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.
Проверил:
ассистент Глазырина О.В.

Казань, 2024 год

Содержание

1 Постановка задачи **3**

2 Ход работы **4**
2.1 Метод прогонки 4
2.2 Метод Якоби 6
2.3 Метод верхней релаксации 8
2.4 Метод наискорейшего спуска 11

3 Выводы **13**

4 Список литературы **14**

5 Листинг программы **15**

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 прогонки и итерационными методами:

1. Якоби,
2. верхней релаксации,
3. 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: $n_1 = 10,\ n_2 = 20,\ \varepsilon = h^3,\ u(x) = x^\alpha (1-x)^\beta$, $p(x) = 1 + x^\gamma,\ g(x) = x + 1,\ \alpha = 4,\ \beta = 1,\ \gamma = 1.$

Для сравнения результатов вычисления составим таблицы и подведём выводы.

2 Ход работы

2.1 Метод прогонки

Метод прогонки является частным случаем метода Гаусса и применяется для решения систем линейных уравнений с трёхдиагональной матрицей. Метод прогонки состоит из двух этапов: прямой ход (определение прогоночных коэффициентов), обратный ход (вычисление неизвестных x_k).

Основными его преимуществами являются простота в реализации и то, что он максимально основан на структуре исходной системы.

Недостатком метода является то, что с каждой итерацией накапливается ошибка округления.

Запишем систему (1) в следующем виде:

$$\begin{cases} -b_1 y_1 + c_1 y_2 = f_1, \\ a_2 y_1 - b_2 y_2 + c_2 y_3 = f_2, \\ \dots \quad \dots \quad \dots \quad \dots, \\ a_i y_{i-1} - b_i y_i + c_i y_{i+1} = f_i, \\ \dots \quad \dots \quad \dots \quad \dots, \\ -a_{n-1} y_{n-1} + b_n y_n = f_n \end{cases} \tag{2}$$

где a, b, c - значения, полученные при заполнении системы (1).

Разрешим первое уравнение системы (2) относительно x_1 и получим:

$$y_1 = \alpha_2 y_2 + \beta_2, \quad \alpha_2 = \frac{c_1}{b_1}, \quad \beta_2 = -\frac{f_1}{b_1}.$$

Из i-того уравнения системы (2) получим:

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = \overline{1, n-1}.$$

Таким образом, прямой ход будет заключаться в нахождении прогоночных коэффициентов:

$$\alpha_{i+1} = \frac{c_i}{b_i - \alpha_i a_i}, \quad i = \overline{2, n-1}, \quad \alpha_1 = \frac{c_0}{b_0}. \tag{3}$$

$$\beta_{i+1} = \frac{\beta_i a_i - f_i}{b_i - \alpha_i a_i} \quad i = \overline{2, n-1}, \quad \beta_1 = -\frac{f_0}{b_0}. \tag{4}$$

Обратный ход будет заключаться в вычислении формул для нахождения неизветсных:

$$\begin{cases} y_i = \alpha_{i+1} y_{i+1} + \beta i + 1, i = \overline{n-1, 0}, \\ y_n = \beta_{n+1}. \end{cases} \tag{5}$$

Формулы (3-5) описывают метод Гаусса, то есть метод прогонки.

Таким образом, после проделанных вычислений составим таблицу для n = 10 и n = 20 с получившимися значениями, в которой первый столбец это номер итерации умноженный на число узлов разбиения. Второй столбец - решение метода. Третий - значения функции в точке. В четвертом столбце находится значения погрешности.

1. Метод прогонки для n = 10:

ih	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

2. Метод прогонки для n = 20:

ih	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

2.2 Метод Якоби

Для больших систем предпочтительнее оказываются итерационные методы. Основная идея этих методов состоит в построении последовательности векторов x_k, k = 1, 2, ..., сходящихся к решению системы $Ax = b$.

За приближенное решение принимается вектор x^k при достаточно большом k. При реализации итераицоннных методов, обычно, достаточно уметь вычилсять вектор Ax при любом заданном векторе x.

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

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

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

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

Формула (7) определяет итерационный метод решения системы (6), называемый методом Якоби или методом простой итерации.

Запишем этот метод для нашей системы:

$$y_i^{k+1} = \frac{a_i}{a_i + a_{i+1} + h^2 g_i} y_{i-1}^k + \frac{a_{i+1}}{a_i + a_{i+1} + h^2 g_i} y_{i+1}^k + \frac{f_i h^2}{a_i + a_{i+1} + h^2 g_i},$$
$$i = \overline{1, n-1}, \quad y_i^0 = 0, \quad y_0^k = y_n^k = 0 \quad \forall k \tag{8}$$

Вычисления продолжать до тех пор, пока не выполнится условие:

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

где r^k - вектор невязки для k-ой итерации $r^k = Ay^k - f, \qquad \varepsilon = h^3$.

1. Результаты метода Якоби для n = 10:

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.1	-0.000695158	-0.000700758	5.5999e-06	145
0.2	-0.00039375	-0.000404145	1.03947e-05	145
0.3	0.00300748	0.00299397	1.35191e-05	145
0.4	0.0116375	0.0116219	1.55989e-05	145
0.5	0.0264841	0.0264685	1.55772e-05	145
0.6	0.0462174	0.0462028	1.46068e-05	145
0.7	0.0660077	0.0659958	1.18485e-05	145
0.8	0.0763388	0.0763303	8.51864e-06	145
0.9	0.0618207	0.0618164	4.28485e-06	145

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

2. Для n = 20:

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.05	-7.79134e-05	-7.83545e-05	4.41101e-07	733
0.1	-8.5795e-05	-8.66476e-05	8.52527e-07	733
0.15	0.000152985	0.000151761	1.22406e-06	733
0.2	0.000889159	0.000887605	1.55335e-06	733
0.25	0.00241095	0.00240912	1.8293e-06	733
0.3	0.00500727	0.00500522	2.05459e-06	733
0.35	0.00893085	0.00892863	2.21834e-06	733
0.4	0.0143611	0.0143588	2.32761e-06	733
0.45	0.0213673	0.0213649	2.37278e-06	733
0.5	0.0298709	0.0298686	2.36437e-06	733
0.55	0.039609	0.0396067	2.29488e-06	733
0.6	0.0500966	0.0500944	2.17716e-06	733
0.65	0.0605893	0.0605873	2.00646e-06	733
0.7	0.0700467	0.0700449	1.79667e-06	733
0.75	0.0770939	0.0770924	1.54618e-06	733
0.8	0.0799855	0.0799842	1.26863e-06	733
0.85	0.0765672	0.0765662	9.65576e-07	733
0.9	0.064239	0.0642384	6.49247e-07	733
0.95	0.0399178	0.0399175	3.24129e-07	733

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

2.3 Метод верхней релаксации

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

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

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

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. Мы исследуем сходимость метода релаксации в случае, когда матрица A симметрична и положительно определена. С этой целью перепишем его в матричном виде. Обозначим через L нижнюю треугольную матрицу с нулевой главной диагональю; элементы, стоящие под главной диагональю матрицы L, соответствующими элементами матрицы . Через D, обозначим диагональную матрицу, на диагонали которой стоят диагональные элементы матрицы A. Понятно, что $A = L + D + L^T$. Нетрудно убедиться, что равенство (9) с учетом введенных обозначений принимает вид:

$$Dx^{k+1} = (1-\omega)Dx^k + \omega(-Lx^{k+1} - L^Tx^k + b).$$

После элементарных преобразований получим, что

$$B\frac{x^{k+1}-x^k}{\omega} + Ax^k = b, \quad \text{где } B = D + \omega L \tag{10}$$

Естественно параметр ω следует выбирать так, чтобы метод релаксации сходился наиболее быстро. В нашем случае выберем $\omega \in (1,2)$ и заполним таблицу, в которой первая строка - это релаксационный параметр, вторая - кол-во итераций, потребовавшихся для достижения заданной точности.

1. Результат метода верхней релаксации при n = 10:

w	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
k	29	28	25	21	16	20	28	39	74

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

При n = 10 погрешность меньше всего при $\omega = 1.5$. Сравним значения при данном ω с методом прогонки и построим таблицу:

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.1	-0.000695158	-0.00869934	0.00800418	16
0.2	-0.00039375	-0.0158154	0.0154216	16
0.3	0.00300748	-0.0193951	0.0224026	16
0.4	0.0116375	-0.0174571	0.0290946	16
0.5	0.0264841	-0.00912885	0.0356129	16
0.6	0.0462174	0.00415891	0.0420585	16
0.7	0.0660077	0.0174865	0.0485212	16
0.8	0.0763388	0.021256	0.0550829	16
0.9	0.0618207	0	0.0618207	16

Таблица 8 - таблица значений метода верхней релаксации при $\omega = 1.5$ и n = 10

2. Повторим дейстия при n = 20:

w	1.05	1.15	1.25	1.35	1.45	1.55	1.65	1.75	1.85	1.95
k	254	214	179	148	120	93	66	46	71	218

Таблица 9 - таблица значений для формулы метода Релаксации при n = 20

При n = 20 погрешность меньше всего при $\omega = 1.75$. Сравним значения при данном ω с методом прогонки:

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.05	-7.79134e-05	-0.00258756	0.00250965	46
0.1	-8.5795e-05	-0.0049969	0.00491111	46
0.15	0.000152985	-0.00706679	0.00721977	46
0.2	0.000889159	-0.00856025	0.00944941	46
0.25	0.00241095	-0.00920148	0.0116124	46
0.3	0.00500727	-0.00871282	0.0137201	46
0.35	0.00893085	-0.00685186	0.0157827	46
0.4	0.0143611	-0.00344859	0.0178097	46
0.45	0.0213673	0.00155737	0.0198099	46
0.5	0.0298709	0.00807957	0.0217914	46
0.55	0.039609	0.0158474	0.0237616	46
0.6	0.0500966	0.0243689	0.0257276	46
0.65	0.0605893	0.0328928	0.0276966	46
0.7	0.0700467	0.0403719	0.0296748	46
0.75	0.0770939	0.0454254	0.0316686	46
0.8	0.0799855	0.0463016	0.0336839	46
0.85	0.0765672	0.0408404	0.0357268	46
0.9	0.064239	0.0264361	0.0378029	46
0.95	0.0399178	0	0.0399178	46

Таблица 10 - таблица значений метода верхней релаксации при $\omega = 1.75$ и n = 20

2.4 Метод наискорейшего спуска

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

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}, \tag{11}$$

показывает, что $gradF(x^k) = 2(Ax^0 - b)$. 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 выеберем 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 $\tau = \tau_* = \frac{(r^0, r^0)}{(Ar^0, r^0)}$.

Таким образом, мы пришли к следующему итерационному методу:

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

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.

Первый столбец таблицы метода наискорейшего спуска - это номер итерации, умноженный на число узлов разбиения, второй - значения, полученные методом прогонки. Третий - значения, полученные методом наискорейшего спуска. В четвертом столбце находится значение погршености. В пятом столбце - кол-во итераций, потребовавшихся для достижения заданной точности.

1. Результат метода наискорейшего спуска при n = 10:

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.1	-0.000695158	-0.00724094	0.00654579	13
0.2	-0.00039375	-0.0133147	0.012921	13
0.3	0.00300748	-0.0163127	0.0193201	13
0.4	0.0116375	-0.0144059	0.0260434	13
0.5	0.0264841	-0.00620665	0.0326907	13
0.6	0.0462174	0.00622007	0.0399974	13
0.7	0.0660077	0.0192136	0.0467941	13
0.8	0.0763388	0.0218845	0.0544543	13
0.9	0.0618207	0	0.0618207	13

Таблица 11 - таблица значений для формулы метода наискорейшего спуска при n = 10

2. При n = 20

ih	y_i	y_i^k	$\lvert y_i - y_i^k \rvert$	k
0.05	-7.79134e-05	-0.003279	0.00320109	75
0.1	-8.5795e-05	-0.00634352	0.00625773	75
0.15	0.000152985	-0.00902848	0.00918146	75
0.2	0.000889159	-0.0110897	0.0119789	75
0.25	0.00241095	-0.0122406	0.0146515	75
0.3	0.00500727	-0.0121903	0.0171976	75
0.35	0.00893085	-0.0106814	0.0196123	75
0.4	0.0143611	-0.00753157	0.0218927	75
0.45	0.0213673	-0.00266403	0.0240313	75
0.5	0.0298709	0.0038321	0.0260388	75
0.55	0.039609	0.011714	0.027895	75
0.6	0.0500966	0.0204382	0.0296584	75
0.65	0.0605893	0.0293412	0.0312481	75
0.7	0.0700467	0.0372015	0.0328451	75
0.75	0.0770939	0.0428759	0.034218	75
0.8	0.0799855	0.0442423	0.0357432	75
0.85	0.0765672	0.0395435	0.0370237	75
0.9	0.064239	0.0257245	0.0385145	75
0.95	0.0399178	0	0.0399178	75

Таблица 12 - таблица значений для формулы метода наискорейшего спуска при n = 20

3 Выводы

В процессе выполнения данной работы были получены знания решения систем линейных алгебраических уравнений методом прогонки и итерационными методами: Якоби, верхней релаксации, наискорейшего спуска. Исходя из этого мы сделали вывод, что решить систему линейных алгебраических уравнений методом верхней релаксации является наиболее эффективным из всех методов, которые мы рассмотрели, так как за наименьшее количество разбиений мы получили более точный результат.

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 <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)
       ;
}
std::vector<double> SweepMethod_result(int n){
    double h = 1.0 / n;
    std::vector<double> alpha(n+1);
    std::vector<double> betta(n+1);
    for(int i = 2; i <= n; i++){
        alpha[i] = a(i,h)/
            ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i-1,
               h));
        betta[i] = (f(i-1, h)*pow(h,2) + betta[i-1] * a(i-1, h))/
            ((1 - alpha[i-1]) * a(i-1, h) + a(i, h) + pow(h, 2) * g(i-1,
               h));
    }
    std::vector<double> y(n+1);
    y[n] = 0;
    for(int i = n-1; i > 0; i--){
        y[i] = alpha[i+1] * y[i+1] + betta[i+1];
    }
    return y;
}
void SweepMethod_tableOutput(int n){
    std::cout << "\033[1m" << "\033[3m" << "Sweep Method\n" << "\033[0m";
        std::cout << std::setw(4) << "i*h" << " | " << std::setw(12) << "
           yi" << " | " << std::setw(12) << " ui"  << " | " << std::setw
           (12) << "|yi - ui|" << std::endl;
```

```
38      double h = 1.0 / n;
39
40      std::vector<double> y = SweepMethod_result(n);
41      for(int i = 1; i < n; i++){
42          double ui = pow(i*h, 4) * (1 - (i*h));;
43          double yi = y[i];
44          std::cout << std::setw(4) << i*h << " | " << std::setw(12) << yi
                << " | "
45              << std::setw(12) << ui << " | " << std::setw(12) << abs(yi -
                    ui) << std::endl;
46      }
47      std::cout << "---\n";
48  }
49
50  void JakobiMethod_tableOutput(int n){
51      std::cout << "\033[1m" << "\033[3m" << "Jakobi Method\n" << "\033[0m"
            ;
52      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;
53
54      double h = 1.0/n;
55      double eps = pow(h, 3);
56      std::vector<double> y_k(n);
57      for(int i = 1; i < n-1; i++){
58          y_k[i] = f(i, h)*pow(h,2) / (a(i, h) + a(i+1, h) + pow(h, 2)*g(i,
                h));
59      }
60      double r = 1;
61      std::vector<double> y_k_1(n);
62      int k_count = 0;
63      while(fabs(r) > eps){
64          y_k = y_k_1;
65          for(int i = 1; i < n; i++){
66              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))
67                  /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
68              if(i == 1) r = fabs((y_k_1[i] - y_k[i]) / y_k_1[i]);
69              else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r);
70          }
71          k_count++;
72      }
73      std::vector<double> yi = SweepMethod_result(n);
```

```
74      // Table output
75      for(int i = 1; i < n; i++){
76          std::cout  << std::setw(4) << i*h << " | " << std::setw(12) << yi
                [i] << " | "
77              << std::setw(12) << y_k_1[i] << " | " << std::setw(12) <<
                    fabs(yi[i] - y_k_1[i]) << " | " << k_count << std::endl;
78      }
79      std::cout << "---\n";
80  }
81
82  void upperRelaxationMethod_tableOutput(int n){
83      std::cout << "\033[1m" << "\033[3m" << "Upper relaxation Method\n" <<
            "\033[0m";
84      std::cout << std::setw(4) << "w" << " | " << "k\n";
85      double h = 1.0/n;
86
87      int min_k = pow(2, 10);
88      double w_min_k = 2e+10;
89      std::vector<double> y_k_lowest(n);
90      for(double w = 1.0 + h; w < 2; w += 0.1) {
91          double h = (1.0/n);
92          double eps = pow(h, 3);
93          std::vector<double> y_k(n);
94          for(int i = 1; i < n-1; i++) {
95              y_k[i] = f(i,h)*h*h / (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h
                    ));
96          }
97          double r = 1;
98          int k = 1;
99          std::vector<double> y_k_1(n);
100         while(fabs(r) > eps) {
101             y_k = y_k_1;
102             for(int i = 1; i < n-1; i++) {
103                 double I = (a(i, h)*y_k_1[i-1] + a(i+1, h)*y_k[i+1] + f(i
                        , h)*pow(h, 2))
104                     /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
105                 y_k_1[i] = (1-w) * y_k[i] + w*I ;
106                 if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
107                 else r = std::max(fabs((y_k_1[i] - y_k[i])/y_k_1[i]), r);
108             }
109             k++;
110         }
111         if(k < min_k){
```

```
112             y_k_lowest = y_k_1;
113             min_k = k;
114             w_min_k = w;
115         }
116         std::cout << std::setw(4) << w << " | " << k << std::endl;
117     }
118     std::vector<double> yi = SweepMethod_result(n);
119     std::cout << "w with minimal k = " << w_min_k << "\n";
120     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;
121     // Table output
122     for(int i = 1; i < n; i++){
123         std::cout  << std::setw(4) << i*h << " | " << std::setw(12) << yi
            [i] << " | "
124             << std::setw(12) << y_k_lowest[i] << " | " << std::setw(12)
                << fabs(yi[i] - y_k_lowest[i]) << " | " << min_k <<  std::
                endl;
125     }
126     std::cout << "---\n";
127 }
128
129 void descentMethod_tableOutput(int n){
130     std::cout << "\033[1m" << "\033[3m" << "Descent Method\n" << "\033[0m
        ";
131     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;
132     double h = 1.0 / n;
133     double eps = pow(h, 3);
134     std::vector<double> y_k(n);
135     for(int i = 1; i < n-1; i++){
136         y_k[i] = f(i, h)*pow(h,2) /
137             (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
138     }
139     double r = 1;
140     std::vector<double> r_k(n);
141     int k_count = 0;
142     while(fabs(r) > eps){
143         r = -1000000000;
144         for(int i = 1; i < n-1; i++){
145             r_k[i] = -a(i, h)*y_k[i-1] + (a(i, h) + a(i+1, h) + pow(h, 2)
                *g(i, h))*y_k[i] - a(i+1, h)*y_k[i+1] - f(i, h)*pow(h, 2);
```

```
            r = std::max(r, r_k[i]);
        }
        std::vector<double> Ar(n);
        for(int i = 1; i < n-1; i++){
            Ar[i] = -a(i, h)*r_k[i-1] + (a(i, h) + a(i+1, h) + pow(h, 2))
               *r_k[i] - a(i+1, h)*r_k[i+1];
        }
        double tauNumerator = 0;
        double tauDenominator = 0;
        for(int i = 0; i < n; i++){
            tauNumerator += pow(r_k[i], 2);
            tauDenominator += Ar[i]*r_k[i];
        }
        double tau = tauNumerator/tauDenominator;
        for(int i = 0; i < n; i++){
            y_k[i] = y_k[i] - tau*r_k[i];
        }
        k_count++;
    }
    std::vector<double> yi = SweepMethod_result(n);
    // Table output
    for(int i = 1; i < n; i++){
        std::cout  << std::setw(4) << i*h << " | " << std::setw(12) << yi
           [i] << " | "
            << std::setw(12) << y_k[i] << " | " << std::setw(12) << abs(
               yi[i] - y_k[i]) << " | " << k_count << std::endl;
    }
    std::cout << "---\n";
}
```