МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1_2

ОТЧЕТ

о выполненном задании

студента 203 учебной группы факультета ВМК МГУ

Кудисова Артёма Аркадьевича

гор. Москва

2020 год

Цель работы

Изучить классические итерационные методы (Зейделя и верхней релаксации), используемые для численного решения систем линейных алгебраических уравнений; изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра.

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Написать программу численного решения данной системы линейных алгебраических уравнений (n – параметр программы), использующую численный алгоритм итерационного метода Зейделя:

$$(D - A^{(-)})(x^{k+1} - x^k) + Ax^k = f,$$

где D, $A^{(-)}$ - соответственно диагональная и нижняя треугольные матрицы, k - номер текущей итерации;

в случае использования итерационного метода верхней релаксации итерационный процесс имеет следующий вид:

$$(D - \omega A^{(-)})(x^{k+1} - x^k)\omega^{-1} + Ax^k = f,$$

где ω - итерационный параметр (при $\omega = 1$ метод верхней релаксации переходит в метод Зейделя).

Предусмотреть возможность задания элементов матрицы системы и ее правой части как во входном файле данных, так и путем задания специальных формул.

Основные цели

1. Решить заданную СЛАУ итерационным методом Зейделя (или более общим методом верхней релаксации);
2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью;

3. Изучить скорость сходимости итераций к точному решению задачи (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра ω (в случае симметрической положительно определенной матрицы системы известно, что для сходимости итераций следует выбирать $0 < \omega < 2$; при $\omega = 1$ метод верхней релаксации совпадает с методом Зейделя);
4. Правильность решения СЛАУ подтвердить системой тестов (например, можно использовать ресурсы on-line системы http://www.wolframalpha.com, пакета Maple и т.п.).

Метод верхней релаксации

Рассмотрим СЛАУ с n неизвестными

$$Ax = f,\ A \in \mathbb{R}^{n\times n},\ x \in \mathbb{R}^{n\times 1},\ f \in \mathbb{R}^{n\times 1},\ |A| \neq 0$$

Запишем каноническую форму линейного одношагового итерационного метода решения СЛАУ

$$B_{k+1}\frac{x_{k+1}-x_k}{\tau_{k+1}} + Ax_k = f_k,\ |B_{k+1}| \neq 0, \tau_{k+1} > 0$$

Представим A как сумму 3 матриц

$$A = D + T_H + T_B,$$

где D - диагональная часть матрицы A, которая содержит элементы a_{ij}, стоящие на главной диагонали;

T_H - нижняя треугольная матрица $(T_H)_{ij} = \begin{cases} a_{ij},\ i > j \\ 0,\ i \leq j \end{cases}$

T_B - верхняя треугольная матрица $(T_B)_{ij} = \begin{cases} a_{ij},\ i < j \\ 0,\ i \geq j \end{cases}$

Суть метода верхней релаксации заключена в простой идее, что

$$B = D + \omega T_H,\ \tau = \omega > 0$$

В таком случае мы можем записать следующую рекуррентную формулу

$$(D + \omega T_H)(x_{k+1} - x_k)\omega^{-1} + Ax_k = f$$

или

$$(D\omega^{-1} + T_H)(x_{k+1} - x_k) + Ax_k = f$$

Раскрыв вторые скобки, получим

$$(D\omega^{-1} + T_H)x_{k+1} + ((1-\omega^{-1})D + T_B)x_k = f$$

Теперь запишем эту формулу построчно:

$$\begin{cases} a_{11}(\omega^{-1}x_1^{k+1} + (1-\omega^{-1})x_1^k) + a_{12}x_2^k + a_{13}x_3^k + ... + a_{1n}x_n^k & = & f_1 \\ a_{21}x_1^{k+1} + a_{22}(\omega^{-1}x_2^{k+1} + (1-\omega^{-1})x_2^k) + a_{23}x_3^k + ... + a_{2n}x_n^k & = & f_2 \\ a_{31}x_1^{k+1} + a_{32}x_2^{k+1} + a_{33}(\omega^{-1}x_3^{k+1} + (1-\omega^{-1})x_3^k) + ... + a_{3n}x_n^k & = & f_3 \\ ..... & & \\ a_{n1}x_1^{k+1} + a_{n2}x_2^{k+1} + a_{n3}x_3^{k+1} + ... + a_{nn}(\omega^{-1}x_n^{k+1} + (1-\omega^{-1})x_n^k) & = & f_n \end{cases}$$

Из этой системы мы можем легко расчитать все компоненты вектора x^{k+1}

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}[f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k],\ i = \overline{1,n}$$

Сходимость

Разберемся со сходимостью метода верхней релаксации.

1. Пусть A - самосопряженная положительно определенная матрица. В противном случае, домножим A и f слева на A^{-1}.

   Самосопряженность означает, что $T_H^* = T_B,\ T_B^* = T_H$. Следовательно

   $$(T_H x, x) = (x, T_H^* x) = (x, T_B x) = (T_B x, x)$$

2. Пусть $B - \frac{\tau}{2}A > 0$. Выясним, когда это возможно:
   $B - \frac{\tau}{2}A = (D + \omega T_H) - \frac{\omega}{2}(D + T_H + T_B) = (1 - \frac{\omega}{2})D + \frac{\omega}{2}(T_H - T_B)$
   Условие положительной определенности:

   $$((B - \frac{\tau}{2}A)x, x) = (1 - \frac{\omega}{2})(Dx, x) > 0$$

   Матрица A положительно определена, значит, все ее диагональные элементы строго больше нуля, следовательно, матрица D тоже положительно определена: $(Dx, x) > 0$

   Таким образом, $1 - \frac{\omega}{2} > 0$, а значит (учитывая то, что $\omega > 0$):

   $$0 < \omega < 2$$

Тогда по теореме Самарского мы можем утверждать, что при любом выборе нулевого приближения x_0 наш итерационный процесс сойдется к решению исходной системы с любой наперед заданной точностью.

Критерий остановки

Критерием остановки является значение невязки приблеженного решения

$$\psi_k = Ax_k - f$$

В ходе решения ψ_k легко считается и, что крайне важно, между невязкой и погрешностью решения $z_k = x_k - x$ существует прямая связь:

$$\psi_k = Ax_k - f = A(z_k + x) - f = Az_k$$
$$z_k = A^{-1}\psi_k$$

Тогда мы получаем следующие важные оценки:

$$||\psi_k|| \leq ||A|| \cdot ||z_k||$$
$$||z_k|| \leq ||A^{-1}|| \cdot ||\psi_k||$$

А значит, погрешность решения стремится к нулю тогда и только тогда, когда к нулю стремится невязка:

$$z_k \to 0 \ <=> \ \psi_k \to 0$$

Еще одним критерием остановки может служить расстояние между текущим решением и предыдущим - если оно не превышает некоторую величину Δ в течение t итераций, то это повод завершить вычисления, иначе существует риск крайне долгого выполнения программы

Описание программы

Программа написана на языке python с использованием библиотеки numpy. Данная библиотека предоставляет широкие возможности для работы с данными в матричном виде.

Структура:

1. def relax(prev_x: np.ndarray, n: int, matrix: np.ndarray, f: np.ndarray, w: float) -> np.ndarray

   Функция, выполяющая 1 итерацию метода верхней релаксации. Возравщает приближенное решение СЛАУ.

   Аргументы:

   - prev_x - предыдущее решение
   - n - количество неизвестных
   - matrix - матрица системы
   - f - правая часть системы
   - w - параметр ω

2. def get_equation() -> dict

   Эта функция отвечает за получение входных данных и их преобразование к нужному формату.

   Возвращается словарь, содержащий матрицу системы, значения уравнений системы, количество неизвестных, параметр ω и требуемую точность (невязку) решения.

3. def dif(first: np.ndarray, second: np.ndarray) -> float

   Принимает 2 вектора одинаковых размеров и возвращает расстояние между ними.

4. def main()

   Главная функция - запускает все остальные функции в необходимом порядке, осуществляет формирование входных аргументов, распоковку возвращаемых значений и вывод ответа.

Значениями параметров Δ и t являются 10^{-6} и 300 соответственно. То есть если в течение 300 итераций расстояние между текущим решением и предыдущим ни разу не было больше 10^{-6}, то программа прекращает дальнейшие вычисления.

Код программы

```
import numpy as np


def relax(prev_x: np.ndarray,
```

```
          n: int,
          matrix: np.ndarray,
          f: np.ndarray,
          w: float) -> np.ndarray:

    x = np.zeros(prev_x.shape, dtype=np.float64)
    for i in range(n):
        sum1 = sum(-x[j] * w * matrix[i][j] / matrix[i][i]
                   for j in range(i))
        sum2 = sum(-prev_x[j] * w * matrix[i][j] / matrix[i][i]
                   for j in range(i, n))
        x[i] = sum1 + sum2 + f[i] * w / matrix[i][i] + prev_x[i]
    return x


def get_equation() -> dict:
    ans = {"matrix": None, "f": None,
           "n": None, "res": None, "w": None}
    try:
        ans["res"] = float(input("First specify residual: "))
        ans["w"] = float(input("Enter the value of w: "))
        if ans["w"] <= 0 or ans["w"] >= 2:
            raise ValueError

        ch = input("\nWould you like to enter formulas for matrices "
                   "or input file?\n"
                   "Print 1 in the first case and 2 in the second: ")

        if ch == '1':
            n = int(input("\nNow specify the count of "
                          "unknown variables: "))
            m = int(input("Now enter the value of m: "))

            ans["f"] = np.array([m * n - x ** 3 for x in range(1, n + 1)],
                                dtype=np.float64)
            ans["matrix"] = np.array([[(i + j) / (m + n) if i == j
                                       else n + m * m + j / m + i / n
                                       for j in range(1, n + 1)]
                                      for i in range(1, n + 1)],
                                     dtype=np.float64)
            ans["n"] = n
        elif ch == '2':
            n = int(input("\nNow specify the count "
                          "of unknown variables: "))
            file = input("Enter file name: ")

            matrix = []
            f = []
```

```
            with open(file, "r") as file:
                for line in file:
                    nums = [float(x) for x in line.strip().split()]
                    if len(nums) != n + 1:
                        print("Wrong input")
                        return ans

                    matrix.append(nums[:n])
                    f.append(nums[n])

            ans["matrix"] = np.array(matrix, dtype=np.float64)
            ans["f"] = np.array([f], dtype=np.float64).T
            ans["n"] = n
        else:
            print("Wrong input")

    except (ValueError, ZeroDivisionError):
        print("Wrong input")
        return ans

    return ans


def dif(first: np.ndarray, second: np.ndarray) -> float:
    tmp = first - second
    return np.sqrt(np.sum(tmp * tmp))


def main():
    MAX_T = 300
    D = 1e-6

    ans = get_equation()

    matrix = ans["matrix"]
    f = ans["f"]
    n = ans["n"]
    res = ans["res"]
    w = ans["w"]

    if matrix is None:
        return

    if np.linalg.det(matrix) == 0:
        print("\nDeterminant is 0!")
        print("Violated the conditions of the problem")
        return
```

```
    f = matrix.T @ f
    matrix = matrix.T @ matrix

    t = 0
    iterations = 0

    x = np.zeros((n, 1), dtype=np.float64)
    residual = dif(matrix @ x, f)

    while residual > res and t < MAX_T:
        iterations += 1
        prev_x = x
        x = relax(prev_x, n, matrix, f, w)
        residual = dif(matrix @ x, f)

        if dif(prev_x, x) < D:
            t += 1
        else:
            t = 0

    print(f'\n{iterations} iterations')
    print(f'\nResidual is {residual:.6f}')
    print("\nAnswer is")
    for x_i in x:
        print(x_i[0])


if __name__ == '__main__':
    main()
```

Тестирование

Приложение 1 - 13, приложение 2 (1 - 4)
Для проверки решений использовался ресурс www.wolframalpha.com

1. $$\begin{cases} 3x_1 - 2x_2 + 2x_3 - 2x_4 = 8 \\ 2x_1 - x_2 + 2x_3 = 4 \\ 2x_1 + x_2 + 4x_3 + 8x_4 = -1 \\ x_1 + 3x_2 - 6x_3 + 2x_4 = 3 \end{cases}$$

   Точное решение: 2, -3, -1.5, 0.5

ω	Невязка <= 0.1	Кол-во итераций	x1	x2	x3	x4
0.1	0.096	94	2.3559941	-0.4332455	-0.4662729	-0.4303484
0.2	0.097	44	2.3544139	-0.4425112	-0.4699996	-0.4269204
0.3	0.086	28	2.3533919	-0.4530452	-0.4744812	-0.4231945
0.4	0.097	19	2.3508389	-0.4644969	-0.4789225	-0.4188322
0.5	0.095	14	2.3488498	-0.4776790	-0.4843293	-0.4140196
0.6	0.077	11	2.3476847	-0.4930012	-0.4908949	-0.4086621
0.7	0.053	9	2.3467140	-0.5108438	-0.4985599	-0.4024876
0.8	0.046	7	2.3442826	-0.5311071	-0.5069327	-0.3951990
0.9	0.053	5	2.3406638	-0.5544108	-0.5164637	-0.3866583
1.0	0.035	3	2.3402352	-0.5819269	-0.5284133	-0.3771323
1.1	0.054	4	2.3314108	-0.6205067	-0.5431380	-0.3627347
1.2	0.059	5	2.3313712	-0.6686247	-0.5635438	-0.3460717
1.3	0.087	6	2.3154703	-0.7314192	-0.5866158	-0.3220508
1.4	0.047	9	2.3048607	-0.8188907	-0.6229777	-0.2905884
1.5	0.054	12	2.2875971	-0.9421073	-0.6729029	-0.2458242
1.6	0.087	15	2.2609887	-1.1251494	-0.7469760	-0.1792956
1.7	0.083	22	2.2213390	-1.4441299	-0.8754187	-0.0638166
1.8	0.086	35	2.1296205	-2.0621583	-1.1235957	0.1609490
1.9	0.093	68	1.9587299	-3.3218528	-1.6297966	0.6151895

ω	Невязка <= 0.001	Кол-во итераций	x1	x2	x3	x4
0.1	0.001000	110923	2.0131453	-2.9064805	-1.4623958	0.4660636
0.2	0.001000	52027	2.0135302	-2.9037429	-1.4612955	0.4650709
0.3	0.001000	32408	2.0139554	-2.9007195	-1.4600805	0.4639745
0.4	0.001000	22612	2.0144216	-2.8974037	-1.4587480	0.4627723
0.5	0.001000	16753	2.0149159	-2.8938892	-1.4573357	0.4614981
0.6	0.001000	12871	2.0154106	-2.8903715	-1.4559223	0.4602230
0.7	0.001000	10128	2.0158556	-2.8872078	-1.4546515	0.4590766
0.8	0.001000	8110	2.0161481	-2.8851293	-1.4538173	0.4583242
0.9	0.001000	6590	2.0161236	-2.8853068	-1.4538903	0.4583907
1.0	0.000999	5432	2.0155704	-2.8892453	-1.4554756	0.4598220
1.1	0.001000	4538	2.0143732	-2.8977652	-1.4589029	0.4629154
1.2	0.001000	3833	2.0125648	-2.9106324	-1.4640777	0.4675858
1.3	0.000999	3250	2.0104131	-2.9259406	-1.4702335	0.4731414
1.4	0.001000	2738	2.0082291	-2.9414786	-1.4764814	0.4787798
1.5	0.001000	2263	2.0062267	-2.9557238	-1.4822091	0.4839488
1.6	0.001000	1804	2.0044949	-2.9680431	-1.4871623	0.4884187
1.7	0.000999	1346	2.0030368	-2.9784158	-1.4913327	0.4921822
1.8	0.000995	865	2.0018157	-2.9871025	-1.4948251	0.4953338
1.9	0.000992	425	1.9992064	-3.0056272	-1.5022519	0.5020285

2. $$\begin{cases} 2x_1 + 3x_2 + x_3 + 2x_4 = 4 \\ 4x_1 + 3x_2 + x_3 + x_4 = 5 \\ x_1 - 7x_2 - x_3 - 2x_4 = 7 \\ 2x_1 + 5x_2 + x_3 + x_4 = 1 \end{cases}$$

Точное решение: -3, -5, 39, -7

ω	Невязка <= 0.1	Кол-во итераций	x1	x2	x3	x4
0.1	0.099998	7218	0.9876784	-1.7926267	7.3336821	-0.1692205
0.2	0.099996	4419	0.7398157	-1.9913887	9.3034098	-0.5964475
0.3	0.099995	3454	0.4917709	-2.1903106	11.2745498	-1.0239258
0.4	0.099987	2944	0.2445230	-2.3886074	13.2393243	-1.4499694
0.5	0.099985	2612	-0.0004045	-2.5850551	15.1856311	-1.8719620
0.6	0.099995	2366	-0.2418759	-2.7787411	17.1044475	-2.2879532
0.7	0.099982	2168	-0.4800028	-2.9697574	18.9966569	-2.6981257
0.8	0.099970	1997	-0.7133604	-3.1569580	20.8509430	-3.1000379
0.9	0.099957	1842	-0.9414239	-3.3399214	22.6631374	-3.4927880
1.0	0.099950	1696	-1.1635500	-3.5181302	24.4281321	-3.8752747
1.1	0.099985	1554	-1.3788805	-3.6908920	26.1391151	-4.2460356
1.2	0.099979	1414	-1.5886381	-3.8591944	27.8057861	-4.6071467
1.3	0.099923	1273	-1.7926150	-4.0228715	29.4264930	-4.9582469
1.4	0.099955	1127	-1.9892163	-4.1806322	30.9885877	-5.2966390
1.5	0.099990	974	-2.1800043	-4.3337382	32.5044617	-5.6249736
1.6	0.099953	810	-2.3663074	-4.4832651	33.9846436	-5.9454916
1.7	0.099952	626	-2.5495149	-4.6303280	35.4401568	-6.2605684
1.8	0.099686	392	-2.7402771	-4.7835404	36.9554104	-6.5881880
1.9	0.089256	162	-4.1370893	-5.9111991	48.0307004	-8.9557262

ω	Невязка <= 0.001	Кол-во итераций	x1	x2	x3	x4
0.1	0.001000	166580	-2.9601224	-4.9679256	38.6833304	-6.9316908
0.2	0.001000	79905	-2.9626009	-4.9699131	38.7030266	-6.9359629
0.3	0.001000	50981	-2.9650822	-4.9719031	38.7227450	-6.9402391
0.4	0.001000	36490	-2.9675514	-4.9738833	38.7423662	-6.9444939
0.5	0.001000	27770	-2.9700036	-4.9758502	38.7618532	-6.9487189
0.6	0.001000	21931	-2.9724183	-4.9777871	38.7810411	-6.9528788
0.7	0.001000	17737	-2.9747964	-4.9796947	38.7999378	-6.9569751
0.8	0.001000	14569	-2.9771322	-4.9815684	38.8184980	-6.9609979
0.9	0.001000	12082	-2.9794123	-4.9833976	38.8366156	-6.9649245
1.0	0.001000	10069	-2.9816273	-4.9851747	38.8542164	-6.9687388
1.1	0.001000	8400	-2.9837917	-4.9869113	38.8714141	-6.9724653
1.2	0.001000	6985	-2.9858858	-4.9885915	38.8880533	-6.9760705
1.3	0.000999	5764	-2.9879292	-4.9902312	38.9042890	-6.9795876
1.4	0.000999	4690	-2.9899004	-4.9918130	38.9199511	-6.9829802
1.5	0.000998	3729	-2.9918127	-4.9933476	38.9351447	-6.9862709
1.6	0.000998	2851	-2.9936701	-4.9948384	38.9499024	-6.9894667
1.7	0.000999	2024	-2.9954975	-4.9963052	38.9644198	-6.9926095
1.8	0.000999	1177	-2.9973969	-4.9978305	38.9795077	-6.9958725
1.9	0.000941	765	-3.0050585	-5.0040111	39.0402721	-7.0088834

3. $$\begin{cases} x_1 - x_2 + x_3 - x_4 = 0 \\ 4x_1 + x_2 - x_4 = 0 \\ 2x_1 + x_2 - 2x_3 + x_4 = 0 \\ 5x_1 + x_2 - 4x_4 = 0 \end{cases}$$

Точное решение: 0, 0, 0, 0

Так как нулевым приближением решения x_0 является нулевой вектор, то решение находится сразу же при любом $\omega \in (0, 2)$.

4. $$\begin{cases} 2x_1 - x_2 + 3x_3 + 4x_4 &= 5 \\ 4x_1 - 2x_2 + 5x_3 + 6x_4 &= 7 \\ 6x_1 - 3x_2 + 7x_3 + 8x_4 &= 9 \\ 8x_1 - 4x_2 + 9x_3 + 10x_4 &= 11 \end{cases}$$

Решения не существует.

Вывод программы:

```
Determinant is 0!
Violated the conditions of the problem
```

5. $$\begin{cases} x_1 - 2x_2 + x_3 + x_4 &= 0 \\ 2x_1 + 2x_2 - x_3 - x_4 &= 0 \\ 3x_1 - x_2 - 2x_3 + x_4 &= 0 \\ 5x_1 + 2x_2 - x_3 + 9x_4 &= -10 \end{cases}$$

Точное решение: 0, -0.6, -0.2, -1

ω	Невязка <= 0.1	Кол-во итераций	x1	x2	x3	x4
0.1	0.099772	441	-0.0266889	-0.6055017	-0.2343196	-0.9870871
0.2	0.099090	214	-0.0264891	-0.6055557	-0.2338702	-0.9871952
0.3	0.098948	138	-0.0263150	-0.6056247	-0.2334339	-0.9872921
0.4	0.098666	100	-0.0259463	-0.6056628	-0.2327281	-0.9874844
0.5	0.099104	77	-0.0255621	-0.6057096	-0.2319773	-0.9876860
0.6	0.097613	62	-0.0244354	-0.6056011	-0.2302751	-0.9882469
0.7	0.097670	51	-0.0234101	-0.6055257	-0.2286776	-0.9887604
0.8	0.095139	43	-0.0214664	-0.6052398	-0.2259399	-0.9897163
0.9	0.099585	36	-0.0207119	-0.6052576	-0.2246099	-0.9901046
1.0	0.095201	31	-0.0177824	-0.6047302	-0.2206779	-0.9915337
1.1	0.085267	27	-0.0138353	-0.6039003	-0.2156251	-0.9934438
1.2	0.099455	22	-0.0133701	-0.6040705	-0.2144578	-0.9937077
1.3	0.097305	18	-0.0099026	-0.6033958	-0.2098810	-0.9953962
1.4	0.044729	14	-0.0007174	-0.6007813	-0.1994779	-0.9997447
1.5	0.067756	16	-0.0004938	-0.5991649	-0.2027572	-0.9996055
1.6	0.088596	20	0.0018475	-0.6008192	-0.1949244	-1.0011122
1.7	0.045601	26	-0.0109993	-0.6027347	-0.2135168	-0.9946696
1.8	0.087957	36	-0.0207518	-0.6042213	-0.2275673	-0.9897570
1.9	0.064895	80	-0.0162062	-0.6039717	-0.2201308	-0.9920784

ω	Невязка <= 0.001	Кол-во итераций	x1	x2	x3	x4
0.1	0.000996	1108	-0.0002665	-0.6000549	-0.2003426	-0.9998711
0.2	0.000986	530	-0.0002636	-0.6000553	-0.2003371	-0.9998726
0.3	0.000979	337	-0.0002605	-0.6000557	-0.2003309	-0.9998742
0.4	0.000984	240	-0.0002588	-0.6000565	-0.2003265	-0.9998752
0.5	0.000977	182	-0.0002519	-0.6000563	-0.2003151	-0.9998786
0.6	0.000981	143	-0.0002456	-0.6000563	-0.2003042	-0.9998819
0.7	0.000988	115	-0.0002368	-0.6000559	-0.2002901	-0.9998863
0.8	0.000983	94	-0.0002219	-0.6000542	-0.2002681	-0.9998937
0.9	0.000922	78	-0.0001918	-0.6000487	-0.2002278	-0.9999084
1.0	0.000967	64	-0.0001807	-0.6000481	-0.2002101	-0.9999140
1.1	0.000885	53	-0.0001436	-0.6000405	-0.2001621	-0.9999320
1.2	0.000816	43	-0.0001097	-0.6000334	-0.2001186	-0.9999484
1.3	0.000776	33	-0.0000789	-0.6000271	-0.2000788	-0.9999633
1.4	0.000890	23	0.0000544	-0.5999761	-0.1999574	-1.0000245
1.5	0.000826	25	0.0000728	-0.5999949	-0.1998816	-1.0000371
1.6	0.000622	34	0.0000609	-0.5999945	-0.1999041	-1.0000310
1.7	0.000941	44	0.0002141	-0.5999450	-0.1997405	-1.0001034
1.8	0.000871	70	-0.0001836	-0.6000350	-0.2002489	-0.9999088
1.9	0.000558	152	0.0001428	-0.5999667	-0.1998189	-1.0000702

6. $n = 50,\ m = 15$

$f_i = mn - i^3$

$A_{ij} = \begin{cases} (i+j)/(m+n),\ i \neq j \\ n + m^2 + \frac{j}{m} + \frac{i}{n},\ i = j \end{cases}$ где $i, j = \overline{1, n}$

ω	Невязка	Кол-во итераций
0.1	619398286.806012	11247
0.2	619398274.756953	12976
0.3	619398443.713319	17497
0.4	619398275.216625	23292
0.5	619398271.920883	30159
0.6	619398468.751245	38169
0.7	619398458.639943	47512
0.8	619398261.743096	58484
0.9	619398471.389885	71495
1.0	619398259.029672	87149
1.1	619398261.694976	106311
1.2	619398269.787557	130290
1.3	619398373.322092	161159
1.4	619398317.977665	202311
1.5	619398369.488518	259970
1.6	619398444.669815	346471
1.7	619398462.166267	490599
1.8	619398253.669367	778981
1.9	619398336.157083	1644176

В конечном счете изменения вектора решений x_k становились крайне малы и программа завершалась досрочно.

Вывод

В ходе работы был рассмотрен с теоретической точки зрения и реализован на практике метод верхней релаксации, а также определен критерий его остановки при достижении заданной точности.

На конкретных тестах была проверена работоспособность и корректность написанной программы.

Было выяснено, что большую роль при решении СЛАУ играет выбор параметра ω. Так при неподходящих значениях ω количество требуемых итераций для достижения заданной точности может быть на 2 (а в некоторых случаях и на 3[1]) порядка больше, чем при оптимальном выборе данного параметра. Подобное увеличение количества итераций является существенным, так как заметно увеличивается время работы программы (особенно если работать приходится с большими матрицами).

[1] - пример 2