Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 4

по курсу «Численные методы линейной алгебры»

«Вычисление собственных значений и собственных векторов симметричной матрицы методом А.М. Данилевского»

Студентка группы ИУ9-72Б Князихин Д. П.

Преподаватель Посевин Д. П.

Москва 2023

1 Цель

Реализовать метод вычисления собственных значений и собственных векторов симметричной матрицы методом А.М. Данилевского.

Реализовать метод поиска собственных значений действительной симметричной матрицы A размером 4x4.

Проверить корректность вычисления собственных значений по теореме Виета.

Проверить выполнение условий теоремы Гершгорина о принадлежности собственных значений соответствующим объединениям кругов Гершгорина.

Вычислить собственные вектора и проверить выполнение условия ортогональности собственных векторов.

Проверить решение на матрице приведенной в презентации.

Продемонстрировать работу приложения для произвольных симметричных матриц размером n x n с учетом выполнения пунктов приведенных выше.

2 Практическая реализация

Класс Main

```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

def funcShow(f, k):
    x = np.linspace(-10, 10, 2000)
    plt.plot(x, f(x, k))
    plt.plot(x, 0*x)
    plt.show()



def f(x, koef):
    sum = 0
    for i in range(len(koef)):
        sum += koef[i] * x**(len(koef)-1-i)
    return sum

delta = 10**(-3)
```

```
def BisectionMethod(xn, xk, F, koef):
    eps = 10**(-10)
    x_new = 0
    while abs(xn-xk) > eps:
        x_new = (xn + xk) / 2

        if np.sign(F(xn, koef)) != np.sign(F(x_new, koef)):
            xk = x_new
        else:
            xn = x_new
    return x_new

def searchSegments(l, R, F, koef):
    Segments = []
    nextPoint = l + delta
    while l < R:
        if np.sign(F(l, koef)) != np.sign(F(nextPoint, koef)):
            Segments.append([l, nextPoint])
            l = nextPoint
        if nextPoint > R:
            break
        nextPoint += delta
    return Segments

def getRoots(l, R, F, koef):
    roots = []
    for i in searchSegments(l, R, F, koef):
        answ = BisectionMethod(i[0], i[1], F, koef)
        roots.append(answ)
        print(i, "  ", answ)
    return roots

def mul_matrix(A, B):
    n = len(A)
    m = len(B[0])
    C = [[0] * m for _ in range(n)]
    for i in range(n):
        for j in range(m):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

```python
def getD(D, indexLine, indexCol, B_list):

    D_new = np.zeros((len(D), len(D)))
    B_new = []
    B_newRevers = []
    for i in range(len(D)):
        B_new.append([])
        B_newRevers.append([])
        for j in range(len(D)):
            if i == j:
                B_new[i].append(1)
                B_newRevers[i].append(1)
            else:
                B_new[i].append(0)
                B_newRevers[i].append(0)

    #B_newRevers =  np.copy(B_new)

    for i in range(len(D)):
        if i == indexCol:
            B_new[indexCol][i] = 1 / D[indexLine][indexCol]
            continue
        B_new[indexCol][i] = (-1) * D[indexLine][i] / D[indexLine][indexCol]


    for i in range(len(D)):
        #print(D[indexLine], indexCol, D[indexLine][i])

        B_newRevers[indexCol][i] = D[indexLine][i]

    B_list.append(B_new)
    # print("B_new:   ")
    # print(B_new)
    # print("B_newRevers:   ")
    # print(*B_newRevers)

    #return
    C = mul_matrix(B_newRevers, D)
    D_new = mul_matrix(C, B_new)

     # print("D_new:   ", D_new)

    if indexCol == 0:
        #print("        ", D_new)
        return [D_new, B_list]
    else:
```

```
        out = getD(D_new, indexLine - 1, indexCol - 1, B_list)
    return out


def eigenValue(matrix):
    values = []
    for i in range(len(matrix)):
        values.append(matrix[i][i])
    return values


def getKoef(P):
    k = [1]
    for i in range(len(P)):
        k.append(-1 * P[0][i])
    return k

def norma(vector):
    sum = 0
    for i in vector:
        sum += i[0]**2
    for i in range(len(vector)):
        vector[i][0] /= (sum**0.5)
    return vector

def getVectors(roots, B_lists):
    print(roots)
    print()
    m = B_lists[0]
    for i in B_lists[1::]:
        m = mul_matrix(m, i)
    print(m)
    print()
    ys = []
    for i in range(len(roots)):
        y = []
        for j in range(len(B_lists[0])):
            y.append([roots[i]**(len(B_lists[0]) - j - 1)])
        ys.append(y)
    xs = []
    print(ys)
    for i in ys:

        print()
        xs.append(norma(mul_matrix(m, i)))
```

```
    return xs

def mul_vector(a, b):
    s = 0
    for i in range(len(a)):
        s += a[i][0]*b[i][0]
    return s

def checkNorm(xs):
    for i in xs:
        for j in xs:
            if i != j:
                s = mul_vector(i, j)
                if abs(s) < 10**(-1):
                    print(i, j, "    ", "TRUE")
                else:
                    print(i, j, "    ", "FALSE")

def Gershgorin(matrix):
    circles = []
    for i in range(len(matrix)):
        a = matrix[i][i]
        r = sum(map(abs, matrix[i]))
        circles.append([a, r])
    return circles

def ShowCrcles(circles, lambds):
    _, axes = plt.subplots()
    print(lambds)
    for i in lambds:
        plt.plot(i, 0, 'ro')
    l, r = 0, 0

    for i in circles:
        print(i)
        d = plt.Circle((i[0], 0), i[1], fill=False)
        radius = i[1]
        l = min(l, i[0] - 2*radius)
        r = max(r, i[0] + 2*radius)
        axes.add_patch(d)

    axes.set_aspect("equal", "box")
    plt.axis('scaled')
    x = np.linspace(l, r, 2000)
    axes.plot(x, 0*x)
```

```python
    plt.title("Gershgorin circles")
    plt.show()

def check_viet(koef, roots):
    l = [1]
    l.append(koef)

    check_sum = sum(roots)

    if abs(check_sum + koef[1]) > 0.001:
        print(koef[1])
        print("ERROR sum ", abs(check_sum + koef[0]))
    check_mul = np.prod(np.array(roots))
    if abs(check_mul - ((-1)**(len(roots)))*koef[-1]) > 0.001:
        print("ERROR mul ",  abs(check_mul - ((-1)**(len(roots)))*koef[-1]))
    return

def main():
   # getRoots(-100, 100, f)
    ms =   [[[3, 4], [1, 2]], [[1,2,3],[4,5,6],[3,2,4]]]  #[[2.2, 1, 0.5, 2],
        [1, 1.3, 2, 1], [0.5, 2, 0.5, 1.6], [2, 1, 1.6, 2]]
    ms = [[[2.2, 1, 0.5, 2], [1, 1.3, 2, 1], [0.5, 2, 0.5, 1.6], [2, 1, 1.6,
        2]]]
    for matrix in ms:
        t = getD(matrix, len(matrix)- 1, len(matrix)- 2, [])
        P, B_lists = t[0], t[1]
        koef = getKoef(P)
        roots = getRoots(-100, 100, f, koef)


        xs =getVectors(roots, B_lists)

        for i in xs:
            print(i)

        print()
        checkNorm(xs)

        funcShow(f, koef)

        ShowCrcles(Gershgorin(matrix), roots)
        check_viet(koef, roots)

    return

main()
```

3 Результаты

График характеристического уравнения матрицы 4×4 представлен на рисунке 1.

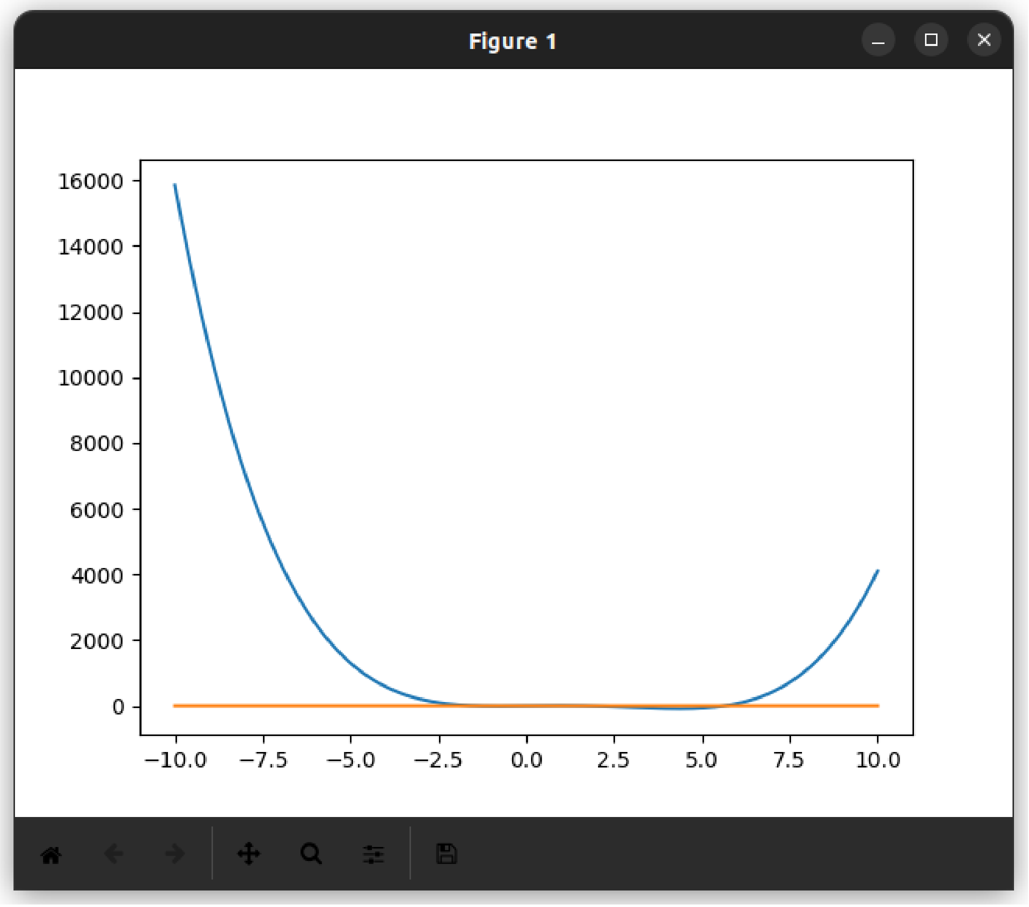

Рис. 1 — График функции

Круги Гершгорина представлен на рисунке 2.

Проверка ортонормированности для матрицы 4×4 на рисунке 3.

Проверка теоремой Виета на рисунке 4.

Результаты работы на произволной матрице 5×5 представленны на рисууках 5- 6- 7- 8

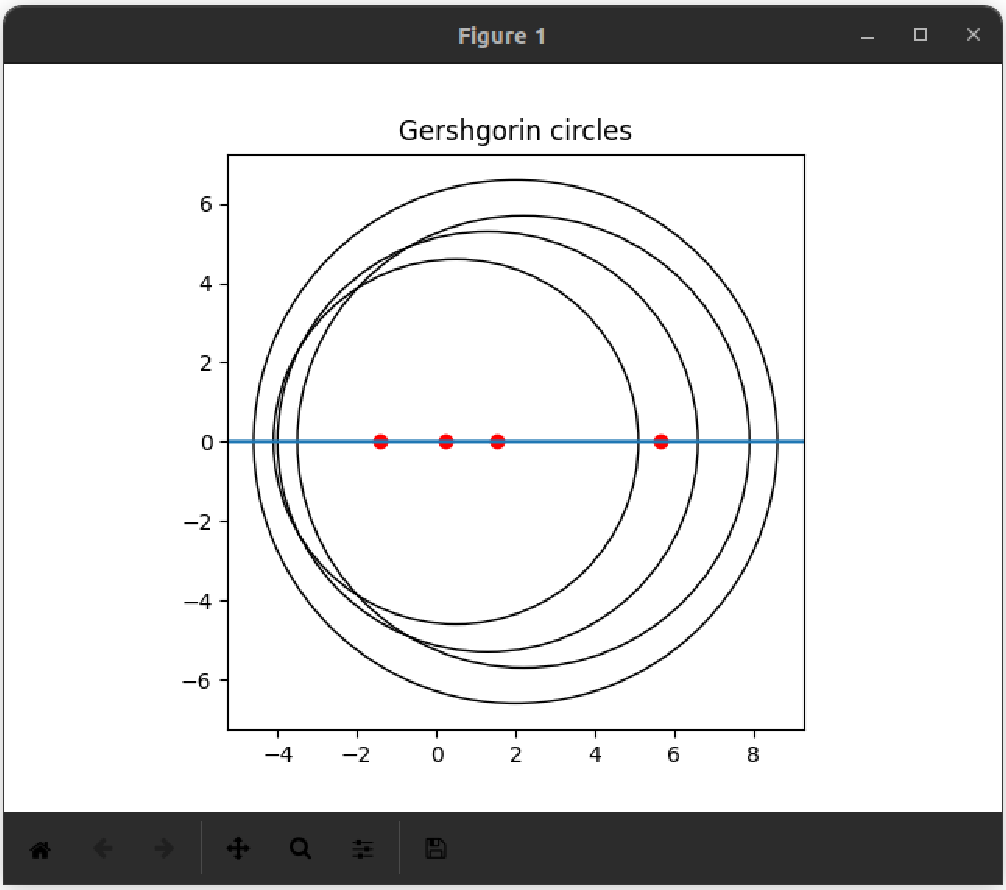

Рис. 2 — Круги Гершгорина

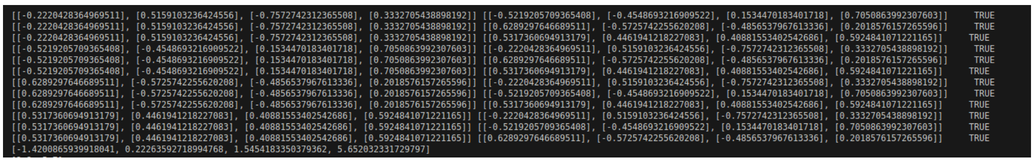

Рис. 3 — Проверка

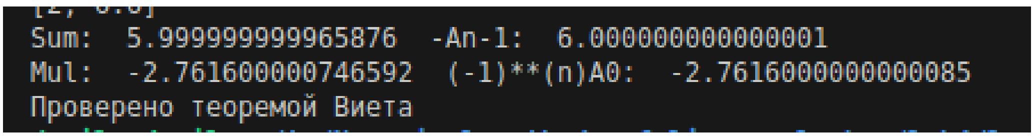

Рис. 4 — Проверка теоремой Виета

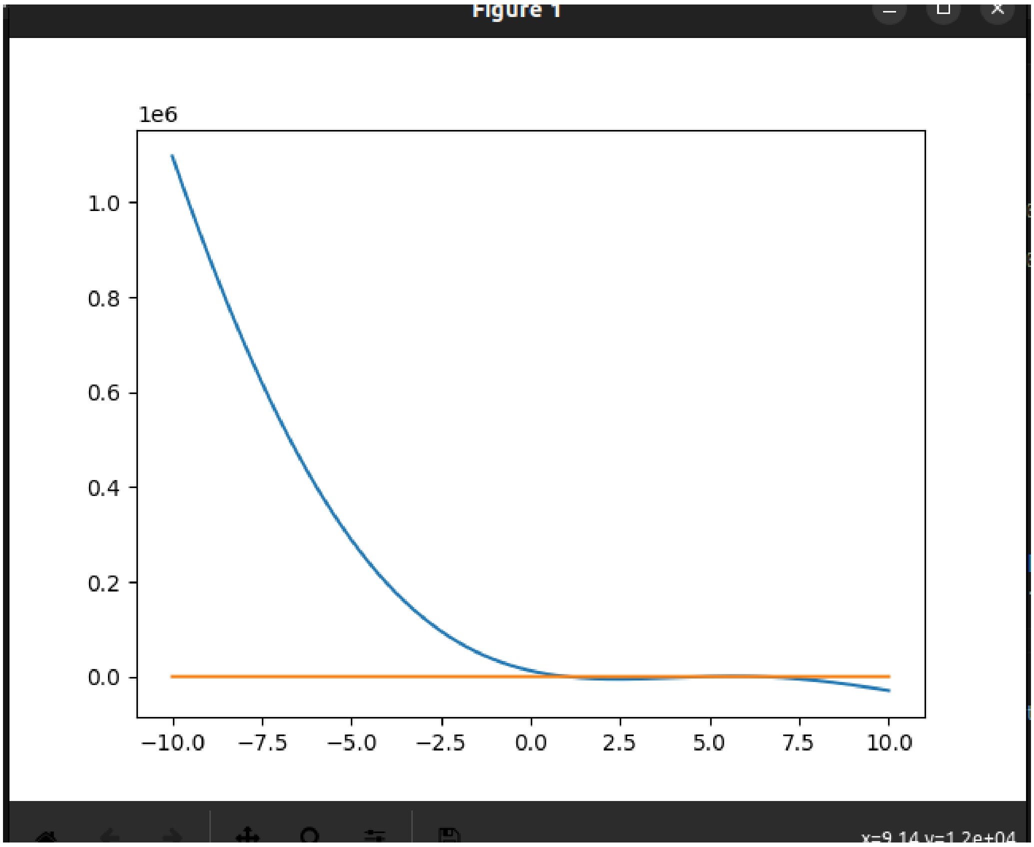

Рис. 5 — График функции

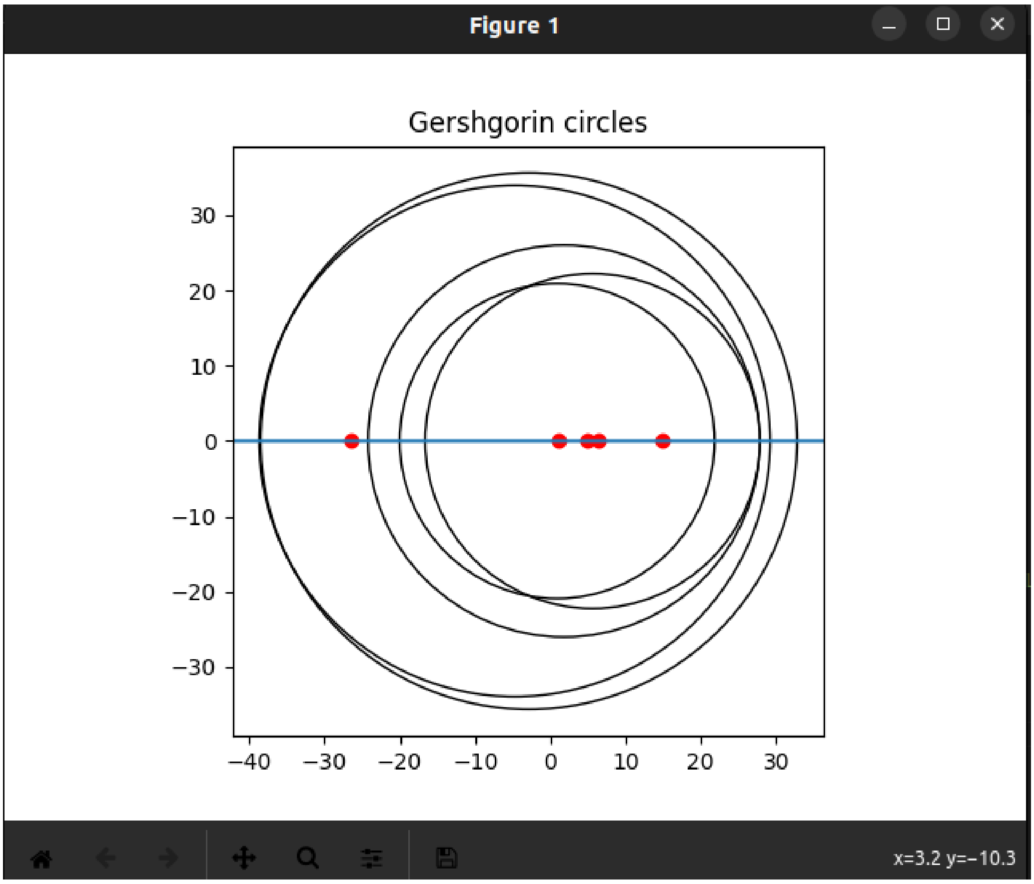

Рис. 6 — Круги Гершгорина

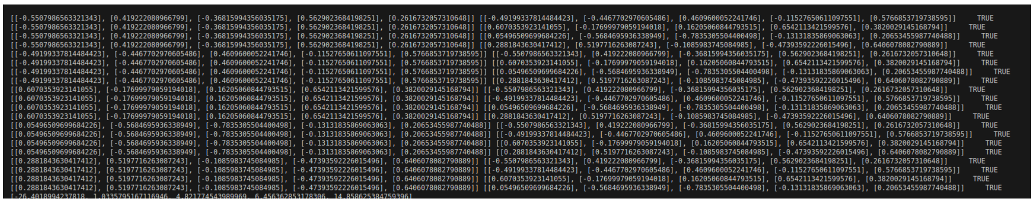

Рис. 7 — Проверка

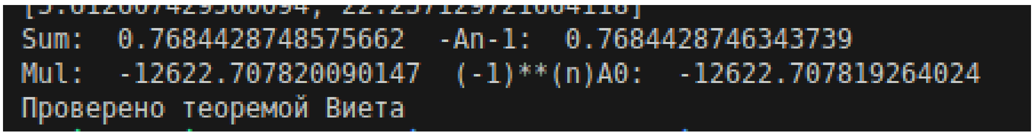

Рис. 8 — Проверка теоремой Виета