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 № 6

по курсу «Численные методы линейной алгебры»

«Изучение скорости сходимости однопараметрического метода»

Студент группы ИУ9-71Б Яровикова А. С.

Преподаватель Посевин Д. П.

Москва 2023

1 Цель работы

Изучить зависимость скорости сходимости однопараметрического метода в зависимости от значения τ.

2 Задание

1. Реализовать однопараметрический метод для положительной симметричной матрицы произвольного размера $N \times N$.

2. Вычислить спектр матрицы A методом Крылова или Данилевского, которые были реализованы ранее, и получить минимальное и максимальное значение спектра λ_{min} и λ_{max}. После чего вычислить $\tau_{opt} 2/(\lambda_{min} + \lambda_{max})$.

3. Построить график зависимости количества итераций n решения уравнения $Ax = f$ однопараметрическим методом в зависимости от значения τ лежащего в пределах от 0 до $2/\lambda_{max}$. Определить τ_{opt} из графика и сравнить с теоретическим значением полученным в пункте 2.

4. Для каждого эксперимента пункта 3 вывести условие сходимости посчитанное по формуле:

$$\|r^{(k)}\|_2^2 \leq \max_i |\mu_i|^2 \cdot \|r^{(k-1)}\|_2^2$$

Решение $x*$ системы уравнения $Ax = f$ для оценки неравенства приведенного выше можно получить путем решения $Ax = f$ методом Гаусса. Другими словами требуется убедиться в том, что выполняются условия теоремы о сходимости однопараметрического метода. Обязательно, дополнительно проверить и показать, что для каждого k модуль максимального значения λ_i меньше 1.

3 Реализация

Исходный код программы представлен в листингах 1– 6.

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 vec_norm(v):
  s = np.sum(v ** 2)
  return np.sqrt(s)

def generate_vec(l, r, n):
  vec = np.random.uniform(l, r, n)
  return vec

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

def increase_diag_elems(a, diag):
    n = len(a)
    for i in range(0, len(a)):
        a[i][i] = diag * sum(abs(a[i][j]) if j != i else 0 for j in
    range(n))
    return a

def calc_diagonal_dominance(a):
  degree = max(abs(a[i][i]) - sum(abs(a[i][j]) if j != i else 0 for j in
     range(len(a))) for i in range(len(a)))
  return degree > 0
```

Листинг 2 — Метод Крылова

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

  p_vec = np.linalg.solve(A, f)
  p.append(1)
  for pp in p_vec:
    p.append(-pp)
  return y, p
```

Листинг 3 — Круги 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])
  intervals = [ (centers[i] - rads[i], centers[i] + rads[i]) for i in
   range(n)]
  intervals = union_intervals(deepcopy(intervals))
  return intervals
```

Листинг 4 — Поиск собственных значений

```
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):
  values = []
  len2 = 10e-7
  for interval in intervals:
    left = interval[0]
    right = interval[1]
    l = int(np.floor((right - left) / len))
    for i in range(l):
      x_left = left + i * len
      x_right = x_left + len
      y_left = polynomy(eqCoeffs, x_left)
      y_right = polynomy(eqCoeffs, x_right)
      alpha = y_left * y_right
      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 — Однопараметрический метод

```
def one_param_method(A, f, x_correct, t, eigenvals):
  eps = 1e-4
  n = len(A)
  iters = 0

  P = (np.eye(n) - t * A)
  g = t * f

  m = []
  for lamb in eigenvals:
    m.append(abs(1 - float(t) * lamb))

  abs_mu_max = max(m)

  x = g
  r = x - x_correct
  while True:
    x_i = P @ x + g
    r_i = P @ r
    iters += 1
    print(f'iteration: {iters}')
    if (vec_norm(r_i) ** 2 <= abs_mu_max ** 2 * vec_norm(r) ** 2 + eps):
      print("the convergence condition is met")
    if (abs_mu_max < 1):
      print("max|mu_i| < 1")

    if vec_norm(x - x_i) < eps:
      break

    x = x_i
    r = r_i

  return x, iters
```

Листинг 6 — Запуск программы

```
n = 5
a = generate_symmetrical_matrix(1, 10, n)
a = increase_diag_elems(a, n)
print('matrix a:')
print(a)
f = generate_vec(1, 10, n)
print(f'\nvector f: {f}')

diag_cond = calc_diagonal_dominance(a)
print(f'\ncheck for diagonal dominance condition: {diag_cond}')

intervals = find_gershgorin_intervals(a)
Ys, P = krylov_algo(a)
vals = find_eigen_values(P, intervals, 10e-3)
print(f'\neigenvalues: {vals}')

eigenmin, eigenmax = min(vals), max(vals)
t_optimal = 2 / (eigenmin + eigenmax)
print(f'\nt_opt: {t_optimal}')
t_left = 0.001
t_right = 2 / eigenmax
print(f'\nt in [{t_left} , {t_right}]')

# correct res
x_correct = np.dot(np.linalg.inv(a), f)
# one param method res
x_, iters = one_param_method(a, f, x_correct, t_optimal, vals)

print(f'\ncorrect res: {x_correct}')
print(f'method res: {x_}, iters: {iters}')

plt.figure(figsize=(15,8))
plt.xlabel('tau')
plt.ylabel('iters')
taus = np.arange(t_left, t_right, 0.00001)
iters = [one_param_method(a, f, x_correct, tau, vals)[1] for tau in taus
    ]
plt.plot(taus, iters)
plt.ylim(0, np.mean(iters))
tau_it = taus[np.argmin(iters)]
plt.legend()
plt.grid()
plt.show()

print(f'\nt_opt on graphic: {tau_it}')
print(f't_opt: {t_optimal}')
```

4 Результаты

Результат запуска методов представлены на рисунках 1 - 3.

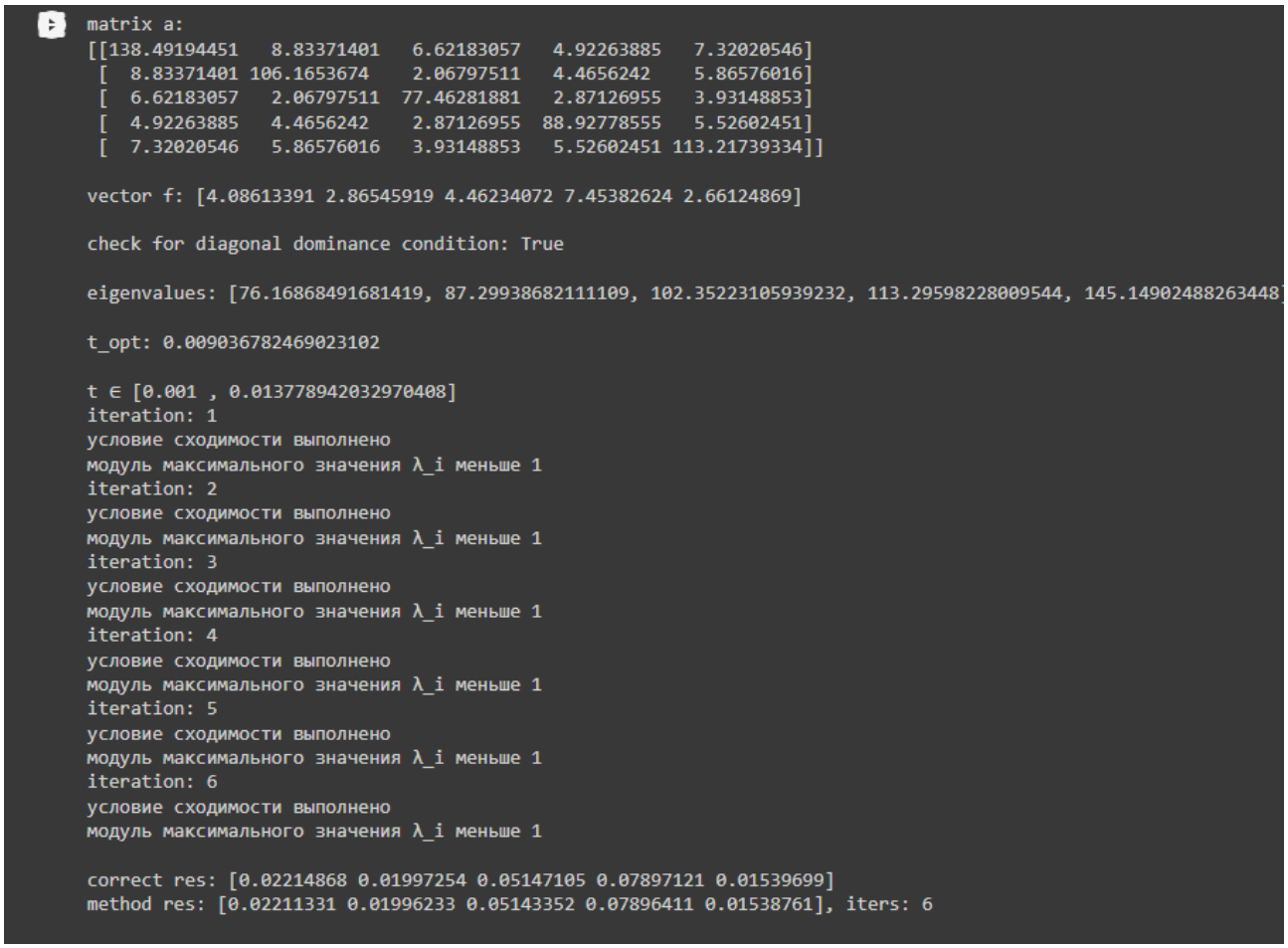

Рис. 1 — Тестовые данные СЛАУ: матрица A размером 5x5, вектор f. Выполнение алгоритма для τ_{opt}

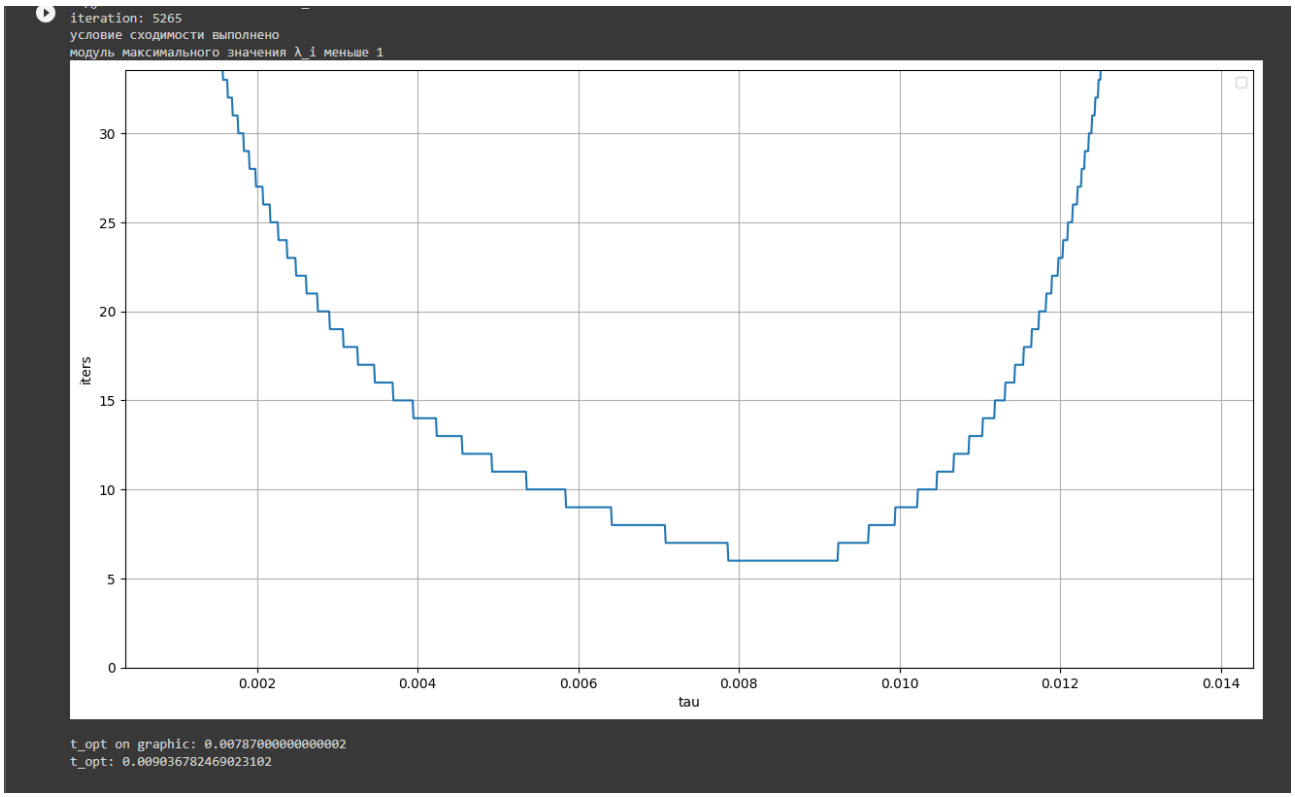

Рис. 2 — График зависимости количества итераций $iters$ решения уравнения $Ax = f$ с матрицей 5x5 однопараметрическим методом в зависимости от значения $\tau \in (0, 2/\lambda_{max})$. Сравнение τ_{opt} из графика с теоретическим значением τ_{opt}

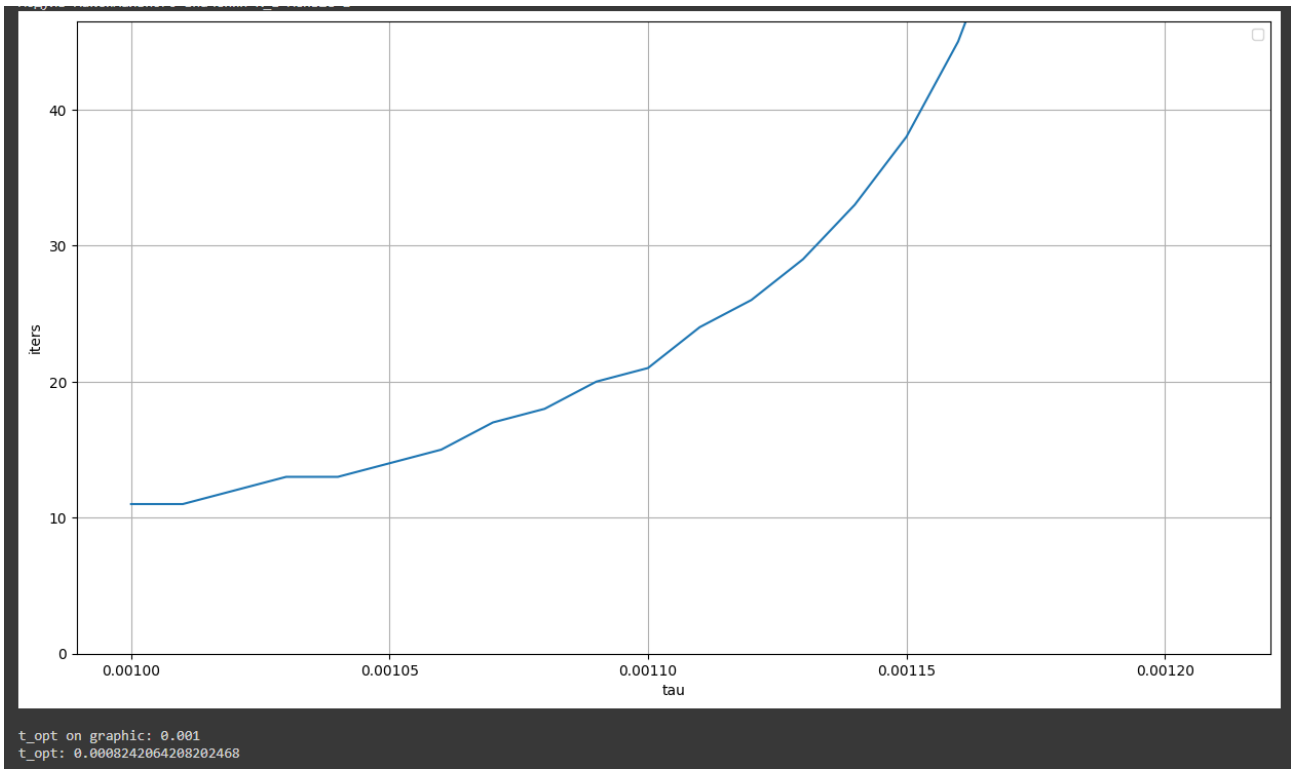

Рис. 3 — График зависимости количества итераций $iters$ решения уравнения $Ax = f$ с матрицей 15x15 однопараметрическим методом в зависимости от значения $\tau \in (0, 2/\lambda_{max})$. Сравнение τ_{opt} из графика с теоретическим значением τ_{opt}

5 Выводы

В результте выполнения данной лабораторной работы был реализован однопараметрический метод для решения СЛАУ с размером матрицы коэффициентов NxN. Реализация была выполнена на языке программирования Python. Также выполнено сравнение теоритического оптимального значения τ_{opt} и значения τ_{opt} по графику зависимости числа итераций от значения τ. Результаты сравнения показали, что полученные значения отличаются незначительно.