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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа **№3**

**«Численное интегрирование»**

по дисциплине «Вычислительная математика**»**

Вариант: **4**

**Преподаватель:**   
Наумова Надежда Александровна

**Выполнил:**

Кирячек Тимофей Алексеевич

**Группа:** Р3209

Санкт-Петербург, 2025 г.

Цель работы: найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

# 1. Вычислительная реализация задачи

1. **Вычислить интеграл**, приведенный в таблице 1, **точно:**

f(x)

Изображение выглядит как График, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Вычислить интеграл по формуле Ньютона–Котеса** при :

[Решение на Wolfram Alpha](https://www.wolframalpha.com/input?i2d=true&i=%5C%2840%29%5C%2840%29-1%5C%2841%29-%5C%2840%29-3%5C%2841%29%5C%2841%29%C3%97%5C%2840%29Divide%5B41%2C840%5D+f%5C%2840%29-3%5C%2841%29%2BDivide%5B216%2C840%5D+f%5C%2840%29Divide%5B%5C%2840%29-8%5C%2841%29%2C3%5D%5C%2841%29%2BDivide%5B27%2C840%5D+f%5C%2840%29Divide%5B%5C%2840%29-7%5C%2841%29%2C3%5D%5C%2841%29%2BDivide%5B272%2C840%5D+f%5C%2840%29-2%5C%2841%29%2BDivide%5B27%2C840%5D+f%5C%2840%29Divide%5B%5C%2840%29-5%5C%2841%29%2C3%5D%5C%2841%29%2BDivide%5B216%2C840%5D+f%5C%2840%29Divide%5B%5C%2840%29-4%5C%2841%29%2C3%5D%5C%2841%29%2BDivide%5B41%2C840%5Df%5C%2840%29-1%5C%2841%29%5C%2841%29+with+f%5C%2840%29x%5C%2841%29+%3D+-3Power%5Bx%2C3%5D+-+5Power%5Bx%2C2%5D+%2B+4x+-+2)

1. **Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона** при :

* **Метод средних прямоугольников**:

[Решение на Wolfram Alpha](https://www.wolframalpha.com/input?i2d=true&i=0.2%5C%2840%29f%5C%2840%29-3%2B0.1%5C%2841%29%2Bf%5C%2840%29-3%2B0.3%5C%2841%29%2Bf%5C%2840%29-3%2B0.5%5C%2841%29%2Bf%5C%2840%29-3%2B0.7%5C%2841%29%2Bf%5C%2840%29-3%2B0.9%5C%2841%29%2Bf%5C%2840%29-3%2B1.1%5C%2841%29%2Bf%5C%2840%29-3%2B1.3%5C%2841%29%2Bf%5C%2840%29-3%2B1.5%5C%2841%29%2Bf%5C%2840%29-3%2B1.7%5C%2841%29%2Bf%5C%2840%29-3%2B1.9%5C%2841%29%5C%2841%29+with+f%5C%2840%29x%5C%2841%29+%3D+-3Power%5Bx%2C3%5D+-+5Power%5Bx%2C2%5D+%2B+4x+-+2)

* **Метод трапеций**:

[Решение на Wolfram Alpha](https://www.wolframalpha.com/input?i2d=true&i=0.2%5C%2840%29Divide%5B%5C%2840%29f%5C%2840%29-3%5C%2841%29%2Bf%5C%2840%29-1%5C%2841%29%5C%2841%29%2C2%5D%2B+f%5C%2840%29-3%2B0.2%5C%2841%29%2B+f%5C%2840%29-3%2B0.4%5C%2841%29%2B+f%5C%2840%29-3%2B0.6%5C%2841%29%2B+f%5C%2840%29-3%2B0.8%5C%2841%29%2B+f%5C%2840%29-3%2B1%5C%2841%29%2B+f%5C%2840%29-3%2B1.2%5C%2841%29%2B+f%5C%2840%29-3%2B1.4%5C%2841%29%2B+f%5C%2840%29-3%2B1.6%5C%2841%29%2B+f%5C%2840%29-3%2B1.8%5C%2841%29%5C%2841%29+with+f%5C%2840%29x%5C%2841%29+%3D+-3Power%5Bx%2C3%5D+-+5Power%5Bx%2C2%5D+%2B+4x+-+2)

* **Метод Симпсона**:

[Решение на Wolfram Alpha](https://www.wolframalpha.com/input?i=0.2%2F3+%28f%28-3%29%2B4*%28f%28-3%2B0.2%29%2Bf%28-3%2B0.6%29%2Bf%28-3%2B1%29%2Bf%28-3%2B1.4%29%2Bf%28-3%2B1.8%29%29%2B2*%28f%28-3%2B0.4%29%2Bf%28-3%2B0.8%29%2Bf%28-3%2B1.2%29%2B+f%28-3%2B1.6%29%29%2Bf%28-1%29%29+with+f%28x%29+%3D+-3x%5E3+-+5x%5E2+%2B+4x+-+2)

1. **Сравнить результаты с точным значением интеграла:**

Точное значение интеграла на интервале вычислено как

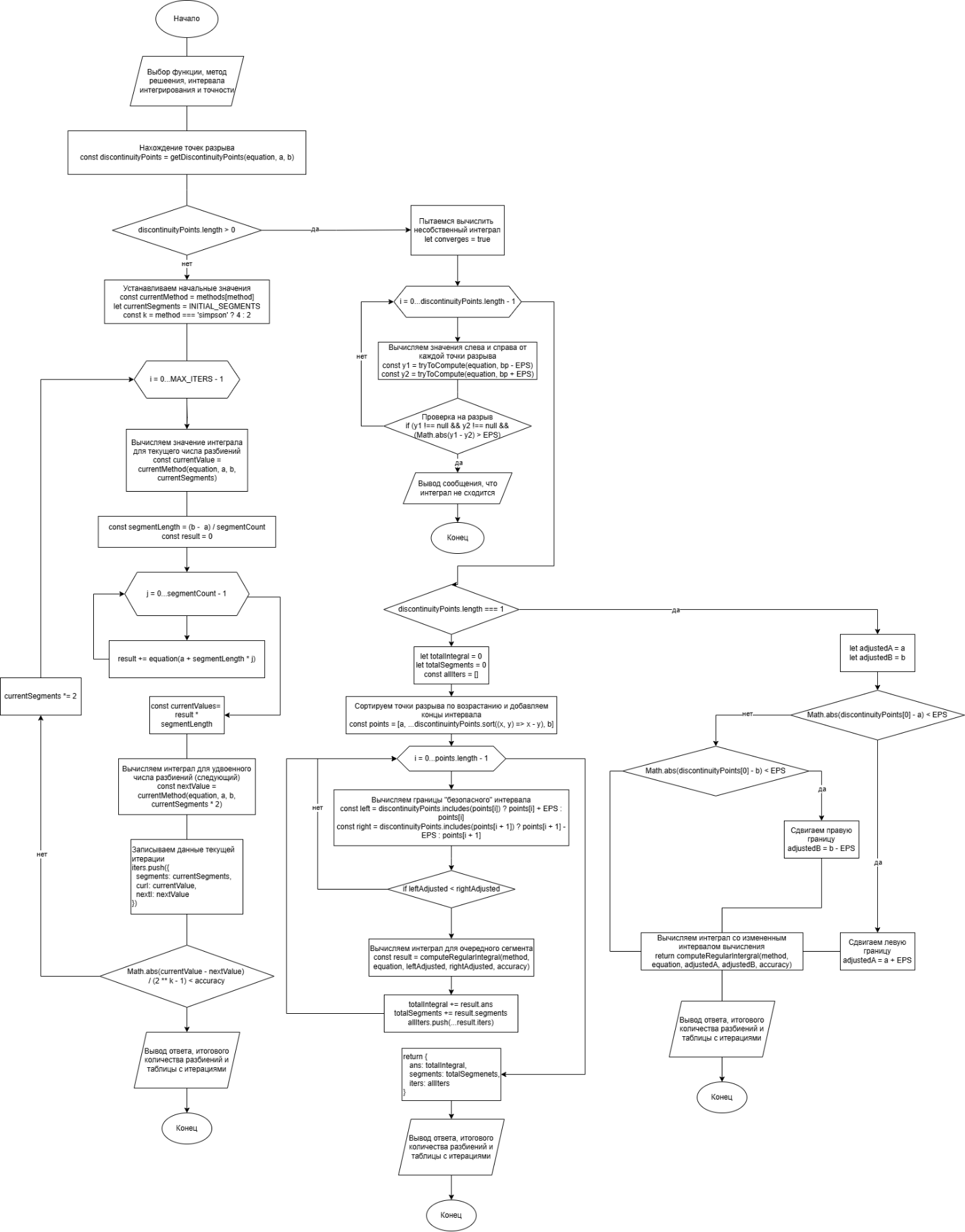
1. Для метода **Ньютона–Котеса** при : , **значения совпадают**.
2. Для метода **средних прямоугольников** при : .
3. Для метода **трапеций** при : .
4. Для метода **Симпсона** при : , **значения совпадают**.
5. **Определить относительную погрешность вычислений для каждого метода.**
6. Для метода **Ньютона–Котеса**: **погрешности нет.**
7. Для метода **средних прямоугольников**:
8. Для метода **трапеций**:
9. Для метода **Симпсона**: **погрешности нет.**

Как видно из результатов, все методы дали относительно малую погрешность, особенно при использовании формулы Ньютона–Котеса и Симпсона. Наилучший результат был получен при использовании формулы Ньютона–Котеса с и формулы Симпсона с , при которых значения интеграла полностью совпали.

# 2. Программная реализация задачи

**Блок-схема**

Вычисление интеграла на примере метода левых прямоугольников



**Листинг программы**

[**https://github.com/TecHeReTiC3141/CalcMaths\_Lab3.git**](https://github.com/TecHeReTiC3141/CalcMaths_Lab3.git)

**utils.ts**

sdfsd

const rectangleLeftMethod = (equation: Equation, a: number, b: number, segmentCount: number) => {  
 const segmentLength = (b - a) / segmentCount  
 const data = Array.from({ length: segmentCount },  
 (\_, i) => a + segmentLength \* i  
 );  
  
 return segmentLength \* data.reduce((acc, x) => acc + equation(x), 0)  
}  
  
const rectangleRightMethod = (equation: Equation, a: number, b: number, segmentCount: number) => {  
 const segmentLength = (b - a) / segmentCount  
 const data = Array.from({ length: segmentCount },  
 (\_, i) => a + (i + 1) \* segmentLength  
 );  
  
 return segmentLength \* data.reduce((acc, x) => acc + equation(x), 0)  
}  
  
const rectangleCenterMethod = (equation: Equation, a: number, b: number, segmentCount: number) => {  
 const segmentLength = (b - a) / segmentCount  
 const data = Array.from({ length: segmentCount },  
 (\_, i) => a + (i + 0.5) \* segmentLength  
 );  
  
 return segmentLength \* data.reduce((acc, x) => acc + equation(x), 0)  
}  
const trapezoidMethod = (equation: Equation, a: number, b: number, segmentCount: number) => {  
 const segmentLength = (b - a) / segmentCount  
 const data = Array.from({ length: segmentCount + 1 },  
 (\_, i) => a + i \* segmentLength  
 );  
  
 return segmentLength / 2 \* (data[ 0 ] + data[ data.length - 1 ] + 2 \* data.slice(1, -1).reduce((acc, x) => acc + equation(x), 0))  
}  
  
const simpsonMethod = (equation: Equation, a: number, b: number, segmentCount: number) => {  
 const segmentLength = (b - a) / segmentCount  
 const data = Array.from({ length: segmentCount + 1 },  
 (\_, i) => a + i \* segmentLength  
 );  
 let evenSum = 0  
 let oddSum = 0  
 data.slice(1, -1).forEach((point, index) => {  
 if (index % 2 === 0) {  
 oddSum += equation(point)  
 } else {  
 evenSum += equation(point)  
 }  
 })  
 return segmentLength / 3 \* (data[ 0 ] + data[ data.length - 1 ] + 4 \* oddSum + 2 \* evenSum)  
}  
  
const methods: Record<IntegralSolvingMethod, MethodFunction> = {  
 [ IntegralSolvingMethod.RectangleLeft ]: rectangleLeftMethod,  
 [ IntegralSolvingMethod.RectangleRight ]: rectangleRightMethod,  
 [ IntegralSolvingMethod.RectangleCenter ]: rectangleCenterMethod,  
 [ IntegralSolvingMethod.Trapezoid ]: trapezoidMethod,  
 [ IntegralSolvingMethod.Simpson ]: simpsonMethod  
};  
  
type SolutionIter = {  
 segments: number  
 curI: number  
 nextI: number  
}  
  
export type SolutionData = {  
 ans: number  
 segments: number  
 iters: SolutionIter[]  
}  
  
const getDiscontinuityPoints = (equation: Equation, a: number, b: number): number[] => {  
 const breakpoints = Array.from<number>({ length: SEGMENTS + 1 })  
 .map((\_, index) => a + (b - a) \* index / SEGMENTS)  
 .filter((point) => !isFinite(equation(point)))  
  
 return breakpoints  
}  
  
const handleImproperIntegral = (  
 method: IntegralSolvingMethod,  
 equation: Equation,  
 a: number,  
 b: number,  
 accuracy: number,  
 discontinuityPoints: number[]  
): SolutionData | ValidationError => {  
 console.log(`! Found discontinuity points: function is discontinuous or undefined at ${discontinuityPoints}`);  
  
 const EPS = 0.00001;  
 let converges = true;  
  
 for (const bp of discontinuityPoints) {  
 const y1 = tryToCompute(equation, bp - EPS);  
 const y2 = tryToCompute(equation, bp + EPS);  
  
 if (y1 !== null && y2 !== null && (Math.abs(y1 - y2) > EPS || (y1 === y2 && y1 !== null))) {  
 converges = false;  
 break;  
 }  
 }  
  
 if (!converges) {  
 console.log('- Integral does not exist: integral does not converge');  
 return ValidationError.noConvenge;  
 }  
  
 console.log('+ Integral converges');  
  
 let totalIntegral = 0;  
 let totalSegments = 0;  
 const allIters: SolutionIter[] = [];  
  
 // Handle single discontinuity case  
 if (discontinuityPoints.length === 1) {  
 let adjustedA = a;  
 let adjustedB = b;  
  
 if (Math.abs(discontinuityPoints[ 0 ] - a) < EPS) {  
 adjustedA = a + EPS;  
 } else if (Math.abs(discontinuityPoints[ 0 ] - b) < EPS) {  
 adjustedB = b - EPS;  
 }  
  
 const result = computeRegularIntegral(method, equation, adjustedA, adjustedB, accuracy);  
 if (checkIfValidationError(result)) return result;  
  
 totalIntegral += result.ans;  
 totalSegments += result.segments;  
 allIters.push(...result.iters);  
 }  
 // Handle multiple discontinuities  
 else {  
 // Sort discontinuities and add endpoints  
 const points = [ a, ...discontinuityPoints.sort((x, y) => x - y), b ];  
  
 // Integrate between each pair of adjacent points  
 for (let i = 0; i < points.length - 1; i++) {  
 const left = points[ i ];  
 const right = points[ i + 1 ];  
  
 // Check if endpoints are discontinuities  
 const leftAdjusted = discontinuityPoints.includes(left) ? left + EPS : left;  
 const rightAdjusted = discontinuityPoints.includes(right) ? right - EPS : right;  
  
 // Only compute if the adjusted interval is valid  
 if (leftAdjusted < rightAdjusted) {  
 const result = computeRegularIntegral(method, equation, leftAdjusted, rightAdjusted, accuracy);  
 if (checkIfValidationError(result)) return result;  
  
 totalIntegral += result.ans;  
 totalSegments += result.segments;  
 allIters.push(...result.iters);  
 }  
 }  
 }  
  
 return {  
 ans: totalIntegral,  
 segments: totalSegments,  
 iters: allIters  
 };  
};  
  
// Helper function to safely compute function values  
const tryToCompute = (equation: Equation, x: number): number | null => {  
 try {  
 const result = equation(x);  
 return isFinite(result) ? result : null;  
 } catch {  
 return null;  
 }  
};  
  
const computeRegularIntegral = (  
 method: IntegralSolvingMethod,  
 equation: Equation,  
 a: number,  
 b: number,  
 accuracy: number  
): SolutionData | ValidationError => {  
 const currentMethod = methods[ method ];  
 let currentSegments = INITIAL\_SEGMENTS;  
 const k = method === IntegralSolvingMethod.Simpson ? 4 : 2;  
 const iters: SolutionIter[] = [];  
  
 const MAX\_ITERS = 20  
 let iter = 0  
  
 while (iter++ < MAX\_ITERS) {  
 const currentValue = currentMethod(equation, a, b, currentSegments);  
 const nextValue = currentMethod(equation, a, b, currentSegments \* 2);  
  
 iters.push({  
 segments: currentSegments,  
 curI: currentValue,  
 nextI: nextValue  
 });  
  
 if (Math.abs(currentValue - nextValue) / (2 \*\* k - 1) < accuracy) {  
 return {  
 ans: nextValue,  
 segments: currentSegments \* 2,  
 iters  
 };  
 }  
  
 currentSegments \*= 2;  
 }  
  
 return ValidationError.noConvenge  
};  
  
export const solveEquation = (method: IntegralSolvingMethod, equation: Equation, a: number, b: number, accuracy: number): SolutionData | ValidationError => {  
 const discontinuityPoints = getDiscontinuityPoints(equation, a, b)  
  
 if (discontinuityPoints.length) {  
 return handleImproperIntegral(method, equation, a, b, accuracy, discontinuityPoints)  
 }  
  
 return computeRegularIntegral(method, equation, a, b, accuracy)  
}  
  
export const checkIfValidationError = (s: ValidationError | SolutionData): s is ValidationError => {  
 return Object.values(ValidationError).includes(s as ValidationError)  
}

**Пример работы программы:**

|  |
| --- |
| Уравнение: Выбираем уравнение из возможных вариантов  Вводим границы интервала в соответствующие поля: -1, 5  Выбираем способ решения – метод средних прямоугольников  Указываем желаемую точность в соответствующее поле  Нажимаем на кнопку Найти решение |

# Вывод

В ходе выполнения лабораторной работы были изучены численные методы интегрирования. В результате работы было разработано веб-приложение, реализующее различные численные методы вычисления определенных интегралов: метод прямоугольников (левых, правых, средних), метод трапеций и метод Симпсона.

В ходе вычислительной реализации задачи были рассчитаны интегралы различными методами и проведено сравнение результатов с точными значениями интегралов.

Также была выполнена дополнительная задача по установлению сходимости рассматриваемых несобственных интегралов 2 рода и их вычислению заданными численными методами в случаях, когда подынтегральная функция терпит бесконечный разрыв в точке a, в точке b или на отрезке интегрирования.