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 № 4.2

по курсу «Численные методы линейной алгебры»

«Вычисление собственных значений и собственных векторов симметричной матрицы методом А.Н. Крылова»

Студент группы ИУ9-71Б Яровикова А. С.

Преподаватель Посевин Д. П.

Москва 2023

1 Цель работы

Реализовать метод вычисления собственных значений и собственных векторов симметричной матрицы методом А.Н. Крылова.

2 Задание

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

2. Проверить корректность вычисления собственных значений по теореме Виета.

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

4. Вычислить собственные вектора и проверить выполнение условия ортогональности собственных векторов.

5. Проверить решение на матрице приведенной в презентации.

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

7. Сравнить результат, полученный методом Крылова, с результатом, полученным методом Данилевского.

3 Реализация

Исходный код программы представлен в листингах 1– 7.

Листинг 1 — Вспомогательные функции

```
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
import sys

def generate_symmetrical_matrix(l, r, n):
    a = np.random.uniform(l, r, (n, n))
    a = np.tril(a) + np.tril(a, -1).T
    return a

def euclidean_norm(vec):
    res = 0
    for el in vec:
        res += el**2
    return np.sqrt(res)

def mul_on_vector(matrix, vector):
    res = []
    for i in range(len(matrix)):
        el = 0
        for j in range(len(vector)):
            el += matrix[i][j] * vector[j]
        res.append(el)
    return res

def scalar_mul(vec1, vec2):
    if len(vec1) < len(vec2):
        length = len(vec2)
    else:
        length = len(vec1)
    res = 0
    for i in range(0, length):
        res += vec1[i] * vec2[i]
    return res

def print_matrix(a):
    for i in range(len(a)):
        print(a[i])

def identity_matrix(n):
  return np.identity(n)
```

Листинг 2 — Метод Крылова

```
def krylov_algo(matrix):
  p = []
  n = len(matrix)
  D = deepcopy(matrix)
  y = [[1] * n]
  A = []
  for i in range(1, n + 1):
    y.append(np.dot(D, y[i - 1]))
  for i in range(n - 1, -1, -1):
    A.append(y[i])
  A = np.transpose(np.array(A))
  f = y[n]
  res_p = np.linalg.solve(A, f)
  p.append(1)
  for pp in res_p:
    p.append(-pp)
  return y, p
```

Листинг 3 — Вычисление кругов (интервалов Гершгорина)

```

def union_intervals(ints):
  union = []
  for start, end in sorted(ints):
    if union and union[-1][1] >= start - 1:
      union[-1][1] = max(union[-1][1], end)
    else:
      union.append([start, end])
  return union

def find_gershgorin_intervals(matrix):
  A = deepcopy(matrix)
  centers = np.diagonal(A)
  n = len(centers)
  rads = []
  for i in range(n):
    rads.append(np.sum(np.abs(A[i])) - centers[i])
  # print()
  # print(rads)
  intervals = [ (centers[i] - rads[i], centers[i] + rads[i]) for i in
    range(n)]
  print(intervals)
  intervals = union_intervals(deepcopy(intervals))
  return intervals
```

Листинг 4 — Поиск собственных значений матрицы

```
def polynomy(a, x):
  # a[0]*x**(N-1) + a[1]*x**(N-2) + ... + a[N-2]*x + a[N-1]
  val = 0
  n = len(a)
  for i in range(n-1, -1, -1):
    val += a[n - 1 - i] * x**i
  return val

def find_eigen_values(eqCoeffs, intervals, len):
  # print(intervals)
  values = []
  len2 = 10e-7
  for interval in intervals:
    left = interval[0]
    right = interval[1]
    # print(left, right)
    l = int(np.floor((right - left) / len))
    # print(l)
    for i in range(l):
      x_left = left + i * len
      x_right = x_left + len
      # print(f'left x: {x_left}')
      # print(f'right x: {x_right}')
      y_left = polynomy(eqCoeffs, x_left)
      y_right = polynomy(eqCoeffs, x_right)
      # print(f'left y: {y_left}')
      # print(f'right y: {y_right}')
      alpha = y_left * y_right
      # print(alpha)
      if (alpha < 0):
        while x_right - x_left >= len2:
          x_middle = (x_right + x_left) / 2
          y_middle = polynomy(eqCoeffs, x_middle)
          beta = y_left * y_middle
          if beta < 0:
            x_right = x_middle
          else:
            x_left = x_middle
        values.append((x_right + x_left) / 2)
      elif y_left == 0:
        values.append(x_left)
      elif y_right == 0:
        values.append(x_right)
  return values
```

Листинг 5 — Поиск собственных векторов матрицы

```
def find_eigen_vectors(y, lambdas, p):
  n = len(p) - 1
  xs = []
  q = []
  for i in range(n):
    x = np.array(y[n - 1])
    q_i = []
    q_i.append(1)
    for j in range(1, n):
      q_i.append(lambdas[i] * q_i[j - 1] + p[j])
      x = x + np.dot(q_i[j], y[n - 1 - j])
    xs.append(x)
  return xs
```

Листинг 6 — Функция отрисовки графика характеристического уравнения

```
def show_chart(eigenvalues, equatationCoeffs):
    left = eigenvalues[0]
    right = eigenvalues[1]

    for i in range(1, len(eigenvalues)):
        if eigenvalues[i] < left:
            left = eigenvalues[i]
        if eigenvalues[i] > right:
            right = eigenvalues[i]

    interval_len = right - left
    left -= interval_len * 0.1
    right += interval_len * 0.1
    xs = np.linspace(left, right, 1000)
    ys = []
    for x in xs:
      ys.append(polynomy(equatationCoeffs, x))

    plt.plot(xs, ys, color='purple')
    plt.yscale("symlog")
    plt.grid()
    plt.show()
```

Листинг 7 — Запус программы

```
n = 4
a = np.array([[2.2, 1, 0.5, 2], [1, 1.3, 2, 1], [0.5, 2, 0.5, 1.6], [2,
    1, 1.6, 2]])
print('matrix:')
print_matrix(a)
intervals = find_gershgorin_intervals(a)
print(f'\nU intervals: {intervals}')

Ys, P = krylov_algo(a)
print(f'\np:\n {P}')
print()
print(f'Y:\n {Ys}')
print()

vals = find_eigen_values(P, intervals, 10e-3)
print(f'\nlambdas: {vals}')

sum_eigens = np.sum(vals)
sp = [sum(a[i][i] for i in range(0, len(a)))]
# print(sum_eigens)
# print(sp)
if (np.abs(sum_eigens - sp) > 0.1):
  print("\nVieta's theorem doesn't work")
else:
  print("\nVieta's theorem works")

ok = 0
for interval in intervals:
  for i in vals:
    if i > interval[1] or i < interval[0]:
      print("\nGershgorin's theorem error")
      ok = 0
      sys.exit()
    else:
      ok = 1
if ok == 1:
  print("\nGershgorin's theorem works")

show_chart(vals, P)

vecs = find_eigen_vectors(Ys, vals, P)
print('\neigin vectors:')
for i in range(len(vecs)):
  print(f' x_{i+1}: {vecs[i]}')

print('\neigin orthonormal vectors:')
for i in range(len(vecs)):
  print(f' x_{i+1}: {vecs[i]/euclidean_norm(vecs[i])}')

ort = 0
for i in range (n - 1):
  for j in range (i + 1, n):
    scal = scalar_mul(vecs[i], vecs[j])
    if np.abs(scal) > 0.1:
      print("\nEigen vectors are not orthogonal")
      print(np.abs(scal))
      ort = 0
      sys.exit()
    else:
      ort = 1

if ort == 1:
  print("\nEigen vectors are orthogonal")
```

4 Результаты

Результат запуска методов представлены на рисунках 1 - 6.

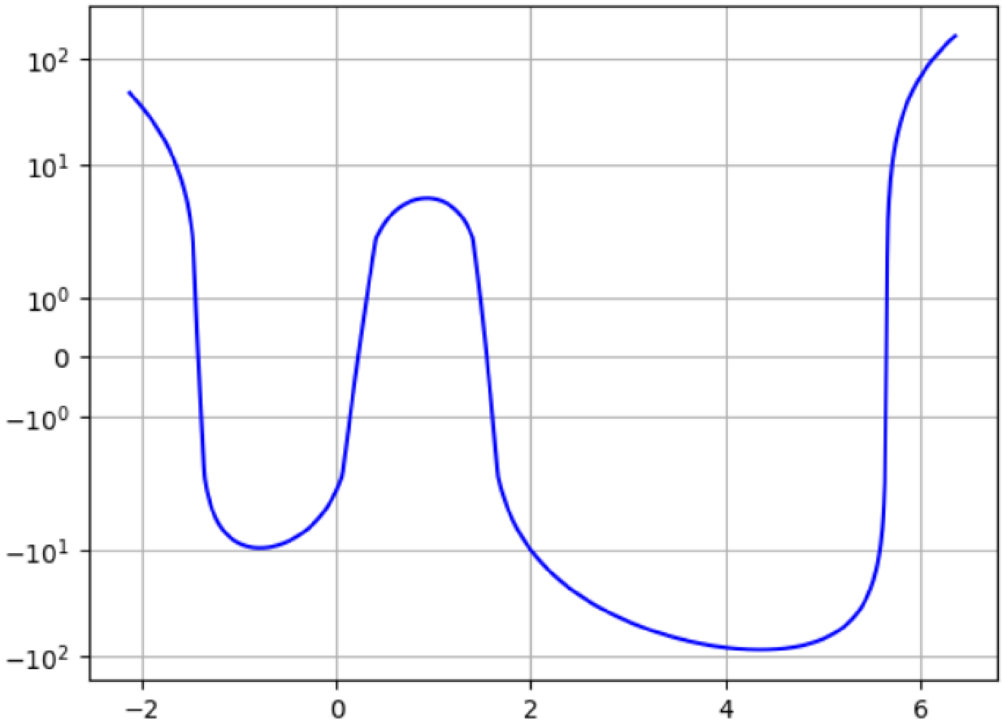
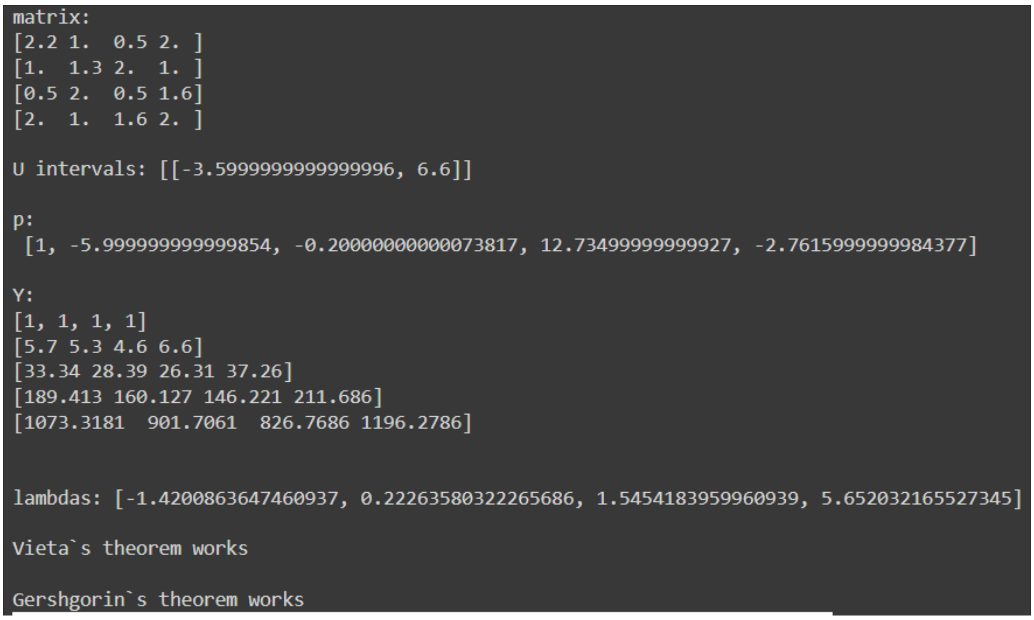


Рис. 1 — График для матрицы размерностью 4x4 для метода Крылова



```
matrix:
[2.2 1.  0.5 2. ]
[1.  1.3 2.  1. ]
[0.5 2.  0.5 1.6]
[2.  1.  1.6 2. ]

U intervals: [[-3.5999999999999996, 6.6]]

p:
 [1, -5.999999999999854, -0.20000000000073817, 12.73499999999927, -2.7615999999984377]

Y:
[1, 1, 1, 1]
[5.7 5.3 4.6 6.6]
[33.34 28.39 26.31 37.26]
[189.413 160.127 146.221 211.686]
[1073.3181  901.7061  826.7686 1196.2786]


lambdas: [-1.4200863647460937, 0.22263580322265686, 1.5454183959960939, 5.652032165527345]

Vieta`s theorem works

Gershgorin`s theorem works
```

Рис. 2 — Поиск собственных значений для матрицы размерности 4x4 для метода Крылова

```
eigin vectors:
 x_1: [-0.99551251  2.31304961 -3.39518518  1.49419607]
 x_2: [ 0.72817136  0.63462338 -0.21408541 -0.9837196 ]
 x_3: [ 2.3043117  -2.09783242 -1.77936803  0.73957869]
 x_4: [165.95001604 139.25314698 127.58762785 184.90892928]

eigin orthonormal vectors:
 x_1: [-0.22204259  0.51591068 -0.75727398  0.33327072]
 x_2: [ 0.52192064  0.45486964 -0.15344684 -0.70508618]
 x_3: [ 0.62892982 -0.57257417 -0.48565375  0.20185771]
 x_4: [0.53173607 0.44619412 0.40881553 0.59248411]

Eigen vectors are orthogonal
```

Рис. 3 — Собственные вектора для матрицы размерности 4x4 для метода Крылова

```
eigin vectors:
 x_1: [-0.22204311  0.51591039 -0.75727407  0.33327063]
 x_2: [-0.52192073 -0.45486916  0.15344709  0.70508637]
 x_3: [ 0.62892977 -0.57257421 -0.4856538   0.20185761]
 x_4: [0.53173694 0.44619371 0.40881477 0.59248416]

Eigen vectors are orthogonal
```

Рис. 4 — Собственные вектора (уже ортонормированные) для матрицы размерности 4x4 для метода Данилевского

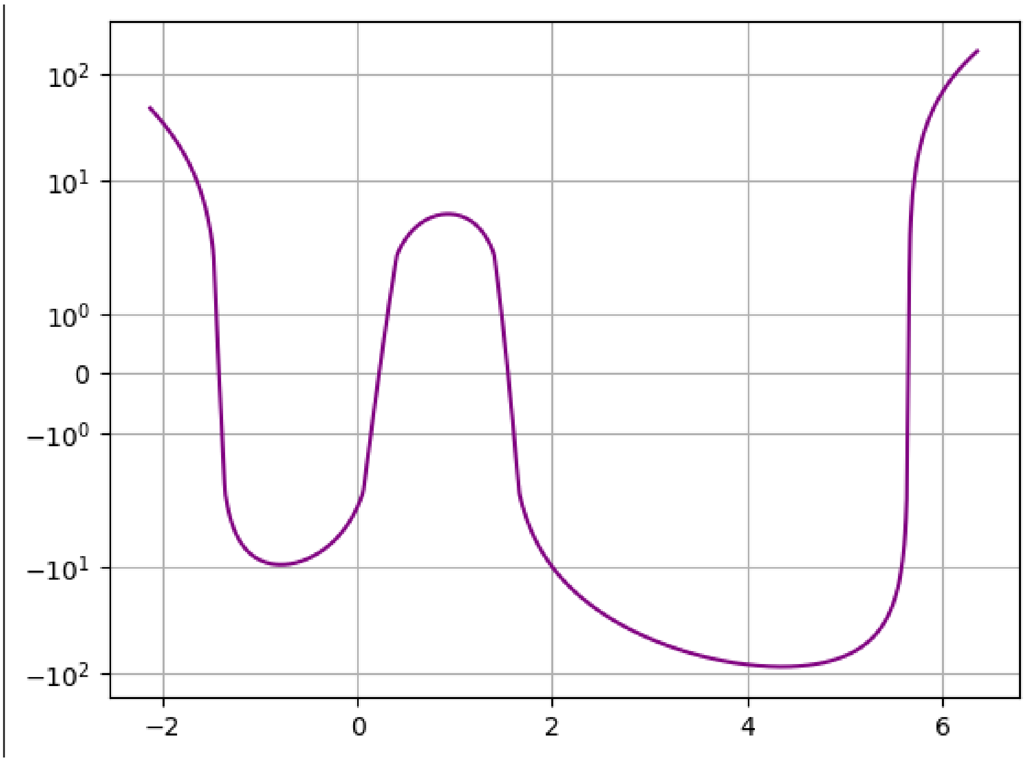


Рис. 5 — График для матрицы размерности 4х4 для метода Данилевского

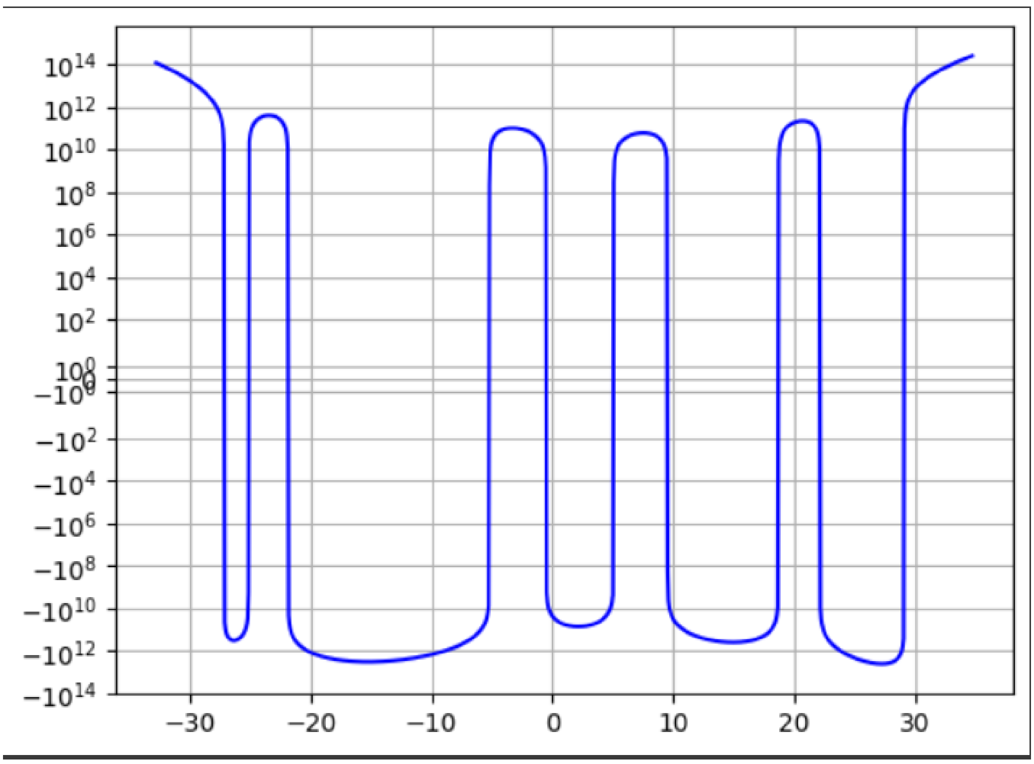


Рис. 6 — График для матрицы размерности 10х10 для метода Крылова

5 Выводы

В результте выполнения данной лабораторной работы был реализован алгоритм, позволяющий анализировать матрицы с использованием метода Крылова, вычислять интервалы Гершгорина и находить их собственные значения и собственные вектора, а также визуализировать характеристическое уравнение для действительных квадратных матриц произвольной размерности n на языке программирования Python.

Результаом работы является успешное нахождение собственных значений и векторов матрицы, что подтверждает корректность алгоритмов. Также была проверена теорема Виета для собственных значений и ортогональность собственных векторов. Графическое представление характеристического уравнения помогло в визуализации результатов.

Сравнение методов А.М. Данилевского и А.Н. Крылова показало, что оба метода нахождения собственных значений и векторов симметричной матрицы дают правильные результаты.