МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа
по курсу «Численные методы»

Нахождение собственных значений и собственных векторов симметричных разреженных матриц большой размерности. Метод Ланцоша.

Выполнила: К.О. Михеева
Группа: 8О-407Б
Преподаватель: Ревизников Д.Л.

Москва, 2023

Условие

Для разреженной симметричной матрицы большого размера найти собственные значения и собственные вектора методом Ланцоша.

Описание

Алгоритм Ланцоша соединяет в себе метод Ланцоша для построения крыловского подпространства с процедурой Рэлея-Ритца. Входными данными алгоритма служат квадратная матрица $A=A^T$ и вектор начального приближения b.

Мы пытаемся найти трехдиагональную симметричную матрицу $T_k=Q_k^T A Q_k$, собственные значения которой приближают собственные значения матрицы A. Иными словами на k-м шаге из ортонормированных векторов Ланцоша строится матрица $Q_k = [q_1,q_2,...,q_k]$ и в качестве приближенных собственных значений матрицы A принимаются числа Ритца.

Пусть $T_k=V \Lambda V^T$ есть спектральное разложение матрицы T_k, столбцы матрицы $Q_k V$ рассматриваются как приближения к соответствующим собственным векторам матрицы A.

Диагональные элементы обозначены как $\alpha j = tjj$, а элементы побочной диагонали $\beta j=tj-1,j = tj,j-1$. После каждой итерации мы вычисляем αj, βj, из которых строится матрица T.

Алгоритм Ланцоша:

1) Заполняем начальные значения

$$q_1 = b/||b||,$$
$$\beta_1 = 0,$$
$$q_0 = 0,$$

где b - произвольный вектор.
Для всех j = 1..k

2) Пусть

$$z = Aq_j$$

3) Вычисляем элемент на позиции tjj матрицы Tk.

$$\alpha_j = q_j^T z$$

4) Два раза повторим полную пере ортогонализации Грамма-Шмидта:

$$z = z - \sum_{i=1}^{j-1} (z^T q_i) q_i$$
$$z = z - \sum_{i=1}^{j-1} (z^T q_i) q_i$$

5) Обновим

$$z = z - \alpha_j q_j - \beta_j q_{j-1}$$

6) Вычисляем элементы на позициях tj,j+1 и tj+1,j

$$\beta_{j+1} = ||z||$$

7) Если βj+1 = 0, то алгоритм завершается

Исходный код

```
import numpy as np

eps = 0.0001

# Вычисляем норму
def norm(b):
    s = 0
    for i in range(len(b)):
        s += b[i] ** 2
    return s

# Определяем знак
def sign(a):
    if a > 0:
        return 1
    elif a < 0:
        return -1
    return 0

# Нахождения квадрата максимального элкмента в матрице A
def max_element(A):
    max_val = 0
    for k in range(1, len(A)):
        if abs(A[k][0]) > eps:
            max_val += A[k][0] ** 2
    return pow(max_val, 2)

def QR(A):
    N = len(A) # Размер матрицы
    newA = np.zeros((N, N)) # Временная матрица, которая
используется для хранения новой матрицы A после каждой итерации
QR-разложения.
    H = np.zeros((N, N)) # Матрица Хаусхолдера
    Q = np.zeros((N, N)) # Матрица ортогонализации
    newQ = np.zeros((N, N)) # Ввременная матрица, используемая для
хранения новой матрицы Q после каждой итерации.
```

```
    Qtemp = np.zeros((N, N)) # Временная матрица, используемая для
хранения промежуточных значений при обновлении матрицы Q.
    Qp = np.zeros((N, N)) # Произведение всех матриц Q, полученных
на каждой итерации.
    R = np.zeros((N, N)) # Верхнетреугольная матрица
    vvt = np.zeros((N, N)) # Произведение вектора v на его
транспонированный вектор.
    v = np.zeros(N) # Вектор Хаусхолдера

    for i in range(N):
        Q[i][i] = 1

    for i in range(N):
        Qp[i][i] = 1

    while max_element(A) > eps:
        for i in range(N - 1):
            j = 0
            while j < i:
                v[j] = 0
                j += 1

            sum_val = sum(pow(A[k][i], 2) for k in range(i, N))
            sum_val = pow(sum_val, 0.5)
            v[j] = A[j][i] + sign(A[j][i]) * sum_val

            for j in range(i + 1, N):
                v[j] = A[j][i]

            vvt = np.zeros((N, N))
            for i in range(N):
                for j in range(N):
                    vvt[i][j] = v[i] * v[j]

            vtv = 0
            for i in range(N):
                vtv += v[i] ** 2

            H = -2 * vvt / vtv
            H[np.arange(N), np.arange(N)] += 1

            newQ = Q @ H
            newA = H @ A

            A = newA
            Q = newQ

            newA = np.zeros((N, N))
```

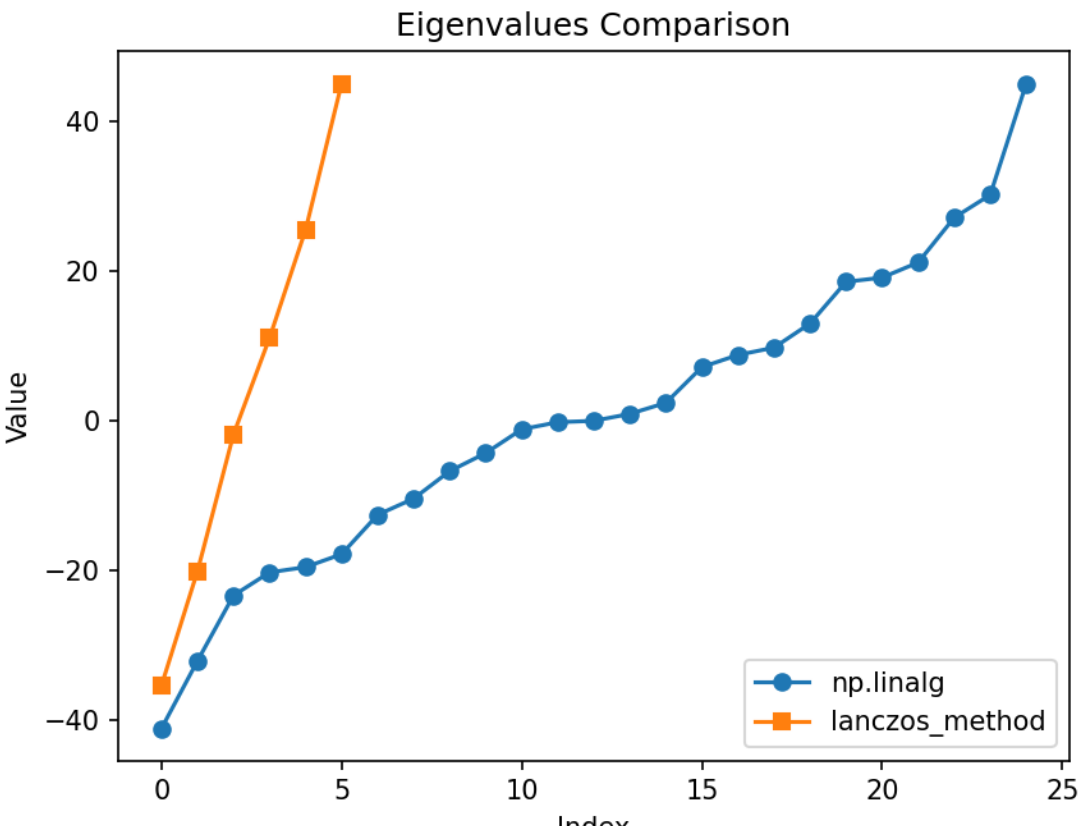
```
           newQ = np.zeros((N, N))

       Qtemp = Qp @ Q
       Qp = Qtemp
       Qtemp = np.zeros((N, N))

       R = A
       A = R @ Q

       # В единичную матрицу
       Q = [[1 if i == j else 0 for j in range(N)] for i in
range(N)]

   lambda1 = A[0][0]


print("___________________________________________________________
______")
   print("\nСобственные значения:\n")
   print("λ1 =", round(lambda1 / 100, 10))
   # Дискриминант
   d = pow(A[1][1] + A[2][2], 2) - 4 * (A[1][1] * A[2][2] -
A[1][2] * A[2][1])
   if d >= 0:
       d = pow(d, 0.5)
       lambda2 = (A[1][1] + A[2][2] + d) / 2
       lambda3 = (A[1][1] + A[2][2] - d) / 2
       print("λ2 =", round(lambda2, 10))
       print("λ3 =", round(lambda3 / 100, 10))
   else:
       d = pow(-d, 0.5)
       print("λ2 =", round(lambda2, 10), "+", round(d / 2, 10),
"i")
       print("λ3 =", round(lambda3 / 100, 10), "-", round(d / 2,
10), "i")


print("___________________________________________________________
______")
   print("\nСобственные векторы:\n")
   for i in range(1, N + 1):
       print(f"X_{i} = {list(np.round(Qp[i - 1], 10))}")

print("___________________________________________________________
______")

matrix = input("Матрица: ") + ".txt"
with open(matrix, "r") as file:
```

```
   N = int(file.readline())
   A = np.zeros((N, N))
   b = np.zeros(N)

   for i in range(N):
       A[i] = list(map(float, file.readline().split()))

   b = list(map(float, file.readline().split()))

# Алгоритм Ланцоша
k = 3 # Размер трехдиагональной матрицы T
q = np.zeros((k + 2, N))
T = np.zeros((k, k))
z = np.zeros(N)
temp = np.zeros(N)
alpha = 0
beta = 0

for i in range(N):
   q[1][i] = b[i] / norm(b)

for i in range(1, k + 1):
   for j in range(N):
       s = np.sum(A[j] * q[i])
       z[j] = s

   # Диагональные элементы T
   alpha = np.sum(q[i] * z)
   T[i - 1][i - 1] = alpha

   if i != 1: # Внеглавные элементы
       T[i - 1][i - 2] = beta
       T[i - 2][i - 1] = beta

   # Полная переортогогализация Грамма-Шмида
   for p in range(2):
       for j in range(N):
           for k in range(1, i):
               s = np.sum(z * q[k])
               temp += q[k] * s
           z -= temp
           temp = np.zeros(N)

   # Обновление вектора
   z -= alpha * q[i] + beta * q[i - 1]

# Эл-ты на позициях T[i][i+1] T[i+1][i]
   beta = norm(z)
```

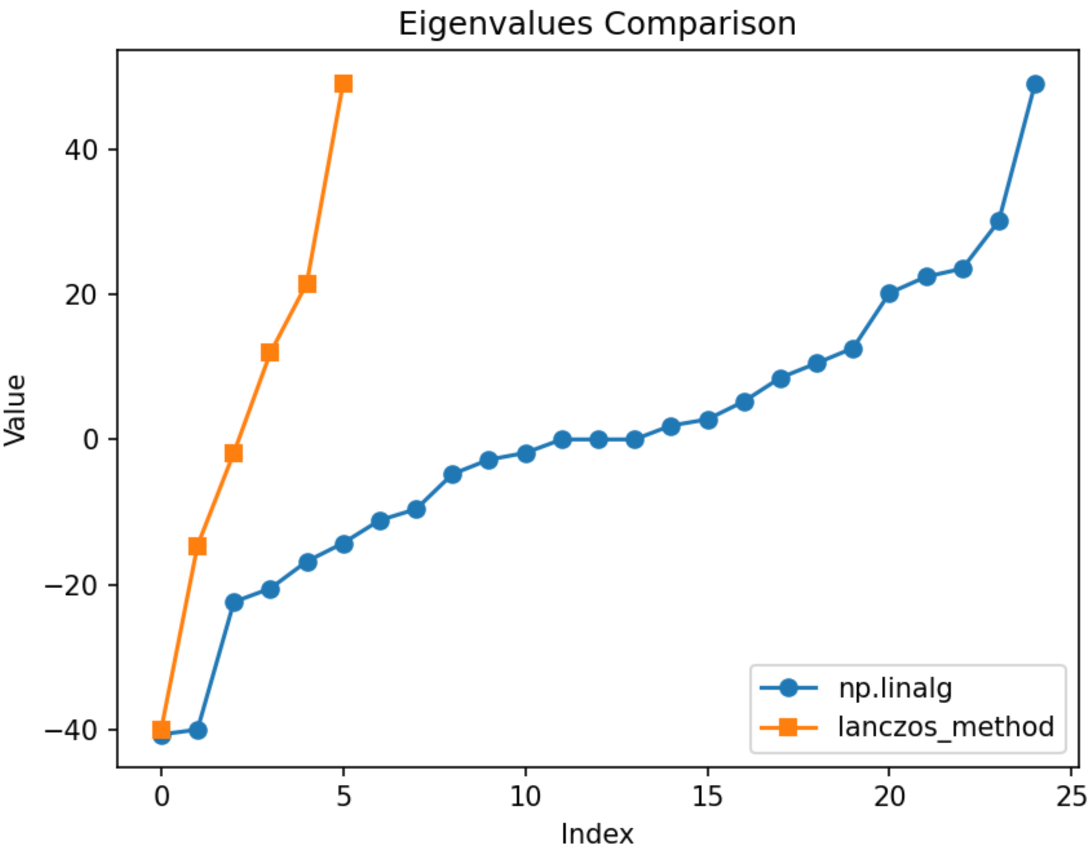
```
    q[i + 1] = z / beta

# Конец алгоритма
    if beta == 0:
        break
QR(T)
```

Вывод программы

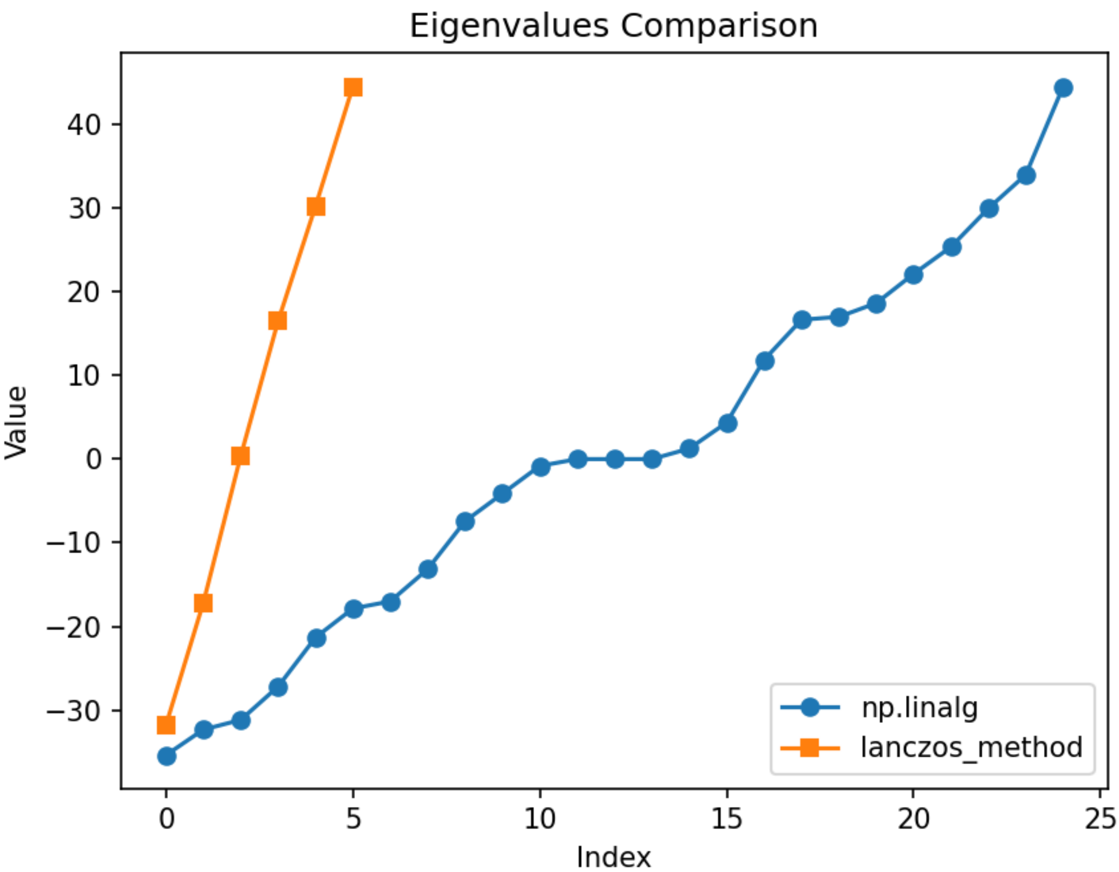
```
     Eigenvalues with np.linalg: [-41.04685849 -32.03433379
-23.29735507 -20.21730832 -19.51008242
 -17.78754726 -12.53758301 -10.36922438  -6.70856575
-4.29603993
  -1.1234521   -0.17753616   0.           0.9329634
2.40495214
   7.17636628   8.79358199   9.77086932  12.99200083
18.549754
  19.11395639  21.1274933   27.11033343  30.15422833
44.97938727]
Eigenvalues with lanczos_method: [-35.32780494 -20.07318299
-1.81659172  11.06098614  25.4769686
  44.97379117]
```



```
Eigenvalues with np.linalg: [-40.70004287 -40.0306709  -22.41816329
-20.55545358 -16.82921736
 -14.29794688 -11.15208625  -9.61944803  -4.7810925   -2.78726403
  -1.85718355  -0.          -0.           0.           1.91046746
   2.77129555   5.24026605   8.50269525  10.51760257  12.60507813
  20.19914865  22.43904978  23.56285412  30.11821603  49.16189564]
Eigenvalues with lanczos_method: [-40.11730414 -14.65696012
-1.82601911  11.93901303  21.52180471
  49.16091427]
```



```
[-35.38224184 -32.33127085 -31.16845777 -27.18579153 -21.32037359
 -17.85950469 -17.         -13.15387105  -7.4272055   -4.12563618
  -0.81716494   0.           0.           0.           1.26113234
   4.37105858  11.86774926  16.65520538  17.          18.64439281
  22.13178264  25.36550266  29.96782008  33.99318298  44.5136912 ]
Eigenvalues with lanczos_method: [-31.79186644 -17.1738335
0.36171442  16.49978348  30.21471894
  44.50849701]
```



Вывод

Выполнив данный курсовой проект, я познакомилась с одним из методов решений частичной задачи СЗ симметричной матрицы - методом Ланцоша, который сводит симметричную вещественную матрицу к симметричной матрицы меньшего размера. Я реализовала его в виде программы, а также повторила и закрепила QR-алгоритм нахождения СЗ, который был изучен мною ранее.