# Задание на 27.02

Кириленко Андрей, ВШЭ

26 февраля 2019

$\boxed{1a}$

Вычислим матрицу Грама для базиса $1,\ x,\ x^2, \ldots, x^{n-1}$ в $L_2[0;1]$:
$A_{ij} =< x^i,\ x^j >= \int_0^1 x^{i+j}dx = \frac{1}{i+j+1}$.
Напишем реализацию:

```
import numpy as np
import matplotlib.pyplot as graph
import math
import time

def getA(n):
    return np.fromfunction(lambda i, j: 1 / (i + j + 1), (n, n))

Result for n=5:
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125      0.11111111]]
```

Листинг 1: Вычисление матрицы Грама

Теперь найдем собственные значения $A$ как функцию от $n$ методом прямых итераций:

```
def forwardIterations(n, iters=1000):
    A = getA(n)
    prev = np.ones(n)
    cur = A @ prev
    res = []
    for i in range(iters):
        prev = cur
        cur = A @ cur
        norm = np.linalg.norm(cur)
        if norm > 1e9:
            cur /= norm
        res.append(np.dot(prev, cur) / np.dot(prev, prev))
    return res
```

Листинг 2: Метод прямых итераций

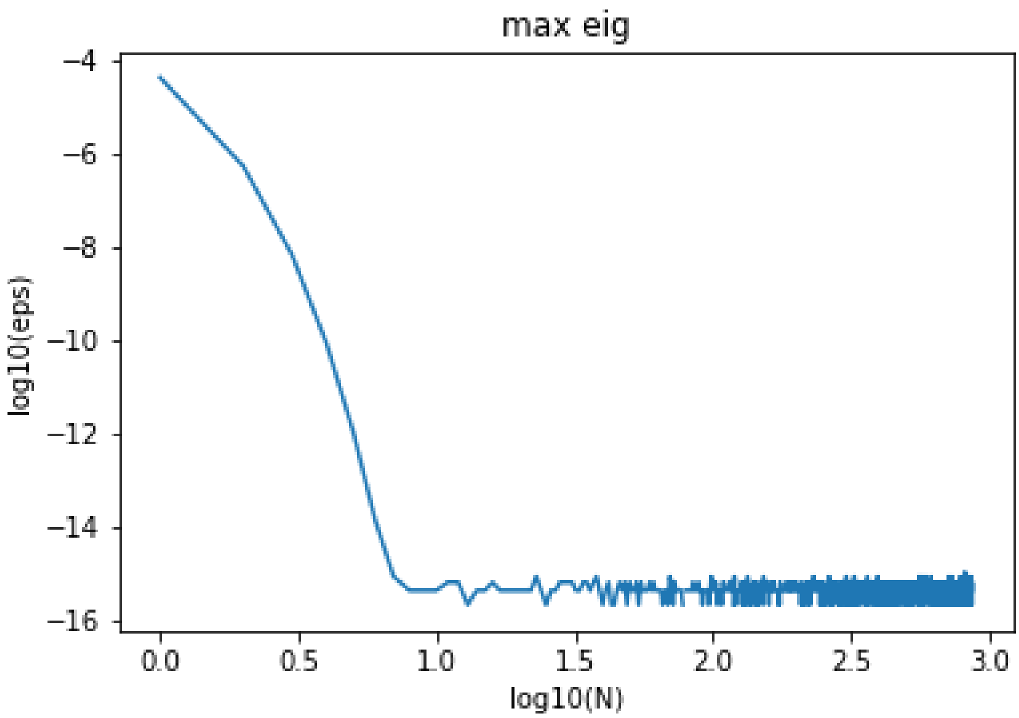
Посмотрим, как быстро мы получим максимально точный ответ. За реальный ответ возьмем библиотечное значение.

```
def convSpeed():
    itersNum = 1000
```

```
    iters = forwardIterations(4, itersNum)
    realResult = max(np.linalg.eig(getA(4))[0])

    x = []
    y = []
    for i in range(itersNum):
        x.append(np.log10(i))
        y.append(np.log10(abs(realResult - iters[i])))

    graph.plot(x, y)
    graph.xlabel('log10(N)')
    graph.ylabel('log10(eps)')
    graph.title('max eig')
    graph.savefig('maxconv.png')
```

Листинг 3: Скорость сходимости

Получим следующий график:



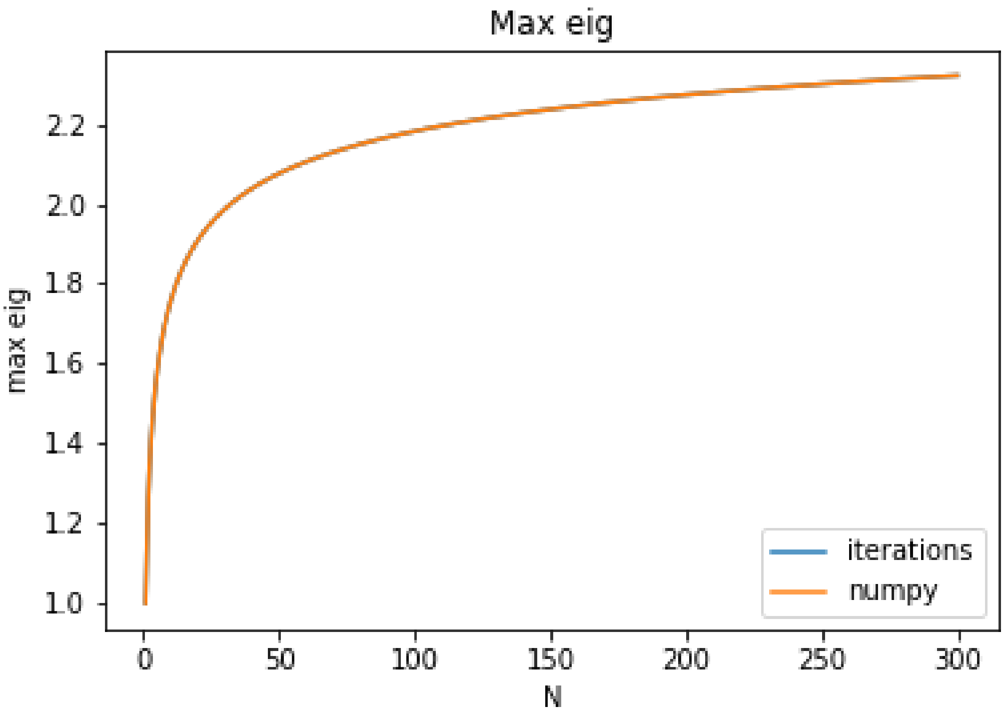

После 10 упираемся в сашинную точность, но сходимость за 10 шагов быстрая. Если быть точным, то сходимость экспоненциальная, что соответствует теоретической оценке $O((\frac{\lambda_{p+1}}{\lambda_1})^k)$

Посмотрим, как зависит максимальное значение с.ч. от $n$.

```
def plotDraft():
    x = []
    y1 = []
    y2 = []
    for i in range(1, 301):
        x.append(i)
        y1.append(forwardIterations(i)[-1])
```

```
        y2.append(max(np.linalg.eigh(getA(i))[0]))
    graph.plot(x, y1, label='iterations')
    graph.plot(x, y2, label='numpy')
    graph.xlabel('N')
    graph.ylabel('max eig')
    graph.title('Max eig')
    graph.legend()
    graph.savefig('draft.png')
    graph.show()
```

Листинг 4: Зависимость максимального собственного значения от размерности матрицы



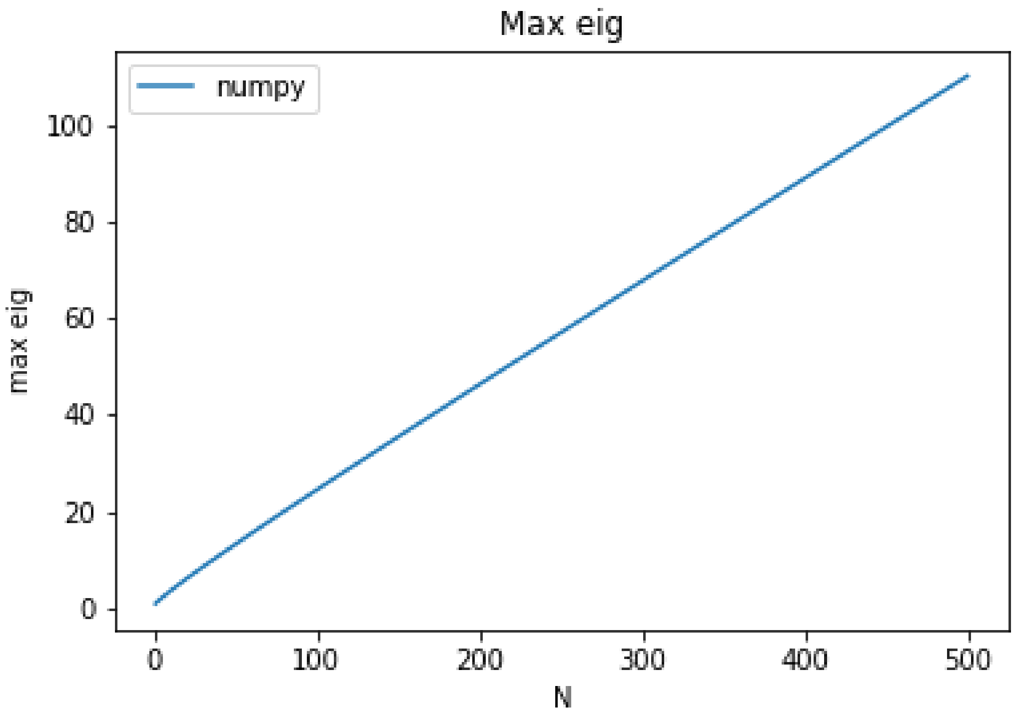

Из такого графика весьма очевидно, что он логарифмический. Однако это не совсем простой логарифм: если повычислять дальше значения собственного числа, то график растет слишком медленно, что позволяет предположить, что основание логарифма также зависит от n. Хочется сказать, что там есть горизонтальная асимптота. Я нашел следующее: `https://msp.org/pjm/2005/219-2/pjm-v219-n2-p09-s.pdf`

На второй странице со ссылкой на статью Taussky утверждается, что асимптотическая оценка радиуса это $\pi$. Тогда, если это предположение верно, то возведя $n$ в степень максимального собственного числа, деленного на $\pi$, мы должны получать прямую. Проверим это

```
def plotDraft():
    x = []
    y2 = []
    for i in range(1, 501):
        x.append(i)
        y2.append(i ** (max(np.linalg.eigh(getA(i))[0]) / np.pi))
    graph.plot(x, y2, label='numpy')
    graph.xlabel('N')
```

```
    graph.ylabel('max eig')
    graph.title('Max eig')
    graph.legend()
    graph.savefig('draft.png')
    graph.show()
```

Листинг 5: Зависимость максимального собственного значения от размерности матрицы проверка гипотезы



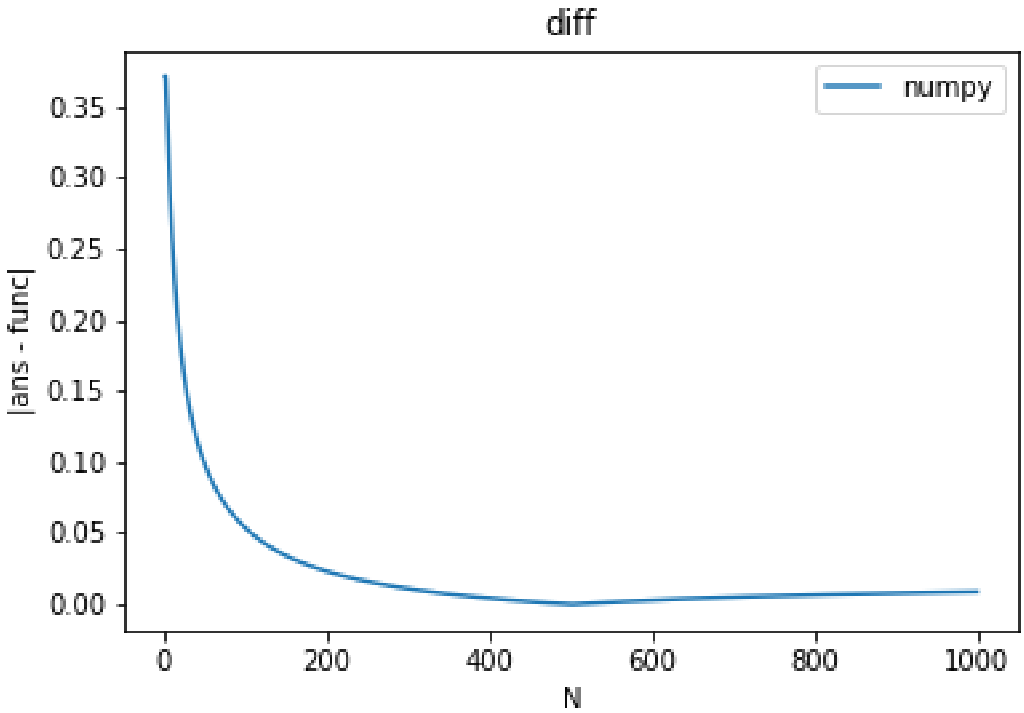

Действительно, получилась прямая. Она проходит через точки (1, 1) и (500, 110.15). Тогда ее уравнение: $y = \frac{109.15}{499}x + \frac{389.85}{499}$.

Из равенства $n^{f(n)/\pi} = \frac{109.15}{499}n + \frac{389.85}{499}$ получаем формулу $f(n) = \pi * log_n(\frac{109.15}{499}n + \frac{389.85}{499})$, что сходится к $\pi$. Построим график отличия формулы от реального значения:

```
def func(n):
    return np.pi * np.log(109.15 * n / 499 + 389.85 / 499) / np.log(n)

def plotDiff():
    x = []
    y2 = []
    for i in range(1, 1001):
        x.append(i)
        y2.append(abs(max(np.linalg.eigh(getA(i))[0]) - func(i) ))

    graph.plot(x, y2, label='numpy')
    graph.xlabel('N')
    graph.ylabel('|ans - func|')
    graph.title('diff')
    graph.legend()
    graph.savefig('draft.png')
```

```
    graph.show()
```

Листинг 6: Погрешность формулы




Действительно, формула весьма точна и погрешность убывает при стремлении к пределу (так как обе сходятся к $\pi$).

После 500 ошибка растет, так как вторая точка была как раз в 500, все логично.

Перейдем теперь к вычислению минимального значения. Раз нужен минимум, то надо взять $\alpha > \lambda_1$.

```
def getMinEig(n, iters=1000, alpha=None):
    alpha = forwardIterations(getA(n), n, iters)[-1] + 1e-6 if alpha is None
   else alpha
    A = getA(n) - alpha * np.identity(n)

    return forwardIterations(A, n, iters)[-1] + alpha

print(getMinEig(3), min(np.linalg.eig(getA(3))[0]), abs(getMinEig(3) - min(np.
   linalg.eig(getA(3))[0])))
0.0026873403557734488 0.002687340355773522 7.32920668600201e-17
```

Листинг 7: Поиск минимального собственного числа

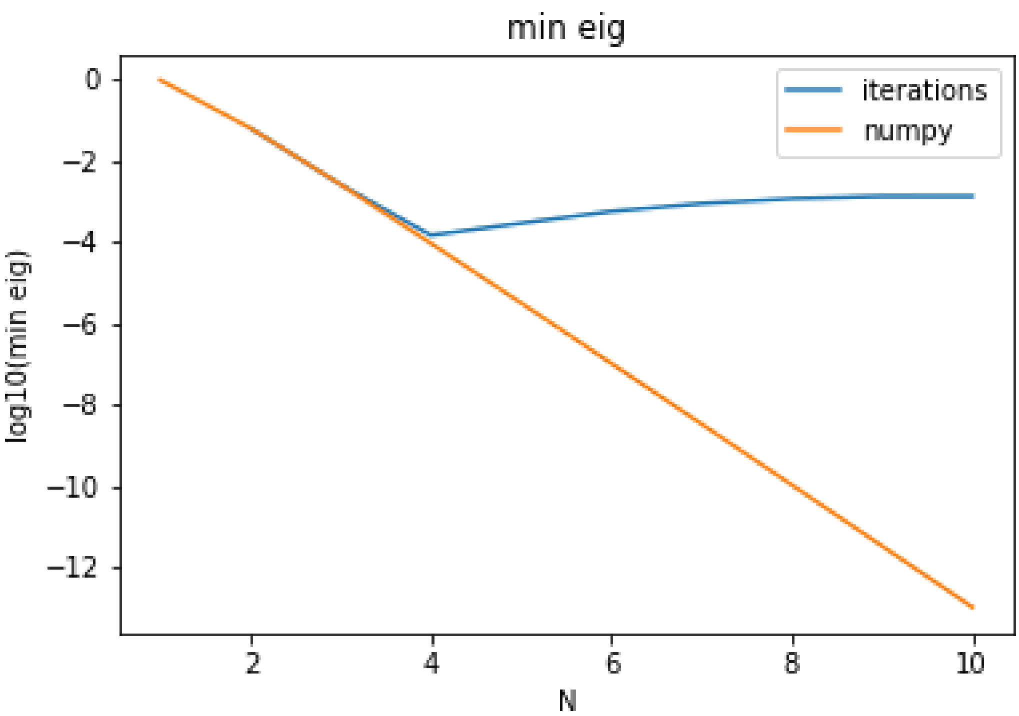
Для размерности 3 получили точный результат, погрешность не будет меньше из-за машинной точности.

Построим график зависимости от $n$.

```
def plotMinEig():
    x = []
    y1 = []
```

```
    y2 = []
    for i in range(1, 11):
        x.append(i)
        y1.append(np.log10(getMinEig(i)))
        y2.append(np.log10(min(np.linalg.eigh(getA(i))[0])))

    graph.plot(x, y1, label='iterations')
    graph.plot(x, y2, label='numpy')
    graph.xlabel('N')
    graph.ylabel('log10(min eig)')
    graph.title('min eig')
    graph.legend()
    graph.savefig('minn.png')
    graph.show()
```

Листинг 8: Зависимость минимального собственного числа от n



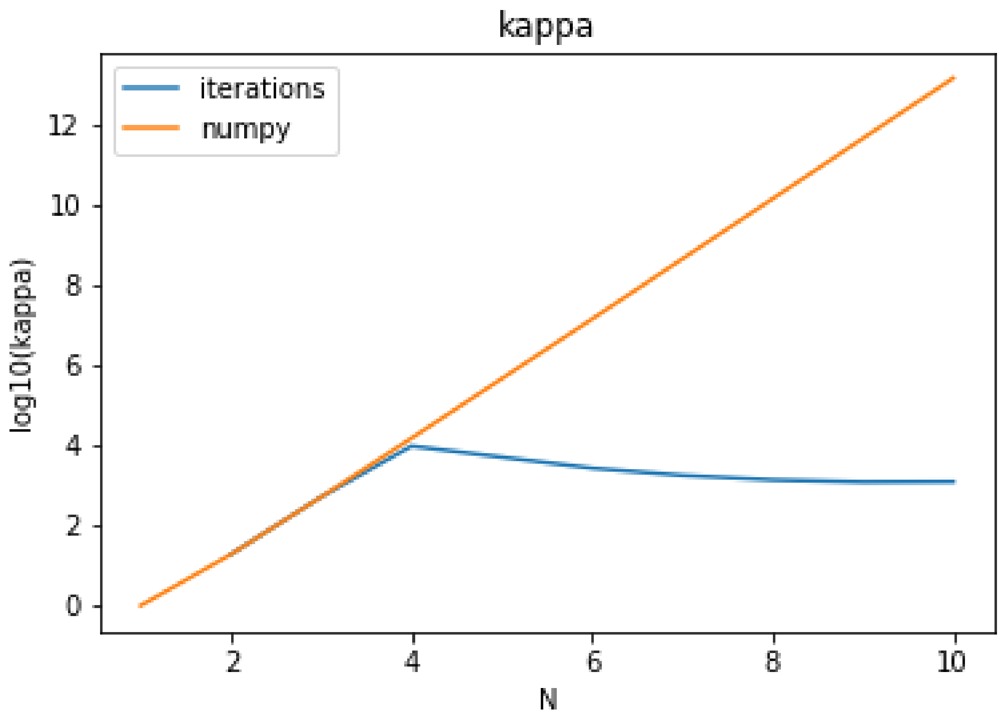

В соответствии с лекцией, после 5 начинается сильное расхождение, и увеличение числа итераций не поможет. Если смотреть на библиотечные результаты, то зависимость обратно экспоненциальная.

Теперь разберемся с $\kappa(n)$.

```
def getK(n):
    return forwardIterations(getA(n), n)[-1] / getMinEig(n)

def plotK():
    x = []
    y1 = []
    y2 = []
    for i in range(1, 11):
```

```
        x.append(i)
        y1.append(np.log10(getK(i)))
        y2.append(np.log10(np.linalg.cond(getA(i))))

    graph.plot(x, y1, label='iterations')
    graph.plot(x, y2, label='numpy')
    graph.xlabel('N')
    graph.ylabel('log10(kappa)')
    graph.title('kappa')
    graph.legend()
    graph.savefig('cap.png')
    graph.show()
```

Листинг 9: Kappa plot




Аналогично, после 5 начинается сильное расхождение. Зависимость исходя из библиотечных значений экспоненциальная. (Логично, так как мы выяснили, что максимальное стремится к пи, а минимальное зависит обратно экспоненциально). Более того, известна асмптотика числа обусловленности: $O(\frac{(1+\sqrt{2})^{4n}}{\sqrt{n}})$. Перевернув это выражение, можно получить асимптотику минимального собственного числа.

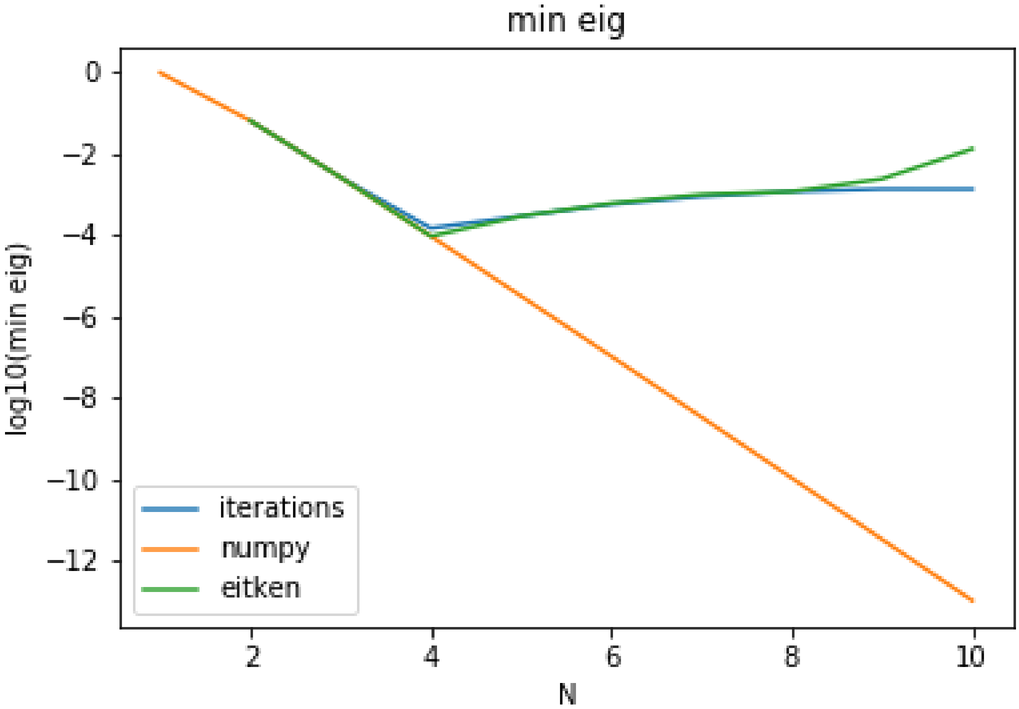
Осталось применить метод Эйткина для ускорения сходимости. Начнем с того, что зная формулу асимптотики выше, уже около 13 (полагая машинную точность как $2^{-64}$) мы упремся в предел машинной точности, поэтому ускорение имеет смысл делать только для небольших $n$. Скопируем теперь Эйткина из первого дз.

```
def calcNext(cur, prev, alpha):
    return np.dot(prev, cur) / np.dot(prev, prev) + alpha

def eitk(n, iters = 1000):
```

```
    alpha = forwardIterations(getA(n), n, iters)[-1] + 1e-6
    A = getA(n) - alpha * np.identity(n)

    prev = np.ones(n)
    cur = A @ prev
    alpha = forwardIterations(getA(n), n)[-1] + 1e-6

    s1 = calcNext(cur, prev, alpha)
    prev = cur
    cur = A @ prev

    s2 = calcNext(cur, prev, alpha)
    prev = cur
    cur = A @ prev

    s3 = calcNext(cur, prev, alpha)
    diff = s3 - (s3 - s2) ** 2 / (s3 - 2 * s2 + s1)

    for i in range(iters):
        prev = cur
        cur = A @ cur
        norm = np.linalg.norm(cur)
        if norm > 1e9:
            cur /= norm
        s1 = s2
        s2 = s3
        s3 = calcNext(cur, prev, alpha)
        if s3 - 2 * s2 + s1 != 0:
            diff = s3 - (s3 - s2) ** 2 / (s3 - 2 * s2 + s1)
    return diff

def plotMinEig2():
    x = []
    y1 = []
    y2 = []
    y3 = []
    for i in range(1, 11):
        x.append(i)
        y1.append(np.log10(getMinEig(i)))
        y2.append(np.log10(min(np.linalg.eigh(getA(i))[0])))
        y3.append(np.log10(eitk(i)))

    graph.plot(x, y1, label='iterations')
    graph.plot(x, y2, label='numpy')
    graph.plot(x, y3, label='eitken')
    graph.xlabel('N')
    graph.ylabel('log10(min eig)')
    graph.title('min eig')
    graph.legend()
    graph.savefig('minne.png')
```

```
        graph.show()
```

Листинг 10: Эйткин




Как видно, точность не улучшилась, точнее, до 5 улучшение прослеживается, однако далее мы опять упираемся в проблемы с точностью вычислений. Что-то мне подсказывает, что итерированный Эйткин ситуацию не спасет.