Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04

Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №1

Вариант 12

Выполнил: Савельева Диана Александровна

P32082

Преподаватель: Машина Екатерина Алексеевна

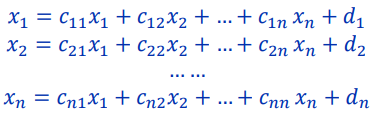
Санкт-Петербург, 2023 г.

***1. Цель работы:***

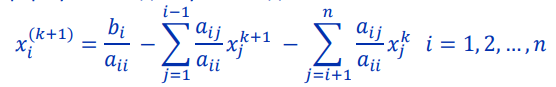
Научиться решать и программировать системы линейных алгебраических уравнений СЛАУ.

***2. Описание метода:*** *Метод Гаусса-Зейделя.*

Метод Гаусса-Зейделя является модификацией метода простой итерации и обеспечивает более быструю сходимость к решению системы. Идея метода: при вычислении компонента 𝑥𝑖 (𝑘+1) вектора неизвестных на (k+1)-й итерации используются 𝑥1 (𝑘+1) , 𝑥2 (𝑘+1) , … , 𝑥𝑖−1 (𝑘+1) , уже вычисленные на (k+1)-й итерации. Значения остальных компонент 𝑥𝑖+1 (𝑘+1) , 𝑥𝑖+2 (𝑘+1) , … , 𝑥𝑛 (𝑘+1) берутся из предыдущей итерации. Так же как и в методе простых итераций строится эквивалентная СЛАУ и за начальное приближение принимается вектор правых частей (как правило, но может быть выбран и нулевой вектор): 𝑥𝑖 0 = (𝑑1, 𝑑2, … , 𝑑𝑛).



Рабочая формула:



Итерационный процесс продолжается до тех пор, пока:

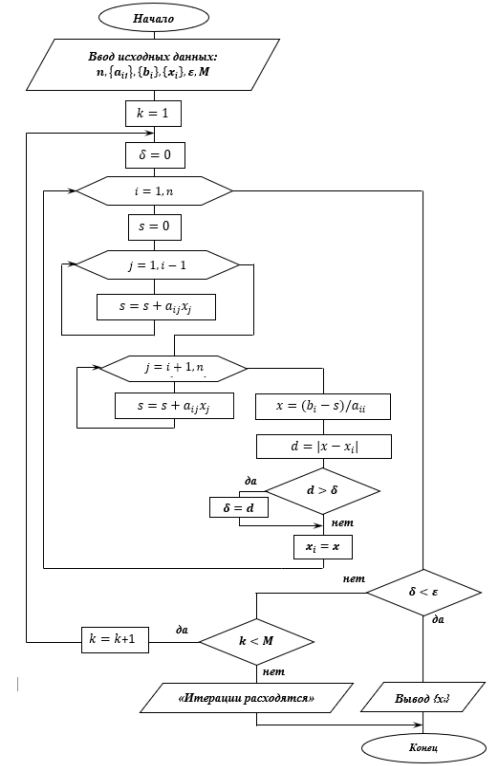


*Достоинства метода:*

* Является универсальным и простым для реализации на ЭВМ.
* Обеспечивает более быструю сходимость (по сравнению с методом постой итерации)

*Недостатки метода:*

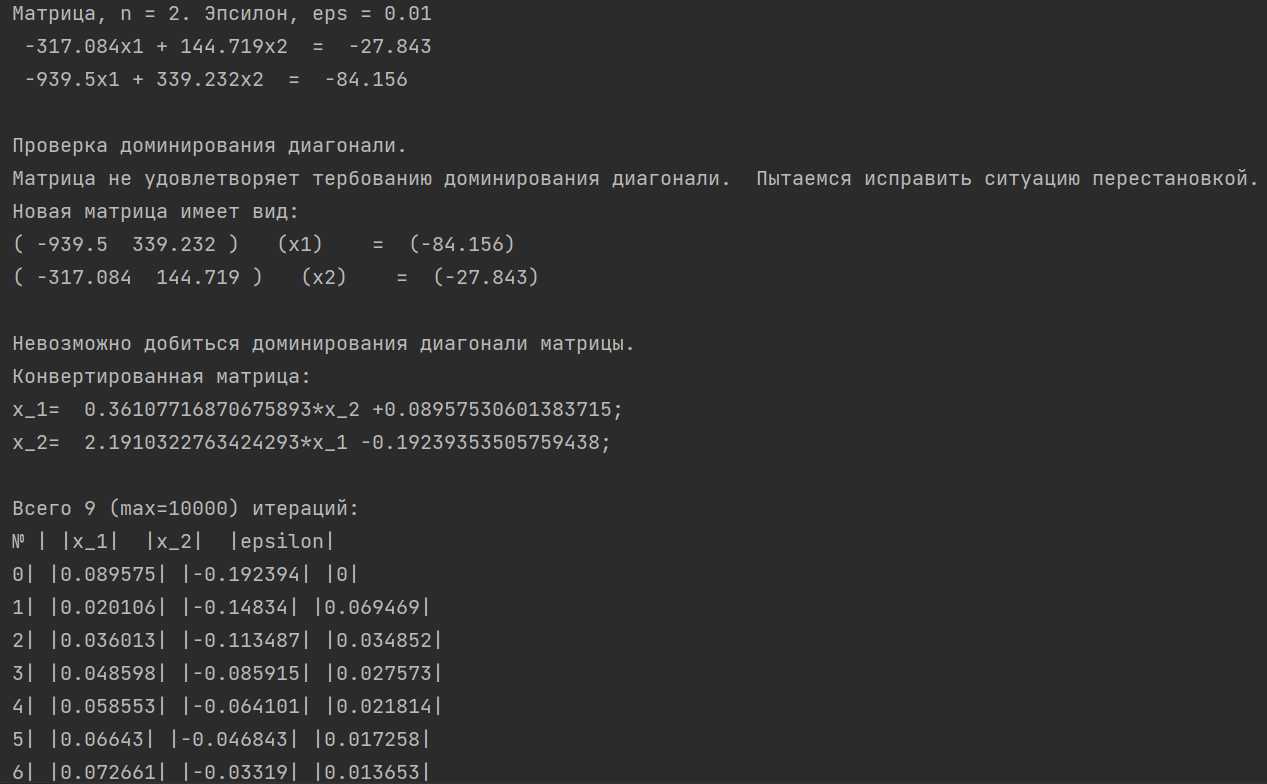
* Является трудоемким

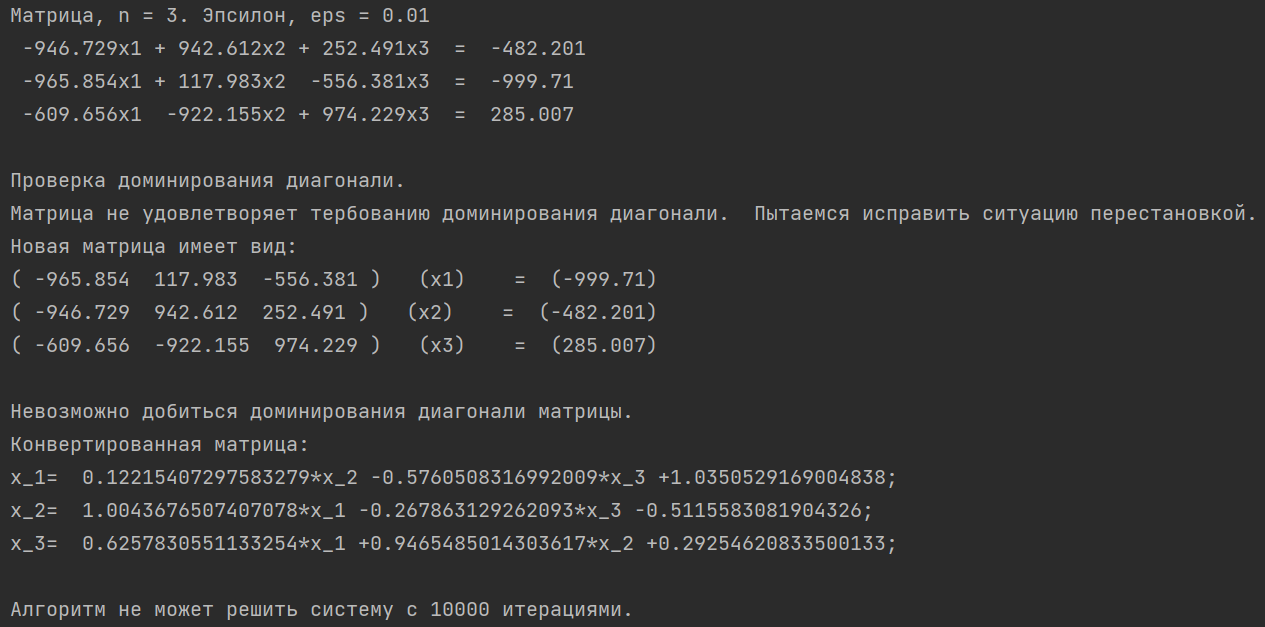


***3. Листинг программы:***

class MatrixMethods:  
  
 def \_\_init\_\_(self, matrix: Matrix):  
 self.\_\_matrix = matrix  
  
 def solve(self):  
 print("Проверка доминирования диагонали. ")  
 if not self.checkDiagonalDominance():  
 print("Матрица не удовлетворяет тербованию доминирования диагонали. "  
 "Пытаемся исправить ситуацию перестановкой.")  
 self.correctDiagonalDominance()  
 if not self.checkDiagonalDominance():  
 print("Невозможно добиться доминирования диагонали матрицы.")  
 else:  
 print("Матрица соответствует требованию доминирования диагонали.")  
 else:  
 print("Матрица соответствует требованию доминирования диагонали.")  
 self.createConvertedMatrix()  
 try:  
 self.algorithmGaussZeidel()  
 except MemoryError as e:  
 print("Memory Error.")  
  
 def checkDiagonalDominance(self) -> bool:  
 dimension = self.\_\_matrix.getDimension()  
 matrix = self.\_\_matrix.getMatrix()  
 for row in range(dimension):  
 row\_sum = 0  
 for column in range(dimension):  
 if row != column:  
 row\_sum += abs(matrix[row][column])  
 if abs(matrix[row][row]) > row\_sum:  
 continue  
 return False  
 return True  
  
 def correctDiagonalDominance(self):  
 *"""Исправить матрицу в соответствии с правилом преобладания диагональных элементов."""* matrix: Matrix = self.\_\_matrix.getMatrix()  
 dimension: int = self.\_\_matrix.getDimension()  
 for column in range(dimension):  
 diagonal\_element = matrix[column][column]  
 dominance\_row\_number = column  
 for row in range(column, dimension):  
 if abs(matrix[row][column]) > abs(diagonal\_element):  
 diagonal\_element = abs(matrix[row][column])  
 dominance\_row\_number = row  
 matrix[dominance\_row\_number], matrix[column] = matrix[column], matrix[dominance\_row\_number]  
 self.\_\_matrix.setMatrix(matrix)  
 print("Новая матрица имеет вид:")  
 print(self.\_\_matrix.createViewStrMatrix())  
  
 def createConvertedMatrix(self):  
 matrix = self.\_\_matrix.getMatrix()  
 dimension = self.\_\_matrix.getDimension()  
 extended\_dimension = self.\_\_matrix.getExtendedDimension()  
 for row in range(dimension):  
 diagonal\_element = matrix[row][row]  
 for column in range(extended\_dimension):  
 if column == extended\_dimension - 1:  
 matrix[row][column] /= diagonal\_element  
 continue  
 if row == column:  
 matrix[row][column] = 0  
 continue  
 matrix[row][column] = (-matrix[row][column]) / diagonal\_element  
 self.\_\_matrix.setConvertedMatrix(matrix)  
 print(self.\_\_matrix.createViewStrConvertedMatrix())  
  
 def algorithmGaussZeidel(self):  
 converted\_matrix = self.\_\_matrix.getConvertedMatrix()  
 dimension = self.\_\_matrix.getDimension()  
 extended\_dimension = self.\_\_matrix.getExtendedDimension()  
 k = 0  
 eps = self.\_\_matrix.getEpsilon()  
 if eps < 1:  
 maxIterationNumber = int(100 / eps)  
 else:  
 maxIterationNumber = int(100 \* eps)  
 iterations = [[0.0 for x in range(extended\_dimension)]  
 for x in range(maxIterationNumber)]  
 for row in range(dimension):  
 iterations[k][row] = converted\_matrix[row][dimension]  
 iterations[k][extended\_dimension - 1] = 0  
 k = k + 1  
 for column in range(dimension):  
 iterations[k][column] = iterations[k - 1][column]  
 try:  
 while True:  
 for row in range(dimension):  
 s = 0  
 for column in range(extended\_dimension):  
 if column == row:  
 continue  
 if column == dimension:  
 s = s + converted\_matrix[row][column]  
 break  
 s = s + converted\_matrix[row][column] \* iterations[k][column]  
 iterations[k][row] = s  
 max\_accuracy = abs(iterations[k][0] - iterations[k - 1][0])  
 for column in range(dimension):  
 if abs(iterations[k][column] - iterations[k - 1][column]) > max\_accuracy:  
 max\_accuracy = abs(iterations[k][column] - iterations[k - 1][column])  
 iterations[k][extended\_dimension - 1] = max\_accuracy  
 k += 1  
 if max\_accuracy <= eps:  
 break  
 for column in range(dimension):  
 iterations[k][column] = iterations[k - 1][column]  
 except IndexError as e:  
 print(f"Алгоритм не может решить систему с {k} итерациями.")  
 k = 20  
  
 print(f"Всего {k} (max={maxIterationNumber}) итераций:")  
 for row in range(k):  
 if row == 0:  
 print(f"№ |", end='')  
 for col in range(extended\_dimension):  
 if col == extended\_dimension - 1:  
 print(f" |epsilon|", end='')  
 break  
 print(f" |x\_{(col + 1)}| ", end='')  
 print("")  
 print(f"{row}| ", end='');  
 for var in range(extended\_dimension):  
 print(f"|{round(iterations[row][var], 6)}| ", end='')  
 print()

***4. Примеры и результаты работы программы***

******

******

***5. Выводы:***

В ходе лабораторной работы я научилась использовать прямые и итерационные методы решения нелинейных СЛАУ, написала программу.