**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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 Вычислить невязки , оценить их значения.

1. **Алгоритм решения.**

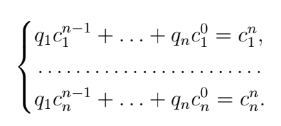
Для построения системы собственных векторов с помощью метода Крылова вначале потребуется построить собственный многочлен

Вначале потребуется построить систему векторов

Всего вектор.

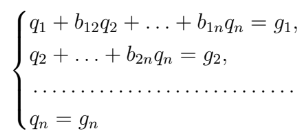
В качестве начального вектора был взят

По координатам этих векторов строится система

**

Её требуется решить некоторым точным методом. Применялся метод Гаусса.

В результате получаем систему



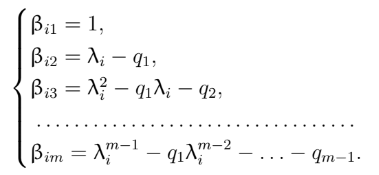
Обратным ходом метода Гаусса вычисляем Они и являются коэффициентами собственного многочлена

**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 .

Собственные векторы ищем в виде

Гдеможно найти из системы

**

При

1. **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)**

1. **Результат и его анализ.**

Коэффициенты собственного многочлена:

[3.19668845 -3.79684757 2.06780624 -0.50824834 0.04409604]

Невязки 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

(-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 Это связанно с тем, что метод Гаусса, который использовался при решении системы имеет сложность

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

Точность:

Метод Крылова является точным методом и невязки для собственного многочлена получились с точностью порядка , что связанно с числом хранимых знаков у типа данных float в Python (17 знаков).

Точность вычисленных собственных векторов получилась порядка

по тем же причинам.