

Министерство науки и высшего образования Российской  
Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ  
УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ  
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра прикладной математики и искусственного интеллекта

Направление подготовки: 01.03.04 – Прикладная математика

**ОТЧЁТ**

По дисциплине «Численные методы»

на тему:

«Вычисление интеграла с помощью квадратурных формул»

Выполнил:  
студент группы 09-221  
Саитов М.А.  
Преподаватель:  
Глазырина О.В.

Казань, 2024 год

## Содержание

<b>1</b>	<b>Постановка задачи</b>	<b>3</b>
<b>2</b>	<b>Ход работы</b>	<b>4</b>
<b>3</b>	<b>Выводы</b>	<b>9</b>
<b>4</b>	<b>Листинг программы</b>	<b>10</b>

# 1 Постановка задачи

## 2 Ход работы

### 1. Метод прогонки:

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.1	-0.000695158	9e-05	0.000785158
0.2	-0.00039375	0.00128	0.00167375
0.3	0.00300748	0.00567	0.00266252
0.4	0.0116375	0.01536	0.00372252
0.5	0.0264841	0.03125	0.00476594
0.6	0.0462174	0.05184	0.00562257
0.7	0.0660077	0.07203	0.00602232
0.8	0.0763388	0.08192	0.00558118
0.9	0.0618207	0.06561	0.00378929

Таблица 1 - таблица значений для формул метода прогонки при  $n = 10$

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.05	-7.79134e-05	5.9375e-06	8.38509e-05
0.1	-8.5795e-05	9e-05	0.000175795
0.15	0.000152985	0.000430313	0.000277328
0.2	0.000889159	0.00128	0.000390841
0.25	0.00241095	0.00292969	0.00051874
0.3	0.00500727	0.00567	0.000662726
0.35	0.00893085	0.00975406	0.000823217
0.4	0.0143611	0.01536	0.000998868
0.45	0.0213673	0.0225534	0.00118616
0.5	0.0298709	0.03125	0.00137908
0.55	0.039609	0.0411778	0.0015688
0.6	0.0500966	0.05184	0.00174344
0.65	0.0605893	0.0624772	0.00188784
0.7	0.0700467	0.07203	0.00198335
0.75	0.0770939	0.0791016	0.00200763
0.8	0.0799855	0.08192	0.0019345
0.85	0.0765672	0.0783009	0.00173375
0.9	0.064239	0.06561	0.001371
0.95	0.0399178	0.0407253	0.000807495

Таблица 2 - таблица значений для формул метода прогонки при  $n = 20$

## 2. Метод Якоби:

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.1	0.102819	9e-05	0.102729
0.2	0.199156	0.00128	0.197876
0.3	0.293056	0.00567	0.287386
0.4	0.388895	0.01536	0.373535
0.5	0.488671	0.03125	0.457421
0.6	0.592915	0.05184	0.541075
0.7	0.697403	0.07203	0.625373
0.8	0.794114	0.08192	0.712194
0.9	0.868238	0.06561	0.802628

Таблица 3 - таблица значений для формулы метода Якоби при  $n = 10$

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.05	0.0562275	5.9375e-06	0.0562215
0.1	0.110114	9e-05	0.110024
0.15	0.162167	0.000430313	0.161737
0.2	0.212985	0.00128	0.211705
0.25	0.263096	0.00292969	0.260167
0.3	0.313096	0.00567	0.307426
0.35	0.363411	0.00975406	0.353657
0.4	0.414498	0.01536	0.399138
0.45	0.466555	0.0225534	0.444002
0.5	0.519757	0.03125	0.488507
0.55	0.573939	0.0411778	0.532761
0.6	0.628847	0.05184	0.577007
0.65	0.683819	0.0624772	0.621342
0.7	0.738021	0.07203	0.665991
0.75	0.790159	0.0791016	0.711058
0.8	0.83867	0.08192	0.75675
0.85	0.881484	0.0783009	0.803183
0.9	0.916161	0.06561	0.850551
0.95	0.939716	0.0407253	0.898991

Таблица 4 - таблица значений для формулы метода Якоби при  $n = 20$

### 3. Метод Зейделя:

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.1	0.103011	9e-05	0.102921
0.2	0.199491	0.00128	0.198211
0.3	0.293725	0.00567	0.288055
0.4	0.389619	0.01536	0.374259
0.5	0.489654	0.03125	0.458404
0.6	0.593779	0.05184	0.541939
0.7	0.698296	0.07203	0.626266
0.8	0.794716	0.08192	0.712796
0.9	0.868608	0.06561	0.802998

Таблица 5 - таблица значений для формулы метода Зейделя при  $n = 10$

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.05	0.0562405	5.9375e-06	0.0562346
0.1	0.110139	9e-05	0.110049
0.15	0.162222	0.000430313	0.161792
0.2	0.213055	0.00128	0.211775
0.25	0.263206	0.00292969	0.260276
0.3	0.313218	0.00567	0.307548
0.35	0.363576	0.00975406	0.353822
0.4	0.41467	0.01536	0.39931
0.45	0.466766	0.0225534	0.444212
0.5	0.519965	0.03125	0.488715
0.55	0.574174	0.0411778	0.532996
0.6	0.629068	0.05184	0.577228
0.65	0.684051	0.0624772	0.621574
0.7	0.738227	0.07203	0.666197
0.75	0.790358	0.0791016	0.711257
0.8	0.838832	0.08192	0.756912
0.85	0.881621	0.0783009	0.80332
0.9	0.916252	0.06561	0.850642
0.95	0.939766	0.0407253	0.899041

Таблица 6 - таблица значений для формулы метода Зейделя при  $n = 20$

#### 4. Метод Релаксации:

$w$	$k$
0.1	497
0.2	58
0.3	24
0.4	13
0.5	8
0.6	6
0.7	4
0.8	3
0.9	3
1	2

Таблица 7 - таблица значений для формулы метода Релаксации при  $n = 10$

$w$	$k$
0.1	2446
0.2	243
0.3	98
0.4	52
0.5	33
0.6	23
0.7	16
0.8	12
0.9	10
1	8

Таблица 8 - таблица значений для формулы метода Релаксации при  $n = 20$

5. Метод наискорейшего спуска:

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.1	0.102819	9e-05	0.102729
0.2	0.199156	0.00128	0.197876
0.3	0.293056	0.00567	0.287386
0.4	0.388895	0.01536	0.373535
0.5	0.488671	0.03125	0.457421
0.6	0.592915	0.05184	0.541075
0.7	0.697403	0.07203	0.625373
0.8	0.794114	0.08192	0.712194
0.9	0.868238	0.06561	0.802628

Таблица 9 - таблица значений для формулы метода наискорейшего спуска при  $n = 10$

$i * h$	$y_i$	$u(ih)$	$ y_i - u(ih) $
0.05	0.0562275	5.9375e-06	0.0562215
0.1	0.110114	9e-05	0.110024
0.15	0.162167	0.000430313	0.161737
0.2	0.212985	0.00128	0.211705
0.25	0.263096	0.00292969	0.260167
0.3	0.313096	0.00567	0.307426
0.35	0.363411	0.00975406	0.353657
0.4	0.414498	0.01536	0.399138
0.45	0.466555	0.0225534	0.444002
0.5	0.519757	0.03125	0.488507
0.55	0.573939	0.0411778	0.532761
0.6	0.628847	0.05184	0.577007
0.65	0.683819	0.0624772	0.621342
0.7	0.738021	0.07203	0.665991
0.75	0.790159	0.0791016	0.711058
0.8	0.83867	0.08192	0.75675
0.85	0.881484	0.0783009	0.803183
0.9	0.916161	0.06561	0.850551
0.95	0.939716	0.0407253	0.898991

Таблица 10 - таблица значений для формулы метода наискорейшего спуска при  $n = 20$



### 3 Выводы

Проделав все вычисления, можно сделать выводы, что более комплексные методы вычисления интеграла, как формула Гаусса и Симпсона, показывают наилучшие результаты за меньшее количество разбиений. В это же время худшие результаты вычисления показывают методы правых прямоугольников и метод трапеций, приводя к довольно большому значению ошибки.

## 4 Листинг программы

```
1 #pragma once
2
3 #include <iostream>
4 #include <math.h>
5 #include <iomanip>
6 #include <vector>
7 #include <algorithm>
8
9 double a(double i, double h){
10     return (1 + i*h);
11 }
12 double g(double i, double h){
13     return (1 + i*h);
14 }
15 double f(double i, double h){
16     return -pow(i*h, 6) + 26*pow(i*h, 4) + 4*pow(i*h, 3) - 12*pow(i*h
17     , 2);
18 }
19 double denominator(int i, double h){
20     return a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h);
21 }
22 void printTable(int n, const std::vector<double> y, const std::vector
23 <double> u){
24     double h = 1.0 / n;
25     std::cout << std::setw(12) << "i*h" << " | " << std::setw(12) <<
26     "yi" << " | " << std::setw(12) << " u(ih)" << " | " << std:::
27     setw(12) << "|yi - u(ih)|" << std::endl;
28     for(int i = 1; i < n; i++){
29         double ui = u[i];
30         double yi = y[i];
31         std::cout << i*h << " & " << std::setw(12) << yi << " & "
32         << std::setw(12) << ui << " & " << std::setw(12) << abs(
33         yi - ui) << "\\\\" << std::endl;
34         std::cout << "\\hline\n";
35     }
36 }
37
38 void SweepMethod_tableOutput(int n){
39     std::cout << "\033[1m" << "\033[3m" << "Sweep Method\n" << "
40     \033[0m";
41     double h = 1.0 / n;
```

```

36     std::vector<double> alpha(n+1);
37     std::vector<double> betta(n+1);
38     for(int i = 2; i <= n; i++){
39         alpha[i] = a(i,h)/
40             ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i
41                 -1, h));
42         betta[i] = (f(i-1, h)*pow(h,2) + betta[i-1] * a(i-1, h))/
43             ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i
44                 -1, h));
45     }
46     std::vector<double> y(n+1);
47     y[n] = 0;
48     for(int i = n-1; i > 0; i--){
49         y[i] = alpha[i+1] * y[i+1] + betta[i+1];
50     }
51     std::vector<double> u(n);
52     for(int i = 1; i < n; i++){
53         u[i] = pow(i*h, 4) * (1 - (i*h));
54     }
55     printTable(n, y, u);
56     std::cout << "---\n";
57 }
58
59 void YakobiMethod_tableOutput(int n){
60     std::cout << "\033[1m" << "\033[3m" << "Yakobi Method\n" << "
61         \033[0m";
62     double h = 1.0/n;
63     double eps = pow(h, 3);
64     std::vector<double> y_k(n);
65     for(int i = 1; i < n-1; i++){
66         y_k[i] = f(i, h)*pow(h,2) / denominator(i, h);
67     }
68     double r = 1;
69     std::vector<double> y_k_1(n);
70     std::vector<double> u(n);
71     while(fabs(r) > eps){
72         y_k = y_k_1;
73         for(int i = 1; i < n; i++){
74             y_k_1[i] = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f(i,
75                 h)*pow(h, 2))
76                 /(denominator(i, h));
77             u[i] = pow(i*h, 4) * (1 - (i*h));
78             if(i == 1) r = fabs((y_k_1[i] - y_k[i]) / y_k_1[i]);

```

```

75         else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
76             );
77     }
78     printTable(n, y_k_1, u);
79     std::cout << "---\n";
80 }
81
82 void ZeidelMethod_tableOutput(int n){
83     std::cout << "\033[1m" << "\033[3m" << "Zeidel Method\n" << "
84         \033[0m";
85     double h = 1.0/n;
86     double eps = pow(h, 3);
87     std::vector<double> y_k(n);
88     for(int i = 1; i < n-1; i++){
89         y_k[i] = f(i, h)*pow(h,2) / denominator(i, h);
90     }
91     double r = 1;
92     std::vector<double> y_k_1(n);
93     std::vector<double> u(n);
94     while(fabs(r) > eps){
95         y_k = y_k_1;
96         for(int i = 1; i < n; i++){
97             y_k_1[i] = (a(i, h)*y_k_1[i-1] + a(i+1, h)*y_k[i+1] + f(i
98                 , h)*pow(h, 2))
99                 /(denominator(i, h));
100             u[i] = pow(i*h, 4) * (1 - (i*h));
101             if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
102             else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
103                 );
104         }
105     }
106     printTable(n, y_k_1, u);
107     std::cout << "---\n";
108 }
109
110 void relaxationMethod_tableOutput(int n){
111     std::cout << "\033[1m" << "\033[3m" << "Relaxation Method\n" << "
112         \033[0m";
113     std::cout << std::setw(3) << "w" << " | " << "k" << std::endl;
114     double h = 1.0 / n;
115     double eps = pow(h, 3);
116     std::vector<double> y_k(n);

```

```

113     for(int i = 1; i < n-1; i++){
114         y_k[i] = f(i, h)*pow(h,2) / denominator(i, h);
115     }
116     double r = 1;
117     std::vector<double> y_k_1(n);
118     std::vector<double> u(n);
119     for(double w = 0.1; w < 1; w += 0.1){
120         int k = 0;
121         while(fabs(r) > eps){
122             y_k = y_k_1;
123             for(int i = 1; i < n; i++){
124                 double I = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f
125                     (i, h)*pow(h, 2))
126                     /(denominator(i, h));
127                 y_k_1[i] = (1 - w) * y_k[i] + w*I;
128                 u[i] = pow(i*h, 4) * (1 - (i*h));
129                 if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
130                 else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i
131                     ]), r);
132             }
133             k++;
134         }
135         r = 1;
136         std::cout << std::setw(3) << w << " & " << k << "\\\\" << std
137             ::endl;
138         std::cout << "\\hline\n";
139     }
140     std::cout << "---\n";
141 }
142 // ???
143 void descentMethod_tableOutput(int n){
144     std::cout << "\033[1m" << "\033[3m" << "Descent Method\n" << "
145         \033[0m";
146     double h = 1.0 / n;
147     double eps = pow(h, 3);
148     std::vector<double> y_k(n);
149     for(int i = 1; i < n-1; i++){
150         y_k[i] = f(i, h)*pow(h,2) / denominator(i, h);
151     }
152     double r = 1;
153     std::vector<double> y_k_1(n);

```

```

152     std::vector<double> u(n);
153     while(fabs(r) > eps){
154         y_k = y_k_1;
155         for(int i = 1; i < n; i++){
156             y_k_1[i] = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f(i,
157                 h)*pow(h, 2))
158                 /(denominator(i, h));
159             u[i] = pow(i*h, 4) * (1 - (i*h));
160             if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
161             else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
162                 );
163         }
164     }
165     printTable(n, y_k_1, u);
166     std::cout << " ---\n";
167 }

```