

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: Р32131

Преподаватель:
Мальшева Т. А.,
Бострикова Д. К.

Санкт-Петербург, 2023 г.

Содержание

Лабораторная работа № 3. Численное интегрирование	2
1. Задание варианта № 21	2
2. Цель работы	2
3. Вычислительная часть работы	3
4. Программная часть работы	3
5. Вывод	13

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 значение 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 программирования 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

```

1 package io.github.elturin.entities.integrator.model
2
3 import io.github.elturin.shared.config.functionWithLabelStore
4 import io.github.elturin.shared.config.maxNIteration
5 import io.github.elturin.shared.lib.length
6 import io.github.elturin.shared.model.settings.IntegrationParameters
7 import io.github.elturin.shared.model.solution.IntegrationResult
8 import kotlin.math.abs
9 import kotlin.math.pow
10
11 class CenterRectangleIntegrator(private val settings:
12     IntegrationParameters) : Integrator {
13
14     init {
15         check(settings.range.length > 0) { "Inspected range mustn't be
16         empty. Check its bounds." }
17         check(functionWithLabelStore.keys.contains(settings.functionLabel)
18         ) { "Function must be in defined set." }
19     }
20
21     override fun integrate(): IntegrationResult {
22         val function: (Double) -> Double = functionWithLabelStore[settings
23         .functionLabel]!!
24
25         var currentIntegrationSum = 0.0
26         var currentDivisions = settings.divisions
27         var deviance: Double = Double.MAX_VALUE
28
29         for(iteration in 0..maxNIteration) {
30             val integralSum = nextIntegralApproximation(
31                 start = settings.range.start,
32                 endInclusive = settings.range.endInclusive,
33                 divisions = currentDivisions,
34                 function = function
35             )
36
37             val preciseIntegralSum = nextIntegralApproximation(

```

```

34         start = settings.range.start,
35         endInclusive = settings.range.endInclusive,
36         divisions = currentDivisions * 2,
37         function = function
38     )
39
40     deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow
(2) - 1)
41     currentIntegrationSum = preciseIntegralSum
42     currentDivisions *= 2
43
44     if (deviance < settings.precision) break
45 }
46
47
48     return IntegrationResult(
49         area = currentIntegrationSum,
50         precision = deviance,
51         divisions = currentDivisions,
52         convergence = false
53     )
54 }
55
56 private fun nextIntegralApproximation(
57     start: Double,
58     endInclusive: Double,
59     divisions: Int,
60     function: (Double) -> Double
61 ): Double {
62     val step = (endInclusive - start) / divisions
63
64     var integralSum = 0.0
65
66     var currentParam = start + step / 2
67
68     repeat(divisions) {
69         integralSum += function(currentParam) * step
70         currentParam += step
71     }
72
73     return integralSum
74 }
75 }
76
77 package io.github.elturin.entities.integrator.model
78
79 import io.github.elturin.shared.config.functionWithLabelStore
80 import io.github.elturin.shared.config.maxNIteration
81 import io.github.elturin.shared.lib.length
82 import io.github.elturin.shared.model.settings.IntegrationParameters
83 import io.github.elturin.shared.model.solution.IntegrationResult
84 import kotlin.math.abs
85 import kotlin.math.pow
86
87 class LeftRectangleIntegrator(private val settings: IntegrationParameters)
88     : Integrator {
89     init {
90         check(settings.range.length > 0) { "Inspected range mustn't be
empty. Check its bounds." }

```



```

91     check(functionWithLabelStore.keys.contains(settings.functionLabel)
92 ) { "Function must be in defined set." }
93 }
94
95 override fun integrate(): IntegrationResult {
96     val function: (Double) -> Double = functionWithLabelStore[settings
97 .functionLabel]!!
98
99     var currentIntegrationSum = 0.0
100     var currentDivisions = settings.divisions
101     var deviance: Double = Double.MAX_VALUE
102
103     for(iteration in 0..maxNIteration) {
104         val integralSum = nextIntegralApproximation(
105             start = settings.range.start,
106             endInclusive = settings.range.endInclusive,
107             divisions = currentDivisions,
108             function = function
109         )
110
111         val preciseIntegralSum = nextIntegralApproximation(
112             start = settings.range.start,
113             endInclusive = settings.range.endInclusive,
114             divisions = currentDivisions * 2,
115             function = function
116         )
117
118         deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(2)
119 - 1)
120
121         currentIntegrationSum = preciseIntegralSum
122         currentDivisions *= 2
123
124         if (deviance < settings.precision ) break
125     }
126
127     return IntegrationResult(
128         area = currentIntegrationSum,
129         precision = deviance,
130         divisions = currentDivisions,
131         convergence = true
132     )
133 }
134
135 private fun nextIntegralApproximation(
136     start: Double,
137     endInclusive: Double,
138     divisions: Int,
139     function: (Double) -> Double
140 ): Double {
141     val step = (endInclusive - start) / divisions
142
143     var integralSum = 0.0
144
145     var currentParam = start
146
147     repeat(divisions) {
148         integralSum += function(currentParam) * step
149         currentParam += step
150     }

```

```

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

```

```

203         divisions = currentDivisions,
204         convergence = false
205     )
206 }
207
208 private fun nextIntegralApproximation(
209     start: Double,
210     endInclusive: Double,
211     divisions: Int,
212     function: (Double) -> Double
213 ): Double {
214     val step = (endInclusive - start) / divisions
215
216     var integralSum = 0.0
217
218     var currentParam = start + step
219
220     repeat(divisions) {
221         integralSum += function(currentParam) * step
222         currentParam += step
223     }
224
225     return integralSum
226 }
227 }
228
229 package io.github.elturin.entities.integrator.model
230
231 import io.github.elturin.shared.config.functionWithLabelStore
232 import io.github.elturin.shared.config.maxNIteration
233 import io.github.elturin.shared.lib.length
234 import io.github.elturin.shared.model.settings.IntegrationParameters
235 import io.github.elturin.shared.model.solution.IntegrationResult
236 import kotlin.math.abs
237 import kotlin.math.pow
238
239 class SimpsonIntegrator(private val settings: IntegrationParameters) :
240     Integrator {
241     init {
242         check(settings.range.length > 0) { "Inspected range mustn't be
243         empty. Check its bounds." }
244         check(functionWithLabelStore.keys.contains(settings.functionLabel)
245         ) { "Function must be in defined set." }
246     }
247
248     override fun integrate(): IntegrationResult {
249         val function: (Double) -> Double = functionWithLabelStore[settings
250         .functionLabel]!!
251
252         var currentIntegrationSum = 0.0
253         var currentDivisions = settings.divisions
254         var deviance: Double = Double.MAX_VALUE
255
256         for(iteration in 0..maxNIteration) {
257             val integralSum = nextIntegralApproximation(
258                 start = settings.range.start,
259                 endInclusive = settings.range.endInclusive,
260                 divisions = currentDivisions,
261                 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
270         currentIntegrationSum = preciseIntegralSum
271         currentDivisions *= 2
272
273         if (deviance < settings.precision) break
274     }
275
276     return IntegrationResult(
277         area = currentIntegrationSum,
278         precision = deviance,
279         divisions = currentDivisions,
280         convergence = false
281     )
282 }
283
284 private fun nextIntegralApproximation(
285     start: Double,
286     endInclusive: Double,
287     divisions: Int,
288     function: (Double) -> Double
289 ): Double {
290     val step = (endInclusive - start) / divisions
291
292     var integralSum = 0.0
293
294     var currentParam = start + step
295
296     repeat(divisions - 1) {
297         integralSum += function(currentParam) * 2 * (2 - it % 2)
298         currentParam += step
299     }
300
301     val resultSum = (function(start) + function(endInclusive) +
integralSum) / 3 * step
302
303     return resultSum
304 }
305
306 package io.github.elturin.entities.integrator.model
307
308 import io.github.elturin.shared.config.functionWithLabelStore
309 import io.github.elturin.shared.config.maxNIteration
310 import io.github.elturin.shared.lib.length
311 import io.github.elturin.shared.model.settings.IntegrationParameters
312 import io.github.elturin.shared.model.solution.IntegrationResult
313 import kotlin.math.abs
314 import kotlin.math.pow
315
316 class TrapezoidIntegrator(private val settings: IntegrationParameters) :

```

```

317 Integrator {
318     init {
319         check(settings.range.length > 0) { "Inspected range mustn't be
empty. Check its bounds." }
320         check(functionWithLabelStore.keys.contains(settings.functionLabel)
) { "Function must be in defined set." }
321     }
322
323     override fun integrate(): IntegrationResult {
324         val function: (Double) -> Double = functionWithLabelStore[settings
.functionLabel]!!
325
326         var currentIntegrationSum = 0.0
327         var currentDivisions = settings.divisions
328         var deviance: Double = Double.MAX_VALUE
329
330         for(iteration in 0..maxNIteration) {
331             val integralSum = nextIntegralApproximation(
332                 start = settings.range.start,
333                 endInclusive = settings.range.endInclusive,
334                 divisions = currentDivisions,
335                 function = function
336             )
337
338             val preciseIntegralSum = nextIntegralApproximation(
339                 start = settings.range.start,
340                 endInclusive = settings.range.endInclusive,
341                 divisions = currentDivisions * 2,
342                 function = function
343             )
344
345             deviance = abs(integralSum - preciseIntegralSum) / (2.0.pow(2)
- 1)
346             currentIntegrationSum = preciseIntegralSum
347             currentDivisions *= 2
348
349             if (deviance < settings.precision ) break
350         }
351
352         return IntegrationResult(
353             area = currentIntegrationSum,
354             precision = deviance,
355             divisions = currentDivisions,
356             convergence = false
357         )
358     }
359
360     private fun nextIntegralApproximation(
361         start: Double,
362         endInclusive: Double,
363         divisions: Int,
364         function: (Double) -> Double
365     ): Double {
366         val step = (endInclusive - start) / divisions
367
368         var integralSum = 0.0
369
370         var currentParam = start + step
371

```

```

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 Kotlin 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>