Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт

По лабораторной работе №4

По дисциплине: прикладная математика

Факультет: ИТиП

Группа: M32001

Авторы:

Андреев Артём Русланович
Слюсаренко Сергей Владимирович
Шевченко Валерий Владимирович

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2022

Реализация алгоритмов:
https://github.com/JabaJabila/ITMO_AppliedMath/tree/main/Lab4

Метод вращений Якоби:

Метод Якобы – итерационный метод, суть которого заключается в том, чтобы переходить по такому преобразованию подобия:

$A_{k+1} = U_{(k)T}\, A_k\, U_k$

Причем U^k – матрица поворота, которая подбирается таким образом, что [i, j] элемент обращается в 0, где i, j координаты максимального не диагонального элемента матрицы A^k

$$
U^k = \begin{array}{c}
\begin{array}{ccccccccc} & & & i & & & j & & \end{array} \\
\left(\begin{array}{ccccccccc}
1 & & & \vdots & & & \vdots & & \\
 & \ddots & & \vdots & & & \vdots & 0 & \\
 & & 1 & \vdots & & & \vdots & & \\
\cdots & \cdots & \cdots & \cos\varphi^{(k)} & \cdots & \cdots & -\sin\varphi^{(k)} & \cdots & \cdots \\
 & & & \vdots & 1 & & \vdots & & \\
 & & & \vdots & & \ddots & \vdots & & \\
 & & & \vdots & & 1 & \vdots & & \\
\cdots & \cdots & \cdots & \sin\varphi^{(k)} & \cdots & \cdots & \cos\varphi^{(k)} & \cdots & \cdots \\
 & & & \vdots & & & \vdots & 1 & \\
 & 0 & & \vdots & & & \vdots & & \ddots \\
 & & & \vdots & & & \vdots & & 1
\end{array}\right)
\end{array}
\begin{array}{c} \\ \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \\ \end{array},
$$

Таким образом на каждой след итерации норма над диагональю будет уменьшаться, так матрица со временем обратится в диагональную матрицу (близкую к диагональной) (а максимальный элемент мы берем, чтобы было как можно меньше итераций). Полученная матрица будет матрицей с собственными значениями на диагонали. А результатом перемножения всех U будет матрица, содержащая собственные вектора, т. е. матрица преобразования

Проверка работы:

```
Матрица 1:
[[17 -2 -2]
 [-2 14 -4]
 [-2 -4 14]]

Собственные числа:
[17.99999572694248, 9.000004273057558, 18.0]

Собственные векторы:
[[ 0.9430385   0.33268362  0.        ]
 [-0.23524284  0.66682892 -0.70710678]
 [-0.23524284  0.66682892  0.70710678]]

Кол-во итераций:
372

Матрица A * собственный вектор №1: [16.97262586 -4.2385054  -4.2385054 ]
Собственное число №1 * собственный вектор №1: [16.97468897 -4.23437012 -4.23437012]

Матрица A * собственный вектор №2: [2.98830579 6.00292195 6.00292195]
Собственное число №2 * собственный вектор №2: [2.99415396 6.00146311 6.00146311]

Матрица A * собственный вектор №3: [  0.         -12.72792206  12.72792206]
Собственное число №3 * собственный вектор №3 [  0.         -12.72792206  12.72792206]
```

```
Матрица 2:
[[5 1 2]
 [1 4 1]
 [2 1 3]]

Собственные числа:
[1.7091019948414063, 3.3975863993390933, 6.893311605819496]

Собственные векторы:
[[ 0.48680054 -0.42647275  0.76232948]
 [ 0.14419043  0.89997661  0.41140154]
 [-0.86153024 -0.09034988  0.49960239]]

Кол-во итераций:
274

Матрица B * собственный вектор №1: [ 0.85513264  0.20203201 -1.46679922]
Собственное число №1 * собственный вектор №1: [ 0.83199177  0.24643615 -1.47244305]

Матрица B * собственный вектор №2: [-1.4130869   3.0830838  -0.22401853]
Собственное число №2 * собственный вектор №2: [-1.44897801  3.05774828 -0.30697152]

Матрица B * собственный вектор №3: [5.22225369 2.90753802 3.43486765]
Собственное число №3 * собственный вектор №3: [5.25497462 2.83591901 3.44391492]
```

Сравнение и анализ:

Матрицы с диагональным преобладанием и матрицы Гильберта с точностью eps = 0.1:

	size n	epsilon	cond(A)	iterations
0	3	0.1	2.727768e+04	466
1	5	0.1	4.259110e+06	3086
2	7	0.1	9.813256e+08	10957
3	9	0.1	1.316968e+11	68888
4	11	0.1	1.936197e+13	95480
5	13	0.1	2.910489e+15	197095
6	15	0.1	1.478949e+17	220798

	size n	epsilon	cond(A)	iterations
0	3	0.1	5.261588e+02	13
1	5	0.1	4.808491e+05	316
2	7	0.1	4.817473e+08	211
3	9	0.1	5.017303e+11	535
4	11	0.1	5.343571e+14	512
5	13	0.1	3.382229e+17	562
6	15	0.1	4.684591e+17	1166

Матрицы с диагональным преобладанием и матрицы Гильберта с точностью eps = 0.01:

	size n	epsilon	cond(A)	iterations
0	2	0.01	2.001000e+03	1
1	3	0.01	1.523286e+04	163
2	4	0.01	3.546846e+05	25260
3	5	0.01	4.413617e+06	32241
4	6	0.01	7.051241e+07	112255
5	7	0.01	8.878063e+08	217189
6	8	0.01	1.109414e+10	276248
7	9	0.01	1.409757e+11	376620
8	10	0.01	1.616522e+12	601036

	size n	epsilon	cond(A)	iterations
0	2	0.01	1.933333e+01	160
1	3	0.01	5.261588e+02	1295
2	4	0.01	1.561379e+04	880
3	5	0.01	4.808491e+05	3605
4	6	0.01	1.511899e+07	2690
5	7	0.01	4.817473e+08	4157
6	8	0.01	1.549362e+10	2262
7	9	0.01	5.017303e+11	5641
8	10	0.01	1.633364e+13	5263

Выводы:

Итерационный метод поиска собственных чисел и векторов Якоби работает довольно медленно на относительно небольших матрицах, если требуемая точность высока, однако позволяет решать полную проблему собственных значений и собственных векторов даже при высоком числе обусловленности матрицы. Сравнивая работу алгоритмов при поиске собственных значений и векторов матриц с диагональным преобладанием и матриц Гильберта видно, что при «диагональных» матрицах метод Якоби на небольших размерах справляется быстрее, но с увеличением размера матрицы, сложность работы метода Якоби на матрицах Гильберта возрастает гораздо медленнее, чем на «диагональных» матрицах.

Приложение

Методы генерации тестовых матриц:

```
import numpy as np import
random
    def
generate_gilbert_matrix(k):
matrix = np.zeros((k, k))
for i in range(k):          for
j in range(k):
            matrix[i][j] = 1 / (i + j + 1)

    return np.array(matrix)
  def
generate_diagonal_matrix(k):
    values = [0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
noise = 10 ** (-k)      matrix = np.zeros((k, k))      for
i in range(k):          for j in range(i, k):
            matrix[i][j] = matrix[j][i] = random.choice(values)
      for i in
range(k):
        matrix[i][i] = -(sum(matrix[i]) - matrix[i][i]) + noise

    return np.array(matrix)
  def
generate_random_symmetric(k):
    matrix = np.zeros((k, k))
for i in range(k):
for j in range(i, k):
            matrix[i][j] = matrix[j][i] = random.random() * random.randint(-10,
10)
    return np.array(matrix)
```

Поиск числа обусловленности матрицы:

```
import numpy as np
  def
get_number(matrix):
    return np.linalg.norm(matrix) * np.linalg.norm(np.linalg.inv(matrix))
```

Метод вращений Якоби:

```
import numpy as np import
math
from scipy import sparse
    def find_max(A):
n = len(A)      max_i,
max_j = 0, 1;      max =
A[0, 1]      for i in
range(n):
        for j in range(i + 1, n):
if (abs(A[i, j]) > abs(max)):
```

```
max_i = i                    max_j = j
max = A[i, j]

    return max_i, max_j
    def find_rotation(A, i,
j):
    k = len(A)      if (A[i,
i] == A[j, j]):
return math.pi / 4

    return math.atan(2.0 * A[i, j] / (A[j, j] - A[i, i])) / 2.0
    def stop_criteria(A,
eps):
    n = len(A)      norma = 0
for i in range(n):          for j
in range(i + 1, n):
norma += A[i, j] ** 2

    return math.sqrt(norma) < eps
    def jacobi_solve(matrix: np.array,
eps):
    n = len(matrix)
iters = 0
    vectors = np.eye(n, n);
      if (n <=
1):
        return matrix, matrix, iters
      while (not stop_criteria(matrix, eps)):
max_i, max_j = find_max(matrix)          deg =
find_rotation(matrix, max_i, max_j)
rotate = np.eye(n, n);

        rotate[max_i, max_i] = rotate[max_j, max_j] = math.cos(deg)
rotate[max_i, max_j] = -math.sin(deg)         rotate[max_j, max_i]
= -rotate[max_i, max_j]

        rotateT = rotate.transpose()

        matrix = np.dot(np.dot(rotateT, matrix), rotate)
vectors = np.dot(vectors, rotate)

        iters += 1

    answer = [matrix[i, i] for i in range(n)]

    return answer, vectors, iters
```