Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Вычислительная математика»

Студент	Фуров Павел Павлович
Группа	РК6-61Б
Тип задания	Лабораторная работа №4
Тема лабораторной работы	Методы решения СЛАУ с разреженными матрицами

Студент _____ **Фуров П.П.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Першин А.Ю.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Оглавление

Задание на лабораторную работу

Требуется (базовая часть):

1. Сформировать матрицу коэффициентов $\boldsymbol{A}$ и правый вектор СЛАУ $\boldsymbol{b}$, используя файлы https://archrk6.bmstu.ru/index.php/f/837681 и https://archrk6.bmstu.ru/index.php/f/837678 соответственно. Данная матрица является положительно определенной.
2. Получить решение с помощью наиболее подходящего прямого метода (из класса методов последовательного исключения). Данное решение мы далее будем называть точным.
3. Вычислить число обусловленности матрицы $\boldsymbol{A}$ и сделать вывод о возможности использования чисел с одинарной точностью для хранения матрицы и проведения всех дальнейших вычислений. При положительном решении далее необходимо использовать числа с одинарной точностью.
4. Понизить число обусловленности как минимум на два порядка, используя предобуславливание.
5. Написать функцию *jacobi(A, b)*, которая возвращает решение СЛАУ $\boldsymbol{Ax} = \boldsymbol{b}$, полученное с помощью метода Якоби.
6. Написать функцию *gauss_seidel(A, b)*, которая возвращает решение СЛАУ $\boldsymbol{Ax} = \boldsymbol{b}$, полученное с помощью метода Гаусса–Зейделя.
7. Для случая с и без предобуславливания:
   (a) Найти решение исходной СЛАУ с помощью метода Якоби.
   (b) Найти решение исходной СЛАУ с помощью метода Гаусса–Зейделя.
   (c) Вывести на экран зависимость среднеквадратичной нормы абсолютной погрешности от номера итерации для каждого из методов.
   (d) Вывести на экран зависимость среднеквадратичной нормы невязки от номера итерации для каждого из методов.
8. Объяснить различия в скорости сходимости для всех рассмотренных случаев.

Требуется (продвинутая часть):

1. Написать функцию visualize_matrix(A), которая визуализирует матрицу как двумерный рисунок, используя следующий подход: элементы матрицы, близкие к нулю, выводятся на экран как белые квадраты (или не

выводятся вовсе), а остальные элементы выводятся как квадраты, имеющие цвет из непрерывной цветовой палитры, которую необходимо выбрать самостоятельно.

2. Визуализировать матрицу $\boldsymbol{A}$ и сделать вывод о ее разреженности. Является ли данная матрица ленточной? Если да, то какова ширина ленты?
3. Написать и подробно описать класс SparseMatrix, позволяющий хранить разреженную матрицу в формате CSR (compressed sparse row, сжатое хранение строкой). Этот формат особенно эффективен при большом количестве перемножений матриц на вектора. В нем исходная матрица представляется в виде трех массивов:
    - массив, хранящий только ненулевые значения, которые берутся порядок из первой непустой строки;
    - массив, хранящий номера столбцов, соответствующих ненулевым значениям из первого массива;
    - массив, i-й элемент которого хранит количество ненулевых элементов в строках до i-1 включительно; в первом его элементе хранится 0, а последний элемент совпадает с число ненулевых элементов исходной матрицы.

    В классе SparseMatrix должны быть реализованы все матричные операции, необходимые в дальнейших пунктах задания.
4. Написать функцию conjugate_gradient_method(A, b), которая возвращает решение СЛАУ $\boldsymbol{Ax} = \boldsymbol{b}$, полученное с помощью метода сопряженных градиентов.:
5. Для случая с и без предобуславливания:

    (a) Найти решение исходной СЛАУ с помощью метода сопряженных градиентов, храня матрицу $\boldsymbol{A}$ в формате CSR.

    (b) Вывести на экран зависимость среднеквадратичной нормы абсолютной погрешности от номера итерации.

    (c) Вывести на экран зависимость среднеквадратичной нормы невязки от номера итерации.
6. Сделать вывод о скорости сходимости метода Якоби, метода Гаусса–Зейделя и метода сопряженных градиентов для данной матрицы коэффициентов и выбрать наилучший метод.

Цель выполнения лабораторной работы

Системы линейных алгебраических уравнений неизбежно появляются как промежуточный или конечный этап в нахождении численных решений в ряде методов вычислительной математики и анализа данных. Часто матрицы большой размерности оказываются разреженными, что требует использования более компактных способов хранения матриц в памяти устройства. СЛАУ, основанные на таких матрицах, чаще всего решаются с использованием итерационных или полупрямых методов. Выбор конкретного метода основан на заранее известных свойствах матрицы и эмпирических проверках.

Цель базовой части:

- Рассмотреть несколько методов решения СЛАУ и изучить скорость их сходимости на примере матрицы коэффициентов, полученной в MSC/NASTRAN

Цели продвинутой части:

- Реализовать метод сжатое хранения разреженной матрицы строкой и проверить его эффективность на примере различных численных методов решения СЛАУ, в том числе метода сопряжённых градиентов.

Ход выполнения лабораторной работы

1. Базовая часть

1.1. Формирование матрицы коэффициентов

Матрица коэффициентов и правый вектор СЛАУ были сформированы с помощью функции numpy.genfromtxt

Листинг 1 – формирование матрицы коэффициентов и правого вектора СЛАУ

```
A = np.genfromtxt('A.csv', delimiter=',')
b = np.genfromtxt('b.csv', delimiter=',')
```

1.2. Получение точного решения

Если матрица является положительно определенной, то из класса прямых методов логично выбрать разложение Холецкого[1]. Также матрица должна быть симметричной. На рисунке 1 указан результат проверки матрицы $\boldsymbol{A}$ на симметричность и положительную определённость.

```
Ввод [10]: np.all(A == A.T)
Out[10]: True

Ввод [12]: np.all(lg.eigvals(A)>0)
Out[12]: True
```

Рисунок 1 – результат проверки матрицы A

Разложением Холецкого является разложение матрицы в виде:

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T, \quad (1)$$

где $\boldsymbol{L}$ – нижняя треугольная матрица со строго положительными элементами на диагонали. Выполнив такое разложение, решение можно получить, решив две СЛАУ: $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ и $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$.

Матрицу $\boldsymbol{L}$ позволяет получить функция *np.lilang.cholesky*.

Листинг 2 – Функция возвращающая «точное» решение.

```
1  def correct_solve(A,b):
2      L=lg.cholesky(A)
3      y=L.I*b
4      x=(L.T).I*y
5      return x
```

1.3. Вычисление числа обусловленности

Число обусловленности $\boldsymbol{K(A) = \|A\| \cdot \|A\|}$ определяется в неравенстве

$$\frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \leq \boldsymbol{K(A)} \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}, \qquad (\mathbf{2})$$

где $\tilde{\boldsymbol{x}}$ – приближение к точному решению, $\boldsymbol{r = A - b\tilde{x}}$ – вектор невязки.

Его можно вычислить с помощью функции *numpy.lilang.cond*, для матрицы $\boldsymbol{A}$ число обусловленности приблизительно равно 417817.

Число обусловленности позволяет оценить, сколько значимых цифр теряется в процессе вычислений:

$$\frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \approx \boldsymbol{10^{-t} K(A)}$$

Матрицу можно считать достаточно обусловленной при $\boldsymbol{K(A) < 10^t}$. Числа одинарной точности с плавающей запятой имеют 7-8 значащих цифр. В случае с матрицей $\boldsymbol{A}$ $\boldsymbol{K(A) < 10^7}$, а значит в дальнейшем можно использовать числа с одинарной точностью, что значительно ускорит процесс вычислений.

1.4. Понижение числа обусловленности

Для понижения числа обусловленности можно воспользоваться матрицей предобуславливания $\boldsymbol{C^{-1}}$, состоящей из корней от обратных диагональных элементов матрицы $\boldsymbol{A}$. Для получения модифицированной матрицы $\boldsymbol{A}$, можно домножить её на матрицу $\boldsymbol{C^{-1}}$ следующим образом:

$$\tilde{\boldsymbol{A}} = \boldsymbol{C^{-1} A (C^{-1})^T}, \qquad (\mathbf{3})$$

Данный вид предобуславливания называется расщеплённым. Тогда исходная СЛАУ модифицируется следующим образом:

$$\boldsymbol{C^{-1}A(C^{-1})^T C^T x = C^{-1}b},$$

$$\boldsymbol{\widetilde{A}y = \widetilde{b}}$$

где $\boldsymbol{\widetilde{A} = C^{-1}A(C^{-1})^T}$, $\boldsymbol{y = C^T x}$, $\boldsymbol{\widetilde{b} = C^{-1}b}$.

Результат предобуславливания показан на рисунке 2.

```
Ввод [11]: Cinv=np.matrix(np.diag(np.sqrt(np.diag(A)))).I
           A_=Cinv*A*Cinv.T
           b_=Cinv*b
           lg.cond(A_)

Out[11]: 1160.4958
```

Рисунок 2 – число обусловленности матрицы $\boldsymbol{\widetilde{A}}$.

Таким образом число обусловленности модифицированной матрицы равно 1160.4958, что на два порядка меньше, чем число обусловленности исходной матрицы $\boldsymbol{A}$.

1.5. Разработка функции jacobi

В матричном виде метод Якоби предполагает разложение матрицы $\boldsymbol{A}$ на диагональную, нижнюю треугольную и верхнюю треугольную составляющие:

$$\boldsymbol{A = D + L + U}$$

Тогда СЛАУ примет вид:

$$\boldsymbol{(D + L + U)x = b}$$

$$\boldsymbol{Dx = b - (L + U)x}$$

С каждой итерацией k+1 находится решение с опорой на решение k-ой итерации:

$$\boldsymbol{x^{k+1} = D^{-1}b - D^{-1}(L + U)x^k}$$

Из данного соотношения мы получаем матрицу $\boldsymbol{T}$ и вектор $\boldsymbol{c}$ в контексте метода простой итерации:

$$\boldsymbol{T_J = D^{-1}(L + U)}$$

$$\boldsymbol{c_J = D^{-1}b}$$

Условие сходимости – спектральный радиус матрицы $\boldsymbol{T_J}$ должен быть меньше единицы. Спектральный радиус – максимальный модуль собственного числа. Для матриц $\boldsymbol{A}$ и $\widetilde{\boldsymbol{A}}$ спектральный радиус матрицы $\boldsymbol{T_J}$ равен 2.5600646, что больше единицы, а значит метод сходиться не будет и стоит также предусмотреть его остановку после определённого количества итераций. Например, пока среднеквадратичная норма относительной погрешности не станет больше 10^n при этом минимальным количеством количеством итераций будет 10, а также для порядка следует предусмотреть остановку, если погрешность станет меньше определённого эпсилон.

Листинг 3 – Функция, возвращающая решение СЛАУ, полученное с помощью метода Якоби, а также среднеквадратичные нормы невязки и абсолютной погрешности каждой итерации.

```
def jacobi(A, b, n=3, pred=False, ones=False):
    min_it=10
    stop=10**n
    D = np.matrix(np.diag(np.diag(A)))
    Dinv=D.I
    T = Dinv * (D-A)
    if ones:
        x_1 = np.ones_like(b)
        if pred: x_1=Cinv.T*x_1
    else: x_1 = np.zeros_like(b)
    c = Dinv * b
    rel = []
    nev = []
    eps=1e-15
    while True:
        x=c-T*x_1
        d=lg.norm(abs(x - x_1))/lg.norm(abs(x))
        x_1=x
        if pred:
            rel.append(lg.norm(Cinv.T*x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(C*b-C*A*(C).T*Cinv.T*x)/np.sqrt(lenx))
        else:
            rel.append(lg.norm(x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(b-A*x)/np.sqrt(lenx))
        if (rel[-1]>stop or d<eps) and (len(rel)>min_it): break
    if pred: return Cinv.T*x, rel, nev
    return x, rel, nev
```

На рисунках 3 и 4 изображены зависимости среднеквадратичной нормы абсолютной погрешности и невязки от номера итерации и от начального приближения. В случае (0) в качестве начального приближения использовался нулевой вектор, в случае (1) – вектор, состоящий из единиц.

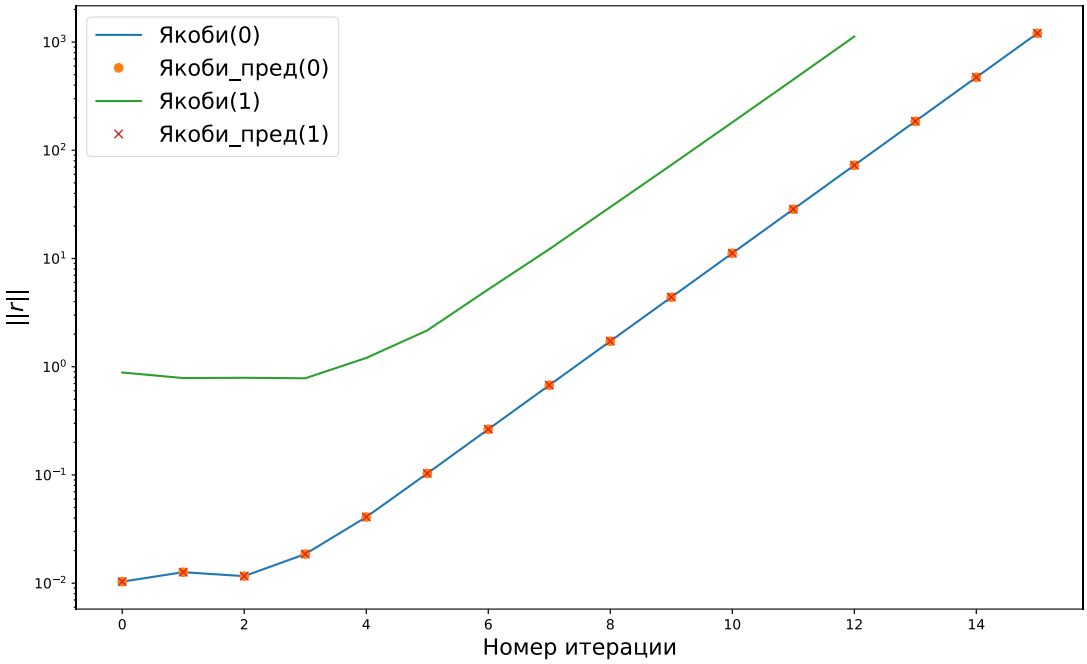

Рисунок 3 – график среднеквадратичной нормы абсолютной погрешности

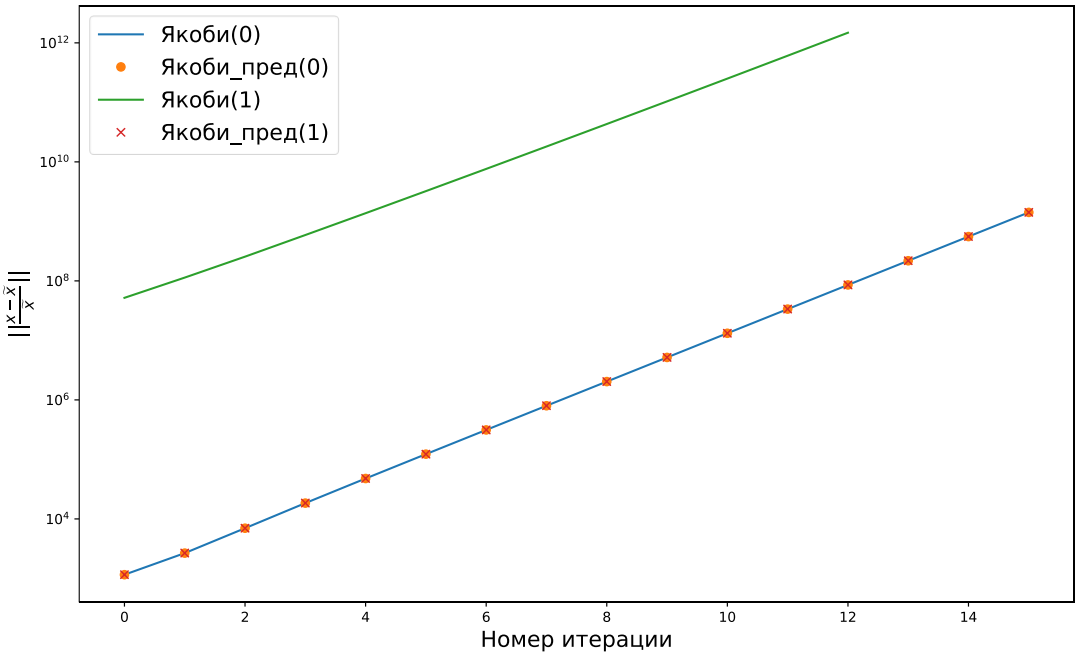

Рисунок 4 – график среднеквадратичной нормы невязки

Для данной СЛАУ метод Якоби расходится. Можно заметить, что предобуславливание практически не влияет на погрешность метода Якоби. При этом, можно заметить, что при отсутствии предобуславливания и ненулевом начальном приближении оба графика смещаются, но имеют такой же вид. Это

означает, что скорость сходимости от числа обусловленности не зависит, но низкое число обусловленности может «спасти» в случае неудачного начального приближения. В дальнейшем, на общих графиках, будут использоваться результаты, где начальным приближением является единичный вектор.

1.6. Разработка функции gauss_seidel

Метод Гаусса-Зейделя является улучшением методом Якоби и также опирается на разложение матрицы $\boldsymbol{A}$ на диагональную, нижнюю треугольную и верхнюю треугольную составляющие:

$$\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}$$

Тогда СЛАУ примет вид:

$$(\boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} = \boldsymbol{b}$$
$$(\boldsymbol{L} + \boldsymbol{D})\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}$$

С каждой итерацией k+1 находится решение с опорой на решение k-ой итерации:

$$\boldsymbol{x}^{\boldsymbol{k+1}} = (\boldsymbol{L} + \boldsymbol{D})^{-1}\boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{D})^{-1}\boldsymbol{U}\boldsymbol{x}^{\boldsymbol{k}}$$

Из данного соотношения мы получаем матрицу $\boldsymbol{T}$ и вектор $\boldsymbol{c}$ в контексте метода простой итерации:

$$\boldsymbol{T}_G = (\boldsymbol{L} + \boldsymbol{D})^{-1}\boldsymbol{U}$$
$$\boldsymbol{c}_G = (\boldsymbol{L} + \boldsymbol{D})^{-1}\boldsymbol{b}$$

Условие сходимости такое же, как и для метода Якоби – спектральный радиус матрицы $\boldsymbol{T}_G$ должен быть меньше единицы. Для матриц $\boldsymbol{A}$ и $\widetilde{\boldsymbol{A}}$ спектральный радиус матрицы $\boldsymbol{T}_G$ равен 0.99386, что немного, но меньше единицы, а значит метод будет медленно сходиться для обеих СЛАУ. В процессе тестирования функции было установлено, что происходит абсолютно идентичное случаю с методом Якоби поведение, за исключением лишь того, что решение сходится, поэтому график будет показан уже в следующем разделе. Также из функции был убран параметр ones, так как выводы относительно него уже сделаны, а на дальнейший анализ он не повлияет.

Листинг 4 – Функция, возвращающая решение СЛАУ, полученное с помощью метода Гаусса-Зейделя, а также среднеквадратичные нормы невязки и абсолютной погрешности каждой итерации.

```
def gauss_seidel(A, b, pred=False):
    min_it=10
    D = np.matrix(np.diag(np.diag(A)))
    U = np.triu(A)-D
    L = np.tril(A)-D
    LDinv=(L+D).I
    T = LDinv*U
    x_1 = np.ones_like(b)
    c = LDinv * b
    rel = []
    nev = []
    eps=10e-5
    while True:
        x=c-T*x_1
        x_1=x
        if pred:
            rel.append(lg.norm(Cinv.T*x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(C*b-C*A*(C).T*Cinv.T*x)/np.sqrt(lenx))
        else:
            rel.append(lg.norm(x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(b-A*x)/np.sqrt(lenx))
        if (rel[-1]<eps) and (len(rel)>min_it): break
    if pred: return Cinv.T*x, rel, nev
    return x, rel, nev
```

1.7. Анализ реализованных методов

На рисунках 5 и 6 изображены графики зависимости среднеквадратичной нормы абсолютной погрешности и невязки от номера итерации для каждого из методов.

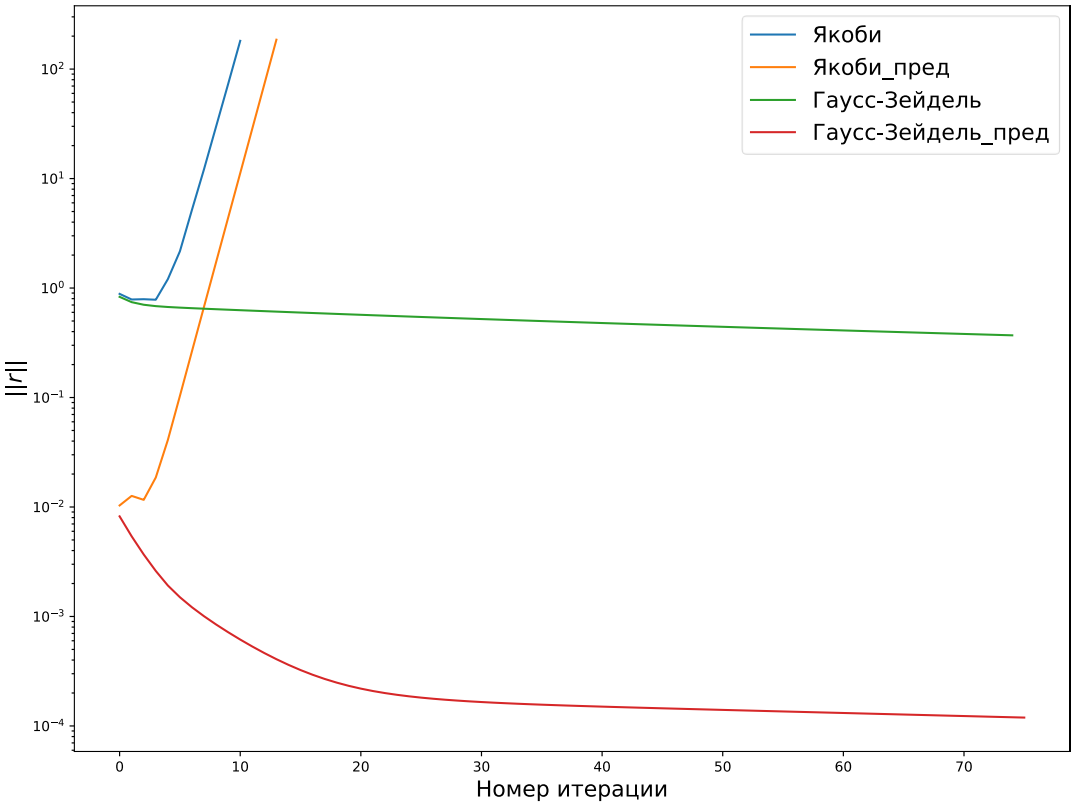

Рисунок 5 – график среднеквадратичной нормы абсолютной погрешности

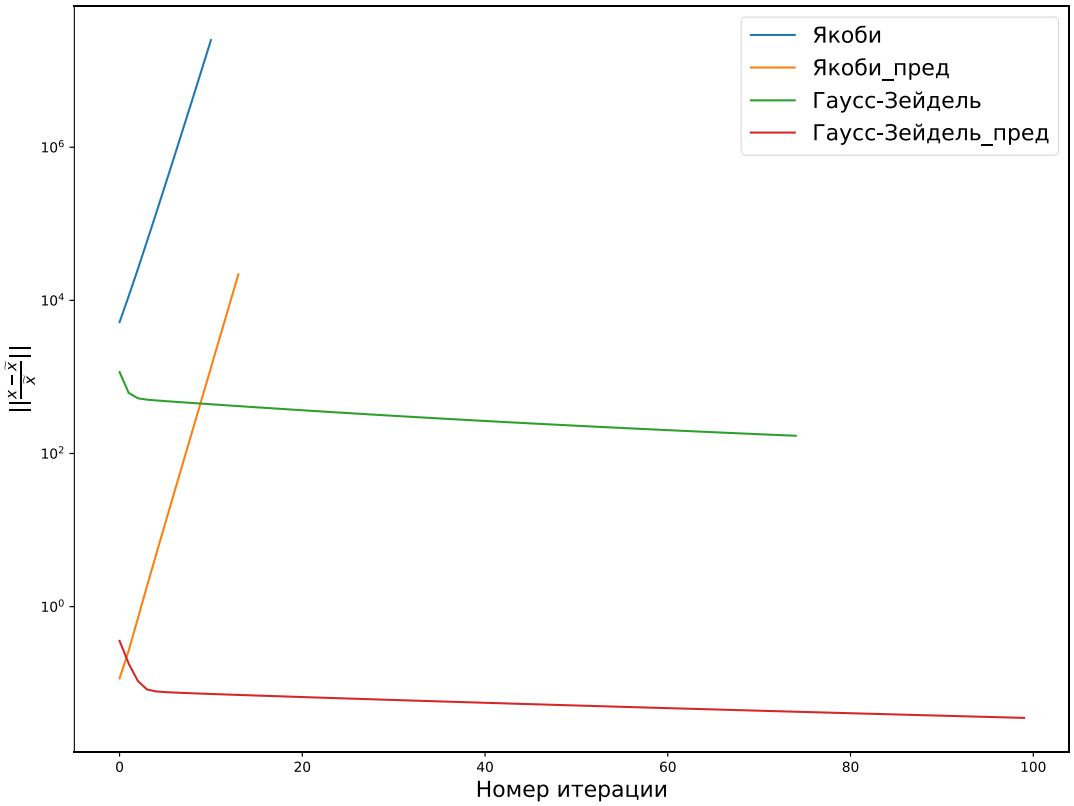

Рисунок 6 – график среднеквадратичной нормы невязки

По данным графикам, а также по предыдущим пунктам можно сделать вывод, что скорость сходимости от числа обусловленности не зависит. Однако из предыдущих двух пунктов можно сделать вывод, что меньшее число обусловленности матрицы помогает в случае, если выбрано неудачное начальное приближение. В данном случае для обоих методов снижение числа обусловленности при неудачных начальных условиях позволяет прийти к тому же результату, что и с большим числом обусловленности при близких начальных условиях. Также можно сделать вывод, что результаты без предобуславливания получились такими из-за вычислительной неустойчивости, так как, в случае метода Гаусса-Зейделя графики стремятся не к нулю, а к некоторым числам, а значит само решение сходится к неправильному. Также можно сделать вывод, что у метода Гаусса-Зейделя (а быть может, и у метода Якоби), существует оптимальное количество итераций, до достижения которого погрешность уменьшается довольно быстро, а после начинает сходиться к решению очень медленно.

2. Продвинутая часть

2.1. Разработка функции visualize_matrix

Для визуализации матрицы использовалась функция *matplotlib.pyplot.imshow*. В качестве цветовой палитры использовалась bwr, так как она наиболее точно подходит для требуемой задачи – визуализировать матрицу так, чтобы элементы, близкие к нулевым были белыми. Также палитра позволяет увидеть знак числа, что тоже может оказаться полезным.

Рисунок 7 – цветовая палитра bwr

При этом крайней правой точке палитры будет соответствовать модуль наибольшего по модулю числа – а крайней левой – модуль наибольшего по модулю числа, взятый со знаком минус. Тогда числа, близкие к нулю, автоматически попадут под белую часть данной палитры.

Листинг 5 – Функция, визуализирующая матрицу

```
def visualize_matrix(A): #matplotlib.colors as color
    max_mat=max(np.max(A), abs(np.min(A)))
    norm=color.Normalize(vmin=-max_mat, vmax=max_mat)
    plt.matshow(A, cmap='bwr', norm=norm)
```

2.2. Визуализация и анализ матрицы

С помошью функции *visualize_matrix* были получены изображения матриц $\boldsymbol{A}$ и $\widetilde{\boldsymbol{A}}$.

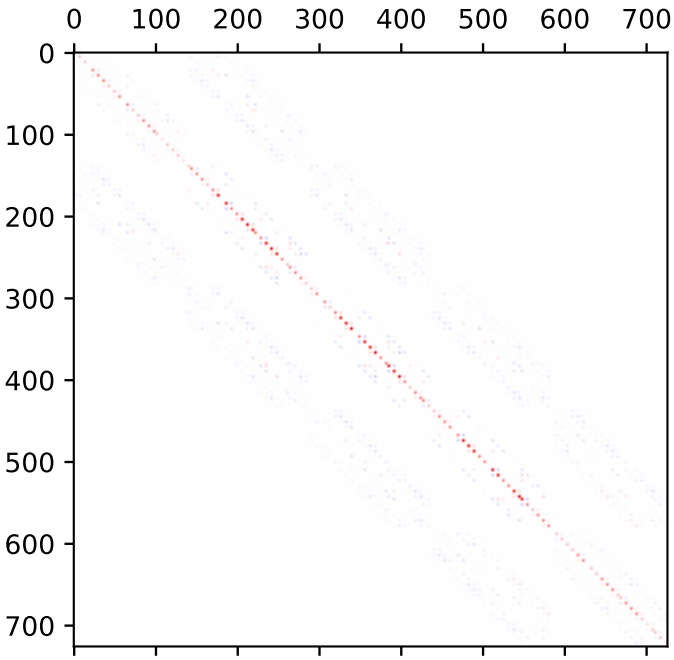

Рисунок 8 – визуализация матрицы $\boldsymbol{A}$

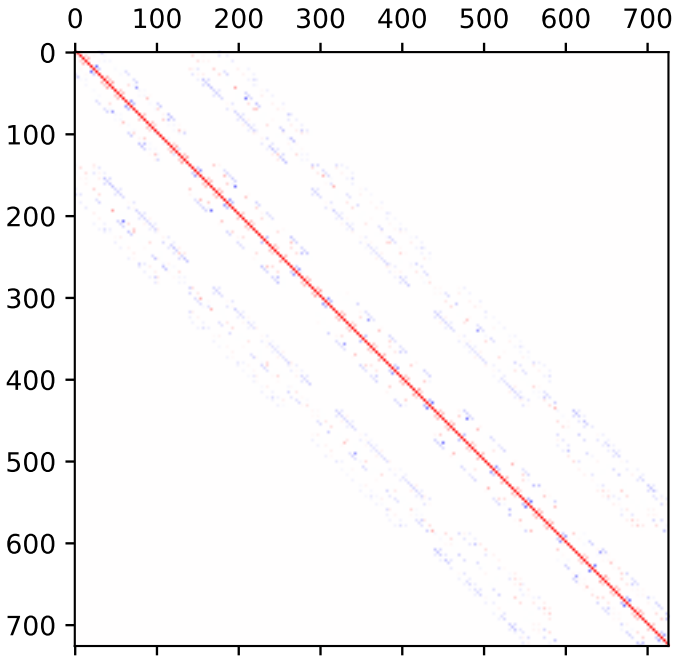

Рисунок 9 – визуализация матрицы $\widetilde{\boldsymbol{A}}$

Визуализация хорошо демонстрирует результат работы предобуславливания – элементы на главной диагонали выровнялись между собой, всё также оставаясь максимальными. На изображениях видно, что обе матрицы являются разреженными, и как частный случай, ленточными. Про матрицу $\boldsymbol{A}$ также известно, что она симметрично, значит ширину ленты можно найти по следующей формуле:

$$w = 1 + 2 \cdot k,$$

где k – индекс самой верхней ненулевой диагонали. В случае функции numpy.diag для главной диагонали $k = 0$, для диагоналей выше главной k>0.

```
Ввод [132]: for i in range(1,A.shape[0]//2):
                for j in np.diag(A, k=i):
                    if abs(j)>1e-15:
                        w=i
                        break
            2*w+1

Out[132]: 377
```

Рисунок 10 – вычисление ширины ленты матрицы $\boldsymbol{A}$.

Таким образом, ширина ленты матрицы $\boldsymbol{A}$ равна 377.

2.3. Разработка класса SparseMatrix

Для хранения разреженной матрицы в формате CSR (сжатое хранение строкой). В этом классе исходная матрица представляется в виде трёх массивов:

- массив _nonzero, хранящий значения ненулевых элементов
- массив _colnums, хранящий номера ненулевых столбцов, соответствующих значениям из _nonzero
- массив _count, i-й элемент которого хранит количество ненулевых элементов в строках до i-1 включительно, а первый элемент хранит 0.

Для умножения матрицы на вектор был перегружен оператор __mul__. Других операций в методе сопряжённых градиентов не требуется. В результате умножения матрицы на вектор формируется вектор res. При расчёте используется внешний цикл по строкам матрицы и внутренний по элементам строки. Во внутреннем цикле индекс k меняется в диапазоне от self._count[i] до self._count[i+1], что соответствует индексам элементов _nonzero текущей строки. Результатом является Nx1, то есть, вектор-столбец.

Листинг 6 – Класс, реализующий сжатое хранение строкой для матрицы A, а также умножение такой матрицы на вектор

```
class SparseMatrix:
    def __init__(self,A):
        self._nonzero=[]
        self._colnums=[]
        self._count=[]
        self._count.append(0)
        for i in range(0, A.shape[0]):
            for j in range(0, A.shape[1]):
                if (abs(A[i,j]) > 1e-15):
                    self._nonzero.append(A[i,j])
                    self._colnums.append(j)
            self._count.append(len(self._colnums))
        self._nonzero=np.array(self._nonzero)
        self._colnums=np.array(self._colnums)
        self._count=np.array(self._count)
    def __mul__(self, b):
        n=len(self._count)-1
        res=np.zeros(n)
        for i in range(n):
            for k in range(self._count[i], self._count[i+1]):
                res[i] = res[i] + self._nonzero[k]*b[self._colnums[k]]
        return np.matrix(res).T
```

2.4. Разработка функции conjugate_gradient_method

Метод сопряжённых градиентов предполагает использование следующих рекурсивных уравнений:

$$t_k = \frac{\langle \boldsymbol{r}^{(k-1)}, \boldsymbol{r}^{(k-1)} \rangle}{\langle \boldsymbol{v}^{(k)}, \boldsymbol{A}\boldsymbol{v}^{(k)} \rangle},$$
$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + t_k \boldsymbol{v}^{(k)},$$
$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - t_k \boldsymbol{A}\boldsymbol{v}^{(k)},$$
$$s_k = \frac{\langle \boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)} \rangle}{\langle \boldsymbol{r}^{(k-1)}, \boldsymbol{r}^{(k-1)} \rangle},$$
$$\boldsymbol{v}^{(k+1)} = \boldsymbol{r}^{(k)} + s_k \boldsymbol{v}^{(k)},$$

где $\boldsymbol{x}^{(0)}, \boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$ и $\boldsymbol{v}^{(1)} = \boldsymbol{r}^{(0)}$ являются начальными условиями. Как и в предыдущих методах, за начальные условия заведомо взят вектор, состоящий из единиц, который позволяет явно продемонстрировать разницу между наличием предобуславливания и его отсутствием. По данным формулам была разработана функция *conjugate_gradient_method*. An-

Листинг 7– Функция, возвращающая решение СЛАУ, полученное с помощью метода сопряжённых градиентов, а также среднеквадратичные нормы невязки и абсолютной погрешности каждой итерации.

```
def conjugate_gradient_method(A, b, An, pred=False): #An - матрица в оригинале
    max_it=100   #An не нужна для самого метода, но нужна для фиксации невязки
    x_1 = np.matrix(np.ones_like(b))           #в случае с предобуславливанием
    r_1 = b - A*x_1
    v = r_1.copy()
    rel = []
    nev = []
    for i in range(max_it):
        t = float((r_1.T * r_1) / (v.T * (A * v)))
        x = x_1 + t * v
        r = r_1 - t * (A * v)
        s = float((r.T*r) / (r_1.T*r_1))
        v = r + s * v
        if pred:
            rel.append(lg.norm(Cinv.T*x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(C*b-C*An*(C).T*Cinv.T*x)/np.sqrt(lenx))
        else:
            rel.append(lg.norm(x - x_cor)/np.sqrt(lenx))
            nev.append(lg.norm(b-An*x)/np.sqrt(lenx))
        r_1 = r
        x_1 = x

    if pred: return  Cinv.T*x,rel,nev
    return x,rel,nev
```

2.5. Анализ полученных решений

Графики полученных зависимостей среднеквадратичной нормы абсолютной погрешности и невязки для метода сопряжённых градиентов были добавлены на графики из п. 1.7. Полученный результат изображён на рисунках 11 и 12.

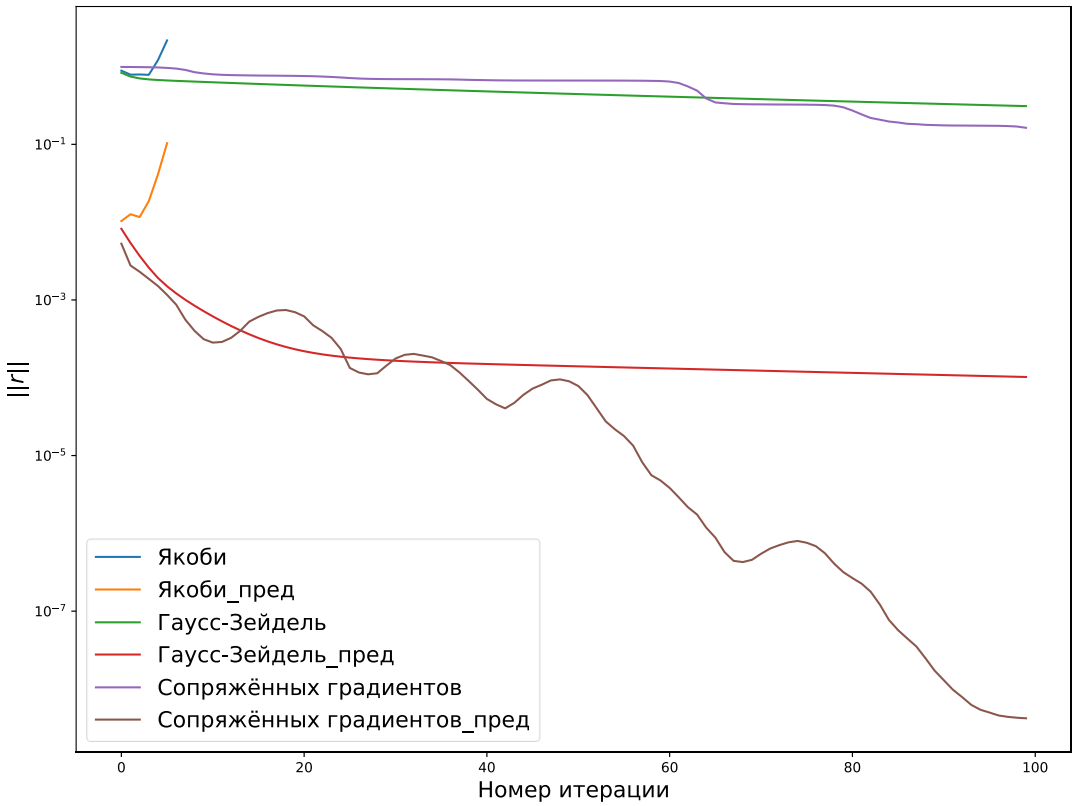

Рисунок 11 – график среднеквадратичной нормы абсолютной погрешности

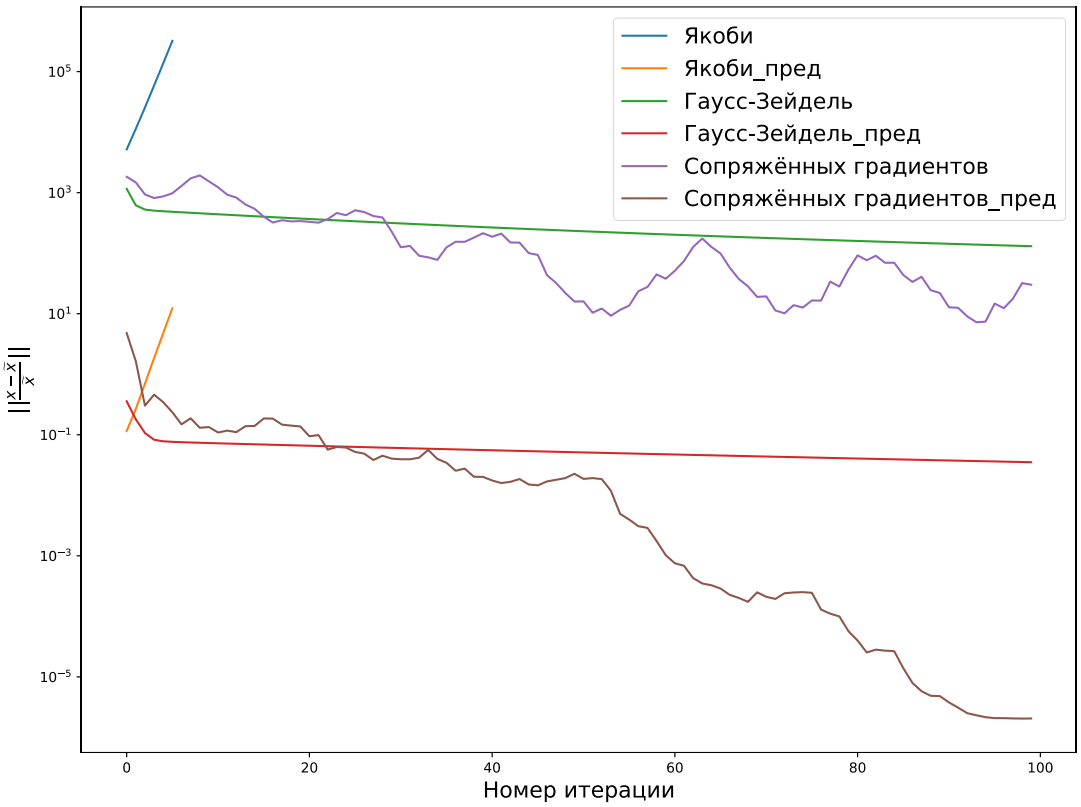

Рисунок 12 – график среднеквадратичной нормы невязки

По данным графикам можно сделать вывод, что предобуславливание оказывает абсолютно аналогичное влияние и на метод сопряжённых градиентов. Самым неэффективным для данной СЛАУ является метод Якоби, так как он расходится. Быстрее и эффективнее всех сходится решение, полученное с помощью метода сопряжённых градиентов. В данном случае большую роль играет тот факт, что в методе Гаусса-Зейделя спектральный радиус матрицы $\boldsymbol{T}$ очень близок к единице, за счёт чего решение сходится очень медленно. Однозначно, для данной матрицы коэффициентов наилучшим методом является метод сопряжённых градиентов с использованием предобуславливания.

Заключение

В ходе выполнения данной лабораторной работы были изучена различные итерационные методы решения СЛАУ. Они позволяют компенсировать ту погрешность вычислений, которая появляется при использовании прямых методов для больших матриц. Было установлено, что число обсусловленности влияет на точность операций над матрицами, что может приводить к большим невязкам полученных решений. Также ни в одном из методов понижение числа обусловленности не сделало хуже, поэтому полезно его понижать (особенно, при использовании прямых и полупрямых методов)

На скорость сходимости итерационных методов главным образом влияет спектральный радиус соответствующих матриц $\boldsymbol{T}$. В данном случае наименее устойчивым оказался метод Якоби – решение, полученное с помощью данного метода, расходится, как и предполагалось. Метод Гаусса-Зейделя оказался более устойчивым, и, как и предполагалось, медленно, но сходился к правильному решению.

Также было выяснено, что графическое отображение матриц позволяет наглядно определить их тип, что во многих случаях позволяет быстрее подобрать правильный метод решения эмпирически. Например, было выяснено, что матрица разреженная, а значит для неё эффективно использовать особый формат хранения – сжатое хранение строкой, который также эффективен при большом количестве умножений матрицы на вектор.

Список использованных источников

1. **Першин А.Ю.** *Лекции по вычислительной математике (черновик)*. [archrk6.bmstu.ru] // Кафедра РК6 МГТУ им. Н.Э. Баумана, Москва, 2020, 145.
2. **Першин А.Ю.** *Вычислительная математика, лекция №13*. Видеохостинг «YouTube» [https://www.youtube.com/watch?v=vEsBYK4aVkA] // (дата обращения (04.06.2021).
3. **Першин А.Ю.** *Вычислительная математика, лекция №14*. Видеохостинг «YouTube» [https://www.youtube.com/watch?v=vF118q0Npkc] // (дата обращения (04.06.2021).
4. **Першин А.Ю.** *Вычислительная математика, лекция №15*. Видеохостинг «YouTube» [https://www.youtube.com/watch?v=PUwJlsvnd2U] // (дата обращения (04.06.2021).
5. Colormap Normalization [https://matplotlib.org/stable/tutorials/colors/colormapnorms.html] // (дата обращения (05.04.2021).
6. Разреженные матрицы (Sparse Matrix) [https://python-school.ru/sparse-matrix/] // (дата обращения (05.04.2021).