Решение обычной проблемы собственных значений методом Фадеева-Леверье.

Описание метода:

В данной реализации представлен прямой метод решения обычной проблемы собственных значений, а именно метод Фадеева-Леверье. Этот метод подразумевает вычисление коэффициентов характеристического полинома.

$$|\mathbf{A} - \lambda\mathbf{I}| = (-1)^n(\lambda^n - p_1\lambda^{n-1} - \ldots - p_n) \equiv (-1)^n \varphi(\lambda)$$

В основе метода лежит метод Леверье, который был модифицирован Фадеевым. Для вычисления коэффициентов будем вычислять последовательность $\{A_k\}$ следующим образом:

$$
\begin{array}{ll}
A_1 = A, & B_1 = A_1 - q_1 \cdot E \\
A_2 = AB_1, & B_2 = A_2 - q_2 \cdot E \\
\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\
A_{n-1} = AB_{n-2} & B_{n-1} = A_{n-1} - q_{n-1} \cdot E \\
A_n = AB_{n-1} & B_n = A_n - q_n \cdot E,
\end{array}
$$

Соответственно сами коэффициенты будут вычилсяться так:

$$
\begin{array}{l}
SpA_1 = q_1, \\
\dfrac{SpA_2}{2} = q_2, \\
\ldots\ldots\ldots\ldots \\
\dfrac{SpA_{n-1}}{n-1} = q_{n-1} \\
\dfrac{SpA_n}{n} = q_n
\end{array}
$$

Где SpA_k – след матрицы, т. е. сумма элементов диагонали.

Тогда с помощью формул выше можно вычислить все коэффициенты характеристического полинома.

Программная реализация.

```
import numpy as np


def get_tr(a: np.matrix):
    res = 0
    n = a.shape[0]
    for i in range(n):
        res += a[i, i]
    return res


def get_coef(a1: np.matrix):
    a = a1.copy()
    n = a.shape[0]
    res = [1, ]
    id = np.matrix(np.eye(n))
    for i in range(n):
        q = get_tr(a) / (i + 1)
        t = q*id
        b = a - q * id
        a = a1 * b
        res.append(-q)
    res.reverse()
    return res

def check_result(a1: np.matrix, c: list):
    res = np.matrix(np.eye(a1.shape[0]))*c[0]
    a = a1.copy()
    for i in range(1, len(c)):
        res += a*c[i]
        a = a*a1
    return res
```

Проверка правильности решения.

Возьму следующую матрицу:

```
t = np.matrix([[1, 0.17, -0.25, 0.54],
               [0.47, 1, 0.67, -0.32],
               [-0.11, 0.35, 1, -0.74],
               [0.55, 0.43, 0.36, 1]])
```

И посчитаю коэффициенты ее характеристического полинома. Для проверки правильности воспользуюсь теоремой Гамильтона-Кэли. При подстановке исходной матрицы в полином должна получаться нулевая. Буду брать норму получившейся матрицы и выводить коэффициенты.

Результат

```
2.702082265109747e-15
[0.6386380400000002, -3.3826119999999995, 5.7650999999999994, -4.0, 1]

Process finished with exit code 0
```

Сравню полученные коэффициенты с результатом в Wolfram Mathematica

```
a = {{1, 0.17, -0.25, 0.54},
   {0.47, 1, 0.67, -0.32},
   {-0.11, 0.35, 1, -0.74},
   {0.55, 0.43, 0.36, 1}};
Det[a - x IdentityMatrix[4]]
```

$$0.638638 - 3.38261\,x + 5.7651\,x^2 - 4\,x^3 + x^4$$

Как видно, сходится.

Теперь возьму матрицу большего размера и заполню ее случайными числами

```
t = np.matrix(np.random.uniform(-1, 1, (20, 20)))
coef = get_coef(t)
print(np.linalg.norm(check_result(t, coef)))
```

Результат:

```
2.7866037250218227e-05

Process finished with exit code 0
```

Здесь погрешность уже больше, но все равно, результат приемлемый.

Таким образом, реализованный алгоритм работает и дает достаточно точные результаты.