**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э.Баумана)**

**Мытищинский филиал**

**ФАКУЛЬТЕТ КОСМИЧЕСКИЙ**

**КАФЕДРА К-1 САУ**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**ПО ДИСЦИПЛИНЕ**

**“Численные методы”**

**НА ТЕМУ:**

**«Решение систем нелинейных уравнений методом итераций и методом Зейделя»**

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**Студент К1-61Б**  **28.04.24 Тимофеев К. А.**

(Группа) (Подпись, дата) (ФИО)

**Руководитель**  **Чернова Т.В.**

(Подпись, дата) (ФИО)

*2024 г.*

# Задачи

1. Найти решение системы методом простых итераций



1. Найти решение системы методом простых итераций



1. Найти решение системы методом Зейделя



1. Найти решение системы методом Зейделя



1. Подтвердить корректность решения системы



методом обратной матрицы.

1. Подтвердить корректность решения системы



методом обратной матрицы.

# Теория

## Метод простых итераций



# Ход работы

Напишем функцию для поиска корней СЛАУ методом простых итераций

Используется модуль numpy.matrix, подробнее о нём в документации:

* cs.mipt.ru/advanced\_python/lessons/lab16.html#numpy-1
* https://physics.susu.ru/vorontsov/language/numpy.html

def SimpleIterationMethod(system : np.matrix[float], epsilon = 0.001):

    delta = np.matrix([[None], [None], [None]])

    x = np.matrix([[0.0], [0.0], [0.0]])

    max = [None]

    comment = ["(˘･\_･˘)"]

    i : int = 0 # current X collunm

    while True:

        newRow = np.matrix([

            [system[0, 1] \* x[1, i] + system[0, 2] \* x[2, i]],

            [system[1, 0] \* x[0, i] + system[1, 2] \* x[2, i]],

            [system[2, 0] \* x[0, i] + system[2, 1] \* x[1, i]]

                           ])

        newRow = np.divide(np.subtract(system[:3, 3:], newRow),

                            np.vstack(( system[0, 0],

                                        system[1, 1],

                                        system[2, 2])))

        x = np.hstack((x, newRow))

        i+=1

        delta = np.hstack((delta, abs(np.subtract(x[:3, i], x[:3, i-1]))))

        max.append(np.max(delta[:3, i:i+1]))

        if  max[len(max)-1] < epsilon:

            comment.append("(ﾉ◕ヮ◕)ﾉ\*:･ﾟ✧")

            break

        comment.append("(˘･\_･˘)")

    return  pd.DataFrame({…

Аналогично для метода Зейделя

def ZeidelMethod(system : np.matrix[float], epsilon = 0.001):

    delta1 = [None]; delta2 = [None]; delta3 = [None]

    x1 = [0.0];      x2 = [0.0];      x3 = [0.0]

    max = [None]

    comment = ["(˘･\_･˘)"]

    i : int = 0 # current Xi collunm

    while True:

        x1.append((system[0, 3] - (system[0, 1] \* x2[i] + system[0, 2] \* x3[i])) / system[0, 0])

        x2.append((system[1, 3] - (system[1, 0] \* x1[i+1] + system[1, 2] \* x3[i])) / system[1, 1])

        x3.append((system[2, 3] - (system[2, 0] \* x1[i+1] + system[2, 1] \* x2[i+1])) / system[2, 2])

        i+=1

        delta1.append(abs(x1[i] - x1[i-1]))

        delta2.append(abs(x2[i] - x2[i-1]))

        delta3.append(abs(x3[i] - x3[i-1]))

        max.append(np.max([delta1[i], delta2[i], delta3[i]]))

        if  max[len(max)-1] < epsilon:

            comment.append("(ﾉ◕ヮ◕)ﾉ\*:･ﾟ✧")

            break

        comment.append("(˘･\_･˘)")

    return  pd.DataFrame({

Матрицы систем уравнений по заданной системе

matrix = np.matrix([

    [0.63, 0.05, 0.15, 0.34],

    [0.05, 0.34, 0.10, 0.32],

    [0.15, 0.10, 0.70, 0.72]

])

и по варианту

matrix12 = np.matrix([

    [45.00, -3.50, 7.400, 25.00],

    [3.100, -6.00, -2.30, -15.0],

    [0.800, 7.400, -5.00, 64.00]

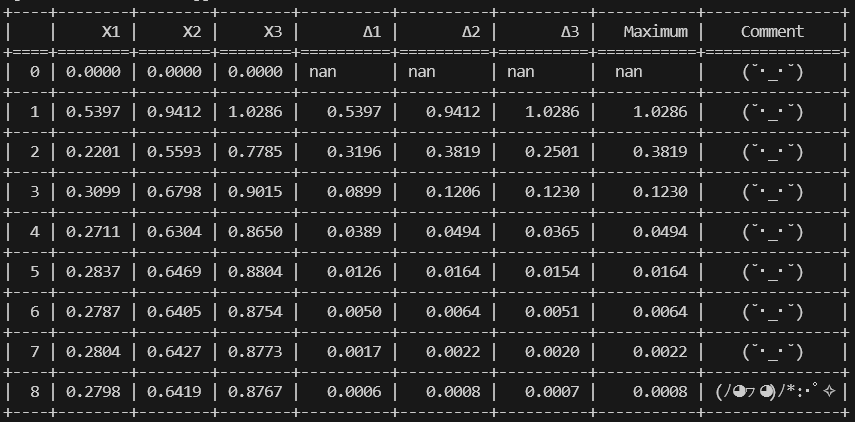
])

## Задание 1

solutionSIM = SimpleIterationMethod(matrix)

print(tab.tabulate(solutionSIM, headers=solutionSIM.keys(), tablefmt="grid", stralign='center', floatfmt=".4f" ))

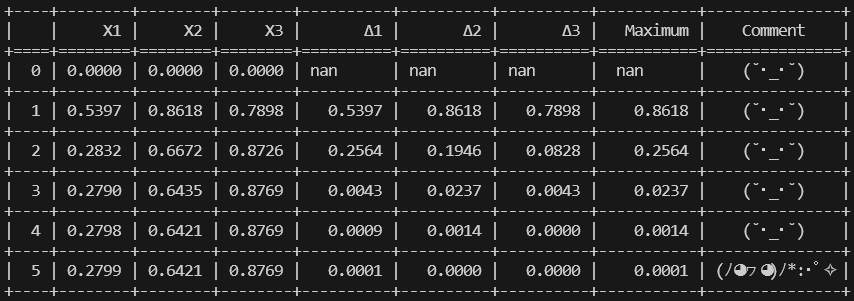
Здесь и далее используется модуль tabulate (as tab) для более эстетичного вывода таблиц решения в консоль.



## Задание 2

solutionZeidel = ZeidelMethod(matrix)

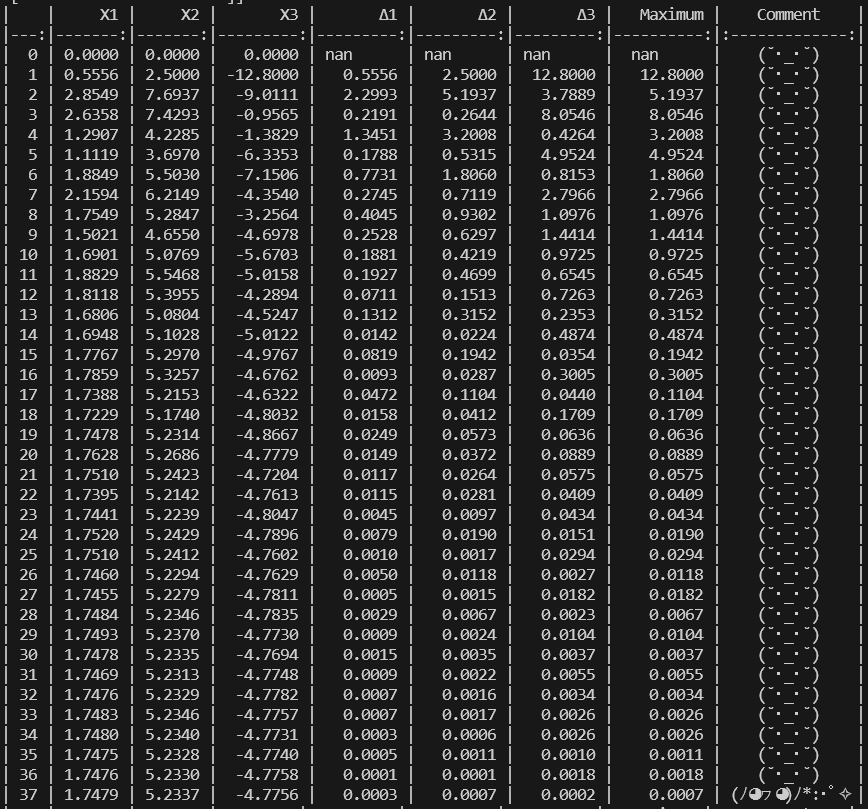
print(tab.tabulate(solutionZeidel, headers=solutionZeidel.keys(), tablefmt="grid", stralign='center', floatfmt=".4f" ))



## Задание 3

solutionSIM12 = SimpleIterationMethod(matrix12)

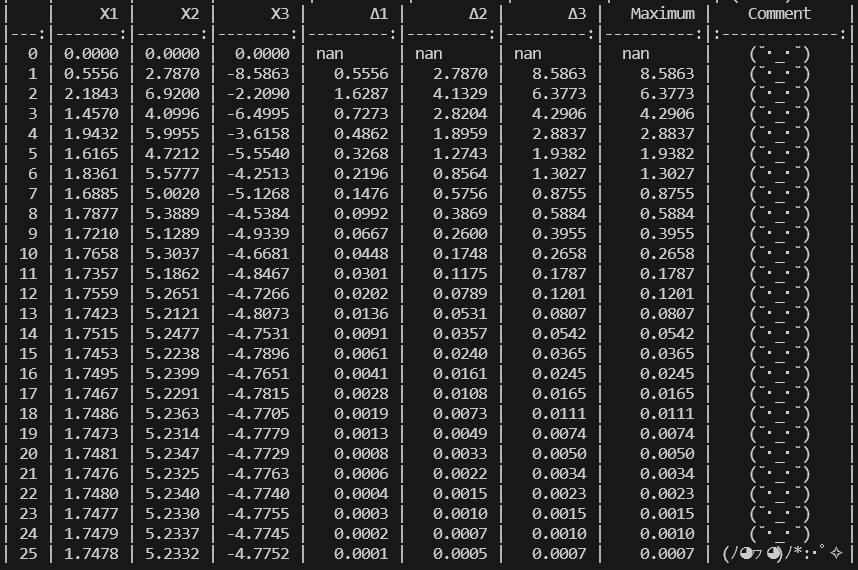
print(tab.tabulate(solutionSIM12, headers=solutionSIM12.keys(), tablefmt="pipe", stralign='center', floatfmt=".4f" ))



## Задание 4

solutionZeidel12 = ZeidelMethod(matrix12)

print(tab.tabulate(solutionZeidel12, headers=solutionZeidel12.keys(), tablefmt="pipe", stralign='center', floatfmt=".4f" ))



## Задание 5

В модуле numpy языка Python есть библиотека linalg позволяющая решать задачи линейной алгебры. Воспользуемся им для подтверждения корректности вышеуказанных алгоритмов.

def invertMatrixMethod(system):

    #находим обратную матрицу среза по первым трем столбцам системы(Aij)

    # и умножаем на последний столбец (B)

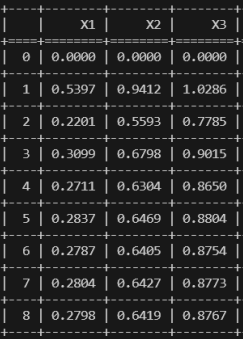
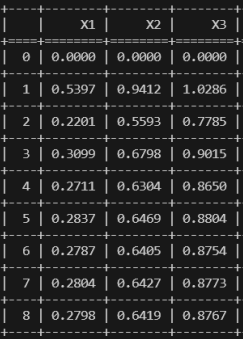
    return np.linalg.inv(system[:3, :3])\*system[:3, 3:4]

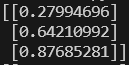
print(invertMatrixMethod(matrix))

print(invertMatrixMethod(matrix12))

Тогда т.к. решение методом простых итераций

и решение методом обратной матрицы

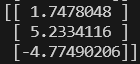


совпадают, то решение считаем действительным.

## Задание 6

Аналогично предыдущему.





## Выводы

Метод Зейделя действительно подтверждает заявленную в теории экономичности по итерациям в своей работе. Оба метода работает штатно, что доказывает корректность алгоритмов.