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 **«Фундаментальные науки»**

КАФЕДРА **«Прикладная математика»**

ДОМАШНЕЕ ЗАДАНИЕ ПО ПРЕДМЕТУ:

МЕТОДЫ ЧИСЛЕННОГО РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Вариант №3

Выполнил:
студент группы ФН2-32М
Матвеев Михаил

Проверил:
Родин А. С.

Москва, 2020

Содержание

1 Постановка домашнего задания

Нужно сформировать матрицу размером 10x10 по следующему принципу. В качестве базовой матрицы берется известная матрица, которая получается после дискретизации одномерного оператора Лапласа методом конечных разностей или методом конечных элементов на равномерной сетке:

$$A_0 = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & \cdots & \cdots & \cdots & 0 \\ & & & & \ddots & & & & \\ 0 & \cdots & \cdots & \cdots & \cdots & -1 & 2 & -1 & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 2 & -1 \end{pmatrix}$$

Для данной матрицы известны аналитические формулы для собственных значений (n=10):

$$\lambda_j^0 = 2(1 - \cos(\frac{\pi j}{n+1})), \quad j = 1, \ldots, n \tag{1}$$

и компонент собственных векторов (вектора имеют 2-норму равную 1):

$$z_j^0(k) = \sqrt{\frac{2}{n+1}} \sin(\frac{\pi j k}{n+1}), \quad k = 1, \ldots, n \tag{2}$$

Итоговая матрица получается по формулам:

$$A = A_0 + \delta A,$$

$$\delta A_{ij} = \begin{cases} \dfrac{c}{i+j}, & i \neq j \\ 0 & i = j \end{cases},$$

$$c = \frac{N_{var}}{N_{var}+1} \varepsilon,$$

где N_{var} - номер варианта (совпадает с номером студента в списке в журнале группы), ε - параметр, значение которого задаётся далее.

2 Задача №1

2.1 Постановка задачи

Взять матрицу A для значения $\varepsilon = 0.1$, убрать последний столбец и сформировать из первых 9 столбцов матрицу $\hat{A}$ размера 10x9. Решить линейную задачу наименьших квадратов для вектора невязки

$$r = \hat{A}x - b,$$

где вектор b размерности 10x1 нужно получить по следующему алгоритму: выбрать вектор x_0 размерности 9x1 и для него вычислить $b = \hat{A}x_0$.

Для решения поставленной задачи использовать QR разложение: для вариантов с четным номером использовать соответствующий алгоритм, основанный на методе вращений Гивенса, для вариантов с нечетным номером – алгоритм, основанный на методе отражений Хаусхолдера. После получения решения сделать оценку величины $\dfrac{\|x - x_0\|_2}{\|x_0\|_2}$.

2.2 Применяемые методы

2.2.1 Преобразования Хаусхолдера

Преобразованием Хаусхолдера (или отражением) называется матрица вида

$$P = I - 2uu^T, \tag{3}$$

где вектор u называется вектором Хаусхолдера, а его норма $\|u\|_2 = 1$. Матрица P симметрична и ортогональна, она называется отражением, потому что вектор Px является отражением вектора x относительно плоскости, проходящей через 0 перпендикулярно к u.

Пусть дан вектор x. Тогда легко найти отражение $P = I - 2uu^T$, аннулирующее в векторе x все компоненты, кроме первой: $Px = [c, 0, \ldots, 0]^T = c \cdot e_1$. Это можно сделать следующим образом. Имеем $Px = x - 2u(u^T x) = c \cdot e_1$, поэтому $u = \dfrac{1}{2(u^T x)}(x - ce_1)$, т.е. u есть линейная комбинация векторов x и e_1. Так как $\|x\|_2 = \|Px\|_2 = |c|$, то u должен быть параллелен вектору $\tilde{u} = x \pm \|x\|_2 e_1$, откуда $u = \dfrac{\tilde{u}}{\|\tilde{u}\|_2}$. Воспользуемся формулой $\tilde{u} = x + sgn(x_1)e_1$, так как в этом случае не будет взаимного сокращения при вычислении первой компоненты в $\tilde{u}$. Итак, имеем

$$\tilde{u} = \begin{bmatrix} x_1 + sgn(x_1) \cdot \|x\|_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad u = \frac{\tilde{u}}{\|\tilde{u}\|_2}$$

Мы будем записывать данное преобразование как $u = house(x)$. После вычисления вектора Хаусхолдера получим отражение Хаусхолдера по формуле (3). Далее умножаем матрицу, которую мы и хотим разложить на матрицы Q и R, на матрицу P слева. Данное действие аннулирует поддиагональные элементы матрицы A. Приведём общий алгоритм QR - разложения, основанный на использовании отражений.

```
input  : Матрица A[m x n]
output: Матрицы R и Q
m, n = A.shape;
Q = np.identity(m);
R = np.copy(A);
for i in range(min(m, n)): u = house(A[i:, i]);
P = I - 2uu^T;
A = P @ A;
Q = Q @ P;
```

Обсудим некоторые детали реализации метода. Для хранения матрицы P_i достаточно запомнить лишь вектор u_i. Эта информация может храниться в столбце i матрицы A_i. Таким образом, QR - разложение может быть записано на место матрицы A, причём P_i хранится в виде вектора u_i в поддиагональных позициях столбца i матрицы A. Так как диагональные позиции заняты элементами R_{ii}, нужно создать дополнительный массив для хранения первых элементов векторов u_i.

Так как мы решаем задачу наименьших квадратов $min \|Ax - b\|_2$ с помощью разложения $A = QR$, решение мы получаем, решая систему уравнений

$$x = R^{-1}Q^T b.$$

В случае неявного QR-разложения (когда Q хранится в факторизованной форме $P_1, \cdots, P_{n-1}$, а P_i хранится в виде вектора u_i в поддиагональных позициях столбца i матрицы A) нужно вычислить вектор $Q^T b$. Это делается следующим образом $Q^T b = P_n P_{n-1} \cdots P_1 b$, поэтому b нужно последовательно умножать на $P_1, P_2, \cdots, P_n$:

Описание алгоритмов

Обычный метод for i in range(n): (цикл по строкам)

1. подготавливаем матрицу P_i (единичная матрица размера m x m, где m - количество строк);
2. в начале каждой итерации выбираем столбец матрицы A (не весь столбец, а элемент на диагонали и элементы под диагональю) в качестве вектора x;
3. применяем к вектору x метод house для получения вектора Хаусхолдера размера m - i;
4. вычитаем из матрицы A внешнее произведение двух векторов для получения матрицы P' размера m - i x m - i;
5. добавляем матрицу P' в матрицу P_i;
6. перемножаем матрицы P_i и R для получения матрицы A_i, у которой обнуляются поддиагональные элементы;
7. домножаем матрицу Q на матрицу P_i (для получения в конце итоговой матрицы Q);

end for

Эффективный метод for i in range(n): (цикл по строкам)

1. в начале каждой итерации выбираем столбец матрицы A (не весь столбец, а элемент на диагонали и элементы под диагональю) в качестве вектора x;
2. применяем к вектору x метод house для получения вектора Хаусхолдера размера m - i;
3. вычитаем из матрицы A внешнее произведение двух векторов;
4. записываем вектор u в поддиагональные элементы матрицы A кроме первого элемента вектора u;
5. записываем первый элемент вектора u в отдельный вектор;

end for

```python
import numpy as np
from math import sqrt, hypot

def house(x):
    u_tilde = np.copy(x)
    u_tilde[0] += np.sign(x[0]) * np.linalg.norm(x)
    return u_tilde / np.linalg.norm(u_tilde)

def qr_householder(A):
    m, n = A.shape
    Q = np.identity(m)
    R = np.copy(A)
    for i in range(n):
        P_i = np.identity(m)
        x = R[i:, i]
        u = house(x)
        P_streak = np.identity(m - i) - 2 * np.outer(u, u)
        P_i[i:, i:] = np.copy(P_streak)
        R = P_i @ R
        Q = Q @ P_i
    return Q[:, :n], R[:n, :]

def qr_householder_effective(A):
    curr_A = np.copy(A)
    list_first_elements = []
    m, n = A.shape
    for i in range(n):
        x = curr_A[i:, i]
        u = house(x)
        curr_A[i:m, i:n] -= 2 * np.outer(u, np.matmul(u, curr_A[i:m, i:n]))
        curr_A[i + 1:m, i] = u[1:]
        list_first_elements.append(u[0])
    return curr_A, list_first_elements
```

1. np.identity - создание двумерного массива, у которого элементы на главной диагонали единицы, остальные элементы - нули;

2. np.outer - внешнее произведение двух векторов;

3. np.linalg.norm - евклидова норма вектора;

2.3 Результаты расчётов

Был взят случайный вектор

$$x_0 = [0.6401, 0.2454, 0.5507, 0.3099, 0.5663, 0.7639, 0.9395, 0.1872, 0.2075]$$

Полученное решение совпадает с начальным условием, величина относительной погрешности $\dfrac{\|x - x_0\|_2}{\|x_0\|_2} = 5.0357e^{-16}$.

2.4 Код решения

```python
def construct_delta_A(N_var, eps):
    par = 10
    delta_A = []
    c = N_var * eps / (N_var + 1)
    for i in range(par):
        temp = []
        for j in range(par):
            temp.append(0 if i == j else c / (i + j))
        delta_A.append(temp)

    delta_A = construct_delta_A(N_var, eps)
    A = A_0 + delta_A

    amnt_deletable_cols = 1
    list_deletable_cols = [9 - i for i in range(amnt_deletable_cols)]
    A_hat = np.delete(A, list_deletable_cols, axis = 1)
    x_0 = np.random.rand(10 - amnt_deletable_cols)
    b = A_hat @ x_0

    Q, R = qr_householder(A_hat)
    assert ((Q @ R - A_hat) < 1e-6).all(), "QR decomposition is wrong"
    x = np.linalg.inv(R) @ Q.T @ b
    value_estimation = np.linalg.norm(x - x_0) / np.linalg.norm(x_0)

    b_effective = np.copy(b)
```

1. amnt deletable cols - количество удаляемых столбцов

2. assert - проверка на верное QR - разложение

3 Задача №2

3.1 Постановка задачи

Для матрицы A найти все ее собственные значения $(\lambda_j, \quad j = 1, \ldots, 10)$ и собственные вектора (z_j , с 2-нормой равной 1) с помощью неявного QR-алгоритма со сдвигом для трех вариантов: $\varepsilon = 10^{-1}, 10^{-3}, 10^{-6}$.

По итогам расчетов нужно сделать сводную таблицу, в которой указать следующие величины: $\left|\lambda_j - \lambda_j^0\right|$ и $\left\|z_j - z_j^0\right\|$ для $j = 1, \ldots, 10$.

3.2 Применяемые методы

3.2.1 Базовый итерационный QR-алгоритм

Описание алгоритма Пока не будет выполнено условие критерия сходимости:

1. выполняем QR - разложение $A_i = Q_i R_i$;

2. производим умножение $A_{i+1} = R_i Q_i$;

Критерий сходимости - "Пока матрица A не станет достаточно близка к верхней треугольной матрице по своей структуре".

Код алгоритма

```
def basic_qr(A):
    curr_A = np.copy(A)
    amnt_iters = 0
    while True:
        amnt_iters += 1
        Q, R = qr_householder(curr_A)
        curr_A = np.dot(R, Q)
        if np.allclose(curr_A, np.triu(curr_A)) or amnt_iters > 400:
            break
```

3.2.2 Приведение матрицы к форме Хессенберга методом Хаусхолдера

Дана квадратная симметричная матрица размера n x n; for i in range(n - 2):

1. подготавливаем матрицу Q (единичная матрица размера n x n);
2. в начале каждой итерации выбираем столбец матрицы A (не весь столбец, а элементы под первой диагональю) в качестве вектора x;
3. применяем к вектору x метод house для получения вектора Хаусхолдера размера n - i - 2 x n - i - 2;
4. вычитаем из единичной матрицы внешнее произведение двух векторов для получения матрицы P' размера n - i - 2 x n - i - 2;
5. добавляем матрицу P' в матрицу Q;
6. домножаем матрицу $A = QAQ^T$;

end for

Код алгоритма

```
def hessenberg_householder(A):
    m, n = A.shape
    curr_A = np.copy(A)
    for i in range(n - 2):
        Q = np.identity(m)
        x = curr_A[i + 1:, i]
        u = house(x)
        P = np.identity(m - i - 1) - 2 * np.outer(u, u)
        Q[i + 1:, i + 1:] = np.copy(P)
        curr_A = Q @ curr_A @ Q.T
```

3.2.3 Итерационный QR-алгоритм со сдвигом

Описание алгоритма Приводим матрицу к форме Хессенберга до начала основного итерационного процесса;

for i in range(n, 0, -1):

Пока не будет выполнено условие критерия сходимости:

1. в качестве сдвига выбираем самый правый нижний элемент матрицы $\sigma = A_{i,i}$;
2. выполняем QR - разложение $A_i - \sigma E = Q_i R_i$;
3. производим умножение $A_{i+1} = R_i Q_i + \sigma E$;
4. если выполняется условие критерия сходимости, то правый нижний элемент матрицы принимается за собственное значение, сохраняется, а матрица размера A[i, i] лишается самой нижней строки и самого правого столбца;

endfor

Критерий сходимости - "Пока самая нижняя строка матрицы A (за исключением правого элемента) и самый правый столбец (за исключением нижнего элемента) не станут близки к нулю".

Код алгоритма

```
amnt_iters = 0
for i in range(n, 0, -1):
    lc_it = 0
    if curr_A.size == 1:
        list_eigenval.append(curr_A[0, 0])
        break
    while True:
        sigma = curr_A[-1, -1]
        Q, R = qr_householder(curr_A - sigma * np.identity(i))
        curr_A = np.dot(R, Q) + sigma * np.identity(i)
        lc_it += 1
        cond = lambda elem: np.allclose(elem, np.zeros(i - 1))
        if cond(curr_A[-1, :-1]) and cond(curr_A[:-1, -1]) or lc_it > 50:
            amnt_iters += lc_it
            list_eigenval.append(curr_A[-1, -1])
            curr_A = np.copy(curr_A[:-1, :-1])
            break
```

3.2.4 Метод Гивенса для неявного QR - алгоритма со сдвигом

Описание алгоритма for i in range(n - 1):

1. подготавливаем единичную матрицу Q размера n x n;
2. в качестве сдвига выбираем самый правый нижний элемент матрицы $\sigma = A_{i,i}$;
3. вычисляем

$$c = \frac{A_{i,i} - \sigma}{\sqrt{(A_{i,i} - \sigma)^2 + A_{i+1,i}}} \quad s = \frac{A_{i+1,i}}{\sqrt{(A_{i,i} - \sigma)^2 + A_{i+1,i}}};$$

4. вводим матрицу

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

   в матрицу Q на позиции $[i, i] - [i + 1, i + 1]$;

5. домножаем матрицу $A = Q^T AQ$;

endfor

Код алгоритма

```
def implicit_givens(A):
    m, n = A.shape
    curr_A = np.copy(A)
    for i in range(n - 1):
        sigma = curr_A[-1, -1]
        Q = np.identity(m)
        b = hypot(curr_A[i, i] - sigma, curr_A[i + 1, i])
        c = (curr_A[i, i] - sigma) / b
        s = curr_A[i + 1, i] / b
        Q[i, i] = Q[i + 1, i + 1] = c
        Q[i + 1, i], Q[i, i + 1] = s, -s
        curr_A = Q.T @ curr_A @ Q
```

3.2.5 Неявный QR-алгоритм со сдвигом

Описание алгоритма Приводим матрицу к форме Хессенберга до начала основного итерационного процесса;

for i in range(n, 0, -1):

Пока не будет выполнено условие критерия сходимости:

- применяем к матрице Q метод Гивенса, описанный ранее;
- если выполняется условие критерия сходимости, то правый нижний элемент матрицы принимается за собственное значение, сохраняется, а матрица размера $A_{i,i}$ лишается самой нижней строки и самого правого столбца;

endfor

Критерий сходимости - "Пока самая нижняя строка матрицы A (за исключением правого элемента) и самый правый столбец (за исключением нижнего элемента) не станут близки к нулю".

Код алгоритма

```
curr_A = hessenberg_householder(curr_A)
for i in range(n, 0, -1):
    local_iters = 1
    if i == 1:
        list_iters.append(local_iters)
        break
    while True:
        curr_A[:i, :i] = implicit_givens(curr_A[:i, :i])
        if np.allclose(curr_A[:i - 1, i - 1], np.zeros(i - 1), atol = 1e-
            list_iters.append(local_iters)
            break
        local_iters += 1
return np.diag(curr_A), np.array(list_iters)
```

3.3 Результаты расчётов

Были проведены расчёты для трёх параметров $\varepsilon - [1e-1, 1e-3, 1e-6]$, выведем четыре таблицы результатов:

- таблица количества итераций, в которой самая первая строка - самое первое найденное собственное значение, и так далее по порядку;
- отсортированная таблица количества итераций, в которой самая первая строка - минимальное собственное значение, и далее по возрастанию;
- таблица $\left|\lambda_j - \lambda_j^0\right|$, где в качестве λ_j^0 берётся (1);
- таблица $\left\|z_j - z_j^0\right\|$ для $j = 1, \ldots, 10$, где в качестве z_j^0 берётся (2);

Таблица 1: Таблица итераций

	1e-01	1e-03	1e-06
0	6	8	14
1	2	2	1
2	5	3	3
3	3	3	2
4	5	4	2
5	2	3	3
6	2	3	4
7	2	2	2
8	2	2	2
9	1	1	1

Таблица 2: Таблица итераций (отсортированная по возрастанию)

	1e-01	1e-03	1e-06
0	5	3	3
1	2	4	2
2	2	3	3
3	2	2	2
4	1	1	1
5	2	2	2
6	3	3	4
7	5	3	2
8	2	2	1
9	6	8	14

Таблица 3: Таблица абсолютных погрешностей для собственных значений

	1e-01	1e-03	1e-06
0	7.103e-02	7.995e-04	8.003e-07
1	1.961e-02	1.689e-04	1.687e-07
2	1.887e-02	1.503e-04	1.500e-07
3	5.989e-04	1.782e-05	1.792e-08
4	9.211e-03	9.795e-05	9.800e-08
5	2.016e-02	1.985e-04	1.985e-07
6	2.391e-02	2.370e-04	2.370e-07
7	2.411e-02	2.427e-04	2.427e-07
8	1.937e-02	1.986e-04	1.987e-07
9	1.216e-02	1.262e-04	1.262e-07

Таблица 4: Таблица норм разностей для собственных векторов

	1e-01	1e-03	1e-06
0	1.776e-01	1.545e-03	1.543e-06
1	1.762e-01	1.500e-03	1.495e-06
2	1.041e-01	9.317e-04	9.306e-07
3	4.239e-02	3.703e-04	3.780e-07
4	3.038e-02	2.838e-04	2.836e-07
5	3.631e-02	3.567e-04	3.542e-07
6	4.926e-02	4.896e-04	4.896e-07
7	5.655e-02	5.748e-04	5.123e-07
8	5.583e-02	5.841e-04	5.843e-07
9	3.866e-02	4.129e-04	4.245e-07

3.4 Код решения

Код алгоритма

```
for curr_eps in eps:
    temp_eigenvec = []
    A = A_0 + construct_delta_A(N_var, curr_eps)
    curr_eigenvalues, list_iters = implicit_shifting_qr(A)

    diff_eigenvalues = abs(np.sort(curr_eigenvalues) - eigenvalue_0)
    eigenval_results.append([f"{i:.3e}" for i in diff_eigenvalues])

    iters_results_sorted.append(list_iters[np.argsort(curr_eigenvalues)])
    iters_results.append(list_iters)
```

4 Выводы

Были решены поставленные задачи с помощью методов Хаусхолдера и Гивенса, программы были написаны на языке Python. Все алгоритмы кода были предоставлены с кратким описанием действий.

Решение первой задачи показало, что после QR разложения ответ совпадает с изначально заданным x_0, что показано оценкой $\dfrac{\|x - x_0\|_2}{\|x_0\|_2}$.

Решение второй задачи показало, что собственные значения, как и собственные вектора, при уменьшении значения ϵ становятся ближе к известным аналитическим собственным значениям и векторам.