Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине «Прикладная математика»

Методы решения СЛАУ

Выполнили Мельник Денис Александрович
Хакимов Руслан Венирович
Чечулин Лев Олегович

Проверил_____
(Фамилия, имя, отчество)

Санкт-Петербург, 2022 г.

ЗАДАНИЕ

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?

Немного теории

Система m линейных алгебраических уравнений с n неизвестными — это система уравнений вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{1n}x_n = b_2, \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{1n}x_n = b_m. \end{cases}$$

Здесь $x_1, x_2, \ldots, x_n$ — неизвестные, которые надо определить. Коэффициенты системы $a_{11}, a_{12}, \ldots, a_{mn}$ и её свободные члены $b_1, b_2, \ldots, b_m$ предполагаются известными. Индексы коэффициента a_{ij} системы обозначают номера уравнения i и неизвестного j, при котором стоит этот коэффициент.

Решение системы уравнений — совокупность n чисел $c_1, c_2, \ldots, c_n$, таких что подстановка каждого c_i вместо x_i в систему обращает все её уравнения в тождества.

Система линейных уравнений может быть представлена в матричной форме как

$$Ax = b,$$

где

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

Прямые (или точные) методы решения СЛАУ позволяют найти решение за определенное количество шагов. К прямым методам относятся метод Гаусса, метод Гаусса — Жордана, метод Крамера, матричный метод и метод прогонки (для трёхдиагональных матриц).

Итерационные методы основаны на использовании повторяющегося процесса. Они позволяют получить решение в результате последовательных приближений. К итерационным методам относятся метод Якоби (метод простой итерации), метод Гаусса — Зейделя, метод релаксации и многосеточный метод.

Ход работы

1. Необходимо реализовать следующие методы:

Метод Гаусса. Состоит из двух этапов. Первый - путем элементарных преобразований приведение к треугольной форме. На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений.

```
def gauss(A : np.double):
    n = A.shape[0]

    x = np.zeros(n)

    iter1=0
    for i in range(n):
        if A[i][i] == 0.0:
            sys.exit('Divide by zero detected!')

        for j in range(i+1, n):
            ratio = A[j][i]/A[i][i]

            for k in range(n+1):
                A[j][k] = A[j][k] - ratio * A[i][k]
                iter1+=1


    x[n-1] = A[n-1][n]/A[n-1][n-1]

    for i in range(n-2,-1,-1):
        x[i] = A[i][n]

        for j in range(i+1,n):
            x[i] = x[i] - A[i][j]*x[j]
            iter1+=n

        x[i] = x[i]/A[i][i]

    print("iter ", iter1)

    return x
```

LU-разложение — это представление матрицы A в виде A=L•U, где L — нижнетреугольная матрица с еденичной диагональю, а U — верхнетреугольная матрица. LU-разложение является модификациеё метода Гаусса.

```
def lu_decomposition(matrix : sparse.csr_matrix):

    n = matrix.shape[0]
    l = sparse.csr_matrix((n, n))
    u = sparse.csr_matrix((n, n))

    for i in range(0, n):

        for j in range(0, n):

            a = matrix[i, j]

            if i == j:
                l.data = np.append(l.data, 1)
                l.indices = np.append(l.indices, j)
                l.indptr[i + 1] = len(l.data)

            if i <= j:
                us = 0.
                for k in range(0, i):
                    us += l[i, k] * u[k, j]
                u.data = np.append(u.data, a - us)
                u.indices = np.append(u.indices, j)

            else:
                ls = 0.
                for k in range(0, j):
                    ls += l[i, k] * u[k, j]
                l.data = np.append(l.data, (a - ls) / u[j, j])
                l.indices = np.append(l.indices, j)
                l.indptr[i + 1] = len(l.data)

        u.indptr[i + 1] = len(u.data)

    return (l, u)
```

Поиск обратной матрицы с помощью LU разложения. Если A=LU, то A^-1 = U^-1 * L^-1

```
def inverse_matrix(matrix : sparse.csr_matrix):
    l, u = lu_decomposition(matrix)
    return sparse.csc_matrix(np.matmul(inverse_up_triangle_matrix(u).toarray(), inverse_low_triangle_matrix(l).toarray()))
```

2. Протестируем

```
def solve_with_lu(A : sparse.csr_matrix, y):
    l, u = lu_decomposition(A)

    return gauss(np.insert(u.toarray(), l.shape[1], gauss(np.insert(l.toarray(), l.shape[1], y, axis=1)), axis=1))
```

```
array_matrix = np.array([
    [2., 1., -1],
    [-3, -1, 2],
    [-2, 1, 2],
])
y = np.double([8, -11, -3])
print(solve_with_lu(array_matrix, y))
```

```
[ 2.  3. -1.]
```

Работает корректно.

3. Метод итерации — это численный и приближенный метод решения СЛАУ. Суть: нахождение по приближённому значению величины следующего приближения, которое является более точным. Метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов (итерационный

процесс). Итерационные методы являются особенно эффективными при решении систем с большим количеством неизвестных (порядка 1000 и более).

Канонический вид метода Зейделя:

$$\begin{cases} B_{k+1} = D + A_1, \\ t_{k+1} = 1. \end{cases}$$

i-я компонента $(k+1)$-го приближения вычисляется по формуле:

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} c_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} c_{ij} x_j^{(k)} + d_i, \quad i = 1, \ldots, n.$$

```
def seidel_f(A, f, eps):
    n = A.shape[0]
    X = np.double([1.] * n)

    d = eps + 0.1
    iter1 = 0
    #t = time.now()
    while d > eps:
        next_X = np.copy(X)
        for i in range(0, n):
            AlXn = sum([A[i, j] * next_X[j] for j in range(0, i)])
            AuXn = sum([A[i, j] * X[j] for j in range(i + 1, n)])
            next_X[i] = (f[i] - AuXn - AlXn) / A[i, i]

        d = np.sqrt(sum((next_X[i] - X[i]) ** 2 for i in range(n)))
        X = next_X
        iter1+=1
    print("iter ", iter1)
```

Метод Якоби. B=D, $\tau = 1$

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

```
def jacobi(A, f, eps):
    x = sparse.csr_matrix(([], [], [0] * (A.shape[0] + 1)), shape=(A.shape[0], 1), dtype=np.float64)
    iter = 0
    d = 1
    n = A.shape[0]

    while d > eps:
        next_x = x.copy()
        k = (A * x - f)
        for i in range(n):
            x[i, 0] = x[i, 0] - k[i, 0] / A[i, i]
            iter+=1
        print(x.toarray())
        d = np.sqrt(sum((next_x[i, 0] - x[i, 0]) ** 2 for i in range(n)))
    print("iterations: ", iter)
    return x
```

4.

Число обусловленности матрицы показывает насколько матрица близка к матрице неполного ранга (для квадратных матриц - к вырожденности). Рассмотрим систему линейных уравнений Ax=в. Если матрица A вырожденная, то для некоторых в решение x не существует, а для других в оно будет неединственным. Следовательно, если A почти вырожденная, то можно ожидать, что малые изменения в A и в вызовут очень большие изменения в x. Если же взять в качестве A единичную матрицу, то решение системы (1) будет x=в. Следовательно, если A близка к единичной матрице, то малые изменения в A и в должны влеч за собой малые изменения в x.

Говорят, что квадратная матрица A_{nn} обладает свойством **диагонального преобладания**, если для каждого $i = 1, \ldots, n$

$$|a_{ii}| \geqslant \sum_{j \neq i} |a_{ij}|,$$

Результаты

```
k =  0
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  3.5596458096434965e-15
 Iter method
iter  30
Расстояние до ист. решения:  0.04123919328453062
--------------------------------------------------
k =  1
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  1.6067562483063646e-13
 Iter method
iter  173
Расстояние до ист. решения:  0.4423873520355031
--------------------------------------------------
k =  2
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  2.605039209663162e-12
 Iter method
iter  673
Расстояние до ист. решения:  4.495641756419441
--------------------------------------------------
k =  3
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  5.620563939326112e-12
 Iter method
iter  5
Расстояние до ист. решения:  20.00643500747079
--------------------------------------------------
```

```
k =  4
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  4.213189752454186e-11
 Iter method
iter  5
Расстояние до ист. решения:  20.025561062674498
--------------------------------------------------
k =  5
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  7.864318227793021e-10
 Iter method
iter  5
Расстояние до ист. решения:  20.027474814285398
--------------------------------------------------
k =  6
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  5.3364718927287644e-09
 Iter method
iter  5
Расстояние до ист. решения:  20.02766620091258
--------------------------------------------------
k =  7
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  5.8982009118213875e-08
 Iter method
iter  5
Расстояние до ист. решения:  20.02768533968996
--------------------------------------------------
```

```
k =  8
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  1.0953799101885922e-06
 Iter method
iter  5
Расстояние до ист. решения:  20.027687253568818
--------------------------------------------------
k =  9
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  1.2638990459725089e-05
 Iter method
iter  5
Расстояние до ист. решения:  20.02768744495673
--------------------------------------------------
k =  10
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  3.651253533613253e-05
 Iter method
iter  5
Расстояние до ист. решения:  20.027687464095518
--------------------------------------------------
k =  11
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  0.00028089160243049906
 Iter method
iter  5
Расстояние до ист. решения:  20.02768746600942
--------------------------------------------------
```

```
k =  12
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  0.005614839595470074
 Iter method
iter  5
Расстояние до ист. решения:  20.02768746620079
--------------------------------------------------
k =  13
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  0.011293848786314899
 Iter method
iter  5
Расстояние до ист. решения:  20.027687466219938
--------------------------------------------------
k =  14
 Gauss
iter  945
iter  945
Расстояние до ист. решения:  0.2635231383473659
 Iter method
iter  5
Расстояние до ист. решения:  20.027687466221828
--------------------------------------------------
```

Видим равное количество итераций для прямого решения. И иногда различ, но меньшее количество итераций для итер метода при заданой точности.

5. Для матрицы Гильберта.

Матрицы Гильберта размерности k строится следующим образом:

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

```
---------- n =  5
LU:
iter  110
iter  110
Отклонение:  2.919433051780108e-11
Iter method:
iter  56
Отклонение:  1.715169237353423
---------- n =  6
LU:
iter  195
iter  195
Отклонение:  4.979907349036579e-10
Iter method:
iter  130
Отклонение:  1.8438404522161544
---------- n =  7
LU:
iter  315
iter  315
Отклонение:  4.281526271706285e-09
Iter method:
iter  193
Отклонение:  1.5909365282705865
---------- n =  8
LU:
iter  476
iter  476
Отклонение:  5.502825567782758e-07
Iter method:
iter  200
Отклонение:  2.111663895688396
```

```
---------- n =  9
LU:
iter  684
iter  684
Отклонение:  5.0948954367438736e-05
Iter method:
iter  191
Отклонение:  2.922328503163068
---------- n =  10
LU:
iter  945
iter  945
Отклонение:  0.0009053814012989534
Iter method:
iter  181
Отклонение:  3.8476519572153745
---------- n =  11
LU:
iter  1265
iter  1265
Отклонение:  0.09441871673604324
Iter method:
iter  174
Отклонение:  4.82106888536911
---------- n =  12
LU:
iter  1650
iter  1650
Отклонение:  0.602112078962517
Iter method:
iter  329
Отклонение:  4.3046399333009475
```

```
---------- n =  13
LU:
iter  2106
iter  2106
Отклонение:  231.71164014120322
Iter method:
iter  434
Отклонение:  3.780522390568156
---------- n =  14
LU:
iter  2639
iter  2639
Отклонение:  170.07085587828314
Iter method:
iter  481
Отклонение:  3.868175984340078
---------- n =  15
LU:
iter  3255
iter  3255
Отклонение:  229.08964040781672
Iter method:
iter  498
Отклонение:  4.35497908724108
---------- n =  16
LU:
iter  3960
iter  3960
Отклонение:  119.51625832977305
Iter method:
iter  500
Отклонение:  5.069551608655771
```

```
---------- n =  17
LU:
iter  4760
iter  4760
Отклонение:  191.91749935143324
Iter method:
iter  493
Отклонение:  5.913936565661055
---------- n =  18
LU:
iter  5661
iter  5661
Отклонение:  428.8134522272799
Iter method:
iter  483
Отклонение:  6.835878073248018
```

6. Сравнение

```
---------------- n =  10
Seidel:
iter  4
Отклонение:  0.00016986206509602062
0.01700282096862793 seconds
LU:
iter  945
iter  945
Отклонение:  3.66205343881779e-15
0.03403592109680176 seconds
---------------- n =  100
Seidel:
iter  3
Отклонение:  2.370829220647561e-06
0.8887491226196289 seconds
LU:
iter  994950
iter  994950
Отклонение:  6.252044293965066e-14
21.38393783569336 seconds
```

Вывод

Скорость решения слау сильно зависит от размеров входной матрицы. Наблюдается зависимость n^3 для прямого метода, сходимость со скоростью геометрической прогрессии для итерационного.

Вывод

В результате выполнения работы был изучен градиентный спуск с использованием различных методов одномерной оптимизации и подбора шага. Изучены зависимости от выбранной точки, от характера функции, от выбранного метода. Было произведено сравнение некоторых характеристик методов, например, кол-во итераций (шагов) при разных условиях. Был изучен метод сопряжённых градиентов, реализовав который, мы поняли, что он значительно ускорил все предыдущие известные нам методы сходимости. Так же мы сделали визуализацию градиентного спуска и написали код с автоматическим его исчислением. После сравнения всех составляющих, мы можем уверенно считать, что градиентный спуск изучен со всех сторон.