Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт

По лабораторной работе №3

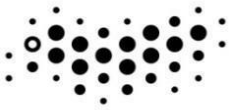
По дисциплине: прикладная математика

Факультет: ИТиП

Группа: М32001

Авторы:

Андреев Артём Русланович
Слюсаренко Сергей Владимирович
Шевченко Валерий Владимирович



УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2022

Лабораторная работа #3

1. Дана квадратная матрица. Требуется найти ее LU-разложение. Реализовать процедуру нахождения обратной матрицы с использованием LU-разложения. Реализовать методы решения системы с использованием LU- разложения. При этом матрица хранится в разреженно-строчном (разреженно-столцовом) формате (см. видео https://youtu.be/uCWNlhXKqQw). Элементы матрицы обрабатывать в порядке, соответствующем формату хранения. Для решения использовать метод Гаусса.

2. Протестировать разработанную программу.

3. Реализовать итерационный методы решения СЛАУ (Метод Зейделя, Якоби или верхней релаксации на выбор).

4. Провести исследование реализованных методов на матрицах, число обусловленности которых регулируется за счет изменения диагонального преобладания (т.е. оценить влияние увеличения числа обусловленности на точность решения). Для этого необходимо решить последовательность СЛАУ

$$A^k x^k = F^k, \quad k = 0, 1, 2, ...,$$

где матрицы A^k строятся следующим образом:

$$a_{ij} = \begin{cases} -\sum_{i \neq j} a_{ij}, i > 1, \\ -\sum_{i \neq j} a_{ij} + 10^{-k}, i > 1, \end{cases},$$

и $a_{ij} \in 0, -1, -2, -3, -4$ выбираются достаточно произвольно, а правая часть F_k получается умножение матрицы A^k на вектор $x* = (1, ..., n)$. Для каждого k, для которого система вычислительно разрешима, оценить погрешность найденного решения.

5. Провести аналогичные исследования на матрицах Гильберта различной размерности.
Матрицы Гильберта размерности k строится следующим образом:

$$a_{ij} = \frac{1}{i + j - 1}, i, j = 1..k.$$

6. Сравните между собой прямой и итерационный методы. Для сравнения используйте матрицы разной размерности: $n = 10$, 50, 10^2, 10^3, 10^4, 10^5, 10^6. Сделайте выводы, зависит ли эффективность метода от размерности матрицы. Если да, какая зависимость наблюдается?

7. Реализовать поиск обратной матрицы с использованием LU- разложения.

Реализация алгоритмов:
https://github.com/JabaJabila/ITMO_AppliedMath/tree/main/Lab3

LU разложение:

```
from scipy import sparse
import numpy as np


def lu_decomposition(a: sparse.csr_matrix):
    n = a.shape[0]
    u = sparse.lil_matrix(a)

    l = sparse.lil_matrix(np.eye(n))

    iteration_counter = 0
    for i in range(0, n):
        for j in range(i, n):
            l[j, i] = u[j, i] / u[i, i]
            iteration_counter += 1

    for k in range(1, n):
        for i in range(k - 1, n):
            for j in range(i, n):
                l[j, i] = u[j, i] / u[i, i]
                iteration_counter += 1

        for i in range(k, n):
            for j in range(k - 1, n):
                u[i, j] = u[i, j] - l[i, k - 1] * u[k - 1, j]
                iteration_counter += 1

    return sparse.csr_matrix(l), sparse.csr_matrix(u), iteration_counter
```

Решение СЛАУ при помощи LU:

```
import numpy as np
from scipy import sparse
from tools.luDecomposition import lu_decomposition


def linear_equations_system_solve(a: sparse.csr_matrix, v: sparse.csr_matrix):
    l, u, iteration_count = lu_decomposition(a)

    t, new_iteration_count = lower(l, v)
    iteration_count += new_iteration_count

    answer, new_iteration_count = upper(u, t)
    return answer, iteration_count + new_iteration_count


def linear_equations_system_solve_triangle(a: sparse.csr_matrix, v:
sparse.csr_matrix, is_lower: bool):
    if is_lower:
        return lower(a, v)
    else:
        return upper(a, v)
```

```
def lower(a: sparse.csr_matrix, v: sparse.csr_matrix):
    x = []
    iteration_counter = 0
    n = v.shape[1]
    m = v.shape[0]

    for k in range(0, m):
        vector_v = v.getrow(k).toarray()[0]
        vector_x = []
        for i in range(0, n):
            for j in range(0, i):
                iteration_counter += 1
                vector_v[i] -= a[i, j] * vector_x[j]

            iteration_counter += 1
            vector_x.append(vector_v[i] / a[i, i])
        x.append(vector_x)

    matrix_x = sparse.csr_matrix(x)
    return matrix_x.transpose(), iteration_counter


def upper(a: sparse.csr_matrix, v: sparse.csr_matrix):
    n = v.shape[0]
    m = v.shape[1]

    x = np.empty(m, dtype=np.dtype)
    iteration_counter = 0

    for k in range(0, m):
        vector_v = v.getcol(m - 1 - k).transpose().toarray()[0]
        vector_x = np.empty(n, dtype=float)
        for i in range(0, n):
            for j in range(n - i, n):
                iteration_counter += 1
                vector_v[n - 1 - i] -= a[n - 1 - i, j] * vector_x[j]

            iteration_counter += 1
            vector_x[n - 1 - i] = (vector_v[n - 1 - i] / a[n - 1 - i, n - 1 -
i])
        x[m - 1 - k] = vector_x

    matrix_x = sparse.csr_matrix(x.tolist())
    return matrix_x.transpose(), iteration_counter
```

Поиск обратной матрицы с помощью LU:

```
from scipy import sparse
from tools.luDecomposition import lu_decomposition
from tools.linearEquationsSystemSolve import
linear_equations_system_solve_triangle


def inverse_matrix(a: sparse.csr_matrix):
    n = a.shape[0]
    e = get_identity_matrix(n)
    l, u, iteration_count = lu_decomposition(a)

    t, new_iteration_count = linear_equations_system_solve_triangle(l, e, True)
```

```
    iteration_count += new_iteration_count

    inverse, new_iteration_count = linear_equations_system_solve_triangle(u, t,
False)
    iteration_count += new_iteration_count
    return inverse, iteration_count


def get_identity_matrix(n: int):
    matrix = sparse.lil_matrix((n, n))
    for i in range(0, n):
        matrix[i, i] = 1

    return matrix
```

Метод Зейделя:

```
import numpy as np
import copy


def is_need_to_complete(x_old, x_new, eps):
    for i in range(len(x_new)):
        if abs(x_old[i] - x_new[i]) > eps:
            return False

    return True


def seidel(A, B, eps):
    count = len(B)
    x = np.array([0.0] * count)

    number_of_iter = 0

    while number_of_iter < 100000:

        x_prev = copy.deepcopy(x)

        for i in range(count):
            s = 0
            for j in range(count):
                number_of_iter += 1
                if j != i:
                    s = s + A[i, j] * x[j]
            x[i] = B[i] / A[i, i] - s / A[i, i]

        if is_need_to_complete(x_prev, x, eps):
            break

    return x, number_of_iter
```

Поиск числа обусловленности матрицы:

```
import numpy as np
from scipy import sparse
from tools.inverseMatrix import inverse_matrix


def norm(A: sparse.csr_matrix):
    result = 0
```

```
    k, n = A.shape
    for i in range(k):
        for j in range(n):
            result += A[i, j] ** 2
    return np.sqrt(result)


def get_conditional_number(a: sparse.csr_matrix):
    a_inverse = inverse_matrix(a)[0]
    return norm(a) * norm(a_inverse)
```

1. **LU разложение**

LU разложение – это представление матрицы A в виде произведения матриц LU, где L – нижняя треугольная матрица, а U – верхняя треугольная. Важно отметить, что все элементы на главной диагонали должны быть ненулевыми, и сама матрица A должна быть обратимой.

LU разложением можно воспользоваться для решения СЛАУ. Пусть у нас уравнение вида Ax = b, где A – наша матрица, x – искомый вектор и b – известный вектор. Заменим матрицу на её LU разложение. Получим LUx = b. Теперь, сначала решим уравнение Lt = b (где t = Ux), а затем уравнение Ux = t. Так как матрицы L и U треугольные, то нахождение векторов t и x является довольно простой задачей.

Также, пользуясь разложением, можно находить обратную матрицу к заданной матрице. Мы, по сути, снова решаем СЛАУ, но в уравнении Ax = b, вместо b ставится единичная матрица (то есть Ax = E). Теперь нам нужно k раз (где k – размерность матрицы) решить СЛАУ $Ax_i = E_i$, складывая получаемые x_i в одну матрицу. В результате, матрица x будет равна обратной матрице.

2. **Пример работы**

Исходная матрица A:

```
A:
[[1. 0. 0. 0. 1.]
 [0. 2. 1. 8. 1.]
 [0. 0. 9. 0. 0.]
 [0. 7. 1. 3. 0.]
 [3. 7. 0. 3. 2.]]
```

Матрица L:

```
L:
[[ 1.          0.          0.          0.          0.        ]
 [ 0.          1.          0.          0.          0.        ]
 [ 0.          0.          1.          0.          0.        ]
 [ 0.          3.5        -0.27777778  1.          0.        ]
 [ 3.          3.5        -0.38888889  1.          1.        ]]
```

Матрица U:

```
U:
[[  1.    0.    0.    0.    1. ]
 [  0.    2.    1.    8.    1. ]
 [  0.    0.    9.    0.    0. ]
 [  0.    0.    0.  -25.   -3.5]
 [  0.    0.    0.    0.   -1. ]]
```

Проверим, что при умножении L на U, получается матрица A:

```
L * U:
[[1. 0. 0. 0. 1.]
 [0. 2. 1. 8. 1.]
 [0. 0. 9. 0. 0.]
 [0. 7. 1. 3. 0.]
 [3. 7. 0. 3. 2.]]
```

Полученная матрица совпала с исходной. Для разложения понадобилось 89 итераций.

Найденная обратная матрица:

```
Inverse A:
[[-2.          0.          0.11111111 -1.          1.        ]
 [ 0.18       -0.06       -0.01777778  0.22       -0.06      ]
 [ 0.          0.          0.11111111  0.          0.        ]
 [-0.42        0.14        0.00444444 -0.18        0.14      ]
 [ 3.          0.         -0.11111111  1.         -1.        ]]
```

Проверим, что при умножении исходной матрицы на обратную, получается единичная матрица:

```
(Inverse A) * A:
[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [-2.22044605e-16  1.00000000e+00  0.00000000e+00 -6.66133815e-16
  -2.22044605e-16]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 5.55111512e-17  1.11022302e-16  0.00000000e+00  1.00000000e+00
   5.55111512e-17]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00]]
```

Полученная матрица является единичной. Для нахождения обратной матрицы (включая LU разложение) понадобилось 239 итераций.

Решение СЛАУ при векторе b равном [1, 2, 3, 4, 5]:

```
Solution of system A with B vector [1, 2, 3, 4, 5] (lu method)
[[-0.66666667  0.58666667  0.33333333 -0.14666667  1.66666667]]
```

При умножении исходной матрицы на полученный вектор получается вектор [1, 2, 3, 4, 5], значит метод работает правильно.

3. Метод Зейделя

Метод Зейделя – итерационный метод решения системы уравнений, для того чтобы описать этот метод, стоит для начала определить обычный итерационный метод, суть которого заключается в том, что мы выводим:

$x_1^{(k+1)}$ из x_2^k, x_3^k … x_n^k

$x_2^{(k+1)}$ из x_1^k, x_3^k … x_n^k

…

$x_n^{(k+1)}$ из x_1^k, x_2^k … x_{n-1}^k

Где x_1^0, x_2^0… x_n^0= 0, 0 … 0
Такую систему не сложно составить и решить итерационно и будет выглядеть она след. образом:

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

Этот метод можно модифицировать таким образом, что для подсчета x_i^k мы можем использовать уже полученные x_i^{k-n}, таким образом мы получаем метод Зейделя:

Метод Зейделя

$$\begin{cases} x_1^{(k+1)} = c_{12}x_2^{(k)} + c_{13}x_3^{(k)} + \ldots + c_{1n}x_n^{(k)} + d_1 \\ x_2^{(k+1)} = c_{21}x_1^{(k+1)} + c_{23}x_3^{(k)} + \ldots + c_{2n}x_n^{(k)} + d_2 \\ \ldots \\ x_n^{(k+1)} = c_{n1}x_1^{(k+1)} + c_{n2}x_2^{(k+1)} + \ldots + c_{n(n-1)}x_{n-1}^{(k+1)} + d_n \end{cases},$$

$$c_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & j \neq i, \\ 0, & j = i. \end{cases} \quad d_i = \frac{b_i}{a_{ii}}, \quad i = 1, \ldots, n.$$

Стоит заметить, что сходимость обоих методов равна сходимости геометрической прогрессии, только в случае с методом Зейделя константа будет меньше

Докажем сходимость итерационного метода, из сходимости итерационного метода следует сходимость метода Зейделя:
Суть итерационного метода заключается в представлении нашей системы в таком виде:

$$Ax = b \iff x = Sx + f$$
$$x^k = Sx^{k-1} + f$$

Рассмотрим вектор ошибки в таком случае:

$$e^k = (x^k - x) = S(x^{k-1} - x)$$

а через k итераций:

$$e^k = (x^k - x) = S(x^{k-1} - x) = \\ = S^k(x^0 - x) = S^k e^0$$

т.е. можно сформулировать критерий сходимости, при котором вектор ошибок будет стремиться к 0

- Критерий сходимости:
  $e^k \to 0 \iff S^k e^0 \to 0 \; \forall e^0 \iff \rho(S) < 1$

4. Исследование методов на матрицах с разным числом обусловленности

Число обусловленности равно 3889308:

```
Diagonal matrix 5x5 #1
a = [[16.00001 -6.      -2.      -2.      -6.     ]
 [-3.      14.00001 -3.      -4.      -4.     ]
 [-6.      -2.      16.00001 -3.      -5.     ]
 [-5.      -1.      -6.      15.00001 -3.     ]
 [-2.      -2.      -6.       0.      10.00001]]
b = [-39.99999 -19.99998   1.00003  20.00004  26.00005]
Conditional number =  3889307.6976931565
>---------------------------------------------------------------<

Solution of system A with B vector (lu method)
[[1. 2. 3. 4. 5.]]
Iteration number
119
>---------------------------------------------------------------<

Solution of system A with B vector (Seidel method)
[-3.0125295  -2.01252834 -1.01252723 -0.01252585  0.98747611]
Iteration number
175
>---------------------------------------------------------------<

Error for lu
9.39959687352353e-15
>---------------------------------------------------------------<

Error for Seidel
0.0001244038041889491
>===============================================================<
```

Число обусловленности равно 3553399:

```
Diagonal matrix 5x5 #2
a = [[11.00001  0.      -3.      -3.      -5.     ]
 [-3.      19.00001 -4.      -6.      -6.     ]
 [-1.      -1.      11.00001 -3.      -6.     ]
 [-2.      -5.      -2.      10.00001 -1.     ]
 [ 0.      -1.      -3.      -6.      10.00001]]
b = [-34.99999 -30.99998 -11.99997  17.00004  15.00005]
Conditional number =  3553398.6090532485
>---------------------------------------------------------------<

Solution of system A with B vector (lu method)
[[1. 2. 3. 4. 5.]]
Iteration number
119
>---------------------------------------------------------------<

Solution of system A with B vector (Seidel method)
[-3.38786479 -2.38786442 -1.3878555  -0.387858    0.6121465 ]
Iteration number
150
>---------------------------------------------------------------<

Error for lu
8.702335715267317e-15
>---------------------------------------------------------------<

Error for Seidel
0.00023732038716445586
>===============================================================<
```

Число обусловленности равно 3332339:

```
Diagonal matrix 5x5 #3
a = [[ 7.00001  0.      -4.      -2.      -1.     ]
 [-6.      15.00001 -6.      -2.      -1.     ]
 [-5.      -1.      14.00001 -5.      -3.     ]
 [-2.      -6.       0.      12.00001 -4.     ]
 [-3.      -1.      -3.      -3.      10.00001]]
b = [-1.799999e+01 -6.999980e+00  3.000000e-05  1.400004e+01  2.400005e+01]
Conditional number =  3332339.4112588516
>---------------------------------------------------------------<

Solution of system A with B vector (lu method)
[[1. 2. 3. 4. 5.]]
Iteration number
119
>---------------------------------------------------------------<

Solution of system A with B vector (Seidel method)
[-2.95850926 -1.9585096  -0.95851207  0.04149073  1.04149382]
Iteration number
200
>---------------------------------------------------------------<

Error for lu
1.0986160316453369e-14
>---------------------------------------------------------------<

Error for Seidel
0.0001101783640187448
>===============================================================<
```

Как можно заметить, число обусловленности не влияет на количество итераций у решения СЛАУ с помощью LU. А вот у метода Зейделя, уменьшение числа обусловленности ведёт к увеличению количество итераций, требуемых для решения СЛАУ.

5. Исследование методов на матрицах Гилберта разных размеров

N = 5

```
Gilbert matrix 5x5
a = [[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125      0.11111111]]
b = [5.         3.55       2.81428571 2.34642857 2.01746032]
Conditional number =  480849.1169944144
>------------------------------------------------------------<

Solution of system A with B vector (lu method)
[[1. 2. 3. 4. 5.]]
Iteration number
119
>------------------------------------------------------------<

Solution of system A with B vector (Seidel method)
[0.99999625 2.02639721 2.82444736 4.32423602 4.82135593]
Iteration number
34725
>------------------------------------------------------------<

Error for lu
0.0
>------------------------------------------------------------<

Error for Seidel
2.205593874610787e-05
>============================================================<
```

N = 10

```
Gilbert matrix 10x10
Solution of system A with B vector (lu method)
Iteration number
714
>------------------------------------------------

Solution of system A with B vector (Seidel method)
Iteration number
100000
>------------------------------------------------

Error for lu
3.076740298213702e-15
>------------------------------------------------

Error for Seidel
0.00032042282607448665
>================================================
```

N = 50

```
Gilbert matrix 50x50
Solution of system A with B vector (lu method)
Iteration number
67574
>------------------------------------------------

Solution of system A with B vector (Seidel method)
Iteration number
100000
>------------------------------------------------

Error for lu
2.412185323914849e-13
>------------------------------------------------

Error for Seidel
0.6016989356720592
>================================================
```

Как мы видим, метод Зейделя, довольно плохо справляется с матрицами Гилберта. Прежде всего, это вызвано большим числом обусловленности и полностью матриц, в случае с 10 и 50, Зейдель упирается в поставленное ограничение 1e6 и даже не доходит до требуемой точности. В то время как, LU метод всё-также работает за фиксированное, от размера матрицы, время.

6. Сравнение методов на матрицах разного размера

N = 10

```
Diagonal matrix 10x10
Solution of system A with B vector (lu method)
Iteration number
714
>---------------------------------------------------

Solution of system A with B vector (Seidel method)
Iteration number
700
>---------------------------------------------------

Error for lu
9.592326932761353e-14
>---------------------------------------------------

Error for Seidel
9.876416618680956e-05
```

N = 50

```
Diagonal matrix 50x50
Solution of system A with B vector (lu method)
Iteration number
67574
>---------------------------------------------------

Solution of system A with B vector (Seidel method)
Iteration number
20000
>---------------------------------------------------

Error for lu
6.547241665279601e-12
>---------------------------------------------------

Error for Seidel
0.00060739727258895 83
```

N = 100

```
Diagonal matrix 100x100
Solution of system A with B vector (lu method)
Iteration number
520149
>---------------------------------------------------

Solution of system A with B vector (Seidel method)
Iteration number
80000
>---------------------------------------------------

Error for lu
6.492561182058435e-11
>---------------------------------------------------

Error for Seidel
0.004695768988163927
```

Уже видно, что метод Зейделя работает намного быстрее, чем LU. Это можно объяснить тем, что у LU асимптотика $O(n^3)$. Но стоит отметить, что LU находит точный ответ, а Зейдель только с заданной точностью.

Исследования работы алгоритмов на матрицах диагонального преобладания:

size n	epsilon	Cond(A)	LU iterations	Seidel iterations
5	0.01	3.129004e+06	119	100
10	0.01	9.223961e+11	714	500
50	0.01	1.499922e+17	67574	15000
100	0.01	1.242334e+17	520149	60000

Исследования работы алгоритмов на матрицах Гильберта:

size n	epsilon	Cond(A)	LU iterations	Seidel iterations
5	0.01	4.808491e+05	119	1375
10	0.01	1.633163e+13	714	16300
50	0.01	1.765061e+19	67574	100000
100	0.01	2.007829e+19	520149	100000

Исследования работы алгоритмов на рандомных матрицах:

size n	epsilon	Cond(A)	LU iterations	Seidel iterations
5	0.01	56.926044	119	23650
10	0.01	48.103055	714	15100
20	0.01	4538.432128	4829	53600
30	0.01	304.302476	15344	50400
40	0.01	3797.965538	35259	41600
50	0.01	694.598459	67574	100000
60	0.01	2001.536347	115289	100800
70	0.01	1506.864974	181404	102900
80	0.01	1398.620929	268919	102400
90	0.01	7886.013817	380834	105300
100	0.01	1693.847731	520149	100000

7. Вывод

Итоги сравнение двух методов решения СЛАУ, итерационного (Зейделя) и прямого (LU разложение) таковы: LU метод зависит от размера матрицы, поэтому большие матрицы он будет обрабатывать крайне долго, но зато не зависит от того, что находится внутри матрицы, а также ответ получается точным. Метод Зейделя же, сильно зависит от величины обусловленности матрицы. Отсюда, на больших матрицах метод Зейделя показывает себя лучше, а LU метод лучше применять на сравнительно небольших матрицах или если метод Зейделя не справляется с заданной матрицей.