## Национальный исследовательский университет «МЭИ» (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №3:

## «Решение систем линейных алгебраических уравнений»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 52

2021 Москва

**Постановка задачи:**

**Задача 3.1.**  Реализовать решение СЛАУ с помощью LU разложения и LU разложения по схеме частичного выбора. Решить систему небольшой размерности с возмущенной матрицей обоими методами, оценить погрешность и сравнить с теоретической оценкой. Проанализировать поведение методов с ростом числа уравнений.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ.**

1. Реализовать метод решения СЛАУ с помощью LU разложения в виде, указанном в приложении. Убедиться в его работоспособности.
2. Реализовать метод решения СЛАУ с помощью LU разложения по схеме частичного выбора в виде, указанном в приложении. Убедиться в его работоспособности.

2\*. Реализовать метод решения СЛАУ с помощью LU разложения по схеме частичного выбора без перестановки строк в виде, указанном в приложении. Убедиться в его работоспособности.

1. Решить систему , размера 5x5, двумя методами. Вектор  задается как , где ,  -- номер варианта. Матрицу  задать как  и к одному элементу прибавить .
2. Вычислить погрешность и сравнить ее с теоретической оценкой. Для вычисления обратной матрицы можно воспользоваться встроенными функциями.
3. Задавая вектор  как , где , решить систему обоими методам для размера матрицы .
4. Построить на одном графике погрешности обоих методов как функций, зависящих от . Прокомментировать полученный результат.

**Условия к задаче 3.1**

 **(52+3 mod 2 = 1)--** решение с помощью LU модифицирует исходную матрицу А; решение с помощью LU по схеме частичного выбора реализовано в виде двух функций, одна из которых возвращает две матрицы – L и U, не модифицируя A, а вторая функция решает систему.

|  |  |
| --- | --- |
| **(52+3 mod 4 = 3)** |  |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy as np  
import numpy.linalg as LA  
from scipy.linalg import lu\_factor, lu\_solve  
import matplotlib.pyplot as plt  
  
#создание матрицы A

def create\_A(n):  
 A = []  
 for i in range(n):  
 row = []  
 for j in range(n):  
 row.append(np.arctan(.1\*(10\*i+j+1)))  
 A.append(np.array(row))  
 return np.array(A)  
  
#создание возмущенной матрицы A\*(изменен элемент А[0,0])

def create\_A\_perturbed(A):  
 A\_perturbed = A.copy()  
 A\_perturbed[0, 0] += .001  
 return A\_perturbed

#создание вектора правой части b  
  
def create\_b(A, n, x\_i):  
 x = []  
 for i in range(n):  
 x.append(x\_i)  
 return np.dot(A, np.array(x))  
  
#функция возвращает матрицы LU, PIV(вектор перестановок строк)

def get\_lu\_and\_piv(A):  
 A1 = A.copy()  
 n = np.size(A, 0)  
 piv = np.zeros(n)  
  
 for i in range(n):  
 A1\_t = A1.transpose()  
 max\_index = np.argmax(A1\_t[i])  
 piv[i] = max\_index  
 A1[i], A1[max\_index] = A1[max\_index], A1[i].copy()  
 for j in range(i+1, n):  
 A1[j][i] /= A1[i][i]  
 for k in range(i + 1, n):  
 A1[j][k] -= A1[j][i] \* A1[i][k]  
  
 return A1, piv

#функция решает систему Аx = b методом LU разложения с выбором главного элемента

def lu\_with\_pivots(A, b):  
 return lu\_solve(get\_lu\_and\_piv(A), b)

#функция решает систему Аx = b методом LU разложения без выбора главного элемента  
  
def lu\_no\_pivots(A, b):  
 n = np.size(A, 0)  
 LU = A.copy()  
 for i in range(n):  
 for j in range(i+1, n):  
 LU[j][i] /= LU[i][i]  
 for k in range(i + 1, n):  
 LU[j][k] -= LU[j][i] \* LU[i][k]

#теперь у нас есть матрица LU. Для решения системы с помощью встроенной в пакет scipy функции lu\_solve необходимо передавать параметры в специальном виде. Необходимо передать вектор перестановок(мы перестановок не делали, так как применяли LU разложение без выбора главного элемента, поэтому piv = [0,1,2,3,…])

piv = []  
 for k in range(n):  
 piv.append(k)  
 piv = np.array(piv)  
  
 return lu\_solve((LU, piv), b)  
  
  
A = create\_A(5)  
b = create\_b(A, 5, 52)

#решение системы Ax=b двумя методами  
  
x\_no\_pivots = lu\_no\_pivots(A, b)  
x\_with\_pivots = lu\_with\_pivots(A, b)  
print(**'Initial System:'**)  
print(**'Solution LU:'**, x\_no\_pivots)  
print(**'Solution LU(with Partial Pivoting):'**, x\_with\_pivots)  
print()

#решение возмущенной системы A\* x=b двумя методами  
  
A\_perturbed = create\_A\_perturbed(A)  
x\_no\_pivots\_perturbed = lu\_no\_pivots(A\_perturbed, b)  
x\_with\_pivots\_perturbed = lu\_with\_pivots(A\_perturbed, b)  
print(**'Perturbed System:'**)  
print(**'Solution LU:'**, x\_no\_pivots\_perturbed)  
print(**'Solution LU(with Partial Pivoting):'**, x\_with\_pivots\_perturbed)  
print()

#вычисляем априорную оценку погрешности с помощью нормы обратной матрицы  
  
A\_inv = LA.inv(A)  
A\_inv\_norm = LA.norm(A\_inv, ord=np.inf)  
prior\_estimate = .001 \* A\_inv\_norm

#вычисляем апостериорную оценку погрешности на примере метрода LU разложения без выбора главного элемента   
  
  
posterior\_estimate = LA.norm(x\_no\_pivots - x\_no\_pivots\_perturbed, ord=np.inf) / LA.norm(x\_no\_pivots, ord=np.inf)  
print(**'Prior error estimation: '**, prior\_estimate)  
print(**'Posterior error estimation: '**, posterior\_estimate)  
print(**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**)

error\_no\_pivots\_list = [] #список погрешностей, метод - LU разложение  
error\_with\_pivots\_list = [] #список погрешностей, метод - LU разложение с выбором главного элемента  
  
for k in range(5, 16):  
 A = create\_A(k)  
 b = create\_b(A, k, 52)  
 x\_no\_pivots = lu\_no\_pivots(A, b)  
 x\_with\_pivots = lu\_with\_pivots(A, b)

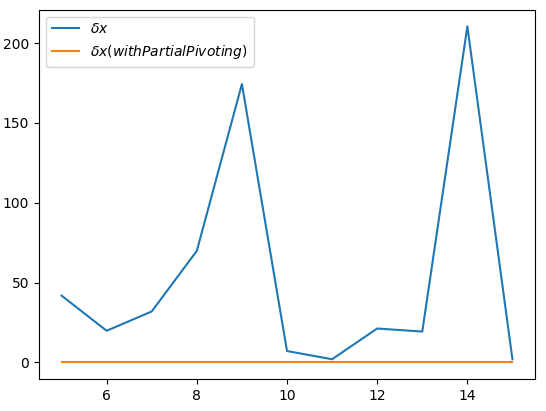
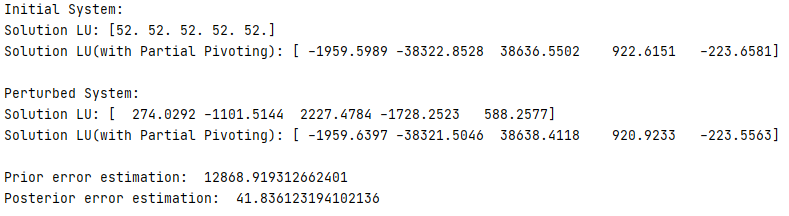
#находим решения для возмущенной системы A\*  
  
 A\_perturbed = create\_A\_perturbed(A)  
 x\_no\_pivots\_perturbed = lu\_no\_pivots(A\_perturbed, b)  
 x\_with\_pivots\_perturbed = lu\_with\_pivots(A\_perturbed, b)

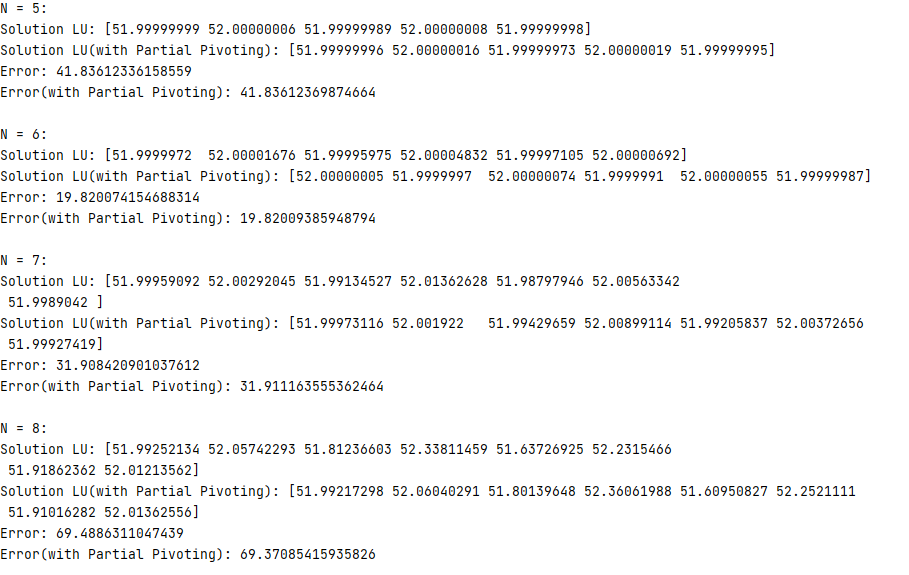
#вычисляем оценки погрешностей для каждого метода и добавляем их с списки  
  
 error\_no\_pivots = LA.norm(x\_no\_pivots - x\_no\_pivots\_perturbed, ord=np.inf) / LA.norm(x\_no\_pivots, ord=np.inf)  
 error\_with\_pivots = LA.norm(x\_with\_pivots - x\_with\_pivots\_perturbed, ord=np.inf) / LA.norm(x\_with\_pivots, ord=np.inf)  
  
 error\_with\_pivots\_list.append(error\_with\_pivots)  
 error\_no\_pivots\_list.append(error\_no\_pivots)  
  
 print(**'N = '** + k.\_\_str\_\_() + **':'**)  
  
 print(**'Error:'**, error\_no\_pivots)  
 print(**'Error(with Partial Pivoting):'**, error\_with\_pivots)  
 print()

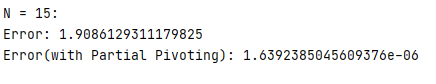
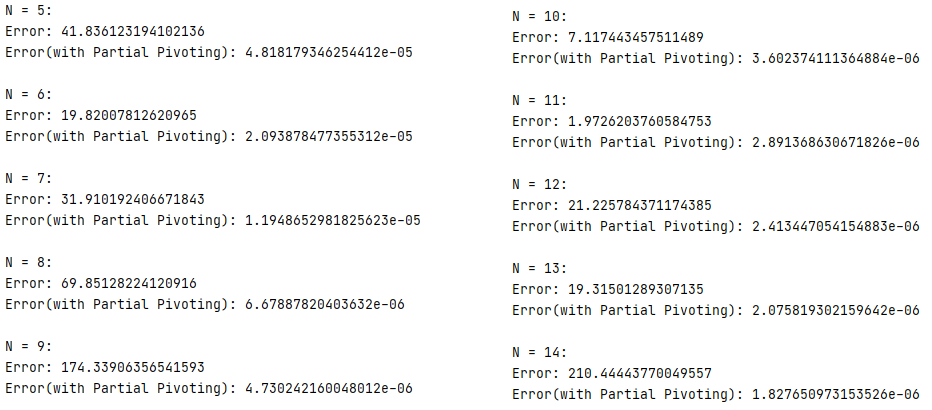
#построение графиков погрешностей для разных размерностей матриц

fig = plt.subplots()  
x\_data = np.linspace(5, 15, 11)  
plt.plot(x\_data, error\_no\_pivots\_list, label=**f'$\delta x$'**)  
plt.plot(x\_data, error\_with\_pivots\_list, label=**f'$\delta x(with Partial Pivoting)$'**)  
plt.legend()  
plt.show()

**Результаты выполнения:**

****

****

****

**Анализ результатов:**

1. Оба разложения будем реализовывать самостоятельно, не использую встроенный пакет.
2. Для LU разложения с выбором главного элемента будем в ходе алгоритма заполнять вектор перестановок piv, чтобы затем воспользоваться встроенной функцией lu\_solve для получения решения.

Вектор перестановок должен интерпретироваться последовательно. Например, если piv=[1,2,2] , то следующее должно быть сделано последовательно (с индексацией на основе нуля):

* 1. Строка 0 меняется на строку 1
  2. Новая строка 1 меняется на строку 2 и
  3. Новая строка 2 остается прежней.

1. LU-разложение стандартное(без выбора главного элемента). Получать само решение будем аналогично. Для этого необходимо передавать параметры в специальном виде. LU-разложение должно быть упаковано в одну матрицу, а также необходимо передать вектор перестановок(мы перестановок не делали, поэтому передаем piv=[0,1,2,…])
2. Получив теоретическую оценку погрешности решения порядка 12000, убеждаемся в том, что система плохо обусловлена. Апостериорная оценка погрешности – 41.8, имеем критическую потерю точности
3. Для метода с выбором главного элемента вне зависсимости от размера матрицы, погрешность небольшая, порядка , тогда как для стандартного LU-разложения погрешность огромная. Для некоторых размеров матрицы удается получить решение с небольшой погрешностью(порядка единицы), тем не менее эта погрешность все равно больше той, что получена методом с выбором главного элемента. И выбирая какие-то конкретные размеры матрицы, на которых погрешность небольшая, мы ограничиваем общность решения.

**Постановка задачи:**

**Задача 3.2** Дана система уравнений *Ax=b* порядка *n* с разреженной матрицей *A*. Решить систему прямым методом.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:**

1. Для указанной в индивидуальном варианте системы уравнений вывести формулы для нахождения неизвестных.
2. Предусмотреть компактное размещение элементов матрицы в памяти ЭВМ, используя одномерные массивы.
3. Подготовить тестовый пример.
4. Решить систему для тестового примера и для указанной в варианте системы уравнений.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.2.52 | 32 | На главной диагонали элементы равны 36, на побочной диагонали элементы равны 22, в 8 строке элементы равны 10, в 24 строке элементы равны 14. |  |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy as np  
from scipy.linalg import lu\_factor, lu\_solve  
  
#решение системы 2х2 методом LU-разложения

#передаем два параметра – элементы вектора b

#зададим им значения по умолчанию – это потребуется для решения тестового примера, с другими значениями вектора b

def solve\_2\_x\_2(b\_left=36+22, b\_right=36+22):  
 return lu\_solve((lu, piv), np.array([b\_left, b\_right]))  
  
#реализация алгоритма решения. Параметр testing – true, если решается тестовый пример, false – если решаем заданную систему

def solve\_a(testing):  
 x = np.zeros(n) # заполняем вектор решения нулями

#заполняем вектор решения начиная с середины, двигаясь в обе стороны  
 for left in range(center\_index\_left, -1, -1):   
 right = n - left – 1

#если наткнулись на заполненную строку, пропускаем ее и ее пару  
 if left == 7 or right == 23:  
 continue

#записываем очередные два элемента вектора решая систему 2х2(в зависимости от параметра testing будут различаться правые части b)

x[left], x[right] = solve\_2\_x\_2() if testing else solve\_2\_x\_2(n + left, n + right)

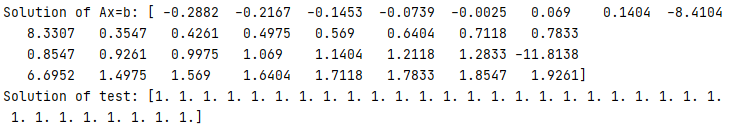
#осталось заполнить четыре компонента вектора решения

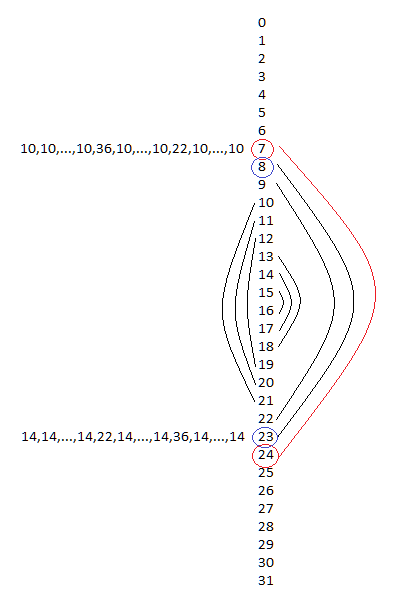
#для этого нужно решить систему 4х4(в зависимости от параметра testing будут различаться правые части b)  
 s = sum(x)  
 b4 = np.array([10\*(n-2) + 36 + 22 - s \* 10, 36 + 22, 14\*(n-2) + 36 + 22 - s \* 14, 36 + 22]) if testing \  
 else np.array([n + 7 - s \* 10, n + 8, n + 23 - s \* 14, n + 24])  
 x[7], x[8], x[23], x[24] = np.linalg.solve(A4, b4)

return x  
  
#матрица 2х2  
A2 = np.array([[36, 22], [22, 36]])

#матрица 4x4  
A4 = np.array([[36, 10, 10, 22], \  
 [0, 36, 22, 0], \  
 [14, 22, 36, 14], \  
 [22, 0, 0, 36]])

#выполняем LU разложение матрицы один раз  
lu, piv = lu\_factor(A2)  
n = 32  
center\_index\_left = round(n / 2) – 1   
  
np.set\_printoptions(precision=4, suppress=True)  
print(**'Solution of Ax=b:'**, solve\_a(testing=False))  
print(**'Solution of test:'**, solve\_a(testing=True))

**Результаты выполнения:**



**Описание алгоритма:**

1. Размерность 32 – четная, значит диагональных коллизий не будет. В центре матрицы наблюдаем следующую картину . Слева и справа в 15-м и 16-м уравнении нули. Значит можем найти , решив систему 2х2
2. Рассмотрим соседние уравнения – 14-ое и 17-ое.

Середину можем выбросить, получим: Слева и справа в 14-м и 17-м уравнении нули. Значит можем найти , решив систему 2х2. Получаем закономерность.

Таким образом, двигаясь из середины будем находить пару , решая одну и ту же систему 2х2, но с разными значениями вектора правой части. Для этой задачи идеально подходит LU-разложение. Получив один раз разложение матрицы А, будем применять его для разных b.

1. Исключения - уравнения под номером 8 и 24(из-за смещения индексов в ЭВМ - 7 и 23).

7-ая строка заполнена числом 10 с двумя коллизиями. 23-я - числом 14, также с двумя коллизиями. При этом они не образуют пару(см. рисунок)

1. Пропустив эти две пары, в векторе решений не будет хватать четырех компонент . Рассмотрим систему из этих четырех уравнений

Заметим, что сейчас в векторе решения нули на местах 7,8,23,24. Преобразуем первое и третье уравнения, учитывая этот факт.

1. Решаем эту систему и заполняем пропуски в векторе решения.
2. Для тестового примера будем использовать вектор . Тогда,

**Анализ результатов:**

Тестовый пример прошел проверку, и мы получили единичный вектор. Значит алгоритм работает корректно.

**Постановка задачи:**

**Задача 3.3.** Решить задачу итерационным методом. указанным в индивидуальном варианте . Вектор правой части задается как , где .

Элементы матрицы *A* задаются формулами: , параметр  задается формулой : , здесь  – номер варианта, *m*- размерность матрицы, указанная в варианте. Вектор ***b*** задается по вектору решения.

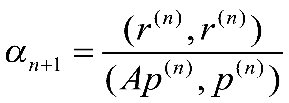
|  |  |  |
| --- | --- | --- |
| **52** | m=25 | Метод сопряженных градиентов |

**Алгоритм метода:**

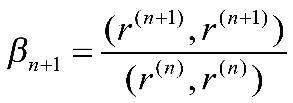
1. Выбираем  https://lh3.googleusercontent.com/4P8dRfovSN2oynOt7p6HmHAaBjELVg8rgmSxvIEhjZlbLvr4eZBZ6vwXpTtt7OEl7AmTqrf2Gl2h7O8vAAi4HINy-9dFbWqJHuwmnGBZPjsl8w-qWyfpo3t51zj8oEJ4JBH2UBc, вычисляем вектор невязки  https://lh3.googleusercontent.com/X8kUj2UUWzb4iztjx4CtjitnZlhLa0raEDG2YFixJLTHeVU7ugQvR36zXDx_ukJxcPwnqBAbD4R1u0-VmcC3po_qqS8fTSI2kWq7bsk0wcY5FYinWjztwXjcr6KQArGluP92t8s;    вектор https://lh5.googleusercontent.com/B4CSpJ3zvV7bn18bCiHxYbWdDz7boXwt_7kJWyStUGoDxhQSv3qHBtrztdvHgtSi_sH8V6qLhMx4erZ37NBG12eXtZYMUhRqg7oG-AKKIxuV6rst3hOgkaEeFkqy5GepZoq39TU (называемый

   направлением спуска )  полагаем равным вектору невязки:  https://lh3.googleusercontent.com/rE4doSkK6VpQe2RTwqc0qSR6qUWaxTXz4s7oPC0Gx_a7vI1XuPM4e943ynGUfGWhewu9m4wGbMTWpq8reQFJ6am1BlSpxRCwCaSyh6yd9OY1nSwppN2hjGYtEH8nR6J8sFFgULE.

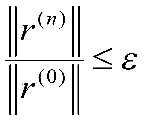
                Далее совершаются итерации:  n=0, 1, …до достижения точности:

2. Вычисляется параметр  и  затем   делается шаг https://lh5.googleusercontent.com/Ws1__Oig-SIkfD0cTkpnye5z8WxqJaAHMHgryptkerZW_ywyP6ma8aFyKy-V7lZLidIRO85TwhlmLoCmXWFsWn-azArjT3L7NGruYzS1dhyf-qA1MPJFe5XEMWoyePRXbnlgPI4

3. Вычисляется  заново вектор невязки: https://lh3.googleusercontent.com/u36_k1smdNnicZXYUDEqImLA0KrrXJJmIr_ZFxpQ_Ov3mMl0YnxC6WefQwJLkUDsd_59ScXBOQ6_UoqzxBPtXh-6iUEWK6JgbWy0UqediuI88dYVB2Ma2FwU2-wSK1UHwtTA6AA

4. Вычисляется еще один параметр метода: 

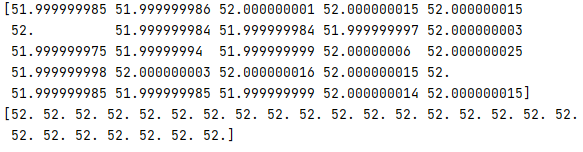
5. Вычисляется новый вектор направления   https://lh3.googleusercontent.com/0vdG6EdkBjFxW88jPzGqLOKnndVx2F5RS6-rJT-zKyLVwhXMpxH6YPJev7P86QRUcPfKivlyH_5IuUQqVGqTAO6DiTW6_b_l3W5ANEO0hLR_wSU8n6E0TDPIGLvYmG2gaQrmTTU

6. Вычисления ведутся до тех пор пока 

**Код программы:**

import numpy as np  
import numpy.linalg as LA  
  
  
def create\_A(m, beta):  
 A = []  
 for i in range(m):  
 row = []  
 for j in range(m):  
 row.append(np.cos(i+j) / .1 / beta + .1 \* beta \* np.exp(-(i - j)\*\*2))  
 A.append(np.array(row))  
 return np.array(A)  
  
  
def create\_b(A, m, x\_i):  
 x = []  
 for i in range(m):  
 x.append(x\_i)  
 return np.dot(A, np.array(x))  
  
  
def solve\_conjugate\_gradient(A, b, epsilon=np.power(10., -9)):  
 x = np.zeros(np.size(b))  
 residual\_0 = np.abs(np.dot(A, x) - b)  
 descent\_direction\_n = residual\_0  
 residual\_n = residual\_0  
  
 while LA.norm(residual\_n) / LA.norm(residual\_0) > epsilon:  
 alpha = np.dot(residual\_n, residual\_n) / np.dot(np.dot(A, descent\_direction\_n), descent\_direction\_n)  
 x = x + alpha \* descent\_direction\_n  
 residual\_n\_next = residual\_n - alpha \* np.dot(A, descent\_direction\_n)  
 beta = np.dot(residual\_n\_next, residual\_n\_next) / np.dot(residual\_n, residual\_n)  
 descent\_direction\_n = residual\_n\_next + beta \* descent\_direction\_n  
 residual\_n = residual\_n\_next  
 return x  
  
  
m = 25  
n = 52  
beta = (np.abs(66-n) + 5) \* m  
A = create\_A(m, beta)  
b = create\_b(A, m, n)  
  
x = solve\_conjugate\_gradient(A, b)  
np.set\_printoptions(precision=9, suppress=True)  
  
print(x)  
print(LA.solve(A, b))

**Результаты выполнения:**

****

**Анализ результатов:**

Решение, полученное методом сопряженных градиентов получено с незначительной погрешностью(сравнивается с решением полученным встроенным методом numpy.linalg.solve)