**Алгоритм решения**

Перед применением алгоритма необходимо проверить условия сходимости метода простой итерации. В случае настоящих входных данных (см. раздел «Постановка задачи») выполняется достаточное условие сходимости (подробнее см. раздел «Результаты и их анализ»). Перед началом алгоритма исходное матричное уравнение домножается на АT для обеспечения симметричности основной матрицы. Тогда матрица исходная матрица А станет симметричной и положительно определённой. Это гарантирует корректные собственные значения матрицы и облегчит нахождение параметра t по формуле:

Будем строить последовательность приближений x(k). Выберем первое приближение x(0). По условию лабораторной работы необходимо взять x(0)= b.

Остальные приближения x(k) будем считать по формуле:

Или так:

где

В покоординатной форме:

**Листинг**

import numpy as np

def simple\_iteration(A, b, x0, t, tol=1e-5, max\_iter=1000):

    """

    Решение СЛАУ методом простой итерации

    Параметры:

        A (numpy.ndarray): Матрица коэффициентов.

        b (numpy.ndarray): Вектор свободных членов.

        x0 (numpy.ndarray): Начальное приближение.

        t (float): Параметр итераций.

        tol (float): Погрешность.

        max\_iter (int): Максимальное число итераций

    Возвращает:

        x (numpy.ndarray): Приближённое решение.

        n\_iter (int): Число выполненных итераций.

    """

    x = x0

    for n\_iter in range(max\_iter):

        # Вычисление нового приближения

        x\_new = x - t \* (A @ x - b)

        # Проверка критерия остановки

        if np.max(np.abs(x\_new - x)) < tol: #np.linalg.norm(x\_new - x, ord=np.inf) < tol:

            return x\_new, n\_iter + 1

        x = x\_new

    raise ValueError("Метод не сошёлся за заданное число итераций.")

def is\_positive\_definite\_eigenvalues(A):

    """Проверка положительной определённости матрицы через собственные значения."""

    eigenvalues = np.linalg.eigvals(A)  # Вычисление собственных значений

    return np.all(eigenvalues > 0)      # Проверка, что все значения положительны

def diagonal\_dominant\_coefficient(A):

    diag = []

    notdiag = []

    for i in range(len(A)):

        diag.append(np.abs(A[i,i]))

        notdiag.append(sum([np.abs(A[i,j]) for j in range(len(A[i])) if i != j]))

        if diag[i] < notdiag[i]:

            return False

    np.savetxt(fname='diag\_dominant\_res.txt', X=list(zip(diag, notdiag)))

    return True

# Пример матрицы A и вектора b

A = np.loadtxt(fname='A.txt')

A\_t = np.transpose(A)

A = A\_t @ A

b = np.loadtxt(fname='B.txt')

b = A\_t @ b

w,\_ = np.linalg.eig(A)

t = 2 / np.abs((np.max(np.real(w)) + np.min(np.real(w))))  # Параметр

x0 = b

another\_b = np.eye(A.shape[0], A.shape[1]) - t \* A

diagonal\_dominant\_coefficient(another\_b)

w, v = np.linalg.eig(another\_b)

with open('spec\_radius.txt', 'w') as f:

    f.write(str(np.max(np.abs(w))))

# Решение СЛАУ

x, iterations = simple\_iteration(A, b, x0, t)

error = (A @ x) - b

np.savetxt(fname='x\_res.txt', X=x)

np.savetxt(fname='error.txt', X=error)

with open('iterations.txt', 'w') as f:

    f.write(str(iterations))

print(f"Решение: {x}")

print(f"Число итераций: {iterations}")

**Результаты и их анализ**

На описанных данных (см. раздел «Постановка задачи») алгоритм выдаёт следующее решение

Х =

Со следующим вектором невязки R:

R = Ax – b

R =

||R|| =

По результатам Лабораторной работы №3 вектор невязки метода Якоби R2 на тех же входных данных:

R2 =

||R2|| =

Где P1 – вектор диагональных элементов матрицы B,

P2 – вектор сумм недиагональных элементов строк матрицы B

Нетрудно видеть, что покоординатно P1 > P2. Значит, матрица B обладает свойством диагонального преобладания. Тогда метод сходится.

В процессе отыскания решения исходной системы линейных уравнений алгоритмом было выполнена 41 итерация по сравнению с 11 итерациями метода Якоби

В методе простой итерации B = E – tA, где t — параметр релаксации. Однако выбор оптимального t может быть сложным.

В методе Якоби матрица итераций , где D — диагональная часть A, а L и U — нижняя и верхняя треугольные части.

Используя встроенные функции библиотеки numpy для вычисления собственных значений матрицы B получим следующее.

Спектральный радиус матрицы B в методе Якоби - 0.3211255411762186

Спектральный радиус матрицы B в методе простой итерации - 0.713924442837992

Таким образом скорость сходимости метода Якоби можно объяснить меньшим спектральным радиусом матрицы B..