x - x0) < EPS) {
              return x0;
          }
          x = evaluate(functionString,x0);
          x0 = evaluate(functionString,x0);
          i++;

          if (i == 10000) {
              System.out.println("Выполнено 10000 итераций, решение не найдено!
              return Double.NaN;
          }
      }
  }
```

2.1.3 Метод Ньютона

Здесь представлен представлен код реализации метода Ньютона для решения нелинейных уравнений. Вот фрагмент кода, демонстрирующий использование данного метода:

```
public static double secant(String functionString,double EPS,double[] limits) t
  String firstDerivative = Derivative.takeDerivative(functionString);
  String secondDerivative = Derivative.takeDerivative(firstDerivative);

  double leftBoundary = limits[0];
  double rightBoundary = limits[1];

  double valueLeftBoundFunction = evaluate(functionString, leftBoundary);
  double valueRightBoundFunction = evaluate(functionString, rightBoundary);

  double valueLeftBoundFirstDerivative = evaluate(firstDerivative, leftBoundary
  double valueRightBoundFirstDerivative = evaluate(firstDerivative, rightBounda

  double valueLeftBoundSecondDerivative = evaluate(secondDerivative, leftBounda
  double valueRightBoundSecondDerivative = evaluate(secondDerivative, rightBoun

  double x0;

  if (valueLeftBoundFunction * valueLeftBoundSecondDerivative > 0){
      x0 = leftBoundary;
  }else if (valueRightBoundFunction * valueRightBoundSecondDerivative > 0){
      x0 = rightBoundary;
  }else {
      System.out.println("Неверно выбран начальный интревал!");
      return Double.NaN;
  }

  double x = x0 - (evaluate(functionString,x0) / evaluate(firstDerivative,x0));
  while(Math.abs(x - x0) >= EPS){
      x0 = x - (evaluate(functionString,x)/evaluate(firstDerivative,x));
      x = x0 - (evaluate(functionString,x0)/evaluate(firstDerivative,x0));
  }
  return x;
```

```
}
```

2.1.4 Меню и ввод результатов

Для удобства использования и демонстрации работы численных методов, в проекте реализовано меню, которое позволяет выбирать нужный метод и вводить необходимые значения.

Чтобы применить один из численных методов для решения уравнения $f(x) = x^3 - 2x + 2 = 0$, пользователь выбирает соответствующий метод из меню и вводит значения параметров, таких как точность (ε) и область поиска решения ($limits$).

После ввода значений метод запускается и вычисляет решение уравнения с заданной точностью и областью поиска. Результат работы метода отображается на экране, позволяя оценить полученное решение.

Таким образом, меню и ввод результатов предоставляют удобный интерфейс для работы с численными методами и исследования уравнения $f(x) = x^3 - 2x + 2 = 0$.

2.1.5 Прочие детали реализации

В этой части мы рассмотрим некоторые дополнительные детали реализации численных методов для решения уравнений:

Аргументы методов:

- `functionString` – строка, представляющая математическую функцию, передаваемую в метод. Она может содержать различные математические операции, переменные и константы.
- `EPS` – точность, с которой методы должны найти решение уравнения. Значение `EPS` определяет требуемую близость результата к истинному значению решения.
- `limits` – массив, указывающий границы области, в которой ищется решение уравнения. Обычно это два значения, представляющие левую и правую границы интервала.

Методы `createFunction` и `evaluate`:

- `createFunction` – этот метод преобразует переданную математическую функцию, заданную в виде строки, во внутренний формат, понятный для программы. Он обеспечивает корректное вычисление значения функции.

- `evaluate` – данный метод принимает функцию и значение аргумента и вычисляет значение функции в заданной точке. Он используется для получения значений функции при решении уравнений.

Метод `takeDerivative`: Для некоторых методов необходимо вычислять производную функции. В проекте реализован метод `takeDerivative`, который вычисляет значение производной переданной функции в заданной точке. Это позволяет использовать численные методы, требующие вычисления производной.

Учет этих деталей реализации помогает эффективно применять численные методы для решения уравнений и получать точные результаты.

2.1.6 Результаты методов для решения уравнений

В данной части представлены результаты численных методов для решения уравнения, а также графическое решение уравнения.

Описание функции: Для иллюстрации работы численных методов мы рассмотрим уравнение $f(x) = x^3 - 2x + 2 = 0$. Именно это уравнение будет использоваться в дальнейшем для применения различных численных методов.

Функция $f(x) = x^3 - 2x + 2 = 0$ представляет собой кубическую функцию с коэффициентами $a = 1$, $b = -2$ и $c = 2$. Наша задача состоит в нахождении корней этого уравнения с помощью численных методов, чтобы найти точные значения x, при которых $f(x) = 0$.

Далее мы будем применять методы для нахождения корней данной функции и сравнивать результаты каждого метода.

Графическое решение уравнения: Прежде чем рассмотреть показания численных методов, давайте визуализируем уравнение графически. Ниже представлен график функции, где мы ищем решение уравнения:

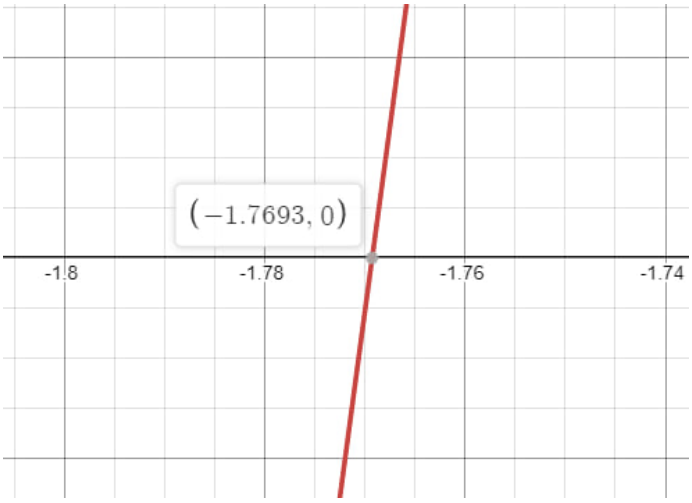

Рис. 1: Графическое решение уравнения

Результаты методов: Теперь рассмотрим результаты работы каждого из трех численных методов: метода деления отрезка пополам, метода простой итерации и метода Ньютона. Ниже представлены фотографии с результатами работы каждого метода:

Эти результаты наглядно демонстрируют эффективность и точность каждого из трех методов при решении данного уравнения.

```
Введите функцию (например, x^3-2x^1+2): x^3-2x^1+2
Введите левую границу
-2
Введите правую границу
-1
Выберите пункт меню:
0.Выйти
1.Ввести новую фунцию
2.Посчитать значение фун-ии при заданом аргументе
3.Найти производную
4.Решение уравнения методом бисекции
5.Решение уравнения методом простых итераций
6.Решение уравнения методом секущих
4
Введите точность (EPS): 0,001
Количество шагов: 10
-1.76904296875
```

(a) Метод деления отрезка пополам

```
Введите функцию (например, x^3-2x^1+2): (x^3+2)/2
Выберите пункт меню:
0.Выйти
1.Ввести новую фунцию
2.Посчитать значение фун-ии при заданом аргументе
3.Найти производную
4.Решение уравнения методом бисекции
5.Решение уравнения методом простых итераций
6.Решение уравнения методом секущих
5
Введите точность (EPS): 0,001
Количество шагов: 2
-1.2206811525195305E50
```

(b) Метод простой итерации

```
Выберите пункт меню:
0.Выйти
1.Ввести новую фунцию
2.Посчитать значение фун-ии при заданом аргументе
3.Найти производную
4.Решение уравнения методом бисекции
5.Решение уравнения методом простых итераций
6.Решение уравнения методом секущих
6
Введите точность (EPS): 0,001
Кол-во шагов: 2
-1.7693044588246745
```

(c) Метод Ньютона

Рис. 2: Результаты работы численных методов

Таблица с результатами: Теперь занесем полученные данные в таблицу, что увидеть какой метод работает эффективнее.

Методы	**Результат**	**Погрешность**	**Шагов**
Метод деления отрезка пополам	-1.769042968751	2.5703124900000596E-4	10
Метод простой итерации	-1.2206811525195305E50	1.2206811525195305E50	2
Метод Ньютона	-1.7693044588246745	4.4588246743693105E-6	2

Таблица 1: Итоги показаний методов

2.2 Система линейных уравнений

В данном разделе представлена система линейных уравнений, а также численные методы для ее решения. В проекте используются метод Гаусса и метод Зейделя.

2.2.1 Методы решения системы линейных уравнений

В проекте реализованы два численных метода для решения системы линейных уравнений: метод Гаусса и метод Зейделя.

Метод Гаусса: Метод Гаусса используется для решения систем линейных уравнений путем приведения их к упрощенной ступенчатой форме. Результатом метода Гаусса является точное решение системы.

Ниже представлен фрагмент кода, в котором происходит вычисление решения системы с использованием данного метода:

```
public static double[] gauss(LinearSystem system) {
        int size = system.getSize();
        double[][] coefficients = system.getCoefficients();
        double[] rightHandSide = system.getRightHandSide();

        // Прямой ход
        for (int i = 0; i < size - 1; i++) {
            for (int j = i + 1; j < size; j++) {
                double ratio = coefficients[j][i] / coefficients[i][i];
                for (int k = i; k < size; k++) {
                    coefficients[j][k] -= ratio * coefficients[i][k];
                }
                rightHandSide[j] -= ratio * rightHandSide[i];
            }
        }

        // Обратный ход
        double[] solution = new double[size];
        for (int i = size - 1; i >= 0; i--) {
            double sum = 0.0;
            for (int j = i + 1; j < size; j++) {
                sum += coefficients[i][j] * solution[j];
```

```
            }
            solution[i] = (rightHandSide[i] - sum) / coefficients[i][i];
        }

        return solution;
    }
```

Метод Зейделя: Другим методом, который мы применим для решения системы линейных уравнений, является метод Зейделя. Вот фрагмент кода, в котором реализовано вычисление решения системы с использованием данного метода:

```
public static double[] seidel(LinearSystem system, double epsilon) {
        int maxIterations = 100000;
        int size = system.getSize();
        double[][] coefficients = system.getCoefficients();
        double[] rightHandSide = system.getRightHandSide();

        double[] solution = new double[size];
        double[] prevSolution = new double[size];
        int iteration = 0;

        while (iteration < maxIterations) {
            for (int i = 0; i < size; i++) {
                prevSolution[i] = solution[i];
            }

            for (int i = 0; i < size; i++) {
                double sum = 0.0;
                for (int j = 0; j < size; j++) {
                    if (j != i) {
                        sum += coefficients[i][j] * solution[j];
                    }
                }
                solution[i] = (rightHandSide[i] - sum) / coefficients[i][i];
            }

            double maxDiff = calculateNormDifference(solution, prevSolution); // Испо
```

```
            if (maxDiff < epsilon) {
                break;
            }

            iteration++;
        }

        return solution;
    }
```

2.2.2 Прочие детали реализации

При использовании метода Гаусса и метода Зейдаля для решения системы линейных уравнений пользователь будет вводить коэффициенты каждого уравнения для левой и правой частей. Для начала, пользователь должен указать размер системы, то есть количество уравнений.

Из полученных данных будет создан объект системы, используя реализацию класса `LinearSystem`. Класс `LinearSystem` включает в себя:

- `coefficients`: двумерный массив, содержащий коэффициенты системы линейных уравнений.
- `rightHandSide`: одномерный массив, содержащий значения правой части уравнений.

Для создания объекта системы из введенных пользователем данных необходимо использовать методы класса `LinearSystem`. Одним из таких методов является `readSystemFromManualInput()`, который позволяет пользователю вводить коэффициенты и значения правой части системы вручную.

Таким образом, после ввода данных и создания объекта системы, вы можете передать этот объект в ваши методы решения системы линейных уравнений, такие как метод Гаусса или метод Зейдаля.

2.2.3 Результаты методов для систем линейных уравнений

Описание системы линейных уравнений: Рассмотрим систему линейных уравнений, состоящую из пяти уравнений:

$$\begin{cases} 7x_1 + 2x_2 - x_3 + x_4 - x_5 = 6 \\ 0x_1 - 6x_2 + x_3 + x_4 - 2x_5 = -4 \\ -3x_1 + x_2 + 8x_3 + x_4 + x_5 = 13 \\ 2x_1 + 2x_2 - 3x_3 + 9x_4 = 4 \\ x_1 - 2x_2 - 3x_3 + x_4 - 9x_5 = -1 \end{cases}$$

Эта система состоит из пяти уравнений с пятью неизвестными: x_1, x_2, x_3, x_4, x_5. Коэффициенты перед неизвестными и правые части уравнений указаны выше.

Результаты методов: Теперь рассмотрим результаты работы двух численных методов: метода Гаусса и метода Зейдаля. Ниже представлены скриншоты с показаниями и работой каждого метода.

Таблица с результатами: Затем можно создать таблицу, в которую вы сможете вносить полученные данные для уравнений:

Метод	**Результаты**				
	x_1	x_2	x_3	x_4	x_5
Метод Гаусса	0,57	1,25	1,68	0,60	-0,60
Метод Зейделя	0,57	1,25	1,68	0,60	-0,60

Таблица 2: Результаты работы методов для уравнений

2.3 Вычисление интеграла

В данной подсекции мы рассмотрим численные методы для вычисления определенного интеграла. Для иллюстрации применимости данных методов, рассмотрим конкретный пример с подынтегральной функцией $f(x) = x^3 - 2x + 2$ и пределами интегрирования от 1 до 2.2.

2.3.1 Метод трапеций

Метод трапеций является одним из простейших численных методов для вычисления интегралов. Он основан на аппроксимации подынтегральной функции с помощью линейных отрезков, образованных верхними и нижними трапециями.

Ниже приведен пример кода, реализующего метод трапеций для вычисления интеграла:

```
public static double integrateByTrapezoidRule(String functionString, double[] limit
      double a = limits[0];
      double b = limits[1];

      DoubleUnaryOperator function = createFunction(functionString);
      double h = (b - a) / n;
      double sum = 0.5 * (function.applyAsDouble(a) + function.applyAsDouble(b));
      for (int i = 1; i < n; i++) {
          double x = a + i * h;
          sum += function.applyAsDouble(x);
      }
      return h * sum;
  }
```

2.3.2 Метод Симпсона

Метод Симпсона является более точным численным методом для вычисления интегралов. Он основан на аппроксимации подынтегральной функции с помощью парабол, проходящих через три точки.

Ниже приведен пример кода, реализующего метод Симпсона для вычисления интеграла:

```
public static double integrateBySimpsonRule(String functionString, double[] limits,
       double a = limits[0];
       double b = limits[1];

       if (n % 2 != 0) {
           throw new IllegalArgumentException("Number of intervals (n) must be even
       }
       DoubleUnaryOperator function = createFunction(functionString);
```

```
        double h = (b - a) / n;
        double sum = function.applyAsDouble(a) + function.applyAsDouble(b);
        for (int i = 1; i < n; i++) {
            double x = a + i * h;
            sum += (i % 2 == 0) ? 2 * function.applyAsDouble(x) : 4 * function.applyA
        }
        return h / 3 * sum;
    }
```

2.3.3 Вычисление погрешности с использованием правила Рунге

Правило Рунге позволяет оценить погрешность численного метода, используя результаты вычислений с разными шагами. Ниже приведены примеры кода, демонстрирующие вычисление погрешности с использованием правила Рунге для метода трапеций и его усовершенствованной версии для метода Симпсона.

Для метода трапеций:

```
public static double estimateErrorByRungeRuleTrapezoid(String functionString, double[
        double a = limits[0];
        double b = limits[1];

        // Вычисляем интеграл с более грубым разбиением
        int nCoarse = n / 2;
        double integralCoarse = integrateByTrapezoidRule(functionString, limits, nCoa

        // Вычисляем интеграл с более точным разбиением
        double integralFine = integrateByTrapezoidRule(functionString, limits, n);

        // Оцениваем погрешность решения по правилу Рунге
        return Math.abs(integralFine - integralCoarse) / (Math.pow(2, 2) - 1);
    }
```

Для метода Симпсона:

```
  public static double estimateErrorByRichardsonSimpson(String functionString, double
        double integral = integrateBySimpsonRule(functionString, limits, n);
        double integral2 = integrateBySimpsonRule(functionString, limits, 2 * n);
```

```
        double error = Math.abs((1 / 15.0) * (integral - integral2));

        return error;
    }
```

2.3.4 Результаты методов для вычисления интегралов

Результаты методов: Теперь рассмотрим результаты работы двух численных методов: метода трапеций с использованием метода Рунге и метода Симпсона с использованием его усовершенствованной версии. Ниже представлены скриншоты с результатами работы каждого метода.

Таблица с результатами: Затем можно создать таблицу, в которую вы сможете вносить полученные данные для каждого метода:

Метод	**Результат**	**Погрешность**
Метод трапеций	4.169856000000002	0.0034559999999999036
Метод Симпсона	4.166400000000002	1.1842378929335003E-16

Таблица 3: Результаты работы методов для интегралов

2.4 Вычисление значений функции с использованием интерполяционных многочленов

С помощью интерполяционных многочленов степени от 1 до 4, мы можем найти значения функции, заданной таблично, в точке $x_0 = 1.1$. Для этого мы будем использовать метод интерполяции, который позволяет аппроксимировать функцию на основе заданных точек.

2.4.1 Инициализация массивов данных

x	-1	0	1	2	3
y	20.8	15.9	14.8	17.96	15.06

В моем проекте для инициализации этих значений, я использовал считывание данных из файлов. Для этого я создал два класса: InitArrayX и InitArrayY.

В классе InitArrayX я реализовал методы fillArrayX и fillArrayXFromFile. Метод fillArrayX считывает значения для массива x с помощью класса Scanner из стандартного ввода. Метод fillArrayXFromFile считывает значения для массива x из файла, переданного в качестве аргумента. Он открывает файл, создает экземпляр Scanner для чтения данных из файла и построчно считывает значения в массив x.

Аналогично, в классе InitArrayY я реализовал методы fillArrayY и fillArrayYFromFile для инициализации массива y. Эти методы также считывают значения из стандартного ввода или из файла.

2.4.2 Интерполяционные многочлены

Метод `interpolateAndPrint` Данный метод выполняет интерполяцию и выводит результаты для каждой степени от 1 до 4. Вот его код:

```
public static void interpolateAndPrint(double x0, double[] x, double[] y) {
for (int degree = 1; degree <= 4; degree++) {
double result = interpolate(x0, x, y, degree);
System.out.println("Степень " + degree + ": " + result);
}
}
```

Метод `interpolate` Данный метод выполняет интерполяцию для заданной степени. Вот его код:

```
public static double interpolate(double x0, double[] x, double[] y, int degree) {
int n = degree + 1;
int[] indices = getIndices(x, x0, n);
double[] xSubset = getSubset(x, indices);
double[] ySubset = getSubset(y, indices);
return computeInterpolation(x0, xSubset, ySubset, degree);
}
```

Метод `getIndices` Данный метод возвращает индексы элементов массива x, которые будут использоваться для интерполяции. Вот его код:

```
private static int[] getIndices(double[] x, double x0, int n) {
int[] indices = new int[n];
int closestIndex = findClosestIndex(x, x0);
int startIndex = Math.max(0, closestIndex - n / 2);
for (int i = 0; i < n; i++) {
indices[i] = startIndex + i;
}
return indices;
}
```

Метод `findClosestIndex` Данный метод находит индекс элемента массива x, ближайшего к значению x0. Вот его код:

```
private static int findClosestIndex(double[] x, double x0) {
int closestIndex = 0;
double closestDistance = Math.abs(x[0] - x0);
for (int i = 1; i < x.length; i++) {
double distance = Math.abs(x[i] - x0);
if (distance < closestDistance) {
closestIndex = i;
closestDistance = distance;
}
}
return closestIndex;
}
```

Метод `getSubset` Данный метод возвращает подмассив arr, состоящий из элементов с заданными индексами. Вот его код:

```
private static double[] getSubset(double[] arr, int[] indices) {
double[] subset = new double[indices.length];
for (int i = 0; i < indices.length; i++) {
subset[i] = arr[indices[i]];
}
return subset;
}
```

Метод `computeInterpolation` Данный метод вычисляет интерполяцию для заданных значений x, y и степени. Вот его код:

```
private static double computeInterpolation(double x0, double[] x, double[] y, int deg
double result = 0.0;
for (int i = 0; i <= degree; i++) {
double term = y[i];
for (int j = 0; j <= degree; j++) {
if (j != i) {
term *= (x0 - x[j]) / (x[i] - x[j]);
}
}
result += term;
}
return result;
}
```

2.4.3 Вычисление погрешности

Метод `computeErrors` Данный метод вычисляет погрешность интерполяции для каждой степени от 1 до 4. Вот его код:

```
public static void computeErrors(double x0, double[] x, double[] y) {
    for (int degree = 1; degree <= 4; degree++) {
        double interpolatedValue = interpolate(x0, x, y, degree);
        double trueValue = computeTrueValue(x0, x, y);

        double error = Math.abs(interpolatedValue - trueValue);
        System.out.println("Степень " + degree + ": Погрешность = " + error);
    }
}
```

Метод `computeTrueValue` Данный метод вычисляет истинное значение функции для заданного значения x_0. Вот его код:

```
public static double computeTrueValue(double x0, double[] x, double[] y) {
    int index = findNearestIndex(x0, x);
```

```
    return y[index];
}
```

Метод `findNearestIndex` Данный метод находит индекс элемента массива x, ближайшего к значению x_0. Вот его код:

```
public static int findNearestIndex(double x0, double[] x) {
    int index = 0;
    double minDiff = Math.abs(x0 - x[0]);

    for (int i = 1; i < x.length; i++) {
        double diff = Math.abs(x0 - x[i]);
        if (diff < minDiff) {
            minDiff = diff;
            index = i;
        }
    }

    return index;
}
```

2.4.4 Результаты работы методов с интерполяционными многочленами

Результаты методов:

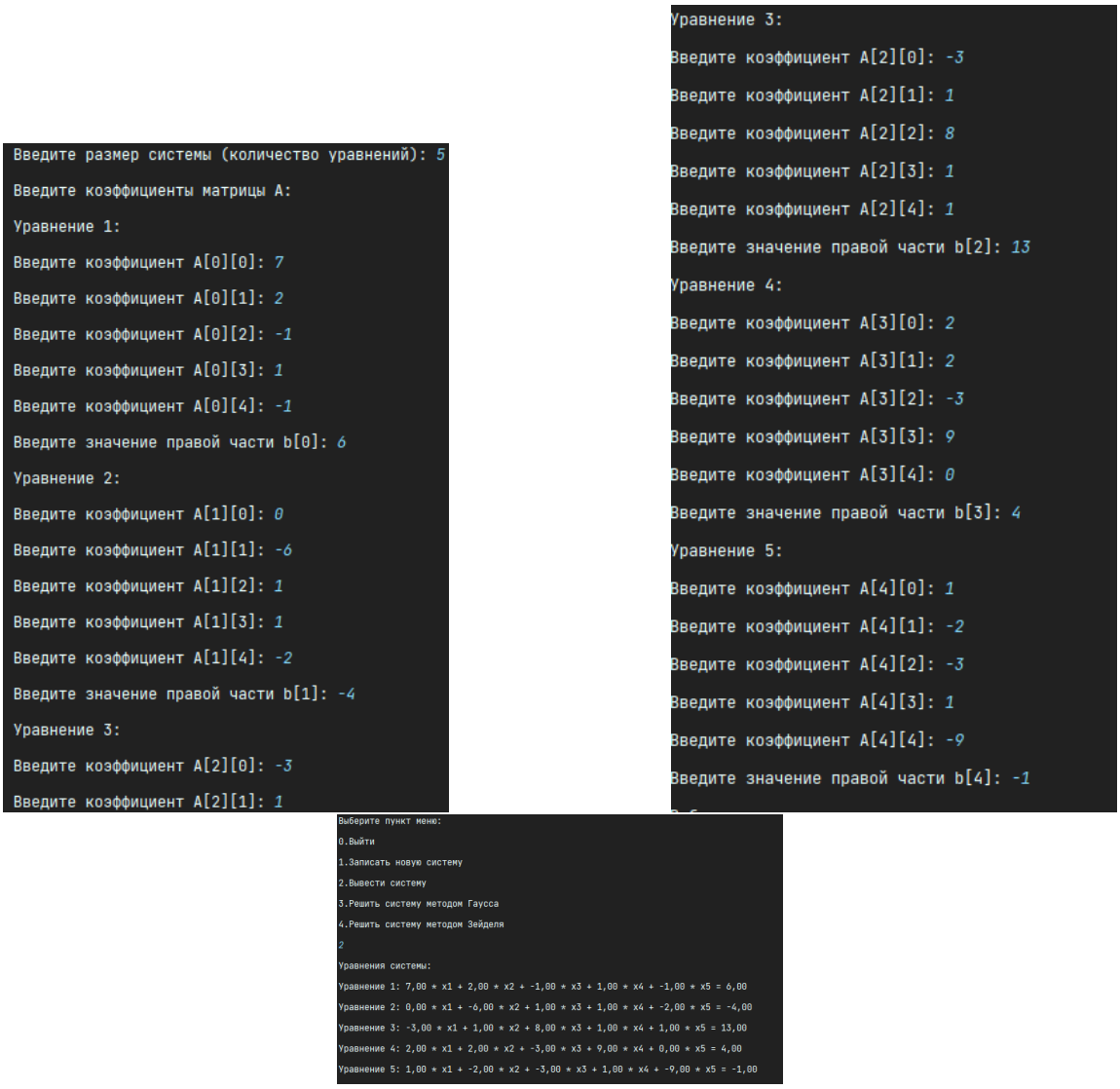

Рис. 3: Ввод данных для системы линейных уравнений

```
Выберите пункт меню:
0.Выйти
1.Записать новую систему
2.Вывести систему
3.Решить систему методом Гаусса
4.Решить систему методом Зейделя
3
Решение с помощью метода Гаусса:
x1 = 0,57
x2 = 1,25
x3 = 1,68
x4 = 0,60
x5 = -0,60
```

(a) Результат работы метода Гаусса

```
Выберите пункт меню:
0.Выйти
1.Записать новую систему
2.Вывести систему
3.Решить систему методом Гаусса
4.Решить систему методом Зейделя
4
Введите точность (EPS): 0,0001
Решение с помощью метода Зейделя:
x1 = 0,57
x2 = 1,25
x3 = 1,68
x4 = 0,60
x5 = -0,60
```

(b) Результат работы метода Зейдаля

Рис. 4: Результаты работы методов для уравнений

```
Выберите пункт меню:
0.Выйти
1.Вычислить интеграл методом трапеций
2.Вычислить интеграл методом Симпосона
1
Интеграл по формуле трапеций: 4.169856000000002
Оценка погрешности по правилу Рунге: 0.0034559999999999036
```

(a) Результат работы метода трапеций

```
Выберите пункт меню:
0.Выйти
1.Вычислить интеграл методом трапеций
2.Вычислить интеграл методом Симпосона
2
Интеграл по формуле Симпсона: 4.166400000000002
Оценка погрешности по правилу Рунге: 1.1842378929335003E-16
```

(b) Результат работы метода Симпсона

Рис. 5: Результаты работы методов для интегралов

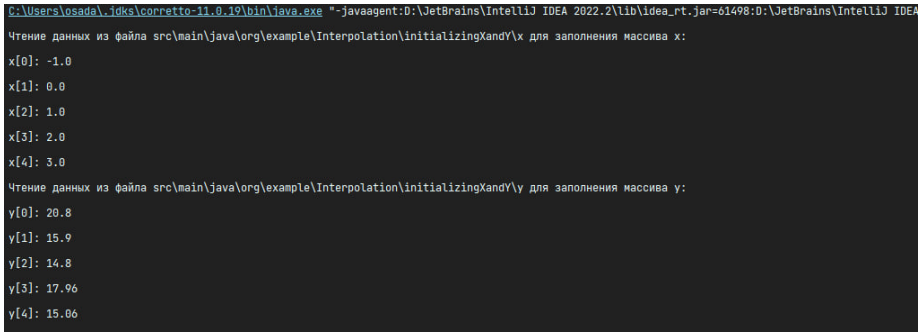

Рис. 6: Заполнение массивовов x и y

```
Введите точку x0
1,1
Степень 1: 14.69
Степень 2: 14.9243
Степень 3: 14.91671
Степень 4: 15.010091749999999
Степень 1: Погрешность = 0.11000000000000121
Степень 2: Погрешность = 0.12429999999999986
Степень 3: Погрешность = 0.11670999999999943
Степень 4: Погрешность = 0.21009174999999836
```

Рис. 7: Значения функции при разных степенях интерполяционного многочлена

Таблица 4: Значения функции и погрешности при разных степенях многочлена

Степень	Значение функции	Погрешность
1	14.69	0.11
2	14.9243	0.1243
3	14.91671	0.11671
4	15.010091749999999	0.21009174999999836

Таблица с результатами:

2.5 Решение задачи Коши

Для решения задачи Коши с начальным условием $y(1) = 0$ на отрезке $x = [1; 2.2]$ с шагом $h = 0.1$, мы использовали три численных метода: метод Эйлера, модифицированный метод Эйлера и метод Рунге-Кутты 4-го порядка.

2.5.1 Метод Эйлера

```
public static Map<String, List<Double>> euler(String functionString, double[] lim
    List<Double> x1 = new ArrayList<>();
    List<Double> y1 = new ArrayList<>();

    double x0 = limits[0];
    double xLast = limits[1];

    x1.add(x0);
    y1.add(y0);

    DoubleBinaryOperator function = createFunction(functionString);

    int n = (int) Math.round((xLast - x0) / h);

    for (int i = 0; i < n; i++) {
        double f = evaluateFunctionTwoVariables(functionString, x0, y0);
        y0 = y0 + h * f;
        x0 = x0 + h;

        x1.add(x0);
```

```
            y1.add(y0);
        }

        Map<String, List<Double>> result = new HashMap<>();
        result.put("x1", x1);
        result.put("y1", y1);

        return result;
    }
```

2.5.2 Модифицированный метод Эйлера

```
    pu public static Map<String, List<Double>> modifiedEuler(String functionString, d
        List<Double> x1 = new ArrayList<>();
        List<Double> y1 = new ArrayList<>();

        double x0 = limits[0];
        double xLast = limits[1];

        x1.add(x0);
        y1.add(y0);

        int n = (int) Math.round((xLast - x0) / h);

        for (int i = 0; i < n; i++) {
            double f1 = evaluateFunctionTwoVariables(functionString, x0, y0);
            double xHalf = x0 + h / 2.0;
            double yHalf = y0 + h / 2.0 * f1;
            double f2 = evaluateFunctionTwoVariables(functionString, xHalf, yHalf);

            double yNext = y0 + h * f2;
            x0 = x0 + h;
            y0 = yNext;

            x1.add(x0);
            y1.add(y0);
        }
```

```
    Map<String, List<Double>> result = new HashMap<>();
    result.put("x1", x1);
    result.put("y1", y1);

    return result;
}
```

2.5.3 Метод Рунге-Кутты 4-го порядка

```
public static Map<String, List<Double>> rungeKutta4(String functionString, double
    List<Double> x1 = new ArrayList<>();
    List<Double> y1 = new ArrayList<>();

    double x0 = limits[0];
    double xLast = limits[1];

    x1.add(x0);
    y1.add(y0);

    int n = (int) Math.round((xLast - x0) / h);

    for (int i = 0; i < n; i++) {
        double k1 = evaluateFunctionTwoVariables(functionString, x0, y0);
        double k2 = evaluateFunctionTwoVariables(functionString, x0 + h / 2.0, y0
        double k3 = evaluateFunctionTwoVariables(functionString, x0 + h / 2.0, y0
        double k4 = evaluateFunctionTwoVariables(functionString, x0 + h, y0 + h *

        double yNext = y0 + h / 6.0 * (k1 + 2 * k2 + 2 * k3 + k4);
        x0 = x0 + h;
        y0 = yNext;

        x1.add(x0);
        y1.add(y0);
    }

    Map<String, List<Double>> result = new HashMap<>();
```

```
        result.put("x1", x1);
        result.put("y1", y1);

        return result;
    }
```

2.5.4 Прочие детали реализации

Для использования методов Эйлера, модифицированного метода Эйлера и метода Рунге-Кутты 4-го порядка вам потребуется отдельный метод, который будет вычислять значение функции двух переменных:

```
 // Вычисление значения функции при заданных аргументах x и y
    public static double evaluateFunctionTwoVariables(String functionString, double x
        DoubleBinaryOperator function = createFunction(functionString);
        try {
            return function.applyAsDouble(x, y);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw ex;
        }
    }

    // Правильная обработка функции с двумя переменными x и y
    public static DoubleBinaryOperator createFunction(String function) throws ScriptE
        function = function.replaceAll("(\\d+(,\\d+)?)\\^(\\d+)", "Math.pow($1, $3)")
        function = function.replaceAll("(\\d+(,\\d+)?)x", "$1*x");
        function = function.replaceAll("x\\^(\\d+)", "Math.pow(x, $1)");
        function = function.replaceAll("(\\d+(,\\d+)?)y", "$1*y");
        function = function.replaceAll("y\\^(\\d+)", "Math.pow(y, $1)");
        function = function.replaceAll("e", "2.7");
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("js");
        engine.eval("function f(x, y) { return " + function + "; }");
        Invocable invocable = (Invocable) engine;
        String finalFunction = function;
        return (x, y) -> {
            try {
```

```
                Object result = invocable.invokeFunction("f", x, y);
                if (result instanceof Number) {
                    return ((Number) result).doubleValue();
                } else {
                    throw new IllegalArgumentException("Function result is not a numb
                }
            } catch (ScriptException | NoSuchMethodException ex) {
                throw new IllegalArgumentException("Invalid function: " + finalFuncti
            }
        };
    }
```

Кроме того, для оценки погрешности этих методов вы используете правило Рунге-Ромберга. Ниже представлен код, который вы можете использовать для оценки погрешности:

```
public static double estimateErrorRungeRomberg(List<Double> solution, double h) {
        int n = solution.size();

        double error = 0.0;
        for (int i = 0; i < n - 1; i++) {
            double y1 = solution.get(i);
            double y2 = solution.get(i + 1);
            double exactValue = (y2 - y1) / (h);

            error += Math.pow(exactValue - y2, 2);
        }

        return Math.sqrt(error);
    }
```

2.5.5 Результаты методов для задачи Коши

Результаты методов: Теперь рассмотрим результаты работы трех численных методов: метода Эйлера, модифицированного метода Эйлера и метода Рунге-Кутты 4-го порядка для решения задачи Коши на заданном отрезке. Ниже приведены скриншоты с результатами работы каждого метода.

```
Введите функцию двух переменных (например,y-x^2): y-x^2
Введите левую границу
1
Введите правую границу
2,2
Введите значение шага h:
0,1
Введите значение y(1.0): 0
```

(a) Меню

```
Метод Эйлера:
Results:
Step 1: x = 1.0, y = 0.0
Step 2: x = 1.1, y = -0.1
Step 3: x = 1.2000000000000002, y = -0.23100000000000004
Step 4: x = 1.3000000000000003, y = -0.3981000000000001
Step 5: x = 1.4000000000000004, y = -0.6069100000000002
Step 6: x = 1.5000000000000004, y = -0.8636010000000003
Step 7: x = 1.6000000000000005, y = -1.1749611000000004
Step 8: x = 1.7000000000000006, y = -1.5484572100000007
Step 9: x = 1.8000000000000007, y = -1.992302931000001
Step 10: x = 1.9000000000000008, y = -2.5155332241000012
Step 11: x = 2.000000000000001, y = -3.128086546510002
Step 12: x = 2.100000000000001, y = -3.8408952011610022
Step 13: x = 2.200000000000001, y = -4.665984721277103
Погрешность метода Эйлера: 0.6977997662239384
```

(b) Результат работы метода Эйлера

```
Улучшенный метод Эйлера:
Results:
Step 1: x = 1.0, y = 0.0
Step 2: x = 1.1, y = -0.11525000000000002
Step 3: x = 1.2000000000000002, y = -0.26565125000000006
Step 4: x = 1.3000000000000003, y = -0.45699463125000017
Step 5: x = 1.4000000000000004, y = -0.6956790675312503
Step 6: x = 1.5000000000000004, y = -0.9887753696220318
Step 7: x = 1.6000000000000005, y = -1.3440967834323454
Step 8: x = 1.7000000000000006, y = -1.7702769456927419
Step 9: x = 1.8000000000000007, y = -2.27685602499048
Step 10: x = 1.9000000000000008, y = -2.8743759076144806
Step 11: x = 2.000000000000001, y = -3.5744853779140016
Step 12: x = 2.100000000000001, y = -4.390056342594972
Step 13: x = 2.200000000000001, y = -5.335312258567445
Погрешность усовершенствованного метода Эйлера: 0.47243654028687754
```

(c) Результат работы модифицированного метода Эйлера

```
Метод Рунге-Кутты 4го порядка:
Results:
Step 1: x = 1.0, y = 0.0
Step 2: x = 1.1, y = -0.11585437500000001
Step 3: x = 1.2000000000000002, y = -0.2670132928307292
Step 4: x = 1.3000000000000003, y = -0.45929317834881445
Step 5: x = 1.4000000000000004, y = -0.6991221746600746
Step 6: x = 1.5000000000000004, y = -0.9936044780375537
Step 7: x = 1.6000000000000005, y = -1.350591438996495
Step 8: x = 1.7000000000000006, y = -1.7787601411286225
Step 9: x = 1.8000000000000007, y = -2.287700244137904
Step 10: x = 1.9000000000000008, y = -2.888009960230758
Step 11: x = 2.000000000000001, y = -3.591402124423194
Step 12: x = 2.100000000000001, y = -4.410821420350552
Step 13: x = 2.200000000000001, y = -5.3605739348133365
Погрешность метода Рунге-Кутты 4-го порядка: 0.45764249490977793
```

(d) Результат работы метода Рунге-Кутты 4-го порядка

Рис. 8: Результаты работы методов для задачи Коши

Графическое показание методов: Затем можно создать таблицу, в которую вы сможете вносить полученные данные для каждого метода, включая значения и погрешности:

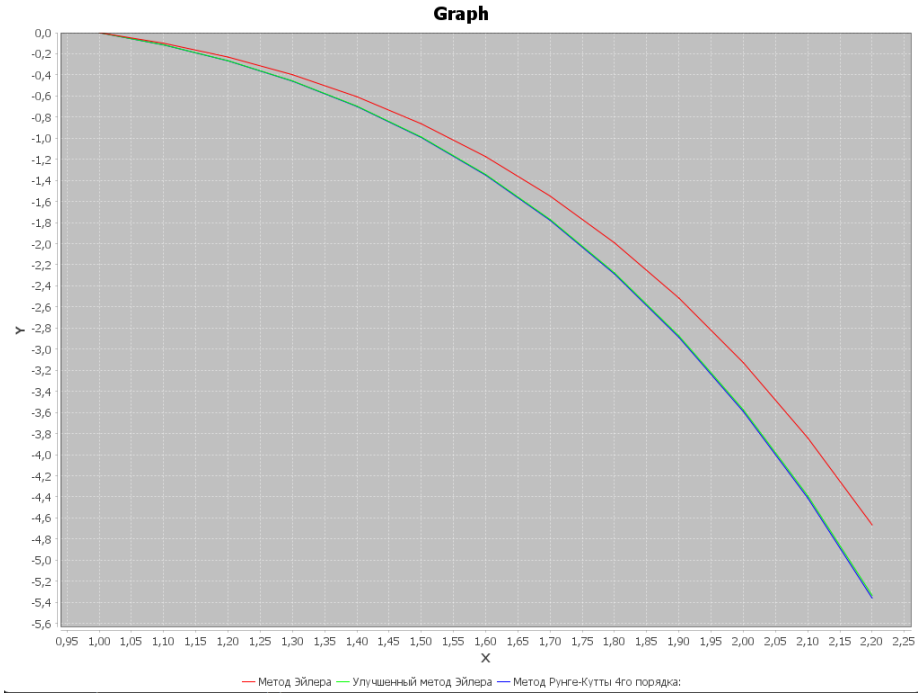

Рис. 9: Графическое решение задачи Коши

Таблица 5: Результаты методов

Шаг	**Метод Рунге-Кутты 4**	**Метод Эйлера**	**Улучшенный метод Эйлера**
1	$x = 1.0, y = 0.0$	$x = 1.0, y = 0.0$	$x = 1.0, y = 0.0$
2	$x = 1.1, y = -0.116$	$x = 1.1, y = -0.1$	$x = 1.1, y = -0.115$
3	$x = 1.2, y = -0.267$	$x = 1.2, y = -0.231$	$x = 1.2, y = -0.266$
4	$x = 1.3, y = -0.459$	$x = 1.3, y = -0.398$	$x = 1.3, y = -0.457$
5	$x = 1.4, y = -0.699$	$x = 1.4, y = -0.607$	$x = 1.4, y = -0.696$
6	$x = 1.5, y = -0.994$	$x = 1.5, y = -0.864$	$x = 1.5, y = -0.989$
7	$x = 1.6, y = -1.351$	$x = 1.6, y = -1.175$	$x = 1.6, y = -1.344$
8	$x = 1.7, y = -1.779$	$x = 1.7, y = -1.548$	$x = 1.7, y = -1.770$
9	$x = 1.8, y = -2.288$	$x = 1.8, y = -1.992$	$x = 1.8, y = -2.277$
10	$x = 1.9, y = -2.888$	$x = 1.9, y = -2.516$	$x = 1.9, y = -2.874$
11	$x = 2.0, y = -3.591$	$x = 2.0, y = -3.128$	$x = 2.0, y = -3.574$
12	$x = 2.1, y = -4.411$	$x = 2.1, y = -3.841$	$x = 2.1, y = -4.390$
13	$x = 2.2, y = -5.361$	$x = 2.2, y = -4.666$	$x = 2.2, y = -5.335$

Таблица 6: Погрешности методов

Метод	**Погрешность**
Метод Эйлера	0.025267741432554836
Улучшенный метод Эйлера	0.6945952787228968
Метод Рунге-Кутты 4-го порядка	6.065186663128941E-6

3 Вывод

В ходе выполнения данной курсовой работы были рассмотрены и решены следующие задачи: определение корней уравнения, решение СЛАУ, вычисление интеграла и использование интерполяционных многочленов. Кроме того, была решена задача Коши с применением различных численных методов.

Для определения корней уравнения графически была использована визуализация функции и выявление точек пересечения с осью абсцисс. Для уточнения одного из корней были применены итерационные методы: метод деления отрезка пополам, метод простой итерации и метод Ньютона. С помощью этих методов был достигнут заданный уровень точности в 0,001.

Для решения СЛАУ были применены соответствующие алгоритмы, такие как метод Гаусса или метод прогонки, в зависимости от характеристик системы. Результаты решения системы линейных алгебраических уравнений позволили найти значения неизвестных переменных.

Вычисление интеграла проводилось с использованием численных методов, таких как метод трапеций и метод Симпсона. Погрешности вычислений были оценены с помощью соответствующих методов, например, с использованием правила Рунге.

Интерполяционные многочлены степени от 1 до 4 были использованы для нахождения значений функции в заданных точках. Погрешность была оценена с учетом выбранной степени многочлена и сравнения с исходными значениями функции.

Наконец, задача Коши была решена с применением методов Эйлера, модифицированного метода Эйлера и метода Рунге-Кутты 4-го порядка. Результаты работы каждого метода были сопоставлены и погрешность была оценена с использованием правила Рунге Ромберга.

В результате выполнения данной курсовой работы были получены численные значения корней уравнения, решения СЛАУ, интеграла, значений функции с использованием интерполяции, а также решение задачи Коши. Оценка погрешностей позволяет учесть точность и достоверность полученных результатов.