



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего
образования

«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ЛАБОРАТОРНАЯ РАБОТА №10

По основной образовательной программе подготовки бакалавров
направлению 01.03.02 Прикладная математика и информатика
профиль «Системное программирование»

Студент группы Б9122-01.03.02сп(4)
Кириенко Денис Олегович

(подпись)

« ____ » _____ 2024 г.

Преподаватель старший преподаватель
(должность, ученое звание)

Журавлев Павел Викторович

(подпись)

(ФИО)

« ____ » _____ 2024 г.

г. Владивосток
2024

Постановка задачи

Реализовать и протестировать метод “вращения с преградами” для решения полной проблемы собственных значений.

Теоретическое описание метода

Метод предназначен для решения полной проблемы собственных значений невырожденной симметрической матрицы A . Решается она с помощью сходящихся итерационных процессов.

Входные данные:

1. Невырожденная симметрическая матрица A
2. Положительное число p , определяющее точность решения

Результатом работы метода является диагональная матрица D , на диагонали которой расположены все собственные значения данной матрицы.

Практическая часть

На каждой итерации строится матрица вращения T_{ij} , посредством которой происходит переход в следующую итерацию: $A^{m+1} = T_{ij}^T A^m T_{ij}$.

В методе осуществляется два цикла: общий - с итератором k , и вложенный - с итератором m .

Для построения матрицы вращения T_{ij} необходимо выполнение нескольких шагов:

1. Вычислить преграду σ_k

Вычисление преграды имеет несколько реализаций. В данной работе преграда на шаге k вычислялась так: $\sigma_k = \sqrt{(\max |a_{ij}^{(m)}|) * 10^{-k}}$, где k - шаг, m - номер итерации матрицы A , а $a_{ij}^{(m)}$ - всякое значение матрицы $A^{(m)}$.

2. Найти значения i и j

Парой индексов (i, j) обозначается наибольший по модулю недиагональный элемент матрицы $A^{(m)}$ на шаге m , который соответствует условию $|a_{ij}^{(m)}| \geq \sigma_k$.

3. Найти значения c и s

Значения c и s должны быть такими, что $s^2 + c^2 = 1$. Вычисляются они по заданным формулам:

$$c = \sqrt{\frac{1}{2} \left(1 + \frac{|a_{ii}^{(m)} - a_{jj}^{(m)}|}{d} \right)},$$

$$s = \operatorname{sgn}[a_{ij}^{(m)}(a_{ii}^{(m)} - a_{jj}^{(m)})] \sqrt{\frac{1}{2} \left(1 - \frac{|a_{ii}^{(m)} - a_{jj}^{(m)}|}{d} \right)}, \quad d = \sqrt{(a_{ii}^{(m)} - a_{jj}^{(m)})^2 + 4a_{ij}^{(m)2}}, \quad \text{где } (i, j) -$$

пара индексов, найденная на шаге 2.

После выполнения указанных шагов строится матрица T_{ij} . Она основывается на единичной матрице, однако отличие матрицы вращения от единичной состоит в том, что:

1. Значение на позиции (i, i) равняется с
2. Значение на позиции (i, j) равняется $-s$
3. Значение на позиции (j, i) равняется s
4. Значение на позиции (j, j) равняется с

После этого находится $A^{m+1} = T_{ij}^T A^m T_{ij}$ и так происходит переход на следующую итерацию.

Итерации заканчиваются на шаге m , если не удастся найти наибольший по модулю недиагональный элемент матрицы $A^{(m)}$, модуль которого больше либо равен текущей преграде, то есть элемент с шага 2 построения матрицы вращения.

По окончании всех итераций должна получиться диагональная матрица $D = A^{(m)}$, состоящая из собственных значений исходной матрицы A .

Работа выполнялась посредством языка программирования python и математической библиотеки numpy.

Заключение

Результаты тестирования кода метода дают неплохую среднюю точность, равную различию вычисленного реализованным в данной работе алгоритмом собственного значения в 16 знаке после запятой по сравнению со значением, вычисленным встроенными средствами библиотеки numpy.

Также хочется отметить количество итераций, обозначенное на фотографии ниже именем steps: в среднем на матрице 6x6 алгоритм выполняется за 50-70 итераций, а на матрице 20x20 в среднем за 850 итераций при сохранении средней точности в $e-16$.

matrix					
0.642129	-0.292183	-0.559402	-0.203087	0.0539575	0.047709
-0.292183	0.750159	0.389467	0.0527544	0.679333	-0.0718878
-0.559402	0.389467	0.842848	0.111532	0.2934	0.0572483
-0.203087	0.0527544	0.111532	0.713165	-0.0831191	-0.324435
0.0539575	0.679333	0.2934	-0.0831191	1	-0.0935304
0.047709	-0.0718878	0.0572483	-0.324435	-0.0935304	0.441477

steps: 59

delta:

0: 3.174543961037557e-16
1: 8.326672684688674e-17
2: 4.440892098500626e-16
3: 5.551115123125783e-16
4: 8.881784197001252e-16
5: 2.220446049250313e-16

Уточнения:

- Тестирование проводилось с параметром точности p равным 8
- Под средней точностью подразумевается среднее значение всех порядков различия вычисленных значений со значениями, данными библиотекой numpy

В заключение можно сказать, что метод в реализованном в данной работе алгоритме показал достаточную точность и эффективность.

Приложение

```
import sys

import numpy as np

from labs.funcs import *

sys.stdout = open("./labs/output.txt", "w")

def rotation_with_barriers(
    A: np.ndarray,
    p: int = 4,
) -> np.ndarray:
    D = A.copy()
    n = D.shape[0]

    if np.linalg.det(D) == 0:
        raise ValueError("matrix is singular")

    counter = 0

    for K in range(1, p + 1):
        sigma = np.sqrt(np.max(np.abs(np.diag(np.diag(M))))) * 10 ** (-K)
        # sigma = 10 ** (-K)

        while True:
            if counter > 1e5:
                raise ValueError("inf cycle")

            mx_val = -np.inf
            idx = ()

            for i in range(n):
                for j in range(n):
                    if D[i, j] > mx_val and np.abs(D[i, j]) >= sigma and i != j:
                        mx_val = D[i, j]
                        idx = (i, j)

            if mx_val == -np.inf:
```

```

        break

    i, j = idx[0], idx[1]

    d = np.sqrt((D[i, i] - D[j, j]) ** 2 + 4 * D[i, j] ** 2)
    s = np.sign(D[i, j] * (D[i, i] - D[j, j])) * np.sqrt(
        1 / 2 * (1 - np.linalg.norm(D[i, i] - D[j, j]) / d)
    )
    c = np.sqrt(1 / 2 * (1 + np.linalg.norm(D[i, i] - D[j, j]) / d))

    # print(f"K: {K} \nsigma: {sigma} \ni,j: {i+1,j+1} \nmx_val: {mx_val}")
    # print(f"c:\t{c}\ts:\t{s}")
    # print_matrix(D)

    T = np.eye(n)
    T[i, i] = T[j, j] = c
    T[i, j] = -s
    T[j, i] = s

    D = T.T @ D @ T

    counter += 1

print(f"steps: {counter}")

return np.diag(D)

size = (20, 20)
M = generate_symmetric_matrix(*size).astype(np.double)
M /= np.max(M)

# print_matrix(M, "matrix")

ans = rotation_with_barriers(M, p=8) # max(p)=8
np_ans = np.linalg.eigvals(M)

ans = np.array(sorted(ans))
np_ans = np.array(sorted(np_ans))

print(
    f"""
delta:
{'.'.join(f"{i[0]}: {abs(i[1] - ans[i[0]])}\n" for i in enumerate(np_ans))}
"""
)

```