Московский Авиационный Институт

(национальный исследовательский университет)

**Факультет Компьютерных наук и прикладной математики**

Кафедра Вычислительной математики и программирования

**Курсовой проект по дисциплине**

**«Численные методы»**

**4 курс, 7 семестр**

**Численное решение систем линейных алгебраических уравнений с несимметричными разреженными матрицами большой размерности.**

Студент: Орусский В. Р.

Группа: М8О-406Б-21

Преподаватель: Ревизников Д. Л.

Москва, 2025

[Постановка задачи 3](#_Toc192672977)

[Описание 4](#_Toc192672978)

[Метод бисопряжённых градиентов 4](#_Toc192672979)

[Код 5](#_Toc192672980)

[Результаты работы 7](#_Toc192672981)

[Выводы 8](#_Toc192672982)

# Постановка задачи

**Задача**: Реализовать решение систем линейных алгебраических уравнений с несимметричными разреженными матрицами большой размерности методом бисопряженных градиентов.

# Описание

### Метод бисопряжённых градиентов

Создадим случайную матрицу, поставим плотность (density) = 0.6, таким образом, наша матрица будет на 40% состоять из нулей. Остальные значения будут находиться в диапазоне от 0 до 1. Можно выбрать размер матрицы от 4 до n.

Рассмотрим СЛАУ с вещественными коэффициентами(Ax=b). В основу алгоритма ложится идея проекционного метода и использование свойства A-бисопряженности системы векторов, а именно: векторы p – невязки бисопряжённые, если скалярные произведения (Api,pj)=0, (Api,pj)=0. Фактически, данное условие эквивалентно биортогональности относительно энергетического скалярного произведения.

Общая концепция проекционных методов предписывает нам выбрать два подпространства. В нашем случае подпространства мы выберем два подпространства, задаваемые матрицей системы А, вектором невязки на нулевой итерации.

Метод основан на построении биортогонального базиса p пространства K\_k(A,r0) K\_k(A,r0) и вычислении поправки такой, что новое приближение на очередной итерации было бы ортогонально второму подпространству Крылова. Базисные вектора строятся до тех пор, пока не будет достигнут критерий остановки итерационного процесса, а каждое последующее приближение формируется, как сумма приближения на предыдущей итерации и найденной поправки.

Критерием останова итерационного процесса является достижение невязкой значения, которое меньше некоторого наперед заданного эпсилон.

# Код

import numpy as np

from random import randint

from scipy.sparse import rand

import numpy as np

from numpy.linalg import norm

from scipy.sparse import diags, csc\_matrix

from scipy.sparse.linalg import bicgstab, spilu

import time

Генерация матрицы:

shape = int(input())

if shape < 3:

        exit()

matrix = rand(shape, shape, density=0.6, random\_state=randint(112, 154))

for i in matrix.toarray().round(3):

    for j in i:

        print(f'{j} ', end=" ")

    print('\n')

print('\n')

b = np.random.randint(5, 53, shape)

for i in b:

    print(f'{i} ', end=" ")

print('\n')

print(type(matrix))

Реализация алгоритма:

class BiCGMethod:

    def \_\_init\_\_(self, matrix, b, x0=None, eps=1e-5):

        self.matrix = matrix

        self.b = b

        self.eps = eps

        self.shape = matrix.shape[0]

        self.x0 = np.array([0] \* self.shape) if x0 is None else x0

        self.k = 0

    def solve(self):

        r0 = self.b - self.matrix @ self.x0 *# невязка*

        x0 = self.x0 *# начальное приближение*

        r2 = r0 *# выбирается вектор*

        rho0 = 1

        alpha0 = 1

        omega0 = 1

        v0 = np.array([0] \* self.shape)

        p0 = np.array([0] \* self.shape)

        while True:

            rho = r2 @ r0

            beta = (rho \* alpha0) / (rho0 \* omega0)

            p = r0 + beta \* (p0 - omega0 \* v0)

            v = self.matrix @ p

            alpha = rho / (r2 @ v)

            s = r0 - alpha \* v

            t = self.matrix @ s

            omega = (t @ s) / (t @ t)

            x = x0 + omega \* s + alpha \* p

            r = s - omega \* t

            self.k += 1

            if norm(r) < self.eps: *# норма заданной невязки*

                break

            r0 = r

            rho0 = rho

            alpha0 = alpha

            omega0 = omega

            v0 = v

            p0 = p

            x0 = x

        return x

    def print\_solution(self):

        start\_timeBiCGM = time.time()

        x = self.solve()

        print("BiCGMethod time: --- %s seconds ---\n" % (time.time() - start\_timeBiCGM))

        start\_timeNumPy = time.time()

        x2 = np.linalg.solve(self.matrix.toarray(), self.b)

        print("NumPy time: --- %s seconds ---\n" % (time.time() - start\_timeNumPy))

*# with open(self.output, 'w') as f:*

        print('My solve:\n')

        print(f'{x.round(5)}\n')

        print(f'EPS = {self.eps}\n')

        print(f'Shape = {self.shape}\n')

        print(f'Count of iterations = {self.k}\n')

*# print(f'Mean = {np.mean(x)}\n') # среднее*

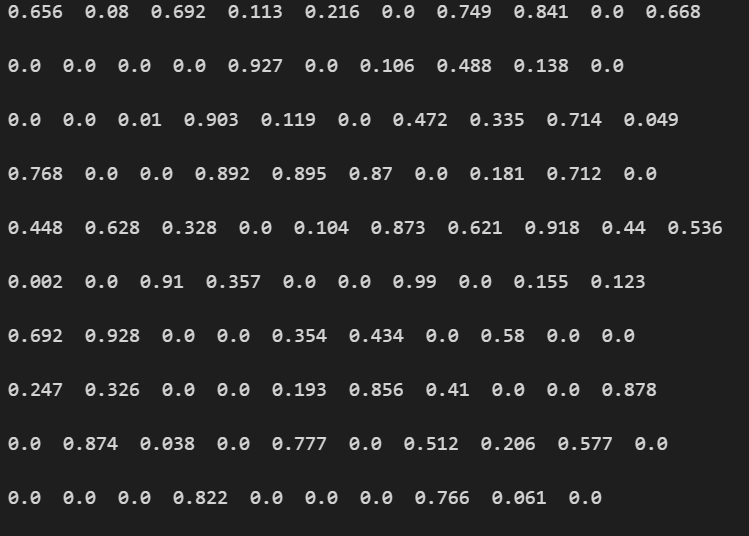
        print('\nNumPy solve:\n')

        print(f'{x2.round(5)}\n')

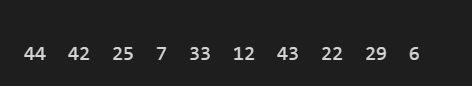
*# print(f'Mean = {np.mean(x2)}\n')*

# Результаты работы

Для матрицы с плотностью 0.6 и следующими входными данными:



Столбец свободных коэффицентов:



Имеем решение следующего вида:

BiCGMethod time: --- 0.008104562759399414 seconds ---

NumPy time: --- 0.00514531135559082 seconds ---

My solve:

[75.04408 -46.1016 -187.69614 -19.61134 15.11262 16.795

219.22775 36.50329 -95.36002 -101.12381]

EPS = 1e-05

Shape = 10

Count of iterations = 13

NumPy solve:

[75.04408 -46.1016 -187.69614 -19.61134 15.11262 16.795

219.22775 36.50329 -95.36002 -101.12381]

# Выводы

В ходе данной лабораторной работы был изучен алгоритм бисопряженных градиентов. Этот алгоритм хорошо зарекомендовал себя для решения систем линейных алгебраический уравнений, поскольку обладает некоторыми важными свойствами: он численно устойчив; не меняется вид матрицы в процессе решения; эффективен для решения систем большой размерности с разреженной матрицей, поскольку в методе самая трудоемкая операция - умножение матрицы на вектор; применим для решения систем с несимметричными матрицами.