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 искусственого интеллекта

Направление подготовки: 01.03.04 – Прикладная математика

ОТЧЁТ

По дисциплине «Численные методы»
на тему:
«Система линейных алгебраических уравнений»

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

Казань, 2024 год

Содержание

1 Постановка задачи

2 Ход работы

1. Метод прогонки:

$i * h$	y_i	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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	$u(ih)$	$\lvert y_i - u(ih)\rvert$
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, показыают наилучшие результаты за меньшее количество разбиений. В это же время худшие результаты вычисления показыают методы правых прямоугольников и метод трапеций, приводя к довольно большому значению ошибки.

4 Листинг программы

```cpp
#pragma once

#include <iostream>
#include <math.h>
#include <iomanip>
#include <vector>
#include <algorithm>

double a(double i, double h){
    return (1 + i*h);
}
double g(double i, double h){
    return (1 + i*h);
}
double f(double i, double h){
    return -pow(i*h, 6) + 26*pow(i*h, 4) + 4*pow(i*h, 3) - 12*pow(i*h
        , 2);
}
double denominator(int i, double h){
    return (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
}
void printTable(int n, const std::vector<double> y, const std::vector
    <double> u){
    double h = 1.0 / n;
    std::cout << std::setw(12) << "i*h" << " | " << std::setw(12) <<
        "yi" << " | " << std::setw(12) << " u(ih)"  << " | " << std::
        setw(12) << "|yi - u(ih)|" << std::endl;
    for(int i = 1; i < n; i++){
        double ui = u[i];
        double yi = y[i];
        std::cout  << i*h << " & " << std::setw(12) << yi << " & "
            << std::setw(12) << ui << " & " << std::setw(12) << abs(
                yi - ui) << "\\\\" << std::endl;
        std::cout << "\\hline\n";
    }

}
std::vector<double> SweepMethod_result(int n){
    double h = 1.0 / n;
    std::vector<double> alpha(n+1);
    std::vector<double> betta(n+1);
```

```
37      for(int i = 2; i <= n; i++){
38          alpha[i] = a(i,h)/
39              ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i
                    -1, h));
40          betta[i] = (f(i-1, h)*pow(h,2) + betta[i-1] * a(i-1, h))/
41              ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i
                    -1, h));
42      }
43      std::vector<double> y(n+1);
44      y[n] = 0;
45      for(int i = n-1; i > 0; i--){
46          y[i] = alpha[i+1] * y[i+1] + betta[i+1];
47      }
48      return y;
49  }
50  void SweepMethod_tableOutput(int n){
51      std::cout << "\033[1m" << "\033[3m" << "Sweep Method\n" << "
            \033[0m";
52          std::cout << std::setw(4) << "i*h" << " | " << std::setw(12)
                << "yi" << " | " << std::setw(12) << " ui"  << " | " <<
                std::setw(12) << "|yi - ui|" << std::endl;
53      double h = 1.0 / n;
54      std::vector<double> y = SweepMethod_result(n);
55      std::vector<double> u(n);
56      for(int i = 1; i < n; i++){
57          u[i] = pow(i*h, 4) * (1 - (i*h));
58      }
59      for(int i = 1; i < n; i++){
60          double ui = u[i];
61          double yi = y[i];
62          std::cout << std::setw(4) << i*h << " | " << std::setw(12) <<
                yi << " | "
63              << std::setw(12) << ui << " | " << std::setw(12) << abs(
                    yi - ui) << std::endl;
64      }
65      std::cout << "---\n";
66  }
67
68  void YakobiMethod_tableOutput(int n){
69      std::cout << "\033[1m" << "\033[3m" << "Yakobi Method\n" << "
            \033[0m";
70      std::cout << std::setw(4) << "i*h" << " | " << std::setw(12) << "
            yi" << " | " << std::setw(12) << " yi_k"  << " | " << std::
```

```
        setw(12) << "|yi - yi_k|" << " | " << "k" << std::endl;
71
72      double h = 1.0/n;
73      double eps = pow(h, 3);
74      std::vector<double> y_k(n);
75      for(int i = 1; i < n-1; i++){
76          y_k[i] = f(i, h)*pow(h,2) / (a(i, h) + a(i+1, h) + pow(h, 2)*
                g(i, h));
77      }
78      double r = 1;
79      std::vector<double> y_k_1(n);
80      int k_count = 0;
81      while(fabs(r) > eps){
82          y_k = y_k_1;
83          for(int i = 1; i < n; i++){
84              y_k_1[i] = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f(i,
                    h)*pow(h, 2))
85                  /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
86              if(i == 1) r = fabs((y_k_1[i] - y_k[i]) / y_k_1[i]);
87              else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
                    );
88          }
89          k_count++;
90      }
91      std::vector<double> yi = SweepMethod_result(n);
92      // Table output
93      for(int i = 1; i < n; i++){
94          std::cout  << std::setw(4) << i*h << " | " << std::setw(12)
                << yi[i] << " | "
95              << std::setw(12) << y_k_1[i] << " | " << std::setw(12) <<
                    abs(yi[i] - y_k_1[i]) << " | " << k_count << std::
                    endl;
96      }
97      std::cout << "---\n";
98  }
99
100 void ZeidelMethod_tableOutput(int n){
101     std::cout << "\033[1m" << "\033[3m" << "Zeidel Method\n" << "
            \033[0m";
102     std::cout << std::setw(4) << "i*h" << " | " << std::setw(12) << "
            yi" << " | " << std::setw(12) << " yi_k"  << " | " << std::
            setw(12) << "|yi - yi_k|" << " | " << "k" << std::endl;
103
```

```
104     double h = 1.0/n;
105     double eps = pow(h, 3);
106     std::vector<double> y_k(n);
107     for(int i = 1; i < n-1; i++){
108         y_k[i] = f(i, h)*pow(h,2) /
109             a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h);
110     }
111     double r = 1;
112     std::vector<double> y_k_1(n);
113     int k_count = 0;
114     while(fabs(r) > eps){
115         y_k = y_k_1;
116         for(int i = 1; i < n; i++){
117             y_k_1[i] = (a(i, h)*y_k_1[i-1] + a(i+1, h)*y_k[i+1] + f(i
                , h)*pow(h, 2))
118                 /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
119             if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
120             else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
                );
121         }
122         k_count++;
123     }
124     std::vector<double> yi = SweepMethod_result(n);
125     // Table output
126     for(int i = 1; i < n; i++){
127         std::cout  << std::setw(4) << i*h << " | " << std::setw(12)
             << yi[i] << " | "
128             << std::setw(12) << y_k_1[i] << " | " << std::setw(12) <<
                 abs(yi[i] - y_k_1[i]) << " | " << k_count << std::
                endl;
129         //std::cout << "\\hline\n";
130     }
131     std::cout << "---\n";
132 }
133 
134 void relaxationMethod_tableOutput(int n){
135     std::cout << "\033[1m" << "\033[3m" << "Relaxation Method\n" << "
        \033[0m";
136     //std::cout << std::setw(3) << "w" << " | " << "k" << std::endl;
137     double h = 1.0 / n;
138     double eps = pow(h, 3);
139     std::vector<double> y_k(n);
140     for(int i = 1; i < n-1; i++){
```

```
141         y_k[i] = f(i, h)*pow(h,2) /
142             (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
143     }
144     double r = 1;
145     std::vector<double> y_k_1(n);
146     std::vector<double> u(n);
147     for(double w = 0.1; w < 1; w += 0.1){
148         int k = 0;
149         while(fabs(r) > eps){
150             y_k = y_k_1;
151             for(int i = 1; i < n; i++){
152                 double I = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f
                        (i, h)*pow(h, 2))
153                     /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
154                 y_k_1[i] = (1 - w) * y_k[i] + w*I;
155                 u[i] = pow(i*h, 4) * (1 - (i*h));
156                 if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
157                 else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i
                        ]), r);
158             }
159             k++;
160         }
161         r = 1;
162         std::cout << std::setw(3) << w << " | " << k << std::endl;
163     }
164     std::cout << "---\n";
165 
166 }
167 
168 // ???
169 void descentMethod_tableOutput(int n){
170     std::cout << "\033[1m" << "\033[3m" << "Descent Method\n" << "
           \033[0m";
171     double h = 1.0 / n;
172     double eps = pow(h, 3);
173     std::vector<double> y_k(n);
174     for(int i = 1; i < n-1; i++){
175         y_k[i] = f(i, h)*pow(h,2) /
176             (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
177     }
178     double r = 1;
179     std::vector<double> y_k_1(n);
180     std::vector<double> u(n);
```

```
181     while(fabs(r) > eps){
182         y_k = y_k_1;
183         for(int i = 1; i < n; i++){
184             y_k_1[i] = (a(i, h)*y_k[i-1] + a(i+1, h)*y_k[i+1] + f(i,
                    h)*pow(h, 2)) /
185                  (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
186             u[i] = pow(i*h, 4) * (1 - (i*h));
187             if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
188             else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r
                    );
189         }
190     }
191     //printTable(n, y_k_1, u);
192     std::cout << "---\n";
193 }
```