Московский авиационный институт

(Национальный исследовательский университет)

**Факультет «Информационных технологий и прикладной математики»**

**Курсовая работа**

по курсу

**«Численные методы»**

VI семестр

Тема: «Метод бисопряжённых градиентов»

Работу выполнил:

Шевчук П.В.

Группа: 8O – 304Б

Оценка:

Дата:

Москва, 2019

Постановка задачи

Реализовать решение систем линейных алгебраических уравнений с

несимметричными разреженными матрицами большой размерности.

Метод бисопряженных градиентов.

Описание метода

Для начала создадим случайную матрицу, поставим плотность (density) = 0.4, 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.

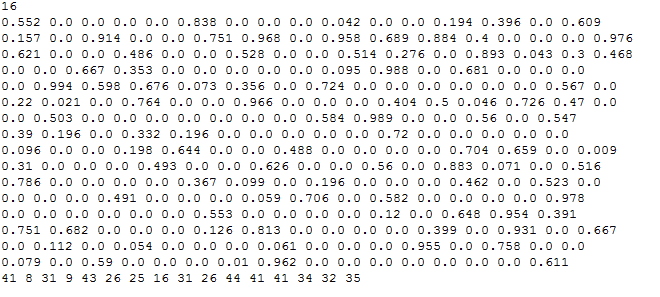
Рассмотрим СЛАУ с вещественными коэффициентами(Ax=b). В основу алгоритма ложиться идея проекционного метода и использование свойства A-бисопряженности системы векторов, а именно: векторы  p – невязки бисопряжённые, если скалярные произведения (Api,pj)=0(Api,pj)=0. Фактически, данное условие эквивалентно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

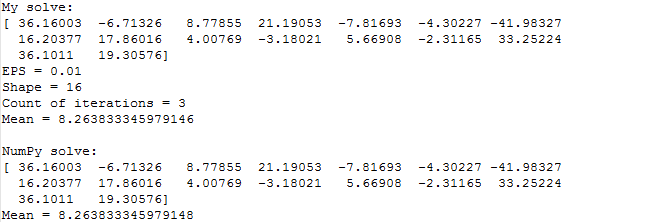
Метод основан на построении биортогонального базиса p пространства  K\_k(A,r0) K\_k(A,r0) и вычислении поправки такой, что новое приближение на очередной итерации было бы ортогонально второму подпространству Крылова. Базисные вектора строятся до тех пор, пока не будет достигнут критерий остановки итерационного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

Тесты

Сгенерируем СЛАУ, матрица коэффициентов, которая 16 на 16 и 16 чисел правой части:

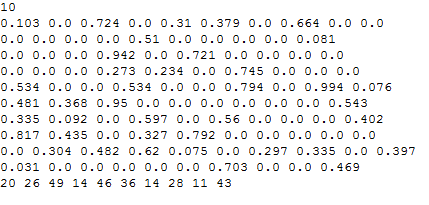


Найдем необходимые коэффициенты x\_n, посмотрим их среднее арифметическое, а также сравним с решение от функций встроенной библиотеки Numpy:

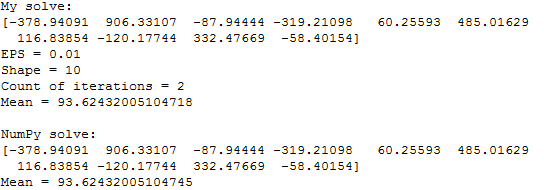


Как видим, наш ответ полностью совпал.

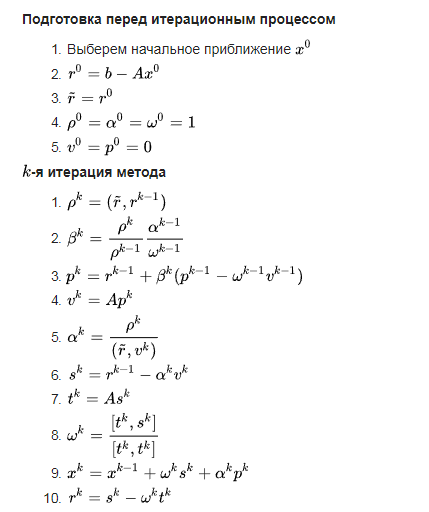
Построим решение для n=10:



Ответ:



Алгоритм



Критерий остановки итерационного процесса**:** норма заданной невязки меньше ипсилон.

Код

1. **Генерация матрицы**

import argparse

import numpy as np

from random import randint

from scipy.sparse import rand

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--shape', required=True, type=int,

help='Shape of matrix')

parser.add\_argument('--output', required=True, help='Output file')

args = parser.parse\_args()

if args.shape < 3:

exit()

shape = args.shape

matrix = rand(shape, shape, density=0.4, random\_state=randint(112, 154))

with open(args.output, 'w') as f:

f.write(f'{shape}\n')

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

for j in i:

f.write(f'{j} ')

f.write('\n')

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

for i in d:

f.write(f'{i} ')

f.write('\n')

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. **Реализация алгоритма**

import argparse

import numpy as np

from numpy.linalg import norm

from scipy.sparse import diags, csc\_matrix

from scipy.sparse.linalg import bicgstab, spilu

# python .\bicgstab.py --input rand\_mtrx100 --output out1 --eps 0.01

def get\_matrix(filename, is\_diag):

with open(filename) as f:

shape = int(f.readline())

matrix = [[float(num) for num in line.split()]

for \_, line in zip(range(shape), f)]

if is\_diag:

matrix[0].insert(0, 0)

matrix[-1].append(0)

a, b, c = zip(\*matrix)

matrix = diags([a[1:], b, c[:-1]], [-1, 0, 1])

matrix = csc\_matrix(matrix)

else:

matrix = csc\_matrix(matrix)

b = np.array([float(num) for num in f.readline().split()])

return matrix, b

class BiCGStab:

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

self.output = 'res\_default' if output\_file is None else output\_file

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):

x = self.solve()

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

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

f.write('My solve:\n')

f.write(f'{x.round(5)}\n')

f.write(f'EPS = {self.eps}\n')

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

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

f.write(f'Mean = {np.mean(x)}\n')

f.write('\nNumPy solve:\n')

f.write(f'{x2.round(5)}\n')

f.write(f'Mean = {np.mean(x2)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input file')

parser.add\_argument('--output', required=True, help='Output file')

parser.add\_argument('--eps', type=float, help='Epsilon', default=1e-2)

parser.add\_argument('--diag', help='If matrix is diag', action='store\_true')

args = parser.parse\_args()

matrix, b = get\_matrix(args.input, args.diag)

solver = BiCGStab(matrix, b, output\_file=args.output, eps=args.eps)

solver.print\_solution()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Вывод

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. Вычислительная мощность:  N/4