МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ №1

Вариант 1-3, 2-6

ОТЧЕТ

о выполненном задании

студента 204 учебной группы факультета ВМК МГУ
Васильева Руслана Леонидовича

Москва, 2019

Содержание

1 Подвриант 1

1.1 Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Написать программу, решающую систему линейных алгебраических уравнений заданного пользователем размера (n - параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента.

1.2 Цели и задачи практической работы

1. Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента
2. Вычислить определитель матрицы
3. Вычислить обратную матрицу
4. Определить число обусловленности
5. Исследовать вопрос вычислительной устойчивости метода Гаусса
6. Подтвердить правильность решения СЛАУ системой тестов

1.3 Методы решения

Рассмотрим СЛАУ в матричном виде: $Ax = f$ $(*)$, где

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \qquad f = \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$$

Считая, что $\det A \neq 0$, решим систему методом Гаусса, который разделяется на 2 этапа:

1. **Прямой ход:**
   С помощью элементарных преобразований строк $(*)$ получим эквивалентную систему с обнуленными элементами, лежащими ниже главной диагонали матрицы A. Для этого разделим все члены первого уравнения на $a_1 1 \neq 0$ (в противном случае меняем местами первую строку с той, где $a_m 1 \neq 0$, – такой элемент найдется в силу невырожденности матрицы). Затем вычтем из остальных уравнение первое, умноженное на первый коэффициент соответственно. Далее проделаем те же шаги для СЛАУ, полученного вычеркиванием первой строки и первого столбца – таким образом, получим СЛАУ с верхней треугольной матрицей, главная диагональ которой состоит из единиц.

2. **Обратный ход:**
   Также с помощью элементарных преобразований строк обнулим элементы, находящиеся над главной диагональю матрицы A.

Чтобы погрешность в методе Гаусса не нарастала, используется процедура выбора главного элемента: на этапе прямого хода стобцы переставляются таким образом, чтобы диагональный элемент был максимальным по модулю.

Заметим, что метод Гаусса можно использовать для вычисления определителя – при делении ведущей строки на ее диагональный элемент определитель также делится на этот элемент. Кроме того, каждая перестановка меняет знак определителя. Получается, определитель исходной системы равен

$$detA = (-1)^k a_{11} a'_{22} \dots a'_{nn},$$

где k – число перестановок столбцов в процессе редукции матрицы A к треугольной.

Для вычисления обратной матрицы будем применять элементарные преобразования из метода Гаусса к расширенной матрице $[A|I]$, где I - единичная матрица. Расширенная матрица преобразуется к $[I|A^{-1}]$.

Введем матричную норму

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$$

(в программе также предусмотрены $\|A\|_\infty$ и $\|A\|_F$). Число обусловленности: $\mu_A = \|A\|\|A^{-1}\|$.

1.4 Реализация

1.4.1 Основные функции

Подключим библиотеки для эффективной работы с матрицами

```
import numpy as np
import pandas as pd
```

Стандартный метод Гаусса

```
def naive_gauss(matrix):
    B = matrix.copy()
#     Решаем СЛАУ Ax=f, где матрица B = [A | f] получена
#     добавлением столбца свободных членов к основной матрице
    n = len(B) # Размерность
#     Прямой ход
    for i in range(n):
#         Находим строку с ненулевым ведущим элементом
#         и переставляем с текущей
        swap = np.flatnonzero(B[i:, i])[0] + i
        B[[swap, i]] = B[[i, swap]]
#         Преобразуем строки
        B[i, i:] /= B[i][i]
        for j in range(i + 1, n):
            B[j, i:] -= B[j][i] * B[i, i:]
#     Обратный ход
    for j in reversed(range(n)):
        B[:j, j:] -= B[:j, j, np.newaxis] * B[j, j:]
#     Извлекаем решение (последний столбец)
    return B[:, -1]
```

Метод Гаусса с выбором главного элемента

```
def gauss(matrix, save=False):
    if save:
        B = matrix.copy()
    else:
        B = matrix
#     Решаем СЛАУ Ax=f, где матрица B = [A | f] получена
#     добавлением столбца свободных членов к основной матрице
    n = len(B)
```

```
    order = np.arange(n)  # Заведем массив перестановок
#     Прямой ход
    for i in range(n):
#         Запоминаем номер главного элемента и переставляем столбцы
        main = abs(B[i, i:n]).argmax() + i
        order[i], order[main] = order[main], order[i]
        B[:, [i, main]] = B[:, [main, i]]
#         Преобразуем строки
        B[i, i:] /= B[i][i]
        for j in range(i + 1, n):
            B[j, i:] -= B[j][i] * B[i, i:]
#     Обратный ход
    for j in reversed(range(n)):
        B[:j, j:] -= B[:j, j, np.newaxis] * B[j, j:]
#     Извлекаем решение (последний столбец)
    return B[:, -1][np.argsort(order)]
```

Определитель матрицы

```
def det(matrix, save=True, eps=1e-12):
#     Найдем определитель матрицы, модифицируя
#     прямой ход метода Гаусса
    if save:
        A = matrix.copy()
    else:
        A = matrix
    n = len(A)
    d = np.float64(1)
    for i in range(n):
        main = abs(A[i, i:n]).argmax() + i
        A[:, [i, main]] = A[:, [main, i]]
        if (abs(A[i][i]) < eps):
            return 0
#         Умножим на главный элемент с соответствующим знаком
        d *= (2 * (main == i) - 1) * A[i][i]
        A[i, i:] /= A[i][i]
        for j in range(i + 1, n):
            A[j, i:] -= A[j][i] * A[i, i:]
    return d
```

Обратная матрица

```
def inverse(A):
#     Присоединяем к исходной матрице единичную
    AI = np.append(A, np.identity(len(A)), axis=1)
#     Выполняем все преобразования метода Гаусса на присоединенной
    gauss(AI)  # По умолчанию матрица не копируется
    return(AI[:, (len(AI)):])
```

Нормы

```
def eu(x):
    return np.sqrt(np.square(x).sum())

def mx_norm(A, ord=1):
    if ord == 1:
        return abs(A).sum(axis=0).max()
    if np.isinf(ord):
        return abs(A).sum(axis=1).max()
    if ord == 'fro':
        return eu(A)
```

Число обусловленности

```
def cond(A, ord=1):
    return mx_norm(A, ord=ord) * mx_norm(inverse(A), ord=ord)
```

1.4.2 Тестирование

Начнем тестирование на матрицах, заданных численно:

```
tests = []
for i in range(1, 4):
    tests.append(pd.read_csv(f'1-3_{i}.csv',
                             header=None,
                             dtype=np.float64).values)
    print(tests[i - 1], end='\n\n')
```

Здесь и далее голубым контуром выделяется вывод программы

```
[[ 2.  5.  4.  1. 20.]
 [ 1.  3.  2.  1. 11.]
 [ 2. 10.  9.  7. 40.]
 [ 3.  8.  9.  2. 37.]]

[[ 6.  4.  5.  2.  1.]
 [ 3.  2.  4.  1.  3.]
 [ 3.  2. -2.  1. -7.]
 [ 9.  6.  1.  3.  2.]]

[[  2.   1.   1.   0.   2.]
 [  1.   3.   1.   1.   5.]
 [  1.   1.   5.   0.  -7.]
 [  2.   3.  -3. -10.  14.]]
```

Начнем с вычисления определителей:

```
for i in range(3):
    print(f'det A = {det(tests[i][:, :-1])}')
    print(f'check: {np.linalg.det(tests[i][:, :-1])}\n')
```

Для проверки используется библиотечная функция `np.linalg.det`

```
det A = -3.0000000000000155
check: -2.9999999999999982

det A = 0
check: 0.0
```

```
det A = -242.0
check: -242.00000000000009
```

Как видим, вторая матрица является вырожденной, поэтому метод Гаусса к ней применять нельзя. Немного изменим ее (чтобы использовать для дальнейшего тестирования).

```
tests[1][1][1] = 5
tests[1][2][3] = 3
print(tests[1])
print(det(tests[1]))
```

```
[[ 6.  4.  5.  2.  1.]
 [ 3.  5.  4.  1.  3.]
 [ 3.  2. -2.  3. -7.]
 [ 9.  6.  1.  3.  2.]]
234.0
```

Применим к каждой из матриц обычный метод Гаусса, метод Гаусса с выбором главного элемента и встроенную реализацию:

```
for i in range(3):
    print(naive_gauss(tests[i], save=True))
    print(gauss(tests[i], save=True))
    print(np.linalg.solve(tests[i][:, :-1], tests[i][:, -1]))
```

```
[ 1.  2.  2. -0.]
[ 1.00000000e+00  2.00000000e+00  2.00000000e+00 -8.10462808e-15]
[ 1.00000000e+00  2.00000000e+00  2.00000000e+00 -1.60189322e-15]
[ 0.95726496  0.87179487 -0.07692308 -3.92307692]
[ 0.95726496  0.87179487 -0.07692308 -3.92307692]
[ 0.95726496  0.87179487 -0.07692308 -3.92307692]
[ 1.  2. -2. -0.]
[ 1.  2. -2. -0.]
[ 1.  2. -2. -0.]
```

Может показаться странным, что метод Гаусса без выбора главного элемента сработал лучше на первой матрице. Однако причина проста: это последний элемент, а значит в стандартной реализации ошибка на нем не наращивается – мы уже столкнулись с вычислительной неустойчивостью метода.

Найдем обратные матрицы (для основных матриц систем) и числа обусловленности.

```
for i in range(3):
    A = B[i][:, :-1]
    print(inverse(A))
    print(np.linalg.inv(A))
    print(cond(A))
    print()
```

```
[[ 15.         -21.           2.          -4.        ]
 [ -6.66666667  10.33333333  -1.           1.66666667]
 [ -0.33333333  -0.33333333   0.           0.33333333]
 [  5.66666667  -8.33333333   1.          -1.66666667]]
[[ 1.50000000e+01 -2.10000000e+01  2.00000000e+00 -4.00000000e+00]
 [-6.66666667e+00  1.03333333e+01 -1.00000000e+00  1.66666667e+00]
 [-3.33333333e-01 -3.33333333e-01 -1.13242749e-16  3.33333333e-01]
 [ 5.66666667e+00 -8.33333333e+00  1.00000000e+00 -1.66666667e+00]]
1039.9999999999998

[[ 0.07264957 -0.22222222 -0.16666667  0.19230769]
 [-0.28205128  0.33333333  0.          0.07692308]
 [ 0.23076923 -0.          0.         -0.15384615]
 [ 0.26923077 -0.          0.5        -0.34615385]]
[[ 7.26495726e-02 -2.22222222e-01 -1.66666667e-01  1.92307692e-01]
 [-2.82051282e-01  3.33333333e-01  6.40513283e-18  7.69230769e-02]
 [ 2.30769231e-01 -1.70803542e-17 -1.28102657e-17 -1.53846154e-01]
 [ 2.69230769e-01 -3.84307970e-17  5.00000000e-01 -3.46153846e-01]]
17.94871794871795

[[ 0.65289256 -0.16528926 -0.10743802 -0.01652893]
 [-0.21900826  0.37190083 -0.00826446  0.03719008]
 [-0.08677686 -0.04132231  0.2231405  -0.00413223]
 [ 0.09090909  0.09090909 -0.09090909 -0.09090909]]
[[ 0.65289256 -0.16528926 -0.10743802 -0.01652893]
 [-0.21900826  0.37190083 -0.00826446  0.03719008]
 [-0.08677686 -0.04132231  0.2231405  -0.00413223]
 [ 0.09090909  0.09090909 -0.09090909 -0.09090909]]
11.545454545454547
```

Как видим, у первой матрицы большое число обусловленности. Возмутим ее правую часть:

```
t = tests[0].copy()
```

```
print(t)
print(gauss(t, save=True))
np.random.seed(1)
t[:, -1] += np.random.normal(0, scale=0.1, size=len(t))
print(t)
print(gauss(t))
```

```
[[ 2.  5.  4.  1. 20.]
 [ 1.  3.  2.  1. 11.]
 [ 2. 10.  9.  7. 40.]
 [ 3.  8.  9.  2. 37.]]
[ 1.00000000e+00  2.00000000e+00  2.00000000e+00 -8.10462808e-15]
[[ 2.          5.          4.          1.         20.16243454]
 [ 1.          3.          2.          1.         10.93882436]
 [ 2.         10.          9.          7.         39.94718282]
 [ 3.          8.          9.          2.         36.89270314]]
[5.04475961 0.15894387 1.93048141 1.55627031]
```

Видно, что плохая обусловленность матрицы сказывается на неустойчивости решения.

Протестируем алгоритмы на системе из Приложения 2 (6-1 вариант), она определяется следующим образом:

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

где $i, j \in \{1, \ldots, n\}, n = 25, m = 10$. Сгенерируем:

```
# Вариант 6 (Приложение 2)
def gen_matrix(sep=False):
    n = 25
    m = 10
    tmp = np.arange(n) + 1.0
    b = np.square(tmp) - n
    A = (tmp[:, np.newaxis] + tmp) / (m + n)
    np.fill_diagonal(A, n + m ** 2 + (np.arange(n) + 1) * (1 / m + 1 / n))
    if sep:  # Будет использоваться во второй главе
        return A, b
    else:
        return np.concatenate((A, b[:, np.newaxis]), axis=1)
```

```
big = gen_matrix()
print(f'det: {det(big[:, :-1])}, check: {np.linalg.det(big[:, :-1])}')
print(cond(big[:, :-1]))
print(naive_gauss(big))
print(gauss(big))
print(np.linalg.solve(big[:, :-1], big[:, -1]))
```

```
det: 3.755666049264206e+52, check: 3.755666049264226e+52
1.416476312678141
[-0.35440646 -0.3376128  -0.30487312 -0.25621903 -0.19168209 -0.11129375
 -0.01508538  0.09691172  0.22466633  0.36814734  0.52732368  0.70216441
  0.89263863  1.09871555  1.32036444  1.55755465  1.81025564  2.07843691
  2.36206807  2.6611188   2.97555886  3.30535807  3.65048637  4.01091373
  4.38661024]
[-0.35440646 -0.3376128  -0.30487312 -0.25621903 -0.19168209 -0.11129375
 -0.01508538  0.09691172  0.22466633  0.36814734  0.52732368  0.70216441
  0.89263863  1.09871555  1.32036444  1.55755465  1.81025564  2.07843691
  2.36206807  2.6611188   2.97555886  3.30535807  3.65048637  4.01091373
  4.38661024]
[-0.35440646 -0.3376128  -0.30487312 -0.25621903 -0.19168209 -0.11129375
 -0.01508538  0.09691172  0.22466633  0.36814734  0.52732368  0.70216441
  0.89263863  1.09871555  1.32036444  1.55755465  1.81025564  2.07843691
  2.36206807  2.6611188   2.97555886  3.30535807  3.65048637  4.01091373
  4.38661024]
```

Матрица хорошо обусловлена, но размерность велика, а значит, из-за особенностей метода Гаусса, точность вычислений не будет высокой. В следующей главе мы рассмотрим более эффективный алгоритм для решения данной системы.

А сейчас обратим внимание на непозволительно большое значение определителя. Хотя его вычисление не является приоритетной задачей, покажем, как поступить грамотнее: пронормируем матрицу и воспользуемся свойством определителя.

```
A, _ = gen_matrix(sep=True)
c = abs(A).max()
print(c)
A /= c
print(det(A) * (c ** len(A)))
```

```
128.5
3.7556660492642087e+52
```

1.5 Вывод

Таким образом, были исследованы, реализованы и протестированы алгоритмы, основанные на методе Гаусса. Показано, что для плохо обусловленных матриц алгоритм является вычислительно неустойчивым. Хотя его можно использовать в качество простого способа решения СЛАУ с невырожденной матрицей, вычисления определителя и поиска обратной матрицы, появление возмущений неизбежно, например, при преобразовании вектора правых частей в методе Гаусса из-за ошибок округления при выполнении арифметических операций.

2 Подвриант 2

2.1 Постановка задачи

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

$$(D + A_H)(x^{k+1} - x^k) + Ax^k = f \tag{1}$$

где D, A_H – соответственно диагональная и нижняя треугольные матрицы, k – номер текущей итерации.

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

$$(D + \omega A_H)\frac{x^{k+1} - x^k}{\omega} + Ax^k = f \tag{2}$$

где ω - итерационный параметр (при $\omega = 1$ метод верхней релаксации совпадает с методом Зейделя).

2.2 Цели практической работы

1. Исследовать метод верхней релаксации
2. Реализовать алгоритм
3. Разработать критерий остановки процесса для приближенного решения с заданной точностью
4. Изучить скорость сходимости итераций к точному решению задачи

2.3 Метод и алгоритм решения

Работаем со СЛАУ $(*)$ из первой главы. Будем проводить вычисления, исплользуя формулу рекуррентную формулу:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left(f_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>=i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

Заметим, что хранить промежуточные значения нет необходимости, поэтому шаг можно упрощенно записать:

$$x_i = x_i + \frac{\omega}{a_{ii}} (f_i - (a_i, x))$$

$$x_i += \frac{\omega}{a_{ii}} (f_i - (a_i, x))$$

Достижение необходимой точности обуславливается соотношением:

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

позволяющим оценить погрешность z_k через невязку ψ_k.

2.4 Реализация

2.4.1 Основные функции

Считаем, что функции из первой главы определены.

Невязка

```
def residual(A, x, f):
    return np.matmul(A, x) - f
```

Оценка точности

```
def prec(A, x, f):
    return mx_norm(residual(A, x, f)) * mx_norm(inverse(A))
```

Метод верхней релаксации

```
def sor(A, f, x0, w, eps=1e-10, info=False, max_iter=1e10):
    n = len(A)
    x = x0.copy()
    itercnt = 0
    while prec(A, x, f) > eps and itercnt < max_iter:
        for i in range(n):
            x[i] += w / A[i][i] * (f[i] - (A[i] * x).sum())
        if info:
            itercnt += 1
            print(f'x: {x}')
            print(f'res: {residual(A, x, f)}')
            print(f'norm: {eu(residual(A, x, f))}')
            print()
    if info:
        print(f'Число итераций: {itercnt}')
    return x
```

Параметры:

- A – основная матрица
- f – матрица свободных членов
- $x0$ – начальное приближение
- ω – фактор релаксации
- eps – необходимая точность

- $info$ – печать служебной информации
- max_iter – ограничение на число итераций

2.4.2 Тестирование

Смоделируем для начала примеры из учебника:

```
n = 2
A = np.array([[1, 1], [1, 2]])
b = np.array([0, 1])
x0 = np.zeros(2)
print(f'Численный ответ: {sor(A, b, x0, 1, 1e-1, True)}') # Метод Зейделя
print()
print(f'Численный ответ: {sor(A, b, x0, 4/3, 1e-1, True)}') # Оптимальный параметр
```

```
x: [0.  0.5]
res: [0.5 0. ]
norm: 0.5

x: [-0.5   0.75]
res: [0.25 0.  ]
norm: 0.25

x: [-0.75   0.875]
res: [0.125 0.   ]
norm: 0.125

x: [-0.875   0.9375]
res: [0.0625 0.    ]
norm: 0.0625

x: [-0.9375   0.96875]
res: [0.03125 0.     ]
norm: 0.03125

Число итераций: 5
Численный ответ: [-0.9375   0.96875]

x: [0.         0.66666667]
res: [0.66666667 0.33333333]
norm: 0.7453559924999298

x: [-0.88888889  1.03703704]
res: [0.14814815 0.18518519]
norm: 0.2371527495345499
```

```
x: [-1.08641975  1.04526749]
res: [-0.04115226  0.00411523]
norm: 0.04135751284411891

x: [-1.03155007  1.00594422]
res: [-0.02560585 -0.01966164]
norm: 0.03228373682323514

x: [-0.99740893  0.99629122]
res: [-0.00111772 -0.0048265 ]
norm: 0.0049542296166406805

Число итераций: 5
Численный ответ: [-0.99740893  0.99629122]
```

Видим, что нормы невязок и приближенные решения соответствуют данным из методического пособия [1]: поведение невязок, а также сравнение членов итерационной последовательности с точным решением системы $x = \{-1, 1\}$ показывают сходимость процесса, более быструю, чем в методе Зейделя. Выбранное значение параметра $\omega = 4/3$ оказалось близким к оптимальному.

Условием сходимости метода верхней релаксации является положительная определенность и симметричность основной матрицы. Матрицы из варианта, которые заданы численно, не удовлетворяют данному условию, поэтому рассмотрим дополнительные тесты.

Сгенерируем теперь матрицы с помощью специально написанной функции, а также вспомним о матрице, заданной функционально (первая глава).

```
A1, A2 = gen(3), gen(5)
f1, f2 = np.random.normal(size=3), np.random.normal(size=5)
print(np.concatenate((A1, f1[:, np.newaxis]), axis=1))
print()
print(np.concatenate((A2, f2[:, np.newaxis]), axis=1))
A3, f3 = gen_matrix(sep=True)
```

```
[[ 4.76405235  1.3205252   0.9644132  -1.45436567]
 [ 1.3205252   4.86755799 -0.56431754  0.04575852]
 [ 0.9644132  -0.56431754  2.89678115 -0.18718385]]

[[ 6.76405235 -0.28856034  0.56139078  1.28728376 -0.34271591  1.53277921]
 [-0.28856034  5.95008842  0.65145815  0.69543011  0.53210855  1.46935877]
 [ 0.56139078  0.65145815  5.76103773 -0.04174162  0.65414972  0.15494743]
```

```
 [ 1.28728376  0.69543011 -0.04174162  5.3130677  -0.79813038  0.37816252]
 [-0.34271591  0.53210855  0.65414972 -0.79813038  7.26975462 -0.88778575]]
```

Попробуем подобрать оптимальные параметры ω.

```
omegas = np.linspace(0.1, 1.9, num = 10)
best_resids = [1, 1, 1]
best_x = [np.zeros(3), np.zeros(5), np.zeros(25)]
best_w = [0.1, 0.1, 0.1]
A = [A1, A2, A3]
f = [f1, f2, f3]
print(omegas)
for w in omegas:
    for i in range(3):
        x = sor(A[i], f[i], np.zeros(len(f[i])), w=w, max_iter=10)
        if eu(residual(A[i], x, f[i])) < best_resids[i]:
            best_x[i] = x
            best_resids[i] = eu(residual(A[i], x, f[i]))
            best_w[i] = w
print(best_w)
```

```
[0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9]
[1.3, 0.9, 1.1]
```

Получается, значения параметра $\omega = 1.3$, 0.9 и 1.1 близки к оптимальным для данных матриц. Убедимся, что метод верхней релаксации дает верное решение:

```
%%time
for i in range(3):
    print(sor(A[i], f[i], np.zeros(len(f[i])), w=best_w[i], eps=1e-5))
    print(np.linalg.solve(A[i], f[i]))
```

```
[-0.35184966  0.11350801  0.07463414]
[-0.35185029  0.11350726  0.07463434]
[ 0.24102577  0.27722387 -0.01296288 -0.04384248 -0.13469606]
[ 0.24102869  0.27722596 -0.01296356 -0.04384424 -0.13469629]
[-0.35440692 -0.33761329 -0.30487363 -0.25621955 -0.1916826  -0.11129424
 -0.01508584  0.09691129  0.22466596  0.36814702  0.52732343  0.70216422
  0.8926385   1.09871548  1.32036443  1.5575547   1.81025573  2.07843705
```

```
  2.36206824  2.66111899  2.97555905  3.30535827  3.65048655  4.0109139
  4.38661039]
[-0.35440646 -0.3376128  -0.30487312 -0.25621903 -0.19168209 -0.11129375
 -0.01508538  0.09691172  0.22466633  0.36814734  0.52732368  0.70216441
  0.89263863  1.09871555  1.32036444  1.55755465  1.81025564  2.07843691
  2.36206807  2.6611188   2.97555886  3.30535807  3.65048637  4.01091373
  4.38661024]
CPU times: user 28.1 ms, sys: 191 µs, total: 28.3 ms
Wall time: 26.3 ms
```

2.5 Вывод

Таким образом, был исследован и реализован метод верхней релаксации. Его преимуществом является быстрая сходимость при грамотном подборе фактора релаксации. Однако, в отличие от метода Гаусса, он применим только к определенному классу матриц.

Список литературы

[1] Костомаров Д. П. *Вводные лекции по численным методам: учебное пособие* / Д. П. Костомаров, А. П. Фаворский. — Москва: Логос, 2004.

[2] Ильин В. А. *Линейная алгебра и аналитическая геометрия: учебник для вузов* / В. А. Ильин, Г. Д. Ким. — Москва: Проспект, Изд-во МГУ им. М. В. Ломоносова, 2012.