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

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 вращений найти спектр и 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. Точность $\varepsilon = 10^{-5}$.

2. Алгоритм решения.

Итерация метода имеет следующий вид

$$A^{k+1} = \left(T_{ij}^k\right)^T A^k T_{ij}^k .$$

Матрица T_{ij}^k является матрицей вращений вида

$$T_{ij} = \begin{array}{c} \\ i \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \cos\varphi & \dots & -\sin\varphi & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & \sin\varphi & \dots & \cos\varphi & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array} .$$

При умножении на матрицы вращений меняются $i-$ые и $j-$ые строки и столбцы и можно уменьшить число операций используя соответствующие формулы. Пусть $B = T_{kl}A$, тогда получаем

$$\begin{aligned} b_{ik} &= a_{ik}\, cos\varphi + a_{il}\, sin\varphi \\ b_{il} &= -a_{ik}\, sin\varphi + a_{il}\, cos\varphi \end{aligned} \qquad i = 1, \dots, n$$

Остальные элементы остаются неизменными.

И для $C = T_{kl}^{T} B$ получаем

$$\begin{aligned} c_{ki} &= b_{ki}\, cos\varphi + b_{li}\, sin\varphi \\ c_{li} &= -b_{ki}\, sin\varphi + b_{li}\, cos\varphi \end{aligned} \qquad i = 1, \dots, n$$

$sin\varphi, cos\varphi$ считаем по следующим формулам

$$\cos\varphi = \sqrt{\frac{1}{2}\left(1 + \frac{1}{\sqrt{1+\mu^2}}\right)}, \quad \sin\varphi = \operatorname{sign}\mu \sqrt{\frac{1}{2}\left(1 - \frac{1}{\sqrt{1+\mu^2}}\right)}, \quad \mu = \operatorname{tg} 2\varphi .$$

где $\operatorname{tg} 2\varphi = \dfrac{2a_{kl}}{a_{kk} - a_{ll}}$.

Также пусть $\sigma_i(A) = \sum_{j=1, i \neq j}^{n} |a_{ij}|^2$, $\overline{i = 1, n}$ сумма квадратов недиагональных элементов на каждой строке. $t(A) = \sigma_1 + \sigma_2 + \cdots + \sigma_n$ близость матрицы к диагональной.

Критерий остановки итерационного процесса

$$\left| t(A^{k+1}) \right| \leq \varepsilon$$

Способ выбора индексов *ij* у матрицы вращений.

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

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

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

3. 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)
```

4. 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

$$\varphi_i(\lambda_i) = P_n(\lambda_i).$$

Собственный многочлен был подсчитан методом Данилевского и имеет точность 10^{-16}.
$\varphi_i(\lambda_i) =$

2.3112870373154237e-07
8.279974543501378e-08
-3.863054033603763e-08
-8.784332219957669e-09
-3.2410955376482864e-07

Невязки

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

(-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 $O(n)$.Получилось добиться такой сложности благодаря способу выбора оптимального элемента. Если бы каждый раз выбирался самый большой недиагональный элемент в матрице, то сложность бы была $O(n^2)$.

Точность:

Метод является итерационным и даёт наперёд заданную точность.
Невязки $\varphi_i(\lambda_i)$ получились с точностью даже выше, чем заданная (10^{-7} против 10^{-5}). Невязки r_i 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 10^{-5}.