МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №7

По курсу «Вычислительные методы алгебры»

Метод Крылова. Вычисление собственных векторов.

Вариант №5

Работу выполнил:
студент 3 курса 7 группы
Шатерник Артём
Преподаватель:
Будник А. М.

Минск 2023

1. Постановка задачи.

Дана матрица следующего вида

A

0.5757	-0.0758	0.0152	0.0303	0.1061
0.0788	0.9014	0.0000	-0.0606	0.0606
0.0455	0.0000	0.7242	-0.2121	0.1212
-0.0909	0.1909	0.0000	0.7121	-0.0303
0.3788	0.0000	0.1364	0.0152	0.8484

Требуется методом Крылова построить систему собственных векторов матрицы $A^T A$. Вычислить невязки $\varphi_i(\lambda_i) = P_n(\lambda_i)$ и $r_i = A^T A x_i - \lambda_i x_i$, оценить их значения.

2. Алгоритм решения.

Для построения системы собственных векторов с помощью метода Крылова вначале потребуется построить собственный многочлен $P(\lambda)$.
Вначале потребуется построить систему векторов

$$c^0, c^1 = Ac^0, \dots, c^n = A^n c^0.$$

Всего $n + 1$ вектор.
В качестве начального вектора c^0 был взят $(1, 0,0,0,0)$.
По координатам этих векторов строится система

$$\begin{cases} q_1 c_1^{n-1} + \dots + q_n c_1^0 = c_1^n, \\ \dots\dots\dots\dots\dots\dots\dots \\ q_1 c_n^{n-1} + \dots + q_n c_n^0 = c_n^n. \end{cases}$$

Её требуется решить некоторым точным методом. Применялся метод Гаусса. В результате получаем систему

$$\begin{cases} q_1 + b_{12} q_2 + \dots + b_{1n} q_n = g_1, \\ q_2 + \dots + b_{2n} q_n = g_2, \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ q_n = g_n \end{cases}$$

Обратным ходом метода Гаусса вычисляем q_i. Они и являются коэффициентами собственного многочлена

$$P_n(\lambda) = \lambda^n - q_1 \lambda^{n-1} - \dots - q_n.$$

Вычисление собственных векторов.

Для вычисления собственных векторов нужны собственные значения, подсчитанные из собственного многочлена или другим методом. Были взяты собственные значения, вычисленные методом вращений с точностью 10^{-16}.

Собственные векторы ищем в виде

$$x_i = \beta_{i1}c^{n-1} + \beta_{i2}c^{n-2} + \ldots + \beta_{in}c^0.$$

Где β_{ij} можно найти из системы

$$\begin{cases} \beta_{i1} = 1, \\ \beta_{i2} = \lambda_i - q_1, \\ \beta_{i3} = \lambda_i^2 - q_1\lambda_i - q_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \beta_{im} = \lambda_i^{m-1} - q_1\lambda_i^{m-2} - \ldots - q_{m-1}. \end{cases}$$

При $m = n$.

3. Листинг программы.

```
import numpy as np
import math

def gaussFunc(matrix):
    copy_matrix = np.copy(matrix)
    for nrow, row in enumerate(copy_matrix):
        divider = row[nrow]
        row /= divider
        for lower_row in copy_matrix[nrow+1:]:
            factor = lower_row[nrow]
            lower_row -= factor*row
    return copy_matrix

def gauss_reverse(matrix):
    n_row=matrix.shape[0]
    x=[None]*n_row
    for i in range(n_row-1, -1,-1):
        x[i]=matrix[i,-1]-np.dot(matrix[i, i+1:n_row], x[i+1:])
    return(np.array(x))

size = 5
a_matrix = []
with open('input.txt') as file:
    i = 0
    for line in file:
        a_matrix.append([float(x) for x in line.split(' ')])
        i += 1
a_matrix = np.array(a_matrix)

# Симметричный вид
a_matrix = np.matmul(a_matrix.T, a_matrix)
# Начальный вектор
```

```
c = [1.]
[c.append(0) for i in range(size - 1)]
c_list = [c]
# Строим остальные c (всего n + 1)
for i in range(size):
    c_list.append(np.matmul(a_matrix, c_list[i]).tolist())
b_matrix = np.zeros((size + 1, size))
b_matrix[size] = c_list[size]
for i in range(size):
    b_matrix[size - i - 1] = c_list[i]
b_matrix = b_matrix.T
q_list = gauss_reverse(gaussFunc(b_matrix))
print("Коэффициенты собственного многочлена:")
print(q_list)
print()
# Вычисление собственных векторов
# Собственные значения, вычисленные методом вращений с точностью 10^(-
16)
lambda_list = [0.191009010539545, 0.879661472553097,
0.383558316492653, 0.597703453393687, 1.144756197021018]
beta_matrix = [[1] for i in range(size)]
for i in range(size):
    for j in range(size - 1):
        res = pow(lambda_list[i], j + 1)
        res -= q_list[j]
        for k in range(j):
            res -= q_list[k] * pow(lambda_list[i], j - k)
        beta_matrix[i].append(res)
print('Матрица beta: ')
print(np.array(beta_matrix))
x_vectors = []
for i in range(size):
    vector = np.array([0 for n in range(size)])
    for j in range(size):
        vector = vector + beta_matrix[i][j] *  np.array(c_list[size -
j - 1])
    x_vectors.append(vector)
print("\nСобственные значения и соответствующие им собственные
векторы")
for i in range(size):
    print(np.round(lambda_list[i], 5) ,end=': ( ')
    for j in range(size):
        print(np.round(x_vectors[i][j], 5) ,end=' ')
    print(')')

# Невязки
for i in range(size):
    res = np.matmul(a_matrix, x_vectors[i]) - lambda_list[i] *
x_vectors[i]
    print('(', end='')
    for j in range(size):
        print(format(res[j], '.4e'), end=' ')
    print(')')
```

```
    print('Норма невязки: ', end='')
    print(format(np.linalg.norm(res, 1), '.4e'))
p = q_list
for i in range(size):
    sum = pow(lambda_list[i], size)
    for j in reversed(range(size)):
        sum -= pow(lambda_list[i], j) * p[size - j - 1]
    print(sum)
```

4. Результат и его анализ.

Коэффициенты собственного многочлена:

[3.19668845 -3.79684757 2.06780624 -0.50824834 0.04409604]

Невязки $\varphi_i(\lambda_i) = P_n(\lambda_i)$ для собственного многочлена

-1.1102230246251565e-16
-8.049116928532385e-15
-8.326672684688674e-17
5.551115123125783e-17
-2.220446049250313e-16

Собственные значения и соответствующие им собственные векторы

0.19101: (0.02929 0.00035 0.00827 0.00559 -0.02343)
0.87966: (-0.0 0.00013 -1e-05 4e-05 0.0)
0.38356: (-0.00161 -0.00044 0.00371 0.00292 -1e-05)
0.5977: (0.00077 -0.00091 -0.00161 0.00234 0.00094)
1.14476: (0.02839 0.0039 0.0207 -0.0099 0.04051)

Невязки $r_i = A^T A x_i - \lambda_i x_i$

(-1.8995e-16 2.5153e-17 -1.1905e-16 7.3943e-17 -2.7929e-16)
Норма невязки: 6.8738e-16
(7.5406e-15 3.7324e-17 -2.1386e-17 -1.2685e-17 1.3366e-16)
Норма невязки: 7.7457e-15
(-9.1507e-17 3.6402e-17 -6.5703e-17 3.7513e-17 -8.0680e-17)
Норма невязки: 3.1180e-16
(-2.0177e-16 2.2551e-17 -1.0983e-16 3.0358e-17 -8.2074e-17)
Норма невязки: 4.4658e-16
(-3.4001e-16 4.0766e-17 1.3878e-17 -2.4286e-17 1.7347e-16)
Норма невязки: 5.9241e-16

Использовалась первая норма.

Экономичность:

Метод имеет сложность $O(n^3)$. Это связанно с тем, что метод Гаусса, который использовался при решении системы имеет сложность $O(n^3)$.

Как и в методе Данилевского возможно возрастание числа операций, если алгоритма метода Гаусса возможен только на $m < n$ шагов, но данный случай в работе не рассматривался.

Точность:

Метод Крылова является точным методом и невязки для собственного многочлена получились с точностью порядка 10^{-16}, что связанно с числом хранимых знаков у типа данных float в Python (17 знаков).
Точность вычисленных собственных векторов получилась порядка 10^{-16} по тем же причинам.