Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

**Лабораторная работа № 2**

**«Итерационные методы решения систем линейных уравнений.»** по учебной дисциплине «Методы численного анализа»

**Выполнили:**

студент гр. 153504 Климкович Н. В.

студент гр. 153504 Тиханёнок И. А.  
 студент гр. 153504 Тарасенко Ф. П.

**Проверила:**

ст. преподаватель кафедры информатики Стройникова Е. Д.

Минск 2022

Содержание

[1. Цель работы 3](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.30j0zll)

[2. Теоретические сведения 3](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.1fob9te) [3. Программная реализация 9](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.26in1rg) [4. Выводы 18](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.lnxbz9)

**1. Цель работы:**

1. Изучить теоретические сведения об итерационных методах решения СЛАУ.

2. Составить алгоритмы решения СЛАУ указанными методами, применимыми для организации вычислений на ЭВМ.

3. Составить программу решения СЛАУ по разработанному алгоритму.  
4. Продемонстрировать работу программы на примере СЛАУ и проверить правильность работы программы.

**2.Теоретические сведения:**

Прямые методы применяют главным образом для решения задач малой размерности, когда нет ограничений в доступной оперативной памяти ЭВМ или необходимости выполнения чрезмерно большого числа арифметических операций. Большие системы уравнений, возникающие в основном в приложениях, как правило, являются разреженными. Методы исключения для систем с разряженным и матрицами неудобны, например, тем, что при их использовании большое число нулевых элементов превращается в ненулевые, и матрица теряет свойство разреженности. В противоположность им при использовании итерационных методов в ходе итерационного процесса матрица не меняется, и она, естественно, остается разреженной. Большая эффективность итерационных методов по сравнению с прямыми методами тесно связанна с возможность. Существенного использования разреженности матриц.   
 *Итерационные методы* основаны на построении сходящейся к точному решению ***x*** рекуррентной последовательности.

**Матричные нормы**

Рассмотрим векторное пространство . Пусть  − вектор данного пространства.

*Нормой вектора*называется функция  от вектора , для которой выполняются следующие аксиомы:

(Н1): ,   ;

(Н2): , , ∈;

(Н3): , .

Пример:

 – кубическая норма;

 − евклидова норма;

 – октаэдрическая норма.

Существуют и другие менее употребляемые векторные нормы.

Векторное пространство вместе с заданной в нем нормой называется *нормированным векторным пространством*.

Пример. Рассмотрим нормированные векторные пространства с различными нормами в двумерном пространстве **R**2 и соответствующие им единичные окрестности начала координат:

: *x*2

*x*1 с нормой ;

: *x*2

*x*1

с нормой ;

: *x*2

*x*1 c нормой .

Рассмотрим квадратную матрицу .

*Нормой матрицы* *А*, индуцированной нормой вектора , называется число

.

Из этого определения непосредственно следует, что для любой индуцированной нормы справедливы следующие свойства:

1) , где *Е* – единичная матрица;

2) , где *О* – нулевая матрица;

3) ;

4) ,

т. е.

.

Легко видеть, что для индуцированной нормы матрицы выполняются соотношения:

(Н1'): ,   ;

(Н2'): ;

(Н3'): ;

(Н4'): .

*Нормой матрицы*называется функция ||*A*|| от матрицы *А*, для которой выполняются аксиомы (Н1') – (Н4').

Таким образом, индуцированные матричные нормы представляют собой частный случай матричных норм. Из аксиомы (Н4') очевидно следует, что



Найдем матричную норму , индуцированную векторной нормой :





Таким образом, .

Докажем, что . Для этого необходимо показать, что .

По определению  для любого .

Предположим, что в  максимум достигается при *i*=*i*0,т. е. .

Возьмем ненулевой вектор , такой, что . Тогда . Следовательно,



То есть, действительно, .

Совершенно аналогично можно показать, что

.

Сложнее дело обстоит с матричной нормой, индуцированной евклидовой векторной нормой.

*Спектральным радиусом* матрицы *А* называется число

μ(*А*) = max{|λ1|,…, |λ*n*|},

где λ1,…, λ*n* – собственные значения данной матрицы.

**Лемма 1.** Для любой матричной нормы μ(*А*) ≤ ||*A*||.

*Доказательство.* Действительно, пусть λ – собственное значение матрицы *А* и μ(*А*) = |λ|. Построим матрицу *В* того же порядка, что и *А*, у которой первый столбец совпадает с собственным вектором  матрицы *А*, соответствующим собственному значению λ, а остальные столбцы – нулевые. Тогда очевидно *АВ*=*λВ*. Используя аксиомы из определения матричной нормы, получим

|λ| ||*В*|| ≤ ||*A*|| ||*В*||,

и поскольку *В* ≠ *О*, и, следовательно, ||*В*|| ≠ 0, получаем |λ| ≤ ||*A*||.

По аналогии с векторами в  можно определить сходимость последовательности матриц поэлементно, считая, что *A*(*k*) → *A* при *k* → ∞ тогда и только тогда, когда *aij*(*k*) → *aij* для всех *i, j* = 1,…, *n.* Отметим, что, как и в случае с векторами, для любой матричной нормы из условия сходимости по норме ||*A*(*k*) − *A*|| → 0 всегда следует сходимость *A*(*k*) → *A* при *k* → ∞*.*

На практике часто приходится иметь дело с матричной геометрической прогрессией

*E* + *A* + *A*2 + … + *Ak* + …

и встает вопрос о ее сходимости.

**Лемма 2.** Для того чтобы *Ak* → *O* при *k* → ∞,необходимо и достаточно, чтобы все собственные значения матрицы *А* были по модулю меньше единицы.

*Доказательство*. Докажем лемму для случая симметричной матрицы *А*. Из линейной алгебры известно, что в этом случае

*А* = *Т*–1*ΛT*,

где *Λ* – диагональная матрица с действительными собственными значениями λ1, λ2, …, λ*n* на главной диагонали. Соответственно

*Аk* = *Т*–1*ΛkT*,

где на главной диагонали диагональной матрицы *Λk* стоят элементы

λ1*k*, λ2*k*,…, λ*nk*.

Таким образом, каждый элемент матрицы *Аk* является линейной комбинацией λ1*k*, λ2*k*,…, λ*nk* с коэффициентами, не зависящими от *k*.

Следовательно, если все собственные значения λ1, λ2,…, λ*n* по модулю меньше единицы, то все элементы матрицы *Аk* стремятся к нулю при *k* → ∞, т. е. *Ak* → *O.*

Обратно,

*Λk* = *ТАkТ*–1,

и, следовательно, все λ1*k*, λ2*k*,…, λ*nk* стремятся к нулю при *Ak* → *O.* Последнее означает, что все числа λ1, λ2,…, λ*n* по модулю меньше единицы.

**Лемма 3.** Для того чтобы ряд *E* +*A* +*A*2 + … + *Ak* + …сходился, необходимо и достаточно, чтобы все собственные значения матрицы *А* были по абсолютной величине меньше единицы. В этом случае матрица *Е* − *А* имеет обратную и

*E* + *A* + *A*2 + … + *Ak* + … = (*Е* − *А*)–1*.*

*Доказательство.* 1. Пусть матричный ряд сходится. Это равносильно сходимости *n*2 числовых рядов. Тогда в силу необходимого признака сходимости числового ряда каждый элемент матрицы *Ak* стремится к нулю при *k* → ∞, следовательно, *Ak* → *O* при *k* → ∞. В силу леммы 2 последнее равносильно тому, что все собственные значения матрицы *А* по модулю меньше единицы.

2. Пусть все собственные значения матрицы *А* по модулю меньше единицы. Отметим сразу, что матрица *Е* − *А* невырождена, поскольку ее определитель |*Е* − *А*| = |*A* − *E*| не может обращаться в 0 (иначе среди собственных значений матрицы *А* было бы и число 1). Рассмотрим тождество

(*E* + *A* + *A*2 + … + *Ak*)(*Е* − *А*) = *Е* − *Аk*+ 1,

откуда следует

*E* + *A* + *A*2 + … + *Ak* = (*Е* − *А*)–1 − *Аk*+ 1(*Е* − *А*)–1.

*Ak* → *O* при *k* → ∞ согласно лемме 2*.* Следовательно,

*E* + *A* + *A*2 + … + *Ak* → (*Е* − *А*)–1

при *k* → ∞, т. е. ряд сходится.

Покажем теперь, что норма , индуцированная евклидовой нормой вектора, совпадает с , где λ – наибольшее собственное значение матрицы *А*\**А*, (*А*\* – матрица, полученная из *А* транспонированием).

Прежде всего убедимся, что λ ≥ 0. Действительно, поскольку

(*А*\**А*)*\** = *А*\*(*А*\*)\* = *А*\**А*,

то матрица *А*\**А* – симметрическая. Кроме того,



для любого вектора . То есть *А*\**А* – симметрическая неотрицательно определенная матрица. Как известно из линейной алгебры, все собственные значения такой матрицы действительны и неотрицательны. Более того, существует ортонормированный базис из собственных векторов  данной матрицы, соответствующих ее собственным значениям

λ1 ≥ λ2 ≥ … ≥ λ*n*.

Рассмотрим произвольный вектор  единичной евклидовой нормы и разложим его по базису :

.

Тогда



и, следовательно,

.

Отсюда

.

С другой стороны, для нормы ||*A*||, индуцированной евклидовой векторной нормой, справедливо

, где , откуда

.

В итоге, действительно матричная норма, индуцированная евклидовой векторной нормой, имеет вид

.

Такая норма матрицы *А* называется *спектральной*.

В частном случае, когда матрица *А* симметрическая, *А*\**А* = *А*2 и поскольку собственные значения этой матрицы совпадают с квадратами λ12, λ22,…, λ*n*2 собственных значений матрицы *А*, то спектральная норма матрицы совпадает с наибольшим по абсолютной величине собственным значением матрицы *А*, т. е. равна спектральному радиусу матрицы *А*:

 = μ(*А*).

Спектральная норма матрицы неудобна в практическом плане из-за трудности ее вычисления. Поэтому наряду с нормами ||*A*||1 и ||*A*||3, индуцированными кубической и октаэдрической векторными нормами, мы будем пользоваться евклидовой нормой матрицы

,

где Sp *A* – след матрицы *А*, т. е. сумма ее элементов, стоящих на главной диагонали. Как легко проверить, эту норму можно записать также в виде



Нетрудно проверить, что для введенной нормы  выполняются все аксиомы из определения матричной нормы. Кроме того, эта норма является *согласованной* *с* *евклидовой* *векторной нормой* в том смысле, что для нее и евклидовой векторной нормы всегда выполняется соотношение

.

В дальнейшем мы ограничимся рассмотрением матричных норм, согласованных с соответствующими векторными нормами.

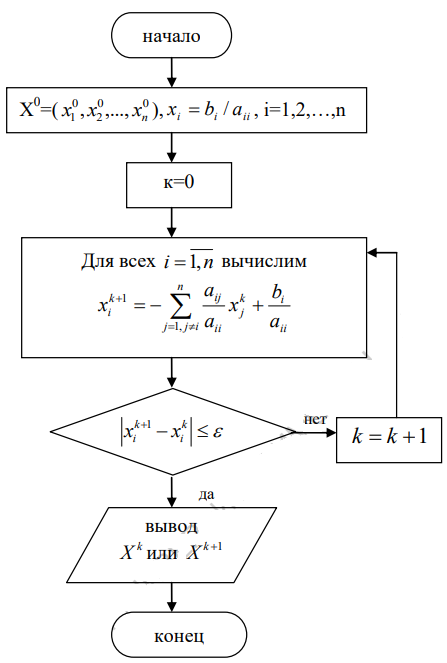
**Метод релаксации**

Методы простых итераций и Зейделя сходятся примерно с такой же скоростью, как геометрическая прогрессия со знаменателем ||*B*||. Если норма матрицы *B* близка к 1, то сходимость очень медленная. Для ускорения сходимости используется **метод релаксации**. Суть его в том, что полученное по методу простых итераций или Зейделя очередное значение координаты *xik* пересчитывается по формуле

*xik* = ω*xik* + (1 − ω)*xik*− 1,

где 0 < ω ≤ 2, ω ≠ 1, – параметр релаксации.

Если ω < 1, то имеет место *нижняя релаксация*, если ω > 1 – *верхняя релаксация*. Параметр ω подбирается так, чтобы сходимость метода  
достигалась за минимальное число итераций.  
  
 **Метод простых итераций** Метод простых итераций относится к итерационным методам, которые основываются на многократном уточнении – приближенном решении задачи *A x = b.* Верхний индекс – номер итерации.



*АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ*

Шаг 1. Исходное уравнение A x = b приводится к виду

,

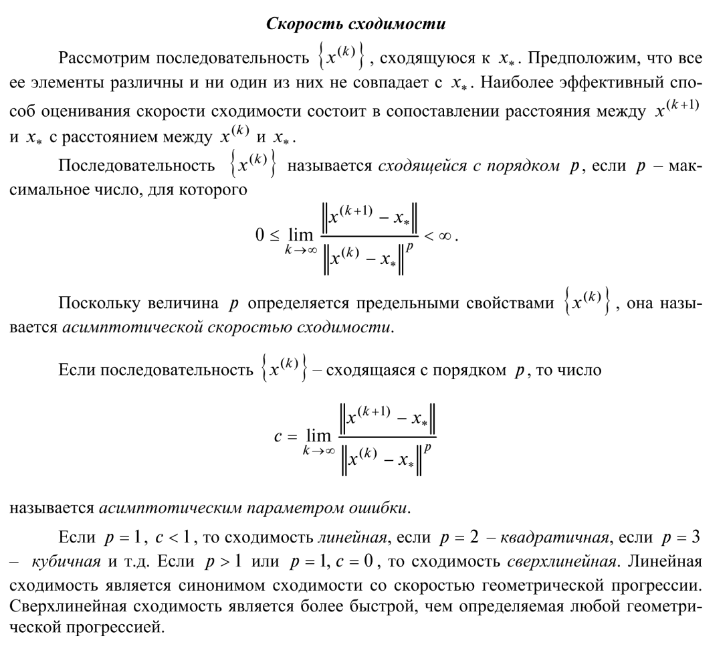
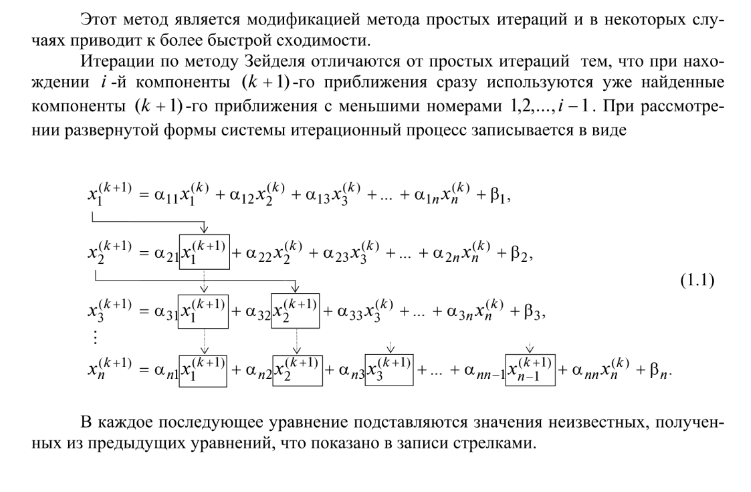
Где – квадратная матрица, – вектор, *i,j = 1,…,n.* При том, для сходимости итераций (норма матрицы должна быть меньше единицы)

Шаг 2. Находим некое начальное приближение , где – произвольный вектор и далее многократно выполняется уточнение решения согласно рекуррентному соотношению:

Шаг 3. Завершение итераций происходит при выполнении условия

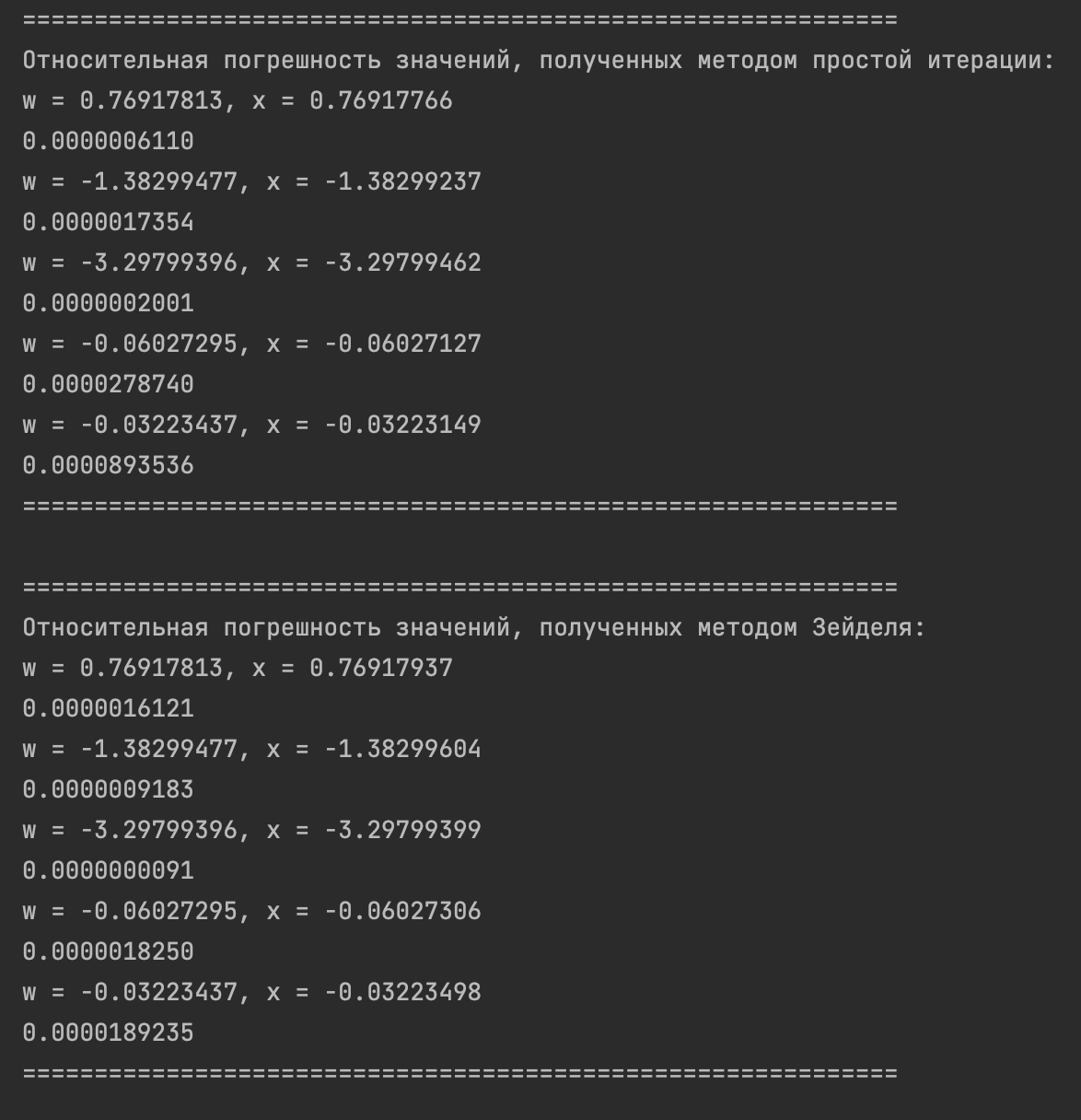
**З а м е ч а н и е:** обычно используют начальное приближение , однако в некоторых случаях использование данного начального приближения может быть нецелесообразно, поскольку будет находиться далеко от искомого решения . Порой начальное приближение берут исходя из грубой прикидки

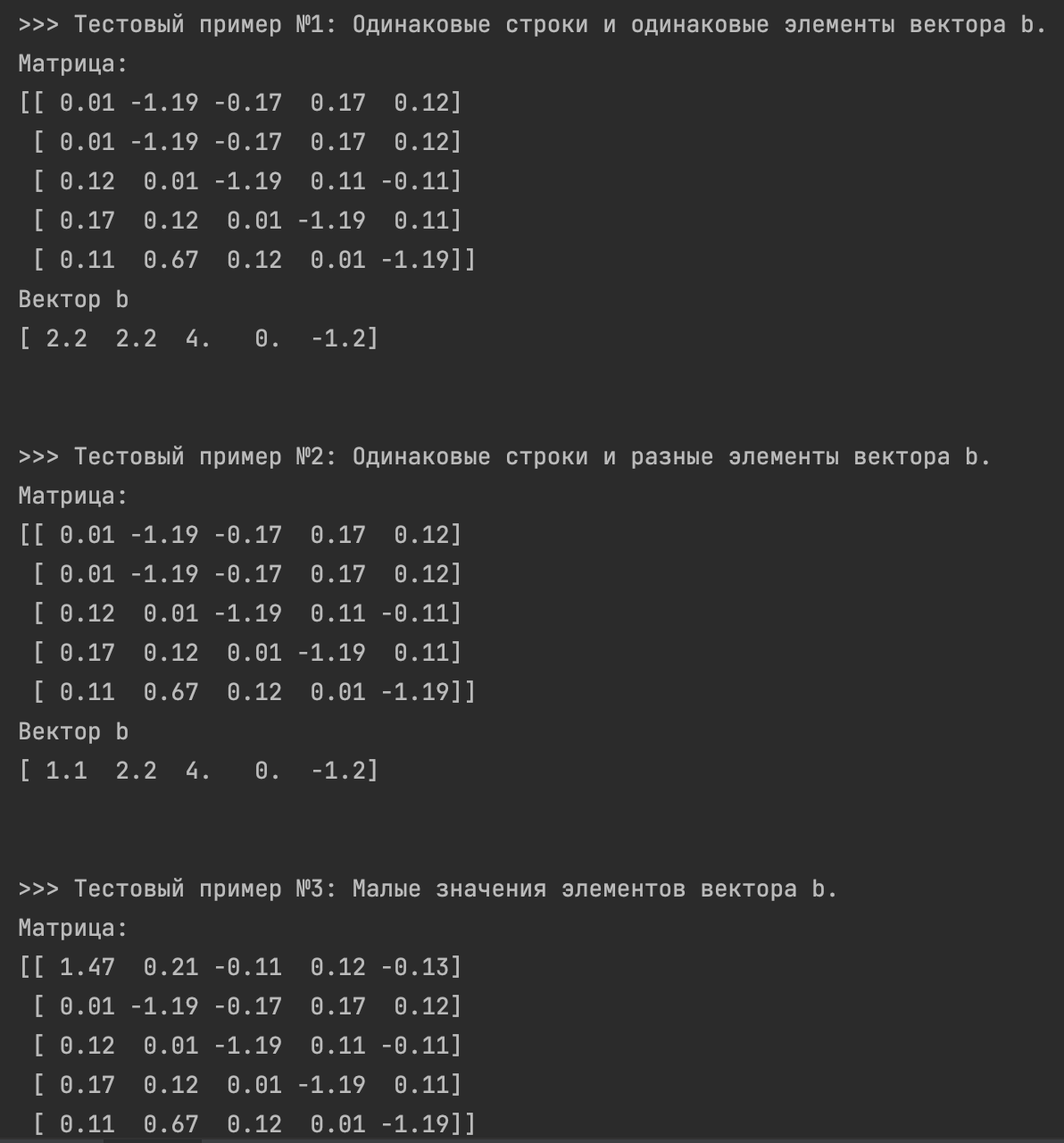
**УСЛОВИЯ СХОДИМОСТИ МЕТОДА ПРОСТЫХ ИТЕРАЦИЙ:**

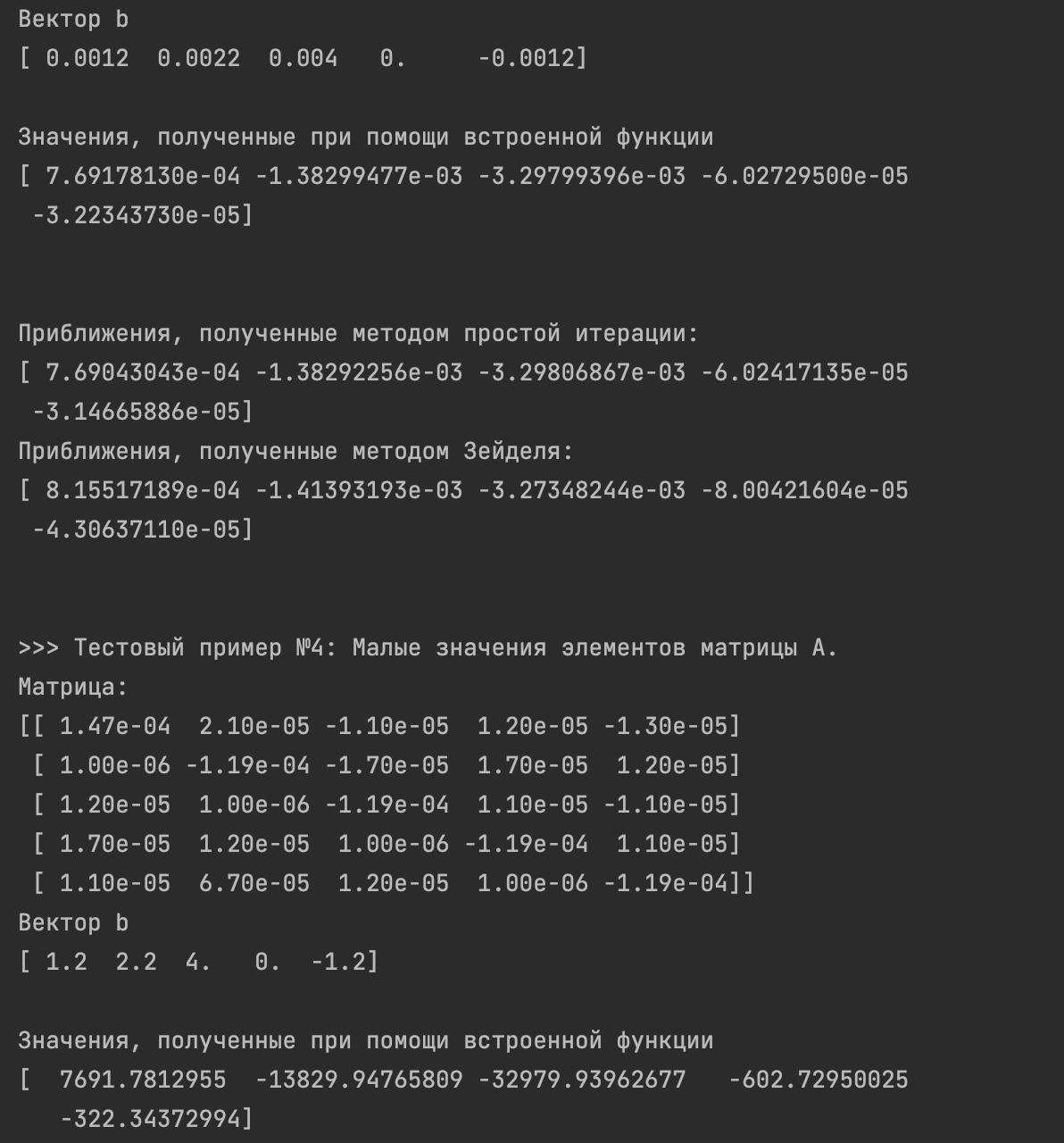
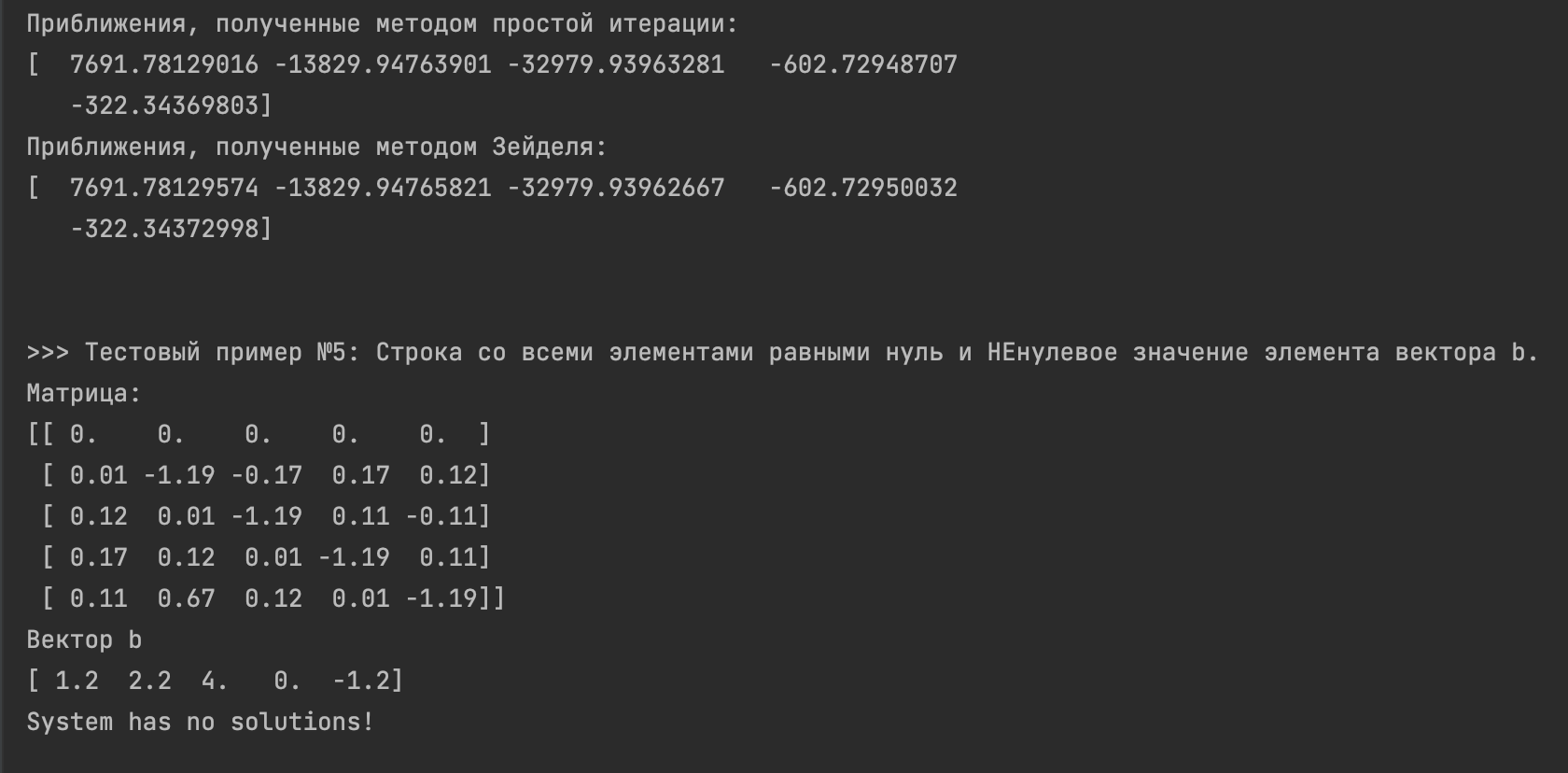
 **Метод Зейделя  
**

**3. Программная реализация**

C = np.array([  
 [0.01, 0, -0.02, 0, 0],  
 [0.01, 0.01, -0.02, 0, 0],  
 [0, 0.01, 0.01, 0, -0.02],  
 [0, 0, 0.01, 0.01, 0],  
 [0, 0, 0, 0.01, 0.01]  
])  
D = np.array([  
 [1.33, 0.21, 0.17, 0.12, -0.13],  
 [-0.13, -1.33, 0.11, 0.17, 0.12],  
 [0.12, -0.13, -1.33, 0.11, 0.17],  
 [0.17, 0.12, -0.13, -1.33, 0.11],  
 [0.11, 0.67, 0.12, -0.13, -1.33]  
])  
b = np.array([1.2, 2.2, 4.0, 0.0, -1.2])  
b = b.transpose()  
A = 14 \* C + D  
  
  
# print(A)  
  
# Проверяет случаи с одинаковыми строками в матрице  
def check\_for\_equal\_rows(matrix, ixes):  
 for i in range(len(matrix) - 1):  
 for j in range(i + 1, len(matrix)):  
 equal = True  
 for k in range(len(matrix[0])):  
 if matrix[i][k] != matrix[j][k]:  
 equal = False  
 if equal == True:  
 if ixes[i] != ixes[j]:  
 print('Matrix has two or more equal rows but b-vector values' +  
 ' differ.\nMatrix has no solutions!')  
 exit()  
 for i in range(len(matrix) - 1):  
 for j in range(i + 1, len(matrix)):  
 equal = True  
 for k in range(len(matrix[0])):  
 if matrix[i][k] != matrix[j][k]:  
 equal = False  
 if equal == True:  
 if ixes[i] == ixes[j]:  
 print('Matrix has two or more equal rows with equal b-vector' +  
 ' values and no equal rows with differend b-vector ' +  
 'values.\nMatrix has infinity of solutions!')  
 exit()  
  
 # меняет местами 2 столбца  
  
  
def swap\_columns(a, i, j):  
 for k in range(len(a)):  
 a[k][i], a[k][j] = a[k][j], a[k][i]  
  
  
# Находит нули на главной дтиагонали  
def check\_zeros\_diag(matrix, b):  
 for i in range(0, len(matrix)):  
 if matrix[i][i] == 0:  
 check = True  
 for j in range(0, len(matrix[0])):  
 if matrix[i][j] != 0 and matrix[j][i] != 0:  
 swap\_columns(matrix, i, j)  
 check = False  
 break  
 if check:  
 if b[i] == 0:  
 print('System has infinity solutions!')  
 else:  
 print('System has no solutions!')  
 exit()  
  
  
# Проверяет, являются ли элементы главной диагонали максимальными в строке  
def max\_on\_diag(matrix):  
 allgood = True  
 for i in range(len(matrix)):  
 maxim = math.fabs(matrix[i][0])  
 k = 0  
 for j in range(0, len(matrix[0])):  
 if math.fabs(matrix[i][j]) > maxim:  
 maxim = math.fabs(matrix[i][j])  
 k = j  
 if k != i:  
 allgood = False  
 break  
 return allgood;  
  
  
# Находит транспонентную матрицу  
def find\_transponent(matrix):  
 new\_matrix = matrix.copy()  
 nstr = len(matrix[0])  
 for i in range(len(matrix)):  
 for j in range(0, nstr):  
 new\_matrix[i][j] = matrix[j][i]  
 return new\_matrix  
  
  
# Выполняет умножение двух матриц  
def matrix\_multiplying(M1, M2):  
 new\_matrix = M1.copy()  
 for i in range(len(new\_matrix)):  
 for j in range(0, len(new\_matrix[0])):  
 new\_matrix[i][j] = 0  
 for i in range(len(M1)):  
 for j in range(0, len(M2[0])):  
 for k in range(0, len(M1[0])):  
 new\_matrix[i][j] += M1[i][k] \* M2[k][j]  
 return new\_matrix  
  
  
# Приводит матрицу к виду, удобному для итераций  
def matrix\_transform(matrix, vec):  
 for i in range(len(matrix)):  
 nstr = len(matrix[0])  
 for j in range(0, nstr):  
 if i != j:  
 matrix[i][j] /= (-matrix[i][i])  
 vec[i] /= matrix[i][i]  
 for i in range(len(matrix)):  
 matrix[i][i] = 0  
 return (matrix, vec)  
  
  
# Итерирует в поисках вектора решений СЛАУ  
def simple\_iterating\_func(matrix, h, vec):  
 summa = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  
 for i in range(len(summa)):  
 summa[i] = h[i]  
 for i in range(len(matrix)):  
 a = 0  
 for j in range(0, len(matrix[0])):  
 a += matrix[i][j] \* h[j]  
  
 summa[i] = a + vec[i]  
 # print('\nsumma')  
 # print(summa)  
 for i in range(len(summa)):  
 difer = summa[i] - h[i]  
  
 if math.fabs(difer) > 0.00001:  
 return simple\_iterating\_func(matrix, summa, vec)  
 else:  
 return summa  
  
  
# Реализует метод простых итераций  
def simple\_iteration\_method(matrix, ixes):  
 check\_zeros\_diag(matrix, ixes)  
 good\_matrix = max\_on\_diag(matrix)  
 # print(matrix)  
 vec = ixes.copy()  
 if good\_matrix == False:  
 T = find\_transponent(matrix)  
 matrix = matrix\_multiplying(T, matrix)  
 vec = T.dot(ixes)  
 for i in range(len(matrix)):  
 if matrix[i][i] == 0:  
 print('Error: Unable to bring the matrix to the form!')  
 exit()  
 vec2 = vec.copy()  
 matrix, vec = matrix\_transform(matrix, vec2)  
 h = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  
 print('Приближения, полученные методом простой итерации:')  
 answer = simple\_iterating\_func(matrix, h, vec)  
 print(answer)  
  
  
# Итерирует по методу Зейделя, находя вектор - решение СЛАУ  
def zeidel\_iterating\_func(matrix, h, vec):  
 summa = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  
 for i in range(len(summa)):  
 summa[i] = h[i]  
 for i in range(len(matrix)):  
 a = 0  
 for j in range(0, len(matrix[0])):  
 a += matrix[i][j] \* summa[j]  
  
 summa[i] = a + vec[i]  
 # print(summa)  
 for i in range(len(summa)):  
 difer = summa[i] - h[i]  
  
 if math.fabs(difer) > 0.00001:  
 return zeidel\_iterating\_func(matrix, summa, vec)  
 else:  
 return summa  
  
  
# Реализует метод Зейделя  
def zeidel\_iteration\_method(matrix, ixes):  
 check\_zeros\_diag(matrix, ixes)  
 good\_matrix = max\_on\_diag(matrix)  
 # print(matrix)  
 vec = ixes.copy()  
 if good\_matrix == False:  
 T = find\_transponent(matrix)  
  
 matrix = matrix\_multiplying(T, matrix)  
  
 vec = T.dot(ixes)  
  
 for i in range(len(matrix)):  
 if matrix[i][i] == 0:  
 print('Error: Unable to bring the matrix to the form!')  
 exit()  
 matrix, vec = matrix\_transform(matrix, vec)  
 h = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  
 print('Приближения, полученные методом Зейделя:')  
 answer = zeidel\_iterating\_func(matrix, h, vec)  
 print(answer)  
  
  
'''  
print('\n\n')  
AA = A.copy()\*300  
print(AA)  
bb = b.copy()  
#zeidel\_iteration\_method(AA, bb)  
'''  
  
  
# Относительная погрешность  
def inaccurancy(w, x):  
 temp = (abs(w - x) / abs(x))  
 print(f'w = {w}, x = {x}')  
 print("%.10f" % temp)  
  
  
'''  
AA = A.copy()  
aaa = AA[2].copy()  
AA[2] = AA[4].copy()  
AA[4] = aaa.copy()  
aaa = AA[1].copy()  
AA[1] = AA[3].copy()  
AA[3] = aaa.copy()  
bb = b.copy()  
bbb = bb[2]  
bb[2] = bb[4].copy()  
bb[4] = bbb  
bbb = bb[1]  
bb[1] = bb[3].copy()  
bb[3] = bbb  
print('\n\n\n')  
  
print(bb)  
  
print('\n\n\n')  
  
print(np.linalg.solve(AA,bb))  
print('\n\n\n')  
simple\_iteration\_method(AA,bb)  
'''  
'''  
AA = A.copy()  
bb = b.copy()  
  
print('\nЗначения, полученные при помощи встроенной функции')  
print(np.linalg.solve(AA,bb))  
#print('\nЗначения, полученные при помощи метода простой итерации')  
simple\_iteration\_method(AA,bb)  
AA = A.copy()  
bb = b.copy()  
#print('\nЗначения, полученные при помощи метода Зейделя')  
zeidel\_iteration\_method(AA,bb)  
  
'''  
  
print('=============================================================')  
print('Относительная погрешность значений, полученных методом простой итерации')  
inaccurancy(0.76917813, 0.76917766)  
inaccurancy(-1.38299477, -1.38299237)  
inaccurancy(-3.29799396, -3.29799462)  
inaccurancy(-0.06027295, -0.06027127)  
inaccurancy(-0.03223437, -0.03223149)  
print('=============================================================\n')  
  
print('\n=============================================================')  
print('Относительная погрешность значений, полученных методом Зейделя')  
inaccurancy(0.76917813, 0.76917937)  
inaccurancy(-1.38299477, -1.38299604)  
inaccurancy(-3.29799396, -3.29799399)  
inaccurancy(-0.06027295, -0.06027306)  
inaccurancy(-0.03223437, -0.03223498)  
print('=============================================================')  
  
  
'''  
  
print('\n\nТП1\nОдинаковые строки и одинаковые элементы вектора b')  
AA = A.copy()  
bb = b.copy()  
AA[0]=AA[1]  
bb[0] = bb[1]  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
#check\_for\_equal\_rows(AA,bb)  
  
print('\n\nТП2\nОдинаковые строки и разные элементы вектора b')  
AA = A.copy()  
bb = b.copy()  
AA[0]=AA[1]  
bb[0] = bb[1]/2  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
#check\_for\_equal\_rows(AA,bb)  
  
print('\n\nТП5\nМалые значения элементов вектора b')  
AA = A.copy()  
bb = b.copy()  
bb/=1000  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
print('\nЗначения, полученные при помощи встроенной функции')  
print(np.linalg.solve(AA,bb))  
print('\n')  
simple\_iteration\_method(AA,bb)  
AA = A.copy()  
bb = b.copy()  
bb/=1000  
zeidel\_iteration\_method(AA,bb)  
  
print('\n\nТП6\nМалые значения элементов матрицы А')  
AA = A.copy()  
bb = b.copy()  
AA/=10000  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
print('\nЗначения, полученные при помощи встроенной функции')  
print(np.linalg.solve(AA,bb))  
  
simple\_iteration\_method(AA,bb)  
AA = A.copy()  
bb = b.copy()  
AA/=10000  
zeidel\_iteration\_method(AA,bb)  
  
  
print('\n\nТП3\nСтрока со всеми элементами равными нуль и НЕнулевое значение элемента вектора b')  
AA = A.copy()  
bb = b.copy()  
AA[0]=np.zeros(len(A))  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
simple\_iteration\_method(AA,bb)  
  
print('\n\nТП4\nСтрока со всеми элементами равными нулю и нулевое значение элемента вектора b')  
AA = A.copy()  
bb = b.copy()  
AA[0]=np.zeros(len(A))  
bb[0] = 0  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
#zeidel\_iteration\_method(AA,bb)  
  
print('\n\nТП7\nБольшие значения элементов матрицы и вектора b')  
AA = A.copy()  
bb = b.copy()  
bb\*=1000  
AA\*=1000  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
print('\nЗначения, полученные при помощи встроенной функции')  
print(np.linalg.solve(AA,bb))  
  
simple\_iteration\_method(AA,bb)  
AA = A.copy()  
bb = b.copy()  
bb\*=1000  
AA\*=1000  
zeidel\_iteration\_method(AA,bb)  
  
print('\n\nТП8\nБольшие значенияэлементов матрицы А')  
AA = A.copy()  
bb = b.copy()  
AA\*=10000  
print('Матрица:')  
print(AA)  
print('Вектор b')  
print(bb)  
print('\nЗначения, полученные при помощи встроенной функции')  
print(np.linalg.solve(AA,bb))  
  
simple\_iteration\_method(AA,bb)  
AA = A.copy()  
bb = b.copy()  
AA\*=10000  
zeidel\_iteration\_method(AA,bb)  
'''  
  
print('\n\nТП9\nНаибольшие значения в строке располагаются не на главной диагонаи')  
AA = A.copy()  
aaa = AA[2].copy()  
AA[2] = AA[4].copy()  
AA[4] = aaa.copy()  
aaa = AA[1].copy()  
AA[1] = AA[3].copy()  
AA[3] = aaa.copy()  
bb = b.copy()  
bbb = bb[2]  
bb[2] = bb[4].copy()  
bb[4] = bbb  
bbb = bb[1]  
bb[1] = bb[3].copy()  
bb[3] = bbb  
  
print('\n=============================================================\nМатрица:')  
print(AA)  
print('=============================================================\n')  
  
print('=============================================================\nВектор b:')  
print(bb)  
print('=============================================================\n')  
  
print('=============================================================')  
simple\_iteration\_method(AA.copy(), bb.copy())  
print('=============================================================\n')  
  
print('=============================================================')  
zeidel\_iteration\_method(AA, bb)  
print('=============================================================\n')  
  
print('=============================================================\nЗначения, полученные при помощи встроенной функции\nпакета решений numpy:')  
print(np.linalg.solve(AA, bb))  
print('=============================================================')  
  
print('\n\n==========================================================================================\nВывод: метод Зейделя затрачивает меньше итераций до заданной точности,\n'  
 'но метод простых итераций даёт более точный результат вычислений.\n\n'  
 'Это заметно при больших/маленьких значений элементов матрицы или вектора свободных членов,\n'  
 'но не одновременно, если и то, и другое одинаковой разрядности.\n==========================================================================================\n')  
  
# Черновик:Вывод: у Зейделя меньше итераций до нужной точности, но более точный результат дает простая итерация, особенно заметно при больших/маленьких значениях матрицы/вектора своб. чл.  
# не одовременно. Если и то и другое примерно одикаковой разрядности, то без разницы

**Пример работы программы:  
**

****

**  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
4. Выводы**

Таким образом, в ходе выполнения лабораторной работы были изучены итерационные методы решения СЛАУ, а именно, метод простых итераций и метод Зейделя; был составлен алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ; была составлена программа решения СЛАУ по разработанному алгоритму; были решены численно тестовые примеры и проверена их правильность в нашей программе, а также мы сравнили трудоемкости решения методом простых итераций и методом Зейделя. Пришли к выводу, что метод Зейделя уменьшает количество итераций.