МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «Введение в численные методы» Задание №1

Численные методы решения систем линейных алгебраических уравнений

Отчет о выполненном задании

студента 202 учебной группы факультета ВМК МГУ
Мартьянова Артема Олеговича

Москва
2022

Решение СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента

Постановка задачи

Программная реализация метода Гаусса и метода Гаусса с выбором главного элемента для решения системы линейных алгебраических уравнение порядка n (n- параметр программы). Вид системы: $Ax = f$.
Так же программа должна предосталвлять пользователю возможность ввода данных как из файла, так и с помощью специальных формул (способ задания начальных условий выбирается при вводе). Формулы имеют следующий вид:

$$A_{ij} = \begin{cases} \dfrac{i+j}{n+m}, i \neq j \\ \mathrm{n} + \mathrm{m}^2 + \frac{j}{m} + \frac{i}{n}, \ i = j \end{cases}$$

$$m = 15$$
$$n = 50$$
$$f_i = mn - i^3$$

Цели и задачи данной практической работы:

1. Решить СЛАУ, если она совместна.
2. Вычислить определитель матрицы A.
3. Найти обратную матрицу A^{-1}, если она существует.
4. Найти число обусловленноси матрицы. $M_A = ||A^{-1}|| * ||A||$, в случае если существует матрица A^{-1}.
5. Исследовать вопрос вычислительной устойчивости метода Гаусса при больших n.
6. Подтвердить корректность работы программы с использованием набора тестов

Краткое описание алгоритма

Для решения системы линейных алгебраических уравнений вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = f_1 \\ ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = f_n \end{cases}$$

с невырожденной матрицей $A = (a_{ij})$ мы воспользуемся простотой решения неоднородной системы в случае, когда матрица коэффициентов - единичная. То есть основная идея алгоритма- привести матрицу коэффициентов к единичной. Воспользоваться данным аспектом линейной алгебры мы имеем право, так как матрица A невырождена.
Рассмотрим первую итерацию нашего алгоритма. На остальные легко продолжается по индукции. Так же более детальное обоснование можно получить в источнике [1].
Не ограничивая общности рассуждения, можем считать, что первый элемент первой строки ненулевой. Поделим всю строку на него, таким образом получим $a_{11} = 1$. На каждой итерации будем домножать a_{11} на a_{i1} и вычитать из i-ой строки первую, тем самым обнулим весь первый столбец и получим систему вида:

$$\begin{cases} x_1 + \frac{a_{12}}{a_{a11}}x_2 + ... + \frac{a_{1n}}{a_{11}}x_n = \frac{f_1}{a_{11}} \\ 0x_1 + (a_{22} - \frac{a_{12}}{a_{a11}})x_2 + ... + (a_{2n} - \frac{a_{1n}}{a_{11}})x_n = f_2 - \frac{f_1}{a_{11}} \\ ... \\ 0x_1 + (a_{n2} - \frac{a_{12}}{a_{a11}})x_2 + ... + (a_{nn} - \frac{a_{1n}}{a_{11}})x_n = f_n - \frac{f_1}{a_{11}} \end{cases}$$

На следующей итерации проводим аналогичные действия с коэффициентом перед x_2 во второй строке и всеми строками ниже. И так далее.

Так мы приводим матрицу A к треугольной. Для приведения её к единичной, нужно совершить такие же действия относительно верхней половины матрицы. Данный процесс носит название: ”обратный ход”.

Таким образом, после приведения матрицы коэффициентов к единичной решением исходной системы будет являться свободный вектор- столбец b (вектор f после всех преобразований).

Обратную матрицу будем искать методом Гаусса-Жордана, который обоснован в источнике [2].

Число обусловленности

$$M_A = ||A|| * ||A^{-1}||$$

считаем по норме:

$$||A|| = \max_{1<=j<=n} \sum_{i=1}^{n} |a_{ij}|$$

При больших n будет накапливаться ошибка, связанная с неточностью вычислений на ЭВМ. Оценим погрешность суммарной ошибки.
Допустим вектор f был округлён до f^*, тогда наша система примет вид:
$Ax^* = f^*$
Решаем две системы (теоретическую и с явной погрешностью) и вычисляем:
$\delta f = f^* - f$
$\delta x = x^* - x$
Теперь мы можем оценить абсолютную погрешность по норме:

$$||x|| = \sum_{i=1}^{n} |x_i|$$

Метод Гаусса с выбором главного элемента имеет похожую конценпцию. Теперь будем проводить преобразования не для первого ненулевого элемента в строке, а для максимального. Так как при таком подходе мы будем оперировать числами по модулю не превосходящими 1, то и суммарная погрешность будет меньше.

Более детальные обоснования можно найти в источниках [1] и [3].

Описание программы

Код написан на языке python3.7 с использованием фреймворка NumPy. Программа состоит из одного файла gauss.py, в котором реализовано решение системы линейных алгебраических уравнений методом Гаусса и методом Гаусса с выбором главного элемента.
Код состоит из следующих функций:

1. `def det(A)`

   Данная функция возвращает определитель матрицы . Если он оказывается нулевой, то программа пишет об этом на стандартный поток и завершает исполнение.

2. `def reverse(A)`

   Данная функция возвращает обратную матрицу к матрице A

3. `def creat_matrix(n, m)`

   Данная функция возвращает матрицу, созданную по формуле, определяемую вариантом.

4. `def read_matrix(file_path)`

   Данная функция считывает матрицу коэффициентов и свободный вектор- столбец из файла.

5. `def gauss(A, f, mainElement = False)`

   Данная функция решает систему методом Гаусса и возвращает решение. Параметр $mainElement$ отвечает за выбор метода Гаусса. False - обычный метод Гаусса, True - метод Гаусса с выбором главного элемента.

6. `def cond_num(A)`

   Данная функция вычисляет и возвращает число обусловленности матрицы A.

7. `def main()`

   Основная функция программы.

Исходный код программы

```
import numpy as np

def det(A):
    ans = np.linalg.det(A)
    eps = 0.00001
    if abs(ans) < eps:
        ans = 0
    return ans

def reverse(A):
    return np.linalg.inv(A)

def creat_matrix(n = 50, m = 15):
    A = np.zeros((n, n), dtype=np.float32)
    f = np.zeros((n, 1), dtype=np.float32)
    for i in range(n):
        for j in range(n):
            if i != j:
                A[i][j] = (i + j + 2) / (m + n)
            elif i == j:
                A[i][j] = n + m ** 2 + (j + 1) / m + (i + 1) / n
    for i in range(n):
        f[i, 0] = m * n - i ** 3
    return A, f

def read_matrix(file_path):
    Af = np.loadtxt(file_path)
    return Af[:,:-1], Af[:,-10:]

def gauss(A, f, mainElement=False):
    A = A
    N = len(A)
    B = f
    for col in range(N):
        if mainElement:
            cr = None
            for r in range(col, len(A)):
                cr = r if cr is None or abs(A[r][col]) > abs(A[cr][col]) else cr
            A[cr], A[col] = A[col], A[cr]
            B[cr], B[col] = B[col], B[cr]
        if A[col][col] == 0:
            tmp = 0
            for i in range(col + 1, N):
                if A[i][col] != 0:
```

```
                    tmp = i
                    break
            if not tmp:
                print("           ")
                return None
            else:
                A[col] += A[tmp]
                f[col] += f[tmp]
        div = A[col][col]
        A[col] = [a / div for a in A[col]]  #
        B[col] /= div
        #                           ,
        for r in range(col + 1, len(A)):
            mul = -A[r][col]
            A[r] = [(a + k * mul) for a, k in zip(A[r], A[col])]
            B[r] += B[col] * mul
    X = [0 for b in B]
    for i in range(N - 1, -1, -1):
        X[i] = B[i] - sum(x * a for x, a in zip(X[(i + 1):], A[i][(i + 1):]))
    return X

def cond_num(A):
    norm = 0
    inv_norm = 0
    A_1 = reverse(A)
    n = A.shape[1]
    for i in range(n):
        if np.sum(np.abs(A[: , i ])) > norm:
            norm = np.sum(np.abs(A[: , i ]))
        if np.sum(np.abs(A_1[: , i ])) > inv_norm:
            inv_norm = np.sum(np.abs(A_1[: , i ]))
    return inv_norm * norm

def main():
    print("Choose the way of matrix generation:")
    print("0 - making using formula")
    print("1 - reading from file")
    flag = int(input())
    if (flag):
        print("Insert file path\n")
        path = input()
        A, f = read_matrix(path)
    else:
        print("Would you prefer to insert size and variable?")
        print("0 - yes")
        print("1 - no")
```

```
        flag = int(input())
        if (flag):
            print("Insert size and system variable:")
            n = int(input())
            m = int(input())
            A, f = creat_matrix(n, m)
        else:
            A, f = creat_matrix()
    #             ,            ,
    _det = det(A)
    if not _det:
        print("You have inserted matrix with zero determinant\n")
        return
    print("Choose solution method:")
    print("0 - Gauss Method")
    print("1 - Gauss Method with choosing dominant element")
    flag = int(input())
    if (flag):
        ans = gauss(A, f, True)
    else:
        ans = gauss(A, f, False)
    A_1 = reverse(A)
    print("Determinant of coefficient matrix: ", _det)
    print("Inversed coefficients matrix: ", A_1)
    print("Condition number: ", cond_num(A))
    print("System solution: ", ans)
    return

main()
```

Тестирование и результаты эксперимента

Тестирование программы производилось с использованием предоставленных тестов, тестов, сгенерированных с использованием готовых формул и тестов, сгенерированных самостоятельно.
Ниже приведены входные данные и результаты работы программы на этих данных.

Тестовая система №1 из варианта 1-12:

$$A = \begin{pmatrix} 2 & -2 & 0 & 1 \\ 2 & 3 & 1 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 3 & 1 & -1 \end{pmatrix} f = \begin{pmatrix} -3 \\ -6 \\ -0 \\ -2 \end{pmatrix}$$

Результаты работы теста №1:

$$x = \begin{pmatrix} -2.000 \\ 1.000 \\ 4.001 \\ 3.000 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 0.26 & 0.17 & 0.08 & -0.09 \\ -0.17 & -0.04 & 0.09 & 0.13 \\ 0.38 & -0.47 & -0.32 & 1.15 \\ 0.13 & -0.42 & 0.04 & 0.45 \end{pmatrix}$$

$$\det(A) = -53.001$$
$$M_A = 21.9622$$

Тестовая система №2 из варианта 1-12:

$$A = \begin{pmatrix} 1 & 3 & 2 & 1 \\ 2 & -1 & 3 & -2 \\ 3 & -5 & 4 & -3 \\ 1 & 17 & 4 & -23 \end{pmatrix} f = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Результаты работы теста №2

$$\det(A) = 0.0$$

Тестовая система №3 из варианта 1-12

$$A = \begin{pmatrix} 45 & -28 & 34 & -52 \\ 36 & -23 & 29 & -43 \\ 47 & -32 & 36 & -48 \\ 27 & -19 & 22 & -35 \end{pmatrix} f = \begin{pmatrix} 9 \\ 3 \\ -17 \\ 6 \end{pmatrix}$$

Результаты работы теста №3

$$x = \begin{pmatrix} -1.00005 \\ 2.00027 \\ -4.00008 \\ -3.00027 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 0.33 & 0.22 & 0.00 & -0.22 \\ 0.08 & -0.37 & -0.14 & 0.38 \\ -0.49 & 0.76 & 0.03 & -0.24 \\ -0.09 & -0.10 & 0.09 & -0.15 \end{pmatrix}$$

$$\det(A) = -1440.00005$$
$$M_A = 260.0777779$$

Сгенерированные по формуле тесты.

Тестовая система №4:

$$A = \begin{pmatrix} 0.667 & 5.50 \\ 5.0 & 1.333 \end{pmatrix} f = \begin{pmatrix} 2.0 \\ 1.0 \end{pmatrix}$$

Результаты работы теста №4

$$x = \begin{pmatrix} 0.1064 \\ 0.3507 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -0.05 & -0.21 \\ -0.19 & -0.03 \end{pmatrix}$$

n = 2
m = 1
$$\det(A) = -26.6111107$$
$$M_A = 1.626305$$

Тестовая система №5:

$$A = \begin{pmatrix} 0.222 & 21.7 & 21.95 & 22.2 & 22.45 \\ 21.65 & 0.444 & 22.15 & 22.4 & 22.65 \\ 21.85 & 22.1 & 0.6667 & 22.6 & 22.85 \\ 22.05 & 22.3 & 22.55 & 0.8889 & 23.05 \\ 22.25 & 22.5 & 22.75 & 23.0 & 1.1111 \end{pmatrix} f = \begin{pmatrix} 19 \\ 12 \\ -7 \\ -44 \\ -105 \end{pmatrix}$$

Результаты работы теста №5

$$x = \begin{pmatrix} -2.3295 \\ -1.9929 \\ -1.1100 \\ 0.5760 \\ 3.3114 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -0.04 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & -0.04 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & -0.04 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & -0.04 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.04 \end{pmatrix}$$

n = 5
m = 4
det$(A) = 19919364.5335$
M_A = 7.4871448

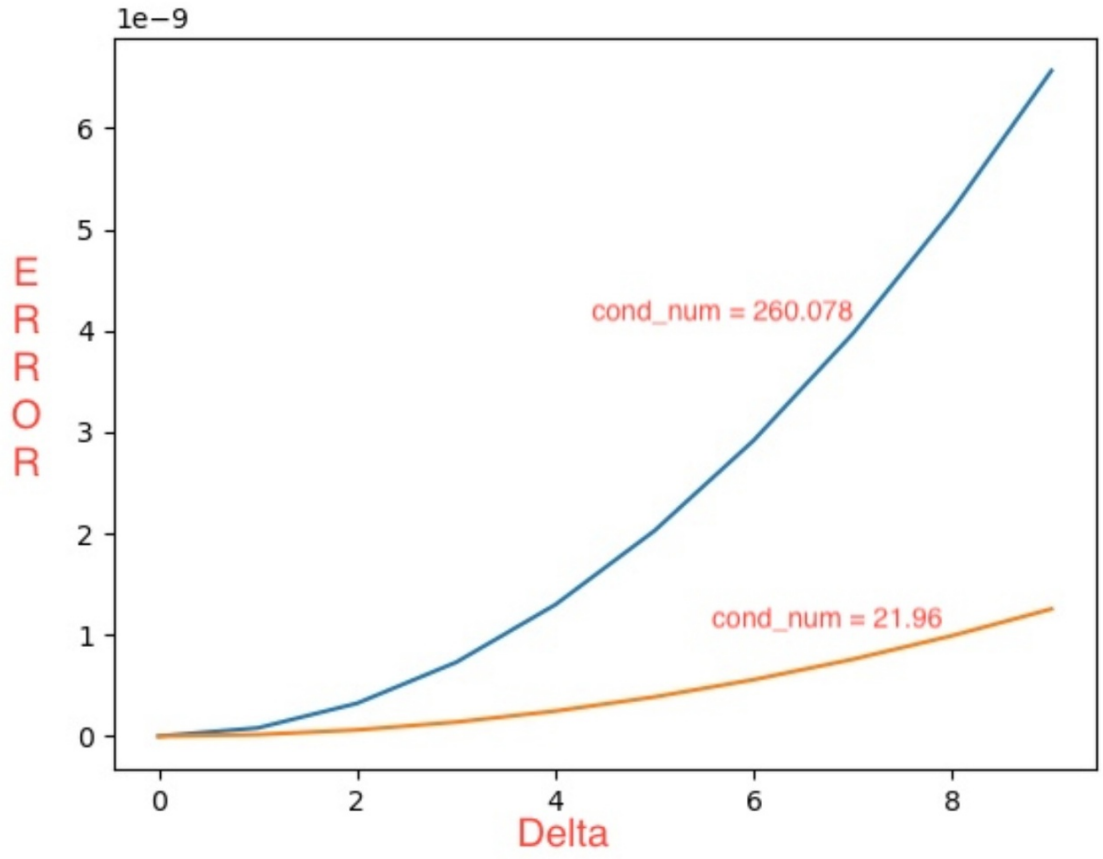
Тестовая система №6 (задана вариантом)

В силу размеров матрицы (50 на 50) для данного теста будет предоставлен лишь ответ, который не содержит обратную матрицу

Решение данной системы приложено в отдельном файле $solve\_slau$, так же в силу своих размеров.

n = 50
m = 15
det$(A) = -5.8929672e + 123$
M_A = 98.04656089

В процессе нашего исследование был проведён анализ устойчивоти метода Гаусса и метода Гаусса с выбором главного элемента
Коррекность решений была подтверждения посредством интернет-ресурса Wolframe. Приведём результаты нашего исследования в виде графика зависимости сдвига вектора от сдвига свободного вектора. Ошибка мерилась с помощью метрики Mean Squared Error или MSE.



Выводы

В ходе решения данной работы мы реализовали прямые методы решения систем линейных алгебраических уравнений - методы Гаусса и Гаусса с выбором главного элемента. Так же было произведено исследование устойчивости данных методов. Из результатов исследования можно сделать вывод, что данные методы хорошо работают на матрицах с ”небольшим”числом обусловленности.
В случае ”большой”обусловленности матрицы A мы получаем достаточно низкую точность вычислений, что, конечно, неприемлемо.

Решение СЛАУ итерационными методами. Метод верхней релаксации

Цели работы

Изучить один из итерацационных методов (метод верхней релаксации), используемый для решения систем линейных алгебраических уравнений, а так же скорость его сходимости в зависимости от итерационного параметра.

Постановка задачи

Необходимо написать программу, реализующую метод верхней релаксации для решения СЛАУ $Ax = f$. Элементы матрицы задаются методами, аналогичными описанным в предыдущей главе. Формулы для вычисления элементов матрицы имеют вид:

$$A_{ij} = \begin{cases} \mathrm{q}_m^{i+j} + 0.1 \cdot (j - i), i \neq j \\ (\mathrm{q}_m - 1)^{i+j}, i = j \end{cases}$$

$$q_m = 1.001 - 2 \cdot m \cdot 10^{-3}$$
$$m = 4$$
$$n = 100$$
$$f_i = n \cdot e^{\frac{x}{i}} \cdot cos(\frac{x}{i})$$

В ходе программы необходимо осуществить следующие пункты:

1. Решить заданную СЛАУ методом верхней релаксации.
2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной СЛАУ с заданной точностью.
3. Изучить скорость сходимости итераций к точному решению задачи в зависимости от итерационного параметра ω.
4. Подтвердить корректность работы программы с использованием набора тестов, сгенерированных самостоятельно.

Краткое описание алгоритма

Рассмотрим систему линейных алгебраических уравнения вида:

$$A_0 x = f_0$$

с невырожденной матрицей $A_0 = (a_{ij})$.
Следующие рассуждения приведены для случая, когда все собственные значения матрицы A больше нуля, лишь в таком случае мы можем сказать, что алгоритм работает корректно работу программы.
Так как гарантию корректной работы алгоритма верхней релаксации мы можем дать лишь для симметричних положительно определённых матриц, для начала умножим исходное уравнение на матрицу A_0^T, в результате получим систему, $A_0^T A_0 x = A_0^T f$ эквивалентную изначальной. Далее введем обозначения:

$$A = A_0^T A_0,\ f = A_0^T f_0$$

Далее выберем нулевой начальный вектор $x_1 = 0$, и будем на каждой итерации по данному x_i вычислять x_{i+1} с использованием следующей формулы:

$$x_{i+1} = x_i + \omega * (D + \omega A^-)(f - Ax_i)$$

Здесь D - диагональная и A^- нижнетреугольная матрицы, такие что
$A^+ = A - D - A^-$ - верхнетреугольная матрица.
В силу того, что выбранная матричная норма является согласованной с векторной, выполняется следующее неравенство:

$$||\delta x_i|| \leq |A^{-1}|| * ||\delta f_i||, \text{ где } \delta x_i = x_i - x,\ \delta f_i = Ax_i - f$$

В соответствии с этим неравенством, мы должны будем совершать итерации до тех пор, пока в правой части не получим величину необходимой точности ε.

Структура программы и спецификация функций

Код программы написан на языке программирования python3.7 с использованием фреймворка NumPy. Программа состоит из одного файла relax.py, реализующей метод верхней релаксации.
Программа состоит из следующий функций:

1. `def det(A)`

   Данная функция вычисляет определитель матрицы.

2. `def creat_matrix(n=100, m=4, x)`

   Данная функция генерирует матрицу и свободный вектор по формулам, которые заданы вариантом.

3. `def read_matrix(file_path)`

   Данная функция считывает матрицу и свободный вектор из файла.

4. `def up_relax(A, f, w=0.01, eps=0.001)`

   Данная функция находит решение системы линейных алгебраических уравнений методом верхней релаксации с точностью eps.

5. `def main()`

   Основная функция программы.

Исходный код программы

```
import numpy as np

def det(A):
    ans = np.linalg.det(A)
    eps = 0.00001
    if abs(ans) < eps:
        ans = 0
    return ans

#
def creat_matrix(x, n=100, m=4):
    A = np.zeros((n, n), dtype=np.float32)
    f = np.zeros((n, 1), dtype=np.float32)

    q = 1.001 - 2 * m * 0.001

    for i in range(n):
        for j in range(n):
            if i != j:
                A[i][j] = q ** (i+1 + j+1) + 0.1 * (j - i)
            elif i == j:
                A[i][j] = (q - 1) ** (i+1 + j+1)

    for i in range(n):
        f[i, 0] = n * np.exp(x / (i + 1)) * np.cos(x)
    return A, f

#
def read_matrix(file_path):
    Af = np.loadtxt(file_path)
    return Af[:,:-1], Af[:,-10:]

def upRelax(A, f, x_0, omega, eps):
    max_iter = 500000
    itr = 0
    tmp = x_0[:]
    residual = np.linalg.norm((A @ tmp) - f)
    for _ in range(max_iter):
        itr += 1
        for i in range(A.shape[0]):
            sigma = 0
            for j in range(A.shape[1]):
                if j != i:
                    sigma += A[i, j] * tmp[j]
            tmp[i] = (1 - omega) * tmp[i] + (omega / A[i, i]) * (f[i] - sigma)
```

```
        residual = np.linalg.norm((A @ tmp) - f)
        if (residual <= eps):
            break

    return tmp.tolist(), itr

def main():
    print("                  :")
    print("0 -            ")
    print("1 -          ")
    flag = int(input())
    if flag:
        print("            \n")
        path = input()
        A, f = read_matrix(path)
    else:
        print("                                        ?")
        print("0 -   ")
        print("1 -  ")
        flag = int(input())
        if (flag):
            print("                           :")
            n = int(input())
            m = int(input())
            print("         ")
            x = float(input())
            A, f = creat_matrix(x, n, m)
        else:
            print("         ")
            x = float(input())
            A, f = creat_matrix(x)

    _det = det(A)
    if not _det:
        print("                  ")
        return -1

    f = A.T @ f
    A = A.T @ A

    print("                     :")
    print("0 -   ")
    print("1 -  ")
    flag1 = int(input())
    if flag1:
        w = float(input())
```

```
    print("                  :")
    print("0 -   ")
    print("1 -  ")
    flag2 = int(input())
    if flag2:
        eps = float(input())

    if flag1 and flag2:
        x, iters = upRelax(A, f, w, eps)
    elif flag1:
        x, iters = upRelax(A, f, w)
    elif flag2:
        x, iters = upRelax(A, f, 0.01, eps)
    else:
        x, iters = upRelax(A, f)

    print("       :")
    print(*x)

    print("            = ", iters)
    return 0

#         -
main()
```

Тестирование и результаты эксперимента

Тестирование программы производилось с использованием предоставленных тестов, а также тестов, сгенерированных с использованием готовых формул. Ниже приведены входные данные и результаты работы программы на этих данных. Помимо непосредственно решения системы, программа выводит количество итераций, которые затратились для достижения нужной точности.

Тестовая система №1 из варианта 1-12:

$$A = \begin{pmatrix} 2 & -2 & 0 & -1 \\ 2 & 3 & 1 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 3 & 1 & -1 \end{pmatrix} f = \begin{pmatrix} -3 \\ -6 \\ -0 \\ -2 \end{pmatrix}$$

Результаты работы теста №1:

$$x = \begin{pmatrix} 0.1538 \\ -0.3845 \\ 7.0766 \\ 4.0768 \end{pmatrix}$$

ε = 0.0001
ω = 0.4 iterations: 522
ω = 0.7 iterations: 242
ω = 1.0 iterations: 132
ω = 1.3 iterations: 72
ω = 1.6 iterations: 25
ω = 1.9 iterations: 119

Тестовая система №2 из варианта 1-12:

$$A = \begin{pmatrix} 1 & 3 & 2 & 1 \\ 2 & -1 & 3 & -2 \\ 3 & -5 & 4 & -3 \\ 1 & 17 & 4 & -23 \end{pmatrix} f = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Результаты работы теста №2

$$\det(A) = 0.0$$

Тестовая система №3 из варианта 1-12:

$$A = \begin{pmatrix} 45 & -28 & 34 & -52 \\ 36 & -23 & 29 & -43 \\ 47 & -32 & 36 & -48 \\ 27 & -19 & 22 & -35 \end{pmatrix} f = \begin{pmatrix} 9 \\ 3 \\ -17 \\ 6 \end{pmatrix}$$

Результаты работы теста №3:

$$x = \begin{pmatrix} 0.9999 \\ 2.0000 \\ -3.9999 \\ -2.9999 \end{pmatrix}$$

ε = 0.0001
ω = 0.4 iterations: 96486
ω = 0.7 iterations: 45491
ω = 1.0 iterations: 24619
ω = 1.3 iterations: 11517
ω = 1.6 iterations: 12206
ω = 1.9 iterations: 48355

Теперь проверим нашу программу на матрицах, сгенерированных по формуле, например, в точке $x = 3.0$.

Тестовая система №4:

$$A = \begin{pmatrix} 3.996 & -0.897 & 1.196 \\ -1.097 & 15.968 & -0.895 \\ 0.796 & -1.095 & 63.808 \end{pmatrix} f = \begin{pmatrix} -9.224 \\ -3.393 \\ -2.431 \end{pmatrix}$$

Результаты работы программы

$$x = \begin{pmatrix} -2.38881 \\ -0.37746 \\ -0.01478 \end{pmatrix}$$

ε = 0.0001
n = 3
x = 3.0
ω = 0.4 iterations: 255678
ω = 0.7 iterations: 141542
ω = 1.0 iterations: 80222
ω = 1.3 iterations: 155254
ω = 1.6 iterations: 241000
ω = 1.9 iterations: 322511

Проверка правильности решений производилось с помощью метода подстановки в систему, а также интернет- ресурса Wolframe.В ходе работы мы считали количество итераций при разных ω на разных матрицах. Построим графики зависимостей количества итераций от ω для всех тестов.

График зависимости числа итераций от итерационного параметра ω для матрицы 1-12 №1

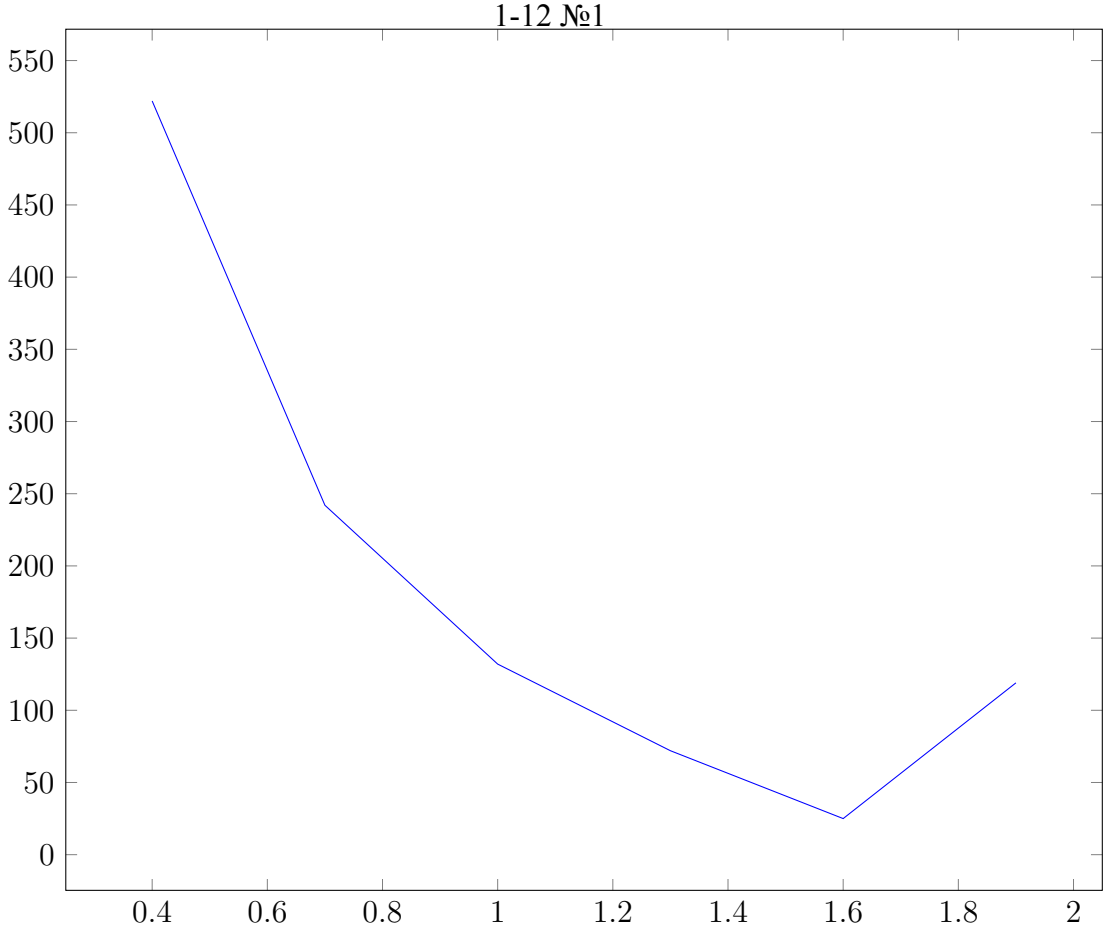


График зависимости числа итераций от итерационного параметра ω для матрицы 1-12 №3

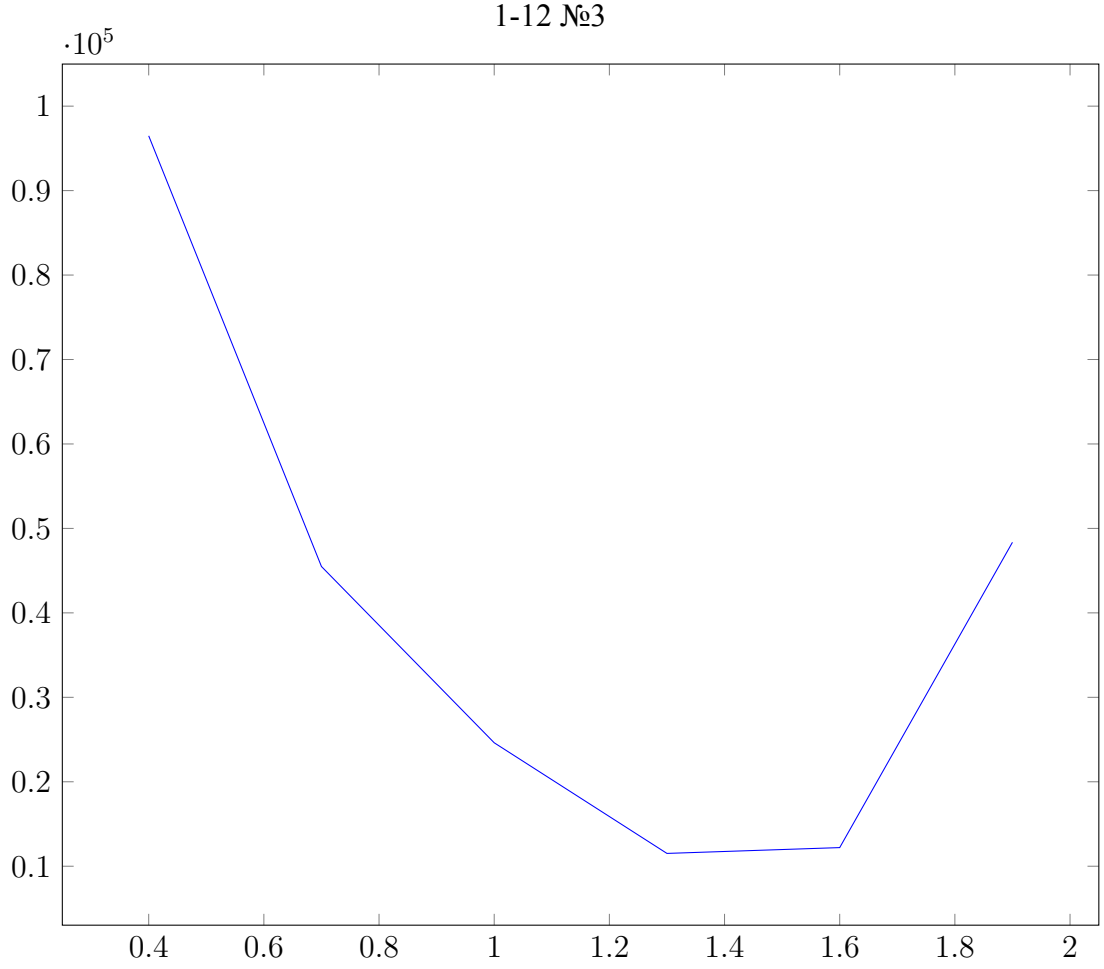
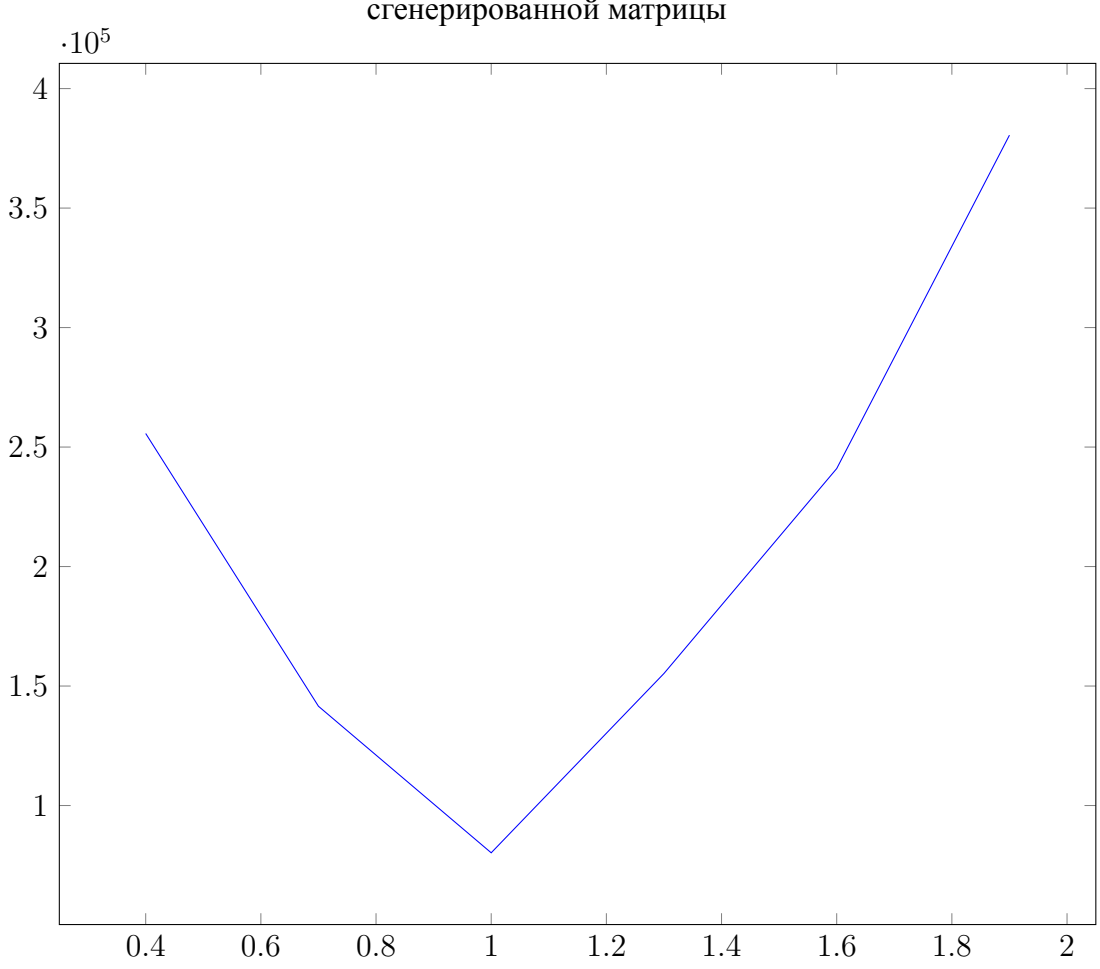


График зависимости числа итераций от итерационного параметра ω для сгенерированной матрицы



Выводы

Мы провели исследование и релаизацию метода верхней релаксации для решения систем линейных алгебраических уравнений, исследовали скорость его сходимости в зависимости от итерационного параметра ω. Анализируя графики зависимостей количества итераций от итерационного параметра ω на исследуемом промежутке [0.4, 1.9], приходим к выводу, что не существует оптимального параметра для всех матриц сразу. Для отдельных матриц нужно подбирать персонально, но на данных тестах наблюдается общая тенденция: ω находится в некоторой окрестности единицы.

Список цитируемой литературы

Список литературы

[1] Костомаров Д. П., Фаворский А. П. Вводные лекции по численным методам: Учеб. Пособие. – М.: Университетская книга, Логос, 2006

[2] Ильин В. А., Ким Г. Д. Линейная алгебра и аналитическая геометрия: учебник. 3-е изд., перераб., 2014

[3] Самарский А А. Введение в численные методы. Москва: Издательство «Наука», 1982