**Министерство образования Республики Беларусь**

**Белорусский государственный университет**

Факультет прикладной математики и информатики

Лабораторная работа №6

По курсу «Вычислительные методы алгебры»

**Итерационный метод вращений**

Вариант №5

Работу выполнил:

студент 3 курса 7 группы

**Шатерник Артём**

Преподаватель:

**Будник А. М**.

**Минск 2023**

1. **Постановка задачи.**

Дана матрица следующего вида

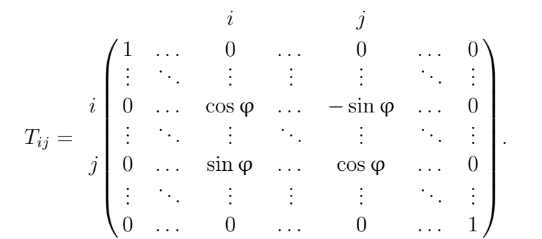
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0.5757 | -0.0758 | 0.0152 | 0.0303 | 0.1061 |  |
|  |  |  |  |  |  |
| 0.0788 | 0.9014 | 0.0000 | -0.0606 | 0.0606 |  |
|  |  |  |  |  |  |
| 0.0455 | 0.0000 | 0.7242 | -0.2121 | 0.1212 |  |
|  |  |  |  |  |  |
| -0.0909 | 0.1909 | 0.0000 |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

Требуется методом вращений найти спектр и систему собственных векторов матрицы Вычислить невязки оценить их значения. Точность

1. **Алгоритм решения.**

Итерация метода имеет следующий вид

Матрица является матрицей вращений вида

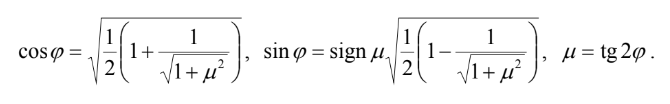


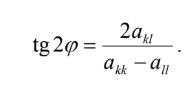
При умножении на матрицы вращений меняются строки и столбцы и можно уменьшить число операций используя соответствующие формулы. Пусть тогда получаем

Остальные элементы остаются неизменными.

И для получаем

считаем по следующим формулам





где

Также пусть сумма квадратов недиагональных элементов на каждой строке. близость матрицы к диагональной.

Критерий остановки итерационного процесса

**Способ выбора индексов у матрицы вращений.**

Берём индексы соответствующие оптимальному элементу матрицы. Его будем выбирать следующим образом.

* Будем хранить суммы
* В начале каждой итерации выбираем максимальную из этих сумм.
* В качестве оптимального элемента берём максимальны недиагональный элемент из строки, которой соответствует сумма

Также при таком способе выбора элемента при каждой итерации будет изменяться только суммы, так что нет нужды пересчитывать их все на каждой итерации.

1. **Листинг программы.**

**def build\_sum(sum, matrix):  
 for i in range(matrix.shape[0]):  
 for j in range(matrix.shape[0]):  
 if i != j:  
 sum[i] += matrix[i][j]\*\*2  
 return sum  
  
def find\_optim\_elem(sum, matrix):  
 max\_sum = max(sum)  
 max\_index = sum.index(max\_sum)  
 row\_list = matrix[max\_index].tolist()  
 row\_list.pop(max\_index)  
 return [max\_index, matrix[max\_index].tolist().index(max(row\_list, key=abs))]  
  
  
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\_copy = np.copy(a\_matrix)  
a\_matrix = np.array(a\_matrix)  
# Точность  
epsilon = 1e-5  
# Симметричный вид  
a\_matrix = np.matmul(a\_matrix.T, a\_matrix)  
diag\_sum = [0 for i in range(size)]  
diag\_sum = build\_sum(diag\_sum, a\_matrix)  
for i in range(size):  
 for j in range(size):  
 print(np.round(a\_matrix[i][j], 5), end='')  
 print(" ", end='')  
 print()  
# След матрицы  
print(np.trace(a\_matrix))  
# Метод вращений  
n = 0  
u\_matrix = np.identity(size)  
while True:  
 n += 1  
 optim\_index = find\_optim\_elem(diag\_sum, a\_matrix)  
 k, l = optim\_index[0], optim\_index[1]  
 tg = 2 \* a\_matrix[k][l] / (a\_matrix[k][k] - a\_matrix[l][l])  
 cos = math.sqrt(0.5 \* (1 + 1 / (math.sqrt(1 + tg\*\*2))))  
 sin = math.sqrt(0.5 \* (1 - 1 / (math.sqrt(1 + tg\*\*2))))  
 if tg < 0:  
 sin = sin \* -1  
 # A  
 b\_matrix = np.copy(a\_matrix)  
 for i in range(size):  
 b\_matrix[i][k] = a\_matrix[i][k] \* cos + a\_matrix[i][l] \* sin  
 b\_matrix[i][l] = -1 \* a\_matrix[i][k] \* sin + a\_matrix[i][l] \* cos  
 a\_matrix = np.copy(b\_matrix)  
 # U  
 u\_copy = np.copy(u\_matrix)  
 for i in range(size):  
 u\_copy[i][k] = u\_matrix[i][k] \* cos + u\_matrix[i][l] \* sin  
 u\_copy[i][l] = -1 \* u\_matrix[i][k] \* sin + u\_matrix[i][l] \* cos  
 u\_matrix = np.copy(u\_copy)  
 # A  
 for i in range(size):  
 b\_matrix[k][i] = a\_matrix[k][i] \* cos + a\_matrix[l][i] \* sin  
 b\_matrix[l][i] = -1 \* a\_matrix[k][i] \* sin + a\_matrix[l][i] \* cos  
 a\_matrix = np.copy(b\_matrix)  
 k\_sum = 0  
 for i in range(size):  
 if i != k:  
 k\_sum += a\_matrix[k][i]\*\*2  
 diag\_sum[k] = k\_sum  
 l\_sum = 0  
 for i in range(size):  
 if i != l:  
 l\_sum += a\_matrix[l][i]\*\*2  
 diag\_sum[l] = l\_sum  
 if abs(sum(diag\_sum)) <= epsilon:  
 break  
print("Num of iterations: " + str(n))  
for i in range(size):  
 for j in range(size):  
 print(np.round(a\_matrix[i][j], 5), end='')  
 print(" ", end='')  
 print()  
print()  
  
# Столбцы - собственные векторы соответствующие собственным значениям  
for i in range(size):  
 for j in range(size):  
 print(np.round(u\_matrix[i][j], 5), end='')  
 print(" ", end='')  
 print()  
  
print("\nСобственные значения и соответствующие им векторы: ")  
for i in range(size):  
 print(np.round(a\_matrix[i][i], 5) ,end=': ( ')  
 for j in range(size):  
 print(np.round(u\_matrix[j][i], 5) ,end=' ')  
 print(')')**

**# Невязки  
temp = np.matmul(a\_copy.T, a\_copy)  
for i in range(size):  
 res = np.matmul(temp, u\_matrix.T[i]) - a\_matrix[i][i] \* u\_matrix.T[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, 2), '.4e'))**

**# Из метода Данилевского  
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.**

Собственные значения и соответствующие им векторы:

0.19101: ( 0.75335 0.00961 0.21605 0.14289 -0.60438 )

0.87966: ( -0.00436 0.95178 -0.11009 0.28386 0.03745 )

0.38356: ( -0.32193 -0.08912 0.74177 0.58155 -4e-05 )

0.5977 : ( 0.24177 -0.28462 -0.4987 0.72634 0.29029 )

1.14475: ( 0.51995 0.07123 0.3772 -0.18232 0.74098 )

Число итераций: 13.

Невязки для собственных значений вида

Собственный многочлен был подсчитан методом Данилевского и имеет точность

2.3112870373154237e-07

8.279974543501378e-08

-3.863054033603763e-08

-8.784332219957669e-09

-3.2410955376482864e-07

Невязки

(-4.1150e-04 3.2264e-04 8.9487e-04 -6.9461e-04 -3.5213e-04 )

Норма невязки: 1.2964e-03

(-1.0281e-04 2.9648e-04 3.2433e-04 -8.4262e-04 -2.0671e-04 )

Норма невязки: 9.7796e-04

(1.0703e-03 -1.2309e-04 6.7276e-04 -2.8440e-04 9.0390e-04 )

Норма невязки: 1.5847e-03

(-9.3787e-04 -9.1076e-04 -1.5455e-04 -4.3727e-04 7.1672e-04 )

Норма невязки: 1.5614e-03

(-4.8148e-04 -4.9524e-05 1.1225e-03 9.1321e-04 -4.1094e-06 )

Норма невязки: 1.5259e-03

**Применялась первая норма.**

Экономичность:

Сложность одной итерации .Получилось добиться такой сложности благодаря способу выбора оптимального элемента. Если бы каждый раз выбирался самый большой недиагональный элемент в матрице, то сложность бы была .

Точность:

Метод является итерационным и даёт наперёд заданную точность.

Невязки получились с точностью даже выше, чем заданная

( против ). Невязки получились с точностью ниже заданной, так как в этих невязках используются и собственные значения и собственные векторы, для которых были использованы не точные, а приближённые значения с точностью .