

Вычисление функций от матриц размеров $n = 2$ и 3 .

Повелители Матрицы.

Воронцов Игорь

Поршин Игорь

Савченко Ольга

Введение

- В проекте сравнивается эффективность (скорость и точность) различных алгоритмов вычисления аналитических функций от маленьких матриц
- Хотим определить, какие алгоритмы самые эффективные
- **Гипотеза:** встроенные реализации матричных функций, которые обычно используют, одинаково работают для матриц разных размеров. Оптимизации под маленькие матрицы не сделано, поэтому эти алгоритмы можно улучшить.
- Матрицы маленького размера играют огромную роль в практических приложениях: программирование графики, видео, управление роботами и дронами, создание обучающих данных для систем огневого поражения БПЛА, решение систем дифференциальных уравнений, и так далее. Но обычно даже специализированные библиотеки используют те же методы, что и `scipy`.
- Качество работы алгоритмов проверяем по максимально достижимой точности и по скорости работы при достижении одинаковой точности. В случае, если метод не может достичь точности встроенного алгоритма, сравниваем скорость работы при максимальной точности вычислений, которую он может дать.

Описание методов

Были рассмотрены вычисление степеней матриц, в том числе дробных; вычисление квадратных корней из матриц, экспонент и логарифмов.

Рассмотрены методы:

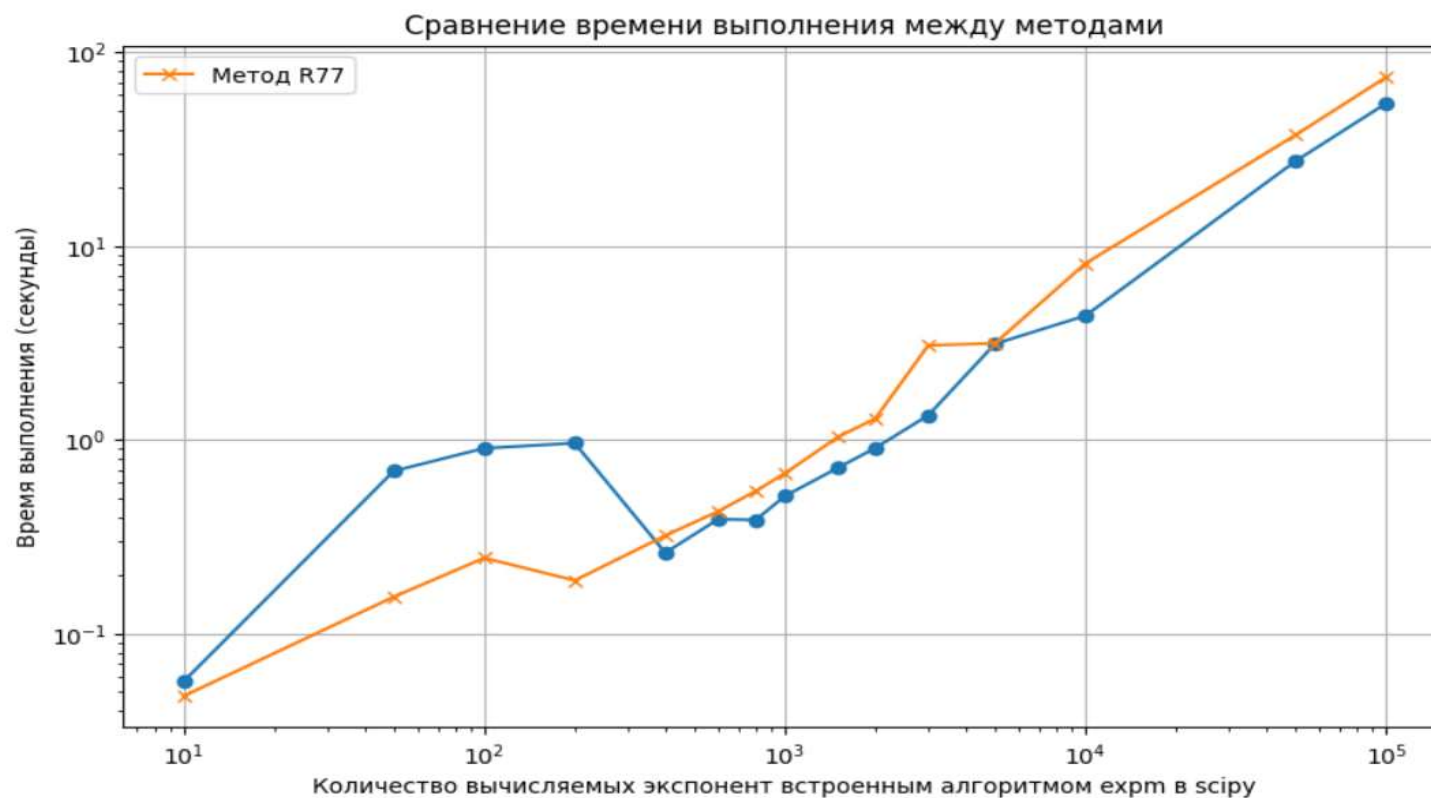
- Встроенные – это аппроксимация Паде переменного порядка с масштабированием и многократным возведением в квадрат, быстрый алгоритм возведения в целую степень, алгоритм Шура-Паде для вычисления дробных степеней матрицы и матричного логарифма, блочный алгоритм Шура для квадратного корня.
- Реализованные самостоятельно – это использование аналитических формул, ряд Тейлора, собственная реализация аппроксимаций Паде, метод гибридных чисел, вычисление экспоненты с помощью второго замечательного предела, вычисление степеней с помощью экспоненты от логарифма, метод Ньютона для матричного квадратного корня, диагонализация матриц, разложение Шура, вычисление экспоненты симметричных матриц с помощью нахождения собственных чисел через решение кубического уравнения, а также вычисление экспоненты кососимметричных матриц и логарифма ортогональных с помощью метода, который основан на формуле Родригеса.

Аналитический метод на основе гибридных чисел.

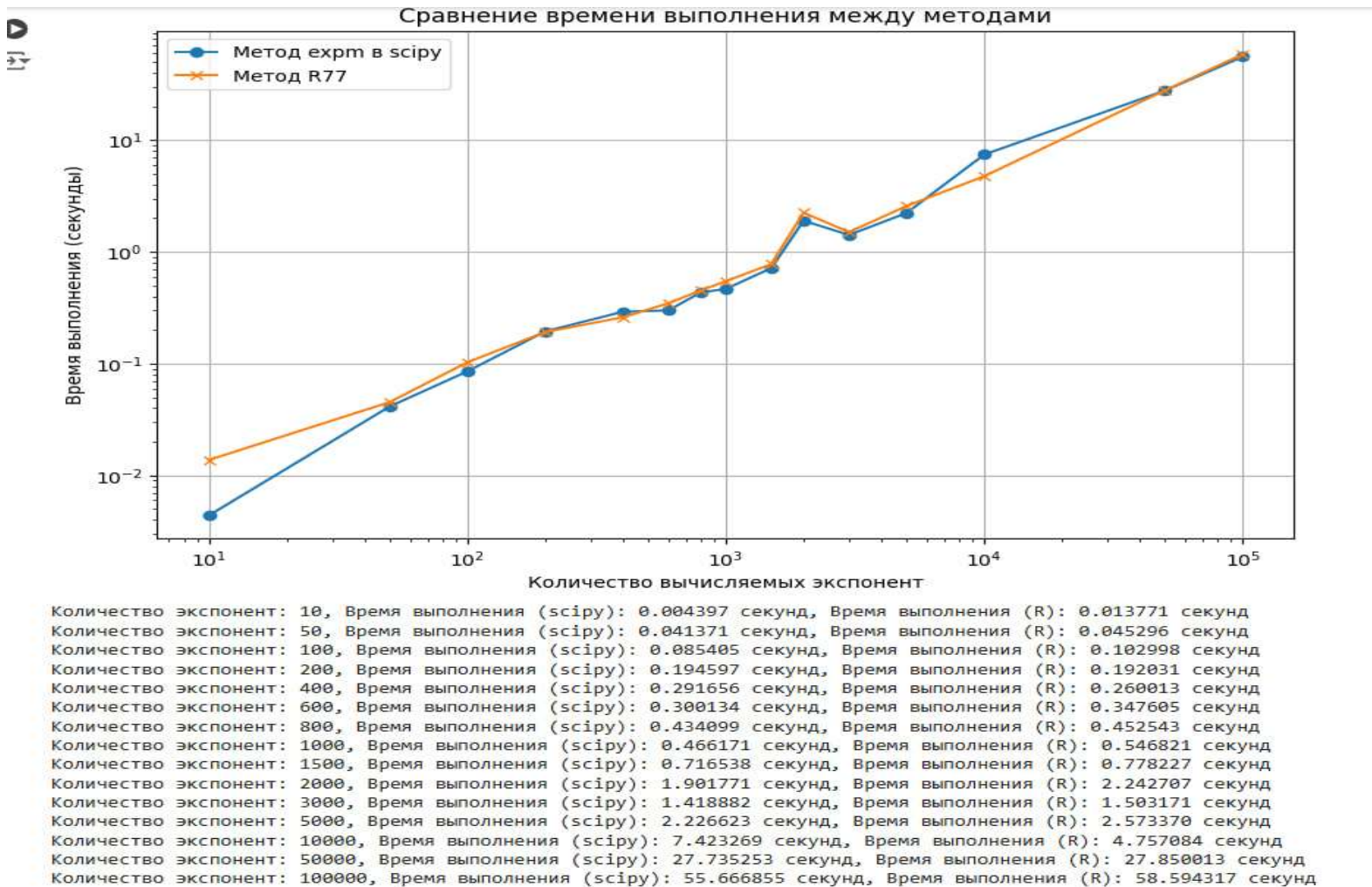
Для матриц размера 2 на 2 существует теория, рассматривающая эти матрицы как гибридные числа (параболические, эллиптические, гиперболические, которые затем делятся на времениподобные, пространственно-подобные и светоподобные). Для каждого вида гибридных чисел есть собственное полярное представление и собственный вариант обобщенной формулы Муавра.

Вычисление по этой формуле в соответствии с алгоритмом, приведенным в научной статье, было реализовано. В результате сравнений оказалось, что алгоритм уступает другим аналитическим методам из-за использования плохо обусловленных в окрестностях некоторых значений функций: логарифмов и отношений обычных и гиперболических синусов к своему малому аргументу. Метод можно доработать, улучшив вычисление каждой функции, но в этом нет необходимости – более общий аналитический подход работает лучше.

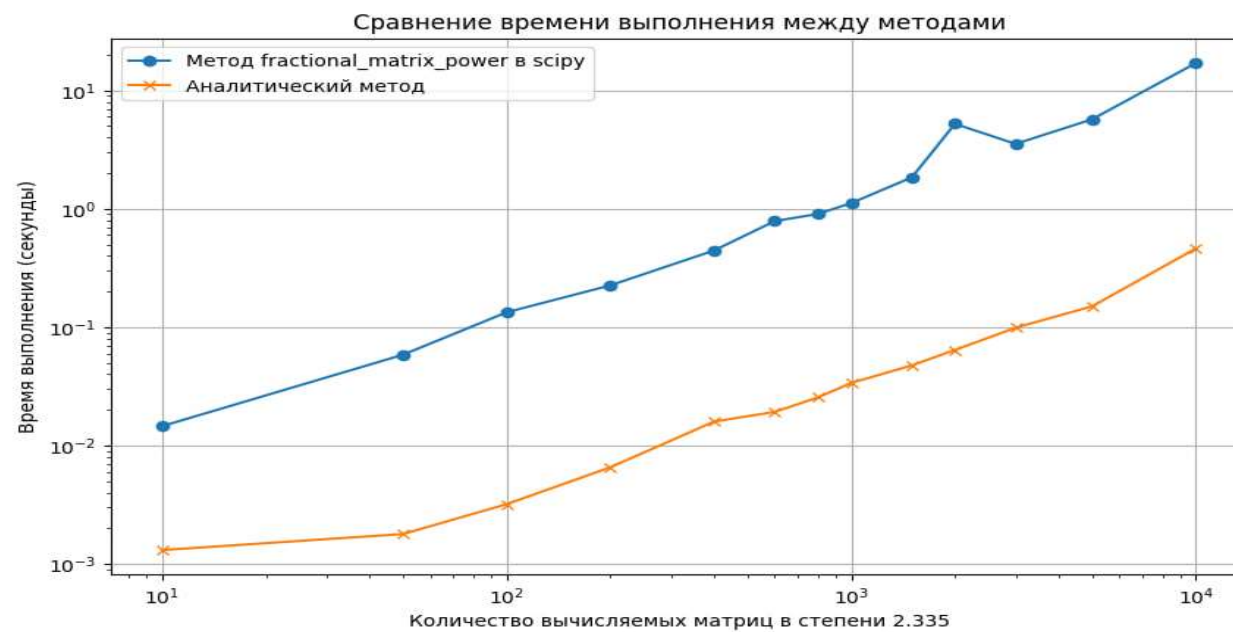
Аппроксимации Паде – собственные и встроенные.



Для получения точности не хуже встроенного метода потребовался 14-й порядок аппроксимации. Встроенный работает быстрее. После оптимизации вычислений удалось приблизить скорость к работе встроенного алгоритма.

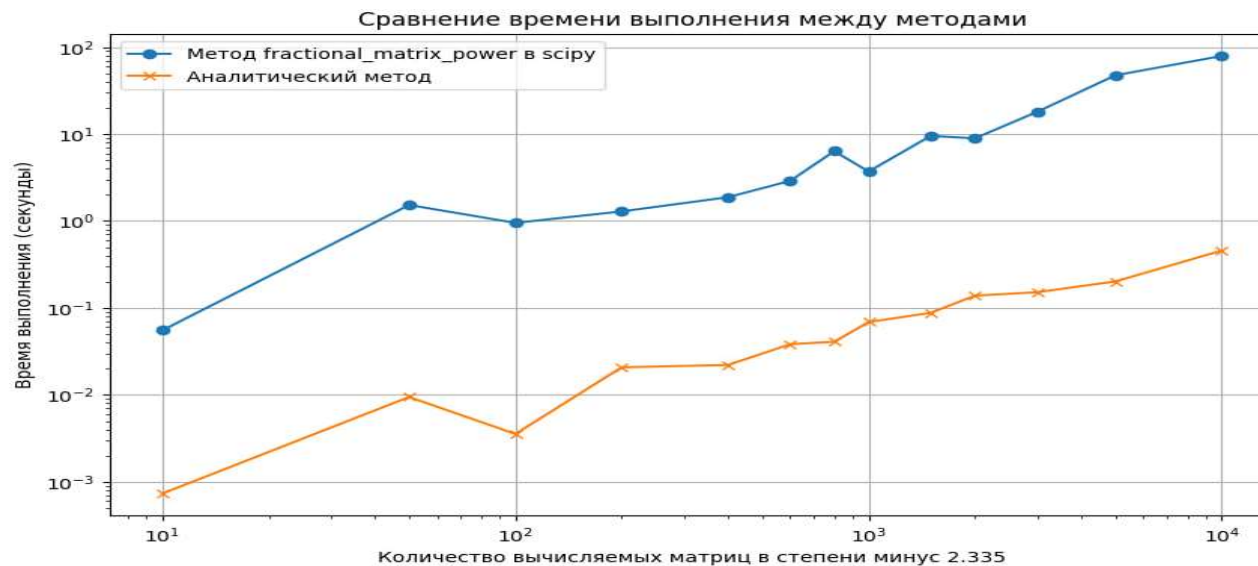


Аналитический метод намного быстрее библиотечного вычисляет дробную степень матрицы.



Количество степеней: 10, Время выполнения (scipy): 0.014578 секунд, Время выполнения (R): 0.001307 секунд
Количество степеней: 50, Время выполнения (scipy): 0.058470 секунд, Время выполнения (R): 0.001786 секунд
Количество степеней: 100, Время выполнения (scipy): 0.133583 секунд, Время выполнения (R): 0.003193 секунд
Количество степеней: 200, Время выполнения (scipy): 0.225527 секунд, Время выполнения (R): 0.006550 секунд
Количество степеней: 400, Время выполнения (scipy): 0.444829 секунд, Время выполнения (R): 0.015930 секунд
Количество степеней: 600, Время выполнения (scipy): 0.788174 секунд, Время выполнения (R): 0.019233 секунд
Количество степеней: 800, Время выполнения (scipy): 0.904253 секунд, Время выполнения (R): 0.025478 секунд
Количество степеней: 1000, Время выполнения (scipy): 1.118044 секунд, Время выполнения (R): 0.033791 секунд
Количество степеней: 1500, Время выполнения (scipy): 1.845243 секунд, Время выполнения (R): 0.047562 секунд
Количество степеней: 2000, Время выполнения (scipy): 5.226793 секунд, Время выполнения (R): 0.064189 секунд
Количество степеней: 3000, Время выполнения (scipy): 3.536763 секунд, Время выполнения (R): 0.098982 секунд
Количество степеней: 5000, Время выполнения (scipy): 5.709252 секунд, Время выполнения (R): 0.149979 секунд
Количество степеней: 10000, Время выполнения (scipy): 17.036253 секунд, Время выполнения (R): 0.462106 секунд

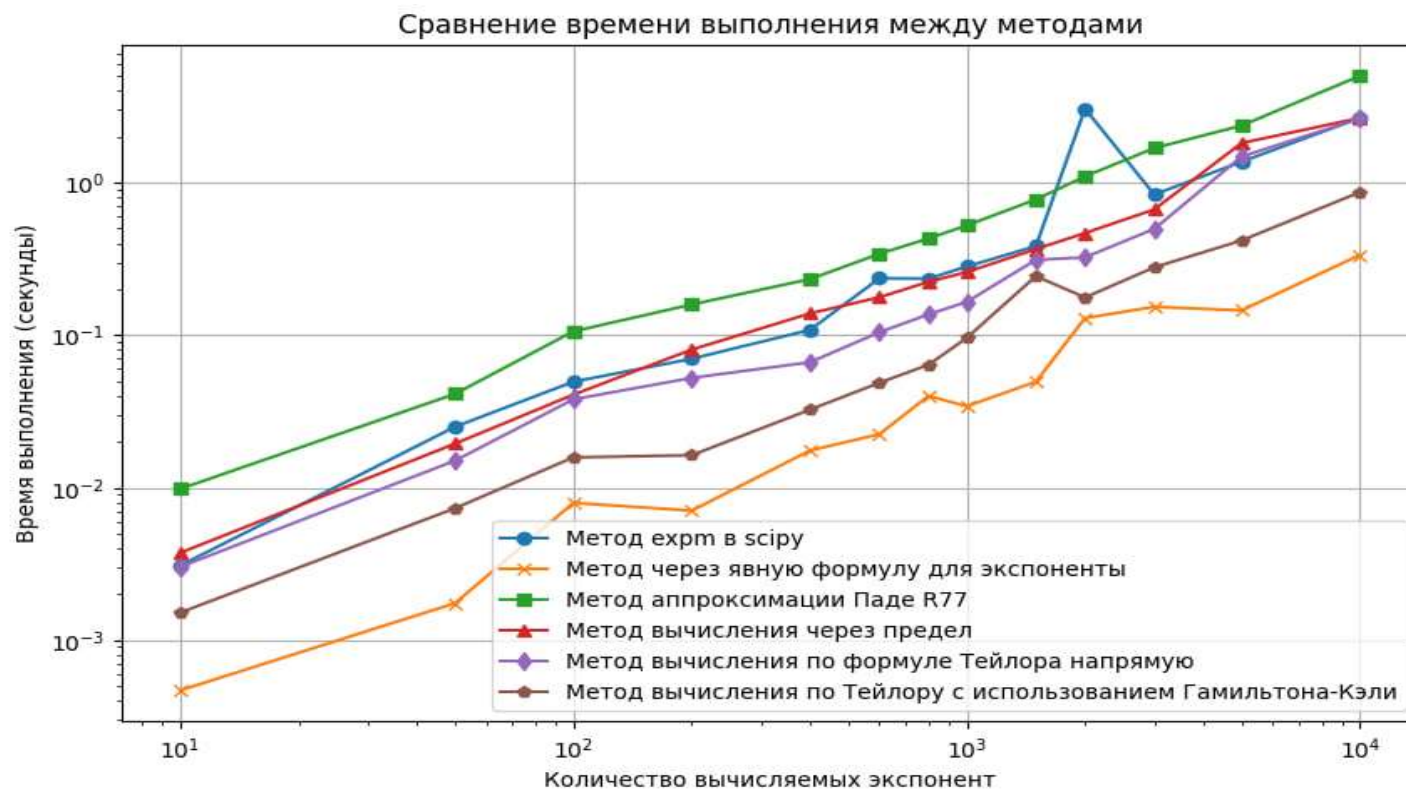
Если степень отрицательная и дробная, аналитический метод в сотни раз быстрее.



```
# Вывод результатов
for n, t_scipy, t_R in zip(num_power, execution_times, execution_times_R):
    print(f"Количество степеней: {n}, Время выполнения (scipy): {t_scipy:.6f} секунд, Время выполнения (R): {t_R:.6f} секунд")
```

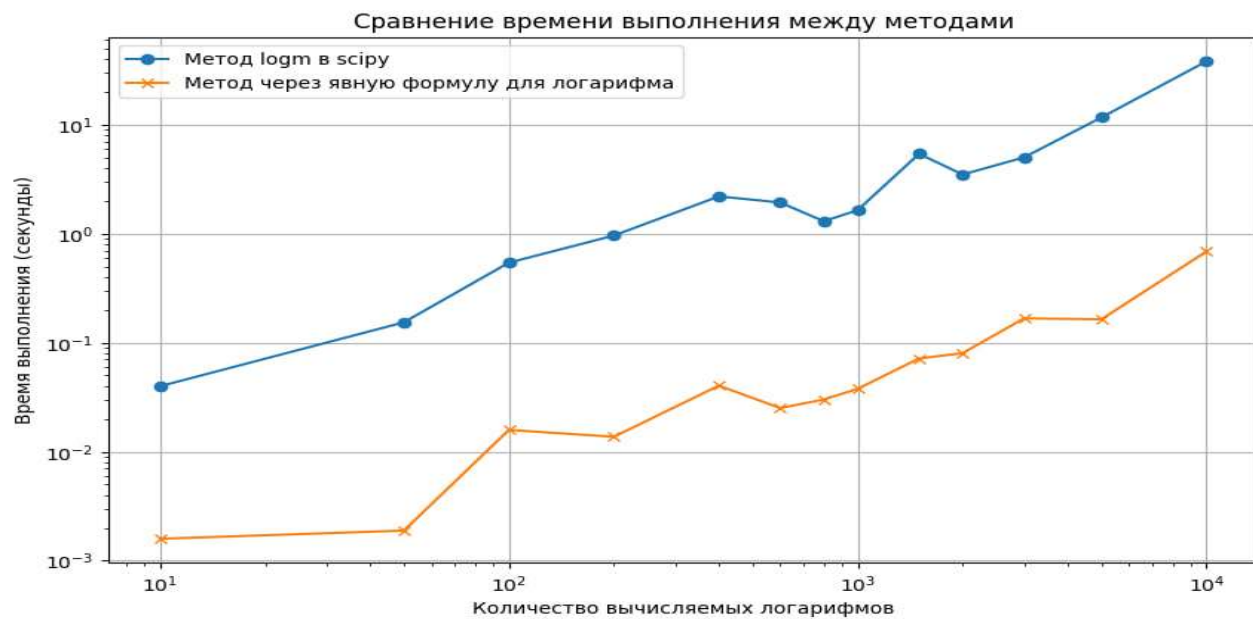
Количество степеней: 10, Время выполнения (scipy): 0.055180 секунд, Время выполнения (R): 0.000732 секунд
Количество степеней: 50, Время выполнения (scipy): 1.526783 секунд, Время выполнения (R): 0.009457 секунд
Количество степеней: 100, Время выполнения (scipy): 0.951416 секунд, Время выполнения (R): 0.003553 секунд
Количество степеней: 200, Время выполнения (scipy): 1.288175 секунд, Время выполнения (R): 0.020760 секунд
Количество степеней: 400, Время выполнения (scipy): 1.877531 секунд, Время выполнения (R): 0.022062 секунд
Количество степеней: 600, Время выполнения (scipy): 2.891839 секунд, Время выполнения (R): 0.038275 секунд
Количество степеней: 800, Время выполнения (scipy): 6.361767 секунд, Время выполнения (R): 0.040813 секунд
Количество степеней: 1000, Время выполнения (scipy): 3.702780 секунд, Время выполнения (R): 0.068814 секунд
Количество степеней: 1500, Время выполнения (scipy): 9.520500 секунд, Время выполнения (R): 0.087835 секунд
Количество степеней: 2000, Время выполнения (scipy): 8.920035 секунд, Время выполнения (R): 0.138425 секунд
Количество степеней: 3000, Время выполнения (scipy): 17.961743 секунд, Время выполнения (R): 0.151571 секунд
Количество степеней: 5000, Время выполнения (scipy): 47.364666 секунд, Время выполнения (R): 0.201518 секунд
Количество степеней: 10000, Время выполнения (scipy): 78.989968 секунд, Время выполнения (R): 0.456645 секунд

Сравнение методов вычисления матричных экспонент от матриц 2 на 2.



Аналитический метод и ускорение расчета степеней матрицы в формуле Тейлора через уравнение Гамильтона-Кэли значительно опережают библиотечную реализацию.

Встроенный матричный логарифм удалось обогнать более чем в 50 раз!



Количество логарифмов: 10, Время выполнения (scipy): 0.039851 секунд, Время выполнения с помощью аналитической формулы 0.001590 секунд
Количество логарифмов: 50, Время выполнения (scipy): 0.153577 секунд, Время выполнения с помощью аналитической формулы 0.001888 секунд
Количество логарифмов: 100, Время выполнения (scipy): 0.541420 секунд, Время выполнения с помощью аналитической формулы 0.015835 секунд
Количество логарифмов: 200, Время выполнения (scipy): 0.959634 секунд, Время выполнения с помощью аналитической формулы 0.013700 секунд
Количество логарифмов: 400, Время выполнения (scipy): 2.195255 секунд, Время выполнения с помощью аналитической формулы 0.040346 секунд
Количество логарифмов: 600, Время выполнения (scipy): 1.928695 секунд, Время выполнения с помощью аналитической формулы 0.025115 секунд
Количество логарифмов: 800, Время выполнения (scipy): 1.298252 секунд, Время выполнения с помощью аналитической формулы 0.030085 секунд
Количество логарифмов: 1000, Время выполнения (scipy): 1.639570 секунд, Время выполнения с помощью аналитической формулы 0.037729 секунд
Количество логарифмов: 1500, Время выполнения (scipy): 5.394557 секунд, Время выполнения с помощью аналитической формулы 0.071582 секунд
Количество логарифмов: 2000, Время выполнения (scipy): 3.483544 секунд, Время выполнения с помощью аналитической формулы 0.080075 секунд
Количество логарифмов: 3000, Время выполнения (scipy): 5.000974 секунд, Время выполнения с помощью аналитической формулы 0.166964 секунд
Количество логарифмов: 5000, Время выполнения (scipy): 11.621601 секунд, Время выполнения с помощью аналитической формулы 0.163855 секунд
Количество логарифмов: 10000, Время выполнения (scipy): 37.989411 секунд, Время выполнения с помощью аналитической формулы 0.685144 секунд

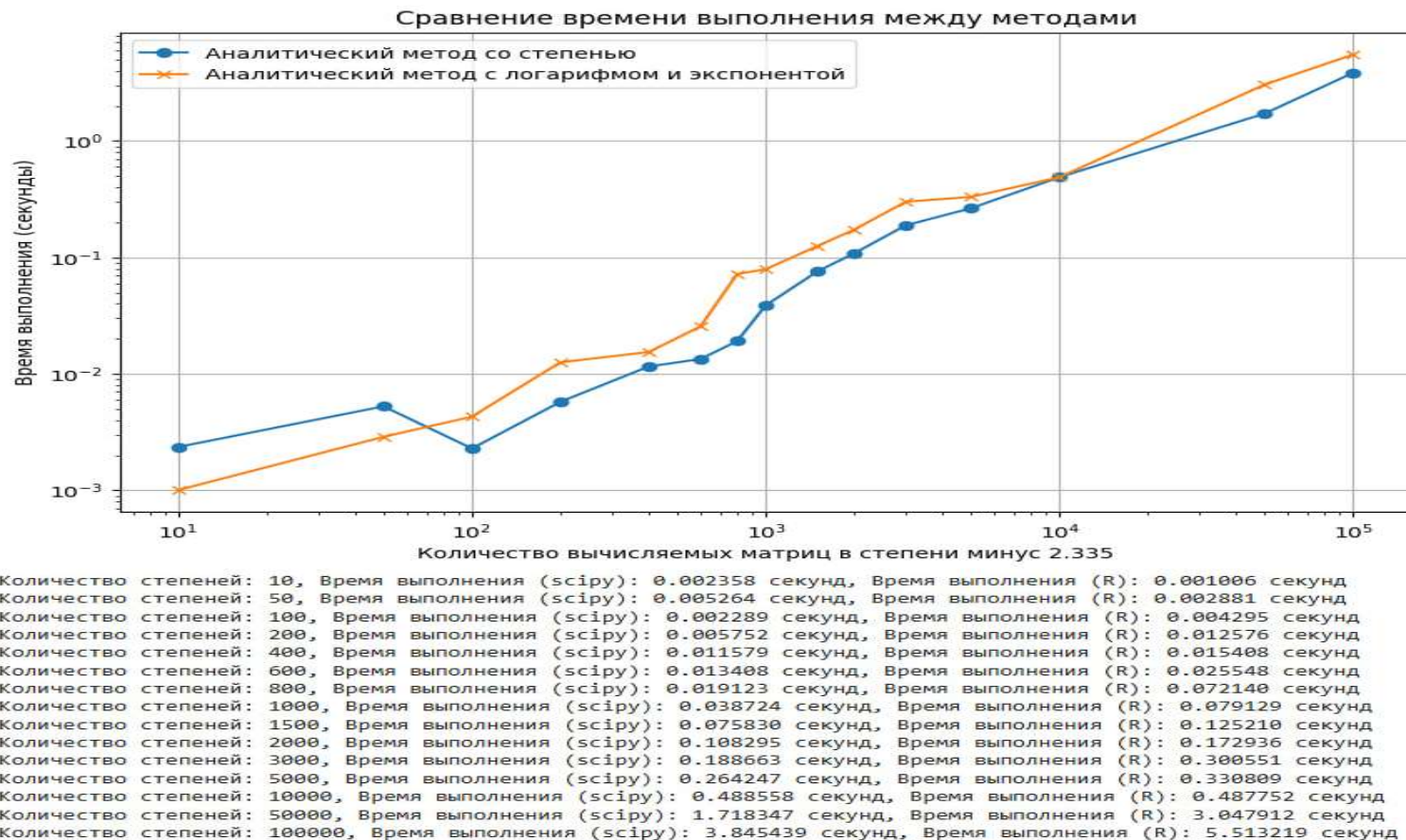
При этом точность аналитического расчета матричного логарифма на уровне машинной независимо от вида матрицы. Библиотечный алгоритм на некоторых матрицах дает точность хуже до 10-15 раз в зависимости от матрицы. Лучшую точность встроенный не смог продемонстрировать ни на одной матрице!

```
AA = np.array([[1, 2], [3, 4]])

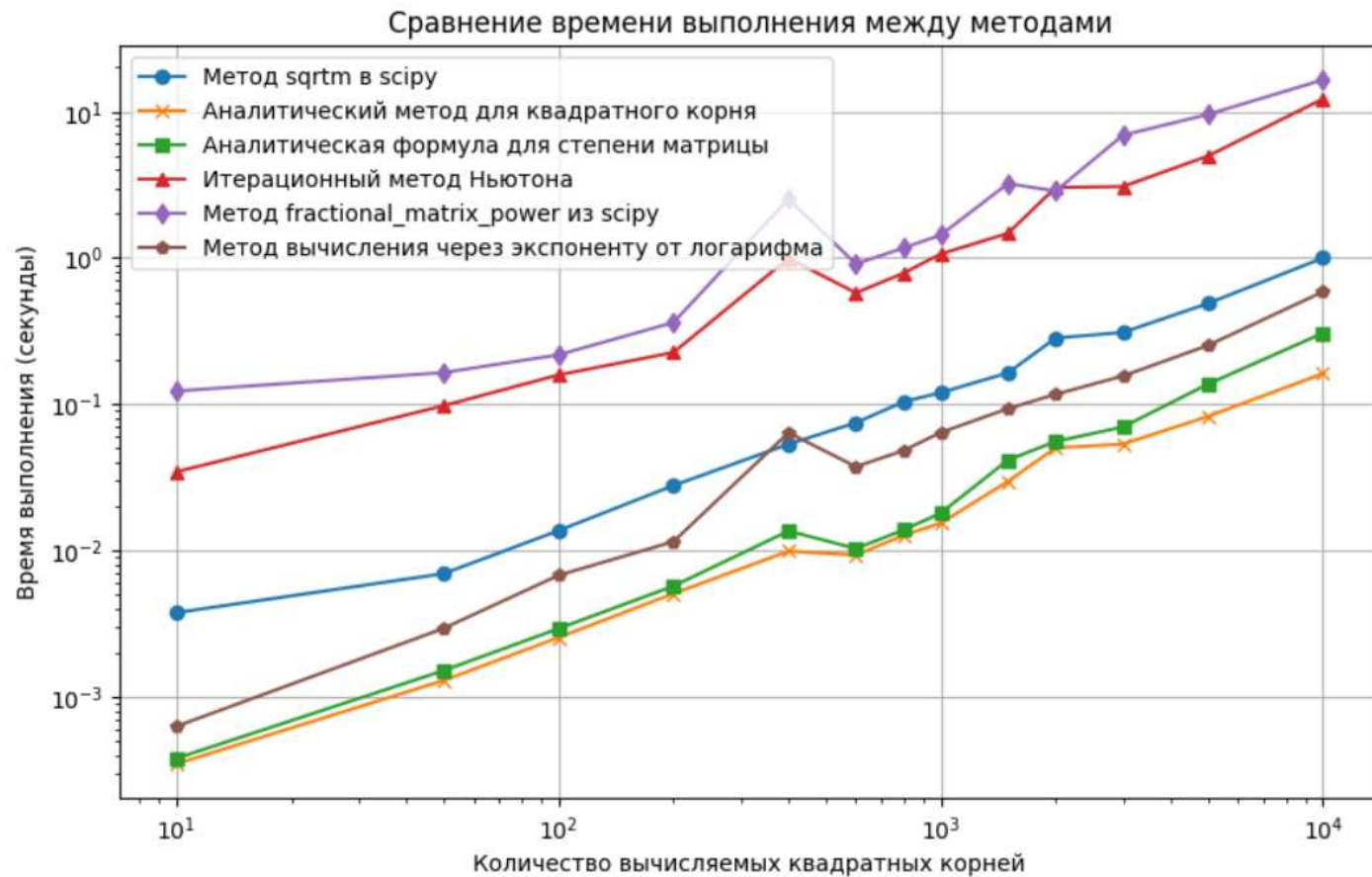
for q in range(2000):
    A = AA + q*np.array([[1, 0], [0, 1]])*0.01
    if q % 100 == 0:
        print('точность встроенного метода', np.linalg.norm(logm(expm(A))-A, 'fro')/np.linalg.norm(A, 'fro'), end = ' | ')
        print('точность аналитического метода', np.linalg.norm(matrix_log_analytic(expm(A))-A, 'fro')/np.linalg.norm(A, 'fro'))
```

точность встроенного метода 3.783890250755347e-15		точность аналитического метода 3.3674731643402955e-16
точность встроенного метода 6.044928789819412e-15		точность аналитического метода 1.3896279205500544e-15
точность встроенного метода 2.762433558681622e-15		точность аналитического метода 1.5427844197272194e-16
точность встроенного метода 1.2316818592266832e-15		точность аналитического метода 1.2316818592266832e-16
точность встроенного метода 3.473461024916715e-15		точность аналитического метода 1.1047673502100373e-15
точность встроенного метода 2.1475107019845614e-15		точность аналитического метода 4.529676449456192e-16
точность встроенного метода 7.707661349566876e-16		точность аналитического метода 3.006498163255289e-16
точность встроенного метода 1.7713031823906393e-15		точность аналитического метода 5.290427728586987e-16
точность встроенного метода 4.250206091619259e-16		точность аналитического метода 1.3191415199864766e-16
точность встроенного метода 1.949958342364329e-15		точность аналитического метода 7.237466655017797e-16
точность встроенного метода 1.6544208329073209e-15		точность аналитического метода 4.3867328517029085e-16
точность встроенного метода 1.3621582844854534e-15		точность аналитического метода 1.181522192307309e-15
точность встроенного метода 5.171617404750989e-16		точность аналитического метода 2.2853996708612937e-16
точность встроенного метода 8.848280553651172e-16		точность аналитического метода 7.885066783010756e-16
точность встроенного метода 3.2875882652459006e-16		точность аналитического метода 3.41841023485266e-16
точность встроенного метода 1.216992480482259e-15		точность аналитического метода 2.8652181583621774e-16
точность встроенного метода 1.0707751022501119e-15		точность аналитического метода 4.2732504141856975e-16
точность встроенного метода 3.7575781364353835e-16		точность аналитического метода 1.9369182016661636e-16
точность встроенного метода 1.3567149463006938e-15		точность аналитического метода 4.014075956623856e-16
точность встроенного метода 2.035689575619724e-15		точность аналитического метода 2.016625634589731e-16

Аналитический метод вычисления степени матрицы напрямую немного эффективнее, чем метод, основанный на взятии экспоненты от логарифма.

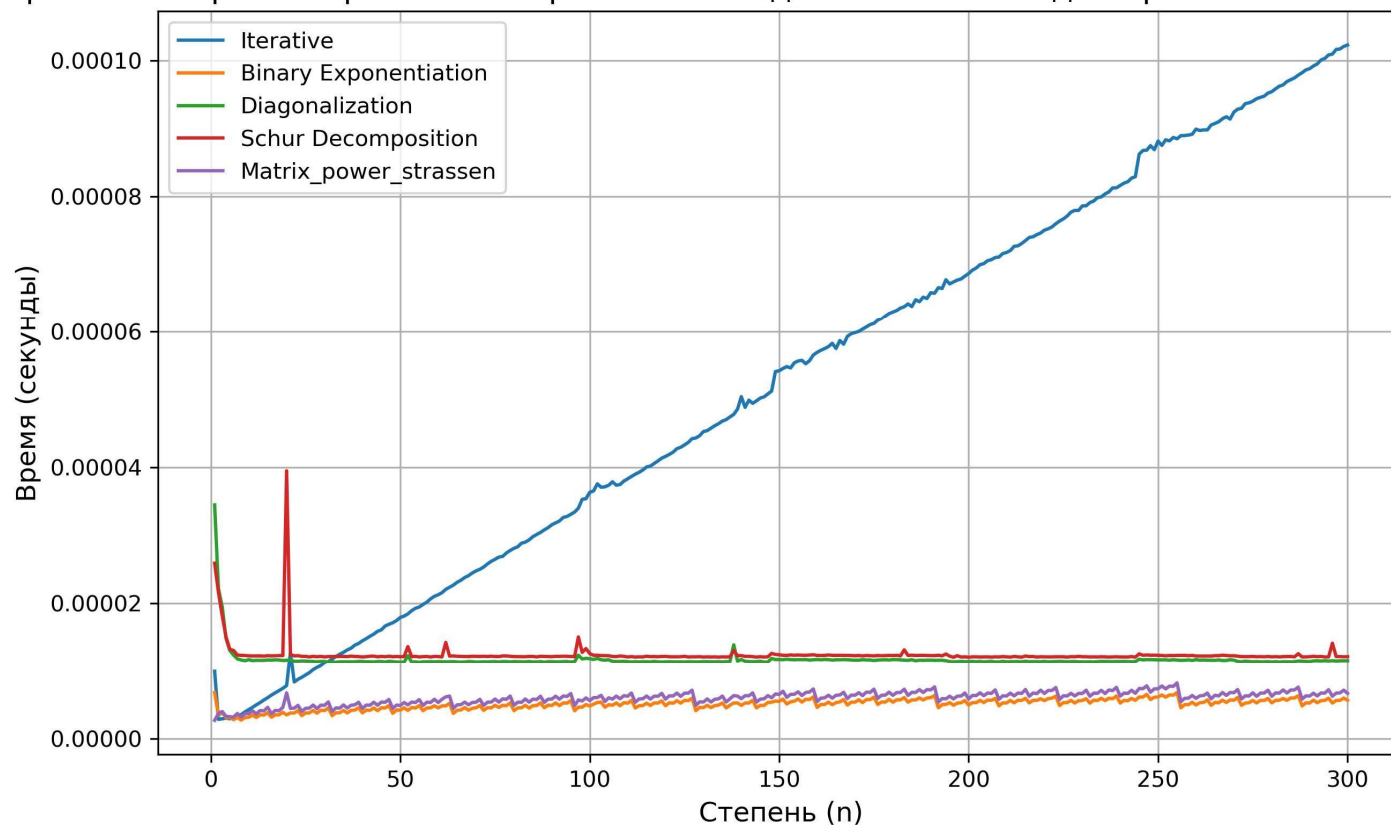


Аналитический метод расчета квадратного корня оказался самым эффективным, однако не сильно превзошел аналитический метод возведения в степень с показателем 0.5

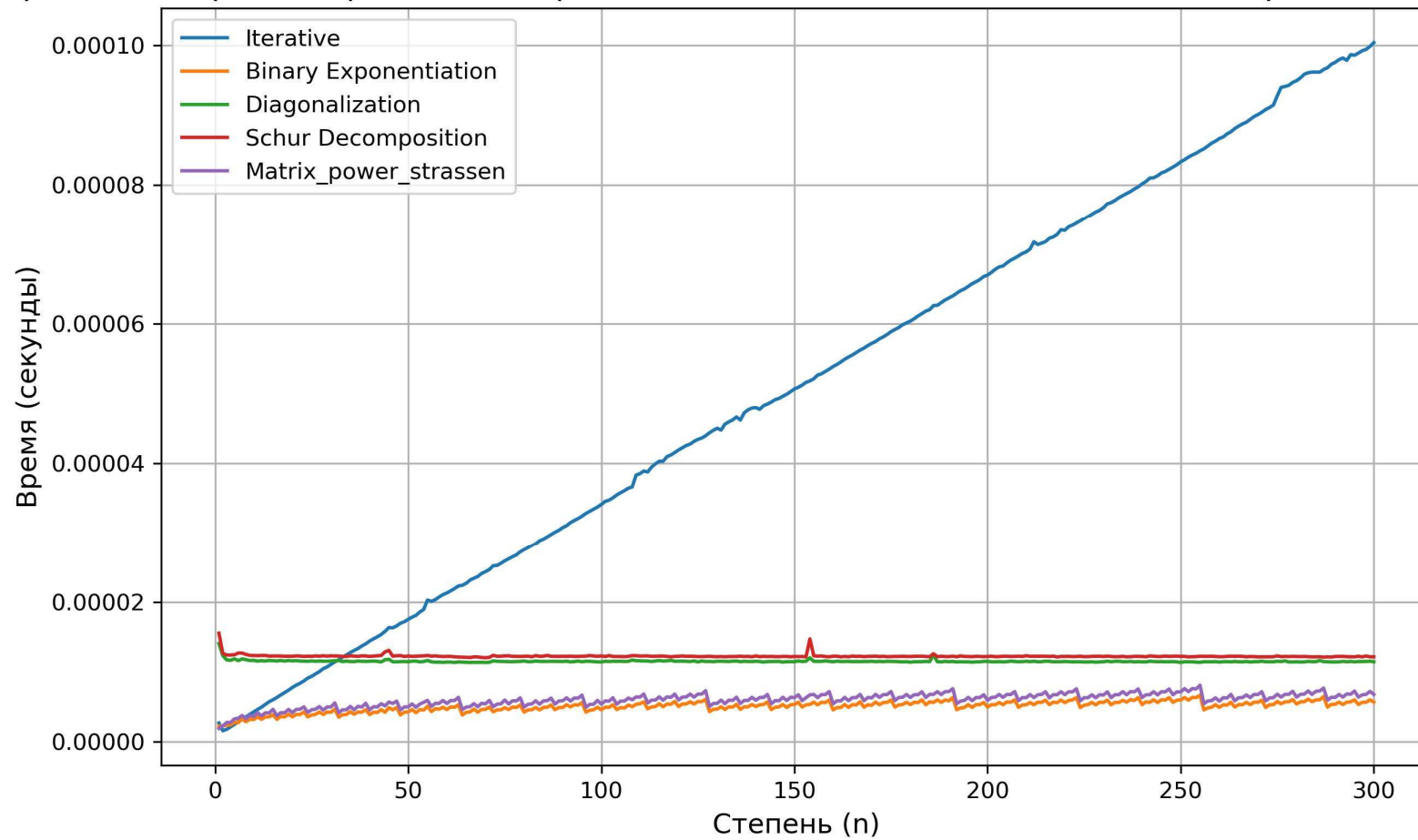


Были сделаны попытки превзойти алгоритм бинарного возведения в натуральную степень, который также реализован в библиотеке numpy, для матриц 3 на 3 разными методами. Этот алгоритм всё равно оказался лучше прочих методов, как и было в матрицах 2 на 2.

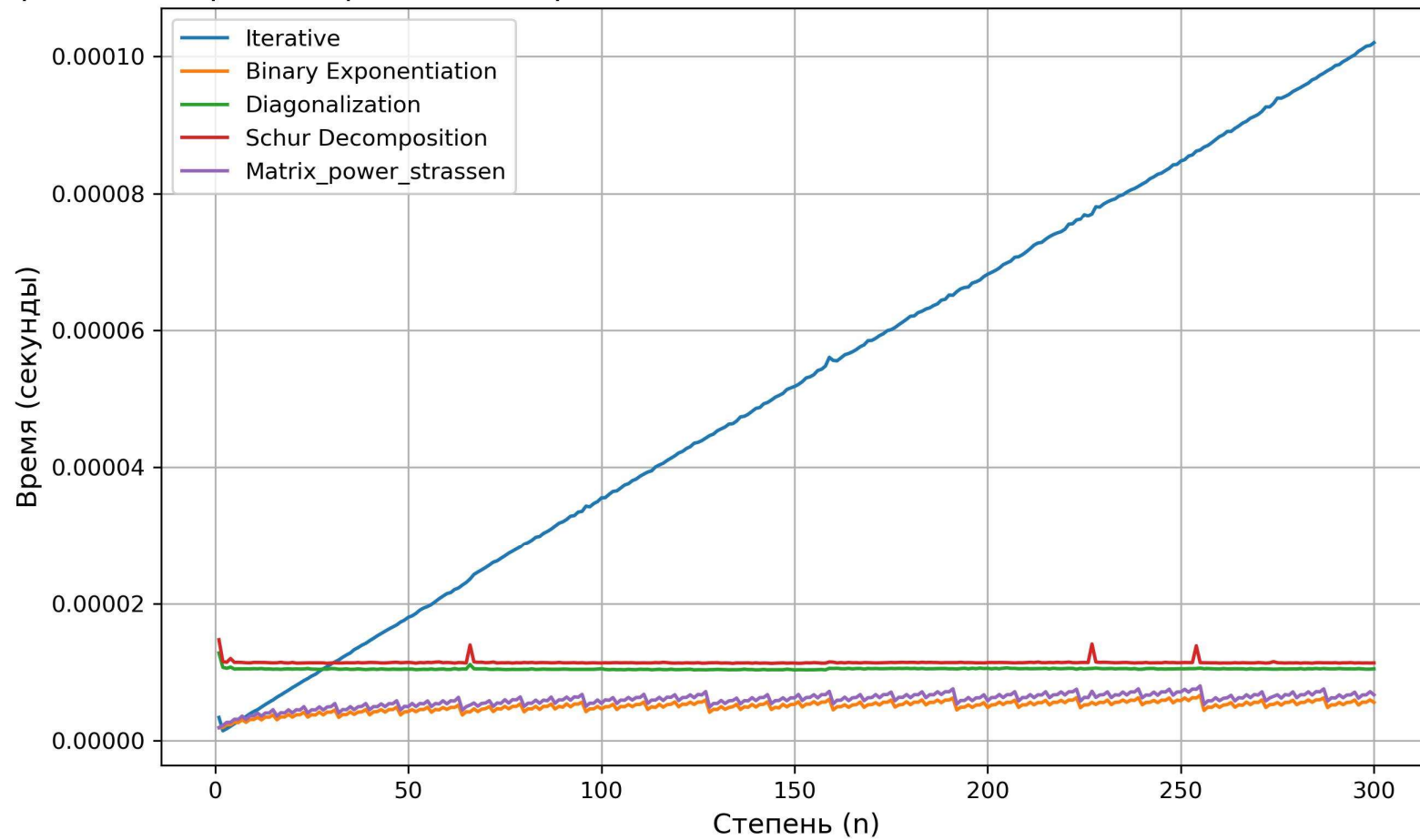
Сравнение времени работы алгоритмов возведения в степень для произвольных матриц 3x3



Сравнение времени работы алгоритмов возведения в степень для симметричных матриц 3x3



Сравнение времени работы алгоритмов возведения в степень для диагональных матриц 3x3



Формула Родригеса: Матричная экспонента для кососимметрических матриц 3x3

Кососимметрическая матрица A размера 3x3 имеет вид:

$$\begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix}$$

где a, b, c - вещественные числа. Матричная экспонента матрицы A:

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

Положим:

$$\theta = \sqrt{a^2 + b^2 + c^2}$$

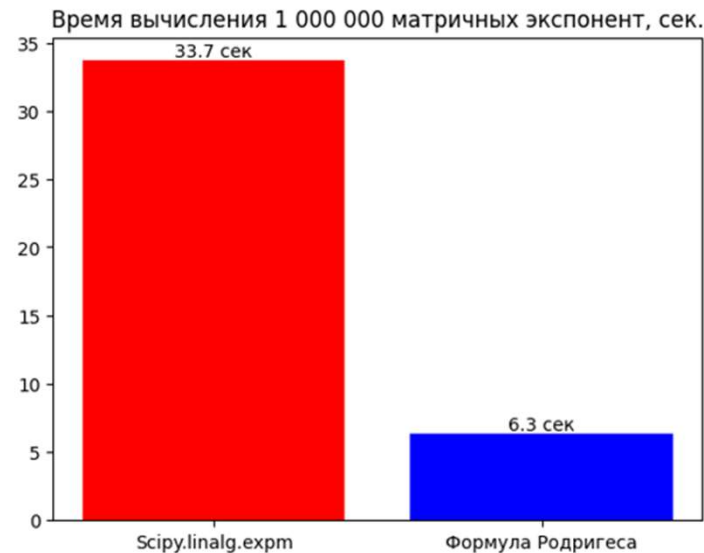
Тогда матричную экспоненту можно найти по формуле Родригеса:

$$e^A = I_3 + \frac{\sin \theta}{\theta} A + \frac{1 - \cos \theta}{\theta^2} A^2$$

Сравнение точности и скорости алгоритмов

Кастомная имплементация матричной экспоненты через формулу Родригеса сравнивалась с реализацией из библиотеки Scipy: `scipy.linalg.expm`.

Сравнение точности и скорости реализаций производилось на 1_000_000 сгенерированных кососимметрических матриц размера 3 x 3. Матричные экспоненты, получаемые с помощью реализации через формулу Родригеса, были неотличимы от полученных благодаря функции из Scipy вплоть до машинной точности, но на их вычисление понадобилось меньше времени:



Матричный логарифм от матрицы вращения

Пусть A – кососимметрическая матрица размера 3×3 , тогда:

$$e^A = R$$

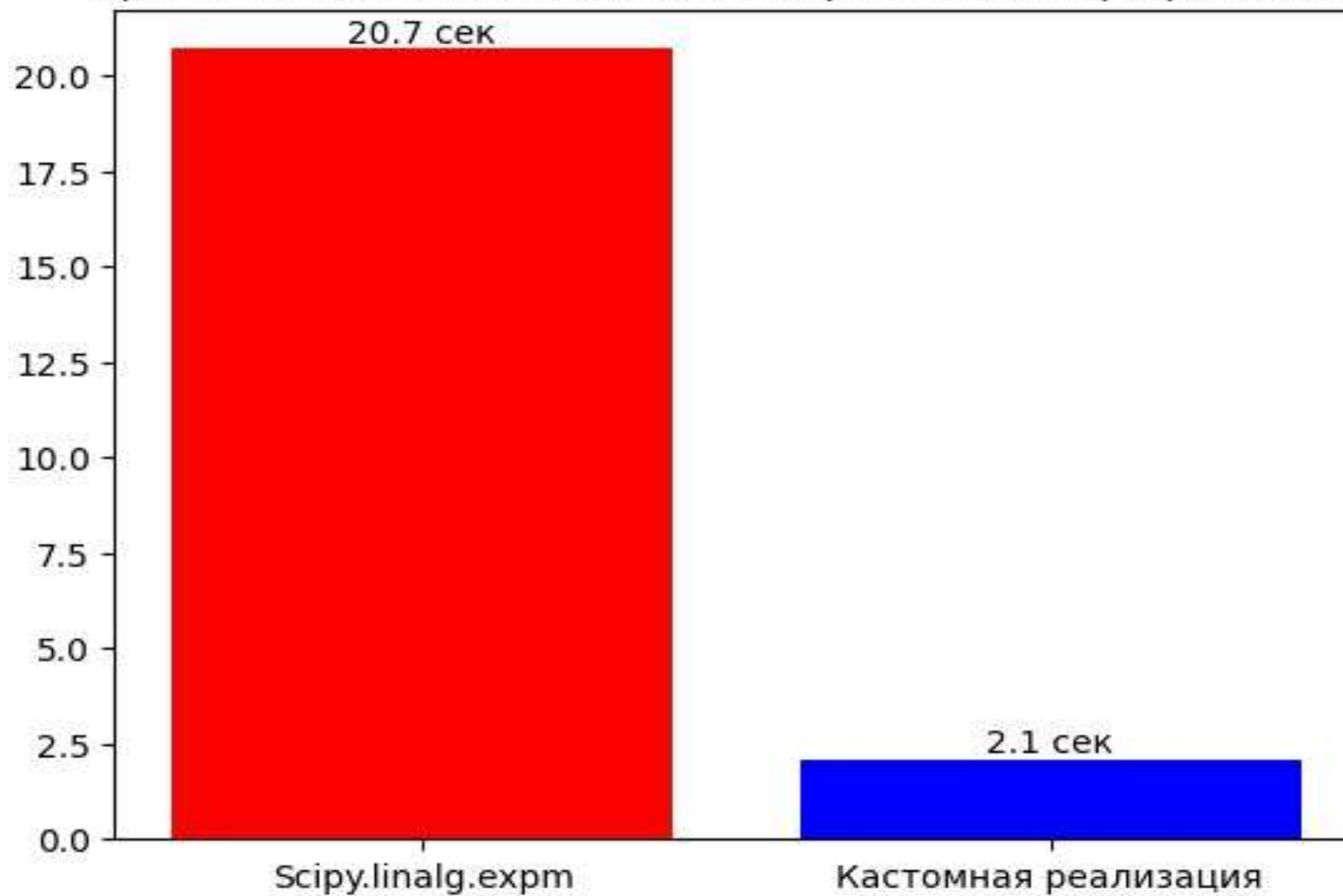
Причём для любой матрицы вращения размера 3×3 существует такая кососимметрическая матрица, что верно выражение выше.

Для матрицы вращения её матричный логарифм (кососимметрическая матрица A) может быть найден по формуле:

$$A = \log R = \frac{\theta}{2 \sin \theta} (R - R^T)$$

$$\text{tr}(R) = 1 + 2 \cos \theta$$

Время вычисления 1 000 000 матричных логарифмов, сек.



Матричная экспонента для симметрических матриц 3x3

Пусть $\phi_A(\lambda)$ – характеристический многочлен симметрической матрицы A 3 степени. По теореме Гамильтона-Кэли знаем, что:

$$\phi_A(A) = 0$$

Каждое слагаемое из ряда для матричной экспоненты можем представить как:

$$\frac{A^k}{k!} = Q_k \phi_A(A) + R_k$$

где $\deg(R_k) \leq 2$. Тогда:

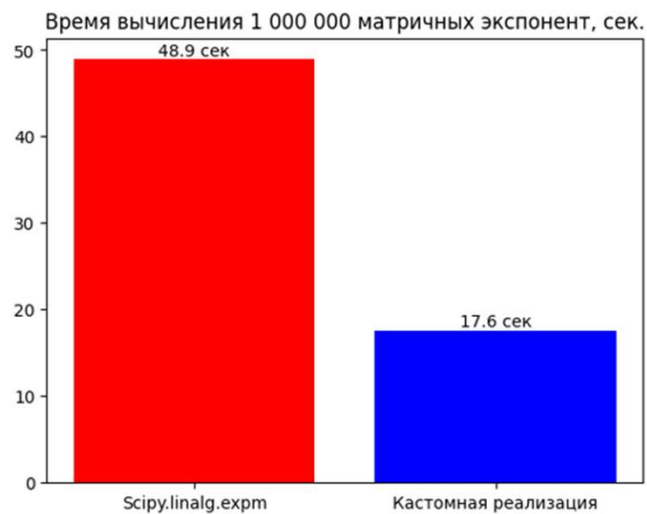
$$e^A = \sum_{k=0}^{\infty} Q_k \phi_A(A) + R_k = \sum_{k=0}^{\infty} R_k = xA^2 + yA + z$$

где x, y, z – вещественные числа, вычисляемые на основе собственных чисел A

Сравнение точности и скорости алгоритмов

Кастомная имплементация матричной экспоненты для симметрических снова сравнивалась с реализацией из библиотеки Scipy: `scipy.linalg.expm`.

Аналогично экспериментах с кососимметрическими матрицами сравнение точности и скорости реализаций производилось на 1_000_000 сгенерированных симметрических матриц размера 3 x 3. Вычисления с помощью кастомного алгоритма снова получились быстрее:



Выводы

- Мы сумели превзойти библиотечные реализации матричных функций от маленьких матриц.
- Было проведено множество сравнений методов и проверены гипотезы.

Удалось

- Ускорить вычисление дробной отрицательной степени матрицы в несколько сотен раз по сравнению с библиотечной реализацией. Для дробной положительной – в десятки раз. Встроенный метод `sqrtm` в `scipy` для вычисления квадратного корня удалось обогнать приблизительно в 7 раз при той же точности.
- Был рассмотрен как перспективный метод алгоритм вычисления экспоненты через предел. Для ряда приложений, в которых используют аппроксимации Паде не выше 8-го порядка, он актуален.
- Было показано превосходство аналитических методов во всех случаях, кроме возведения матриц в целую степень (для них лучше всего работает стандартный алгоритм).
- Показано, что для маленьких матриц ряд Тейлора быстрее работает при той же точности, чем аппроксимация Паде. С ростом размера матриц аппроксимации Паде начинают обгонять ряд Тейлора.
- Среди разных аналитических подходов были найдены самые лучшие.
- Для симметричных и кососимметричных матриц размера 3 на 3 реализовали алгоритмы вычисления экспоненты на основе научных статей, которые работают значительно лучше, чем алгоритмы `scipy`.

Для развития проекта следует:

- Продолжить исследование матриц 3 на 3. Сделать намного больше разных тестов для 2 на 2. Также обобщить на матрицы чуть большего размера.
- Разобрать множество случаев матриц специального вида.
- Написать собственный сверхбыстрый движок геометрической алгебры. Почти все известные используют библиотечные матричные функции и поэтому работают неэффективно.

Ссылки

- <https://github.com/IgorOberon/NLA-project-small-matrix>
- <https://bivector.net/lib.html> - библиотеки геометрической алгебры для разных приложений в компьютерной графике и робототехнике. Все используют для вычисления матричных функций обычные библиотеки, реализующие те же самые методы, что и `scipy`.
- https://github.com/numpy/numpy/blob/v2.1.0/numpy/linalg/_linalg.py - здесь можно прочитать про детали реализации матричных функций в `numpy`.
- https://github.com/scipy/scipy/blob/main/scipy/linalg/_matfuncs.py – здесь можно прочитать про детали реализации матричных функций в `scipy`.
- <https://toomanydigits.online/> авторский курс вычислительной математики.

Вклад участников проекта.

- Поиск статей, слайды для презентации, блокнот – участвовали все.
- Постановка общей задачи, аналитические методы для матриц 2 на 2 – **Воронцов Игорь.**
- Собственный вариант аппроксимаций Паде – **Воронцов Игорь.**
- Составление каталога научных статей и скачивание тех из них, к которым не было изначально прямого доступа – **Савченко Ольга.**
- Исследование методов вычисления экспонент и логарифмов через формулу Родригеса для кососимметрических матриц и сравнение их со Scipy реализацией – **Савченко Ольга.**
- Исследование методов вычисления матричных экспонент от симметрических матриц и сравнение их со Scipy реализацией – **Савченко Ольга.**
- Разложение Шура, бинарное возведение в степень, диагонализация матриц, использование метода Штрассена, итерационные методы – **Поршин Игорь.**
- Обработка исследований в теке, анализ материала и поиск методов для возведения в степень матриц 3 на 3, программная реализация – **Поршин Игорь.**