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 09-222
Романов И. И.
Проверил:
ассистент Глазырина О.В.

Казань, 2024 год

Содержание

1 Постановка задачи **3**

2 Ход работы **3**
2.1 Метод прогонки 3
2.2 Метод Зейделя 5
2.3 Метод верхней релаксации 7
2.4 Метод наискорейшего спуска 9

3 Выводы **11**

4 Список литературы **12**

5 Листинг программы **13**

1 Постановка задачи

Решить систему линейных алгебраических уравнений:

$$\begin{cases} (a_1 + a_2 + h^2 g_1) y_1 - a_2 y_2 = f_1 h^2, \\ \quad \dots \quad \dots \quad \dots \quad \dots \\ -a_i y_{i-1} + (a_i + a_{i+1} + h^2 g_i) y_i - a_{i+1} y_{i+1} = f_i h^2, \\ \quad \dots \quad \dots \quad \dots \quad \dots \\ (a_{n-1} + a_n + h^2 g_{n-1}) y_{n-1} - a_{n-1} y_{n-2} = f_{n-1} h^2. \end{cases} \tag{1}$$

Здесь $a_i = p(ih),\ g_i = q(ih),\ f_i = f(ih),\ f(x) = -(p(x)u'(x))' + q(x)u(x)$, $h = 1/n,\ p,\ q,\ u$ − заданные функции.

Данную систему решить методом прогоки и итерационными методами:

1. метод Зейделя.
2. метод нижней релаксации.
3. метод наискорейшего спуска.

Во всех итерационных методах вычисления продолжать до выполнения условия:

$$\max_{1 \le i \le n-1} \left| r_i^k \right| \le \varepsilon,$$

r − вектор невязки, ε − заданное число.

Исходные данные: $n = 10,\ n = 50,\ \varepsilon = h^3,\ u(x) = x^\alpha (1-x)^\beta$, $p(x) = 1 + x^\gamma,\ g(x) = x + 1,\ \alpha = 1,\ \beta = 3,\ \gamma = 2.$

Для сравнения результатов вычисления составим соответсвтенные таблицы и подведём выводы.

2 Ход работы

2.1 Метод прогонки

Метод прогонки состоит из двух этапов: прямой ход (определение прогоночных коэффициентов), обратных ход (вычисление неизвестных y_i).

Основным преимуществом является экономичность − максимально использование структуры исходной системы.

К недостаткам же можно отнести то, что с каждой итерацией накапливается ошибка округления.

Реализуем прямой ход метода и найдём прогоночные коэффициенты:

$$\begin{cases} \alpha_1 = \dfrac{a_1}{a_0 + a_1 + h^2 g_1}, \\ \alpha_{i+1} = \dfrac{a_{i+1}}{a_i + a_{i+1} + h^2 g_i - \alpha_i a_i}, \quad i = \overline{2, n-1}; \end{cases} \tag{2}$$

$$\begin{cases} \beta_1 = \dfrac{f_0 h^2}{a_0 + a_1 + h^2 g_1}, \\ \beta_{i+1} = \dfrac{f_i h^2 + \beta_i a_i}{a_i + a_{i+1} + h^2 g_i - \alpha_i a_i}, \quad i = \overline{2, n-1}; \end{cases} \tag{3}$$

Обратных ход метода:

$$\begin{cases} y_n = \beta_{n+1}; \\ y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = \overline{n-1, 0}; \end{cases} \tag{4}$$

Формулы (2-4) являются методом Гаусса, записанным применительно трёхдиагональной системы уравнений. Метод может быть реализован только в случае, когда в формулах (3) и (4) все знаменатели отличны от нуля, то есть условие выполняется, когда матрица системы (2) имеет диагональное препобладание.

Проделаем вычисления и составим таблицу для $n = 10$ (Таблица 1), для $n = 50$ (Таблица 2):

ih	y_i	$u(ih)$	$\lvert y_i - u(ih)\rvert$
0,0	0,000000	0,000000	0,000000
0,1	0,071338	0,072900	0,001562
0,2	0,099583	0,102400	0,002817
0,3	0,099491	0,102900	0,003409
0,4	0,083074	0,086400	0,003326
0,5	0,059738	0,062500	0,002762
0,6	0,036421	0,038400	0,001979
0,7	0,017694	0,018900	0,001206
0,8	0,005825	0,006400	0,000575
0,9	0,000812	0,000900	0,000088
1,0	0,000384	0,000000	0,000384

Таблица 1 - значения метода прогонки для $n = 10$

ih	y_i	$u(ih)$	$\lvert y_i - u(ih)\rvert$
0,0	0,000000	0,000000	0,000000
0,10	0,072724	0,072900	0,000176
0,20	0,083074	0,086400	0,003326
0,30	0,102460	0,102900	0,000440
0,40	0,085983	0,086400	0,000417
0,50	0,062186	0,062500	0,000314
0,60	0,038220	0,038400	0,000180
0,70	0,018838	0,018900	0,000062
0,80	0,006408	0,006400	0,000008
0,90	0,000925	0,000900	0,000025
1,0	0,000011	0,000000	0,000011

Таблица 2 - значения метода прогонки для $n = 50$

2.2 Метод Зейделя

Для больших систем вида $Ax = b$ предпочтительнее оказываются итерационные методы. Основная идея данных методов состоит в построении последовательности векторов x^k, $k = 1, 2, \ldots$, сходящейся к решению исходной системы. За приближенное решение принимается вектор x^k при достаточно большом k.

Будем считать, что все диагональные элементы матрицы из полной системы $Ax = b$ отличны от нуля. Представим эту систему, разрешая каждое уравнение относительно переменной, стоящей на главной диагонали:

$$x_i = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \quad i = \overline{1, n}. \tag{5}$$

Выберем некоторое начальное приближение $x^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T$. Построим последовательность векторов $x^1, x^2, \ldots$, определяя вектор x^{k+1} по уже найденному вектору x^k при помощи соотношения:

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^k - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}, \quad i = \overline{1, n}. \tag{6}$$

Формула (6) определяют итерационный метод решения системы (5), называемый методом Якоби или методом простой итерации.

Метод Якоби допускает естественную модификацию: при вычислении x_i^{k+1} будем использовать уже найденные компоненты вектора x^{k+1}, то есть x_1^{k+1}, x_2^{k+1}, $\ldots$, x_{i-1}^{k+1}. В результате приходим к итерационному методу Зейделя:

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}, \quad i = \overline{1, n}. \tag{7}$$

Запишем формулу (7) для нашей системы (1):

$$y_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_j}{a_i + a_{i+1} + h^2 g_i} y_j^{k+1} \; - \sum_{j=i+1}^{n} \frac{a_j}{a_i + a_{i+1} + h^2 g_i} y_j^k \; + \; \frac{f_i h^2}{a_i + a_{i+1} + h^2 g_i},$$

$$i = \overline{1, n-1}; \tag{8}$$

Вычисления продолжаем, пока не выполнится условие:

$$\max |r_i^k| \leq \varepsilon,$$

где r^k – вектор невязки для k-той итерации $r^k = Ay^k - f, \quad \varepsilon = h^3$.

Составим таблицы вычесленных результатов для $n = 10$, для $n = 50$, в которых будем сравнивать значения метода прогонки и метода Зейделя для точки i, $i = \overline{0, n-1}$, найдём модуль их разности и значение k, при котором была достигнута необходимая точность.

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,071338	0,068591	0,002747	19
1	0,099583	0,094675	0,004908	19
2	0,099491	0,093153	0,006339	19
3	0,083074	0,076065	0,007009	19
4	0,059738	0,052761	0,006977	19
5	0,036421	0,030057	0,006363	19
6	0,017694	0,012371	0,005322	19
7	0,005825	0,001806	0,004019	19
8	0,000812	-0,001799	0,002611	19
9	0,000384	-0,000850	0,001234	19

Таблица 3 - значения метода Зейделя для $n = 10$

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,018796	0,018695	0,000101	851
4	0,072724	0,072233	0,000491	851
9	0,102046	0,101135	0,000912	851
14	0,102460	0,101242	0,001218	851
19	0,085983	0,084599	0,001384	851
24	0,062186	0,060785	0,001402	851
29	0,038220	0,036937	0,001283	851
34	0,018838	0,017783	0,001055	851
39	0,006408	0,005658	0,000750	851
44	0,000925	0,000518	0,000408	851
49	0,000011	-0,000054	0,000065	851

Таблица 4 - значения метода Зейделя для $n = 50$

2.3 Метод верхней релаксации

Во многих ситуациях существенного ускорения сходимости можно добиться за счет введения так называемого итерационного параметра. Рассмотрим итерационный процесс:

$$x_i^{k+1} = (1-\omega)x_i^k + \omega\Bigg(-\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}x_j^{k+1} - \sum_{j=i+1}^{n}\frac{a_{ij}}{a_{ii}}x_j^k + \frac{b_i}{a_{ii}}\Bigg),$$

$$i = 1, 2, \ldots, n, \quad k = 0, 1, \ldots \ . \tag{9}$$

Этот метод называется методом релаксации – одним из наиболее эффективных и широко используемых итерационных методов для решения систем линейных алгебраических уравнений. Значение ω – называется релаксационным параметром. При $\omega = 1$ метод переходит в метод Зейделя. При $\omega \in (1, 2)$ – это метод верхней релаксации, при $\omega \in (0, 1)$ – метод нижней релаксации. Ясно, что по затратам памяти и объему вычислений на каждом шаге итераций метод релаксации не отличается от метода Зейделя.

Преобразуем формулу (2.3) относительно нашей системы:

$$y_i^{k+1} = (1-\omega)y_i^k + \omega\Bigg(-\sum_{j=1}^{i-1}\frac{a_j}{a_i + a_{i+1} + h^2 g_i}y_j^{k+1} - \sum_{j=i+1}^{n}\frac{a_j}{a_i + a_{i+1} + h^2 g_i}y_j^k + \frac{f_i h^2}{a_i + a_{i+1} + h^2 g_i}\Bigg),$$

$$i = \overline{1, n-1}; \tag{10}$$

Параметр ω следует выбирать так, чтобы метод релаксации сходился наиболее быстро. Нужно отметить, что оптимальный параметр для метода верхней релаксации лежит вблизи 1,8. Заполним таблицы, в которых приведём значения параметра ω и количство итераций k:

ω	k
1.1	15
1.2	12
1.3	9
1.4	7
1.5	6
1.6	8
1.7	11
1.8	18
1.9	34

Таблица 5 - значения ω и соответствующие значения k для $n = 10$

ω	k
1.02	818
1.14	639
1.30	452
1.38	375
1.54	245
1.62	189
1.78	93
1.86	51
1.94	102

Таблица 6 - значения ω и соответствующие значения k для $n = 50$

Для вычислений выберем $\omega = 1,86$. Составим таблицы результатов для $n = 10$, $n = 50$, в которых будем сравнивать значения метода прогонки и метода верхней релаксации для точки i, $i = \overline{0, n-1}$, найдём модуль их разности и значение k:

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,071338	0,071702	0,000364	18
1	0,099583	0,099998	0,000416	18
2	0,099491	0,099766	0,000275	18
3	0,083074	0,083252	0,000178	18
4	0,059738	0,059116	0,000622	18
5	0,036421	0,035560	0,000861	18
6	0,017694	0,016902	0,000792	18
7	0,005825	0,005242	0,000583	18
9	0,000384	0,000246	0,000138	18

Таблица 7 - значения метода верхней релаксации для $n = 10$

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,018796	0,018671	0,000125	51
9	0,102046	0,101108	0,000938	51
14	0,102460	0,101328	0,001132	51
24	0,062186	0,061123	0,001063	51
29	0,038220	0,037341	0,000880	51
34	0,018838	0,018184	0,000653	51
39	0,006408	0,005988	0,000420	51
44	0,000925	0,000719	0,000206	51

Таблица 8 - значения метода верхней релаксации для $n = 50$

2.4 Метод наискорейшего спуска

Существуют итерационные методы, позволяющие за счет некоторой дополнительной работы на каждом шаге итераций автоматически настраиваться на оптимальную скорость сходимости. К их числу относятся методы, основанные на замене системы (6) эквивалентной задачей минимизации некоторого функционала.

Опишем итерационный метод наискорейшего спуска. Будем двигаться из точки начального приближения x^0 в направлении наибыстрейшего убывания функционала F, то есть следующее приближение будем разыскивать так: $x^1 = x^0 - \tau \mathrm{grad}F(x^0)$. Формула:

$$F'_{x_i}(x) = 2\sum_{j=1}^{n} a_{ij}x_j - 2b_i; \tag{11}$$

, которая является производной функции $F(x)$ по переменной x_i, показывает, что $\mathrm{grad}F(x^0) = 2(Ax^0 - b)$. Вектор $r_0 = Ax^0 - b$ принято называть невязкой. Для сокращения записей удобно обозначить 2τ вновь через τ. Таким образом, $x^1 = x^0 - \tau r^0$.

Параметр τ выберем так, чтобы значение $F(x^1)$ было минимальным. Получим $F(x^1) = F(x^0 - \tau r^0) = F(x^0) - 2\tau(r^0, r^0) + \tau^2(Ar^0, r^0)$, следовательно, минимум $F(x^1)$ достигается при $\tau = \tau_* = \dfrac{(r^0, r^0)}{(Ar^0, r^0)}$.

Таким образом, мы пришли к следующему итерационнму методу:

$$x^{k+1} = x^k - \tau_* r^k, \quad r^k = Ax^k - b, \quad \tau_* = \frac{(r^k, r^k)}{(Ar^k, r^k)},\ k = 0, 1, \dots\ . \tag{12}$$

Метод (12) называют методом наискорейшего спуска. По сравнению с методом простой итерации этот метод требует на каждом шаге итераций проведения дополнительной работы по вычислению параметра τ_*. Вследствие этого происходит адаптация к оптимальной скорости сходимости.

Составим таблицы результатов для $n = 10$, $n = 50$, в которых будем сравнивать значения метода прогонки и метода верхней релаксации для точки i, $i = \overline{0, n-1}$, найдём модуль их разности и значение k:

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,071338	0,068795	0,002543	22
1	0,099583	0,095015	0,004568	22
2	0,099491	0,093633	0,005858	22
3	0,083074	0,076716	0,006359	22
4	0,059738	0,053639	0,006099	22
5	0,036421	0,031081	0,005340	22
6	0,017694	0,013500	0,004194	22
7	0,005825	0,002806	0,003019	22
8	0,000812	-0,000835	0,001647	22
9	0,000384	-0,000349	0,000733	22

Таблица 9 - значения метода наискорейшего спуска для $n = 10$

i	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0	0,018796	0,018713	0,000083	617
5	0,081561	0,081089	0,000472	617
10	0,104022	0,103249	0,000772	617
15	0,100175	0,099229	0,000946	617
20	0,081546	0,080564	0,000982	617
25	0,057220	0,056321	0,000900	617
30	0,033867	0,033129	0,000738	617
35	0,015762	0,015226	0,000535	617
40	0,004798	0,004465	0,000332	617
45	0,000495	0,000343	0,000152	617
46	0,000224	0,000108	0,000116	617
47	0,000080	-0,000005	0,000085	617
48	0,000022	-0,000033	0,000056	617
49	0,000011	-0,000015	0,000026	617

Таблица 10 - значения метода наискорейшего спуска для $n = 50$

3 Выводы

В процессе выполнения работы были изучены методы решения заданной системы линейных алгебраических уравнений вида:

$$\begin{cases} (a_1 + a_2 + h^2 g_1) y_1 - a_2 y_2 = f_1 h^2, \\ \quad \dots \quad \dots \quad \dots \quad \dots \\ -a_i y_{i-1} + (a_i + a_{i+1} + h^2 g_i) y_i - a_{i+1} y_{i+1} = f_i h^2, \\ \quad \dots \quad \dots \quad \dots \quad \dots \\ (a_{n-1} + a_n + h^2 g_{n-1}) y_{n-1} - a_{n-1} y_{n-2} = f_{n-1} h^2. \end{cases}$$

при помощи:

1. метод прогонки.
2. метод Зейделя.
3. метод нижней релаксации.
4. метод наискорейшего спуска.

После вычисления результатов решения системы линейных алгебраических уравнений можно сделать вывод, что наилучшим методом дря решения является метод верхней релаксации при итерационном параметре $\omega = 1,86$. Данный метод показывается наилучшие результаты вычисления корней системы за наименьшее количество итераций.

4 Список литературы

1. Глазырина Л.Л., Карчевский М.М. Численные методы: учебное пособие. — Казань: Казан. ун-т, 2012. — 122

2. Глазырина Л.Л.. Практикум по курсу «Численные методы». Решение систем линейных уравнений: учеб. пособие. — Казань: Изд-во Казан. ун-та, 2017. — 52 с.

5 Листинг программы

```cpp
#pragma once

#include <algorithm>
#include <cmath>
#include <iostream>
#include <vector>

using namespace std;

double f_x(double x) {
  return -pow(x, 5) + 22 * pow(x, 4) - 36 * pow(x, 3) + 28 * pow(x, 2) -
         19 * x + 6;
}

double u_x(double x) { return x * pow((1 - x), 3); }

double p_x(double x) { return 1 + pow(x, 2); }

double q_x(double x) { return 1 + x; }

vector<double> func_vec(int n) {
  double h = 1. / n;
  vector<double> b(n + 1, 0);
  for (int i = 1; i <= n; ++i) {
    b[i - 1] = f_x(i * h) * pow(h, 2);
  }
  return b;
}

vector<double> calculate_mtx_vec_mult(vector<vector<double>> &A,
                                      vector<double> &x) {
  int n = A.size();
  int m = x.size();
  vector<double> result(n, 0.0);

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      result[i] += A[i][j] * x[j];
    }
  }

```

```
42    return result;
43  }
44
45  vector<double> calculate_r(vector<vector<double>> &A, vector<double> &b,
46                             vector<double> &x) {
47    vector<double> Ax = calculate_mtx_vec_mult(A, x);
48    vector<double> r(Ax.size());
49
50    for (size_t i = 0; i < r.size(); ++i) {
51      r[i] = Ax[i] - b[i];
52    }
53
54    return r;
55  }
56
57  double calculate_error_dec(vector<double> &x, vector<vector<double>> &A,
58                             vector<double> &b) {
59    vector<double> r = calculate_r(A, b, x);
60    double max_err = 0.0;
61    for (int i = 0; i < r.size(); ++i) {
62      if (abs(r[i]) > max_err) max_err = abs(r[i]);
63    }
64
65    return max_err;
66  }
67  double calculate_error(vector<double> &new_x, vector<double> &old_x) {
68    double max_err = 0.0;
69    for (int i = 0; i < old_x.size(); ++i) {
70      double err = abs(abs(new_x[i]) - abs(old_x[i]));
71      if (err > max_err) max_err = err;
72    }
73
74    return max_err;
75  }
76
77  vector<vector<double>> create_matrix(int n) {
78    double h = 1. / n;
79
80    vector<vector<double>> matrix_res(n + 1, vector<double>(n + 1, 0.0));
81
82    for (int i = 0; i < n; ++i) {
83      if (i == 0) {
84        double b = (p_x((i + 1) * h) + p_x((i + 2) * h) +
```

```
                    (pow(h, 2) * q_x((i + 1) * h)));
      double c = -p_x((i + 2) * h);
      matrix_res[i][i] = b;
      matrix_res[i][i + 1] = c;
      continue;
    }

    if (i == (n - 1)) {
      double b = (p_x((i + 1) * h) + p_x((i + 2) * h) +
                  (pow(h, 2) * q_x((i + 1) * h)));
      double a = -p_x((i + 1) * h);
      matrix_res[i][i] = b;
      matrix_res[i][i - 1] = a;
      continue;
    }

    double a = -p_x((i + 1) * h);
    double b =
        (p_x((i + 1) * h) + p_x((i + 2) * h) + (pow(h, 2) * q_x((i + 1) *
            h)));
    double c = -p_x((i + 2) * h);

    matrix_res[i][i] = b;
    matrix_res[i][i + 1] = c;
    matrix_res[i][i - 1] = a;
  }

  return matrix_res;
}

//Алгорит Томаса
void progonka_method(vector<vector<double>> &A, vector<double> &x, int n,
                     vector<double> &b) {
  vector<double> alpha(n + 1), betta(n + 1);

  double h = 1.0 / n;

  // прямойход
  alpha[0] = A[0][1] / A[0][0];
  betta[0] = (b[0]) / A[0][0];

  for (int i = 1; i < n; ++i) {
    double del = 1.0 / (A[i][i] - alpha[i - 1] * A[i][i - 1]);
```

```
127     alpha[i] = A[i][i + 1] * del;
128     betta[i] = (-A[i][i - 1] * betta[i - 1] + b[i]) * del;
129   }
130
131   // обратныйход
132   x[n - 1] = betta[n - 1];
133   for (int i = n - 2; i >= 0; --i) {
134     x[i] = -alpha[i] * x[i + 1] + betta[i];
135   }
136
137   for (int i = 1; i <= n; ++i) {
138     printf(
139         "ih = %4.2lf | y_i = %9.6lf | u(ih) = %8.6lf | |y_i - u(ih)| = "
140         "%8.6lf\n",
141         i * h, x[i - 1], u_x(i * h), abs(x[i - 1] - u_x(i * h)));
142   }
143 }
144
145 double calculate_new_x(int i, vector<double> &x, int n,
146                        vector<vector<double>> &A, vector<double> &b) {
147   double h = 1.0 / n;
148   double sum = 0.0;
149
150   if (i > 0) {
151     for (int j = 0; j <= i - 1; ++j) {
152       sum += A[i][j] * x[j];
153     }
154   }
155
156   if (i < n - 1) {
157     for (int j = i + 1; j < n + 1; ++j) {
158       sum += A[i][j] * x[j];
159     }
160   }
161
162   return (b[i] - sum) * (1.0 / A[i][i]);
163 }
164
165 int Seidel_method(int n, vector<double> &x, vector<vector<double>> &A,
166                   vector<double> &b) {
167   double h = 1.0 / n;
168   int k = 0;
169   vector<double> new_x(n, 0.);
```

```
170   double error = 1.0;
171
172   while (error > 1.0 / pow(n, 3)) {
173     for (int i = 0; i < n; ++i) {
174       new_x[i] = calculate_new_x(i, new_x, n, A, b);
175     }
176
177     error = calculate_error_dec(new_x, A, b);
178
179     x = new_x;
180
181     k++;
182   }
183
184   return k;
185 }
186 double calculate_new_x_relax(int i, vector<double> &x, int n,
187                              vector<vector<double>> &A, double omega,
188                              vector<double> &b) {
189   double sum = 0.0;
190
191   if (i > 0) {
192     for (int j = 0; j <= i - 1; ++j) {
193       sum += A[i][j] * x[j];
194     }
195   }
196
197   if (i < n - 1) {
198     for (int j = i + 1; j < n + 1; ++j) {
199       sum += A[i][j] * x[j];
200     }
201   }
202
203   double new_x = (b[i] - sum) * (1.0 / A[i][i]);
204   return x[i] + omega * (new_x - x[i]);
205 }
206
207 int relax_top(int n, vector<double> &x, vector<vector<double>> &A,
208                 vector<double> &b) {
209   double h = 1.0 / n;
210   double omega = 1.86;
211     double error = 1.0;
212     int k = 0;
```

```
213     vector<double> new_x(n, 0.);
214
215     while (error > 1.0 / pow(n, 3)) {
216       for (int i = 0; i < n; ++i) {
217         new_x[i] = calculate_new_x_relax(i, new_x, n, A, omega, b);
218       }
219
220       error = calculate_error_dec(new_x, A, b);
221
222       x = new_x;
223
224       k++;
225     }
226
227     return k;
228 }
229 double calculate_tau(vector<double> &r, vector<double> &Ar) {
230   double a = 0.;
231   double b = 0.;
232   for (size_t i = 0; i < r.size(); ++i) {
233     a += r[i] * r[i];
234     b += Ar[i] * r[i];
235   }
236   if (abs(b) < 1e-10) {
237     return 0.0;
238   }
239   return a * (1.0 / b);
240 }
241
242 double calculate_new_x_spusk(int i, vector<double> &x, int n,
243                              vector<vector<double>> &A, vector<double> &b
                                    ) {
244   vector<double> r = calculate_r(A, b, x);
245   vector<double> Ar = calculate_mtx_vec_mult(A, r);
246
247   return x[i] - calculate_tau(r, Ar) * r[i];
248 }
249
250 int spusk(int n, vector<double> &x, vector<vector<double>> &A,
251           vector<double> &b) {
252   double h = 1.0 / n;
253   int k = 1;
254   vector<double> new_x(n, 0.);
```

```
  double error = 1.0;

  while (error >= 1.0 / pow(n, 3)) {
    for (int i = 0; i < n; ++i) {
      new_x[i] = calculate_new_x_spusk(i, new_x, n, A, b);
    }

    x = new_x;

    error = calculate_error_dec(new_x, A, b);

    k++;
  }

  return k;
}

void m_print(int k, vector<double> &y_i, vector<double> &y_ik) {
  for (int i = 0; i < y_i.size() - 1; ++i) {
    printf(
        "ih = %3d | y_i = %9.6lf | y_ik = %9.6lf | |y_i - y_ik| = %9.6lf
           | k = "
        "%d\n",
        i, y_i[i], y_ik[i], abs(y_i[i] - y_ik[i]), k);
  }
}
```

```
#include "header.hpp"

using namespace std;

int main() {
  int n = 50.0;

  std::vector<double> y_i_p(n + 1), y_i_s(n + 1), y_i_r(n + 1), y_i_sp(n
     + 1),
      b(n);
  vector<vector<double>> A = create_matrix(n);
  b = func_vec(n);
  printf("Прогонка\n");
  progonka_method(A, y_i_p, n, b);
  int k_s = Seidel_method(n, y_i_s, A, b);
  int k_r = relax_top(n, y_i_r, A, b);
```

```
16    int k_dec = spusk(n, y_i_sp, A, b);
17    printf("Прогонка - Зейдель\n");
18    m_print(k_s, y_i_p, y_i_s);
19    printf("Прогонка - Верхняярелаксация \n");
20    m_print(k_r, y_i_p, y_i_r);
21    printf("Прогонка - Спуск\n");
22    m_print(k_dec, y_i_p, y_i_sp);
23    return 0;
24 }
```