**Министерство образования Республики Беларусь**

**Белорусский государственный университет**

Факультет прикладной математики и информатики

Лабораторная работа №5

По курсу «Вычислительные методы алгебры»

**Метод Данилевского**

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 |  |

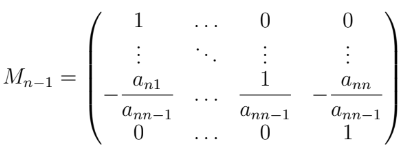
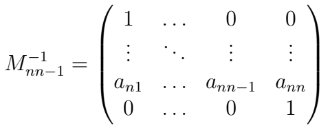
Требуется методом Данилевского построить собственный многочлен матрицы Вычислить невязки

1. **Алгоритм решения.**

Матрица имеет следующий вид

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.49146 | 0.01004 | 0.09337 | - 0.05595 | 0.3955 |
| 0.01004 | 0.85471 | - 0.00115 | 0.07902 | 0.0408 |
| 0.09337 | - 0.00115 | 0.5433 | - 0.15107 | 0.20511 |
| - 0.05595 | 0.07902 | - 0.15107 | 0.55689 | - 0.03484 |
| 0.3955 | 0.0408 | 0.20511 | - 0.03484 | 0.75032 |

Метод Данилевского основан на приведении матрицы к виду Фробениуса, при таком виде сразу видны коэффициенты собственного многочлена этой матрицы. Для приведения к такому виду применяются преобразования подобия . Матрицы строятся следующим образом



Обозначим , тогда

и

В таком случае вычисляется следующим образом

Всего потребуется итерация.

Элементы первой строки и будут коэффициентами собственного многочлена матрицы.

1. **Листинг программы.**

**import numpy as np  
import math  
  
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)  
for i in range(size):  
 for j in range(size):  
 print(np.round(a\_matrix[i][j], 5), end='')  
 print(" ", end='')  
 print()  
print(np.trace(a\_matrix))  
# Метод Данилевского  
for i in reversed(range(size - 1)):  
 m\_vec = []  
 for j in range(size):  
 if j != i:  
 m\_vec.append(-1 \* a\_matrix[i + 1][j] / a\_matrix[i + 1][i])  
 else:  
 m\_vec.append(1 / a\_matrix[i + 1][i])  
 m\_inv\_vec = np.identity(size)  
 m\_inv\_vec[i] = a\_matrix[i + 1]  
# Умножаем на M справа  
 b\_matrix = np.zeros((size, size))  
 for n in range(size):  
 for k in range(size):  
 if k != i:  
 b\_matrix[n][k] = a\_matrix[n][k] + a\_matrix[n][i] \* m\_vec[k]  
 else:  
 b\_matrix[n][k] = a\_matrix[n][i] \* m\_vec[i]  
 a\_matrix = np.copy(b\_matrix)  
# Умножаем на M^-1 слева  
 for j in range(size):  
 sum = 0  
 for k in range(size):  
 sum += a\_matrix[k][j] \* m\_inv\_vec[i][k]  
 b\_matrix[i][j] = sum  
 a\_matrix = np.copy(b\_matrix)  
print(a\_matrix)  
print(a\_matrix[0])**

**# Невязки**

**p = [3.1966884499999972, -3.7968475734971836, 2.0678062361750578, -0.5082483413019262, 0.044096040836178144]  
for i in range(size):  
 sum = pow(a\_matrix[i][i], size)  
 for j in reversed(range(size)):  
 sum -= pow(a\_matrix[i][i], j) \* p[size - j - 1]  
 print(sum)**

1. **Результат и его анализ.**

Первая строка полученной матрицы имеет вид

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3.19668 | -3.79684 | 2.06780 | -0.50824 | 0.04409 |

Эти числа являются коэффициентами собственного многочлена

*.*

То есть это .

Возьмём собственные значения матрицы, подсчитанные с помощью метода вращений с точностью 16 знаков и найдём невязки вида:

Получим

8.326672684688674e-17

2.636779683484747e-16

6.938893903907228e-17

9.71445146547012e-17

-1.249000902703301e-16

Экономичность:

Метод имеет сложность Это говорит о том, что его использование при больших размерностях матриц затруднительно.

Также проблемой являются нерегулярные случаи, которые увеличивают число операций, но в данной работе не рассматривались.

Точность:

Невязки для собственных значений имеют 16 и 17 порядок. Это говорит о том, что метод является точным, приближённые значения мы получаем только из-за числа хранимых знаков у переменных типа float в Python (17 знаков) и связанных с этим округлений.