УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники  
Направление подготовки 09.03.04 Программная инженерия  
Дисциплина «Вычислительная математика»

Отчёт  
По лабораторной работе №1  
  
Вариант №15

Студент   
Кириенко Никита Алексеевич  
Группа P3212

Преподаватель  
Малышева Татьяна Алексеевна

Санкт-Петербург, 2021 год

**Цель работы**

Цель работы: реализовать численный метод для решения системы линейных алгебраических уравнений методом простых итераций.

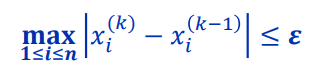
1. № варианта определяется как номер в списке группы согласно ИСУ.
2. В программе численный метод должен быть реализован в виде от-дельной подпрограммы или класса, в который исходные данные передаются в качестве параметров, выходные - тоже (либо возвращаемое значение).
3. Размерность матрицы n<=20 (задается из файла или с клавиатуры - по выбору конечного пользователя).
4. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).
5. Обязательно: Тестовые данные на матрице большого размера (5\*5 / 6\*6...) + в отчёт с решением.

Для итерационных методов должно быть реализовано:

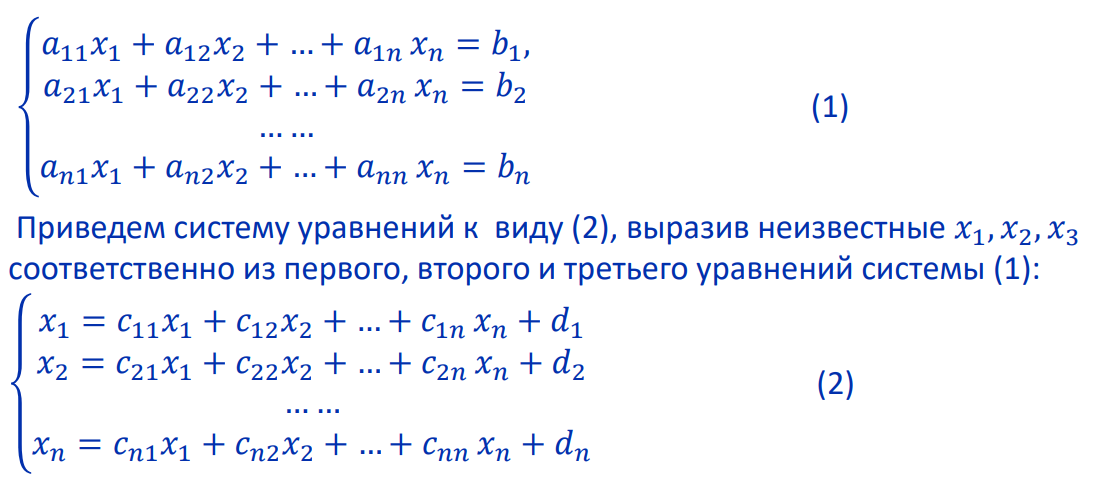
* Точность задается с клавиатуры/файла
* Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
* Вывод вектора неизвестных:
* Вывод количества итераций, за которое было найдено решение.
* Вывод вектора погрешностей:

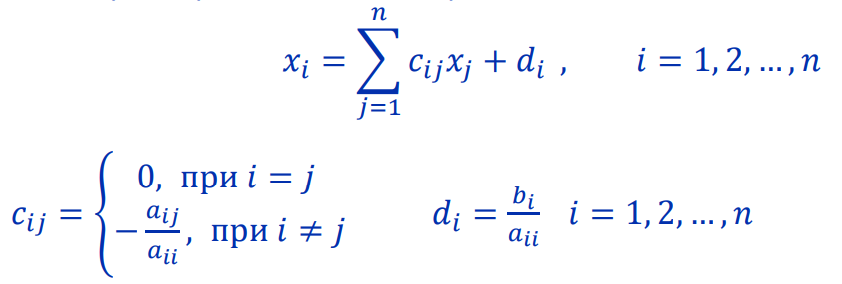
**Описание метода, расчетные формулы**

Расчеты ведутся итерационно, пока не будет достигнут критерий окончания – желаемая точность. В данной работе используется критерий по абсолютным отклонениям, наиболее простой и часто используемый способ – это сравнение между собой соответствующих неизвестных по двум соседним итерациям (k) и (k-1):

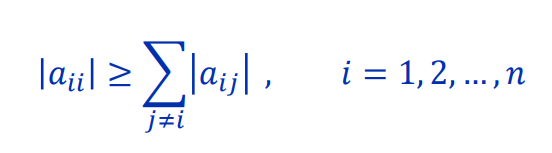


Имея матрицу коэффициентов, преобразуем ее к другому виду и вычисляем значения вектора неизвестных X по формуле (на основе изначальных значений X = 0 или D):





Перед началом расчетов проверяем изначальную матрицу на выполнение условия преобладания диагональных элементов:



Если таковое отсутствует, пытаемся расставить ряды так, чтобы оно появилось. Если не удается, завершаем работу.

**Листинг программы**

(некоторые части опущены, так как не представляют интереса)

matrix.py

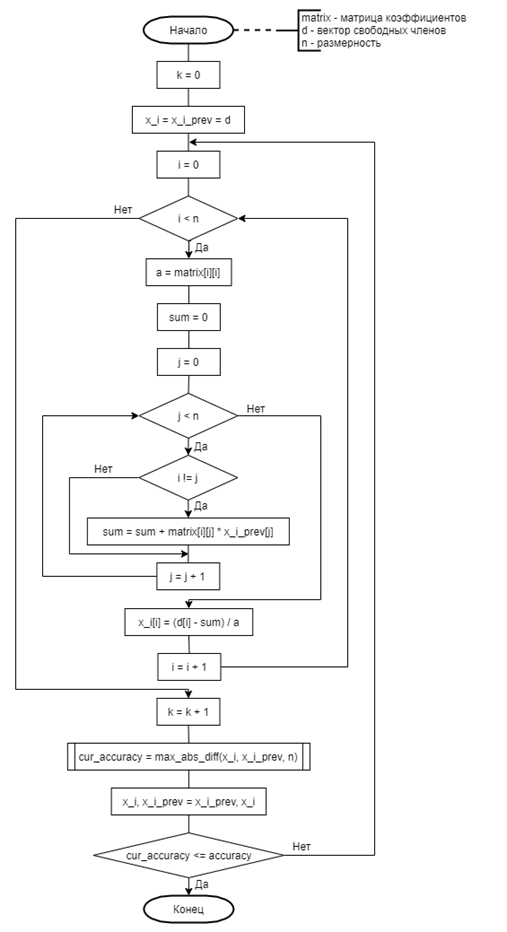
|  |
| --- |
| def find\_dominant\_index(coefficients: []) -> (int, bool):for i in range(len(coefficients)):  coefficient\_abs\_sum = sum(list(map(abs, coefficients)))  diff = 2 \* abs(coefficients[i]) - coefficient\_abs\_sum  if diff >= 0:  return i, diff != 0  return None, False   class Matrix:  def \_\_init\_\_(self, size: int, data: [], right=None):  ...  ...  @property  def diagonally\_dominant(self) -> bool:  at\_least\_one\_strict = False  for row in range(self.size):  (dominant\_index, strict) = find\_dominant\_index(self.coefficients[row])  if dominant\_index != row:  return False  if strict:  at\_least\_one\_strict = True  return True and at\_least\_one\_strict   def make\_diagonally\_dominant(self) -> bool:  new\_data = [None] \* self.size  new\_right = [0] \* self.size  for row in range(self.size):  *# find index of dominant element in row* (dominant\_index, \_) = find\_dominant\_index(self.coefficients[row])  *# if present and unique, add to new data* if (dominant\_index is not None) and (new\_data[dominant\_index] is None):  new\_data[dominant\_index], new\_right[dominant\_index] = self.coefficients[row], self.right[row]  else:  return False  *# save changes* self.coefficients, self.right = new\_data, new\_right  return True |

Полный листинг: <https://github.com/notgurev/se-compmaths-lab-1> (ниже еще solve.py, экономлю место)

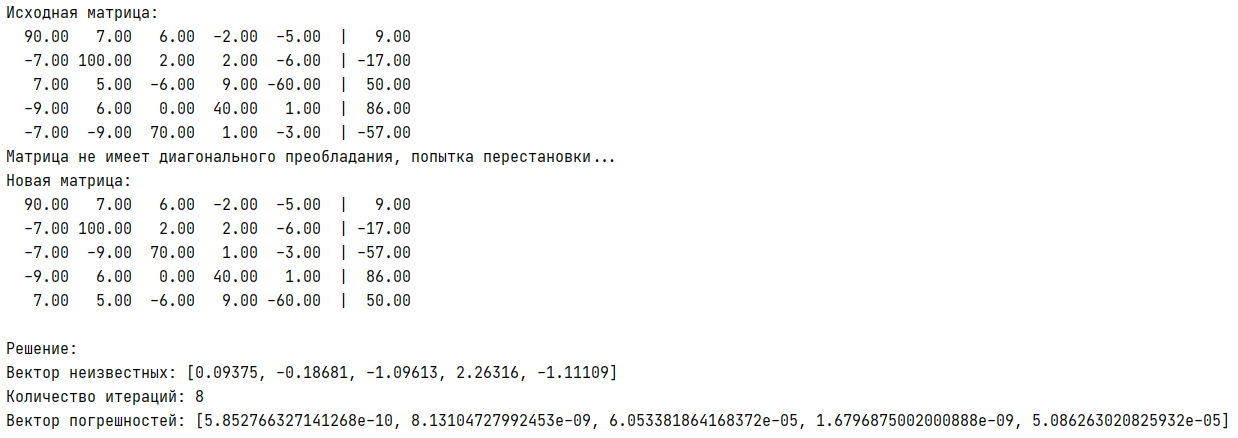
solve.py

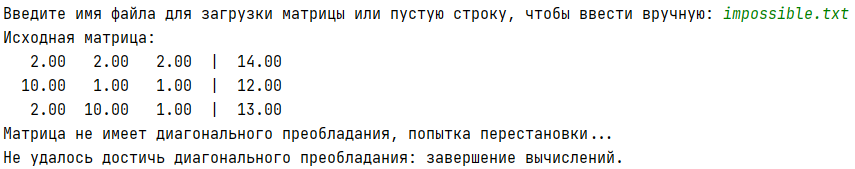
|  |
| --- |
| class Solution:  def \_\_init\_\_(self, x\_vector, iterations, error\_vector):  ...  ...   def solve(matrix: Matrix, precision):  x\_vector = [0] \* matrix.size  *# x\_vector = matrix.right*  *# проверка диагонального преобладания* if not matrix.diagonally\_dominant:  print(**'Матрица не имеет диагонального преобладания, попытка перестановки...'**)  if not matrix.make\_diagonally\_dominant():  print(**'Не удалось достичь диагонального преобладания: завершение работы.'**)  exit(-1)  else:  print(**'Новая матрица: '**)  print(matrix)   *# вычисление матрицы C* c\_matrix = []  for row in range(matrix.size):  a\_ii = matrix.coefficients[row][row]  row\_vector = []  for column in range(matrix.size):  c = -matrix.coefficients[row][column] / a\_ii if row != column else 0  row\_vector.append(c)  c\_matrix.append(row\_vector)   *# итерации* iterations = 0  error\_vector = [0] \* len(x\_vector)  while True:  iterations += 1  for i in range(len(x\_vector)):  a = matrix.coefficients[i][i]  b = matrix.right[i]  cx\_array\_sum = sum(map(lambda c\_ij: c\_ij \* x\_vector[i], c\_matrix[i]))   prev = x\_vector[i]  x\_vector[i] = cx\_array\_sum + b / a  error\_vector[i] = abs(x\_vector[i] - prev)   if max(error\_vector) <= precision:  break   return Solution(x\_vector, iterations, error\_vector) |

**Блок-схема численного метода**



**Примеры и результаты работы**





**Вывод**

Вывод: Я научился решать системы линейных алгебраических уравнений методом простых итераций и реализовал данный чиленный метод в программе на Python.