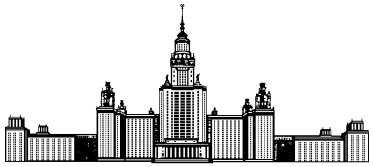
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

«Сравнение скорости вычисления собственных значений положительно определённых матриц при помощи QR алгоритма»

Выполнил:

студент 1 курса магистратуры 517 группы

Королев Николай Сергеевич

Преподаватель:

канд. техн. наук, доцент

Русол Андрей Владимирович

Москва, 2021

1 Постановка задачи

Исследовать способы ускорения вычисления собственных значений положительно определённых матриц.

1.1 Постановка задачи о вычислении собственных значений положительно определённой матрицы

Дана положительно определённая матрица A размера $N \times N$. Необходимо вычислить все N её собственных значений $\lambda_1, \lambda_2, \ldots, \lambda_N$.

2 QR алгоритм

Для нахождения всех собственных значений положительно определённой матрицы A можно воспользоваться QR алгоритмом, который выглядит следующим образом:

1. Обозначим $A_0 := A,\ k := 0$.
2. Представить матрицу A_k в виде произведения унитарной матрицы Q_k и верхнетреугольной матрицы R_k. $Q_k R_k = A_k$
3. Вычислить $A_{k+1} := R_k Q_k$
4. Увеличить k на единицу. $k := k + 1$
5. Повторить шаги 2-4 до тех пор пока внедиагональные элементы матрицы A_k не станут близкими к нулю.
6. Значения на диагонали матрицы A_k будут являться приближением собственных значений матрицы A.

2.1 Доказательство корректности алгоритма

Заметим, что все матрицы A_k для $k = 0, 1, \ldots$ являются подобными, т.к. $A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k = Q_k^T A_k Q_k$, а значит их собственные значения совпадают.

Также для положительно определённой матрицы A известно [1], что внедиагольные элементы матрицы A_k будут стремиться к нулю при $k \to \infty$.

3 Вычислительные эксперименты

QR алгоритм был реализован тремя различными способами на языке Python 3 при помощи библиотеки Numpy для использования векторизации вычислений, после чего лучшая из имплементаций была ускорена при помощи JIT-компилятора Numba. Также алгоритм был имплементирован на языке программирования C++, но без использования параллельных вычислений. Результаты измерений для различных размеров матрицы A приведены в таблице 1. Все имплементации выложены в открытый доступ на Github[1].

Имплементация	$N = 10$	$N = 20$	$N = 30$	$N = 40$	$N = 50$
Numpy 1	4.6 ± 0.1	30.9 ± 0.9	138.2 ± 2.1	119.5 ± 1.8	290.5 ± 10.1
Numpy 2	4.7 ± 0.2	30.1 ± 0.5	139.2 ± 3.5	117.0 ± 2.8	290.8 ± 4.5
Numpy 3	5.4 ± 0.3	32.0 ± 1.3	132.5 ± 2.5	113.5 ± 3.9	262.3 ± 6.9
Numba JIT	0.6 ± 0.0	4.2 ± 0.1	24.8 ± 0.5	27.3 ± 0.3	83.4 ± 1.0
Numbda JIT с доп. пар.	0.6 ± 0.0	4.2 ± 0.0	24.0 ± 0.1	26.6 ± 0.2	79.0 ± 1.0
Numba guvectorize	0.5 ± 0.0	4.2 ± 0.1	24.7 ± 0.2	$\mathbf{26.3 \pm 0.4}$	$\mathbf{78.4 \pm 0.5}$
C++	$\mathbf{0.3 \pm 0.0}$	$\mathbf{3.1 \pm 0.1}$	$\mathbf{21.5 \pm 0.2}$	30.3 ± 0.9	87.8 ± 0.5

Таблица 1: Время выполнения (в миллисекундах) QR алгоритма различных имплементаций при различных размерах исходной матрицы. В таблице приведено среднее время выполнения ± среднеквадратичное отклонение времени по 7 замерам времени, в каждом из которых алгоритм запускался 100 раз.

3.1 Анализ полученных результатов

Полученные результаты показывают, что JIT-компилятор Numba позволяет добиться ускорения примерно в 4 раза в сравнении с обычным использованием Numpy,

[1] `https://github.com/CrafterKolyan/eigenvalues-speed-comparison/blob/main/experiments/`

выигрывая по скорости на больших матрицах даже у C++. Что не менее важно, для того, чтобы воспользоваться JIT-компилятором, не требуется тратить дополнительное время на разработку решения. Также видно, что при этом компилятор Numba даже не нужно дополнительно настраивать, т.к. в любом случае получается выигрыш в скорости работы.

Кроме того, отметим, что библиотечная функция из Numpy (`numpy.linalg.eigvals`) работает примерно в 10 раз быстрее, чем самая быстрая из реализаций представленных в работе, но неизвестно какой конкретно алгоритм применяется в библиотечной функции, поэтому сравнение скоростей работы функции из библиотеки и представленных имплементаций может быть некорректно.

4 Заключение

В процессе выполнения работы были получены следующие результаты:

- Использование JIT-компилятора Numba позволяет получить повышение скорости работы в 4 раза по сравнению с обычным Numpy при реализации QR алгоритма;
- JIT-компилятор Numba вместе с использованием Numpy позволяет получить более быструю реализацию QR алгоритма на больших матрицах, нежели чем аналогичная реализация на C++ без параллелизации, кроме того, время затраченное на написание кода с использованием Numba и Numpy сильно меньше, чем реализация аналогичного решения на C++.

Список литературы

[1] *Olver Peter J.* Orthogonal bases and the QR algorithm. — 2010. — Pp. 25–26.