Национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной Инженерии и Компьютерной Техники

Вариант №12

Лабораторная работа №6

«Численное дифференцирование»

По дисциплине:

«Вычислительная математика»

Работу выполнила:

Студентка группы P3212

Никонова Наталья Игоревна

Преподаватель:

Малышева Татьяна Алексеевна

Санкт-Петербург

2022

Цель лабораторной работы

Решить задачу Коши численными методами Рунге-Кутта 4-ого порядка и Адамса

Рабочие формулы

Метод Рунге-Кутта 4-ого порядка

$$y_i = y_{i-1} + \frac{1}{6}(k_1 + 2 * k_2 + 2 * k_3 + k_4)$$

$$k_1 = h * f(x_{i-1}, y_{i-1})$$

$$k_2 = h * f(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{k_1}{2})$$

$$k_3 = h * f(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{k_2}{2})$$

$$k_4 = h * f(x_{i-1} + h, y_{i-1} + k_3)$$

Метод Адамса

Предиктор: $y_i = y_{i-1} + \frac{h}{24}(55f_{i-1} - 59f_{i-2} + 37f_{i-3} - 9f_{i-4})$

Корректор: $y_i = y_{i-1} + \frac{h}{24}(9f_i + 19f_{i-1} - 5f_{i-2} + f_{i-3})$

Листинг программы

Исходный кодъ: https://github.com/nanikon/computational_math/tree/lab6/differential

Метод Рунге-Кутта:

```kotlin
object RungeKuttaMethod {
    fun compute(diff: DiffEquation, xs: List<BigDecimal>, y0: BigDecimal, eps: BigDecimal): List<BigDecimal> {
        val ys = mutableListOf(y0)
        var n = 2
        for (i in 1 until xs.size) {
            n /= 2
            var newY = calculateInterval(diff.function, xs[i - 1], xs[i], ys[i - 1], n)
            var oldY: BigDecimal
            do {
                oldY = newY
                n *= 2
                newY = calculateInterval(diff.function, xs[i - 1], xs[i], ys[i - 1], n)
            } while ((oldY - newY).divideN(15).abs() >= eps) // 2 ^ 4 - 1 = 16 - 1 = 15
            ys.add(newY)
        }
        return ys
    }
```

```
    private fun calculateInterval(
        function: (x: BigDecimal, y: BigDecimal) -> BigDecimal,
        a: BigDecimal,
        b: BigDecimal,
        y0: BigDecimal,
        n: Int
    ): BigDecimal {
        val h = (b - a).divideN(n)
        val xs = mutableListOf(a)
        var elem = a + h
        while (elem <= b) {
            xs.add(elem)
            elem += h
        }

        val ys = mutableListOf(y0)
        for (i in 1 ≤ until < xs.size) {
            ys.add(calculateY(function, xs[i - 1], ys[i - 1], h))
        }
        return ys.last()
    }
```

```
    private fun calculateY(
        function: (x: BigDecimal, y: BigDecimal) -> BigDecimal,
        x: BigDecimal,
        y: BigDecimal,
        h: BigDecimal
    ): BigDecimal {
        val hDiv2 = h.divideN( number: 2)
        val k1 = h * function(x, y)
        val k2 = h * function( x: x + hDiv2,  y: y + k1.divideN( number: 2))
        val k3 = h * function ( x: x + hDiv2,  y: y + k2.divideN( number: 2))
        val k4 = h * function( x: x + h,  y: y + k3)
        return y + (k1 + BigDecimal( val: 2) * k2 + BigDecimal( val: 2) * k3 + k4).divideN( number: 6)
    }
}
```

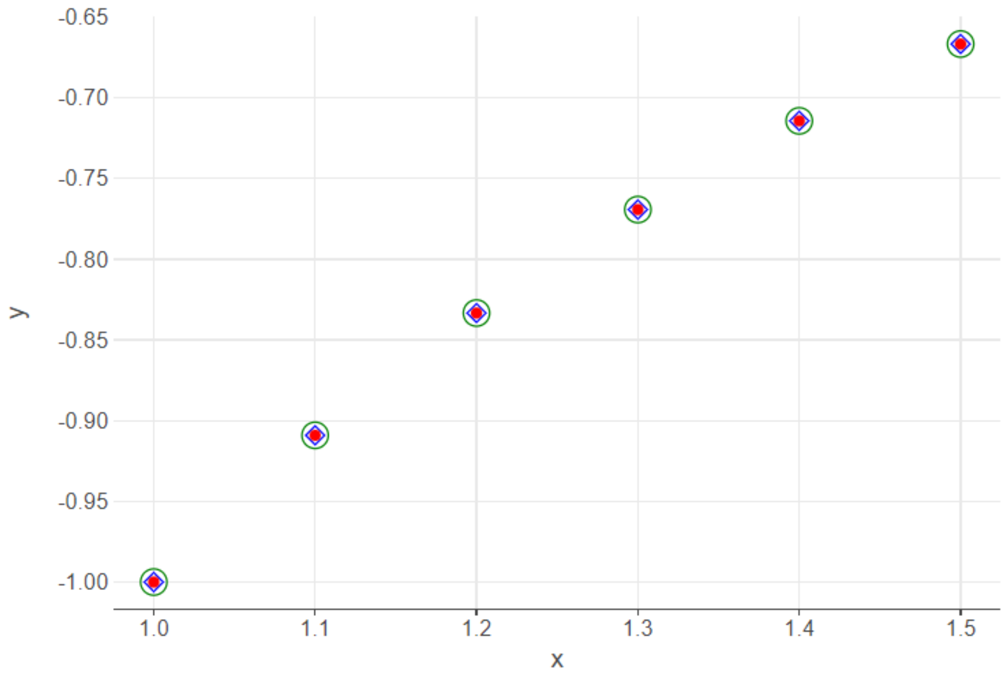
Метод Адамса

```
object AdamsMethod {
    nanikon *
    fun compute(
        diff: DiffEquation,
        xs: List<BigDecimal>, y4: List<BigDecimal>,
        h: BigDecimal,
        eps: BigDecimal
    ): List<BigDecimal> {
        val ys = y4.toMutableList()
        for (i in 4 ≤ until < xs.size) {
            var newY = ys[i - 1] +
                    h.divideN( number: 24) * (
                    BigDecimal( val: 55) * diff.function(xs[i - 1], ys[i - 1]) -
                            BigDecimal( val: 59) * diff.function(xs[i - 2], ys[i - 2]) +
                            BigDecimal( val: 37) * diff.function(xs[i - 3], ys[i - 3]) -
                            BigDecimal( val: 9) * diff.function(xs[i - 4], ys[i - 4])
                    )
            var oldY = newY
            var count = 0
            do {
                oldY = newY
```

```
                newY = ys[i - 1] +
                        h.divideN( number: 24) * (
                        BigDecimal( val: 9) * diff.function(xs[i], oldY) +
                                BigDecimal( val: 19) * diff.function(xs[i - 1], ys[i - 1]) -
                                BigDecimal( val: 5) * diff.function(xs[i - 2], ys[i - 2]) +
                                BigDecimal( val: 1) * diff.function(xs[i - 3], ys[i - 3])
                        )
                count++
                if (count == 100) {
                    println("Корректирующие значения расходятся. Нужная точность не достигается")
                    break
                }
            } while ((oldY - newY).abs() >= eps)
            ys.add(newY)
        }
        return ys
    }
}
```

Результат выполнения программы

```
Выберите уравнение для решения и введите его номер:
1 - y' = y + (1 + x) * y^2
2 - y' = (x - y)^2 + 1
3 - y' = y * sin(x)
1
Введите левую границу интервала. Это должно быть дробное число
1
Введите правую границу интервала. Это должно быть дробное число больше, чем 1
1.5
Введите начальное значение y0 в точке x0=1. Это должно быть дробное число
-1
Введите длину интервала. Это должно быть дробное число  и помещаться в интервал целое число раз
0.1
Введите погрешность. Это должно быть дробное число между 0 и 1
0.00001
Аналитическое точное решение уравнения: y = -e^x / (x * e^x + 0.000000)
Сеточная функция
Аргумент:       1,000000 1,100000 1,200000 1,300000 1,400000 1,500000
Точное:         -1,000000 -0,909091 -0,833333 -0,769231 -0,714286 -0,666667
Рунге-Кутта:    -1,000000 -0,909091 -0,833334 -0,769231 -0,714286 -0,666667
Адамса:         -1,000000 -0,909091 -0,833334 -0,769231 -0,714279 -0,666658
```



Выводы

В ходе выполнения лабораторной работы я познакомилась с одношаговыми и многошаговыми численными методами решения дифференциальных уравнений. А также ощутила на себе всю их возможную неточность – при слишком большой длине шага метод Адамса может или сходится к значению, далекому от истинного, или не сходиться в принципе.