Методы вычисления SVD
Численная реализация

Singular Matrixmen

Кафанов Степан
Дворянков Роман
Иудин Егор
Мусаева Асият

Анализ методов вычисления SVD

- Методы Якоби (односторонний и двухсторонний)
- Метод разделяй и властвуй
- Рандомизированный SVD

- Построение набора матриц для сравнения
- Сравнение методов по времени и по ошибке
- Для измерение качества выбрана норма Фробениуса

Рандомизированный метод

Цель: построить матрицу Q с небольшим числом попарно ортогональных векторов, которая будет приближать матрицу A с заданной точностью

$$\|(I - QQ^*)A\| < \varepsilon$$

и далее рассматривать SVD-разложение матрицы Q^*A

Поиск желаемого Q производится путем генерации Гауссовых случайных векторов и нахождения для них ОНБ

Рандомизированный метод

Пусть нам удалось найти такую матрицу Q

$$A \approx QQ^*A$$

Рассмотрим следующую матрицу и ее SVD-разложение

$$B = Q^*A \qquad B = \tilde{U}\Sigma V^*$$

Определим

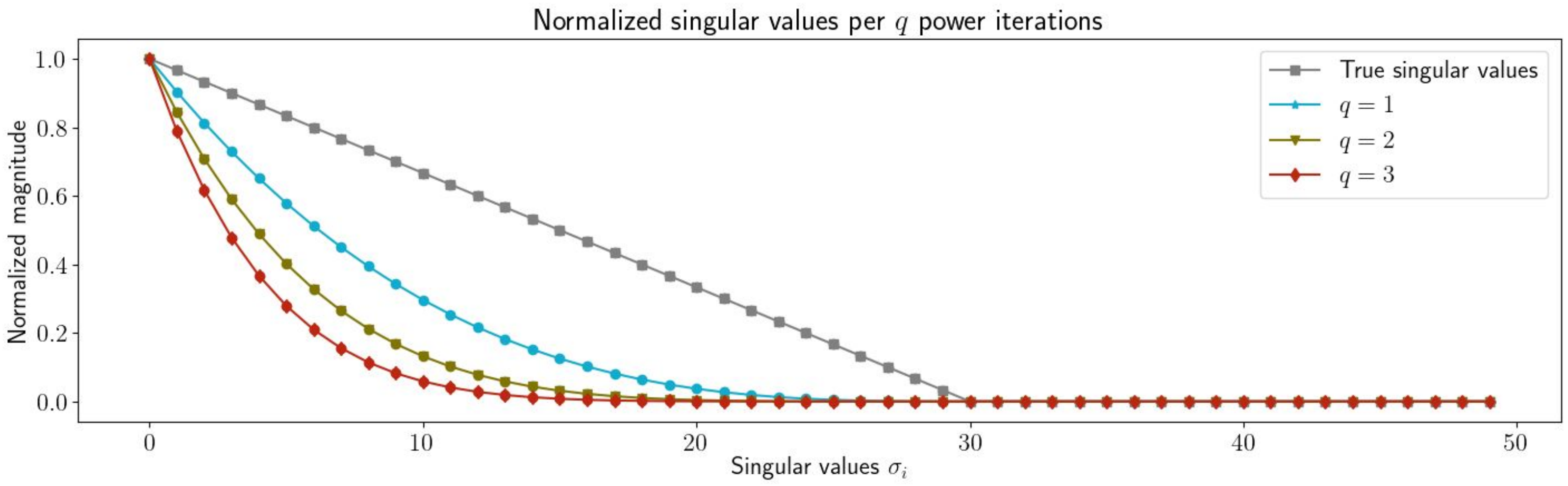
$$U = Q\tilde{U}$$

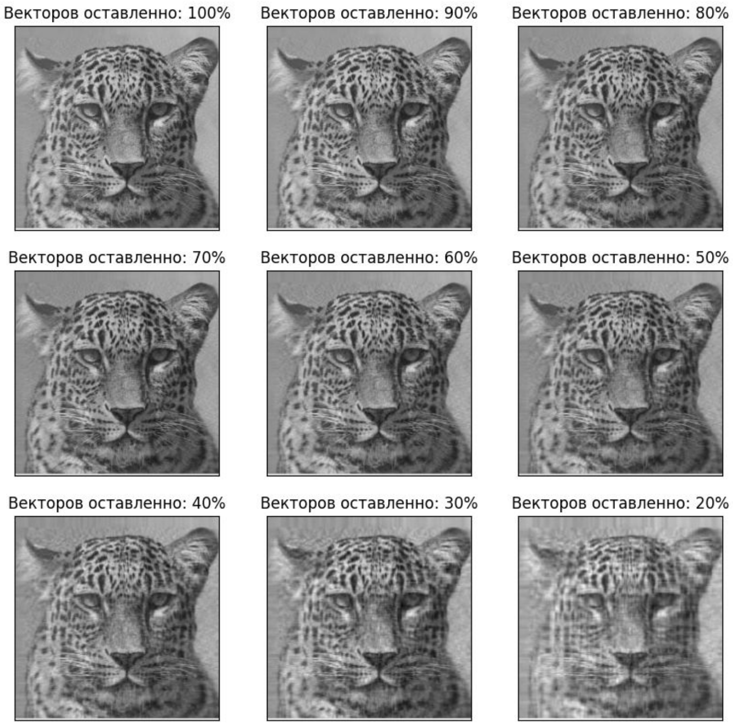
Тогда полученные матрицы U, Σ, V^* являются разложение *A*

Рандомизированный метод

Методы улучшений результата:

- Oversampling
- Power Iterations
- автоопределение оптимального ранга проекционной матрицы



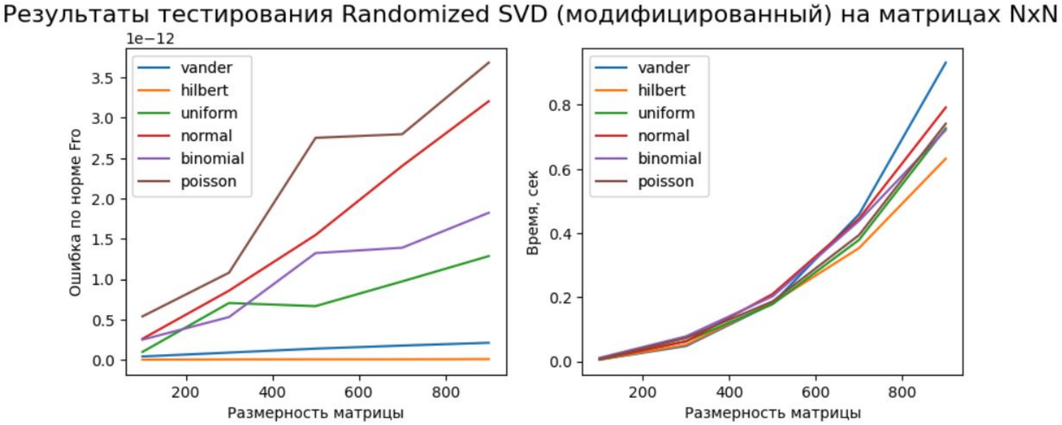


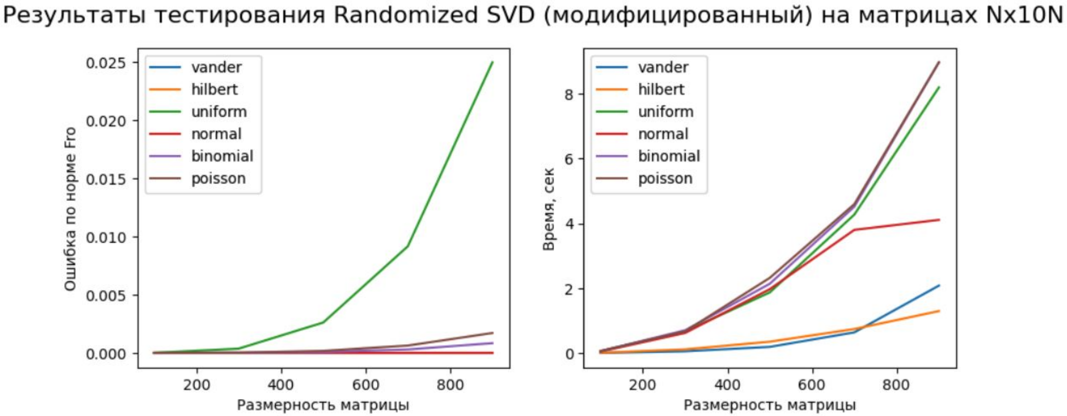
Изменение качества сжатия при увеличении доли отсекаемых векторов

Модифицированный рандомизированный метод

Среднее значение порядков чисел обусловленности для матриц NxN

- vander – 1e22
- hilbert – 1e21
- uniform – 1e4
- normal – 1e3
- binomial – 1e4
- poisson – 1e4

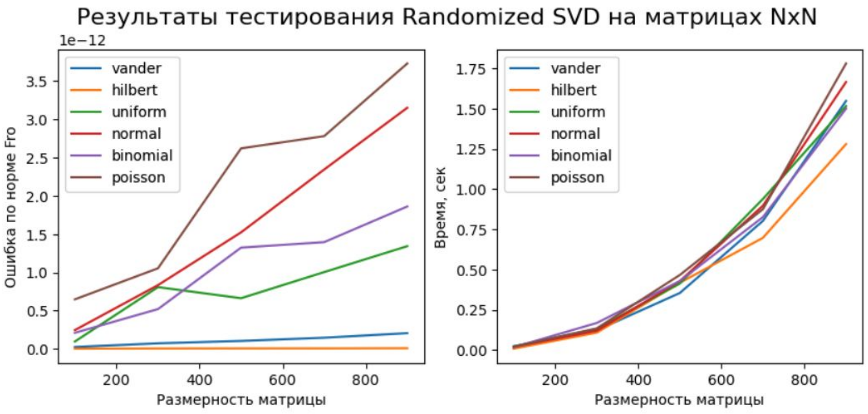


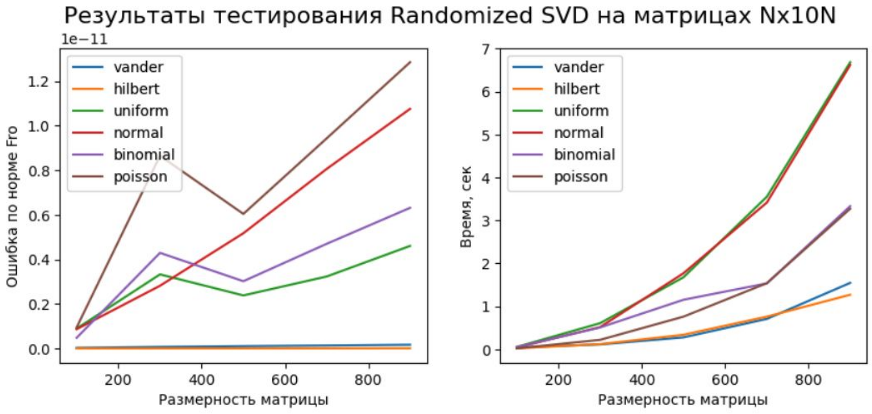


Обычный рандомизированный метод

Среднее значение порядков чисел обусловленности для матриц NxN

- vander – 1e22
- hilbert – 1e21
- uniform – 1e5
- normal – 1e3
- binomial – 1e4
- poisson – 1e5





Метод Якоби

Jacobi-Kogbetliznt algorithm (double-sided Jacobi method)

$$\begin{pmatrix} \alpha & 0 & -\beta & 0 \\ 0 & 1 & 0 & 0 \\ \beta & 0 & \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^T \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \begin{pmatrix} \alpha & 0 & -\beta & 0 \\ 0 & 1 & 0 & 0 \\ \beta & 0 & \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Формула приведения к диагональному виду:

$$\Sigma = J_{lk} J_{l(k-1)} \ldots J_{l0} A J_{r0} \ldots J_{r(k-1)J_{rk}} = U^* A V$$

$$U^* = J_{lk} J_{l(k-1)} \ldots J_{l0}$$

$$V = J_{r0} \ldots J_{r(k-1)} J_{rk}$$

Метод Якоби

Jacobi-Hestenes algorithm (single-sided Jacobi algorithm)

Каждая итерация соответствует ортогонализации двух векторов матрицы, полученной на предыдущем шаге

$$A_{(i)} = A_{(i-1)} J_{l(i-1)}$$

Формула приведения к диагональному виду:

$$A J_0 \ldots J_{l(k-1)} = B$$

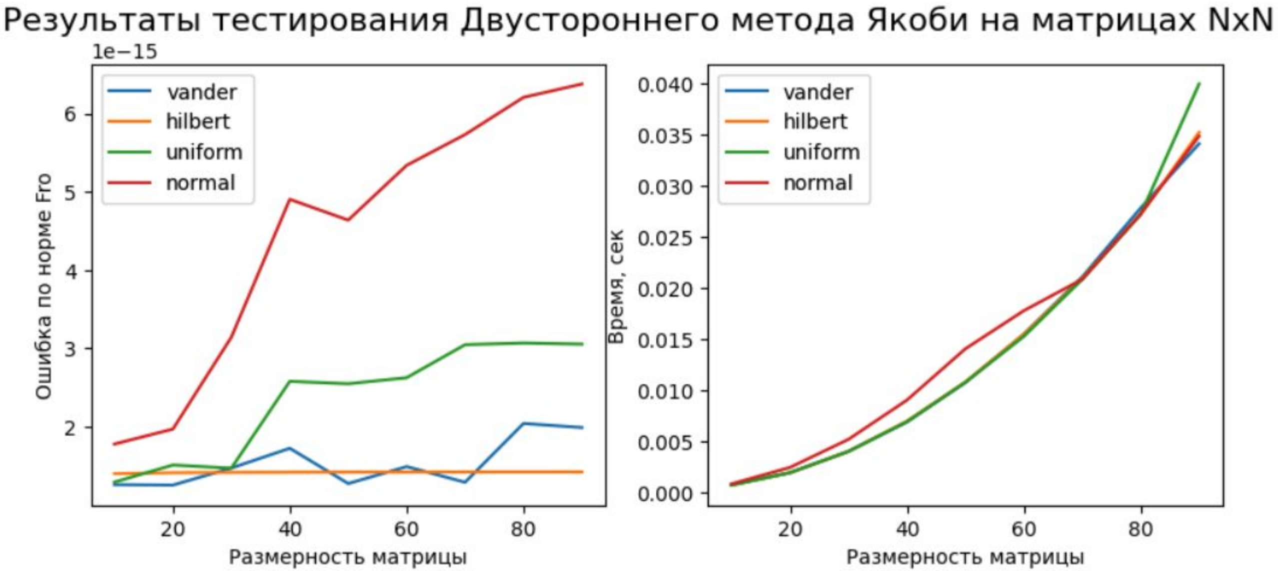
$$B = U\Sigma$$

$$A = U\Sigma V^*$$

Метод Якоби

Среднее значение порядков чисел обусловленности для матриц NxN

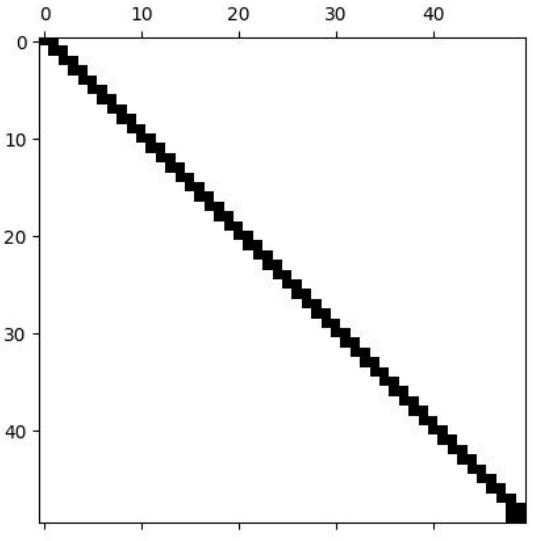
- vander – 1e19
- hilbert – 1e18
- uniform – 1e3
- normal – 1e2



Метод разделяй и властвуй

Шаг первый: приводим матрицу к трехдигоальной матрице, каждая итерация соответствует умножению слева и справа на матрицы Хаусхолдера, работающие с i-ым столбцом и i-ой строкой:

$$A_{(i)} = H_i A_{(i-1)} G_i$$



Формула приведения:

$$B = U_1^T A V_1$$

$$U_1 = H_1 H_2 \ldots H_n$$

$$V_1 = G_1 G_2 \ldots G_{n-1}$$

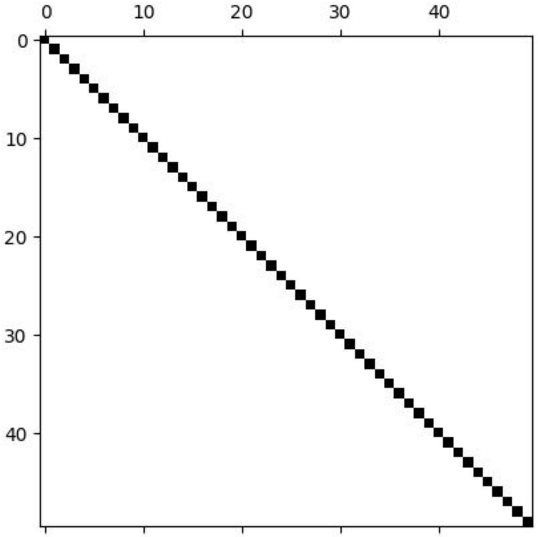
Метод разделяй и властвуй

Шаг второй: разбиваем полученную на вход матрицу на две подматрицы, если она большая и выполняем этот шаг с двумя подматрицами, иначе переходим к следующему шагу.

$$T = \begin{bmatrix} T_1' & B \\ B^\top & T_2' \end{bmatrix} \rightarrow T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^*$$

Метод разделяй и властвуй

Шаг третий: для полученных подматриц применяем QR алгоритм со сдвигом, сам сдвиг вычисляется следующим образом:



$$\mu = a_m - \frac{\operatorname{sign}(\delta) b_{m-1}^2}{\left(|\delta| + \sqrt{\delta^2 + b_{m-1}^2}\right)}$$

$$B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix} \quad \delta = \frac{(a_{m-1} - a_m)}{2}$$

Метод разделяй и властвуй

Шаг четвертый: после того, как все маленькие матрицы были диагализованы, необходимо их объединить:

$$T_1 = Q_1 \Lambda_1 Q_1^*, \quad T_2 = Q_2 \Lambda_2 Q_2^* \quad \longrightarrow$$

$$\begin{bmatrix} Q_1^* & 0 \\ 0 & Q_2^* \end{bmatrix} T \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} = D + \rho u u^*, \quad D = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$$

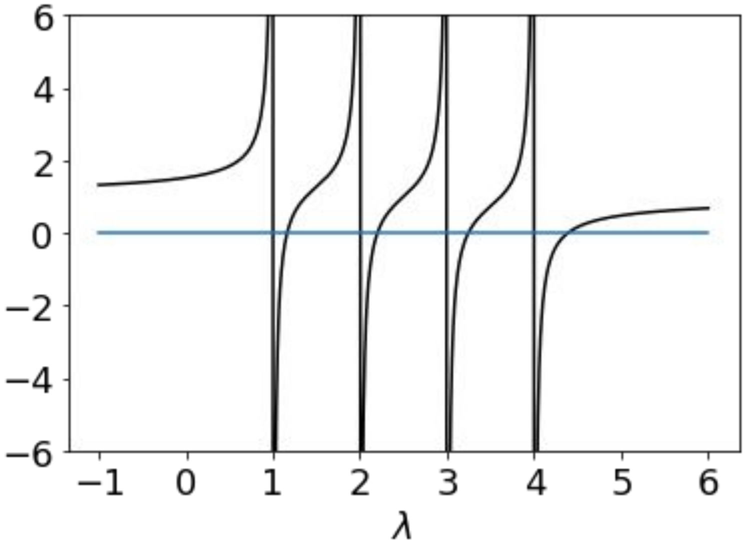
В итоге осталось найти СЗ и СВ для матрицы вида диагональная матрица плюс матрица малого ранга и вернуть результат на этап рекурсии выше.

Метод разделяй и властвуй

Шаг пятый: ищем СЗ и СВ, для этого мы должны решить уравнение:

$$\det(D + \rho uu^* - \lambda I) = \det(D - \lambda I)\det(I + \rho(D - \lambda I)^{-1}uu^*) = 0.$$

$$\det(I + \rho(D - \lambda I)^{-1}uu^*) = 1 + \rho \sum_{i=1}^{n} \frac{|u_i|^2}{d_i - \lambda} = 0$$



Это уравнение называется вековым.

Метод разделяй и властвуй

Решение векового уравнения следует производить, приближая его подходящей функцией, стандартный Ньютон плохо подходит для этой задачи.

Лучше, например, приближать “гиперболой” : $$f(\lambda) \approx c_0 + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda}$$

При этом мы пользуемся тем фактом, что СЗ лежат между d_i и d_{i+1}.

Для каждого СЗ получаем его СВ в виде $(D - \alpha_i I)^{-1} u$ и отнормируем его.

Сложности представленных методов

Метод Якоби вычисления сингулярных чисел и векторов	
Последовательный алгоритм	
Последовательная сложность	$O(n^3)$
Объём входных данных	n^2
Объём выходных данных	$2n^2 + n$
Параллельный алгоритм	
Высота ярусно-параллельной формы	$O(n^2)$
Ширина ярусно-параллельной формы	$O(n)$

Метод "Разделяй и властвуй" вычисления собственных значений и векторов трёхдиагональной матрицы	
Последовательный алгоритм	
Последовательная сложность	$c\frac{4}{3}n^3$
Объём входных данных	$2n + 1$
Объём выходных данных	$n(n+1)$
Параллельный алгоритм	
Параллельная сложность	$O(n^{2.3})$ крайне редко: $O(n^2)$

Планы

- Модернизировать Random
- Устранить проблемы, возникшие в методе разделяй и властвуй

Выводы

- Как и ожидалось, методы Якоби плохо масштабируются и при увеличении размеров матрицы затрачивают существенное время
- Модифицированный рандомизированный SVD плохо работает на равномерно распределенных матрицах

Ссылка на гитхаб

- https://github.com/Mr-Grag-Universe/NLA_project.git