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
(научно-исследовательская работа)

Обучающийся: Греков Лев Евгеньевич 05-104

Преподаватель:
доцент, к.н. Насибуллин Рамиль Гайсаевич

Дата сдачи контрольной: ______________

Казань – 2023

1 Задание №1. Подпункт 1

Дано: Значения x для первого набора узлов:

$$x_k = a + \frac{k(b-a)}{4} \quad k = \overline{0,4}, \quad a = -1, \quad b = 1 \quad \text{и функция:} \quad f(x) = \frac{e^{x/2} + 2}{2x + 1},$$

Найдем x_i и y_i для построения полинома. Вычисления производились програмно програмно.

$$\begin{aligned}
x_0 &= -1.0, & y_0 &= 1.4002311856967333 \\
x_1 &= -0.5, & y_1 &= 1.778800783071405 \\
x_2 &= 0.0, & y_2 &= 2.2599210498948734 \\
x_3 &= 0.5, & y_3 &= 2.8714264686559408 \\
x_4 &= 1.0, & y_4 &= 3.648721270700128
\end{aligned}$$

Для нахождения Интерполяционного полинома L_5(f, x) для первого набора узлов будем использовать формулу:

$$L_5(f, x) = \sum_{k=0}^{5} f(x_k) \cdot l_k(x) \quad \text{где, } l_k(x) = \prod_{j=1, j \neq k}^{n} \frac{x - x_j}{x_k - x_j}$$

Реализуем формулу для нажождения Фундаментальных Полиномов:

```
private fun createFundamentalPoly(xk: Double): Polynomial {
    val acc = Polynomial(1.0)
    for (it in _points.keys) {
        if (xk != it) {
            acc *= Polynomial(-it, 1.0) / (xk - it)
        }
    }
    return acc
}
```

Найдём Фундаментальные Полиномы $l_i(x)$ и умножим на y_i.

$$l_0 \cdot y_0 = 0.9334875x^4 - 0.9334875x^3 - 0.2333719x^2 + 0.2333719x$$
$$l_1 \cdot y_1 = -4.7434688x^4 + 2.3717344x^3 + 4.7434688x^2 - 2.3717344x$$
$$l_2 \cdot y_2 = 9.0396842x^4 - 11.2996052x^2 + 2.259921$$
$$l_3 \cdot y_3 = -7.6571372x^4 - 3.8285686x^3 + 7.6571372x^2 + 3.8285686x$$
$$l_4 \cdot y_4 = 2.4324808x^4 + 2.4324808x^3 - 0.6081202x^2 - 0.6081202x$$

Суммируем полученные значения

```
private fun createLagrangePoly(): Polynomial {
    val result = Polynomial(mapOf(0 to 0.0))
    for ((x, fx) in _points.entries) {
        result += createFundamentalPoly(x) * fx
    }
    return result
}
```

Получим:

$$L_5(f, x) = 0.0050465x^4 + 0.0421591x^3 + 0.2595087x^2 + 1.0820859x + 2.259921$$

Подставим контрольные точки в Полином

$$x^{(1)} = -\frac{5}{8} \quad L(f, x^{(1)}) = 1.675465215449604$$
$$x^{(2)} = \frac{5}{7} \quad L(f, x^{(2)}) = 3.181919701299438$$
$$x^{(3)} = \frac{2}{7} \quad L(f, x^{(3)}) = 2.5912897647763558$$

И найдём погрешности:

$$r_1 = f(x_1) - L(f, x_1) = 1.6754899416283353 - 1.675465215449604 = 2.472617873139349 \times 10^{-5}$$
$$r_2 = f(x_2) - L(f, x_2) = 3.1818728878908455 - 3.181919701299438 = -4.681340859225003 \times 10^{-5}$$
$$r_3 = f(x_3) - L(f, x_3) = 2.5913116923401254 - 2.5912897647763558 = 2.192756376961924 \times 10^{-5}$$

Как можно заметить, погрешности крайне малы

Дадим общую оценку погрешности:

$$r(x) = |f(x) - L_5(f,x)|$$

$$\max_{x\in(-1,1)} r(x) = \frac{\max_{x\in(-1,1)} f^{(5)}(\xi)}{5!} * \max_{x\in(-1,1)} \omega_5(x)$$

где

$$\omega_5(x) = \sum_{j=1}^{5}(x - x_j)$$

$$f^{(5)}(x) = \frac{32}{243} \cdot 2^{\frac{2x+1}{3}} \cdot (\ln(2))^5 + \frac{1}{32} \cdot e^{\frac{x}{2}} \quad \text{- Возрастающая функция,}$$

тогда

$$\max_{x\in(-1,1)} f^{(5)}(\xi) = f^{(5)}(1) \approx 0.0936631$$

Найдём максимум $\omega_5(x)$:

$$\omega_5'(x) = 5.0x^4 - 3.75x^2 + 0.25 = 0$$

в точке x = -8.2221643 принимает максимальное значение

$$\max_{x\in(-1,1)} \omega_5(\xi) \approx 0.1134823$$

Подставим в исходную формулу:

$$\max_{x\in(-1,1)} r(x) = \frac{0.0936631}{120} * 0.1134823 \approx 8.857586678 \times 10^{-5}$$

2 Задание №1. Подпункт 2

Проделаем то же самое для других исходных значений

$$x_k = \frac{b-a}{2}\cos\frac{(2k-1)\pi}{10} + \frac{b+a}{2} \quad k = \overline{1,5}, \quad a = -1, \quad b = 1 \quad f(x) = \frac{e^{x/2}+2}{2x+1},$$

Найдем x, y для построения полинома:

$$\begin{aligned}
&x_1 = 0.9510565162951535, && y_1 = 3.564138165038646\\
&x_2 = 0.5877852522924731, && y_2 = 2.994758365912109\\
&x_3 = 6.123233995736766 \times 10^{-17} = 0, && y_3 = 2.2599210498948734\\
&x_4 = -0.587785252292473, && y_4 = 1.7056028812167219\\
&x_5 = -0.9510565162951535, && y_5 = 1.4334125673651974
\end{aligned}$$

Интерполяционный полином L_5(f, x) для второго набора узлов:

$$L_5(f,x) = \sum_{k=1}^{5} f(x_k)\cdot l_k(x)$$

$$\begin{aligned}
l_1 \cdot y_1 &= 3.5244136x^4 + 3.3519166x^3 - 1.217655x^2 - 1.1580587x\\
l_2 \cdot y_2 &= -7.7529933x^4 - 4.5570951x^3 + 7.0126483x^2 + 4.1219313x\\
l_3 \cdot y_3 &= 7.2317474x^4 + 0.0x^3 - 9.0396842x^2 + 2.259921\\
l_4 \cdot y_4 &= -4.4155575x^4 + 2.5953996x^3 + 3.9939093x^2 - 2.347561x\\
l_5 \cdot y_5 &= 1.4174363x^4 - 1.348062x^3 - 0.4897122x^2 + 0.465744x
\end{aligned}$$

$$L_5^{(II)}(f,x) = 0.0050465x^4 + 0.042159x^3 + 0.2595062x^2 + 1.0820556x + 2.259921$$

$$L_5^{(I)}(f,x) = 0.0050465x^4 + 0.0421591x^3 + 0.2595087x^2 + 1.0820859x + 2.259921$$

Заметим, что Полиномы для первого и второго узлов совпадают, с небольшой погрешностью

Подставим контрольные точки в Полином

$$x^{(1)} = -\frac{5}{8} \quad L(f, x^{(1)}) = 1.6754832493654448$$
$$x^{(2)} = \frac{5}{7} \quad L(f, x^{(2)}) = 3.1818967439622385$$
$$x^{(3)} = \frac{2}{7} \quad L(f, x^{(3)}) = 2.5912809014975813$$

И найдём погрешности:

$$r_1 = f(x_1) - L(f, x_1) = 1.6754899416283353 - 1.6754832493654448 = 6.6922628905174975 \times 10^{-6}$$
$$r_2 = f(x_2) - L(f, x_2) = 3.1818728878908455 - 3.1818967439622385 = -2.3856071392991396 \times 10^{-5}$$
$$r_3 = f(x_3) - L(f, x_3) = 2.5913116923401254 - 2.5912809014975813 = 3.079084254409281 \times 10^{-5}$$

Дадим общую оценку погрешности:

$$r(x) = |f(x) - L_5(f, x)|$$

$$\max_{x \in (-0.951, 0.951)} f^{(5)}(\xi) = f^{(5)}(0.951) \approx 0.09147274$$

Найдём максимум $\omega_5(x)$:

$$\omega_5^{'}(x) = 5.0x^4 - 3.75x^2 + 0.3125 = 0$$

в точке x = -0.809017 принимает максимальное значение

$$\max_{x \in (-0.951, 0.951)} \omega_5(\xi) \approx 0.0625$$

Подставим в исходную формулу:

$$\max_{x \in (-0.951, 0.951)} r(x) = \frac{0.09147274}{120} * 0.0625 \approx 4.7642052083 \times 10^{-5}$$

Таким образом, мы получили 2 практически идентичных полинома, однако у второго погрешность меньше примерно в 2 раза, так как этот полином строится по узлам Чебышева.

3 Задание №2

Дано:

$$x_k = a + \frac{k(b-a)}{4} \quad k = \overline{0,4}, \quad a = -1, \quad b = 1 \quad \text{и функция:} \quad f(x) = \frac{e^{x/2}+2}{2x+1},$$

Построим Интерполяционный полином Ньютона:

$$P_n(x) = f(x_0)+(x-x_0)f(x_0;x_1)+(x-x_0)(x-x_1)f(x_0;x_1;x_2)+\ldots+(x-x_0)\ldots(x-x_{n-1})f(x_0;\ldots;x_n),$$

Разделенные Разности вычисляются по формуле:

$$f(x_0;x_1;\ldots;x_n) = \sum_{j=0}^{n} \frac{f(x_j)}{\prod_{\substack{i=0\\ i\neq j}}^{n}(x_j - x_i)}$$

Реализуем её програмно:

```
private fun dividedDifference(k: Int): Double = (0..k).sumOf { j ->
    val multiplication = (0..k)
        .filter { i -> i != j }
        .fold(1.0) { acc, i -> acc * (_points[j].first - _points[i].first)}
    _points[j].second / multiplication
}
```

Т.к (x-x_0)...(x-x_n) содержат в себе произведение предыдущих полиномов, вычислять новый полином удобно умножая текущий на (x-x_i)

```
private val lastFundPoly: Polynomial = Polynomial(1.0);

private fun fundPoly(j: Int): Polynomial =
    if (j == 1) lastFundPoly else lastFundPoly *
    Polynomial(-_points[j - 2].first, 1.0)
```

Полином Ньютона позволяет добавлять новые узлы к существующему полиному. Мы воспользуемся этой возможностью и будем динамически обновлять существующую структуру

```
fun addPoint(x: Double, f: Double) {
        _points.add(Pair(x, f))
        this += fundPoly(n) * dividedDifference(n - 1)
    }
```

Точки у нас такие-же как в первом задании, вычислим разделенные разности для этих точек

$$
\begin{aligned}
f(x_1) &= 1.4002311856967333 \\
f(x_1, x_2) &= 0.7571391947493433 \\
f(x_1, x_2, x_3) &= 0.20510133889759352 \\
f(x_1, x_2, x_3, x_4) &= 0.03711264331840303 \\
f(x_1, x_2, x_3, x_4, x_5) &= 0.0050464992378125295
\end{aligned}
$$

И базисные Полиномы:

$$
\begin{aligned}
(x - x_1) &= x + 1.0 \\
(x - x_1)(x - x_2) &= x^2 + 1.5x + 0.5 \\
(x - x_1)(x - x_2)(x - x_3) &= x^3 + 1.5x^2 + 0.5x \\
(x - x_1)(x - x_2)(x - x_3)(x - x_4) &= x^4 + x^3 - 0.25x^2 - 0.25x
\end{aligned}
$$

Умножим разделенные разносит на соответствующие полиномы, сложим произведения и получим:

$$L_5(f, x) = 0.0050464x^4 + 0.0421591x^3 + 0.2595087x^2 + 1.082085x + 2.259921$$

4 Задание №3

Чтобы выполнить данное задание, нужно найти полином $H_N(f,x)$ N-й степени, где $N = \sum_{i=1}^{5} m_i - 1$, а m_i - порядок i-го узла. В нашем случае $m_i = 2$, $i = 1,5$. Тогда $N = \sum_{i=1}^{5} 2 - 1 = 10 - 1 = 9$. Получается, что $H_N(f,x)$ - полином 9-й степени. Для построения полинома используем следующую формулу:

$$H_N(f,x) = L_n(f,x) + \omega_n(x) H_{N-n}(x),$$

$L_5(f,x)$ и ω_5 известны из предыдущих заданий.
Нужно найти: $H_4(x) = A + Bx + Cx^2 + Dx^3 + Ex^4$

Воспользуемся следующей формулой:

$$H_9^{'}(f,x_i) = L_5^{'}(f,x_i) + (\omega_5(x_i) H_4(x_i))^{'}, \quad i = \overline{1,5}$$
$$H_9^{'}(f,x_i) = L_5^{'}(f,x_i) + \omega_5^{'}(x_i) H_4(x) + \omega_5(x_i) H_4^{'}(x_i), \quad i = \overline{1,5}$$

$\omega_5(x_i) H_4^{'}(x_i)$ занулится, так-как $\omega_5(x_i) = 0$ Тогда получим систему

$$\begin{cases} H_4(x) &= \dfrac{f'(x_1) - L_5'(f,x_1)}{\omega_5'(x_1)} \\ H_4(x) &= \dfrac{f'(x_2) - L_5'(f,x_2)}{\omega_5'(x_2)} \\ H_4(x) &= \dfrac{f'(x_3) - L_5'(f,x_3)}{\omega_5'(x_3)} \\ H_4(x) &= \dfrac{f'(x_4) - L_5'(f,x_4)}{\omega_5'(x_4)} \\ H_4(x) &= \dfrac{f'(x_5) - L_5'(f,x_5)}{\omega_5'(x_5)} \end{cases}$$

Для поиска производной функции в точки используем численную апроксимацию, используя формулу конечной разности

```
fun calculateDerivativeAtPoint(
    f: (Double) -> Double,
    x: Double,
    h: Double = 1e-5): Double
        = (f(x + h) - f(x - h)) / (2 * h)
```

Также реализуем метод для взятия производной от полинома чтобы вычислить $L_5'(f, x_4)$ и $\omega_5'(x_5)$

```
fun derivative(derivOrder: Int): Polynomial {
        val derivativeCoeffs = mutableMapOf<Int, Double>()
        for ((exp, coeff) in _coeffs) {
            if (exp >= derivOrder) {
                val newExp = exp - derivOrder
                val newCoeff = coeff * factorial(exp) /
                   factorial(newExp)
                derivativeCoeffs[newExp] = newCoeff
            }
        }
        return Polynomial(derivativeCoeffs)
    }
```

Для решения Системы Линейных уравнений я програмно реализовал метод Крамера. Он занимает достаточно много места, поэтому оставлю его в Приложении.

Ниже код выполняющий все вышеописанные шаги

```
    fun thirdExercise(){
        val points =
            getPoints({ k -> a + k * (b-a)/4.0 },0,4)
        val lagrange = LagrangePolynomial(points,fileName)
        val w = lagrange.findW()

        val h4List = mutableListOf<Double>()
        points.keys.forEach{
            val funcD =
                calculateDerivativeAtPoint(ownFunction,it)
            val l5d = lagrange.derivative(1)(it)
            val wd = w.derivative(1)(it)
            h4List.add( (funcD - l5d)/wd )
         }


```

```
        val system = generateMatrix(points)
        val solution = SystemSolver.solveLinearSystem(
           system,h4List.toDoubleArray())
        val H4 = Polynomial(*solution)
        val H9 = lagrange + w * H4
    }
```

В ходе вычислений получим (Вывожу без округления)

$$H_4(x) = 2.1953142513729695 \times 10^{-8}x^4 + 1.638186899714056 \times 10^{-7}x^3 + 2.6712106835113036 \times 10^{-6}x^2 + 3.8941280090940335 \times 10^{-5}x + 4.8499778651933667 \times 10^{-4}$$

$$H_9(f, x) = 2.1953142513729695 \times 10^{-8}x^9 + 1.638186899714056 \times 10^{-7}x^8 + 2.6437692553691416 \times 10^{-6}x^7 + 3.873650672847608 \times 10^{-5}x^6 + 4.8166426145057595 \times 10^{-4}x^5 + 0.004997863592371791x^4 + 0.04155356312573682x^3 + 0.2595184143857672x^2 + 1.0822071493921115x + 2.2599210498948734$$

Я, если честно, не смог построить график полинома в графических приложениях(чтобы сравнить его график с графиком функции) в виду его громостткости. Поэтому дополнительно высчитал погрешности в контрольных точках в целях самопроверки.

$$r_1 = f(x^{(1)}) - H_9(f, x^{(1)}) = 1.6754899416283 - 1.6754899415337985 = 9.453682281446 \times 10^{-11}$$
$$r_2 = f(x^{(2)}) - H_9(f, x^{(2)}) = 3.1818728878908 - 3.181872888118346 = -2.2750024086803 \times 10^{-10}$$
$$r_3 = f(x^{(3)}) - H_9(f, x^{(3)}) = 2.5913116923401 - 2.591311692355613 = -1.5487611193520 \times 10^{-11}$$

И так как Полином Эрмита интерполирует прозводную функцию, проверим контрольные точки для производной

$$r_1^d = f'(x^{(1)}) - H_9'(f, x^{(1)}) = 0.8019703602313 - 0.8019703613396714 = -1.1083415296653 \times 10^{-9}$$
$$r_2^d = f'(x^{(2)}) - H_9'(f, x^{(2)}) = 1.5245083644144 - 1.5245083654018565 = -9.874336903692 \times 10^{-10}$$
$$r_3^d = f'(x^{(3)}) - H_9'(f, x^{(3)}) = 1.2411625438968 - 1.2411625435652034 = 3.316309449274 \times 10^{-10}$$

Таким образом, можно наблюдать, у полинома Эрмита-Фейера точность производных и самого полинома на порядки выше, чем у полинома Лагранжа.

5 Приложение

Метод Крамера

```
object SystemSolver {
    fun solveLinearSystem(coefficients: Array<DoubleArray>,
        constants: DoubleArray): DoubleArray {
        val n = constants.size
        val solution = DoubleArray(n)

        val determinantA = calculateDeterminant(
           coefficients)

        for (i in 0..<n) {
            val modifiedCoefficients = Array(n) {
               DoubleArray(n) }
            for (j in 0..<n) {
                for (k in 0..<n) {
                    modifiedCoefficients[j][k] =
                       coefficients[j][k]
                    if (k == i) {
                        modifiedCoefficients[j][k] =
                           constants[j]
                    }
                }
            }

            val determinantModified = calculateDeterminant(
               modifiedCoefficients)

            solution[i] = determinantModified /
               determinantA
        }

        return solution
    }
```

```
    private fun calculateDeterminant(matrix: Array<DoubleArray>): Double {
        val n = matrix.size

        if (n == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
        }

        var determinant = 0.0

        for (i in 0..<n) {
            val subMatrix = Array(n - 1) { DoubleArray(n - 1) }
            for (j in 1..<n) {
                for (k in 0..<n) {
                    if (k < i) {
                        subMatrix[j - 1][k] = matrix[j][k]
                    } else if (k > i) {
                        subMatrix[j - 1][k - 1] = matrix[j][k]
                    }
                }
            }
            determinant += matrix[0][i] * calculateDeterminant(subMatrix) * if (i % 2 == 0) 1 else -1
        }
        return determinant
    }
}
```

Код класса Polynomial

```

open class Polynomial(coeffs: Map<Int, Double>) {
    protected val _coeffs: MutableMap<Int, Double> =
        mutableMapOf()

    init {
        setFiltered(coeffs)
    }

    protected fun setFiltered(rawCoeffs: Map<Int, Double>){
        _coeffs.clear()
        _coeffs.putAll(rawCoeffs.filter { (k,v) -> v neq
            0.0 && k >= 0 }.toMutableMap())
        if(_coeffs.isEmpty()){
            _coeffs[0] = 0.0
        }
    }

    //
    val coeffs: Map<Int,Double>
        get() = _coeffs.toMap()

    val size : Int
        get() = _coeffs.size
    val highDegree : Int
        get() =  _coeffs.keys.max()?: 0
    val minorDegree : Int
        get() = _coeffs.keys.min()?: 0


    constructor(vararg coeffs: Double) : this (coeffs.
        mapIndexed { index, value -> index to value }.toMap()
        )
```

```
    constructor(coeffs: MutableList<Double>) : this (coeffs.mapIndexed { index, value -> index to value }.toMap())
    constructor(other: Polynomial) : this(HashMap(other._coeffs))

    //
    operator fun times(scalar: Double) = Polynomial(_coeffs.map { (k, v) -> k to scalar * v }.toMap())
    operator fun timesAssign(scalar: Double){
        _coeffs.keys.forEach { _coeffs[it] = _coeffs[it]!! * scalar}
        setFiltered(coeffs)
    }
    operator fun div(scalar: Double) =
        Polynomial(_coeffs.map { (k, v) -> if (scalar eq 0.0) throw ArithmeticException("Division by zero") else k to 1.0 / scalar * v }
            .toMap())

    //
    operator fun plus(other: Polynomial) = Polynomial(_coeffs.toMutableMap().also {
        other._coeffs.forEach { (k2, v2) -> it[k2] = v2 + (it[k2] ?: 0.0) }
    })
    operator fun plusAssign(other: Polynomial) {
        other._coeffs.forEach { (k2, v2) -> _coeffs[k2] = v2 + (_coeffs[k2] ?: 0.0) }
        setFiltered(coeffs)
    }
    operator fun minus(other: Polynomial) = Polynomial(_coeffs.toMutableMap().also {
        other._coeffs.forEach { (k2, v2) -> it[k2] = -v2 +
```

```
            (it[k2] ?: 0.0) }
    })
    operator fun times(other: Polynomial) = Polynomial(mutableMapOf<Int, Double>().also {
        _coeffs.forEach { (k1, v1) ->
            other._coeffs.forEach { (k2, v2) ->
                it[k1 + k2] = v1 * v2 + (it[k1 + k2] ?: 0.0)
            }
        }
    })
    operator fun timesAssign(other: Polynomial){
        val c = mutableMapOf<Int,Double>()
        _coeffs.forEach { (k1, v1) ->
            other._coeffs.forEach { (k2, v2) ->
                c[k1 + k2] = v1 * v2 + (c[k1 + k2] ?: 0.0)
            }
        }
        _coeffs.apply {
            clear()
            setFiltered(c)
            putAll(c)
        }
    }
    private fun divide(divisor: Polynomial): Pair<Polynomial, Polynomial> {

        if(divisor.coeffs[0] == 0.0) throw ArithmeticException("forbidden to divide by zero");

        val divisorList = (0..divisor.highDegree).map { divisor._coeffs.getOrDefault(it,0.0)}.toMutableList()
```

```
        val remainder = (0..this.highDegree).map {_coeffs.
           getOrDefault(it,0.0)}.toMutableList()

        val quotient = MutableList(remainder.size - divisor
           .size + 1){0.0}

        for(i in quotient.indices){
            val coeff : Double = remainder[remainder.size -
                i - 1] / divisorList.last();
            quotient[quotient.size - i - 1] = coeff;

            for(j in divisorList.indices){
                remainder[remainder.size - i - j - 1] -=
                   coeff * divisorList[divisorList.size - j
                   - 1]
            }
        }

        return Pair(Polynomial(quotient), Polynomial(
           remainder))
    }
    operator fun rem(other: Polynomial) = divide(other).
       second;
    operator fun div(other: Polynomial) = divide(other).
       first;

    //
    operator fun get(degree: Int) = _coeffs[degree] ?: 0.0
    operator fun invoke(scalar: Double) =
        _coeffs.entries.sumOf { (k, v) -> v * scalar.pow(k)
            }
    fun derivative(derivOrder: Int): Polynomial {
        val derivativeCoeffs = mutableMapOf<Int, Double>()
        for ((exp, coeff) in _coeffs) {
```

```
            if (exp >= derivOrder) {
                val newExp = exp - derivOrder
                val newCoeff = coeff * factorial(exp) / factorial(newExp)
                derivativeCoeffs[newExp] = newCoeff
            }
        }
        return Polynomial(derivativeCoeffs)
    }

    //                              Any

    override operator fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other !is Polynomial) return false
        return this._coeffs == other._coeffs
    }
    override fun hashCode(): Int = _coeffs.keys.hashCode() * 17 + _coeffs.values.hashCode() * 31

    override fun toString() = toString("x")

    fun toString(variable: String) = _coeffs.toSortedMap(reverseOrder()).map{ (k, v) ->
        buildString {
            if (v.neq(0.0, 1e-12)) {
                append(if (v > 0.0 || v.eq(0.0, 1e-12)) if (k != _coeffs.keys.max()) "+" else "" else "-")
                if (abs(v) neq 1.0 || k == 0) append(abs(v))
                if (k != 0) append(variable)
                if (k > 1) append("^$k")
            }
```

```
        }
    }.joinToString("")

}
```

Полином Лагранжа

```
    class LagrangePolynomial(points: Map<Double,Double>) : Polynomial() {
    private val _points: MutableMap<Double, Double>
    init {
        _points = points.toMutableMap()
        if(_points.isEmpty()) _coeffs[0] = 0.0
        else _coeffs.apply {
            clear()
            putAll(createLagrangePoly().coeffs)
        }
    }
    val points: Map<Double,Double>
        get()= _points.toMap()

    private fun createLagrangePoly(): Polynomial {
        val result = Polynomial(mapOf(0 to 0.0))
        val i = 0;
        for ((x, fx) in _points.entries) {
            val fundamentalPoly = createFundamentalPoly(x)
            println("             $i           .                : $fundamentalPoly")
            result += fundamentalPoly * fx
        }
        return result
    }

    private fun createFundamentalPoly(xk: Double): Polynomial {
```

```
        val acc = Polynomial(1.0)
        for (it in _points.keys) {
            if (xk != it) {
                acc *= Polynomial(-it, 1.0) / (xk - it)

            }
        }
        return acc
    }

        fun findW() : Polynomial {
        val result = Polynomial(1.0)
        points.keys.forEach {
            result *= Polynomial(-it, 1.0)
        }
        return result
    }

}
```

Полином Ньютона

```
class NewtonPolynomial2(points: Map<Double,Double>) :
   Polynomial() {

    private val n: Int
        get() = _points.size

    private val _points: MutableList<Pair<Double,Double>> =
         mutableListOf()

    private val lastFundPoly: Polynomial = Polynomial(1.0);
    init {
        _coeffs[0] = 0.0
        this.addPoints(points.toList().toMutableList())
    }
```

```
    private fun fundPoly(j: Int): Polynomial {
        if (j != 1) {
            lastFundPoly *= Polynomial(-_points[j-2].first, 1.0)
        }
        return lastFundPoly
    }

    private fun dividedDifference(k: Int): Double = (0..k).sumOf { j ->
        val multiplication = (0..k)
            .filter { i -> i != j }
            .fold(1.0) { acc, i -> acc * (_points[j].first - _points[i].first)}
        _points[j].second / multiplication
    }
    fun addPoint(x: Double, f: Double) {
        _points.add(Pair(x, f))
        this += fundPoly(n) * dividedDifference(n - 1)
    }
    fun addPoints( pointsList: List<Pair<Double,Double>>) =
        pointsList.forEach{this.addPoint(it.first,it.second)}
}
```