МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий математики и механики**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

**ОТЧЕТ**

По курсу «Вычислительные методы»

На тему:

**Методы решения СЛАУ**

**Выполнил**:студент группы 381806-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Напылов Е.И.

Подпись

**Научный руководитель**:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Эгамов А.И.

Подпись

Нижний Новгород

2020

# Содержание

[Содержание 2](#_Toc59302982)

[Введение 3](#_Toc59302983)

[Постановка задачи 4](#_Toc59302984)

[Алгоритмы решения СЛАУ 5](#_Toc59302985)

[Метод Крамера 5](#_Toc59302986)

[Метод Гаусса 6](#_Toc59302987)

[Метод простых итераций 7](#_Toc59302988)

[Метод Зейделя 8](#_Toc59302989)

[Метод верхних релаксаций 9](#_Toc59302990)

[Метод Жордана-Гаусса 10](#_Toc59302991)

[Руководство к программе 11](#_Toc59302992)

[Генерация случайной системы 11](#_Toc59302993)

[Ввод данных с клавиатуры 11](#_Toc59302994)

[Тестирование 12](#_Toc59302995)

[Тест ввода с клавиатуры 12](#_Toc59302996)

[Тест случайных данных 13](#_Toc59302997)

[Заключение 14](#_Toc59302998)

[Список литературы 15](#_Toc59302999)

[Приложение 16](#_Toc59303000)

[main.py 16](#_Toc59303001)

[methods.py 17](#_Toc59303002)

[test.py (юнит тесты) 22](#_Toc59303003)

# Введение

При решении практических задач очень часто возникает необходимость в решении систем линейных алгебраических уравнений. СЛАУ используются в компьютерной графике, физических задачах, задачах оптимизации и т.п.

Методы решения СЛАУ можно разделить на 2 вида – точные и итерационные.

Точные методы позволяют получить значение неизвестных с точностью, которая ограничивается только представлением числа в компьютере. Однако, такой высокой точности можно достичь только при тщательной разработке программы.

Итерационные методы основаны на последовательном приближении вектора неизвестных к точному решению. Итерационные алгоритмы работают до тех пор, пока не будет достигнута заданная точность (или пока не будет превышен лимит итераций).

# Постановка задачи

Требуется разработать программу, которая позволит сравнить различные методы решения СЛАУ:

1. Метод Крамера
2. Метод Гаусса
3. Метод Зейделя
4. Метод простых итераций
5. Метод верхних релаксаций
6. Метод Жордана-Гаусса (мой выбор)

Программа должна обладать следующими функциями:

1. Ввод системы с клавиатуры
2. Генерация случайной системы
3. Решение системы
4. Замер времени работы для сравнения алгоритмов

# Алгоритмы решения СЛАУ

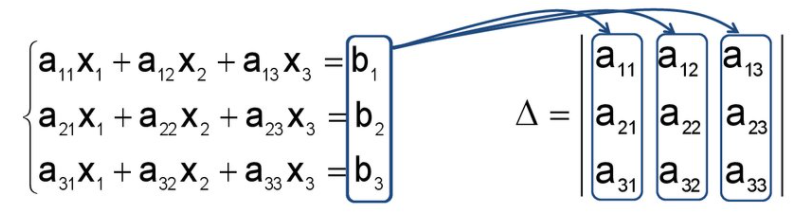
## Метод Крамера

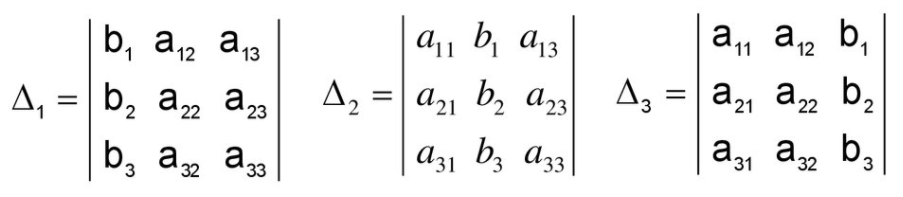
Метод Крамера основан на вычислении определителей матрицы (отсюда следует его высокая вычислительная сложность)

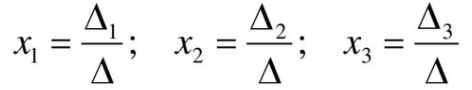
Если определитель матрицы не равен нулю, то система уравнений имеет единственное решение, которое вычисляется по следующему алгоритму.

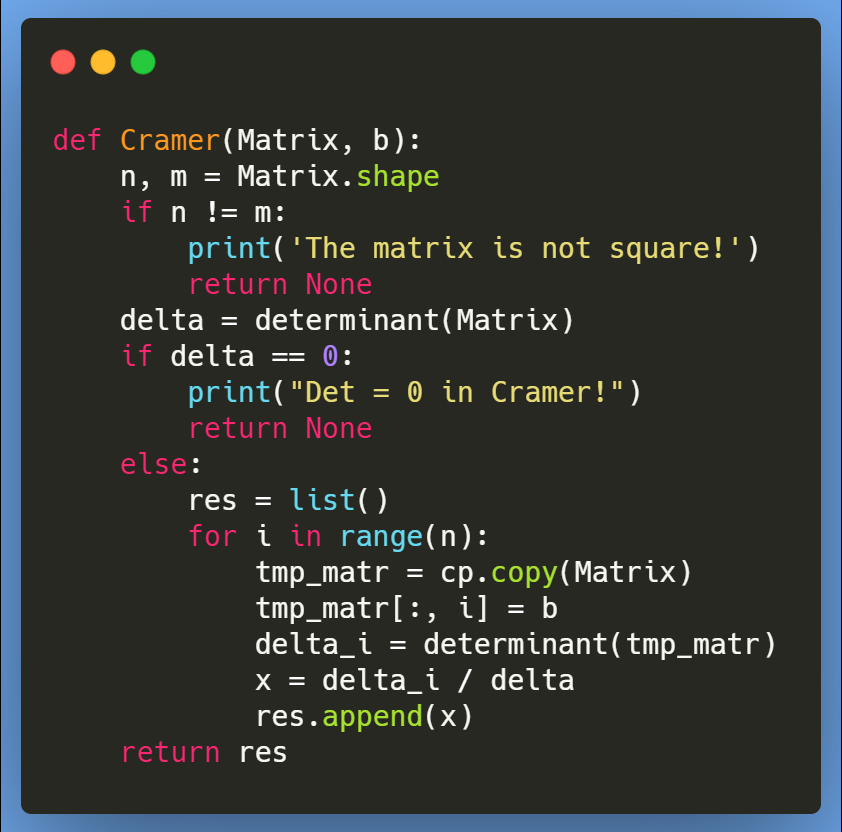
Алгоритм:

1. Вычисляется определитель Δ исходной матрицы
2. Столбец свободных членов (правая часть) поочередно подставляется вместо i-го столба матрицы. Вычисляется определитель Δi полученной матрицы
3. Вектор неизвестных вычисляется по формуле xi= Δ / Δi









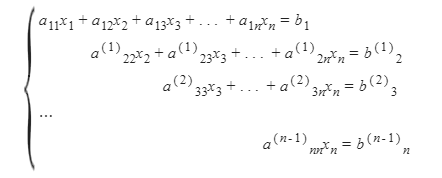
## Метод Гаусса

Метод Гаусса основан на элементарных преобразования строк матрицы.

Алгоритм состоит из двух частей – прямого и обратного хода.

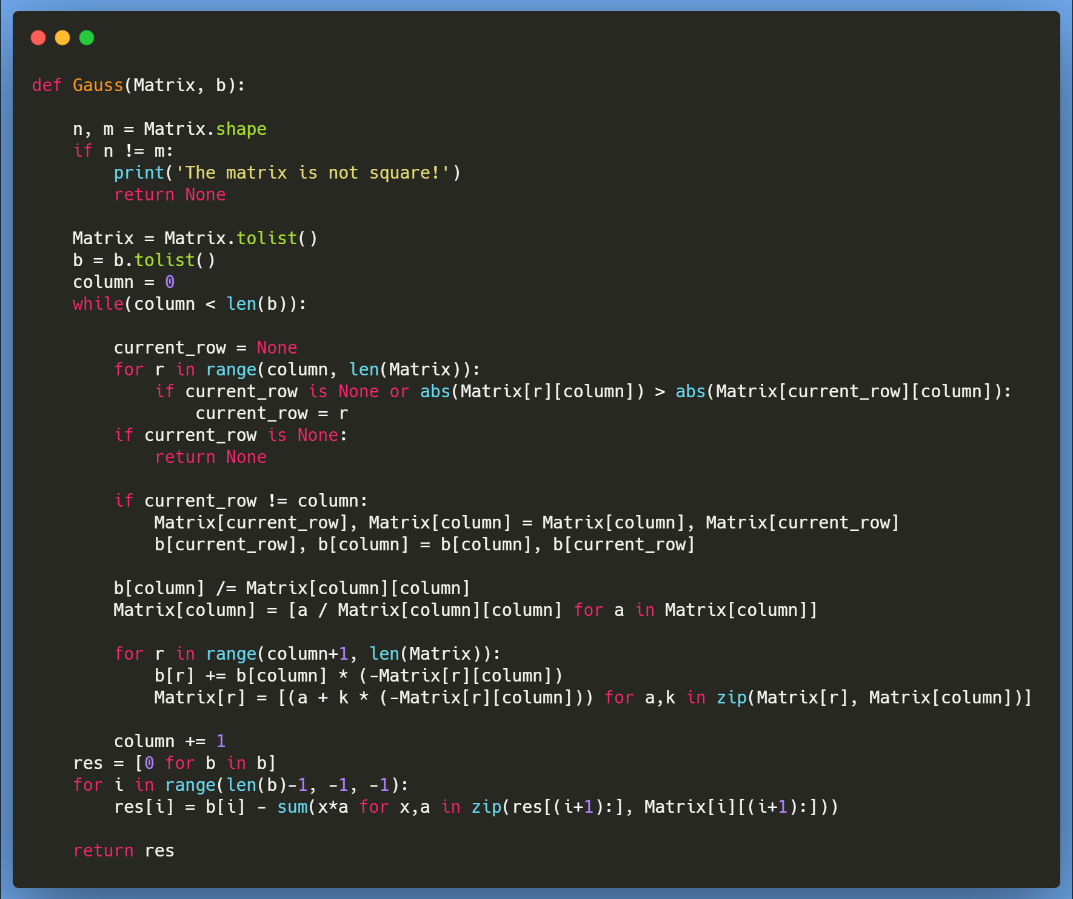
Прямой ход – преобразование к верхне-треугольному виду:

1. Выбирается строка с наибольшим по модулю коэффициентом перед текущей переменной
2. К остальным строкам прибавляется эта строка, умноженная на обнуляющий коэффициент



Обратный ход – вычисление неизвестных:

1. Вычисляется последний неизвестный
2. Вычисляется предпоследний неизвестный с использованием найденных
3. И так далее до первого неизвестного



## Метод простых итераций

Смысл данного метода заключается в последовательном приближении к точному решению.

Алгоритм работает до тех пор, пока не будет достигнута заданная точность или не будет превышено максимальное число итераций.

В качестве начального приближения обычно выбирают столбец свободных членов.

Перед началом итерирования необходимо привести систему



к следующему виду:



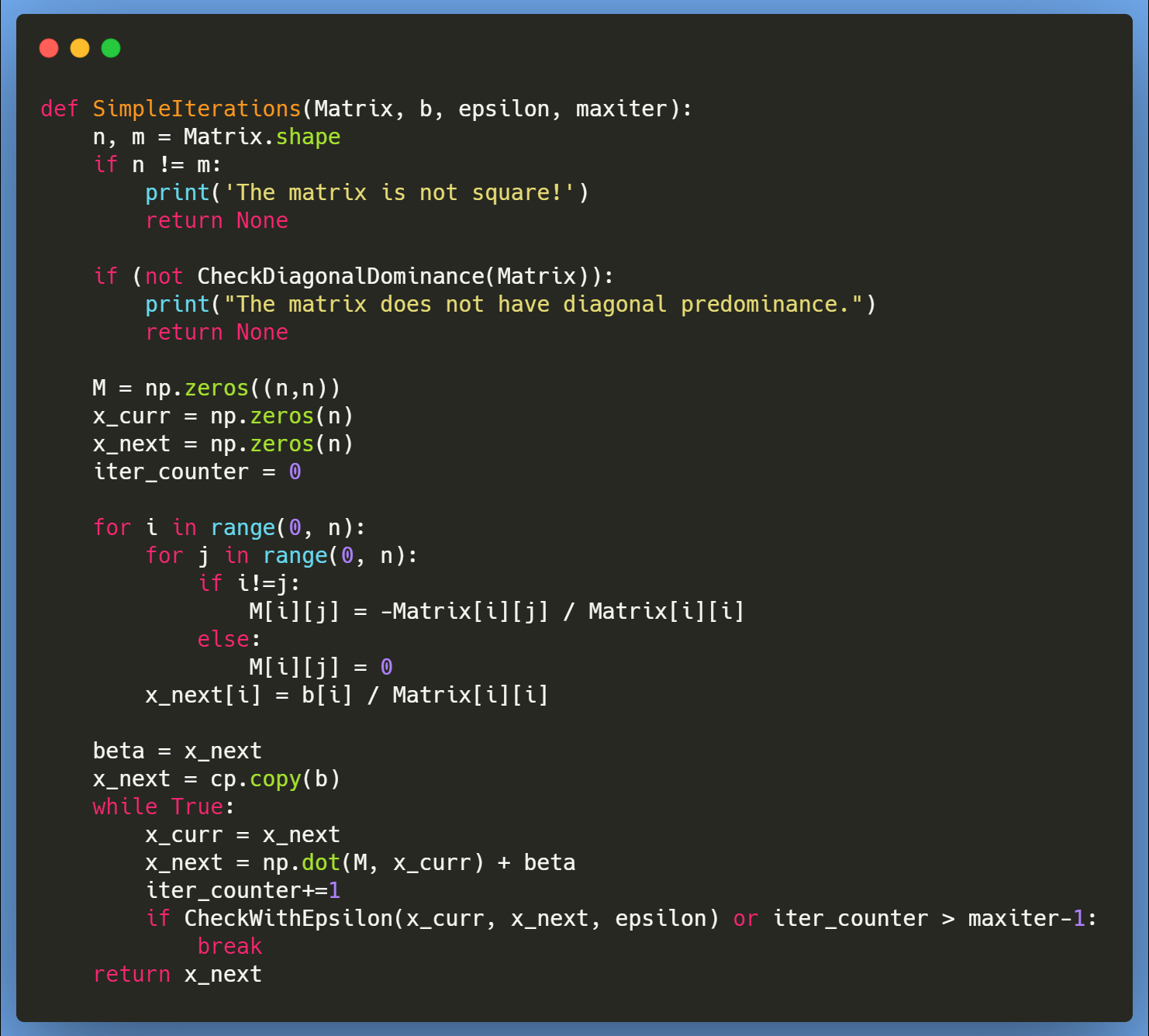
Решение системы находится как предел последовательности:



Сравнение точности производится по формуле:



Если норма матрицы B меньше 1, то система имеет единственное решение и итерационный процесс сходится со скоростью геометрической прогрессии.



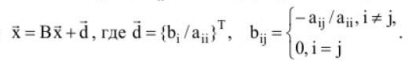
## Метод Зейделя

Данный метод является модификацией метода Якоби. Отличие заключается в том, что для приближения текущего хi используются приближения хi-1…x1.

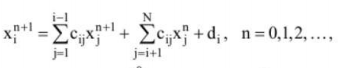
Перед начало итерирования систему из вида:



нужно привести к виду:



Приближение неизвестных производится по формуле:





## Метод верхних релаксаций

Метод имеет параметр ω, изменяя который можно получить различную скорость сходимости итерирования.

Смысл алгоритма показан в следующей формуле:

\frac{(D+\omega L)(x^{(s+1)}-x^{(s)})}{\omega}+Ax^{(s)}=b

Здесь D – диагональная матрица, с диагональю равной диагонали исходной,

L – матрица с под диагональными элементами из матрицы A и нулевой диагональю,

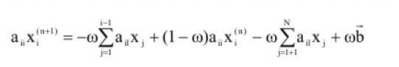
R - матрица с над диагональными элементами из матрицы A и нулевой диагональю,

A = L + D + R

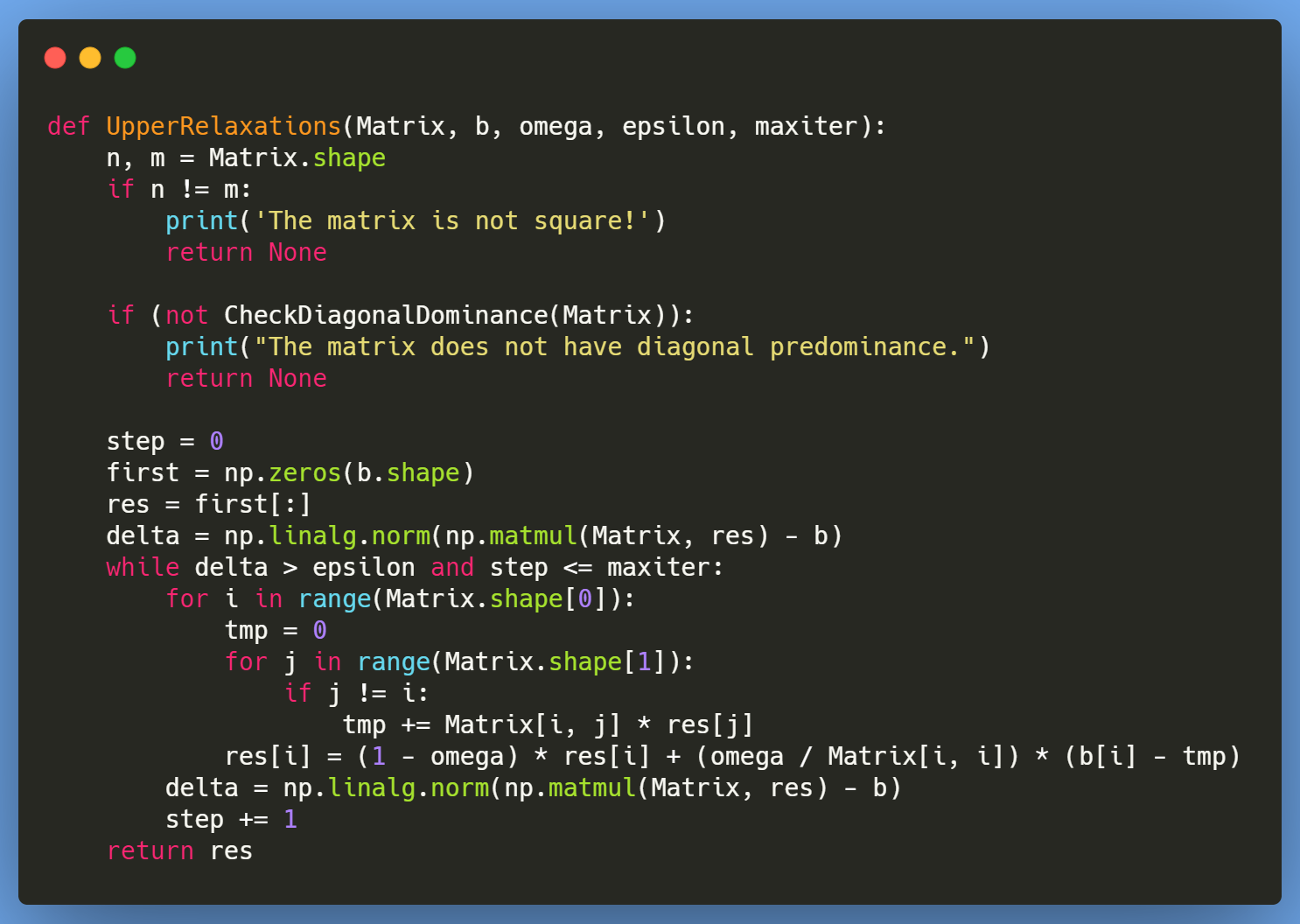
Для верхней релаксации ω лежит в интервале (1…2)

От выбора параметра зависит скорость сходимости итерационного процесса.

Итерирование происходит по формуле:



Стоит отметить, что данный метод не требует второго вектора для хранения приближения.



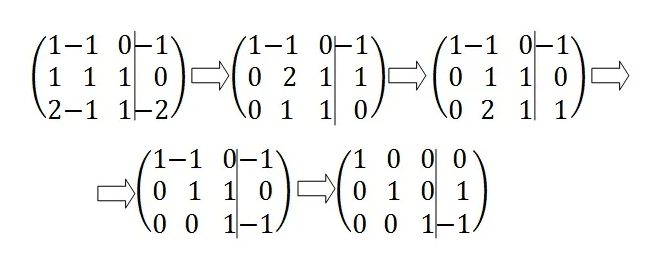
## Метод Жордана-Гаусса

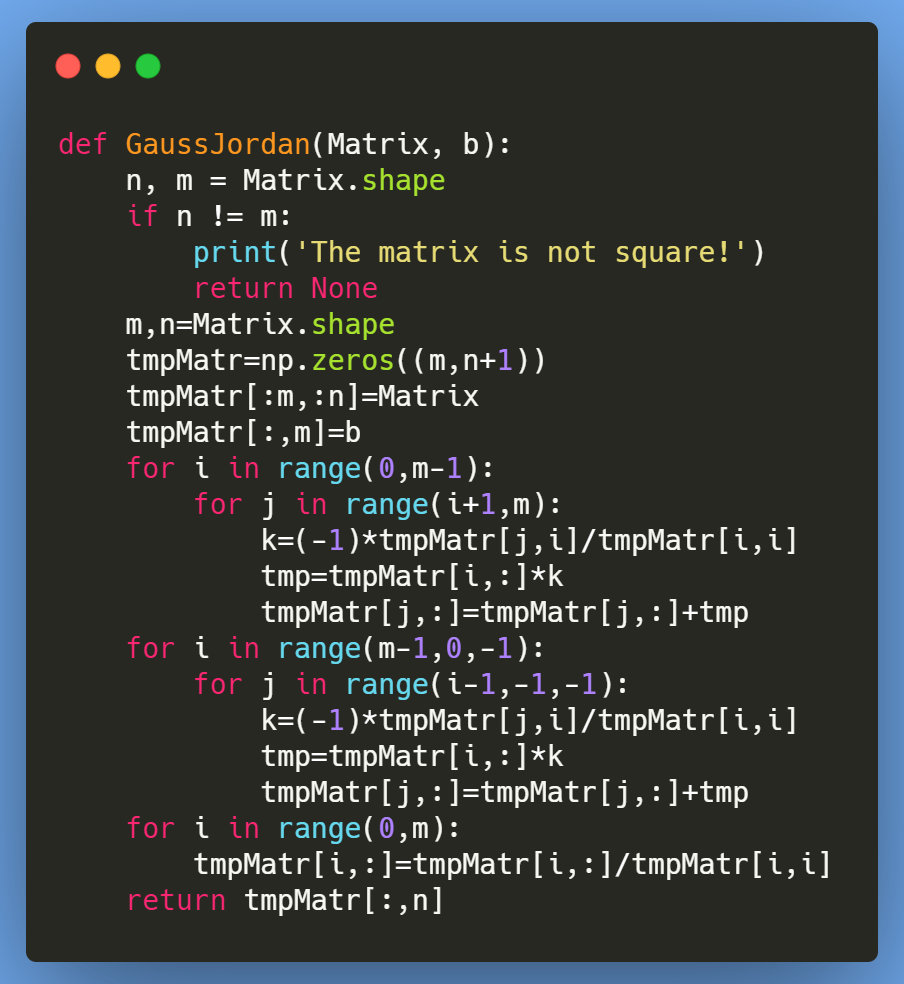
Идея данного метода заключается в приведении матрицы системы к диагональной (далее единичной) путем элементарных преобразований.

Первый этап метода Жордана-Гаусса аналогична методу Гаусса (прямой ход Гаусса).

Второй этап (обратный ход) метода Жордана-Гаусса заключается в обнулении всех элементов матрицы коэффициентов системы линейных уравнений, выше ведущих элементов.

Удобнее показать алгоритм на следующем примере:

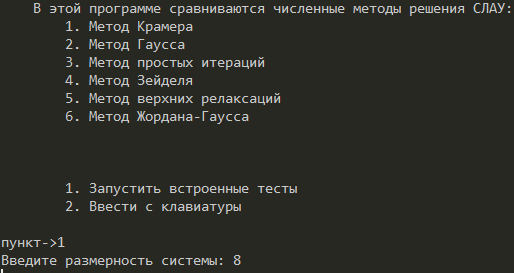




# Руководство к программе

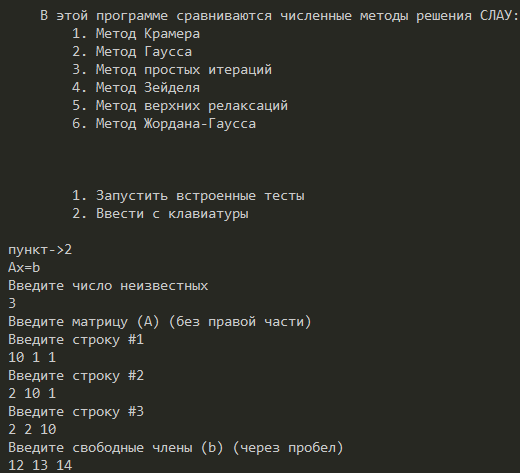
## Генерация случайной системы

Для запуска тестов со случайной системой необходимо выбрать первый пункт в меню и ввести размерность системы уравнений.



## Ввод данных с клавиатуры

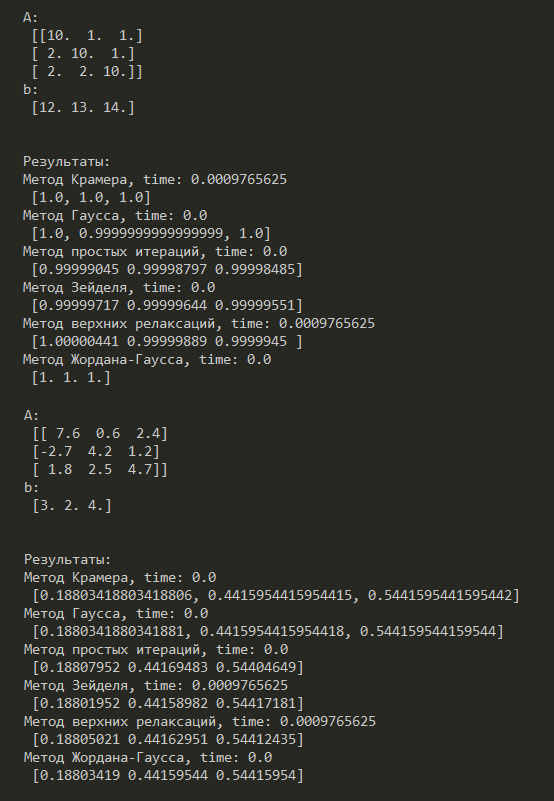
Для ввода данных с клавиатуры необходимо выбрать первый пункт в меню. Далее нужно ввести число неизвестных. Затем нужно последовательно ввести строки матрицы (без правой части). После этого вводится столбец свободных членов (тоже в виде строки)



# Тестирование

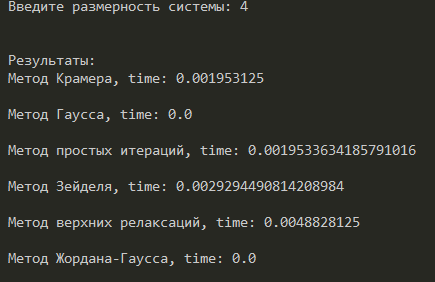
## Тест ввода с клавиатуры

Точность: 0.0001, лимит итераций 100

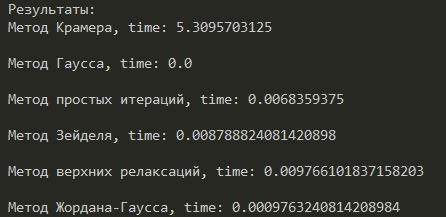


## Тест случайных данных

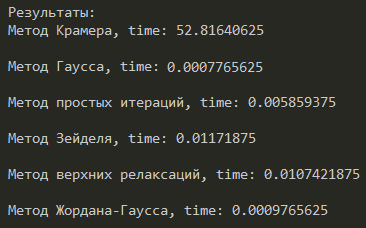
Размерность СЛАУ: 4, точность: 0.0001, лимит итераций 100



Размерность СЛАУ: 8, точность: 0.0001, лимит итераций 100



Размерность СЛАУ: 9, точность: 0.0001, лимит итераций 100



**Вывод**: нетрудно заметить, что метод Крамера работает медленнее всех, что объясняется огромной сложностью вычисления определителей. Остальные алгоритмы решения СЛАУ показали приблизительно одинаковые результаты, которые показывают, что их вполне можно применять на практике.

# Заключение

В ходе работы я узнал о том, что алгоритмы решения СЛАУ необходимы в большом количестве практических задач. В данной работе я рассмотрел 6 способов решения системы линейных уравнений. Выявил их достоинства и недостатки. После изучения алгоритмов в теории я разработал программу, которая сравнивает быстродействие данных алгоритмов. В результате тестирования я установил, что все методы, кроме метода Крамера могут применяться в решении настоящих прикладных задачах. Метод Крамера продемонстрировал крайне низкую производительность, что связано с огромной вычислительной сложностью получения определителей.

# Список **литературы**

1. Волков Е.А., «Численные методы», 2008
2. Жидков Е.Н., «Вычислительная математика», 2013
3. Лутц М., «Изучаем Python», том 1, 2019

# Приложение

## main.py

from methods import \*

import time

def hello():

    print("""

    В этой программе сравниваются численные методы решения СЛАУ:

        1. Метод Крамера

        2. Метод Гаусса

        3. Метод простых итераций

        4. Метод Зейделя

        5. Метод верхних релаксаций

        6. Метод Жордана-Гаусса

    """)

def userInput():

    print('Ax=b')

    n = int(input('Введите число неизвестных\n'))

    print('Введите матрицу (A) (без правой части)')

    tmpM = []

    for i in range(0, n):

        tmpM.append(list(map(float,input(f'Введите строку #{i+1}\n').split())))

    Matrix = np.array(tmpM)

    b = np.array(list(map(float,input(f'Введите свободные члены (b) (через пробел)\n').split())))

    return Matrix, b

if \_\_name\_\_ == '\_\_main\_\_':

    hello()

    print("""\n

        1. Запустить встроенные тесты

        2. Ввести с клавиатуры

    """)

    num = int(input('пункт->'))

    if num == 2:

        Matrix, b = userInput()

        print('A:\n', Matrix)

        print('b:\n', b)

        t0 = time.time()

        resCramer = Cramer(Matrix, b)

        timeCramer = time.time() - t0

        t0 = time.time()

        resGauss = Gauss(Matrix, b)

        timeGauss = time.time() - t0

        t0 = time.time()

        resSimpleIter = SimpleIterations(Matrix, b, 0.0001, 100)

        timeSimpleIter = time.time() - t0

        t0 = time.time()

        resSeidel = Seidel(Matrix, b, 0.0001, 100)

        timeSeidel = time.time() - t0

        t0 = time.time()

        resUpRelax = UpperRelaxations(Matrix, b, 0.5, 0.0001, 100)

        timeUpRelax = time.time() - t0

        t0 = time.time()

        resGaussJordan = GaussJordan(Matrix, b)

        timeGaussJordan = time.time() - t0

        print('\n\nРезультаты:')

        print(f'Метод Крамера, time: {timeCramer}\n', resCramer)

        print(f'Метод Гаусса, time: {timeGauss}\n', resGauss)

        print(f'Метод простых итераций, time: {timeSimpleIter}\n', resSimpleIter)

        print(f'Метод Зейделя, time: {timeSeidel}\n', resSeidel)

        print(f'Метод верхних релаксаций, time: {timeUpRelax}\n', resUpRelax)

        print(f'Метод Жордана-Гаусса, time: {timeGaussJordan}\n', resGaussJordan)

    else:

        N = int(input('Введите размерность системы: '))

        Matrix, b = specialSystem(N)

        t0 = time.time()

        resCramer = Cramer(Matrix, b)

        timeCramer = time.time() - t0

        t0 = time.time()

        resGauss = Gauss(Matrix, b)

        timeGauss = time.time() - t0

        t0 = time.time()

        resSimpleIter = SimpleIterations(Matrix, b, 0.0001, 100)

        timeSimpleIter = time.time() - t0

        t0 = time.time()

        resSeidel = Seidel(Matrix, b, 0.0001, 100)

        timeSeidel = time.time() - t0

        t0 = time.time()

        resUpRelax = UpperRelaxations(Matrix, b, 0.5, 0.0001, 100)

        timeUpRelax = time.time() - t0

        t0 = time.time()

        resGaussJordan = GaussJordan(Matrix, b)

        timeGaussJordan = time.time() - t0

        print('\n\nРезультаты:')

        print(f'Метод Крамера, time: {timeCramer}\n')

        print(f'Метод Гаусса, time: {timeGauss}\n')

        print(f'Метод простых итераций, time: {timeSimpleIter}\n')

        print(f'Метод Зейделя, time: {timeSeidel}\n')

        print(f'Метод верхних релаксаций, time: {timeUpRelax}\n')

        print(f'Метод Жордана-Гаусса, time: {timeGaussJordan}\n')

## methods.py

import numpy as np

import copy as cp

import math

def specialSystem(n):

    Matrix = np.zeros((n,n))

    b = np.zeros(n)

    k = 2

    d = 1

    for i in range(n):

        for j in range(n):

            if i == j:

                Matrix[i, j] = d

                d+=1

            else:

                Matrix[i, j] = k

                k+=1

        b[i] = np.sum(Matrix[i])

    return Matrix, b

# ======================== Cramer ======================== #

def determinant(Matrix):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    elif n == 1:

        return Matrix

    else:

        if n == 2:

            return Matrix[0][0]\*Matrix[1][1]-Matrix[0][1]\*Matrix[1][0]

        else:

            det = 0

            for i in range(n):

                submatrix = Matrix[1:, :]

                submatrix = np.delete(submatrix, i, 1)

                det += ((-1)\*\*i)\*Matrix[0][i]\*determinant(submatrix)

    return det

def Cramer(Matrix, b):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    delta = determinant(Matrix)

    if delta == 0:

        print("Det = 0 in Cramer!")

        return None

    else:

        res = list()

        for i in range(n):

            tmp\_matr = cp.copy(Matrix)

            tmp\_matr[:, i] = b

            delta\_i = determinant(tmp\_matr)

            x = delta\_i / delta

            res.append(x)

    return res

# ======================== ====== ======================== #

# ======================== Gauss ========================= #

def Gauss(Matrix, b):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    Matrix = Matrix.tolist()

    b = b.tolist()

    column = 0

    while(column < len(b)):

        current\_row = None

        for r in range(column, len(Matrix)):

            if current\_row is None or abs(Matrix[r][column]) > abs(Matrix[current\_row][column]):

                current\_row = r

        if current\_row is None:

            return None

        if current\_row != column:

            # меняем местами

            Matrix[current\_row], Matrix[column] = Matrix[column], Matrix[current\_row]

            b[current\_row], b[column] = b[column], b[current\_row]

        # делим строку на число

        b[column] /= Matrix[column][column]

        Matrix[column] = [a / Matrix[column][column] for a in Matrix[column]]

        for r in range(column+1, len(Matrix)):

            # остальные строки + строка\*коэфф

            b[r] += b[column] \* (-Matrix[r][column])

            Matrix[r] = [(a + k \* (-Matrix[r][column])) for a,k in zip(Matrix[r], Matrix[column])]

        column += 1

    res = [0 for b in b]

    #res = np.zeros(n)

    # ОБРАТНЫЙ ХОД

    for i in range(len(b)-1, -1, -1):

        res[i] = b[i] - sum(x\*a for x,a in zip(res[(i+1):], Matrix[i][(i+1):]))

    return res

# ======================== ====== ======================== #

# ================== Simple Iterations =================== #

def CheckDiagonalDominance(Matrix):

    n = Matrix.shape[0]

    for i in range(0, n):

        sm = 0

        for j in range(0, n):

            if i != j:

                sm+=abs(Matrix[i][j])

        if abs(Matrix[i][i]) < sm:

            return False

    return True

def CheckWithEpsilon(x\_prev, x\_curr, eps):

    # расстояние межу векторами < эпсилон

    n = x\_prev.shape[0]

    sm = 0.0

    for i in range(0, n):

        sm += (x\_prev[i] - x\_curr[i]) \*\* 2

    sm = math.sqrt(sm)

    if sm < eps:

        return True

    else:

        return False

def SimpleIterations(Matrix, b, epsilon, maxiter):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    # if (not CheckDiagonalDominance(Matrix)):

    #     print("The matrix does not have diagonal predominance. This method doesn't work")

    #     return None

    M = np.zeros((n,n))

    x\_curr = np.zeros(n)

    x\_next = np.zeros(n)

    iter\_counter = 0

    for i in range(0, n):

        for j in range(0, n):

            if i!=j:

                M[i][j] = -Matrix[i][j] / Matrix[i][i]

            else:

                M[i][j] = 0

        x\_next[i] = b[i] / Matrix[i][i]

    beta = x\_next

    x\_next = cp.copy(b)

    while True:

        x\_curr = x\_next

        x\_next = np.dot(M, x\_curr) + beta

        iter\_counter+=1

        if CheckWithEpsilon(x\_curr, x\_next, epsilon) or iter\_counter > maxiter-1:

            break

    return x\_next

# ================== ====== ========== =================== #

# ======================== Seidel ======================== #

def Seidel(Matrix, b, epsilon, maxiter):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    # if (not CheckDiagonalDominance(Matrix)):

    #     print("The matrix does not have diagonal predominance. This method doesn't work")

    #     return None

    M = np.zeros((n,n))

    x\_curr = np.zeros(n)

    x\_next = np.zeros(n)

    iter\_counter = 0

    for i in range(0, n):

        for j in range(0, n):

            if i!=j:

                M[i][j] = -Matrix[i][j] / Matrix[i][i]

            else:

                M[i][j] = 0

        x\_next[i] = b[i] / Matrix[i][i]

    beta = cp.copy(x\_next)

    x\_next = np.zeros(n)

    while True:

        x\_curr = cp.copy(x\_next)

        for i in range(0, n):

            sm = 0

            if i > 0:

                x\_curr[i] = x\_next[i]

            for j in range(0, n):

                sm += M[i][j] \* x\_curr[j]

            x\_next[i] = sm + beta[i]

        iter\_counter+=1

        if CheckWithEpsilon(x\_curr, x\_next, epsilon) or iter\_counter > maxiter-1:

            break

    return x\_next

# ======================== ====== ======================== #

# ======================== Relax ======================== #

def UpperRelaxations(Matrix, b, omega, epsilon, maxiter):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    # if (not CheckDiagonalDominance(Matrix)):

    #     print("The matrix does not have diagonal predominance. This method doesn't work")

    #     return None

    step = 0

    first = np.zeros(b.shape)

    res = first[:]

    delta = np.linalg.norm(np.matmul(Matrix, res) - b)

    while delta > epsilon and step <= maxiter:

        for i in range(Matrix.shape[0]):

            tmp = 0

            for j in range(Matrix.shape[1]):

                if j != i:

                    tmp += Matrix[i, j] \* res[j]

            res[i] = (1 - omega) \* res[i] + (omega / Matrix[i, i]) \* (b[i] - tmp)

        delta = np.linalg.norm(np.matmul(Matrix, res) - b)

        step += 1

    return res

# ======================== ===== ======================== #

# ======================== Gauss-Jordan ======================== #

def GaussJordan(Matrix, b):

    n, m = Matrix.shape

    if n != m:

        print('The matrix is not square!')

        return None

    m,n=Matrix.shape

    tmpMatr=np.zeros((m,n+1))

    tmpMatr[:m,:n]=Matrix

    tmpMatr[:,m]=b

    for i in range(0,m-1):

        for j in range(i+1,m):

            k=(-1)\*tmpMatr[j,i]/tmpMatr[i,i]

            tmp=tmpMatr[i,:]\*k

            tmpMatr[j,:]=tmpMatr[j,:]+tmp

    for i in range(m-1,0,-1):

        for j in range(i-1,-1,-1):

            k=(-1)\*tmpMatr[j,i]/tmpMatr[i,i]

            tmp=tmpMatr[i,:]\*k

            tmpMatr[j,:]=tmpMatr[j,:]+tmp

    for i in range(0,m):

        tmpMatr[i,:]=tmpMatr[i,:]/tmpMatr[i,i]

    return tmpMatr[:,n]

# ======================== ============ ======================== #

## test.py (юнит тесты)

import unittest

from methods import \*

# run: python -m unittest -v test.py

class TestDeterminant(unittest.TestCase):

    def test\_Det\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        self.assertAlmostEqual(determinant(A), None)

    def test\_Det\_one\_num(self):

        A = np.array([[5]])

        self.assertAlmostEqual(determinant(A), 5)

    def test\_Det\_2x2(self):

        A = np.array([[1, 5], [4, 2]])

        self.assertAlmostEqual(determinant(A), -18)

    def test\_Det\_3x3(self):

        A = np.array([[1, -2, 3], [0, 7, 4], [5, 3, -3]])

        self.assertAlmostEqual(determinant(A), -178)

    def test\_Det\_4x4(self):

        A = np.array([[0, 3, -1, 1], [1, 2, 0, 0], [0, 4, 3, 5], [2, -1, -4, -2]])

        self.assertAlmostEqual(determinant(A), -58)

class TestCramer(unittest.TestCase):

    def test\_Cramer\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(Cramer(A, b), None)

    def test\_Cramer\_zero\_det(self):

        A = np.array([[1, 1, -2], [2, -3, -1], [1, -4, 1]])

        b = np.array([2, 1, 3])

        self.assertEqual(Cramer(A, b), None)

    def test\_Cramer\_2x2(self):

        A = np.array([[1, -2], [3, -4]])

        b = np.array([1, 7])

        for v1, v2 in zip(Cramer(A, b), [5, 2]):

            self.assertAlmostEqual(v1, v2)

    def test\_Cramer\_3x3(self):

        A = np.array([[1, 2, -3], [3, 2, -4], [2, -1, 0]])

        b = np.array([1, 0, -1])

        for v1, v2 in zip(Cramer(A, b), [-2, -3, -3]):

            self.assertAlmostEqual(v1, v2)

    def test\_Cramer\_4x4(self):

        A = np.array([[1, 3, 5, 7], [3, 5, 7, 1], [5, 7, 1, 3], [7, 1, 3, 5]])

        b = np.array([12, 0, 4, 16])

        for v1, v2 in zip(Cramer(A, b), [1, -1, 0, 2]):

            self.assertAlmostEqual(v1, v2)

class TestGauss(unittest.TestCase):

    def test\_Gauss\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(Gauss(A, b), None)

    def test\_Gauss\_2x2(self):

        A = np.array([[1, -2], [3, -4]])

        b = np.array([1, 7])

        for v1, v2 in zip(Gauss(A, b), [5, 2]):

            self.assertAlmostEqual(v1, v2)

    def test\_Gauss\_3x3(self):

        A = np.array([[1, 2, -3], [3, 2, -4], [2, -1, 0]])

        b = np.array([1, 0, -1])

        for v1, v2 in zip(Gauss(A, b), [-2, -3, -3]):

            self.assertAlmostEqual(v1, v2)

    def test\_Gauss\_4x4(self):

        A = np.array([[1, 3, 5, 7], [3, 5, 7, 1], [5, 7, 1, 3], [7, 1, 3, 5]])

        b = np.array([12, 0, 4, 16])

        for v1, v2 in zip(Gauss(A, b), [1, -1, 0, 2]):

            self.assertAlmostEqual(v1, v2)

class TestDiagonalDominance(unittest.TestCase):

    def test\_Diagonal\_Dominance\_false(self):

        A = np.array([[2, 2, 10], [10, 1, 1], [2, 10, 1]])

        b = np.array([14, 12, 13])

        self.assertFalse(CheckDiagonalDominance(A))

    def test\_Diagonal\_Dominance\_true(self):

        A = np.array([[10, 1, 1], [2, 10, 1], [2, 2, 10]])

        b = np.array([12, 13, 14])

        self.assertTrue(CheckDiagonalDominance(A))

class TestCheckWithEpsilon(unittest.TestCase):

    def test\_CheckWithEpsilon\_true(self):

        vec1 = np.array([1, 2, 3])

        vec2 = np.array([1, 2.005, 3])

        epsilon = 0.01

        self.assertTrue(CheckWithEpsilon(vec1, vec2, epsilon))

    def test\_CheckWithEpsilon\_false(self):

        vec1 = np.array([1, 2, 3])

        vec2 = np.array([1, 2.005, 3])

        epsilon = 0.000001

        self.assertFalse(CheckWithEpsilon(vec1, vec2, epsilon))

class TestSimpleIterations(unittest.TestCase):

    def test\_SimpleIterations\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(SimpleIterations(A, b, 0.00000001, 100), None)

    def test\_SimpleIterations\_not\_diagonal\_dominance(self):

        A = np.array([[2, 2, 10], [10, 1, 1], [2, 10, 1]])

        b = np.array([14, 12, 13])

        self.assertEqual(SimpleIterations(A, b, 0.00000001, 100), None)

    def test\_SimpleIterations(self):

        A = np.array([[10, 1, 1], [2, 10, 1], [2, 2, 10]])

        b = np.array([12, 13, 14])

        for v1, v2 in zip(SimpleIterations(A, b, 0.00000001, 100), [1, 1, 1]):

            self.assertAlmostEqual(v1, v2)

    def test\_SimpleIterations(self):

        A = np.array([[7.6, 0.6, 2.4], [-2.7, 4.2, 1.2], [1.8, 2.5, 4.7]])

        b = np.array([3, 2, 4])

        for v1, v2 in zip(SimpleIterations(A, b, 0.001, 100), [0.188, 0.441, 0.544]):

            self.assertAlmostEqual(v1, v2, places=2)

class TestSeidel(unittest.TestCase):

    def test\_Seidel\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(Seidel(A, b, 0.00000001, 100), None)

    def test\_Seidel\_not\_diagonal\_dominance(self):

        A = np.array([[2, 2, 10], [10, 1, 1], [2, 10, 1]])

        b = np.array([14, 12, 13])

        self.assertEqual(Seidel(A, b, 0.00000001, 100), None)

    def test\_Seidel(self):

        A = np.array([[10, 1, 1], [2, 10, 1], [2, 2, 10]])

        b = np.array([12, 13, 14])

        for v1, v2 in zip(Seidel(A, b, 0.00000001, 100), [1, 1, 1]):

            self.assertAlmostEqual(v1, v2)

    def test\_Seidel(self):

        A = np.array([[7.6, 0.6, 2.4], [-2.7, 4.2, 1.2], [1.8, 2.5, 4.7]])

        b = np.array([3, 2, 4])

        for v1, v2 in zip(Seidel(A, b, 0.001, 100), [0.188, 0.441, 0.544]):

            self.assertAlmostEqual(v1, v2, places=2)

class TestRelax(unittest.TestCase):

    def test\_Relax\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(UpperRelaxations(A, b, 0.5, 0.0001, 100), None)

    def test\_Relax\_not\_diagonal\_dominance(self):

        A = np.array([[2, 2, 10], [10, 1, 1], [2, 10, 1]])

        b = np.array([14, 12, 13])

        self.assertEqual(UpperRelaxations(A, b, 0.5, 0.0001, 100), None)

    def test\_Relax(self):

        A = np.array([[10, 1, 1], [2, 10, 1], [2, 2, 10]])

        b = np.array([12, 13, 14])

        for v1, v2 in zip(UpperRelaxations(A, b, 1.5, 0.0001, 100), [1, 1, 1]):

            self.assertAlmostEqual(v1, v2)

    def test\_Relax(self):

        A = np.array([[7.6, 0.6, 2.4], [-2.7, 4.2, 1.2], [1.8, 2.5, 4.7]])

        b = np.array([3, 2, 4])

        for v1, v2 in zip(UpperRelaxations(A, b, 1.5, 0.0001, 100), [0.188, 0.441, 0.544]):

            self.assertAlmostEqual(v1, v2, places=2)

class TestGaussJordan(unittest.TestCase):

    def test\_GaussJordan\_invalid\_shape(self):

        A = np.array([[1, 2], [2, 3], [3, 4]])

        b = np.array([1, 2, 3])

        self.assertEqual(GaussJordan(A, b), None)

    def test\_GaussJordan\_2x2(self):

        A = np.array([[1, -2], [3, -4]])

        b = np.array([1, 7])

        for v1, v2 in zip(GaussJordan(A, b), [5, 2]):

            self.assertAlmostEqual(v1, v2)

    def test\_GaussJordan\_3x3(self):

        A = np.array([[1, 2, -3], [3, 2, -4], [2, -1, 0]])

        b = np.array([1, 0, -1])

        for v1, v2 in zip(GaussJordan(A, b), [-2, -3, -3]):

            self.assertAlmostEqual(v1, v2)

    def test\_GaussJordan\_4x4(self):

        A = np.array([[1, 3, 5, 7], [3, 5, 7, 1], [5, 7, 1, 3], [7, 1, 3, 5]])

        b = np.array([12, 0, 4, 16])

        for v1, v2 in zip(GaussJordan(A, b), [1, -1, 0, 2]):

            self.assertAlmostEqual(v1, v2)