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 A^T для обеспечения симметричности основной матрицы. Тогда матрица исходная матрица A станет симметричной и положительно определённой. Это гарантирует корректные собственные значения матрицы и облегчит нахождение параметра t по формуле:

$$t = \frac{2}{(\lambda_{max}(A)+\lambda_{min}(A))}$$

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:

$$x^{(k+1)} = x^{(k)} - t \cdot \left(Ax^{(k)} - b\right)$$

Или так:

$$x^{(k+1)} = Bx^{(k)} + c$$

где

$$B = E - tA, \qquad c = tb$$

В покоординатной форме:

$$x_i^{(k+1)} = x_i^{(k)} - t\sum_{j=1}^{k} a_{ij}x_j^{(k)} + tb_i.$$

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

$$X = \begin{bmatrix} -5.9996753463136887 06 \\ 2.9998407489843184 06 \\ 1.9999751343491267 71 \\ -4.0000898969835789 74 \\ 4.9998547749369279 81 \end{bmatrix}$$

Со следующим вектором невязки R:

R = Ax – b

$$R = \begin{bmatrix} -6.5714782488957013 13e-06 \\ -1.0861085453228724 87e-06 \\ -5.4348807783100028 25e-06 \\ 3.0705589200152871 85e-06 \\ -1.2588597764207065 67e-06 \end{bmatrix}$$

$||R|| = 1.2588597764207065 67e-06$

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

$$R2 = \begin{bmatrix} 7.6656863967627941 75e-07 \\ 3.9615900782052904 11e-08 \\ 6.9279319614423684 47e-07 \\ 7.8573480610799606 44e-08 \\ -2.6108074564490380 00e-06 \end{bmatrix}$$

$||R2|| = 2.6108074564490380 00e-06$

$$P1 = \begin{bmatrix} 9.5460000000000000 41e-01 \\ 1.4945999999999999 29e+00 \\ 1.2007000000000001 01e+00 \\ 1.1806000000000000 94e+00 \\ 1.4067000000000000 61e+00 \end{bmatrix} \quad P2 = \begin{bmatrix} 3.7670000000000003 48e-01 \\ 3.3160000000000000 59e-01 \\ 6.2810000000000010 27e-01 \\ 5.1739999999999997 10e-01 \\ 8.7919999999999998 15e-01 \end{bmatrix}$$

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 $\boldsymbol{B} = -\boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})$, 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..