**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:

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

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. Зейделя,
3. релаксации,
4. 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, $\varepsilon$ – 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 $x_k$).

Основным его преимуществом является 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:

$$\begin{cases} -b_1y_1 + c_1y_2 = f_1, \\ a_2y_1 - b_2y_2 + c_2y_3 = f_2, \\ \dots \quad \dots \quad \dots \quad \dots, \\ a_iy_{i-1} - b_iy_i + c_iy_{i+1} = f_i, \\ \dots \quad \dots \quad \dots \quad \dots, \\ -a_{n-1}y_{n-1} + b_ny_n = f_n \end{cases} \tag{2}$$

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

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

$$y_1 = \alpha_2y_2 + \beta_2, \quad \alpha_2 = \frac{c_1}{b_1}, \quad \beta_2 = -\frac{f_1}{b_1}.$$

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:

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

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

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 $= 1,\ 2,\ \ldots,$ 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:

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

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

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

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

Запишем этот метод для нашей системы:

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

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:

| $ih$ | $y_i$ | $u(ih)$ | $\|y_i - u(ih)\|$ |
|---|---|---|---|
| 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

2. Для n = 20:

| $ih$ | $y_i$ | $u(ih)$ | $\|y_i - u(ih)\|$ |
|---|---|---|---|
| 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

## 2.3 Метод верхней релаксации

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

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

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

Этот метод называется методом релаксации — одним из наиболее эффективных и широко используемых итерационных методов для решения систем линейных алгебраических уравнений. Значение $\omega$ - 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 А 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 А. Понятно, что $A = L + D + L^T$. Нетрудно убедиться, что равенств0 (9) с учетом введенных обозначений принимает вид:

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

После элементарных преобразований получим, что

$$B\frac{x^{k+1} - x^k}{\omega} + Ax^k = b, \tag{10}$$

где $B = D + \omega L$

Естественно параметр $\omega$ следует выбирать так, чтобы метод релаксации сходился наиболее быстро. В нашем случае выберем $\omega \in 1, 2$ и заполним таблицу, в которой первый столбец - это релаксационный параметр, второй - 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 $\omega$ с методом прогонки:

| $ih$ | $y_i$ | $u(ih)$ | $\lvert y_i - u(ih)\rvert$ | $k$ |
|---|---|---|---|---|
| 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. Результат метода верхней релаксации 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 $\omega$ с методом прогонки:

| $ih$ | $y_i$ | $u(ih)$ | $\|y_i - u(ih)\|$ | $k$ |
|---|---|---|---|---|
| 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 $2\tau$ вновь через $\tau$. Таким образом, $x^1 = x^0 - \tau r^0$.

Параметр $\tau$ 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 (11) называют методом наискорейшего спуска. По сравнению с методом простой итерации этот метод требует на каждом шаге итераций проведения дополнительной работы по вычислению параметра $\tau_*$. Вследствие этого происходит адаптация к оптимальной скорости сходимости.

Первый столбец таблицы метода наискорейшего спуска - это номер итерации, 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:

| $ih$ | $y_i$ | $u(ih)$ | $\lvert y_i - u(ih)\rvert$ | $k$ |
|---|---|---|---|---|
| 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

| $ih$ | $y_i$ | $u(ih)$ | $\|y_i - u(ih)\|$ | |
|---|---|---|---|---|
| 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);
}
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);
    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;
    double h = 1.0 / n;

    std::vector<double> y = SweepMethod_result(n);
    for(int i = 1; i < n; i++){
        double ui = pow(i*h, 4) * (1 - (i*h));;
        double yi = y[i];
        std::cout << std::setw(4) << i*h << " | " << std::setw(12) << yi << " | "
             << std::setw(12) << ui << " | " << std::setw(12) << abs(yi - ui) << std::endl;
    }
    std::cout << "---\n";
}

void JakobiMethod_tableOutput(int n){
    std::cout << "\033[1m" << "\033[3m" << "Jakobi Method\n" << "\033[0m";
    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;

    double h = 1.0/n;
    double eps = pow(h, 3);
    std::vector<double> y_k(n);
```

```
    for(int i = 1; i < n-1; i++){
        y_k[i] = f(i, h)*pow(h,2) / (a(i, h) + a(i+1, h) + pow(h, 2)*g(i,
            h));
    }
    double r = 1;
    std::vector<double> y_k_1(n);
    int k_count = 0;
    while(fabs(r) > eps){
        y_k = y_k_1;
        for(int i = 1; i < n; i++){
            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))
                /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
            if(i == 1) r = fabs((y_k_1[i] - y_k[i]) / y_k_1[i]);
            else r = std::max(fabs((y_k_1[i] - y_k[i]) / y_k_1[i]), r);
        }
        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_1[i] << " | " << std::setw(12) <<
                fabs(yi[i] - y_k_1[i]) << " | " << k_count << std::endl;
    }
    std::cout << "---\n";
}

void upperRelaxationMethod_tableOutput(int n){
    std::cout << "\033[1m" << "\033[3m" << "Upper relaxation Method\n" <<
        "\033[0m";
    std::cout << std::setw(4) << "w" << " | " << "k\n";
    double h = 1.0/n;

    int min_k = pow(2, 10);
    double w_min_k = 2e+10;
    std::vector<double> y_k_lowest(n);
    for(double w = 1.0 + h; w < 2; w += 0.1) {
        double h = (1.0/n);
        double eps = pow(h, 3);
        std::vector<double> y_k(n);
        for(int i = 1; i < n-1; i++) {
```

```
            y_k[i] = f(i,h)*h*h / (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
        }
        double r = 1;
        int k = 1;
        std::vector<double> y_k_1(n);
        while(fabs(r) > eps) {
            y_k = y_k_1;
            for(int i = 1; i < n-1; i++) {
                double current_el = (a(i, h)*y_k_1[i-1] + a(i+1, h)*y_k[i+1] + f(i, h)*pow(h, 2))
                    /(a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
                y_k_1[i] = w * current_el + (1-w) * y_k[i];
                if(i == 1) r = fabs((y_k_1[i] - y_k[i])/y_k_1[i]);
                else r = std::max(fabs((y_k_1[i] - y_k[i])/y_k_1[i]), r);
            }
            k++;
        }
        if(k < min_k){
            y_k_lowest = y_k_1;
            min_k = k;
            w_min_k = w;
        }
        std::cout << std::setw(4) << w << " & " << k << " \\hline" << std::endl;
    }
    std::vector<double> yi = SweepMethod_result(n);
    std::cout << "w with minimal k = " << w_min_k << "\n";
    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;
    // 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_lowest[i] << " & " << std::setw(12)
            << fabs(yi[i] - y_k_lowest[i]) << " & " << min_k << " \\hline" <<  std::endl;
    }
    std::cout << "---\n";
}

void descentMethod_tableOutput(int n){
```

```cpp
    std::cout << "\033[1m" << "\033[3m" << "Descent Method\n" << "\033[0m
        ";
    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;
    double h = 1.0 / n;
    double eps = pow(h, 3);
    std::vector<double> y_k(n);
    for(int i = 1; i < n-1; i++){
        y_k[i] = f(i, h)*pow(h,2) /
            (a(i, h) + a(i+1, h) + pow(h, 2)*g(i, h));
    }
    double r = 1;
    std::vector<double> r_k(n);
    int k_count = 0;
    while(fabs(r) > eps){
        r = -1000000000;
        for(int i = 1; i < n-1; i++){
            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 tau = 0;
        for(int i = 1; i < n; i++){
            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];
            }
            tau = tauNumerator/tauDenominator;
        }
        for(int i = 1; i < n; i++){
            y_k[i] = y_k[i] - tau*r_k[i];
        }
        k_count++;
    }
    std::vector<double> yi = SweepMethod_result(n);
```

```cpp
    // 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 << "\\\\ \\hline"<<
            std::endl;
    }
    std::cout << "---\n";
}
```