Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт математики и механики им. Н. И. Лобачевского
Направление: 01.03.01 – Математика

КОНТРОЛЬНАЯ РАБОТА ПО ЧИСЛЕННЫМ МЕТОДАМ №3
(научно-исследовательская работа)

Вариант № 5

Обучающийся: Греков Лев Евгеньевич 05-104

Преподаватель:
доцент, к.н. Насибуллин Рамиль Гайсаевич

Дата сдачи контрольной: ______________

Казань – 2023

Задания

1. Вычислить интеграл по формулам левых и правых прямоугольников
2. Вычислить интеграл по формуле средних прямоугольников
3. Вычислить с помощью квадратурной формулы Симпсона.
4. Вычислить с помощью квадратурной формулы Гаусса $n = 6$.
5. Вычислить с помощью интерполяционной квадратурной формулы $n = 10$.
6. Вычислить с помощью квадратурной формулы Гаусса $n = 6, p(x) = \frac{1}{\sqrt{1-x^2}}$.
7. Оценить погрешность для каждого случая.

$$I_1 = \int_{1.2}^{2.0} \frac{\sqrt{2x^2 + 1.6}dx}{2x + \sqrt{0.5x^2 + 3}} \qquad I_2 = \int_{0.5}^{1.0} \frac{\sin(0.5x + 0.4)dx}{1.2 + cos(x^2 + 0.4)}$$

1 Левые и Правые Прямоугольники

Начну с того, что я создал класс $NumericIntegral$, Который содержит поля с пределами интегрирования, подинтегральную функцию. Он реализует методы, находящиие значения квадратурных формул

```
class NumericIntegral(
    private val function: (Double) -> Double,
    lowerBound: Double,
    upperBound: Double,
) {
    private val a: Double
    private val b: Double

    init {
        this.a = minOf(lowerBound, upperBound)
        this.b = maxOf(lowerBound, upperBound)
    }
    ...
```

Первые 2 метода реализуют квадратурные формулы Левых и правых прямоугольников.

```
    fun leftRectangleMethod(n: Int): Double {
        val h = (b - a) / n
        val points = (0 until n)
            .map {i-> a + i * h }
            .associateWith { x -> function(x) }
        return points.values.sum() * h
    }

    fun rightRectangleMethod(n: Int): Double {
        val h = (b - a) / n
        val points = (1..n)
            .map {i-> a + i * h }
            .associateWith { x -> function(x) }
        return points.values.sum() * h
    }
```

По сути это програмное представление формул:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left[f_0 + f_1 + \ldots + f_{n-1}\right]$$

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left[f_1 + f_2 + \ldots + f_n\right]$$

Для левых и правых прямоугольников соотвветственно.
Результаты вычислений x_k приводить не буду, так как считаю, что они создадут визуальный шум
P.S В "ПРИЛОЖЕНИИ"загружу полный вывод консоли моей программы, там можно посмотреть промежуточные вычисления точек и остальные подсказки, которые я оставлял для себя.

В резульате вычислений получились значения (Для двух интегралов I_1, I_2):

Левые Прямоугольники: $I_1 \approx 0.3935142817035969$ $I_2 \approx 0.41690958455423466$
Правые Прямоугольники: $I_1 \approx 0.39414496939930116$ $I_2 \approx 0.49125191356242043$

2 Средние Прямоугольники

По такому же принципу как выше реализуем формулу:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left[\frac{f_{1/2}+f_{3/2}+\ldots+f_{n-1/2}}{2}\right]$$

```
    fun middleRectangleMethod(n: Int): Double {
        val h = (b - a) / n
        val vals =  (0 until n)
            .map { a + it * h }
            .map { (it + (it + h)) / 2 }
            .associateWith { x -> function(x) }

        return vals.values.sum() * h
    }
```

Код может выглядеть сложным, но на самом деле *vals* представляет собой массив, ассоциирующий $\frac{x_k+x_{k+1}}{2}$ с $f(\frac{x_k+x_{k+1}}{2})$.

Получаем резульате вычислений

Средние Прямоугольники: $I_1 \approx 0.39382072329248163 \quad I_2 \approx 0.45088605222312456$

3 Метод Симпсона

Для приближенного вычисления интегралов методом Симсона я использовал не классическую формулу Симпсона, а её иное представление: Оно визуально понятнее и легче програмируемое

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a)+f(b)+4\sum_{i=1,3,5,\ldots}^{n-1} f(a+i\frac{b-a}{n})+2\sum_{i=2,4,6,\ldots}^{n-2} f(a+i\frac{b-a}{n})\right]$$

Метод Симпсона: $I_1 \approx 0.3938237302855468 \qquad I_2 \approx 0.45198477254620356$

```
    fun simpsonMethod(n: Int): Double {
        val h = (b - a) / n
        val xValues = (0 until n).map { a + it * h }
        val first = function(a) + function(b)
        val second = 4 * xValues.filterIndexed { index, _ -> index % 2 == 1 }.sumOf { function(it) }
        val third = 2 * xValues.filterIndexed { index, _ -> index % 2 == 0 && index != 0 && index != n }.sumOf { function(it) }
        return (first + second + third) * h/3.0}
```

4 квадратурная формулы Гаусса с $\rho = 1$

Сразу приведу програмную реализацию. Далее будет разбор

```
    fun gaussianMethodWithRo1(n: Int): Double{
        var result = 0.0
        val legendrePoly = LegendrePolynomial(n)
        val roots = OptimizationMethods.findRoots(-1.0, 1.0, legendrePoly::invoke, legendrePoly.derivative(1)::invoke)
        if (roots.size != legendrePoly.size) throw Exception("
                                                  ")
        val mappedRoots = LegendrePolynomial
         .mapInterval(roots,a,b)
         .forEachIndexed{i, xk ->
             val Ak = LagrangePolynomial.createFundamentalPoly(mappedRoots,xk).rhimanIntegral(a,b)
             result += Ak * function(xk)
         }
        return result}
```

Для решения этого задания с создал класс Полинома Лежандра. Ничего хитрого в его реализации нет

```
class LegendrePolynomial(degree: Int, ck: Double = 1.0) : Polynomial() {
    init {
        _coeffs.clear()
        val a =  Polynomial(mapOf(0 to -1.0,2 to 1.0)).pow(degree).derivative(degree) * ck
        _coeffs.putAll(a.coeffs)
    }
    companion object{
        fun mapInterval(t: Double, a: Double, b: Double): Double {
            if (a eq b) return a
            val (minValue, maxValue) = if (a < b) a to b else b to a
            return (minValue + maxValue)/2.0 + (maxValue-minValue) * t /2.0
        }

        fun mapInterval(list: List<Double>, a: Double, b: Double): List<Double> = list.map { mapInterval(it,a,b) }
    }
}
```

Далее нам нужно найти корни полинома Лежандра. Это меня озадачило! Чтобы не пользоваться онлайн-калькуляторами, я решил запрограмировать поиск корней численно. Я не уверен в правильности реализации моего метода. Скорее всего он не будет рабоать на больших интервалах и в случаях, когда функция слишком часто знакочередуется.

Идея следующая: Пробегаемся по интервалу $[a, b]$ с очень маленьким шагом ϵ и храним значения $x_i x_{i-1}$ Если на них разный знак, значит в отрезке $[x_i, x_{i-1}]$ находится Корень. Применяем метод Ньютона для приближенного нахождения корня на этом маленьком промежутке. Повторюсь: реализация не лучшая, но мне не удалось найти готовые методы нахождения корней нелинейных уравнений. И с решением задачи в контексте данного задания, мой метод неплохо справляется

```
fun findRoots(a: Double, b: Double, function: (Double)
    -> Double, derivative: (Double) -> Double, epsilon:
    Double = 10e-6): List<Double> {
    val (start, end) = if (a <= b) a to b else b to a
    var prev = start
    var curr = prev + epsilon
    val roots: MutableList<Double> = mutableListOf()
    while (curr <= end) {
        if(sgn(function(curr)) != sgn(function(prev))){
            val root = newtonMethod(curr,function,
                derivative)
            roots.add(root)
        }
        prev = curr
        curr+= epsilon
    }
    return roots
}

private fun newtonMethod(initialGuess: Double, function
    : (Double) -> Double, derivative: (Double) -> Double,
     tol: Double = 1e-7, maxIter: Int = 1000): Double {
    var x = initialGuess
    var iteration = 0

    while (abs(function(x)) > tol && iteration <
        maxIter) {
        x -= function(x) / derivative(x)
        iteration++
    }
    return x
}
```

После того как мы нашли корни Полинома Лежандра, нужно их отобразить из $[-1, 1]$ в $[a, b]$, используя формулу $x = \frac{a+b}{2} + \frac{b-a}{2} * t$
(Результаты вычислений корней и их перевод в $[a, b]$ также в приложении) Далее находим $A_k = \int_a^b \rho(x) l_k(x)$, где $l_k(x)$ - фундаментальный полином Лагранжа Специально для этих целей, я реализовал Функцию Первообразной для класса Polynomial:

```
fun antiderivative(): Polynomial {
    val antiderivativeCoeffs = mutableMapOf<Int, Double>()
    for ((exponent, coefficient) in _coeffs) {
        val newExponent = exponent + 1
        val newCoefficient = coefficient / newExponent.
           toDouble()

        antiderivativeCoeffs[newExponent] = newCoefficient
    }
    return Polynomial(antiderivativeCoeffs)
}
```

Пользуемся формулой и получаем конечное значение

$$\int_a^b \rho(x) f(x)\, dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

Квадратурная формула Гаусса с $\rho = 1$**:** $I_1 \approx 0.3938236880501 \quad I_2 \approx 0.451948735871309$

5 Интерполяционная формула

Для решения этого задания, я создаю полином Лагранжа по набору точек и беру от него интеграл.

```
    fun interpolationMethod(n: Int): Double{
        val h = (b - a) / (n-1)
        val points = (0 until n)
            .map {i-> a + i * h }
            .associateWith { x -> function(x) }
        val poly = NewtonPolynomial(points)
        return poly.rhimanIntegral(a,b)
    }
```

Полином Лагранжа и Его первообразная

Для **первого Интеграла**:

$$L_{10}(f,x) = -0,000166x^9 + 0,002867x^8 - 0,022604x^7 + 0,107082x^6 - 0,338593x^5 + 0,748090x^4$$
$$-1,166651x^3 + 1,246119x^2 - 0,812185x + 0,726210$$
$$F(L_{10}(f,x)) = -0,000017x^{1}0 + 0,000319x^9 - 0,002825x^8 + 0,015297x^7 - 0,056432x^6$$
$$+0,149618x^5 - 0,291663x^4 + 0,415373x^3 - 0,406093x^2 + 0,726210x$$

Для **второго Интеграла**:

$$L_{10}(f,x) = 5.601606x^9 - 38.484725x^8 + 118.997407x^7 - 215.554201x^6 + 251.396774x^5 -$$
$$-195.124561x^4 + 100.779358x^3 - 33.284919x^2 + 6.602312x - 0.357266$$
$$F(L_{10}(f,x)) = 0.560161x^{10} - 4.276081x^9 + 14.874676x^8 - 30.793457x^7 + 41.899462x^6 -$$
$$-39.024912x^5 + 25.194839x^4 - 11.094973x^3 + 3.301156x^2 - 0.357266x$$

Интерполяционная формула $\rho = 1$: $I_1 \approx 0.39382368806290$ $\quad I_2 \approx 0.45194867933117$

6 Квадратурная формулы Гаусса с $\rho = \frac{1}{\sqrt{1-x^2}}$

```
    fun gaussianMethodWithWithSpecialRo(n: Int): Double{
        val roots: MutableList<Double> = mutableListOf()
        for (k in 1..n){
            val xk = cos((2.0 * k - 1) * PI/(2.0 * n))
            roots.add(xk)
        }
        val mappedRootsPoints = LegendrePolynomial.
           mapInterval(roots,a,b).associateWith(function)
        val Ak = PI/n

        return mappedRootsPoints.values.sum() * Ak
    }
```

Ортогональные полиномы на отрезке $[-1,1]$ с весом $\rho(x) = \frac{1}{\sqrt{1-x^2}}$ являются полиномы Чебышева I рода:

$$T_n(x) = \cos(n \arccos x) \quad \text{с узлами} \quad x_k = \cos\frac{(2k-1)}{2n} \quad A_k = \frac{\pi}{n} \forall k$$

Тогда вычисляем приближение по формуле: $I \approx \frac{\pi}{n} \sum_{k=1}^{n} f(x_k)$

Квадратурная формула Гаусса с $\rho = \dfrac{1}{\sqrt{1-x^2}}$: $I_1 \approx 1.54701853581969 \quad I_2 \approx 1.925435808107$

Перечислим ещё раз результаты всех вычислений для сравнения

Левые Прямоугольники:	$I_1 \approx 0.3935142817035969$	$I_2 \approx 0.41690958455423466$
Правые Прямоугольники:	$I_1 \approx 0.39414496939930116$	$I_2 \approx 0.49125191356242043$
Средние Прямоугольники:	$I_1 \approx 0.39382072329248163$	$I_2 \approx 0.45088605222312456$
Метод Симпсона:	$I_1 \approx 0.3938237302855468$	$I_2 \approx 0.45198477254620356$
Формула Гаусса с $\rho = 1$:	$I_1 \approx 0.3938236880501$	$I_2 \approx 0.451948735871309$
Интерполяционная формула $\rho = 1$:	$I_1 \approx 0.39382368806290$	$I_2 \approx 0.45194867933117$
Формула Гаусса с $\rho = \frac{1}{\sqrt{1-x^2}}$:	$I_1 \approx 1.54701853581969$	$I_2 \approx 1.925435808107$

7 Погрешности

Я очень долго пытался запрограмировать вычисление погрешностей, но там нужно искать 3 производные и это оказалось слишком сложной задачей, а на вычисления в ручную у меня нет сил. Надеюсь на понимание!