Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление системного и прикладного программного обеспечения

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

курса «Вычислительная математика»

по теме: «Численное интегрирование»

Вариант № 21

Выполнил студент:

Тюрин Иван Николаевич

группа: P32131

Преподаватель:

Малышева Т. А.,

Бострикова Д. К.

Санкт-Петербург, 2023 г.

Содержание

Лабораторная работа № 3
Численное интегрирование

1. Задание варианта № 21

, , ,

Вычислительная реализация задачи:

1. *Вычислить интеграл, приведенный в таблице 1, точно.*
2. *Вычислить интеграл по формуле Ньютона – Котеса при* $n = 5$.
3. *Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при* $n = 10$.
4. *Сравнить результаты с точным значением интеграла.*
5. *Определить относительную погрешность вычислений для каждого метода.*
6. *В отчете отразить последовательные вычисления.*

$$\int_0^2 (2x^3 - 5x^2 - 3x + 21)dx$$

, , ,

2. Цель работы

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

i	c_i	x_i	$f(x_i)$	$c_i \cdot f(x_i)$
0	0,1319	0	21,00	2,77
1	0,5208	0,4	19,13	9,96
2	0,3472	0,8	16,42	5,70
3	0,3472	1,2	13,66	4,74
4	0,5208	1,6	11,59	6,04
5	0,1319	2	11,00	1,45
				30,67

Таблица 1.1: Вычисление интеграла методом Ньютона-Котеса для $n = 5$

3. Вычислительная часть работы

Вычислим аналитически значение интеграла.

$$\int_0^2 (2x^3 - 5x^2 - 3x + 21)dx = \left(\frac{1}{2}x^4 - \frac{5}{3}x^3 - \frac{3}{2}x^2 + 21x\right)\Big|_0^2 = 8 - \frac{40}{3} - 6 + 42 = 30\frac{2}{3}.$$

Теперь вычислим численно значание интеграла, оформив вычисления в виде таблиц: 1.2, 1.3, 1.4, 1.1, 1.6. В методе Ньютона-Котеса использовалось разделение с $n = 5$, в остальных методах использовалось $n = 10$. Результаты интегрирования представлены в нижней сводной строчке каждой таблицы.

Как можно видеть, результаты интегрирования отличаются, но не очень сильно. Точнее всего вычисления оказались с помощью метода Симсона и Ньютона-Котеса для $n = 5$.

Относительные погрешности методов:

- правых прямоугольников — $0,033$,
- левых прямоугольников — $0,032$,
- центральных прямоугольников — $0,0002$,
- Ньютона-Котеса — $< 0,0001$,
- Трапеций — $< 0,0001$,
- Симпсона — $< 0,0001$.

4. Программная часть работы

Основную часть программной реализации на языке программировани Kotlin можно посмотреть в листинге 1.1. Весь код представлен в личном репозитории [1].

i	c_i	x_i	$f(x_i)$	$c_i \cdot f(x_i)$
0,5	0,2	0,1	20,65	4,13
1,5	0,2	0,3	19,70	3,94
2,5	0,2	0,5	18,50	3,70
3,5	0,2	0,7	17,14	3,43
4,5	0,2	0,9	15,71	3,14
5,5	0,2	1,1	14,31	2,86
6,5	0,2	1,3	13,04	2,61
7,5	0,2	1,5	12,00	2,40
8,5	0,2	1,7	11,28	2,26
9,5	0,2	1,9	10,97	2,19
				30,66

Таблица 1.2: Вычисление интеграла методом прямоугольников (центральных)

i	c_i	x_i	$f(x_i)$	$c_i \cdot f(x_i)$
0	0,2	0	21,00	4,20
1	0,2	0,2	20,22	4,04
2	0,2	0,4	19,13	3,83
3	0,2	0,6	17,83	3,57
4	0,2	0,8	16,42	3,28
5	0,2	1	15,00	3,00
6	0,2	1,2	13,66	2,73
7	0,2	1,4	12,49	2,50
8	0,2	1,6	11,59	2,32
9	0,2	1,8	11,06	2,21
				31,68

Таблица 1.3: Вычисление интеграла методом прямоугольников (левых)

i	c_i	x_i	$f(x_i)$	$c_i \cdot f(x_i)$
1	0,2	0,2	20,22	4,04
2	0,2	0,4	19,13	3,83
3	0,2	0,6	17,83	3,57
4	0,2	0,8	16,42	3,28
5	0,2	1	15,00	3,00
6	0,2	1,2	13,66	2,73
7	0,2	1,4	12,49	2,50
8	0,2	1,6	11,59	2,32
9	0,2	1,8	11,06	2,21
10	0,2	2	11,00	2,20
				29,68

Таблица 1.4: Вычисление интеграла методом прямоугольников (правых)

i	x_i	$f(x_i)$
0	0	21,00
1	0,2	20,22
2	0,4	19,13
3	0,6	17,83
4	0,8	16,42
5	1	15,00
6	1,2	13,66
7	1,4	12,49
8	1,6	11,59
9	1,8	11,06
10	2	11,00
		30,68

Таблица 1.5: Вычисление интеграла методом Трапеций

i	x_i	$f(x_i)$
0	0	21,00
1	0,2	20,22
2	0,4	19,13
3	0,6	17,83
4	0,8	16,42
5	1	15,00
6	1,2	13,66
7	1,4	12,49
8	1,6	11,59
9	1,8	11,06
10	2	11,00
		30,67

Таблица 1.6: Вычисление интеграла методом Симсона

```
package io.github.e1turin.entities.integrator.model

import io.github.e1turin.shared.config.functionWithLabelStore
import io.github.e1turin.shared.config.maxNIteration
import io.github.e1turin.shared.lib.length
import io.github.e1turin.shared.model.settings.IntegrationParameters
import io.github.e1turin.shared.model.solution.IntegrationResult
import kotlin.math.abs
import kotlin.math.pow

class CenterRectangleIntegrator(private val settings:
    IntegrationParameters) : Integrator {

     init {
         check(settings.range.length > 0) { "Inspected range mustn't be
    empty. Check its bounds." }
         check(functionWithLabelStore.keys.contains(settings.functionLabel)
    ) { "Function must be in defined set." }
     }

     override fun integrate(): IntegrationResult {
         val function: (Double) -> Double = functionWithLabelStore[settings
    .functionLabel]!!

         var currentIntegrationSum = 0.0
         var currentDivisions = settings.divisions
         var deviance: Double = Double.MAX_VALUE

         for(iteration in 0..maxNIteration) {
             val integralSum = nextIntegralApproximation(
                 start = settings.range.start,
                 endInclusive = settings.range.endInclusive,
                 divisions = currentDivisions,
                 function = function
             )

             val preciseIntegralSum = nextIntegralApproximation(
```

```
                start = settings.range.start,
                endInclusive = settings.range.endInclusive,
                divisions = currentDivisions * 2,
                function = function
            )

            deviance = abs(integralSum - preciseIntegralSum)  / (2.0.pow
    (2) - 1)
            currentIntegrationSum = preciseIntegralSum
            currentDivisions *= 2

            if (deviance < settings.precision ) break
        }


        return IntegrationResult(
            area = currentIntegrationSum,
            precision = deviance,
            divisions = currentDivisions,
            convergence = false
        )
    }

    private fun nextIntegralApproximation(
        start: Double,
        endInclusive: Double,
        divisions: Int,
        function: (Double) -> Double
    ): Double {
        val step = (endInclusive - start) / divisions

        var integralSum = 0.0

        var currentParam = start + step / 2

        repeat(divisions) {
            integralSum += function(currentParam) * step
            currentParam += step
        }

        return integralSum
    }
}

package io.github.e1turin.entities.integrator.model

import io.github.e1turin.shared.config.functionWithLabelStore
import io.github.e1turin.shared.config.maxNIteration
import io.github.e1turin.shared.lib.length
import io.github.e1turin.shared.model.settings.IntegrationParameters
import io.github.e1turin.shared.model.solution.IntegrationResult
import kotlin.math.abs
import kotlin.math.pow

class LeftRectangleIntegrator(private val settings: IntegrationParameters)
    : Integrator {

    init {
        check(settings.range.length > 0) { "Inspected range mustn't be
    empty. Check its bounds." }
```

```
91          check(functionWithLabelStore.keys.contains(settings.functionLabel)
    ) { "Function must be in defined set." }
92      }
93
94      override fun integrate(): IntegrationResult {
95          val function: (Double) -> Double = functionWithLabelStore[settings
    .functionLabel]!!
96
97          var currentIntegrationSum = 0.0
98          var currentDivisions = settings.divisions
99          var deviance: Double = Double.MAX_VALUE
100
101         for(iteration in 0..maxNIteration) {
102             val integralSum = nextIntegralApproximation(
103                 start = settings.range.start,
104                 endInclusive = settings.range.endInclusive,
105                 divisions = currentDivisions,
106                 function = function
107             )
108
109             val preciseIntegralSum = nextIntegralApproximation(
110                 start = settings.range.start,
111                 endInclusive = settings.range.endInclusive,
112                 divisions = currentDivisions * 2,
113                 function = function
114             )
115
116             deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(2)
    - 1)
117             currentIntegrationSum = preciseIntegralSum
118             currentDivisions *= 2
119
120             if (deviance < settings.precision ) break
121         }
122
123
124         return IntegrationResult(
125             area = currentIntegrationSum,
126             precision = deviance,
127             divisions = currentDivisions,
128             convergence = true
129         )
130     }
131
132     private fun nextIntegralApproximation(
133         start: Double,
134         endInclusive: Double,
135         divisions: Int,
136         function: (Double) -> Double
137     ): Double {
138         val step = (endInclusive - start) / divisions
139
140         var integralSum = 0.0
141
142         var currentParam = start
143
144         repeat(divisions) {
145             integralSum += function(currentParam) * step
146             currentParam += step
147         }
```

```
148
149         return integralSum
150     }
151 }
152
153 package io.github.e1turin.entities.integrator.model
154
155 import io.github.e1turin.shared.config.functionWithLabelStore
156 import io.github.e1turin.shared.config.maxNIteration
157 import io.github.e1turin.shared.lib.length
158 import io.github.e1turin.shared.model.settings.IntegrationParameters
159 import io.github.e1turin.shared.model.solution.IntegrationResult
160 import kotlin.math.abs
161 import kotlin.math.pow
162
163 class RightRectangleIntegrator(private val settings: IntegrationParameters
    ) : Integrator {
164
165     init {
166         check(settings.range.length > 0) { "Inspected range mustn't be
    empty. Check its bounds." }
167         check(functionWithLabelStore.keys.contains(settings.functionLabel)
    ) { "Function must be in defined set." }
168     }
169
170     override fun integrate(): IntegrationResult {
171         val function: (Double) -> Double = functionWithLabelStore[settings
    .functionLabel]!!
172
173         var currentIntegrationSum = 0.0
174         var currentDivisions = settings.divisions
175         var deviance: Double = Double.MAX_VALUE
176
177         for(iteration in 0..maxNIteration) {
178             val integralSum = nextIntegralApproximation(
179                 start = settings.range.start,
180                 endInclusive = settings.range.endInclusive,
181                 divisions = currentDivisions,
182                 function = function
183             )
184
185             val preciseIntegralSum = nextIntegralApproximation(
186                 start = settings.range.start,
187                 endInclusive = settings.range.endInclusive,
188                 divisions = currentDivisions * 2,
189                 function = function
190             )
191
192             deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(2)
    - 1)
193             currentIntegrationSum = preciseIntegralSum
194             currentDivisions *= 2
195
196             if (deviance < settings.precision ) break
197         }
198
199
200         return IntegrationResult(
201             area = currentIntegrationSum,
202             precision = deviance,
```

```
            divisions = currentDivisions,
            convergence = false
        )
    }

    private fun nextIntegralApproximation(
        start: Double,
        endInclusive: Double,
        divisions: Int,
        function: (Double) -> Double
    ): Double {
        val step = (endInclusive - start) / divisions

        var integralSum = 0.0

        var currentParam = start + step

        repeat(divisions) {
            integralSum += function(currentParam) * step
            currentParam += step
        }

        return integralSum
    }
}

package io.github.e1turin.entities.integrator.model

import io.github.e1turin.shared.config.functionWithLabelStore
import io.github.e1turin.shared.config.maxNIteration
import io.github.e1turin.shared.lib.length
import io.github.e1turin.shared.model.settings.IntegrationParameters
import io.github.e1turin.shared.model.solution.IntegrationResult
import kotlin.math.abs
import kotlin.math.pow

class SimpsonIntegrator(private val settings: IntegrationParameters) :
    Integrator {

    init {
        check(settings.range.length > 0) { "Inspected range mustn't be
    empty. Check its bounds." }
        check(functionWithLabelStore.keys.contains(settings.functionLabel)
    ) { "Function must be in defined set." }
    }

    override fun integrate(): IntegrationResult {
        val function: (Double) -> Double = functionWithLabelStore[settings
    .functionLabel]!!

        var currentIntegrationSum = 0.0
        var currentDivisions = settings.divisions
        var deviance: Double = Double.MAX_VALUE

        for(iteration in 0..maxNIteration) {
            val integralSum = nextIntegralApproximation(
                start = settings.range.start,
                endInclusive = settings.range.endInclusive,
                divisions = currentDivisions,
                function = function
```

```
259             )
260
261             val preciseIntegralSum = nextIntegralApproximation(
262                 start = settings.range.start,
263                 endInclusive = settings.range.endInclusive,
264                 divisions = currentDivisions * 2,
265                 function = function
266             )
267
268             deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(4)
    - 1)
269             currentIntegrationSum = preciseIntegralSum
270             currentDivisions *= 2
271
272             if (deviance < settings.precision ) break
273         }
274
275         return IntegrationResult(
276             area = currentIntegrationSum,
277             precision = deviance,
278             divisions = currentDivisions,
279             convergence = false
280         )
281     }
282
283     private fun nextIntegralApproximation(
284         start: Double,
285         endInclusive: Double,
286         divisions: Int,
287         function: (Double) -> Double
288     ): Double {
289         val step = (endInclusive - start) / divisions
290
291         var integralSum = 0.0
292
293         var currentParam = start + step
294
295         repeat(divisions - 1) {
296             integralSum += function(currentParam) * 2 * (2 - it % 2)
297             currentParam += step
298         }
299
300         val resultSum = (function(start) + function(endInclusive) +
    integralSum) / 3 * step
301
302         return resultSum
303     }
304 }
305
306 package io.github.e1turin.entities.integrator.model
307
308 import io.github.e1turin.shared.config.functionWithLabelStore
309 import io.github.e1turin.shared.config.maxNIteration
310 import io.github.e1turin.shared.lib.length
311 import io.github.e1turin.shared.model.settings.IntegrationParameters
312 import io.github.e1turin.shared.model.solution.IntegrationResult
313 import kotlin.math.abs
314 import kotlin.math.pow
315
316 class TrapezoidIntegrator(private val settings: IntegrationParameters) :
```

```
    Integrator {

     init {
         check(settings.range.length > 0) { "Inspected range mustn't be
    empty. Check its bounds." }
         check(functionWithLabelStore.keys.contains(settings.functionLabel)
    ) { "Function must be in defined set." }
     }

     override fun integrate(): IntegrationResult {
         val function: (Double) -> Double = functionWithLabelStore[settings
    .functionLabel]!!

         var currentIntegrationSum = 0.0
         var currentDivisions = settings.divisions
         var deviance: Double = Double.MAX_VALUE

         for(iteration in 0..maxNIteration) {
             val integralSum = nextIntegralApproximation(
                 start = settings.range.start,
                 endInclusive = settings.range.endInclusive,
                 divisions = currentDivisions,
                 function = function
             )

             val preciseIntegralSum = nextIntegralApproximation(
                 start = settings.range.start,
                 endInclusive = settings.range.endInclusive,
                 divisions = currentDivisions * 2,
                 function = function
             )

             deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(2)
     - 1)
             currentIntegrationSum = preciseIntegralSum
             currentDivisions *= 2

             if (deviance < settings.precision ) break
         }

         return IntegrationResult(
             area = currentIntegrationSum,
             precision = deviance,
             divisions = currentDivisions,
             convergence = false
         )
     }

     private fun nextIntegralApproximation(
         start: Double,
         endInclusive: Double,
         divisions: Int,
         function: (Double) -> Double
     ): Double {
         val step = (endInclusive - start) / divisions

         var integralSum = 0.0

         var currentParam = start + step

```

```
372         repeat(divisions - 1) {
373             integralSum += function(currentParam)
374             currentParam += step
375         }
376 
377         val resultSum = ((function(start) + function(endInclusive)) / 2 +
    integralSum) * step
378 
379         return resultSum
380     }
381 }
```

Листинг 1.1: Реализация на языке программирования Kolin основной логики методов интегрирования

5. Вывод

В ходе выполнения данной лабораторной работы углубили понимание работы методов исленного интегрирования.

Выяснили на практике, что разные методы по разному точны и имеют определенные сферы применения. Наиболее точными оказались метод Симсона и метод Ньютона-Котеса для $n = 5$.

Литература

[1] Ссылка на личный репозиторий GitHub: https://github.com/e1turin/itmo-comp-math/tree/main/lab-3