**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Вычислительная математика**

**Лабораторная работа №3**

**Вариант 8**

Группа: P3266

Выполнили:

Тажибаева Е.В Коляда А.А

Проверила:

Машина Е. А.

Г. Санкт-Петербург

2024

**Цель работы**

Найти приближенное значение определенного интеграла с требуемой точностью различными численными методами .

**Порядок выполнения работы**

1. = = (2,3) =
2. Метод Ньютона-Котеса, n=6

f(x0)=f(a)=f(2)=-6

Разбиваем (2,3) на 6 равных отрезков

f(x1)=f(2,16)=-2,0417

f(x2)=2,8889

f(x3)=8,875

f(x4)=16

f(x5)=24,3472

f(x6)= f(3)=34

Погрешность : R = |10,58 – 10,58332| = 0,00332

1. Метод средних прямоугольников

n=10 h=

|  |  |  |
| --- | --- | --- |
| i | x | f(x) |
| 1 | 2,05 | -4,90963 |
| 2 | 2,15 | -2,47988 |
| 3 | 2,25 | 0,296875 |
| 4 | 2,35 | 3,438625 |
| 5 | 2,45 | 6,963375 |
| 6 | 2,55 | 10,88913 |
| 7 | 2,65 | 15,23388 |
| 8 | 2,75 | 20,01563 |
| 9 | 2,85 | 25,25238 |
| 10 | 2,95 | 30,96213 |

Погрешность: R=|10,58-10,56625|=0,01375

1. Метод трапеций n=10, h=0,1

f(x0)=f(2)=-6 f(x6)=f(2,6)=13,008

f(x1)=f(2,1)=-3,737 f(x7)=f(2,7)=17,569

f(x2)= f(2,2)=-1,136 f(x8)= f(2,8)=22,576

f(x3)= f(2,3)= 1,821 f(x9)=f(2,9)=28,047

f(x4)=f(2,4)=5,152 f(x10)=f(3)=34

f(x5)=f(2,5)=8,875

Погрешность: |10,58-10,6175|= 0,0375

1. Метод Смпсона n=10 h=0,1

Погрешность : R=|10,58-10,58333|=0,00333

**Вывод численной части:** В ходе работы было проведено сравнительное исследование различных численных методов для приближенного вычисления определенного интеграла

. Были использованы методы Ньютона-Котеса (n=6), средних прямоугольников (n=10), трапеций (n=10) и Симпсона (n=10).

Точное значение интеграла, полученное аналитически, составило 10,583.

Результаты:

• Метод Ньютона-Котеса (n=6): Полученное значение: 10,58332. Погрешность: 0,00332.

• Метод средних прямоугольников (n=10): Полученное значение: 10,56625. Погрешность: 0,01375.

• Метод трапеций (n=10): Полученное значение: 10,6175. Погрешность: 0,0375.

• Метод Симпсона (n=10): Полученное значение: 10,58333. Погрешность: 0,00333.

**Листинг программы**

Базовый класс

public abstract class MethodBaseLb3 implements ISMethod{  
 */\*\*  
 \* результат расчета  
 \*/* protected double result = 0;  
 */\*\*  
 \* Количество разбиений интервала интегрирования для достижения  
 \* заданной точности  
 \*/* protected int lastIntervals = 0;  
  
 */\*\*  
 \* Проверка входных параметров  
 \* @param params  
 \*/* @Override  
 public void checkParams(CalcParams params) {  
 if (params.startingDelta <= 0) throw new CalcErrorException("Число интервалов должно быть больше 0");  
 if (params.xa == params.xb) throw new CalcErrorException("Пределы интегрирования не могут быть равны");  
 }  
  
 */\*\*  
 \* Метод рассчитывает интеграл с заданной точностью по методу Рунге  
 \* @param fun подинтегральная функция  
 \* @param params заданные параметры (интервал интегрирования и число разбиения интервала  
 \* @return  
 \*/* @Override  
 public double calculate(ISFunction fun, CalcParams params) {  
  
 //Проверяем параметры  
 checkParams(params);  
  
 var newIntervals = fun.splitInterval(params);  
  
 double localResult = 0;  
 for(var p: newIntervals) {  
 localResult += calculateInterval(fun, p);  
 }  
 result = localResult;  
 return result;  
 }  
  
 private double calculateInterval(ISFunction fun, CalcParams params) {  
  
 //Количество разбиений интервала указано в double (наследство от laba2)  
 // Таким образом мы получаем целое число из него  
 int n = (int)(params.startingDelta + 0.5);  
 //Результат на предыдущей итерации расчета  
 double previousResult;  
 //Устанавливаем начальное значение в несуществующее число  
 result = Double.*NaN*;  
 do {  
 previousResult = result;  
 //Получаем массив x для текущего количества разбиений  
 //После этого получаем для этих x массив y - значений функцй  
 var ys = fun.getValuesOf(getXs(params.xa, params.xb, n));  
 //Дальше рассчитываем интеграл методом наследником  
 result = calcS(params,ys,n);  
 //Запоминаем поледний N - количество разбиений  
 lastIntervals = n;  
 //Удваиваем количество разбиений для следующего расчета  
 n \*= 2;  
 //Продолжаем цикл пока нет предыдущего результат (первая итерация) или пока  
 // не достигнем нужной точности  
 } while(Double.*isNaN*(previousResult) || Math.*abs*(previousResult - result) > params.precision);  
  
 //Вернем результат расчета  
 return result;  
  
 }  
  
 */\*\*  
 \* Абстрактная функция для расчета интеграла для массива y-ов  
 \* @param params параметры расчета  
 \* @param ys массив значений функции  
 \* @param n количество разбиений  
 \* @return результат расчет интеграла  
 \*/* abstract double calcS(CalcParams params, double[]ys,int n);  
  
 */\*\*  
 \* Строит массив x для указанного интервала и количество разбиений  
 \* В такой реализации метод может быть использован для всех методов расчета интеграла,  
 \* кроме средних прямоугольков. Метод стредних прямоугольников должен переопределить  
 \* этот метод  
 \* @param xa нижняя граница интервала интегрирования  
 \* @param xb верхняя граница интервала интегрирования  
 \* @param n количество разбиений интервала  
 \* @return массив значений х  
 \*/* protected double[] getXs(double xa, double xb, int n){  
 double h = (xb - xa) / n;  
 double[] xs = new double[n + 1];  
 // Формируем массив xs  
 double x = xa;  
 for(int i = 0; i<n;i++){  
 xs[i] = x;  
 x += h;  
 }  
 xs[n] = xb;  
 return xs;  
 }  
  
 */\*\*  
 \* Журнал для вывода сообщений  
 \*/* protected ILogger logger;  
  
 */\*\*  
 \* Setter для журнала  
 \* @param log  
 \*/* @Override  
 public void setLogger(ILogger log) {  
 logger = log;  
 }  
  
 */\*\*  
 \* Getter для результата расчета интеграла  
 \* @return результат расчета  
 \*/* @Override  
 public double getCalculatedRoot() {  
 return result;  
 }  
  
 */\*\*  
 \* Getter для количества разбиений интервла интегрирования  
 \* @return использованное количество разбиений интервала интегрирования  
 \*/* @Override  
 public int getUsedIntervals() {  
 return lastIntervals;  
 }  
  
}

Метод левых прямоугольников

public class MRectangleLeft extends MethodBaseLb3 {  
  
 @Override  
 public String toString() {  
 return "Левых прямоугольников";  
 }  
 protected double calcS(CalcParams params, double[]ys,int n){  
 double res = 0;  
 for (int i = 1; i <= n; i++) res += ys[i - 1];  
 double h = (params.xb - params.xa) / n;  
 res \*= h;  
 return res;  
 }

Метод правых прямоугольников

public class MRectangleRight extends MethodBaseLb3 {  
 @Override  
 public String toString() {  
 return "Правых прямоугольников";  
 }  
  
 @Override  
 double calcS(CalcParams params, double[] ys, int n) {  
 double res = 0;  
 for (int i = 1; i <= n; i++) res += ys[i];  
 double h = (params.xb - params.xa) / n;  
 res \*= h;  
 return res;  
 }  
}

Метод средних прямоугольников

public class MRectangleMiddle extends MethodBaseLb3{  
 @Override  
 public String toString() {  
 return "Средних прямоугольников";  
 }  
  
 @Override  
 double calcS(CalcParams params, double[] ys, int n) {  
 double res = 0;  
 for (int i = 0; i < n; i++) res += ys[i];  
 double h = (params.xb - params.xa) / n;  
 res \*= h;  
 return res;  
 }  
  
 @Override  
 protected double[] getXs(double xa, double xb, int n) {  
 double h = (xb - xa) / n;  
 double[] xs = new double[n];  
 // Формируем массив xs  
 double x = xa + h/2;  
 for(int i = 0; i<n-1;i++){  
 xs[i] = x;  
 x += h;  
 }  
 xs[n-1] = xb -h/2;  
 return xs;  
 }  
}

Метод Симпсона

public class MSimpson extends MethodBaseLb3{  
 @Override  
 public String toString() {  
 return "Симпсона";  
 }  
  
 */\*\*  
 \* Переопределяем метод проверки параметров, чтобы убедиться, что N - четное число  
 \* @param params  
 \*/* @Override  
 public void checkParams(CalcParams params) {  
 int n = (int)(params.startingDelta + 0.5);  
 if(n == 0 || (n % 2) != 0) throw new CalcErrorException("Для метода симпсона N должно быть четным числом");  
 //Для остальных проверок вызываем реализацию от super класса  
 super.checkParams(params);  
 }  
  
 @Override  
 double calcS(CalcParams params, double[] ys, int n) {  
 double h = (params.xb - params.xa)/n;  
 return h/3\*(ys[0] + 4 \* ysum(1,n-1,ys) + 2 \* ysum(2,n-2,ys) + ys[n]);  
  
 }  
  
 */\*\*  
 \* Вспомогательный метод для расчета среза массива или по четным или по нечетным.  
 \* @param left левая гарница среза  
 \* @param right правая граница среза  
 \* @param ys массив y  
 \* @return сумма по срезу  
 \*/* private double ysum(int left,int right, double [] ys){  
 double res = 0;  
 for(int i = left ; i <= right;i+=2) res += ys[i];  
 return res;  
 }  
}

Метод трапеций

public class MTrapeziod extends MethodBaseLb3{  
 @Override  
 public String toString() {  
 return "Трапеций";  
 }  
  
 @Override  
 double calcS(CalcParams params, double[] ys, int n) {  
 double h = (params.xb - params.xa)/n;  
 double res = ((ys[0] + ys[ys.length-1]))/2;  
 for(int i = 1; i < n-1;i++) res += ys[i];  
 res \*= h;  
 return res;  
 }  
}

**Результаты работы программы**

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание**

**Выводы**

• Точность: Методы Ньютона-Котеса и Симпсона показали наибольшую точность, обеспечивая наименьшую погрешность по сравнению с другими методами.

• Сложность: Метод Симпсона требует больше вычислений, чем метод средних прямоугольников и трапеций, но обеспечивает более высокую точность. Метод Ньютона-Котеса, при n = 6, не требует значительных вычислений, но его точность меньше, чем у метода Симпсона.

• Выбор метода: Выбор метода зависит от требуемой точности и доступных вычислительных ресурсов. Если требуется высокая точность, рекомендуется использовать метод Симпсона или метод Ньютона-Котеса с большим количеством узлов. Если требуется более простое решение с допустимой погрешностью, можно использовать методы средних прямоугольников или трапеций.

В целом, полученные результаты подтверждают, что численные методы являются эффективным инструментом для приближенного вычисления определенных интегралов. Выбор наиболее подходящего метода зависит от конкретной задачи и требуемой точности.