МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра вычислительной математики

Отчёт по лабораторной работе №2, вариант 3

(«Методы вычислений»)

Харченко Роман Кириллович

Студента 2 курса 12 группы

Преподаватель

Бондарь Иван Васильевич

Минск 2020

Условие:

Задание 3. Метод Гаусса с выбором ГЭ по всей матрице

1. Написать программу, которая решает СЛАУ A x=b и вычисляет определитель матрицы A методом Гаусса с выбором главного элемента по всей матрице. Применить программу к следующим ниже входным данным и вывести результат.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 \\ 1 & 3 & 9 & 27 & 81 & 243 & 729 & 2187 & 6561 \\ 1 & 4 & 16 & 64 & 256 & 1024 & 4096 & 16384 & 65536 \\ 1 & 5 & 25 & 125 & 625 & 3125 & 15625 & 78125 & 390625 \\ 1 & 6 & 36 & 216 & 1296 & 7776 & 46656 & 279936 & 1679616 \\ 1 & 7 & 49 & 343 & 2401 & 16807 & 117649 & 823543 & 5764801 \\ 1 & 8 & 64 & 512 & 4096 & 32768 & 262144 & 2097152 & 16777216 \\ 1 & 9 & 81 & 729 & 6561 & 59049 & 531441 & 4782969 & 43046721 \end{pmatrix} \quad b = \begin{pmatrix} 9 \\ 511 \\ 9841 \\ 87381 \\ 488281 \\ 2015539 \\ 6725601 \\ 19173961 \\ 48427561 \end{pmatrix}$$

$$A = \begin{pmatrix} -3 & -1 & -1 & 1 & 4 & 2 & 3 & -4 & -1 \\ -3 & -1 & 4 & 1 & 0 & 3 & -5 & 5 & 5 \\ -6 & -2 & 3 & 5 & 3 & -3 & -4 & -3 & 1 \\ -12 & -4 & 6 & 7 & 2 & 0 & 1 & -2 & -1 \\ -24 & -8 & 12 & 14 & 9 & 4 & 4 & -1 & 5 \\ -48 & -16 & 24 & 28 & 18 & 6 & -2 & 1 & -2 \\ -96 & -32 & 48 & 56 & 36 & 12 & -3 & 2 & 4 \\ -192 & -64 & 96 & 112 & 72 & 24 & -6 & -2 & 5 \\ 0 & -1 & 3 & 0 & 2 & 2 & -5 & -3 & 4 \end{pmatrix} \quad b = \begin{pmatrix} 8 \\ 79 \\ -27 \\ 18 \\ 186 \\ 206 \\ 491 \\ 907 \\ 6 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 0 & -5 & -3 & 1 & 1 & 3 & 2 & 2 \\ 2 & 0 & -5 & -5 & 2 & -1 & 4 & -2 & -3 \\ 4 & 0 & -10 & -1 & 5 & -3 & 1 & -5 & -5 \\ 8 & 0 & -20 & -9 & 5 & 1 & 1 & -5 & 4 \\ 16 & 0 & -40 & -18 & 13 & 1 & -1 & 0 & -3 \\ 32 & 0 & -80 & -36 & 26 & -1 & 0 & -1 & -5 \\ 64 & 0 & -160 & -72 & 52 & -2 & 8 & 0 & -2 \\ 128 & 0 & -320 & -144 & 104 & -4 & 16 & -11 & 5 \\ -1 & 1 & 1 & 3 & 0 & -3 & 0 & -5 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 41 \\ -44 \\ -101 \\ -54 \\ -139 \\ -281 \\ -418 \\ -843 \\ -15 \end{pmatrix}$$

2. Найти точное решение указанных систем с использованием библиотеки SymPy или аналогичного программного обеспечения и сравнить результаты.

3. Вычислить число обусловленности в максимум-норме матрицы A из второго тестового задания. Что это означает на практике? Путем решения нескольких СЛАУ с возмущенным вектором b подтвердите связь между числом обусловленности и относительными погрешностями начальных данных и решения.

4. Проведите экспериментальное исследование скорости решения СЛАУ в зависимости от размерности системы, используя для тестов матрицу A и вектор b со случайными числами. Постройте график зависимости времени работы от размерности. Систему какой размерности ваша программа на вашем компьютере может решить за одну минуту?

Задание 1:

Я написал программу на Python, которая позволяет решить систему линейных алгебраических уравнений вида Ax = b методом Гаусса (с выбором главного элемента по всей матрице), а также вычислить определитель матрицы A, а также протестировал её работу на вышеуказанных примерах.

Идея состоит в том, чтобы выбрать среди всей матрицы максимальный (по модулю) элемент, после чего обменять сначала строки, а потом столбцы матрицы так, чтобы этот самый элемент оказался на позиции главного элемента на главной диагонали. Это нужно для того, чтобы улучшить наше число обусловленности, чтобы программа вычисляла решение точнее.

Однако, когда мы меняем строки в матрице, нам также нельзя забывать про то, чтобы поменять элементы ещё и в векторе b, чтобы не потерять правильный ответ. Также необходимо запомнить перестановку столбцов, чтобы в конце расставить найденные компоненты вектора X в нужном порядке. Всё это в программе учтено.

Да, стоит отметить, что я ручками прописали все данные матрицы и вектора из условия для того, чтобы их потом протестировать. Определители матрицы считал с помощью библиотек Python – linalg.det(A).

Код программы:

```
import numpy as np
from numpy.linalg import linalg
import matplotlib.pyplot as plt
import time

SIZE = 9

# Ответ: X1 = [ 1, 1, 1, 1, 1, 1, 1, 1, 1 ]
X1_answer = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1])
# https://matrixcalc.org/slu.html#solve-using-Gaussian-
elimination%28%7B%7B1,1,1,1,1,1,1,1,1,9%7D,%7B1,2,4,8,16,32,64,128,256,511%7D,%7
B1,3,9,27,81,243,729,2187,6561,9841%7D,%7B1,4,16,64,256,1024,4096,16384,65536,87
381%7D,%7B1,5,25,125,625,3125,15625,78125,390625,488281%7D,%7B1,6,36,216,1296,77
76,46656,279936,1679616,2015539%7D,%7B1,7,49,343,2401,16807,117649,823543,576480
1,6725601%7D,%7B1,8,64,512,4096,32768,262144,2097152,16777216,19173961%7D,%7B1,9
,81,729,6561,59049,531441,4782969,43046721,48427561%7D%7D%29
A1 = np.array([[1., 1., 1., 1., 1., 1., 1., 1., 1.],
               [1., 2., 4., 8., 16., 32., 64., 128., 256.],
               [1., 3., 9., 27., 81., 243., 729., 2187., 6561.],
               [1., 4., 16., 64., 256., 1024., 4096., 16384., 65536.],
               [1., 5., 25., 125., 625., 3125., 15625., 78125., 390625.],
               [1., 6., 36., 216., 1296., 7776., 46656., 279936., 1679616.],
               [1., 7., 49., 343., 2401., 16807., 117649., 823543., 5764801.],
               [1., 8., 64., 512., 4096., 32768., 262144., 2097152., 16777216.],
               [1., 9., 81., 729., 6561., 59049., 531441., 4782969.,
43046721.]])

b1 = np.array([9., 511., 9841., 87381., 488281., 2015539., 6725601., 19173961.,
48427561.])

# Ответ: X2 = [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
X2_answer = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
# https://matrixcalc.org/slu.html#solve-using-Gaussian-elimination%28%7B%7B-3,-
1,-1,1,4,2,3,-4,-1,8%7D,%7B-3,-1,4,1,0,3,-5,5,5,79%7D,%7B-6,-2,3,5,3,-3,-4,-
3,1,-27%7D,%7B-12,-4,6,7,2,0,1,-2,-1,18%7D,%7B-24,-8,12,14,9,4,4,-
1,5,186%7D,%7B-48,-16,24,28,18,6,-2,1,-2,206%7D,%7B-96,-32,48,56,36,12,-
3,2,4,491%7D,%7B-192,-64,96,112,72,24,-6,-2,5,907%7D,%7B0,-1,3,0,2,2,-5,-
3,4,6%7D%7D%29
A2 = np.array([[-3., -1., -1., 1., 4., 2., 3., -4., -1.],
               [-3., -1., 4., 1., 0., 3., -5., 5., 5.],
               [-6., -2., 3., 5., 3., -3., -4., -3., 1.],
               [-12., -4., 6., 7., 2., 0., 1., -2., -1.],
               [-24., -8., 12., 14., 9., 4., 4., -1., 5.],
               [-48., -16., 24., 28., 18., 6., -2., 1., -2.],
               [-96., -32., 48., 56., 36., 12., -3., 2., 4.],
               [-192., -64., 96., 112., 72., 24., -6., -2., 5.],
               [0., -1., 3., 0., 2., 2., -5., -3., 4.]])

b2 = np.array([8., 79., -27., 18., 186., 206., 491., 907., 6.])

# Ответ: вообще говоря, целое множество решений: X3 = [ -13/2 + 5/2*x3, -5/2 +
3/2*x3, x3, 4, 5, 6, 7, 8, 9 ]
X3_answer = np.array(['-13/2 + 5/2*x3', '-5/2 + 3/2*x3', 'x3', '4', '5', '6',
'7', '8', '9'])
# https://matrixcalc.org/slu.html#solve-using-Gaussian-elimination%28%7B%7B2,0,-
5,-3,1,1,3,2,2,41%7D,%7B2,0,-5,-5,2,-1,4,-2,-3,-44%7D,%7B4,0,-10,-1,5,-3,1,-5,-
5,-101%7D,%7B8,0,-20,-9,5,1,1,-5,4,-54%7D,%7B16,0,-40,-18,13,1,-1,0,-3,-
139%7D,%7B32,0,-80,-36,26,-1,0,-1,-5,-281%7D,%7B64,0,-160,-72,52,-2,8,0,-2,-
418%7D,%7B128,0,-320,-144,104,-4,16,-11,5,-843%7D,%7B-1,1,1,3,0,-3,0,-5,3,-
```

```
15%7D%7D%29
A3 = np.array([[2., 0., -5., -3., 1., 1., 3., 2., 2.],
               [2., 0., -5., -5., 2., -1., 4., -2., -3.],
               [4., 0., -10., -1., 5., -3., 1., -5., -5.],
               [8., 0., -20., -9., 5., 1., 1., -5., 4.],
               [16., 0., -40., -18., 13., 1., -1., 0., -3.],
               [32., 0., -80., -36., 26., -1., 0., -1., -5.],
               [64., 0., -160., -72., 52., -2., 8., 0., -2.],
               [128., 0., -320., -144., 104., -4., 16., -11., 5.],
               [-1., 1., 1., 3., 0., -3., 0., -5., 3.]])

b3 = np.array([41., -44., -101., -54., -139., -281., -418., -843., -15.])


# Максимум-норма (максимум из суммы модулей элементов строк)
def MaximumNorma(A, size):
    max_sum = 0.
    for i in range(size):
        curr_sum = 0.
        for j in range(size):
            curr_sum += abs(A[i][j])
        if curr_sum > max_sum:
            max_sum = curr_sum
    return max_sum


# Найти число обусловленности матрицы
def ConditionNumber(A, size):
    return MaximumNorma(A, size) * MaximumNorma((linalg.inv(A)), size)


#  Функция, получающая на вход матрицу A, размерность size матрицы A, а также
вектор b, и возвращающая вектор X
def GaussSelectingMainElementThroughoutWholeMatrix(A, size, b):
    ColsTranspVector = np.arange(size)  # Вектор перестановок для столбцов
    for diag_ind in range(size - 1):
        max_element = A[diag_ind][diag_ind]
        max_el_i = diag_ind
        max_el_j = diag_ind
        for i in range(diag_ind, size):
            for j in range(diag_ind, size):
                if abs(A[i][j]) > max_element:
                    max_element = abs(A[i][j])
                    max_el_i = i
                    max_el_j = j
        A[[diag_ind, max_el_i], :] = A[[max_el_i, diag_ind], :]  # Меняем
сначала строки местами
        b[[diag_ind, max_el_i]] = b[[max_el_i, diag_ind]]  # В том числе меняем
и элементы вектора b
        A[:, [diag_ind, max_el_j]] = A[:, [max_el_j, diag_ind]]  # Потом меняем
столбцы местами
        ColsTranspVector[[diag_ind, max_el_j]] = ColsTranspVector[
            [max_el_j, diag_ind]]  # Запоминаем перестановку столбцов
        for i in range(diag_ind + 1, size):  # Зануляем все элементы ниже
главного в текущем столбце
            koeff = (-1) * A[i][diag_ind] / A[diag_ind][diag_ind]
            for j in range(diag_ind,
                           size):  # Ко всей строке и к вектору b мы вынуждены
прибавить тот коэффициент, что прибавли к A[i][diag_ind]
```

```
                A[i][j] += koeff * A[diag_ind][j]
            b[i] += koeff * b[diag_ind]
    transp_X = np.zeros(size)  # Здесь мы уже имеем заполненную в левой нижней
части нулями матрицу...
    j = size - 1
    for i in range(size - 1, -1, -1):
        for l in range(j + 1, size):
            b[i] -= A[i][l] * transp_X[l]
        transp_X[i] = b[i] / A[i][j]
        j -= 1
    X = np.zeros(size)  # Мы нашли вектор X, но его элементы пока перепутаны,
нужно переставить их по нашей перестановке
    for i in range(size):
        X[i] = transp_X[ColsTranspVector[i]]
    return X


X1 = GaussSelectingMainElementThroughoutWholeMatrix(A1, SIZE, b1)
print("Определитель первой матрицы: det(A1) =", linalg.det(A1))
print("Решение первой системы с помощью NymPy: X1 =", X1)
print("Точное решение первой системы: X1 =", X1_answer, '\n')

X2 = GaussSelectingMainElementThroughoutWholeMatrix(A2, SIZE, b2)
print("Определитель второй матрицы: det(A2) =", linalg.det(A2))
print("Решение второй системы с помощью NymPy: X2 =", X2)
print("Точное решение второй системы: X2 =", X2_answer, '\n')

X3 = GaussSelectingMainElementThroughoutWholeMatrix(A3, SIZE, b3)
print("Определитель третьей матрицы: det(A3) =", linalg.det(A3))
print("Решение третьей системы с помощью NymPy: X3 =", X3)
print("Точное решение третьей системы: X3 =", X3_answer, '\n')

print("Число обусловленности максимум-нормы для первой матрицы: X(A1) =",
ConditionNumber(A1, SIZE))
print("Число обусловленности максимум-нормы для второй матрицы: X(A2) =",
ConditionNumber(A2, SIZE))
print("Число обусловленности максимум-нормы для третьей матрицы: X(A3) =",
ConditionNumber(A3, SIZE))

print("Начало эксперимента с замерами времени работы программы на больших
системах...")
ExperimentSizeCheck = 70  # Число размерной матриц для экспериментов... Т.е.
будем смотреть матрицы размерностей { 1, 2, ..., ExperimentSizeCheck - 1,
ExperimentSizeCheck }
SizeArr = np.arange(1, ExperimentSizeCheck + 1)  # Массив абсцисс (размерностей)
для графика
TimeArr = np.zeros(ExperimentSizeCheck)  # Массив ординат (времени в секундах)
для графика
StartTime = time.time()  # Начало отсчета времени (для замера скорости
программы)
previous_time = 0
for i in range(1, ExperimentSizeCheck + 1):
    TimeArr[i - 1] = int(time.time() - StartTime) - previous_time
    previous_time = TimeArr[i - 1]
    A = np.random.rand(i, i)
    b = np.random.rand(i)
    GaussSelectingMainElementThroughoutWholeMatrix(A, i, b)
plt.plot(SizeArr, TimeArr)
plt.xlabel("Размерность системы (в единицах)")
```

```
plt.ylabel("Время решения системы (в секундах)")
plt.show()
print("Общее время работы: %s seconds" % (time.time() - StartTime))
```

После того, как я закончил писать 1-ую часть программы и тестовые системы к ней, я их немедленно протестировал. Вот решение трёх наших систем (три вектора X с девятью компонентами каждый):

```
Определитель первой матрицы: det(A1) = -5056584744963635.0
Решение первой системы с помощью NymPy: X1 = [1.00000001 1.         1.         0.99999999 1.         1.
 1.         1.         1.        ]
Точное решение первой системы: X1 = [1 1 1 1 1 1 1 1 1]

Определитель второй матрицы: det(A2) = 16199.99999999997
Решение второй системы с помощью NymPy: X2 = [1. 2. 3. 4. 5. 6. 7. 8. 9.]
Точное решение второй системы: X2 = [1 2 3 4 5 6 7 8 9]

Определитель третьей матрицы: det(A3) = 2.328581771848799e-12
Решение третьей системы с помощью NymPy: X3 = [  8.  -76.  -44.2   5.    7.    6.    4.    9.  -27.8]
Точное решение третьей системы: X3 = ['-13/2 + 5/2*x3' '-5/2 + 3/2*x3' 'x3' '4' '5' '6' '7' '8' '9']
```

Задание 2

В этом задании меня попросили проверить то, насколько точно были решены системы в моей программе. Для этого я использовал ресурс – https://matrixcalc.org/slu.html . Внутри программы над каждым примером я закомментировал ссылку на решение каждого из них, а также прописал вектора – ответы к нашим системам (которые мне выдал ресурс). На картинке вывода выше видно, что точные решения систем, которые были получены с ресурса по указанной ссылке, несколько отличаются от результатов, полученных моей программой.

Мы можем наблюдать, что решение 1-ой системы – это девять единиц, однако в выводе программе некоторые компоненты не совсем точно равны единице. То есть видно, что программа не без погрешностей. Что касается 2-ой системы, то она решилась довольно неплохо и достаточно точно. А вот в 3-ей системе все совсем интересно. Как подсказал мне ресурс, в 3-ей системе мне решении методом Гаусса вообще получается строка, состоящая из одних нулей. Это значит, что решений вообще может быть бесконечное множество. Кстати, определитель матрицы в этой системе, по идее, равен нулю, что также свидетельствует о неоднозначности решения. Однако моя программа рассчитала определитель матрицы и решение системы. Мы можем видеть, что значение определителя очень близко к нолику, но все-таки не равно ему, что говорит о погрешностях. Соответственно, неудивительно, что программа и решение для системы посчитала… Кстати, можно видеть, что все элементы (4, 5, 6, 7, 8, 9) после неопределенных элементов программа посчитала точно также, как и

ресурс. В целом, можно сказать, что решение СЛАУ с помощью NumPy может быть осложнено погрешностями.

Задание 3

Здесь меня попросили посчитать число обусловленности для максимум-нормы матрицы из второй системы. Для этого я прописал функцию поиска максимум-нормы, которая считает наибольшую по модулю сумму среди элементов строк. Соответственно, я выбирал максимум-норму среди всех строк в матрице (вы можете наблюдать эту функцию в моей программе). Так вот, я взял обратную матрицу от матрицы 2-ой системы (с помощью linalg.inv(A)), после чего посчитал число обусловленности, совершив произведение максимум-нормы для матрицы A2 и максимальной из трёх норм для обратной к матрице A2 матрицы. На практике число обусловленности показывает то, во сколько раз относительная погрешность в ответе может превышать относительную погрешность начальных данных.

```
# Максимум-норма (максимум из суммы модулей элементов строк)
def MaximumNorma(A, size):
    max_sum = 0.
    for i in range(size):
        curr_sum = 0.
        for j in range(size):
            curr_sum += abs(A[i][j])
        if curr_sum > max_sum:
            max_sum = curr_sum
    return max_sum


# Найти число обусловленности матрицы
def ConditionNumber(A, size):
    return MaximumNorma(A, size) * MaximumNorma((linalg.inv(A)), size)


print("Число обусловленности максимум-нормы для первой матрицы: X(A1) =",
ConditionNumber(A1, SIZE))
print("Число обусловленности максимум-нормы для второй матрицы: X(A2) =",
ConditionNumber(A2, SIZE))
print("Число обусловленности максимум-нормы для третьей матрицы: X(A3) =",
ConditionNumber(A3, SIZE))
```

Позже, протестировав программу на разных матрицах и возмущённых векторах, я пришёл к выводу, что связь между числом обусловленности и относительной погрешностью начальных данных с решением действительно наблюдается. А именно, я сделал следующее. Я рассчитал эту самую максимум-норму не только для нашей второй матрицы (которая посчиталась довольно хорошо по сравнению с точным ответом), но и для первой и третьей матриц. При выводе программа вывела интересные результаты:

```
Число обусловленности максимум-нормы для первой матрицы: X(A1) = 17273636865.50361
Число обусловленности максимум-нормы для второй матрицы: X(A2) = 19663.449999999986
Число обусловленности максимум-нормы для третьей матрицы: X(A3) = 2.6373079417881628e+20
```

Здесь мы можем наблюдать, что у нашей второй матрицы относительно не большое число обусловленности по сравнению с остальными матрицами. И не удивительно, т.к. ответ ко второй системе у нас получился достаточно точным. В первой матрице число обусловленности уже похуже (оно значительно больше, чем у второй матрицы), и мы, при решении первой системе, соответственно допустили некоторые неточности (там ответом являлся вектор из единичек, тогда как у нас проскакивали такие числа, как 0.999999 и 0.000001. А в третьей матрице число обусловленности настолько огромное, что совсем не удивительно, что третья система (которая вообще на самом деле имеет множество решений) у нас решилась достаточно плохо. Таким образом, чётко видно, что чем больше число обусловленности, тем выше вероятность того, что мы получим плохое решение при подсчёте ответа на наших начальных данных.

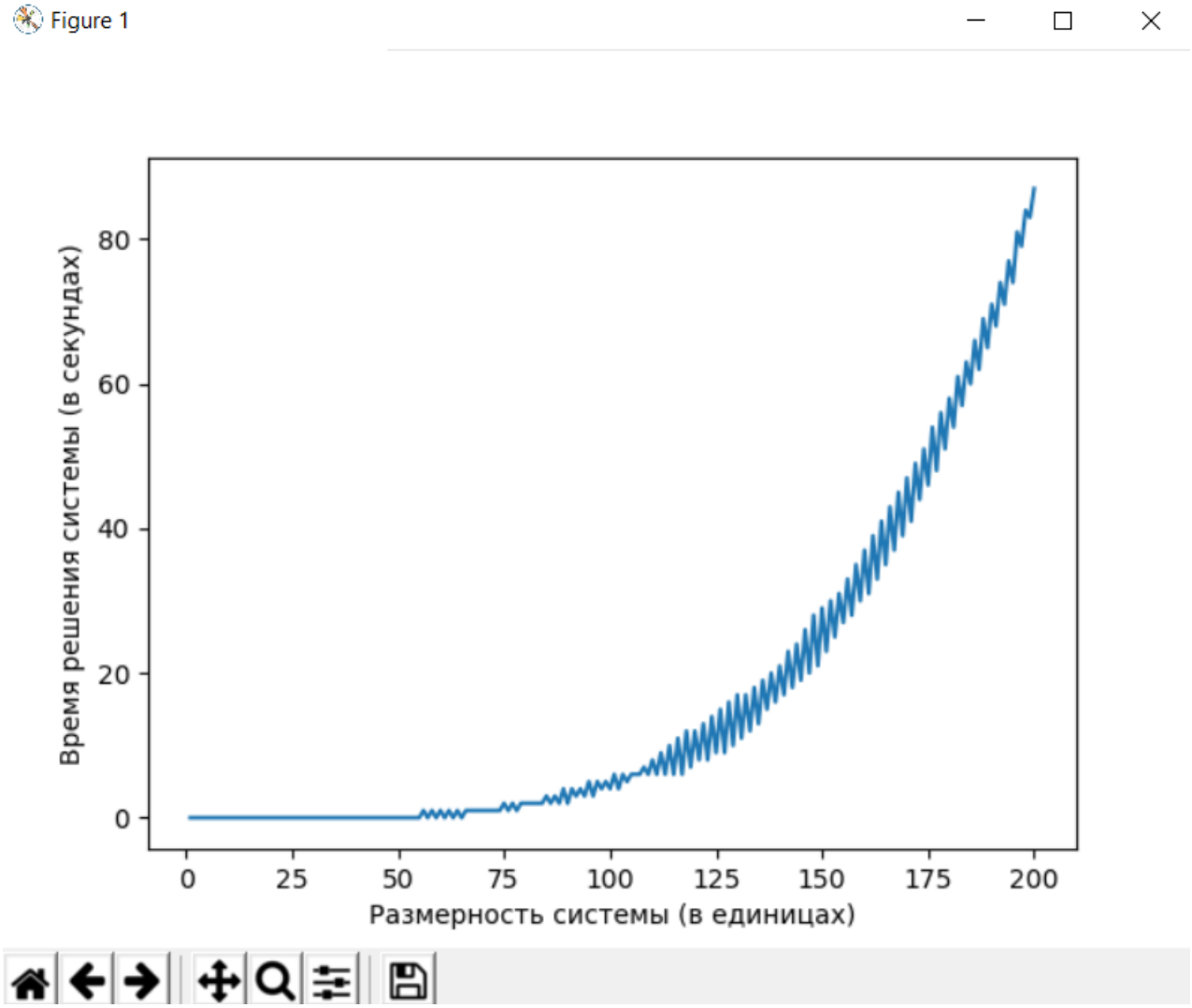
Задание 4

А вот эта часть задания мне понравилась больше всего. Меня попросили проследить то, сколько времени выполняется решение системы в зависимости от её размерности. За эту часть отвечает следующий мой кусочек программы:

```
print("Начало эксперимента с замерами времени работы программы на больших системах...")
ExperimentSizeCheck = 200  # Число размерной матриц для экспериментов... Т.е. будем смотреть матрицы размерностей { 1, 2, ..., ExperimentSizeCheck - 1, ExperimentSizeCheck }
SizeArr = np.arange(1, ExperimentSizeCheck + 1)  # Массив абсцисс (размерностей) для графика
TimeArr = np.zeros(ExperimentSizeCheck)  # Массив ординат (времени в секундах) для графика
start_time = time.time()  # Начало отсчета времени (для замера скорости программы)
previous_time = 0
for i in range(1, ExperimentSizeCheck + 1):
    TimeArr[i - 1] = int(time.time() - start_time) - previous_time
    previous_time = TimeArr[i - 1]
    A = np.random.rand(i, i)
    b = np.random.rand(i)
    Gauss_Selecting_Main_Element_Throughout_Whole_Matrix(A, i, b)
plt.plot(SizeArr, TimeArr)
plt.xlabel("Размерность системыми (в единицах)")
plt.ylabel("Время решения системы (в секундах)")
plt.show()
print("Общее время работы: %s seconds" % (time.time() - start_time))
```

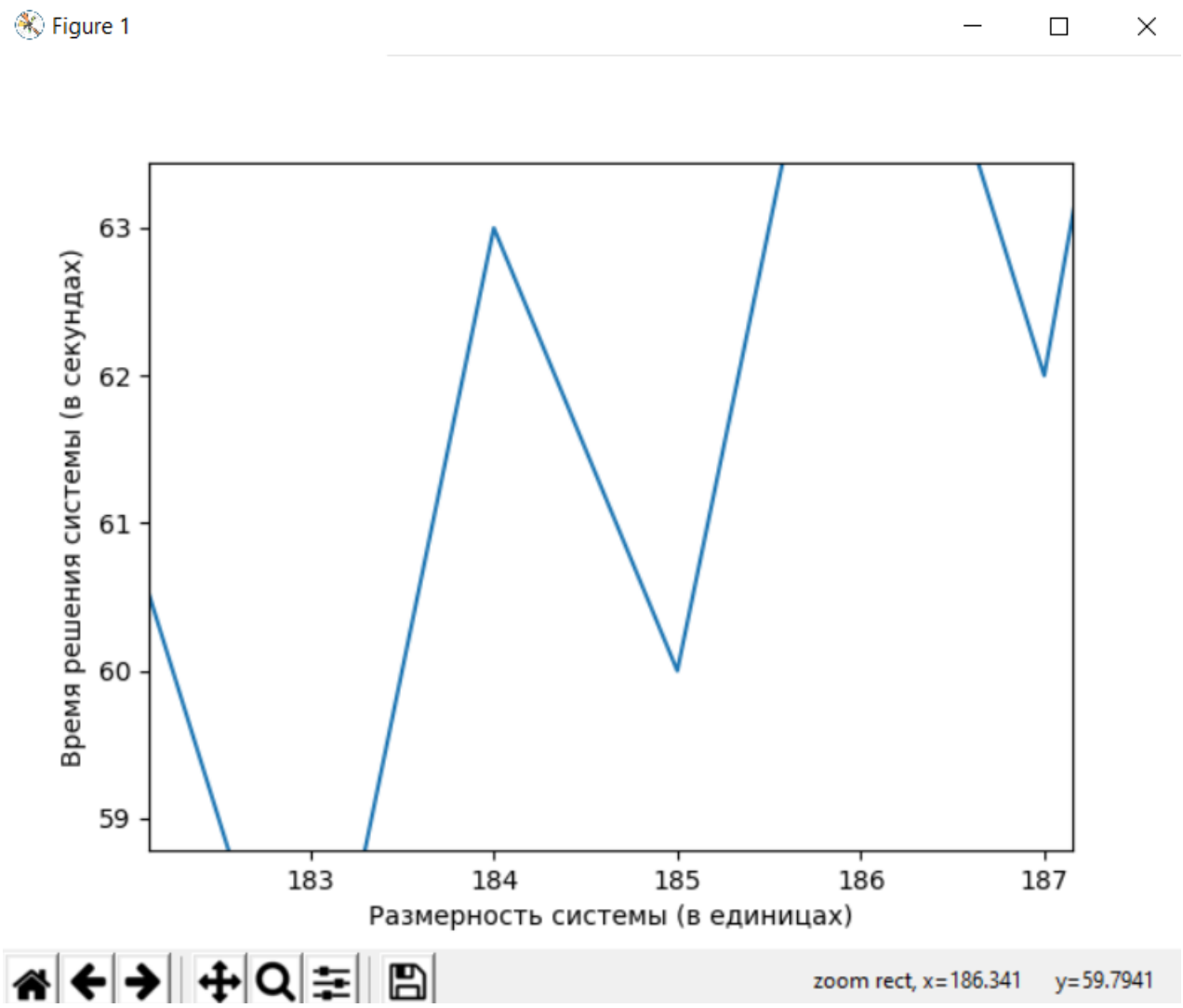
Для этого я генерировал матрицы размерностей 1x1, 2x2, 3x3, …, 198x198, 199x199, 200x200 и соответственно вектора к ним размерностей 1, 2, 3, …, 198,

199, 200 со случайными вещественными числами. После этого программа отработала несколько минут и выдала следующий график:



Я замерял время (в секундах) от размерности системы. Мы можем наблюдать, что где-то до размерности 55 программа решает системы почти мгновенно.

Однако далее, с увеличением размерности время работы программы значительно увеличивается, это видно на графике. Таким образом, система с размерностью 185 уже будет выполняться около минуты:



Что интересно, несмотря на то, что график в целом постепенно поднимается (что неудивительно), на нём можно заметить падения и подъёмы. Должно быть, это как-то связано с погрешностями в вычислениях программы.

Выводы:

Подытожив, можно сказать, что решение СЛАУ методом Гаусса является довольным полезным способам решать различные задачи.

Выбирать элемент можно по-разному. Мы делали это по все матрице, хотя, конечно, лучше бы было делать по столбцам (так не нужно запоминать перестановку и тратить потом время на перемещение столбцов). Его нужно выбирать, чтобы программа как можно точнее вычислила решение.

Также, можно использовать символьные вычисления (например, через SymPy) для совсем точного результата, а ещё посчитать числа обусловленности для матриц, чтобы пронаблюдать закономерности в погрешностях.

В зависимости от размерности подаваемой на вход системы может сильно изменится время работы программы. Повышение времени работы повышается примерно после 50-ти итераций увеличения размерности, что немаловажно знать.

В целом, был получен большой опыт в решении СЛАУ.