Московский Авиационный Институт
(Национальный Исследовательский Университет)

Курсовой проект по курсу
«Численные методы»

SVD-разложение и его практические приложения

Студент:
Колесников Е. В.

Преподаватель:
Абгарян К. К.

21 ноября 2015 г.

Содержание

1 Введение

1.1 Краткое описание SVD-разложения

Одной из самых плодотворных идей в теории матриц является матричное разложение или каноническая форма. В последнее время матричные разложения стали оплотом численных методов линейной алгебры, которые служат основой решения множества проблем.

Из многочисленных разложений матриц, сингулярному разложению, которое представляет из себя факторизацию матрицы $\mathbf{A}$ в произведение $\mathbf{U\Sigma V^H}$, где $\mathbf{U}, \mathbf{V}$ – унитарные матрицы, а $\mathbf{\Sigma}$ – диагональная матрица, отводится особое место. На это есть несколько причин:

- Во-первых, тот факт, что в разложении участвуют унитарные матрицы делает его идеальным механизмом для геометризации преобразования $\mathbf{A}$ в n-мерном пространстве.
- Во-вторых, сингулярное разложение является устойчивым, т.е. малым возмущениям матрицы $\mathbf{A}$ соответствуют малые возмущения матрицы $\mathbf{\Sigma}$ и наоборот.
- В-третьих, диагональная матрица $\mathbf{\Sigma}$ позволяет легко понять является ли матрица A почти вырожденной и, если она таковой является, сингулярное разложение дает возможность понизить ранг матрицы $\mathbf{A}$ с наименьшей погрешностью.
- Последним по порядку, но не по значению является тот факт, что благодаря усилиям математика Gene Golub были получены эффективные, устойчивые алгоритмы вычисления сингулярного разложения на ЭВМ, которые используются уже на протяжении более 40 лет и включены во все математические пакеты сегодняшнего времени.

1.2 Историческая справка

Интригующим является тот факт, что большинство классических матричных разложений были получены задолго до использования матриц и матричной нотации: они задавались с помощью определителей, линейных систем уравнений и, особенно, билинейных и квадратичных форм. Отцом развития данного направления исследований стал Gauss. Один из его алгоритмов, разработанный им в 1823 году, раскладывает матрицу квадратичной формы $\mathbf{x^T Ax}$ в произведение $\mathbf{RD^{-1}R}$, где $\mathbf{D}$ – диагональная матрица, а $\mathbf{R}$ – верхнетреугольная матрица с элементами матрицы $\mathbf{D}$ на главной диагонали. Gauss также разработал алгоритм для эффективного нахождения обратной матрицы, который преобразовывал систему вида $\mathbf{y = Ax}$ к системе вида $\mathbf{x = By}$. Его умение манипулировать квадратичными формами и системами уравнений позволили ему разработать метод наименьших квадратов, который лег в основу методов машинного обучения около 30 лет назад.

Cauchy в 1829 году установил свойства собственных значений и собственных векторов симметричных матриц, рассматривая соответствующие им однородные системы уравнений. В 1846 году Jacobi опубликовал алгоритм диагонализации симметричных матриц, а в посмертной статье 1857 года он получил LU-разложение билинейных форм. Weierstrass в 1868 году получил канонические формы для пар билинейных функций – то, что в наше время называется обобщенной проблемой собственных значений: $\mathbf{Ax} = \mu\mathbf{Bx}$, где $\mathbf{B}$ – некоторая матрица. Подводя итог, иожно сказать, что разработка сингулярного разложения – результат, полученный в результате многолетних исследований билинейных форм.

Сингулярное разложение было первоначально разработано в дифференциальной геометрии при изучении свойств билинейных форм учеными Eugenio Beltrami и Camille Jordan независимо в 1873 и 1874 годах соответственно. James Joseph Sylvester пришел к понятию сингулярного разложения для квадратных матриц в 1889 году и, вероятно, также независимо от Eugenio Beltrami и Camille Jordan. Sylvester называл сингулярные значения каноническими множителями матрицы. Четвертый математик, который открыл сингулярное разложение независимо от других – Autonne в 1915 году, который пришел к сингулярному разложению с помощью полярного разложения. Первое доказательство сингулярного разложения для прямоугольных и комплексных матриц было осуществлено математиками Carl Eckart и Gale Young в 1936 году.

В 1907 году Erhard Schmidt определил аналог сингулярного разложения для интегральных операторов, не зная о том, что параллельно ведется работа над сингулярным разложением для конечномерных матриц. Его теория была дальше развита математиком Émile Picard в 1910 году. Именно Picard первым назвал σ_k сингулярными значениями.

Практические методы вычисления SVD-разложения восходят к работам Kogbetliantz в 1954, 1955 и Hestenes в 1958, алгоритм которых сильно напоминает метод Якоби для вычисления собственных значений с использованием поворотов Гивенса. Однако этот алгоритм был заменен методом, разработанным математиками Gene Golub и William Kahan в 1965, который использует преобразования Хаусхолдера. В 1970 году Golub и Christian Reinsch опубликовали вариант алгоритма Golub/Kahan, который до сих пор является одним из самых используемых.

2 Теория

2.1 Beltrami

Beltrami и Jordan считаются основателями теории сингулярного разложения. Beltrami – за то, что он первым опубликовал работу о сингулярном разложении, а Jordan – за элегантность и полноту своей работы. Работа Beltrami появилась в журнале “Journal of Mathematics for the Use of the Students of the Italian Universities” в 1873 году, основная цель которой заключалась в том, чтобы ознакомить студентов с билинейными формами.

2.1.1 Вывод SVD-разложения

Beltrami начинает свое доказательство с рассмотрения билинейной формы

$$f(x, y) = x^T A y,$$

где A – действительная квадратная матрица n-го порядка. Если сделать замену вида:

$$x = U\xi \text{ и } y = V\eta,$$

то билинейную форму можно переписать в виде:

$$f(x, y) = \xi^T S \eta,$$

где

$$S = U^T A V. \tag{2.1.1}$$

Если потребовать, чтобы матрицы U, V были ортогональными, то существует всего $n^2 - n$ степеней свободы при их задании, которые можно использовать, чтобы обнулить внедиагональные элементы матрицы S.

Докажем это. Будем выяснять количество параметров, задающих ортогональный базис или, что то же самое, элементы ортогональной матрицы. Рассмотрим 2-мерное пространство. Единичный вектор в двумерном пространстве имеет всего 1 степень свободы – угол вращения вокруг начала координат, т.к. длина у него постоянна. Учитывая тот факт, что второй ортогональный вектор, перпендикулярный первому задается однозначно с точностью до знака, но знак не является степенью свободы, мы приходим к выводу, что репер в 2-мерном пространстве задается одной степенью свободы. Рассмотрим 3-мерное пространство. Единичный вектор в нем задается с помощью 2-ух степеней свободы (широта и долгота), а остальные 2 вектора лежат в плоскости, перпендикулярной этому вектору, т.е. в 2-мерном пространстве, которое уже было нами рассмотрено. Т.о. репер в 3-мерном пространстве задается с помощью $2 + 1 = 3$ степеней свободы. Продолжая размышления по индукции, мы приходим к выводу, что репер в n-мерном пространстве задается с помощью $1 + 2 + ... + (n-1) = \dfrac{n(n-1)}{2}$ степеней свободы. Полученный результат показывает, что одна ортогональная матрица задается с помощью $\dfrac{n(n-1)}{2}$ степеней свободы, а так как в нашем случае 2 такие матрицы, то мы имеем $n^2 - n$ степеней свободы при их задании.

Предположим, что S – диагональная матирца, т.е. $S = \Sigma = diag(\sigma_1, ..., \sigma_n)$. Тогда из (2.1.1) и ортогональности V, U следует, что

$$U^T A = \Sigma V^T \tag{2.1.2}$$
$$AV = U\Sigma \tag{2.1.3}$$

Выражая из (2.1.3) отдельно матрицы U и V и, подставляя полученные выражения в (2.1.2), получаем уравнения

$$U^T(AA^T) = \Sigma^2 U^T \tag{2.1.4}$$
$$(A^T A)V = V\Sigma^2 \tag{2.1.5}$$

Покажем, что σ_i – корни уравнений

$$\det(AA^T - \sigma^2 I) = 0, \tag{2.1.6}$$
$$\det(A^T A - \sigma^2 I) = 0. \tag{2.1.7}$$

Для этого покажем, что набор собственных значений инвариантен относительно преобразования подобия $B = U\Lambda U^{-1}$:

$$\det(B - \mu I) = \det(U\Lambda U^{-1} - \mu I) = \det(U(\Lambda - \mu I)U^{-1}) =$$
$$= \det(\Lambda - \mu I) \cdot \det(U) \cdot \det(U)^{-1} = \det(\Lambda - \mu I).$$

Покажем, что (2.1.6), (2.1.7) являются эквивалентными алгебраическими уравнениями. Для этого необходимо доказать, что множество σ_i^2 является множеством собственных значений матриц AA^T и $A^T A$.

Предположим, что λ – собственное значение матрицы $A^T A$, т.е.

$$A^T Ax = \lambda x.$$

Тогда, домножив обе части на A, мы получим следующее выражение:

$$AA^T Ax = \lambda Ax.$$

Если сделать замену переменных $y := Ax$, то мы докажем, что если λ – собственное значение матрицы A^TA, то оно также будет и собственным значением матрицы AA^T:

$$AA^Ty = \lambda y.$$

Таким образом, мы показали, что (2.1.6), (2.1.7) являются эквивалентными алгебраическими уравнениями, т.е. корни σ_i будут одинаковыми в обоих случаях.

Необходимо заметить, что вывод, предложенный Beltrami предполагает, что матрица Σ, а следовательно и матрица A – невырожденные.

Т.к. матрицы A^TA и AA^T симметричные и положительно определенные, то они имеют полный набор действительных, положительных собственных значений.

На основании вышеизложенных размышлений Beltrami написал алгоритм нахождения сингулярного разложения:

1. Найти корни уравнения (2.1.6).
2. Определить U из (2.1.4). Здесь Beltrami добавляет, что столбцы U определены с точностью до фактора ± 1. Этот факт и то, что матрица U ортогональна, справедливы только в том случае, когда все σ_i различные.
3. Определить V из (2.1.2). Этот шаг требует, чтобы Σ была невырожденной.

Таким образом Beltrami вывел SVD-разложение для действительных, квадратных, невырожденных матриц с различными сингулярными значениями.

2.2 Sylvester

Sylvester написал 2 статьи по сингулярному разложению. В первой статье он привел итеративный алгоритм редуцирования квадратичной формы к диагональному виду, а в примечании к статье указал, что аналогичный итерационный метод может диагонализовать билинейную форму. Во-второй статье он написал правило, по которому задача диагонализации билинейной формы сводилась к диагонализации квадратичной формы. Это правило оказалось ничем иным, как алгоритмом Beltrami.

2.2.1 Правило нахождения сингулярного разложения

Sylvester начинает с билинейной формы

$$B(x, y) = x^TAy$$

и рассматривает квадратичную форму

$$M(x) = \sum_i \left(\frac{dB}{dy_i}\right)^2,$$

которая является квадратичной формой вида

$$M(x) = x^TAA^Tx.$$

Пусть $M = \sum \lambda_i \xi_i^2$ – каноническая форма матрицы M. Если B имеет каноническую форму $B = \sum \sigma_i \xi_i \eta_i$, тогда $\sum(\sigma_i \xi)^2$ подобно $M = \sum \lambda_i \xi_i^2$, из чего следует, что $\lambda_i = \sigma_i^2$.

Чтобы найти змену, Sylvester вводит матрицы $M = AA^T$ и $N = A^TA$ и утверждает, что замена x – такая замена, которая диагонализирует матрицу M, а замена для y – такая замена, которая диагонализирует матрицу N. Это верно только в том случае, когда все сингулярные значения матрицы A различны.

2.3 Autonne

2.3.1 Полярное разложение

Теорема: Любую матрицу A можно представить в виде произвдения ортогональной матрицы Q и симметричной матрицы S:

$$A = QS, \tag{2.3.1}$$

причем все собственные векторы симметричной матрицы S – неотрицательные числа.

Геометрический смысл: любое линейное преобразование может быть представлено в виде двух последовательных преобразований:

1. масштабирование вдоль ортогональных направлений на неотрицательные коэффициенты
2. последовательность поворотов с возможными отражениями между ними

Для того, чтобы доказать эту теорему, приведем пример алгоритма для построения данного разложения и покажем, что матрицы Q и S уникальны для каждой матрицы A.

$$\begin{aligned} &A^T A = SQ^T QS = S^2 \\ &S = \sqrt{A^T A} \end{aligned} \tag{2.3.2}$$

Зная S и A можно найти Q:

$$Q = AS^{-1} \tag{2.3.3}$$

Матрицы Q и S уникальны для каждой матрицы A. Покажем, что матрица Q является ортогональной:

$$\begin{aligned} &Q^T Q = S^{-1} A^T A S^{-1} = S^{-1} S^2 S^{-1} = I \\ &QQ^T = AS^{-1} S{-}1A^T = A(S^2)^{-1} A^T = AA^{-1}(A^T)^{-1} A^T = I \end{aligned}$$

2.3.2 Вывод SVD-разложения

Воспользуемся полярным разложением и представим матрицу A в виде:

$$A = QS$$

Воспользуемся свойством спектрального разложения симметричных матриц:

$$S = X\Lambda X^T$$

Симметричные матрицы имеют полный набор собственных значений, а собственные векторы могут быть выбраны ортогональными. Таким образом, приходим к следующей формуле:

$$A = QX\Lambda X^T$$

В данной формуле Q,X – ортогональные матрицы, поэтому их произведением является также ортогональная матрица $Y = QX$.

Подставляя все в одну формулу, получаем SVD-разложение:

$$A = Y\Lambda X^T \tag{2.3.4}$$

Или более традиционная запись:

$$A = U\Sigma V^T \tag{2.3.5}$$

Коэффициенты $\sigma_1, ..., \sigma_n$, стоящие на главной диагонали матрицы Σ, называются сингулярными значениями и эквивалентны собственным значениям матрицы $A^T A$.

2.4 Сингулярные векторы

SVD-разложение – аналог спектрального разложения, который работает с любыми матрицами.

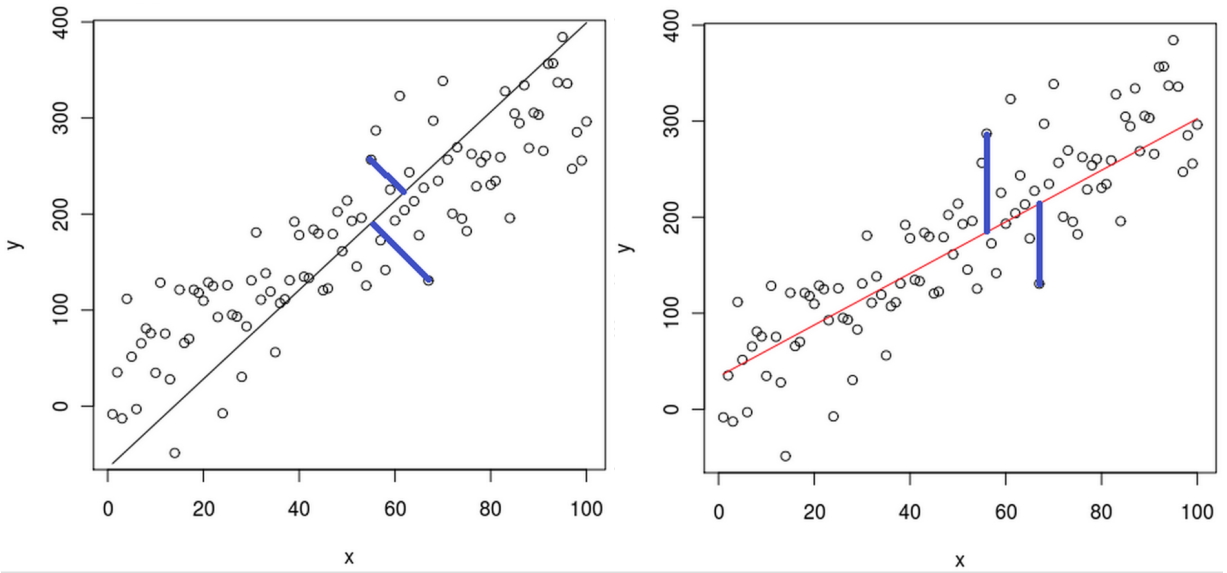
Столбцы матрицы V называются правыми сингулярными векторами и всегда ортогональны друг другу. Столбцы матрицы U называются левыми сингулярными векторами и также ортогональны друг другу.

Чтобы лучше понять, что такое SVD-разложение, представим матрицу размеров $n \times d$ как n точек $\{a^{(i)}\}_{i=1}^{n}$ в d-мерном пространстве и поставим задачу нахождения наилучшего k-размерного подпространства относительно этих n точек.

$$A_{n\times d} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} & \cdots & a_d^{(1)} \\ a_1^{(2)} & a_2^{(2)} & \cdots & a_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{(n)} & a_2^{(n)} & \cdots & a_d^{(n)} \end{pmatrix}$$

В данном контексте “наилучшим” называется пространство с минимальным расстоянием до этих точек. Альтернативной задачей является задача наилучшего приближения, которая обычно решается с помощью МНК. В задаче наилучшего приближения минимизируется вертикальная ошибка, в то время как при нахождении наилучшего подпространства минимизируется перпендикулярное расстояние до точек.

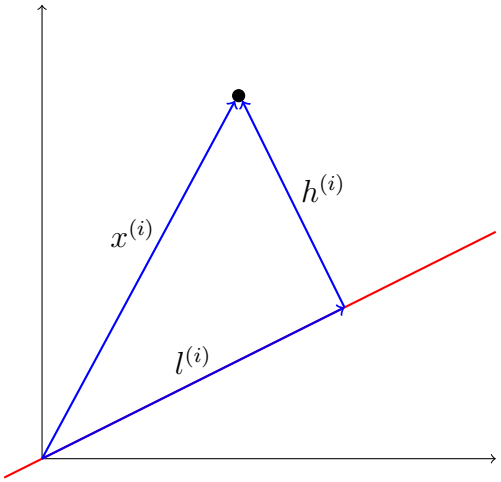
Рис. 1: Минимизируемые ошибки



Рассмотрим проекцию точки $x^{(i)}$ на подпространство, проходящее через начало координат:

$$\sum_{k=1}^{d} (x_k^{(i)})^2 = (l^{(i)})^2 + (h^{(i)})^2,$$

где $l^{(i)}$ – проекция точки $x^{(i)}$ на подпространство, а $h^{(i)}$ – перпендикуляр от точки до подпространства.



$$(h^{(i)})^2 = \sum_{k=1}^{d}(x_k^{(i)})^2 - (l^{(i)})^2$$

Чтобы минимизировать сумму расстояний от точек до прямой можно минимизировать $\sum_{i=1}^{n}\left[\sum_{k=1}^{d}(x_k^{(i)})^2 - (l^{(i)})^2\right]$, однако т.к. $\sum_{i=1}^{n}\sum_{k=1}^{d}(x_k^{(i)})^2$ – константа, то задача минимизации суммы расстояний от точек до прямой эквивалентна задаче максимизации суммы проекций на прямую $\sum_{i=1}^{n}(l^{(i)})^2$.

Определим теперь сингулярные векторы матрицы $A_{n\times d}$. Представим прямую, проходящую через начало координат. Предположим, что $\mathbf{v}$ – единичный вектор вдоль этой прямой. Длину проекции $\mathbf{a^{(i)}}$ (i-ой строки матрицы A) на $\mathbf{v}$ можно представить в виде $|\mathbf{a^{(i)}} \cdot \mathbf{v}|$. Отсюда видно, что сумму квадратов проекции можно представить в виде $||A\mathbf{v}||^2$. Следовательно, наилучшая прямая будет та, которая максимизирует $||A\mathbf{v}||^2$ (по т. Пифагора) и, соответственно, минимизирует сумму квадратов расстояний от точек до прямой.

На основе предыдущих размышлений определим первый собственный вектор $\mathbf{v_1}$ матрицы A, который является вектором-столбцом, как наилучшую прямую, заданную этим вектором, для n точек в d-мерном пространстве.

$$\mathbf{v_1} = \underset{||\mathbf{v}||=1}{\operatorname{argmax}} ||A\mathbf{v}||$$

Значение $\sigma_1(A) = ||A\mathbf{v_1}||$ называется первым сингулярным значением матрицы A. Отметим также, что σ_1^2 – сумма квадратов проекций точек на прямую, заданную вектором $\mathbf{v_1}$.

"Жадный" подход к нахождению 2-мерного подпространства для матрицы A берет $\mathbf{v_1}$ в качестве первого базисного вектора и находит наилучшее 2-мерное подпространство, содержащее $\mathbf{v_1}$. Для любого двумерного подпространства сумма квадратов проекций точек эквивалентна сумме квадратов проекций на $\mathbf{v_1}$ плюс сумма квадратов проекций на $\mathbf{v_2}$. Нет заранее никакой гарантии, что "жадный" подход даст наилучшее решение, однако в данном случае это так.

Второй сингулярный вектор находится следующим образом:

$$\mathbf{v_2} = \underset{\mathbf{v}\perp\mathbf{v_1},||\mathbf{v}||=1}{\operatorname{argmax}} ||A\mathbf{v}||$$

Значение $\sigma_2(A) = ||A\mathbf{v_2}||$ называется вторым сингулярным значением матрицы A.

Аналогично находятся и все оставшиеся векторы d-мерного пространства:

$$\mathbf{v_{k+1}} = \underset{\mathbf{v} \perp \mathbf{v_1}, ..., \mathbf{v_k}, ||\mathbf{v}||=1}{\operatorname{argmax}} ||A\mathbf{v}||$$

Докажем теорему о том, что жадный алгоритм в действительности находит наилучшее подпространство любой размерности.

Пусть A – матрица размерности $n \times d$, где $\mathbf{v_1}, ..., \mathbf{v_r}$ – сингулярные векторы, которые заданы выше. Для $1 \leq k \leq r$, пусть V_k – подпространство, порожденное базисными векторами $\mathbf{v_1}, ..., \mathbf{v_r}$. Тогда для любого k, V_k является наилучшим k-размерным подпространством A.

Будем проводить доказательство по индукции. Для $k = 1$ утверждение, очевидно, выполняется. Предположим, что V_{k-1} – наилучшее $(k-1)$-мерное подпространство. Предположим, что W_k – наилучшее k-мерное подпространство. Выберем базис $\mathbf{w_1}, ..., \mathbf{w_k}$ таким образом, чтобы $\mathbf{w_k}$ было перпендикулярно V_{k-1}. Тогда

$$\sum_{i=1}^{k} ||A\mathbf{w_i}||^2 \leq \sum_{i=1}^{k-1} ||A\mathbf{v_i}||^2 + ||A\mathbf{w_k}||^2,$$

т.к. V_{k-1} – оптимальное $(k-1)$-мерное подпространство. Т.к. $\mathbf{w_k}$ перпендикулярно $\mathbf{v_1}, ..., \mathbf{v_{k-1}}$, то по определению $\mathbf{v_k}$: $||A\mathbf{w_k}||^2 \leq ||A\mathbf{v_k}||^2$. Таким образом

$$\sum_{i=1}^{k} ||A\mathbf{w_i}||^2 \leq \sum_{i=1}^{k} ||A\mathbf{v_i}||^2,$$

что показывает, что V_k по крайней мере не хуже W, а следовательно – оптимально.

Заметим, что n-вектор $A\mathbf{v_i}$ является списком длин (со знаком) проекций строк матрицы A на вектор $\mathbf{v_i}$. Будем смотреть на $||A\mathbf{v_i}|| = \sigma_i(A)$ как на некую "компоненту" матрицы A вдоль вектора $\mathbf{v_i}$. Для того, чтобы такая интерпретация имела смысл, необходимо чтобы сумма квадратов "компонент" матрицы A вдоль каждой $\mathbf{v_i}$ давала бы квадрат всего "содержимого" матрицы A. Это на самом деле выполняется и является матричной аналогией разложения вектора по компонентам ортогонального базиса.

Рассмотрим строку $\mathbf{a_j}$ матрицы A. Т.к. $\mathbf{v_1}, ..., \mathbf{v_r}$ образует линейную оболочку всех строк матрицы A, то $\forall \mathbf{v} \perp \mathbf{v_1}, ..., \mathbf{v_r} : \mathbf{a_j} \cdot \mathbf{v} = 0$. Тогда для каждого $\mathbf{a_j}$ выполняется равенство $\sum_{i=1}^{r} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = ||\mathbf{a_j}||^2$. Суммируя по всем строкам получаем:

$$\sum_{j=1}^{n} ||\mathbf{a_j}||^2 = \sum_{j=1}^{n} \sum_{i=1}^{r} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = \sum_{i=1}^{r} \sum_{j=1}^{n} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = \sum_{i=1}^{r} ||A\mathbf{v_i}||^2 = \sum_{i=1}^{r} \sigma_i^2(A).$$

При этом следует учесть, что $\sum_{j=1}^{n} ||\mathbf{a_j}||^2 = \sum_{j=1}^{n} \sum_{k=1}^{d} a_{jk}^2$ – сумма квадратов всех элементов матрицы A. Таким образом, сумма квадратов сингулярных значений матрицы A на самом деле эквивалентна квадрату всего "содержимого" матрицы A. Существует специальная норма, связанная с этой величиной – норма Фробениуса, обозначаемая $||A||_F$ и определяющаяся

$$||A||_F = \sqrt{\sum_{j,k} a_{jk}^2}$$

Следствием вышеизложенных выкладок служит утверждение о том, что сумма квадратов сингулярных значений матрицы A равна квадрату нормы Фробениуса.

$$\sum_i \sigma_i^2(A) = ||A||_F^2$$

Матрица A может быть полностью описана тем, как она трансформирует векторы $\mathbf{v_i}$. Любой вектор $\mathbf{v}$ может быть представлен в виде линейной комбинации векторов $\mathbf{v_1}, ..., \mathbf{v_r}$ и вектора, перпендикулярного всем $\mathbf{v_i}$. $A\mathbf{v}$ – такая же линейная комбинация $A\mathbf{v_1}, ..., A\mathbf{v_r}$, как и $\mathbf{v}$ – линейная комбинация векторов $\mathbf{v_1}, ..., \mathbf{v_r}$. Таким образом, $A\mathbf{v_1}, ..., A\mathbf{v_r}$ образуют фундаментальное множество векторов, ассоциированных с матрицей A. После нормализации получаем:

$$\mathbf{u_i} = \frac{1}{\sigma_i(A)} A\mathbf{v_i}.$$

Векторы $\mathbf{u_1}, ..., \mathbf{u_r}$ называются левыми сингулярными векторами матрицы A, векторы $\mathbf{v_1}, ..., \mathbf{v_r}$ – правыми сингулярными векторами.

Покажем, что левые сингулярные векторы также являются ортогональными.
Для доказательства данной теоремы забежим несколько вперед и воспользуемся теоремой из (2.5), которая утверждает, что

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

Составим набор матриц $\{B_k\}$ таких, что

$$B_k = A - \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i^T} = \sum_{i=k+1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}. \tag{2.4.1}$$

Из вида уравнения (2.4.1) видно, что правыми сингулярными векторами матрицы B_k являются векторы $\{\mathbf{v_{k+1}}, ..., \mathbf{v_r}\}$, которые в свою очередь являются также подмножеством множества правых сингулярных векторов матрицы A. Таким образом,

$$B_{r-1} = \sigma_r \mathbf{u_r} \mathbf{v_r^T}$$

имеет единственный левый сингулярный вектор $\mathbf{u_r}$. В данном случае система левых сингулярных векторов $\{\mathbf{u_r}\}$ является ортонормированной системой. По индукции предположим, что матрица B_i имеет ортонормированную систему левых сингулярных векторов $\{\mathbf{u_{i+1}}, ..., \mathbf{u_r}\}$. Необходимо показать, что матрица B_{i-1} также имеет ортонормированную систему левых сингулярных векторов. Для этого необходимо показать, что вектор $\mathbf{u_i}$ ортогонален множеству векторов $\{\mathbf{u_{i+1}}, ..., \mathbf{u_r}\}$.
Предположим, что это не так, что $\exists\ j > i$ такие, для которых $\mathbf{u_i^T u_j} \neq 0$. Без ограничения общности предположим, что $\mathbf{u_i^T u_j} > 0$. В таком случае для бесконечно малого $\varepsilon > 0$

$$A\left(\frac{\mathbf{v_i} + \varepsilon \mathbf{v_j}}{||\mathbf{v_i} + \varepsilon \mathbf{v_j}||_2}\right) = \frac{\sigma_i \mathbf{u_i} + \varepsilon \sigma_j \mathbf{u_j}}{||\mathbf{v_i} + \varepsilon \mathbf{v_j}||_2} = \frac{\sigma_i \mathbf{u_i} + \varepsilon \sigma_j \mathbf{u_j}}{\sqrt{||\mathbf{v_i}||_2^2 + ||\varepsilon \mathbf{v_j}||_2^2}} = \frac{\sigma_i \mathbf{u_i} + \varepsilon \sigma_j \mathbf{u_j}}{\sqrt{1 + \varepsilon^2}}.$$

Из построения правых сингулярных векторов известно, что

$$\mathbf{v_i} = \underset{\mathbf{v} \perp \mathbf{v_1}, ..., \mathbf{v_{i-1}}, ||\mathbf{v}|| = 1}{\operatorname{argmax}} ||A\mathbf{v}||_2 \ \Rightarrow\ \forall \mathbf{x} \in \operatorname{span}\{\mathbf{v_i}, ..., \mathbf{v_r}\},\ \mathbf{x} \neq \mathbf{v_i},\ ||A\mathbf{x}||_2 < ||A\mathbf{v_i}||_2 = \sigma_i.$$

Вспомним, что $\sigma_i \mathbf{u_i} = A\mathbf{v_i}$, тогда из вышеприведенных рассуждений получим неравенство

$$\forall \mathbf{x} \in \operatorname{span}\{\mathbf{v_i}, ..., \mathbf{v_r}\},\ \mathbf{x} \neq \mathbf{v_i}:\ ||A\mathbf{x}||_2 < ||\sigma_i \mathbf{u_i}||_2 \Leftrightarrow \mathbf{u_i^T} A\mathbf{x} < \sigma_i \mathbf{u_i^T u_i} \Leftrightarrow \mathbf{u_i^T} A\mathbf{x} < \sigma_i$$

$$\mathbf{u_i^T} A\left(\frac{\mathbf{v_i} + \varepsilon \mathbf{v_j}}{||\mathbf{v_i} + \varepsilon \mathbf{v_j}||_2}\right) = \mathbf{u_i^T} \frac{\sigma_i \mathbf{u_i} + \varepsilon \sigma_j \mathbf{u_j}}{\sqrt{1 + \varepsilon^2}} = (\sigma_i\ + \varepsilon \sigma_j \mathbf{u_i^T u_j})\big(1 - \frac{\varepsilon^2}{2} + O(\varepsilon^4)\big)$$

$$= \sigma_i\ + \varepsilon \sigma_j \mathbf{u_i^T u_j} + O(\varepsilon^2) > \sigma_i$$

Полученное противоречие доказывает, что система левых сингулярных векторов $\{\mathbf{u_i}, ..., \mathbf{u_r}\}$ ортонормирована. Продолжая по индукции, получим, что все левые сингулярные векторы матрицы A образуют ортонормированную систему векторов.

2.5 Сингулярное разложение

Пусть A – матрица размерности $n \times d$, $\mathbf{v_1}, ..., \mathbf{v_r}$ – ее правые сингулярные векторы, которым соответствуют собственные значения $\sigma_1, ..., \sigma_r$. Тогда $\mathbf{u_i} = \frac{1}{\sigma_i} A\mathbf{v_i}, i = 1, ..., r$ – левые сингулярные веторы. Покажем, что

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

Для доказательства этой теоремы, докажем сперва следующую лемму: матрицы A и B идентичны тогда и только тогда, когда $\forall \mathbf{v} : A\mathbf{v} = B\mathbf{v}$.

Необходимость: ясно, что если $A = B$, то $\forall \mathbf{v} : A\mathbf{v} = B\mathbf{v}$.

Достаточность: предположим, что $\forall \mathbf{v} : A\mathbf{v} = B\mathbf{v}$. Пусть $\mathbf{e_i}$ – вектор, все элементы которого нули, за исключением i-ой компоненты, которая равна 1. В таком случае $A\mathbf{e_i}$ – i-ый столбец матрицы A и, следовательно, $A = B$ если $\forall i : A\mathbf{e_i} = B\mathbf{e_i}$.

Воспользуемся только что доказанной леммой для того, чтобы доказать равенство

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

Для любого правого сингулярного вектора $\mathbf{v_j}$ выполняется равенство:

$$A\mathbf{v_j} = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \mathbf{v_j},$$

т.к. $\forall i \neq j : \mathbf{v_i^T} \mathbf{v_j} = 0$, т.к. сингулярные векторы ортогональны. Учитывая тот факт, что любой вектор $\mathbf{v}$ может быть представлен в виде линейной комбинации сингулярных векторов и вектора, перпендикулярного им, то $A\mathbf{v} = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \mathbf{v}$. Используя только что доказанную лемму получаем, что

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

2.6 Геометрическая интерпретация

Геометрическая интерпретация сингулярного разложения заключается в следующем факте из геометрии: *Образом любого линейного преобразования, заданного с помощью матрицы $m \times n$, примененного к единичной сфере является гиперэллипсоид.*

Гиперэллипсоид в $\mathbb{R}^m$ можно определить как поверхность, полученную в результате растяжения единичной сферы в $\mathbb{R}^m$ в $\sigma_1, ..., \sigma_m$, возможно, в ноль раз вдоль каких-то ортогональных направлений $u_1, ..., u_m \in \mathbb{R}^m$. Для определенности предположим, что u_i – единичные векторы. Тогда векторы $\{\sigma_i u_i\}$ – главные полуоси гиперэллипсоида, длины которых $\sigma_1, ..., \sigma_m$. Если $\text{rank}\,(A) = r$, то ровно r длин σ_i ненулевые, и, в частности, если $m \geq n$, то не более, чем n из них ненулевые.

Под единичной сферой подразумевается обычная евклидова сфера в n-мерном пространстве, т.е. единичная сфера для 2-нормы; обозначим ее как S. Тогда AS, образ S под действием преобразования A – гиперэллипсоид.

Первым делом определим n сингулярных значений преобразования A. Ими являются длины n главных полуосей эллипса AS, обозначенных σ_1, σ_2. Обычно принято нумеровать сингулярные значения в убывающем порядке, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n > 0$.

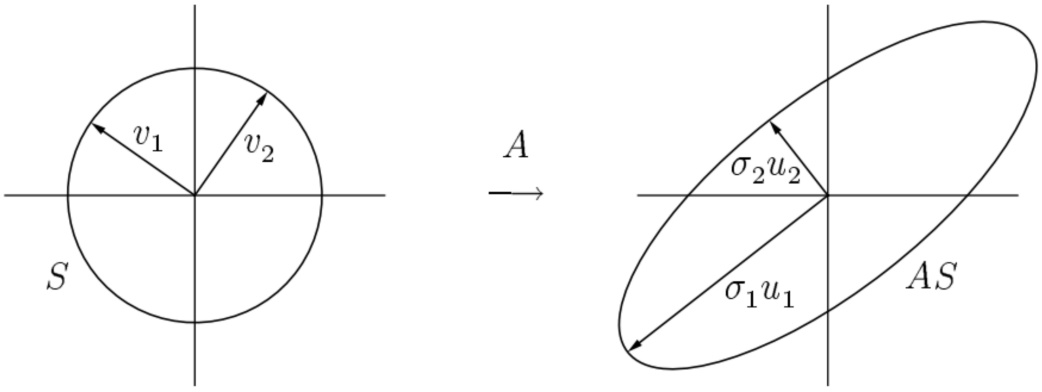


Рис. 2: Преобразование из единичной окружности в эллипс

Далее определим n левых сингулярных векторов преобразования A. Ими являются единичные векторы $\{u_1, u_2\} \in AS$, направленные вдоль главных полуосей эллипса AS, пронумерованные таким образом, чтобы соответствовать номерам сингулярных значений. Поэтому $\sigma_i u_i$ – i-ая по величине полуось.

Последними определим n правых сингулярных векторов преобразования A. Ими являются единичные векторы $\{v_1, v_2\} \in S$, которые являются прообразами главных осей эллипса AS, пронумерованные таким образом, чтобы $Av_j = \sigma_j u_j$.

2.7 Существование и единственность

В (2.1), (2.2), (2.3) и (2.5) было показано, как можно найти сингулярное разложение. Следующим необходимо доказать, что любая матрица может быть разложена таким образом.

Доказательство будем производить по индукции по размерности матрицы $A_{m\times n}$, где A – комплексная матрица размерности $m \times n$. Все нижеизложенные выкладки справедливы для $m \geq n$ (если $m < n$, то необходимо сперва транспонировать матрицу).

Для $n = 1$ (и любой m) матрица A является ветором-столбцом. Возьмем $V = 1,\ \hat{\Sigma} = ||A||_2,\ \hat{U} = \dfrac{A}{||A||_2}$. Тогда очевидно, что мы нашли "сжатое" сингулярное разложение, т.е.

$$A = \hat{U}\hat{\Sigma}V^H.$$

"Полное" сингулярное разложение можно получить, расширив матрицу $\hat{U}$ до U с помощью ортогонализации Грамма-Шмидта и добавив несколько нулей к $\hat{\Sigma}$.

Предположим теперь, что SVD-разложение существует для матрицы размерности $(m-1)\times(n-1)$ и покажем, что оно также существует для матрицы размерности $m\times n$. Возьмем $\mathbf{v}_1 \in \mathbb{C}^n$ таким образом, чтобы

$$\begin{cases} ||\mathbf{v}_1||_2 = 1, \\ ||A||_2 = \sup\limits_{\mathbf{v}_1 \in \mathbb{C}^n} ||A\mathbf{v}_1||_2 > 0. \end{cases}$$

Теперь возьмем

$$\mathbf{u_1} = \frac{A\mathbf{v}_1}{||A\mathbf{v}_1||_2}. \tag{2.7.1}$$

Далее воспользуемся ортогонализацией Грамма-Шмидта для того, чтобы расширить $\mathbf{u}_1$ и $\mathbf{v}_1$ до унитарных матриц, добавляя столбцы $\widetilde{U}_1$ и $\widetilde{V}_1$ соответственно. Т.о. матрицы $U_1,\ V_1$

выглядят следующим образом:

$$U_1 = \begin{bmatrix} \mathbf{u}_1 & \widetilde{U}_1 \end{bmatrix}, \quad V_1 = \begin{bmatrix} \mathbf{v}_1 & \widetilde{V}_1 \end{bmatrix}.$$

В таком случае получаем следующее выражение:

$$U_1^H A V_1 = \begin{bmatrix} \mathbf{u}_1^H \\ \widetilde{U}_1 \end{bmatrix} A \begin{bmatrix} \mathbf{v}_1 & \widetilde{V}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^H A \mathbf{v}_1 & \mathbf{u}_1^H A \widetilde{V}_1 \\ \widetilde{U}_1^H A \mathbf{v}_1 & \widetilde{U}_1^H A \widetilde{V}_1 \end{bmatrix}.$$

Используя (2.7.1) и "специальный" выбор $\mathbf{v}_1$ имеем:

$$\mathbf{u}_1^H A \mathbf{v}_1 = \frac{(A\mathbf{v}_1)^H}{||A\mathbf{v}_1||_2} A\mathbf{v}_1 = \frac{||A\mathbf{v}_1||_2^2}{||A\mathbf{v}_1||_2} = ||A\mathbf{v}_1||_2 = ||A||_2.$$

Для этой величины вводится обозначение $\sigma_1 := ||A||_2$.

Опять, используя (2.7.1), получаем

$$\widetilde{U}_1^H A \mathbf{v}_1 = \widetilde{U}_1^H \mathbf{u}_1 ||A\mathbf{v}_1||_2 = \mathbf{0},$$

т.к. $\widetilde{U}_1^H$ построено с помощью ортогонализации Грамма-Шмидта таким образом, чтобы $\widetilde{U}_1^H \mathbf{u}_1 = \mathbf{0}$.

Теперь покажем, что $\omega^H := \mathbf{u}_1^H A \widetilde{V}_1 = \mathbf{0}^T$. Воспользуемся тем фактом, что $||A||_2 = \sigma_1$. Используя только что доказанные факты, запишем матрицу Σ следующим образом:

$$\begin{cases} \Sigma = U_1^H A V_1 = \begin{bmatrix} \sigma_1 & \omega^H \\ \mathbf{0} & \widetilde{A} \end{bmatrix} \\ A = U_1 \Sigma V_1^T, \ ||A||_2 = \sigma_1 \ \Rightarrow \ \Sigma = U_1^H A V_1, \ ||\Sigma||_2 = \sigma_1. \end{cases}$$

Воспользуемся определением нормы матрицы $||A||_2 = \sup_x ||Ax||_2$. Возьмем специальный вектор $x = \begin{bmatrix} \sigma_1 \\ \omega \end{bmatrix}$, тогда

$$\left\| \begin{bmatrix} \sigma_1 & \omega^H \\ \mathbf{0} & \widetilde{A} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \omega \end{bmatrix} \right\|_2 \geq \sigma_1^2 + \omega^H \omega = \sqrt{\sigma_1^2 + \omega^H \omega} \left\| \begin{bmatrix} \sigma_1 \\ \omega \end{bmatrix} \right\|_2,$$

из чего следует, что $\sigma_1 \geq \sqrt{\sigma_1^2 + \omega^H \omega}$, откуда получается, что $\omega^H = \mathbf{0}^T$.

Т.о. мы показали, что

$$U_1^H A V_1 = \begin{bmatrix} \sigma_1 & \mathbf{0}^T \\ \mathbf{0} & \widetilde{A} \end{bmatrix},$$

т.е. свели задачу с размерности $m \times n$ к $(m-1) \times (n-1)$, которая по условию индукции уже решена: $\widetilde{A} = U_2 \Sigma_2 V_2^H$. Тогда решение исходной задачи можно записать следующим образом:

$$U_1^H A V_1 = \begin{bmatrix} \sigma_1 & \mathbf{0}^T \\ \mathbf{0} & U_2 \Sigma_2 V_2^H \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & U_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & \mathbf{0}^T \\ \mathbf{0} & \Sigma_2 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & V_2 \end{bmatrix}^H,$$

или, если выразить матрицу A:

$$A = U_1 \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & U_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & \mathbf{0}^T \\ \mathbf{0} & \Sigma_2 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & V_2 \end{bmatrix}^H V_1^H,$$

что также является SVD-разложением, т.к. произведение унитраных матриц – унитарная матрица.

Сингулярное разложение в общем случае не является единственным. Однако сингулярные значения $\{\sigma_j\}$ уникальны для каждой матрицы A. Если все сингулярные значения различные, то левые и правые сингулярные векторы определены однозначно с точностью до комплексных знаков. Если все длины главных полуосей гиперэллипса различны, то эти оси однозначно определяют сингулярные векторы и значения с геометрической точки зрения.

3 Математический метод

3.1 Описание алгоритма

3.1.1 Для матриц $n \times n$, $r = n$

Рассмотрим невырожденную квадратную матрицу $A_{n\times n}$, $r = \text{rank}(A) = n$. Самым простым методом нахождения сингулярного разложения является сведение задачи нахождения сингулярного разложения к задаче нахождения спектрального разложения.

В разделах (2.1), (2.2), (2.3) было доказано, что любая квадратная невырожденная матрица A может быть представлена в виде

$$A = U\Sigma V^T, \tag{3.1.1}$$

где U, V – ортогональные матрицы, а Σ – диагональная матрица.

Составим матрицу $B = A^T A$. Она симметричная и положительно определенная, следовательно имеет полный набор действительных положительных собственных значений. Учитывая, что $A = U\Sigma V^T$, перепишем B в виде

$$B = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T \underbrace{U^T U}_{I} \Sigma V^T = V\Sigma^2 V^T.$$

Без особого труда можно заметить, что $B = V\Sigma^2 V^T$ – спектральное разложение матрицы B, где V – матрица, состоящая из столбцов собственных векторов, а $\Lambda := \Sigma^2$ – диагональная матрица, состоящая из собственных значений.

Таким образом, найдя спектральное разложение матрицы B (например, двусторонним методом вращений Якоби, рассмотренном на лекции), мы получим матрицы V, Σ^2. Матрица $\Sigma = \sqrt{\Sigma^2}$ является диагональной матрицей с элементами равными корню квадратному из элементов матрицы Σ^2.

Оставшаяся матрица U находится явным способом из (3.1.1):

$$U = AV\Sigma^{-1},$$

где Σ^{-1} – диагональная матрица с элементами равными обратным значениям элементов матрицы Σ.

3.1.2 Исходный код

```
% Mathematical method for computing SVD for squared nonsingular matrices
function [U,S,V] = svd_math_1(A)
% Input: A is an original squared nonsingular matrix
% Ouput: U,V are orthonormal matrices,
%        S is a diagonal matrix
% U,S,V are constructed in a such way that A = U * S * V'
   [m,n] = size(A); % get matrix dimensions
   r = rank(A); % calculate rank of the matrix
   if m ~= n || r ~= n % if matrix is not squared or nonsingular then this ...
       algorithm would not work
       error('Matrix should be square and nonsingular!');
   end
   B = A'*A; % calculate symmetric positive definite matrix B
   [V,L] = eig(B); % calculate V and L = S^2 using two-sided jacobi's ...
       rotation algorithm
   S = sqrt(L); % square root of S^2
   U = A * V / S; % calculate U
end
```

3.1.3 Для матриц $m \times n$, $r = \min(m, n)$

В разделах (2.4), (2.5) было неявно доказано, что любая матрица $A_{m\times n}$ такая, что $r = \text{rank}(A) = \min(m, n)$ может быть разложена в произведение $A = U\Sigma V^T$.

Основное отличие этого подхода от предыдущего заключается в том, что используемый нами двусторонний алгоритм вращений Якоби для нахождения спектрального разложения не работает для вырожденных матриц.

Предположим, что матрица A имеет размерность $m \times n$ и $\text{rank}(A) = m$. Если мы воспользуемся вышеизложенным алгоритмом, то получим следующий результат:

$$(A^T)_{n\times m}A_{m\times n} = B_{n\times n},$$

где $B_{n\times n}$ – симметрическая положительная полуопределенная матрица, причем $\text{rank}(B_{n\times n}) = m,\ m < n$, т.е. матрица B является вырожденной.

Чтобы обойти эту проблему достаточно рассмотреть два случая:

- Если $\text{rank}(A_{m\times n}) = n$, то $B = A^TA$ и весь алгоритм аналогичен вышеизложенному.
- Если $\text{rank}(A_{m\times n}) = m$, то $B = AA^T$ и алгоритм слегка изменяется. Учитывая (3.1.1), найдем представление B:

$$B = U\Sigma V^T(U\Sigma V^T)^T = U\Sigma\underbrace{VV^T}_{I}\Sigma^TU^T = U\Sigma^2U^T.$$

  Дальнейшие вычисления аналогичны предыдущему алгоритму, за исключением того, что, разложив B с помощью спектрального разложения, мы получим U, Σ и после этого из (3.1.1) найдем V с помощью явной подстановки $V = (\Sigma^{-1}U^TA)^T$.

Рассмотрим правило немного подробнее.
Если $\text{rank}(A_{m\times n}) = n$, то

$$B = (A^T)_{n\times m}A_{m\times n} = B_{n\times n},$$

где B – невырожденная симметричная матрица. Сингулярное разложение в таком случае будет выглядеть так:

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}(V^T)_{n\times n}.$$

Если $\text{rank}(A_{m\times n}) = m$, то

$$B = A_{m\times n}(A^T)_{n\times m} = B_{m\times m},$$

где B – невырожденная симметричная матрица. Сингулярное разложение в таком случае будет выглядеть так:

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times m}(V^T)_{m\times n}.$$

3.1.4 Исходный код

```
% Mathematical method for computing SVD for rectangular m*n matrices such ...
    that rank = min(m,n)
function [U,S,V] = svd_math_2(A)
% Input: A is an original rectangular matrix, rank(A) = min(m,n)
% Ouput: U,V are orthonormal matrices,
%        S is a diagonal matrix
% U,S,V are constructed in a such way that A = U * S * V'
   [m,n] = size(A); % get matrix dimensions
   r = rank(A); % calculate rank of the matrix
   if r ~= min(m,n) % if necessary condition is not satisfied then this ...
       algorithm would not work
```

```
10          error('rank(A) should be equal to min(m,n)!');
11      end
12      % calculate symmetric positive definite matrix B
13      if r == n
14          B = A'*A;
15          [V,L] = eig(B); % calculate V and L = S^2 using two-sided jacobi's ...
                rotation algorithm
16          S = sqrt(L); % square root of S^2
17          U = A * V / S; % calculate U
18      else % r == m
19          B = A*A';
20          [U,L] = eig(B); % calculate U and L = S^2 using two-sided jacobi's ...
                rotation algorithm
21          S = sqrt(L); % square root of S^2
22          V = (S \ U' * A)'; % calculate V
23      end
24  end
```

3.1.5 Для матриц $m \times n$, $r < \min(m, n)$

В разделах (2.4), (2.5) было неявно доказано, что любая матрица $A_{m \times n}$ такая, что $r = \text{rank}(A) < \min(m, n)$, может быть разложена в произведение $A = U \Sigma V^T$.

Используемый нами двусторонний алгоритм вращений Якоби для нахождения спектрального разложения не работает для вырожденных матриц. И в данном случае никак не получится создать невырожденную симметричную матрицу B. Поэтому единственным решением возникшей проблемы является использование другого алгоритма вычисления спектрального разложения, который будет правильно обрабатывать вырожденность матрицы.

Из вышесказанного заключаем, что алгоритм вычисления SVD разложения для всех возможных действительных матриц будет аналогичен (3.1.2), за тем исключением, что функция eig(B) вычисляет спектральное разложение симметричной положительной полуопределенной матрицы B с учетом вырожденности.

После изучения документации языка Matlab становится ясно, какие алгоритмы спектрального разложения используются на практике. По умолчанию, если матрица B симметричная, то используется разложение Холецкого для решения задачи собственных значений и векторов, в противном случае используется обобщенное разложение Шура.

В конкретной задаче известно, что матрица B – симметричная и положительно полуопределенная, следовательно для нахождения собственных векторов и значений можно использовать разложение Холецкого. Более подробно об этом можно почитать в [4], [5].

3.1.6 Исходный код

```
1  % Mathematical method for computing full SVD for rectangular m*n real ...
       matrices
2  function [U,S,V] = svd_math(A)
3  % Input: A is an original nonzero matrix with no additional properties
4  % Ouput: U,V are orthonormal matrices,
5  %        S is a diagonal matrix
6  % U,S,V are constructed in a such way that A = U * S * V'
7      [m,n] = size(A); % get matrix dimensions
8      r = rank(A); % calculate rank of the matrix
9      [V,S2] = eig(A'*A); % calculate V and L = S^2 from symmetric positive ...
           semi-definite matrix A'*A using Cholesky factorization
```

```
10    sing = sqrt(S2(1:r,1:r)); % square root of non-singular part of S^2 -- ...
          calculating singular values
11    S = zeros(m,n); % create zero m-by-n matrix
12    S(1:r,1:r) = sing; % fill the non-singular part of matrix S
13    u = A * V(:,1:r) / sing; % get m-by-r u matrix
14    [U,~] = qr(u); % get orthogonal matrix from qr decomposition
15    U(:,1:r) = u; % create m-by-n matrix from u and U
16  end
```

3.2 Виды SVD разложений

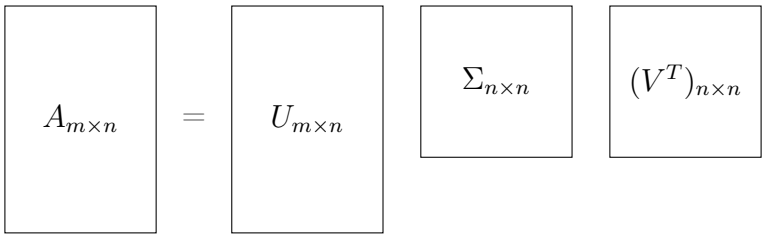
Разница SVD разложений заключается в размерностях получающихся матриц U, Σ, V.

3.2.1 "Тонкий" SVD

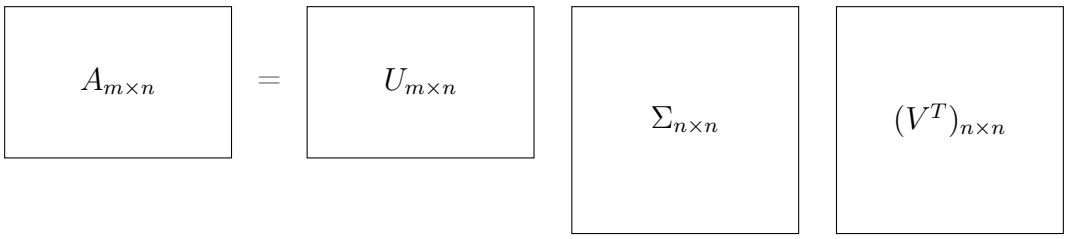
Данный тип разложения использовался в (3.1.2) и (3.1.4).
Предположим, что имеется матрица $A_{m\times n}$, причем $m \geq n$. Тогда сингулярное разложение можно представить в виде произведения матриц следующих размерностей:

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}(V^T)_{n\times n}.$$

Представим данное разложение графически:

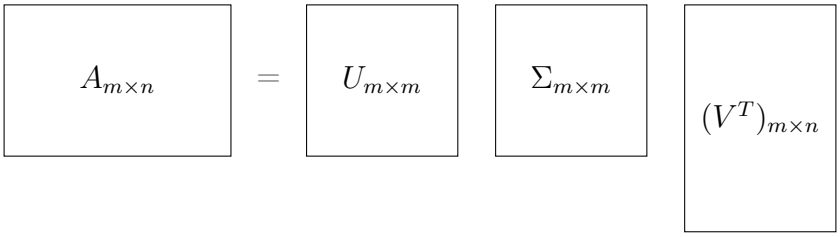


Теперь предположим, что имеется матрица $A_{m\times n}$, причем $m \leq n$. Тогда графическое представление будет выглядеть таким образом:



Однако в (4) при условии, что $m \leq n$ результирующие матрицы имеют вид:

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times m}(V^T)_{m\times n}.$$



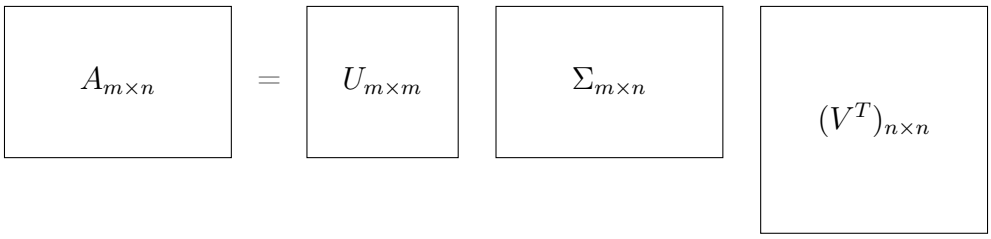
3.2.2 “Полный” SVD

Данный тип разложения использовался в (3.1.6).
Предположим, что имеется матрица $A_{m\times n}$. Тогда сингулярное разложение можно представить в виде произведения матриц следующих размерностей:

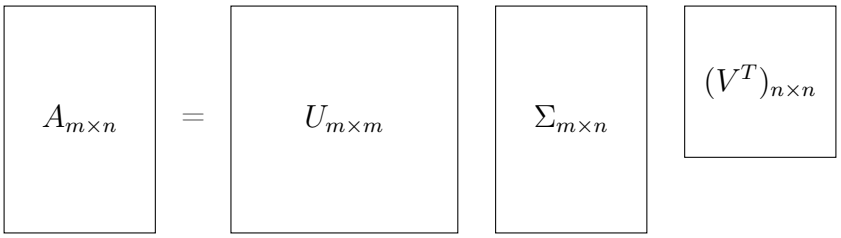
$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}(V^T)_{n\times n}.$$

Тогда графически это разложение можно представить двумя способами в зависимости от того, что больше: m или n.

Если $m < n$, то



Если $m > n$, то

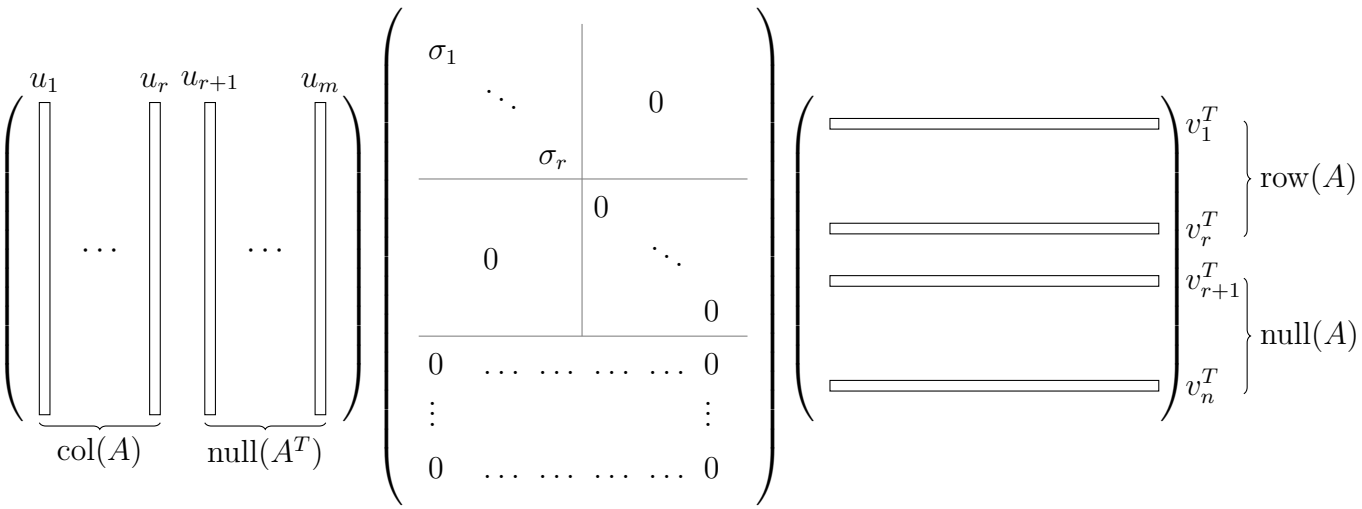


3.2.3 “Сжатый” SVD

Данный тип разложения является очень важным и часто используемым по двум причинам:

1. алгоритм имеет меньшую вычислительную сложность
2. алгоритм экономит память

Для упрощения понимания рассмотрим матрицу $A_{m\times n}$, причем $m > n$. Предположим, что $\operatorname{rank}(A) = r$. Тогда сингулярное разложение в “полном” виде будет выглядеть следующим образом:



Из приведенной иллюстрации легко можно заметить, что матрицу Σ можно представить в виде блочной матрицы

$$\Sigma_{m\times n} = \left(\begin{array}{c|c} \Sigma_{r\times r} & O_{r\times(n-r)} \\ \hline O_{(n-r)\times r} & O_{(n-r)\times(n-r)} \\ \hline O_{(m-n+r)\times r} & O_{(m-n+r)\times(n-r)} \end{array}\right), \tag{3.2.1}$$

поэтому матрицу $\Sigma_{m\times n}$ можно задать матрицей меньшей размерности $\Sigma_{r\times r}$. Векторы $u_{r+1}, ..., u_m$ задают ядро пространства строк, а векторы $v_{r+1}, ..., v_n$ задают ядро пространства столбцов, поэтому их также можно не считать. Т.о. матрицы U, Σ, V свелись к матрицам $\widetilde{U}, \widetilde{\Sigma}, \widetilde{V}$:

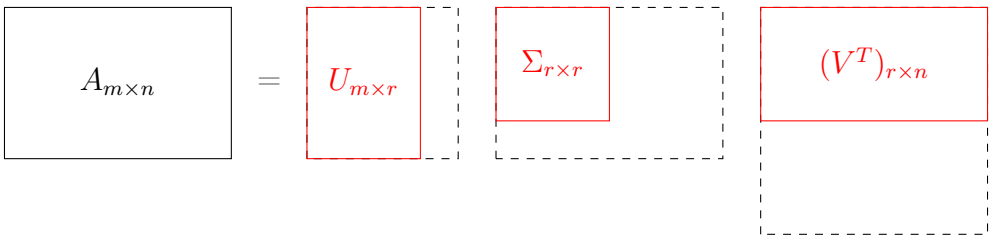
$$U_{m\times m} \to \widetilde{U}_{m\times r}$$
$$\Sigma_{m\times n} \to \widetilde{\Sigma}_{r\times r}$$
$$V_{n\times n} \to \widetilde{V}_{n\times r}$$

Сингулярное разложение можно представить в виде произведения матриц следующих размерностей:

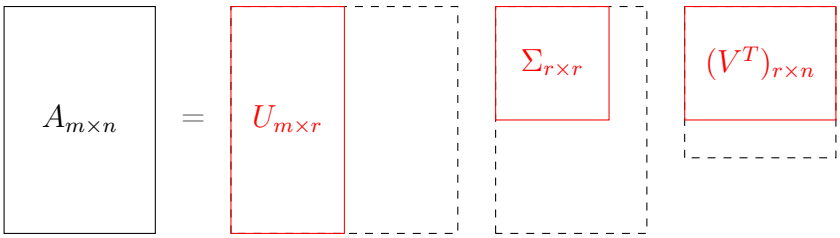
$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}(V^T)_{n\times n} = U_{m\times r}\Sigma_{r\times r}(V^T)_{r\times n}.$$

Графически SVD-разложение теперь можно представить следующими двумя изображениями:

Если $m < n$, то



Если $m > n$, то



4 Односторонний алгоритм вращений Якоби

4.1 Описание алгоритма

Пусть дана матрица A размерности $m \times n$, причем $m \geq n$ и $\operatorname{rank}(A) = n$. Если условие $m \geq n$ не выполняется, то находится сингулярное разложение матрицы $B = A^T$:

$$B = \bar{U}\bar{\Sigma}\bar{V}^T,$$

после чего оригинальное разложение матрицы A получается транспонированием:

$$A = B^T = (\bar{U}\bar{\Sigma}\bar{V}^T)^T = \underbrace{\bar{V}}_{U}\underbrace{\bar{\Sigma}}_{\Sigma}\underbrace{\bar{U}^T}_{V^T}.$$

Условие $\operatorname{rank}(A) = n$ является необходимым.

Предположим, что нам известна матрица M размерности 2×2 такая, что

$$M^T M = \begin{pmatrix} \alpha & \gamma \\ \gamma & \beta \end{pmatrix}.$$

Симметричность выполняется всегда: $(M^T M)^T = M^T (M^T)^T = M^T M$. В таком случае с помощью вращений Якоби $\Theta(M^T M)\Theta^T$,

$$\Theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} := \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

симметричную матрицу $M^T M$ можно привести к диагональному виду:

$$D = \Theta M^T M \Theta^T = (M\Theta^T)^T (M\Theta^T).$$

Перемножив матрицы $\Theta M^T M \Theta^T$, получим следующее выражение:

$$\begin{pmatrix} \alpha c^2 - 2\gamma cs + \beta s^2 & -\gamma s^2 + (\alpha - \beta)cs + \gamma c^2 \\ -\gamma s^2 + (\alpha - \beta)cs + \gamma c^2 & \beta c^2 + 2\gamma cs + \alpha s^2 \end{pmatrix} = \begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix}.$$

Оно имеет всего одну степень свободы θ, поэтому достаточно решить единственное уравнение:

$$-\gamma s^2 + (\alpha - \beta)cs + \gamma c^2 = 0,$$

чтобы задать матрицу вращений Гивенса Θ. Поделив обе части уравнения на $-\gamma c^2$, получим:

$$t^2 + \frac{(\beta - \alpha)}{\gamma} t - 1 = 0;\ t := \frac{s}{c} = \tan\theta.$$

$$t^2 + 2\zeta t - 1 = 0;\ \zeta = \frac{(\beta - \alpha)}{2\gamma}. \tag{4.1.1}$$

Решая уравнение, получаем корни

$$t = -\zeta \pm \sqrt{1 + \zeta^2}.$$

Заметим, что

$$(-\zeta \pm \sqrt{1 + \zeta^2})(\zeta \pm \sqrt{1 + \zeta^2}) = -\zeta^2 + 1 + \zeta^2 = 1,$$

откуда следует, что

$$t = -\zeta \pm \sqrt{1 + \zeta^2} = \frac{1}{\zeta \pm \sqrt{1 + \zeta^2}}.$$

Возьмем в качестве корня уравнения

$$t = \frac{\text{sign}\,(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}.$$

Из тангенса выразим необходимые нам синус и косинус:

$$\begin{cases} c = \dfrac{1}{\sqrt{1+t^2}} \\ s = ct \end{cases}$$

Таким образом мы полностью задали Θ.

Введем замену $U_0 := A; V_0 := I_{n\times n}$. Предположим, что R_0 – какая-то матрица вращений Гивенса. Тогда, вращая матрицу U_0 последовательно с помощью R_0 и R_0^{-1}, мы получим исходную матрицу U_0:

$$A \equiv U_0 = U_0 R_0^{-1} R_0.$$

Воспользовавшись свойством ортогональных матриц ($R^{-1} = R^T$), перепишем равенство в виде:

$$A = U_0 R_0^T R_0.$$

Сделаем замену

$$A = \underbrace{U_0 R_0^T}_{U_1}\underbrace{R_0}_{V_1} = U_1 V_1.$$

Продолжим рассматривать уже U_1. Проведем аналогичные рассуждения: предположим, что R_1 – какая-то матрица вращений Гивенса. Тогда, вращая матрицу U_1 последовательно с помощью R_1 и R_1^{-1}, мы получим исходную матрицу U_1:

$$A \equiv U_1 = U_1 R_1^{-1} R_1.$$

Опять сделаем замену

$$A = \underbrace{U_1 R_1^T}_{U_2}\underbrace{R_1 V_1}_{V_2} = \underbrace{U_0 R_0^T R_1^T}_{U_2}\underbrace{R_1 R_0}_{V_2} = U_2 V_2.$$

Продолжая аналогичные рассуждения, напишем формулу для n-ой итерации:

$$A = \underbrace{U_{n-1} R_{n-1}^T}_{U_n}\underbrace{R_{n-1} V_{n-1}}_{V_n} = \underbrace{U_0 R_0^T ... R_{n-1}^T}_{U_n}\underbrace{R_{n-1} ... R_0}_{V_n} = U_n V_n.$$

Идея алгоритма заключается в том, что в бесконечном цикле неявно конструируется матрица $A^T A$, выбирается пара индексов таких, что $i < j$, строится 2×2 матрица из элементов, стоящих на пересечении i, j-ых строк и столбцов матрицы $A^T A$, после чего она приводится к диагональному виду с помощью вращений Якоби. Итерации повторяются до тех пор, пока не достигается лимит итераций или достаточная точность.

С условием остановки возникает проблема: т.к. мы не конструируем матрицу $U^T U$ в явном виде, а лишь вычисляем 4 ее элемента, то нельзя ввести норму как сумму квадратов внедиагональных элементов. В [3] в качестве нормы предлагают брать

$$||U^T U|| = \frac{|c|}{\sqrt{\alpha\beta}},$$

поэтому критерием остановки является

$$\frac{|c|}{\sqrt{\alpha\beta}} \leq \varepsilon.$$

После окончания итерирования у нас имеется 2 матрицы:

$$U_n = U_0 R_0^T ... R_{n-1}^T$$
$$V_n = R_{n-1} ... R_0$$

Чтобы из матриц U_n, V_n получить сингулярное разложение, необходимо матрицу U_n представить в виде $U_n = \bar{U}_n \Sigma$, где Σ – диагональная матрица, элементы которой заданы следующим образом: $\sigma_i = ||U_n^{(i)}||$, где $U_n^{(i)}$ – i-ый столбец матрицы U_n. В таком случае $\bar{U}_n^{(i)} = \dfrac{U_n^{(i)}}{||U_n^{(i)}||}$.

Уточним, как вычисляются элементы матрицы $U^T U$:

$$B = U^T U = \begin{pmatrix} * & \cdots & * & \cdots & * & \cdots & * \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ * & \cdots & b_{ii} & \cdots & b_{ij} & \cdots & * \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ * & \cdots & b_{ji} & \cdots & b_{jj} & \cdots & * \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ * & \cdots & * & \cdots & * & \cdots & * \end{pmatrix}_{n \times n}$$

b_{ii} вычисляется как скалярное произведение i-ой строки матрицы U^T и i-ого столбца матрицы U, но т.к. i-ый столбец матрицы U после транспонирования становится i-ой строкой матрицы U^T, то вычисление элемента b_{ii} можно представить в виде:

$$\alpha := b_{ii} = \sum_{k=1}^{m} u_{k,i}^2.$$

Аналогично рассуждая, получаем формулу для вычисления b_{jj}:

$$\beta := b_{jj} = \sum_{k=1}^{m} u_{k,j}^2.$$

Т.к. $b_{ij} = b_{ji}$, то для определенности будем вычислять b_{ij}. b_{ij} вычисляется как скалярное произведение i-ой строки матрицы U^T и j-ого столбца матрицы U, а т.к. i-я строка матрицы U^T – i-ый столбец матрицы U, то вычисление b_{ij} можно представить в следующем виде:

$$\gamma := b_{ij} = b_{ji} = \sum_{k=1}^{m} u_{ki} u_{kj}.$$

4.2 Исходный код

В следующем листинге приведен код одностороннего алгоритма вращений Якоби для вычисления сингулярного разложения, написанный на языке Matlab.

```
1 % Jacobi's one-sided rotation algorithm for SVD
2 function [U,S,V] = svd_jacobi(A)
3 % Input: A is an original matrix
4 % Ouput: U,V are orthonormal matrices,
5 %         S is a diagonal matrix
6 % U,S,V are constructed in a such way that A = U * S * V'
```

```
7   [m,n] = size(A); % get matrix dimensions to check necessary condition ...
        and transpose if necessary
8   transposed = 0; % boolean ('true' if matrix has been transposed because ...
        n > m)
9   if n > m
10      A = A';
11      transposed = 1;
12  end
13  % set up necessary constants
14  EPS = 1e-8; % tolerance (accuracy)
15  MAX_STEPS = 40; % max number of iterations
16  % set up initial values
17  U = A;
18  V = eye(n); % identity matrix n*n
19  % start iteration process
20  for steps = 1:MAX_STEPS
21    eps = 0; % norm of U'*U on the current step
22    for j = 2:n % iterate through all rows
23      for k = 1:j-1 % iterate through all columns until diagonal
24        % implicitly calculate 4 elements of U'*U
25        alpha = sum(U(:,k) .^ 2);
26        beta  = sum(U(:,j) .^ 2);
27        gamma = dot(U(:,k), U(:,j));
28        % estimate the norm of U'*U on the current step
29        eps = max(eps, abs(gamma) / sqrt(alpha * beta));
30        if gamma ~= 0 % separately with gamma = 0 in order to get rid of ...
              division by 0
31          zeta = (beta - alpha) / (2 * gamma); % construct zeta
32          t = sign(zeta)/(abs(zeta) + sqrt(1 + zeta ^ 2)); % calculate ...
                root of a quadratic equation
33        else
34          t = 0; % 1 / Inf = 0
35        end
36        c = 1 / sqrt(1 + t^2); % express the cos(theta) using tan(theta)
37        s = t * c; % express the sin(theta) using tan(theta) and cos(theta)
38        % implicitly calculate U = U * R'
39        t = U(:,k);
40        U(:,k) = c*t - s*U(:,j);
41        U(:,j) = s*t + c*U(:,j);
42        % implicitly calculate V = V * R'
43        t = V(:,k);
44        V(:,k) = c*t - s*V(:,j);
45        V(:,j) = s*t + c*V(:,j);
46      end
47    end
48    % escape condition
49    if eps < EPS % if reached enough tolerance
50      break; % stop iteration process
51    end
52  end
53  % check convergence
54  if steps >= MAX_STEPS
55    error('svd_jacobi faild to converge!');
56  end
57  singvals = zeros(1,n); % create array of singular values
58  % separate U,S from U
59  for j = 1:n % iterate through each column
60    singvals(j) = norm(U(:,j)); % set columns' norms as singular values
61    U(:,j) = U(:,j) / singvals(j); % normalize each column
62  end
```

```
63    % set singular values as diagonal elements of S
64    S = diag(singvals);
65    if transposed == 1 % if matrix has been transposed
66        [V,U] = deal(U,V); % swap U,V matrices
67    end
68  end
```

4.3 Анализ работы алгоритма

4.3.1 Анализ решения уравнения 4.1.1

Во время вывода алгоритма первая произвольность в выборе какого-либо параметра возникает во время решения квадратного уравнения

$$t^2 + 2\zeta t - 1 = 0.$$

Как было показано, корней у уравнения 2, и каждое можно представить двумя способами:

$$\left[\begin{array}{l} t_{1,2} = -\zeta \pm \sqrt{1+\zeta^2}, \\ t_{1,2} = \dfrac{1}{\zeta \pm \sqrt{1+\zeta^2}}. \end{array}\right.$$

Анализируя необходимое количество итераций для сходимости алгоритма и точность полученных вычислений, выберем наилучшее решение и его представление.

На Рис. 3 представлен bar chart, с помощью которого можно проанализировать поведение алгоритма в зависимости от того, какой вариант корня был выбран. Для построения данной диаграммы создавались случайные квадратные гауссовы матрицы A нужной размерности, после чего производилось сингулярное разложение с каждым из корней в обоих представлениях и составлялась статистика работы.

Вдоль координаты x заданы размерности матрицы A, вдоль координаты y – среднее количество итераций, которое понадобилось для сходимости алгоритма. Для того, чтобы получить среднее количество итераций, которое требуется для сходимости алгоритма, необходимо произвести вычисления несколько раз. Число, стоящее над каждым набором столбцов, показывает, сколько вычислений было произведено перед тем, как составить результат.

Столбец, повернутый в отрицательную область показывает, что ни одно вычисление не сошлось. Если хотя бы одно вычисление сошлось, то столбец будет в положительной зоне.

Проанализировав данный график, приходим к выводу, что упорядоченный список (лучший → худший) выглядит следующим образом:

1. $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| - \sqrt{1+\zeta^2}}$. При таком выборе алгоритм сходится стандартно за 2-3 итерации.
2. $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$. В данном случае график сходимости больше напоминает логарифмическую кривую.
3. $t = -\operatorname{sign}(\zeta)(|\zeta| + \sqrt{1+\zeta^2})$. Ярко выраженного шаблона при использовании такой формулы график сходимости не имеет. Однако график отдаленно напоминает зашумленную линейную функцию.
4. $t = -\operatorname{sign}(\zeta)(|\zeta| - \sqrt{1+\zeta^2})$. Как видно из диаграммы, при использовании такой формулы алгоритм перестает сходиться, начиная с 7 размерности.

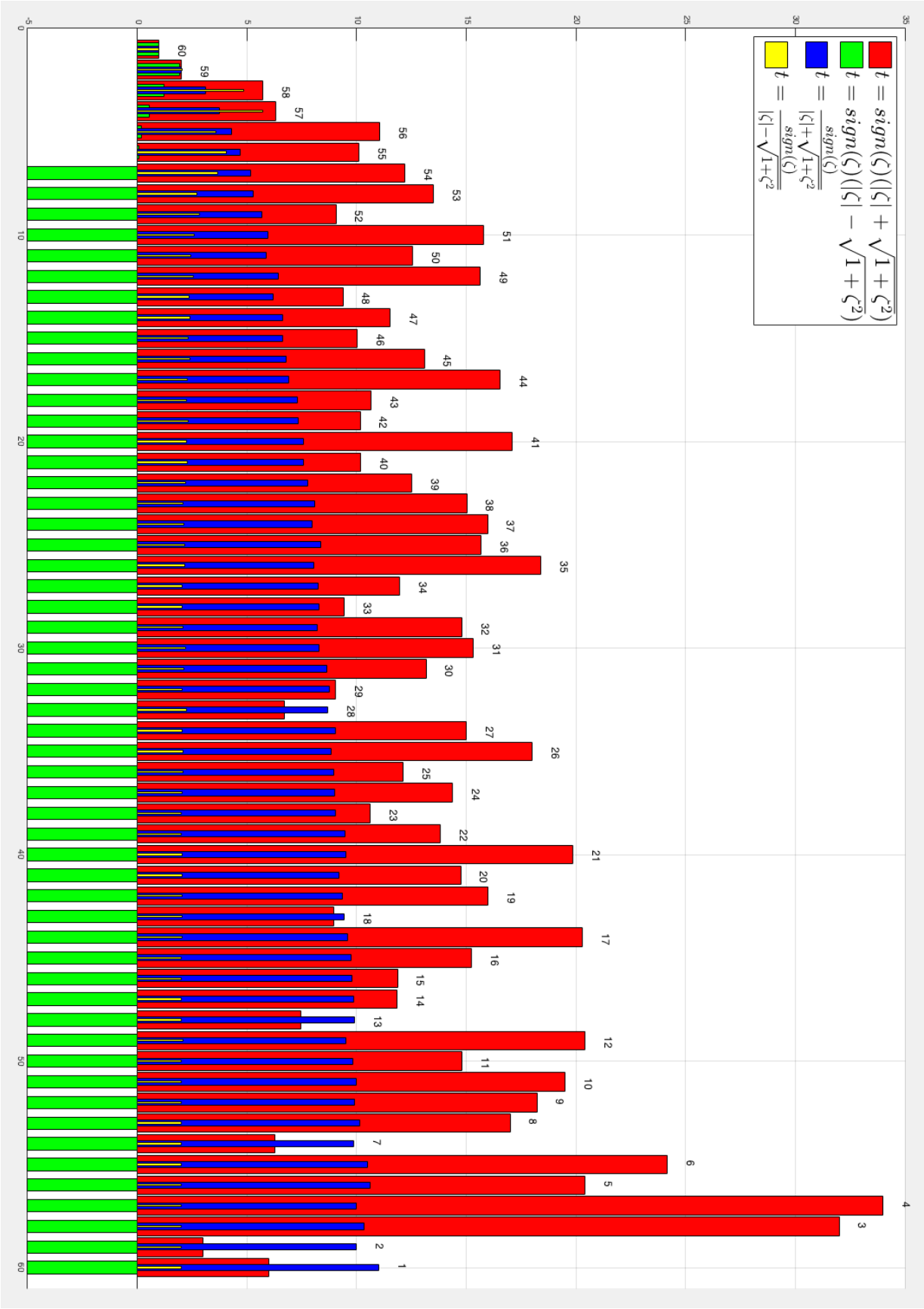
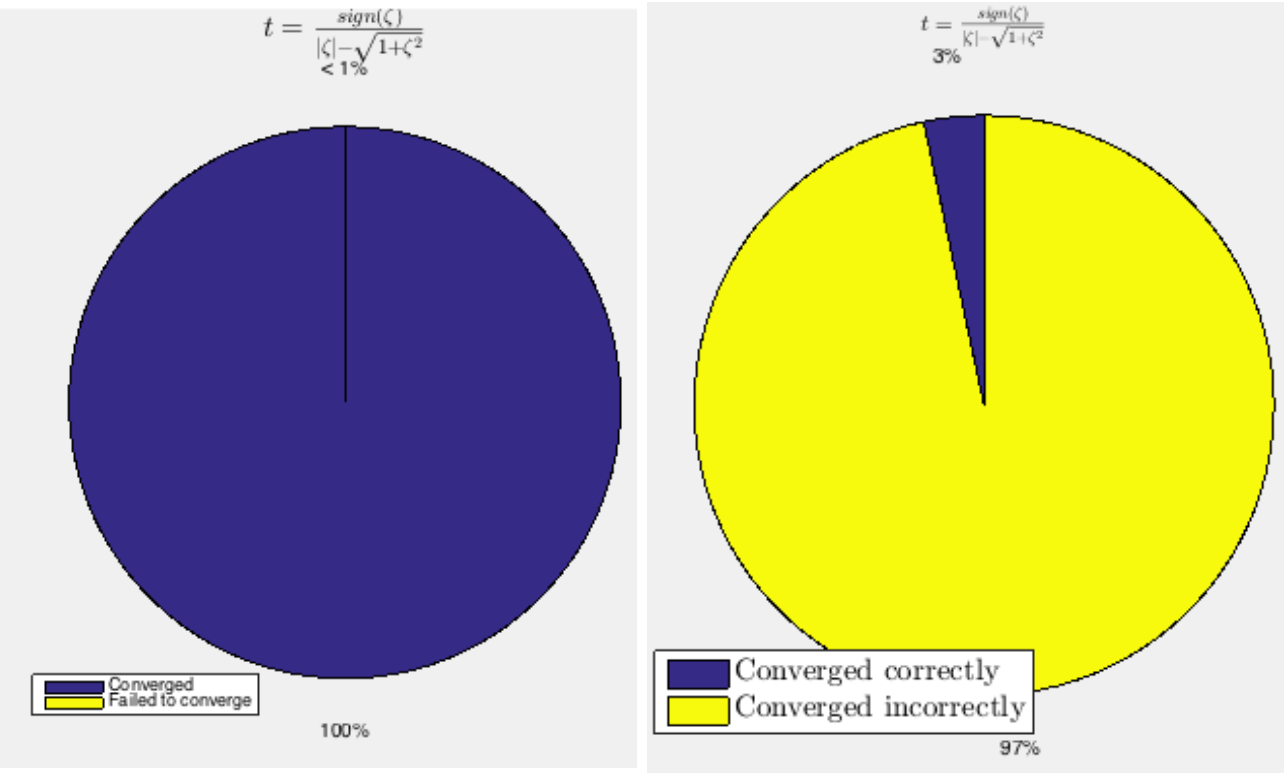


Рис. 3: Диаграмма скорости сходимости в зависимости от размерности матрицы

Однако необходимо также учесть, насколько хороший результат получает алгоритм. Сходимость к какой-то матрице сама по себе ничего не значит, необходимо, чтобы алгоритм

возвращал такие значения, которые соответствуют условиям.

Параллельно с вышеприведенной диаграммой на Рис. 3 строились и круговые диаграммы, которые показывают процент сходимости алгоритма с каждой используемой формулой и качество сходимости.



(a) Процент сходимости при $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| - \sqrt{1+\zeta^2}}$

(b) Процент правильности при $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| - \sqrt{1+\zeta^2}}$

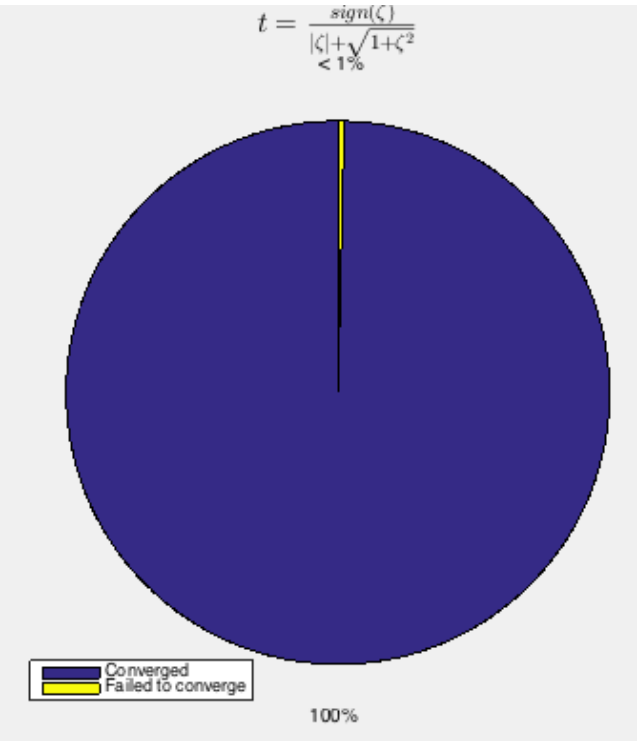
Рис. 4: Анализ сходимости при $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| - \sqrt{1+\zeta^2}}$

Проанализируем сперва самое "лучшее" представление t. На Рис. 4 представлены две круговые диаграммы, которые показывают процент сходимости алгоритма и качество сходимости при $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| - \sqrt{1+\zeta^2}}$. Из Рис. 4a видно, что алгоритм сошелся в 100% проведенных испытаниях, однако из Рис. 4b становится ясно, что алгоритм выдает неправильные значения. Причиной тому может быть неподходящее условие выхода, которое указывает нам на сходимость алгоритма.

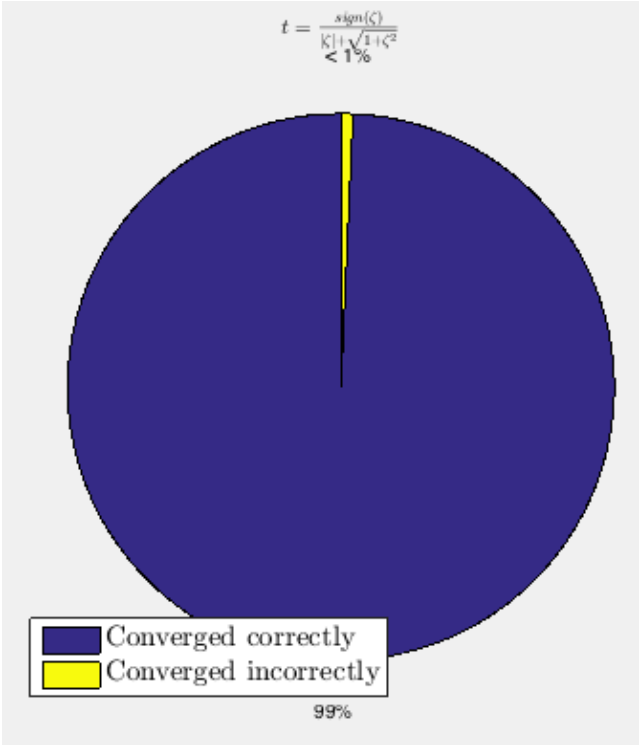
Продолжим аналогично анализировать и другие представления t.

Из Рис. 5a видно, что алгоритм сошелся в 100% проведенных испытаниях, причем из Рис. 5b становится ясно, что алгоритм выдает правильные значения в $> 99\%$ случаев; $< 1\%$ можно списать на вычислительную погрешность, т.к. вычисление признавалось ошибочным, если хотя бы один элемент полученной матрицы отличался от исходной более чем на 10^{-8}. Т.о. мы приходим к выводу, что алгоритм при $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$ работает и быстро, и точно.

Продолжая исследовать остальные два варианта представления t Рис. 7, Рис. 8, мы приходим к выводу, что $t = \dfrac{\operatorname{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$ является наиболее быстрым и точным.



(a) Процент сходимости при $t = \dfrac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$



(b) Процент правильности при $t = \dfrac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$

Рис. 5: Анализ сходимости при $t = \dfrac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$

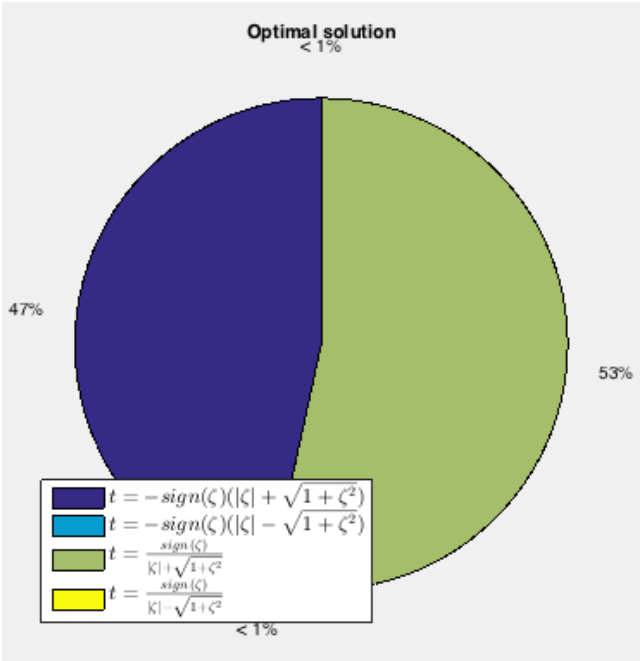
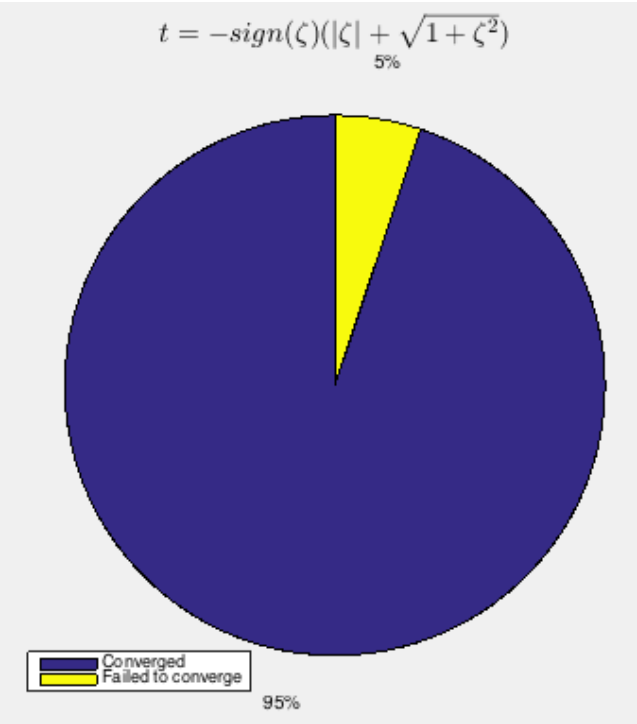
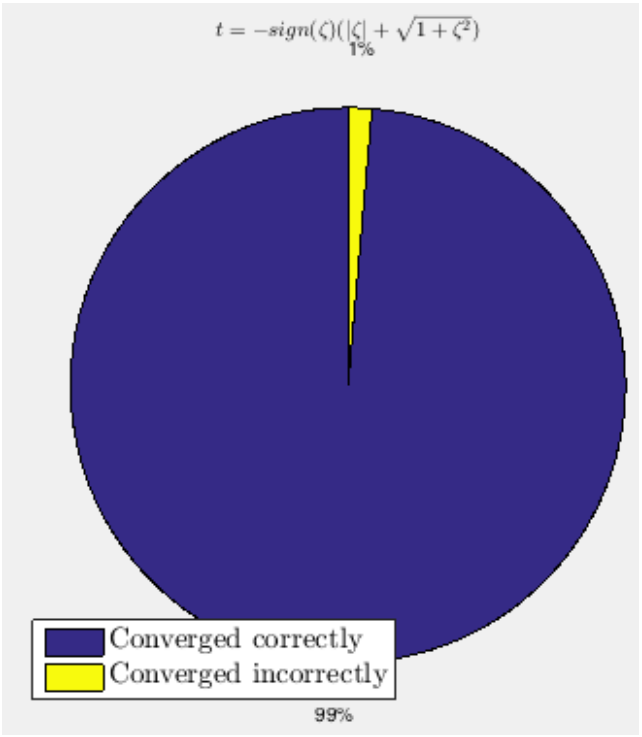


Рис. 6: Оптимальный выбор t

Т.к. на Рис. 3 показаны средние значения, проанализируем Рис. 6, на котором выводится процент оптимальности каждого из представлений. Как можно сразу заметить, представление $t = \dfrac{\text{sign}(\zeta)}{|\zeta| + \sqrt{1+\zeta^2}}$ лидирует по всем критериям, поэтому и в выводе алгоритма, и в исходном коде используется именно такое представление.

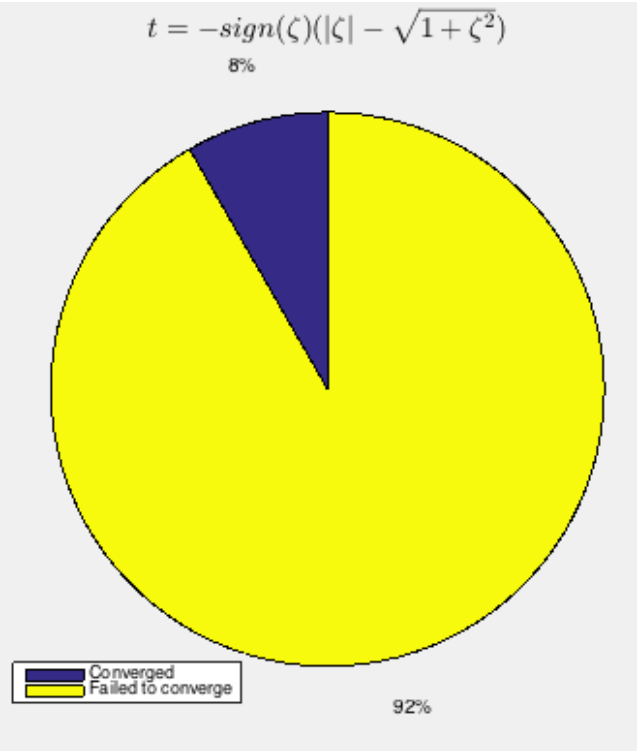


(a) Процент сходимости при $t = -\operatorname{sign}(\zeta)(|\zeta| + \sqrt{1+\zeta^2})$

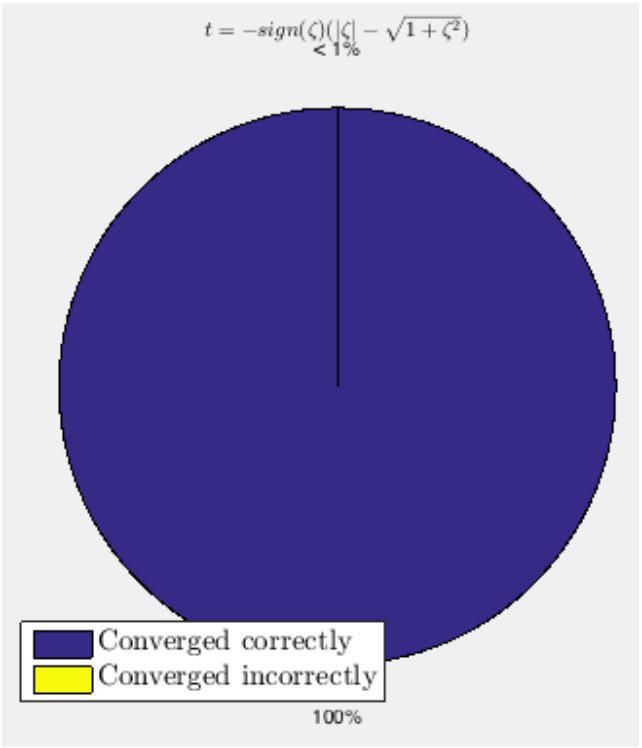


(b) Процент правильности при $t = -\operatorname{sign}(\zeta)(|\zeta| + \sqrt{1+\zeta^2})$

Рис. 7: Анализ сходимости при $t = -\operatorname{sign}(\zeta)(|\zeta| + \sqrt{1+\zeta^2})$



(a) Процент сходимости при $t = -\operatorname{sign}(\zeta)(|\zeta| - \sqrt{1+\zeta^2})$



(b) Процент правильности при $t = -\operatorname{sign}(\zeta)(|\zeta| - \sqrt{1+\zeta^2})$

Рис. 8: Анализ сходимости при $t = -\operatorname{sign}(\zeta)(|\zeta| - \sqrt{1+\zeta^2})$

5 “Стандартный” алгоритм

“Стандартный” алгоритм – самый часто используемый алгоритм, который реализован в Matlab и библиотеке линейной алгебры Lapack. Данный алгоритм является улучшением алгоритма, разработанного математиками Golub и Van Loan. Улучшенный алгоритм представлен математиками J.Demmel и W.Kahan в [6].

5.1 Описание алгоритма

“Стандартный” алгоритм происходит в два этапа:

1. Сперва происходит бидиагонализация матрицы с помощью преобразований Хаусхолдера;

2. Затем с помощью итерационного алгоритма происходит нахождение сингулярного разложения бидиагональной матрицы.

Сложность создания итерационного алгоритма заключается в том, что необходимо минимизировать одновременно и погрешность вычислений, и количество итераций. В оригинале данная задача решается тем, что на каждом шаге итерации выбирается, какой из двух алгоритмов: Golub & Van Loan’s algorithm или Demmel & Kahan’s algorithm подходит для данного случая, но в данной работе будет рассмотрен упрощенный алгоритм: Demmel & Kahan’s zero-shift algorithm.

5.1.1 Преобразование Хаусхолдера

Преобразование Хаусхолдера – линейное преобразование H_u векторного пространства V, которое описывает его отображение относительно гиперплоскости, которая проходит через начало координат.

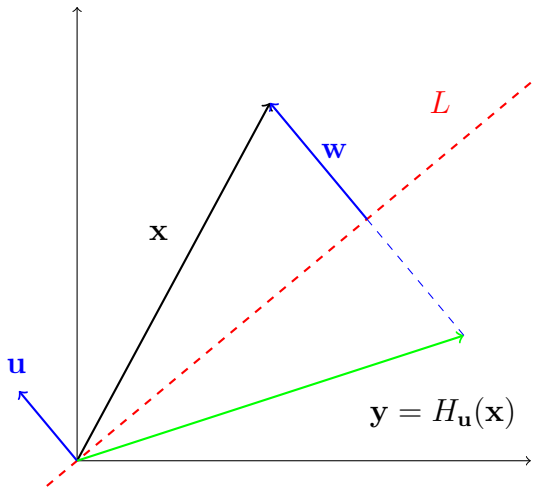


Рис. 9: Отражение

Предположим, что имеется радиус-вектор $\mathbf{x}$ в n-мерном пространстве, который необходимо отразить относительно гиперплоскости L, которая задает $(n-1)$-мерное подпространство, проходящее через начало координат. Такую гиперплоскость можно задать с помощью вектора-нормали $\mathbf{u}$.

Т.е. задача состоит в том, чтобы составить линейное преобразование $H_{\mathbf{u}}$ векторного пространства V, которое бы отражало радиус-вектор $\mathbf{x}$ относительно гиперплоскости, заданной нормалью $\mathbf{u}$:

$$\mathbf{y} = H_{\mathbf{u}}(\mathbf{x}).$$

Из рис.9 понятно, что вектор $\mathbf{w}$ определяется следующим образом:

$$\mathbf{w} = (\mathbf{x}, \mathbf{u})\mathbf{u}.$$

Также можно заметить, что

$$\mathbf{y} = \mathbf{x} - 2\mathbf{w} = \mathbf{x} - 2(\mathbf{x}, \mathbf{u})\mathbf{u}.$$

Учитывая, что скалярное произведение двух столбцов (a, b) можно представить в виде произведения строки и столбца $a^T b$, перепишем полученное равенство:

$$\mathbf{y} = \mathbf{x} - 2(\mathbf{x}^T\mathbf{u})\mathbf{u}.$$

В данном выражении $(\mathbf{x}, \mathbf{u})$ – константа, поэтому можно записать равенство в следующем виде:

$$\mathbf{y} = \mathbf{x} - 2\mathbf{u}(\mathbf{x}^T\mathbf{u}).$$

Воспользовавшись тем свойством, что $a^T b \equiv b^T a$, получим

$$\mathbf{y} = \mathbf{x} - 2\mathbf{u}(\mathbf{x}^T\mathbf{u}) = \mathbf{x} - 2\mathbf{u}(\mathbf{u}^T\mathbf{x}),$$

откуда, вынося $\mathbf{x}$, получим матрицу Хаусхолдера:

$$H_{\mathbf{u}}\mathbf{x} = \mathbf{x} - 2\mathbf{u}\mathbf{u}^T\mathbf{x} = \underbrace{(I - 2\mathbf{u}\mathbf{u}^T)}_{H_{\mathbf{u}}}\mathbf{x}.$$

$$H_{\mathbf{u}} = I - 2\mathbf{u}\mathbf{u}^T.$$

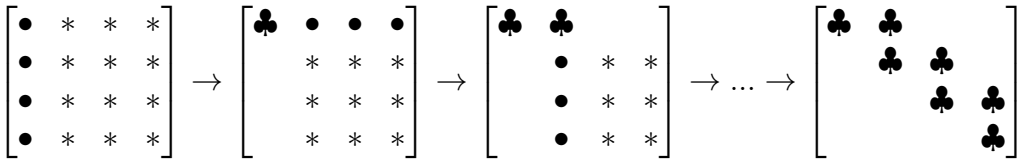
Важным свойством матрицы преобразования Хаусхолдера является тот факт, что она ортонормирована:

$$H_{\mathbf{u}}H_{\mathbf{u}}^T = (I - 2\mathbf{u}\mathbf{u}^T)(I - 2\mathbf{u}\mathbf{u}^T)^T = I - 2\mathbf{u}\mathbf{u}^T - 2\mathbf{u}\mathbf{u}^T + 4\mathbf{u}\mathbf{u}^T = I$$
$$H_{\mathbf{u}}^T H_{\mathbf{u}} = (I - 2\mathbf{u}\mathbf{u}^T)^T(I - 2\mathbf{u}\mathbf{u}^T) = I - 2\mathbf{u}\mathbf{u}^T - 2\mathbf{u}\mathbf{u}^T + 4\mathbf{u}\mathbf{u}^T = I$$

5.1.2 Бидиагонализация матрицы

Бидиагонализация с помощью преобразований Хаусхолдера – итерационный процесс с наперед заданным количеством шагов. Если матрица имеет размерность $m \times n$, то для бидиагонализации требуется $\min(m, n)$ шагов.

Покажем на примере итерационный процесс сведения матрицы $A_{4\times4}$ к бидиагональному виду:



В данном примере можно выделить 2 визуально разных преобразования:

1. обнуление элементов столбца, стоящих под главной диагональю;
2. обнуление элементов строки, стоящих правее бидиагонали.

На практике эти преобразования аналогичные, т.е. если произвести транспонирование, то обнуление элементов строки матрицы A, стоящих правее бидиагонали, становится эквивалентным обнулению элементов стобца, стоящих под главной диагональю подматрицы матрицы A^T.

Представим матрицу

$$A = \begin{bmatrix} \clubsuit & \bullet & \bullet & \bullet \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

в виде блочной матрицы вида

$$A = \left[\begin{array}{c|ccc} \clubsuit & \bullet & \bullet & \bullet \\ & * & * & * \\ & * & * & * \\ \hline & * & * & * \end{array}\right] = \left[\begin{array}{c|ccc} \clubsuit & & \widetilde{A} & \\ \hline & * & * & * \end{array}\right],$$

где матрица $\widetilde{A}$ записывается в виде:

$$\widetilde{A} = \begin{bmatrix} \bullet & \bullet & \bullet \\ * & * & * \\ * & * & * \end{bmatrix}.$$

Тогда, транспонировав матрицу $\widetilde{A}$

$$\widetilde{A}^T = \begin{bmatrix} \bullet & * & * \\ \bullet & * & * \\ \bullet & * & * \end{bmatrix},$$

мы сведем задачу обнуления элементов строки матрицы A, стоящих правее бидиагонали, к задаче обнуления элементов столбца, стоящих под главной диагональю матрицы $\widetilde{A}^T$.

Покажем, как с помощью преобразований Хаусхолдера обнулить элементы столбца, стоящие под главной диагональю. Рассмотрим для простоты матрицу

$$A_{2\times 2} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \end{bmatrix},$$

столбцы которой являются радиус-векторами в пространстве $\mathbb{R}^2$. Необходимо найти такое преобразование Хаусхолдера $H_{\mathbf{u}}$, которое бы выполняло следующее действие:

$$H_{\mathbf{u}} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} \widetilde{x}_1^{(1)} \\ 0 \end{bmatrix}$$

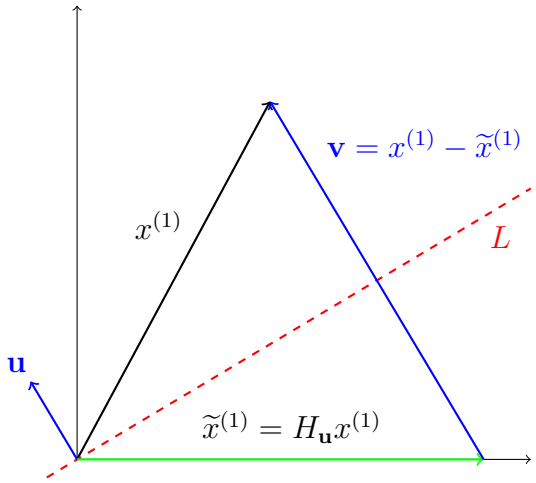


Рис. 10: Обнуление x_2-координаты

Зная исходный вектор и вид получаемого вектора, можно найти вектор $\mathbf{u}$, задающий матрицу преобразования Хаусхолдера. На рис.10 представлена геометрическая интерпретация требуемого преобразования. Т.к. преобразование Хаусхолдера является изометрическим, то

$$||x^{(1)}|| = ||\widetilde{x}^{(1)}||.$$

Это условие можно переписать в виде

$$H_{\mathbf{u}} x^{(1)} = \pm ||x^{(1)}|| e_1.$$

Для того, чтобы найти $\mathbf{u}$, необходимо выразить $\mathbf{v}$, причем необходимо учитывать тот факт, что норма вектора $\mathbf{v}$ стоит в знаменателе, а значит, надо выбрать такое представление, чтобы избежать деления на близкое к нулю значение:

$$\mathbf{v} = x^{(1)} - \widetilde{x}^{(1)} = x^{(1)} - \pm ||x^{(1)}|| e_1 = x^{(1)} \mp ||x^{(1)}|| e_1 = x^{(1)} + \operatorname{sign}(x_1^{(1)}) ||x^{(1)}|| e_1.$$

Учитывая, что $\mathbf{u} = \dfrac{\mathbf{v}}{||\mathbf{v}||}$, получаем выражение:

$$\mathbf{u} = \frac{x^{(1)} + \operatorname{sign}(x_1^{(1)})||x^{(1)}||e_1}{||x^{(1)} + \operatorname{sign}(x_1^{(1)})||x^{(1)}||e_1||}.$$

Описание алгоритма: пусть дана матрица $M_{m\times n}$, которую необходимо привести к бидиагональному виду.

$$M = \begin{bmatrix} * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \end{bmatrix}$$

На каждом шаге обнуляются элементы одного столбца и одной строки. Рассмотрим первые 2 полных шага:

Пусть $A^{(1.1)} = M$. Обнулим элементы первого столбца матрицы

$$A^{(1.1)} = \left[\begin{array}{c|cccc} \overset{a_1}{\bullet} & * & * & * & \dots & * \\ \bullet & * & * & * & \dots & * \\ \bullet & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \bullet & * & * & * & \dots & * \\ \bullet & * & * & * & \dots & * \end{array}\right] \to \left[\begin{array}{c|cccc} \overset{\widetilde{a}_1}{\clubsuit} & \bullet & \bullet & \bullet & \dots & \bullet \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ \hline 0 & * & * & * & \dots & * \end{array}\right] = H_1^1 A^{(1.1)} = \widetilde{A}^{(1.1)}$$

Процесс конструирования матрицы преобразования Хаусхолдера был только что описан, поэтому воспользуемся готовой формулой. Учитывая, что $x^{(1)} := a_1$, найдем вектор $\mathbf{u}$, однозначно задающий матрицу преобразования:

$$\mathbf{u} = \frac{a_1 + \operatorname{sign}(a_{11})||a_1||e_1}{||a_1 + \operatorname{sign}(a_{11})||a_1||e_1||}; \quad H_1^1 = I - 2\mathbf{u}\mathbf{u}^T.$$

Т.о. применяя данное преобразование к матрице $A^{(1.1)}$, обнуляются элементы первого столбца под главной диагональю.

Сведем задачу обнуления элементов первой строки, стоящих правее бидиагонали, к задаче обнуления элементов первой строки, стоящих правее главной диагонали. Для этого достаточно рассмотреть подматрицу $\widetilde{A}_2$ матрицы $\widetilde{A}^{(1.1)}$. (Причем, если размерность матрицы $\widetilde{A}^{(1.1)}$ – $m \times n$, то размерность подматрицы $\widetilde{A}_2$ – $(m-1)\times(n-1)$).

$$\widetilde{A}_2 = \left[\begin{array}{ccccc} \bullet & \bullet & \bullet & \dots & \bullet \\ \hline * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & * \end{array}\right]$$

Сведем задачу обнуления элементов строки матрицы $\widetilde{A}_2$, стоящих правее главной диагонали к задаче обнуления элементов столбца, стоящих под главной диагональю матрицы

$A^{(1.2)} = \widetilde{A}_2^T$.

$$A^{(1.2)} = \overset{\begin{matrix} a_1 & & A_2 \end{matrix}}{\left[\begin{array}{c|ccc} \bullet & * & \dots & * \\ \bullet & * & \dots & * \\ \bullet & * & \dots & * \\ \vdots & \vdots & & \vdots \\ \bullet & * & \dots & * \end{array}\right]} \to \overset{\begin{matrix} \widetilde{a}_1 & & \widetilde{A}_2 \end{matrix}}{\left[\begin{array}{c|ccc} \clubsuit & \bullet & \dots & \bullet \\ 0 & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \dots & * \end{array}\right]} = H_2^1 A^{(1.2)} = \widetilde{A}^{(1.2)}$$

Транспонировав обратно матрицу $\widetilde{A}^{(1.2)}$, мы получим матрицу с обнуленными элементами правее главной диагонали:

$$(\widetilde{A}^{(1.2)})^T = \left[\begin{array}{c|ccc} \clubsuit & 0 & \dots & 0 \\ \hline \bullet & * & \dots & * \\ \bullet & * & \dots & * \\ \vdots & \vdots & & \vdots \\ \bullet & * & \dots & * \end{array}\right]$$

$$\left.\begin{array}{r} A^{(1.2)} = \widetilde{A}_2^T \\ H_2^1 A^{(1.2)} = \widetilde{A}^{(1.2)} \end{array}\right\} \Rightarrow \left.\begin{array}{r} (A^{(1.2)})^T = \widetilde{A}_2 \\ (A^{(1.2)})^T (H_2^1)^T = (\widetilde{A}^{(1.2)})^T \end{array}\right\} \Rightarrow \widetilde{A}_2 (H_2^1)^T = (\widetilde{A}^{(1.2)})^T$$

Составим такие матрицы $H^{(1.1)}, H^{(1.2)}$, чтобы выполнялось следующее равенство:

$$H^{(1.1)} \begin{bmatrix} * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \end{bmatrix} (H^{(1.2)})^T = \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{bmatrix}$$

Мы нашли матрицу $H^{(1.1)} := H_1^1$, которая обнуляет элементы первого столбца, стоящие под главной диагональю:

$$H^{(1.1)} : \underbrace{\begin{bmatrix} * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \end{bmatrix}}_{m \times n} \to \underbrace{\begin{bmatrix} \clubsuit & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{bmatrix}}_{m \times n},$$

а также матрицу H_2^1, которая обнуляет элементы первой строки, стоящих правее бидиа-

гонали матрицы, полученной после применения преобразования $H^{(1.1)}$:

$$H_2^1 : \underbrace{\begin{bmatrix} * & * & * & \dots & * \\ * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & * \end{bmatrix}}_{(m-1)\times(n-1)} \to \underbrace{\begin{bmatrix} \clubsuit & 0 & 0 & \dots & 0 \\ * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & * \end{bmatrix}}_{(m-1)\times(n-1)}$$

Представим матрицу $H^{(1.1)}A^{(1.1)} = \widetilde{A}^{(1.1)}$ в виде блочной матрицы

$$\widetilde{A}^{(1.1)} = \left[\begin{array}{c|c} B_{(m-1)\times 1} & A^{(1.2)}_{(m-1)\times(n-1)} \\ \hline O_{1\times 1} & *_{1\times(n-1)} \end{array}\right],$$

для которой надо составить такую матрицу $H^{(1.2)}$, чтобы

$$\left[\begin{array}{c|c} B_{(m-1)\times 1} & A^{(1.2)}_{(m-1)\times(n-1)} \\ \hline O_{1\times 1} & *_{1\times(n-1)} \end{array}\right] (H^{(1.2)})^T = \left[\begin{array}{c|c} B_{(m-1)\times 1} & \widetilde{A}^{(1.2)}_{(m-1)\times(n-1)} \\ \hline O_{1\times 1} & *_{1\times(n-1)} \end{array}\right].$$

Подходящей матрицей является матрица вида

$$H^{(1.2)} = \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_2^1)_{(n-1)\times(n-1)} \end{array}\right]$$

Покажем, что $H^{(1.2)}$ – именно та матрица, которую мы ищем:

$$(H^{(1.2)})^T = \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_2^1)_{(n-1)\times(n-1)} \end{array}\right]^T = \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_2^1)^T_{(n-1)\times(n-1)} \end{array}\right]$$

$$\left[\begin{array}{c|c} B_{(m-1)\times 1} & A^{(1.2)}_{(m-1)\times(n-1)} \\ \hline O_{1\times 1} & *_{1\times(n-1)} \end{array}\right] \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_2^1)^T_{(n-1)\times(n-1)} \end{array}\right] = \left[\begin{array}{c|c} B_{(m-1)\times 1} & (\widetilde{A}^{(1.2)})^T_{(m-1)\times(n-1)} \\ \hline O_{1\times 1} & *_{1\times(n-1)} \end{array}\right].$$

$$\begin{cases} B_{(m-1)\times 1} \cdot I_{1\times 1} + A^{(1.2)}_{(m-1)\times(n-1)} \cdot O_{(n-1)\times 1} = B_{(m-1)\times 1} \\ B_{(m-1)\times 1} \cdot O_{1\times(n-1)} + A^{(1.2)}_{(m-1)\times(n-1)} \cdot (H_2^1)^T_{(n-1)\times(n-1)} = (\widetilde{A}^{(1.2)})^T_{(m-1)\times(n-1)} \\ O_{1\times 1} \cdot I_{1\times 1} + *_{1\times(n-1)} \cdot O_{(n-1)\times 1} = O_{1\times 1} \\ O_{1\times 1} \cdot O_{1\times(n-1)} + *_{1\times(n-1)} \cdot (H_2^1)^T_{(n-1)\times(n-1)} = *_{1\times(n-1)} \end{cases}$$

Т.о. матрица, получающаяся в результате преобразования $H^{(1)}, H^{(2)}$ матрица, удовлетворяет требованиям: в первом столбце элементы, стоящие под главной диагональю, равны нулю; в первой строке элементы, стоящие правее бидиагонали, также равны нулю.

$$H^{(1.1)}A^{(1.1)}(H^{(1.2)})^T = H^{(1.1)} \begin{bmatrix} * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ * & * & * & * & \dots & * \\ * & * & * & * & \dots & * \end{bmatrix} (H^{(1.2)})^T = \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{bmatrix} = A^{(2.1)}$$

Первый шаг алгоритма полностью выполнен. На втором шаге алгоритм будет работать уже с матрицей $A^{(2.1)}$, для того, чтобы привести ее к виду

$$A^{(2.1)} = \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{bmatrix} \to \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & \clubsuit & \clubsuit & 0 & \dots & 0 \\ 0 & 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & * & \dots & * \\ 0 & 0 & * & * & \dots & * \end{bmatrix} = A^{(3.1)}$$

Матрицу $A^{(2.1)}$ можно представить в виде блочной матрицы

$$A^{(2.1)} = \left[\begin{array}{cc|cccc} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{array}\right] = \left[\begin{array}{c|ccccc} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ \hline 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{array}\right] = \left[\begin{array}{c|ccccc} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ \hline 0 & & & & & \\ 0 & & & & & \\ \vdots & & & M_2 & & \\ 0 & & & & & \\ 0 & & & & & \end{array}\right]$$

$$A^{(2.1)} = \left[\begin{array}{c|c} D_{1\times 1} & R_{1\times(n-1)} \\ \hline O_{(m-1)\times 1} & (M_2)_{(m-1)\times(n-1)} \end{array}\right]$$

Повторяя первый шаг, находим матрицы H_1^2, H_2^2 для матрицы M_2. Однако искомые матрицы $H^{(2.1)}, H^{(2.2)}$ таковы, что

$$H^{(2.1)} \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & * & \dots & * \\ 0 & * & * & * & \dots & * \end{bmatrix} (H^{(2.2)})^T = \begin{bmatrix} \clubsuit & \clubsuit & 0 & 0 & \dots & 0 \\ 0 & \clubsuit & \clubsuit & 0 & \dots & 0 \\ 0 & 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & * & \dots & * \\ 0 & 0 & * & * & \dots & * \end{bmatrix}$$

Т.е. они должны быть сконструированы таким образом, чтобы произвести необходимое преобразование подматрицы M_2, при этом не изменив D и R. Представим матрицы $H^{(2.1)}, H^{(2.2)}$ в виде блочных матриц:

$$H^{(2.1)} = \underbrace{\left[\begin{array}{c|c} H_1^{(2.1)} & H_2^{(2.1)} \\ \hline H_3^{(2.1)} & H_4^{(2.1)} \end{array}\right]}_{m\times m} \qquad H^{(2.2)} = \underbrace{\left[\begin{array}{c|c} H_1^{(2.2)} & H_2^{(2.2)} \\ \hline H_3^{(2.2)} & H_4^{(2.2)} \end{array}\right]}_{n\times n}$$

Найдем сперва элементы матрицы $H^{(2.1)}$:

$$\left[\begin{array}{c|c} H_1^{(2.1)} & H_2^{(2.1)} \\ \hline H_3^{(2.1)} & H_4^{(2.1)} \end{array}\right] \left[\begin{array}{c|c} D_{1\times 1} & R_{1\times(n-1)} \\ \hline O_{(m-1)\times 1} & (M_2)_{(m-1)\times(n-1)} \end{array}\right] = \left[\begin{array}{c|c} D_{1\times 1} & R_{1\times(n-1)} \\ \hline O_{(m-1)\times 1} & (\widetilde{M_2})_{(m-1)\times(n-1)} \end{array}\right]$$

Составим систему уравнений, откуда найдем элементы блочной матрицы $H^{(2.1)}$ и их размерности.

$$\begin{cases} H_1^{(2.1)} \cdot D_{1\times 1} + H_2^{(2.1)} \cdot O_{(m-1)\times 1} = D_{1\times 1} \\ H_1^{(2.1)} \cdot R_{1\times(n-1)} + H_2^{(2.1)} \cdot (M_2)_{(m-1)\times(n-1)} = R_{1\times(n-1)} \\ H_3^{(2.1)} \cdot D_{1\times 1} + H_4^{(2.1)} \cdot O_{(m-1)\times 1} = O_{(m-1)\times 1} \\ H_3^{(2.1)} \cdot R_{1\times(n-1)} + H_4^{(2.1)} \cdot (M_2)_{(m-1)\times(n-1)} = (\widetilde{M_2})_{(m-1)\times(n-1)} \end{cases} \Rightarrow \begin{cases} H_1^{(2.1)} = I_{1\times 1} \\ H_2^{(2.1)} = O_{1\times(m-1)} \\ H_3^{(2.1)} = O_{(m-1)\times 1} \\ H_4^{(2.1)} = (H_1^2)_{(m-1)\times(m-1)} \end{cases}$$

$$H^{(2.1)} = \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(m-1)} \\ \hline O_{(m-1)\times 1} & (H_1^2)_{(m-1)\times(m-1)} \end{array}\right] = \left[\begin{array}{c|c} I_1 & O \\ \hline O & H_1^2 \end{array}\right]$$

Найдем элементы матрицы $H^{(2.2)}$ (необходимо помнить, что матрица транспонируется):

$$\left[\begin{array}{c|c} D_{1\times 1} & R_{1\times(n-1)} \\ \hline O_{(m-1)\times 1} & (\widetilde{M_2})_{(m-1)\times(n-1)} \end{array}\right] \left[\begin{array}{c|c} H_1^{(2.2)} & H_3^{(2.2)} \\ \hline H_2^{(2.2)} & H_4^{(2.2)} \end{array}\right] = \left[\begin{array}{c|c} D_{1\times 1} & R_{1\times(n-1)} \\ \hline O_{(m-1)\times 1} & (\widetilde{M_2})_{(m-1)\times(n-1)} \end{array}\right]$$

Составим вторую систему уравнений, откуда найдем элементы блочной матрицы $H^{(2.2)}$ и их размерности.

$$\begin{cases} D_{1\times 1} \cdot H_1^{(2.2)} + R_{1\times(n-1)} \cdot H_2^{(2.2)} = D_{1\times 1} \\ D_{1\times 1} \cdot H_3^{(2.2)} + R_{1\times(n-1)} \cdot H_4^{(2.2)} = R_{1\times(n-1)} \\ O_{(m-1)\times 1} \cdot H_1^{(2.2)} + (\widetilde{M_2})_{(m-1)\times(n-1)} \cdot H_2^{(2.2)} = O_{(m-1)\times 1} \\ O_{(m-1)\times 1} \cdot H_3^{(2.2)} + (\widetilde{M_2})_{(m-1)\times(n-1)} \cdot H_4^{(2.2)} = (\widetilde{A}^{(2.1)})_{(m-1)\times(n-1)} \end{cases} \Rightarrow \begin{cases} H_1^{(2.1)} = I_{1\times 1} \\ H_2^{(2.1)} = O_{(n-1)\times 1} \\ H_3^{(2.1)} = O_{1\times(n-1)} \\ H_4^{(2.1)} = (H_1^2)_{(n-1)\times(n-1)} \end{cases}$$

В данном случае следует отдельно разобрать второе уравнение, в котором утверждается, что

$$R_{1\times(n-1)} \cdot H_4^{(2.2)} = R_{1\times(n-1)}.$$

Когда мы искали вид матрицы $H^{(1.2)}$, было показано, что она имеет вид:

$$H^{(1.2)} = \left[\begin{array}{c|c} 1 & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_2^1)_{(n-1)\times(n-1)} \end{array}\right].$$

Подматрица $R_{1\times(n-1)}$ имеет вид:

$$R_{1\times(n-1)} = \left[\begin{array}{ccccc} \clubsuit & 0 & 0 & \dots & 0 \end{array}\right],$$

т.е. $R_{1\times(n-1)}$ инвариантна относительно преобразования $H_4^{(2.2)}$.

Таким образом, матрица $H^{(2.2)}$ имеет вид:

$$H^{(2.2)} = \left[\begin{array}{c|c} I_{1\times 1} & O_{1\times(n-1)} \\ \hline O_{(n-1)\times 1} & (H_1^2)_{(n-1)\times(n-1)} \end{array}\right] = \left[\begin{array}{c|c} I_1 & O \\ \hline O & H_1^2 \end{array}\right]$$

Только что было показано, что исходная матрица M соотносится с матрицей, полученной после двух шагов $A^{(3.1)}$ следующим образом:

$$H^{(2.1)}H^{(1.1)} \cdot M \cdot (H^{(1.1)})^T(H^{(2.1)})^T = (H^{(2.1)}H^{(1.1)}) \cdot M \cdot (H^{(2.1)}H^{(1.1)})^T = A^{(3.1)}$$

Продолжая аналогичные рассуждения, можно написать формулы нахождения матриц $H^{(k.1)}, H^{(k.2)}$ на k-ом шаге:

$$H^{(k.1)} = \left[\begin{array}{c|c} I_{(k-1)\times(k-1)} & O_{(n-k)\times(k-1)} \\ \hline O_{(k-1)\times(n-k)} & (H_1^k)_{(n-k)\times(n-k)} \end{array}\right] = \left[\begin{array}{c|c} I_{k-1} & O \\ \hline O & H_1^k \end{array}\right]$$

$$H^{(k.2)} = \left[\begin{array}{c|c} I_{(k-1)\times(k-1)} & O_{(k-1)\times(n-k)} \\ \hline O_{(n-k)\times(k-1)} & (H_2^k)_{(n-k)\times(n-k)} \end{array}\right] = \left[\begin{array}{c|c} I_{k-1} & O \\ \hline O & H_2^k \end{array}\right]$$

Тогда матрица $A^{((k+1).1)}$ находится по формуле:

$$A^{((k+1).1)} = \left(H^{(k.1)} \cdot \ldots \cdot H^{(1.1)} \right) \cdot M \cdot \left(H^{(k.2)} \cdot \ldots \cdot H^{(1.2)} \right)^T.$$

Рассмотрим бидиагонализацию трех возможных видов матриц. Следует отдельно отметить тот факт, что ранг матрицы не влияет на работу алгоритма, т.е. вне зависимости от того $\operatorname{rank}(A) < \min(m, n)$ или $\operatorname{rank}(A) = \min(m, n)$, количество шагов работы алгоритма одинаково в обоих случаях (алгоритм инвариантен относительно ранга матрицы).

Первым делом рассмотрим самый простой случай, когда $m = n$, т.е. матрица квадратная.

$$\overbrace{}^{n \text{ cols}}$$
$$n \text{ rows} \left\{ \begin{bmatrix} * & \ldots & * \\ \vdots & \ddots & \vdots \\ * & \ldots & * \end{bmatrix} \right.$$

На каждом шаге алгоритм работает с одним столбцом и одной строкой. У квадратной матрицы $n \times n$ необходимо обработать $(n-1)$ столбец и $(n-2)$ строки, следовательно бидиагональная матрица B получается из исходной матрицы A следующим образом:

$$B = \left(H^{((n-1).1)} \cdot \ldots \cdot H^{(1.1)} \right) \cdot A \cdot \left(H^{((n-2).2)} \cdot \ldots \cdot H^{(1.2)} \right)^T.$$

Воспользовавшись условием ортонормированности матриц преобразований Хаусхолдера, выпишем разложение матрицы A в виде произведения двух ортонормированных и одной бидиагональной матрицы:

$$A = UBV^T,$$
$$U = \left(H^{((n-1).1)} \cdot \ldots \cdot H^{(1.1)} \right)^T; \quad V = \left(H^{((n-2).2)} \cdot \ldots \cdot H^{(1.2)} \right)^T$$

$$\overbrace{}^{m \text{ cols}} \overbrace{}^{m-n \text{ cols}}$$
$$m \text{ rows} \left\{ \left[\begin{array}{ccc:ccc} * & \ldots & * & * & \ldots & * \\ \vdots & \ddots & \vdots & * & \ldots & * \\ * & \ldots & * & * & \ldots & * \end{array} \right] \right.$$

Рассмотрим случай, когда $m < n$. Из графического представления матрицы, у которой $m < n$ видно, что исходную матрицу $A_{m \times n}$ можно представить в виде блочной матрицы

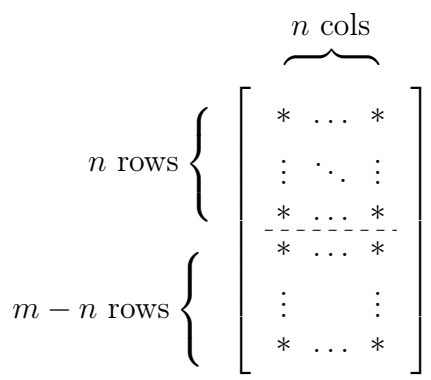
$$A = \left[\; S_{m \times m} \mid C \; \right].$$

Т.к. на каждом шаге алгоритм работает с одним столбцом и одной строкой, а под главной диагональю находится $(m-1)$ столбец, то очевидно, что

$$U = \left(H^{((m-1).1)} \cdot \ldots \cdot H^{(1.1)} \right)^T.$$

Строки матрицы A имеют больше элементов по сравнению с квадратной матрицей, поэтому потребуется обрабатывать больше строк. Если подматрица C является столбцом, то обрабатывается на одну строку больше по сравнению с квадратной матрицей, если C – матрица, имеющая ≥ 2 количество столбцов, то обрабатываются все m строк, и матрица V принимает вид:

$$V = \left(H^{(m.2)} \cdot \ldots \cdot H^{(1.2)} \right)^T.$$



Последний случай, когда $m > n$ похож на предыдущий, за тем исключением, что теперь у столбцов больше элементов, которые необходимо обнулить. В данном случае бидиагональная матрица B получается из исходной матрицы A следующим образом:

$$B = \left(H^{(n.1)} \cdot ... \cdot H^{(1.1)} \right) \cdot A \cdot \left(H^{((n-2).2)} \cdot ... \cdot H^{(1.2)} \right)^T.$$

Главное отличие от случая с квадратной матрицей – обрабатывается последний столбец. Выпишем отдельно матрицы U, V:

$$A = UBV^T,$$
$$U = \left(H^{(n.1)} \cdot ... \cdot H^{(1.1)} \right)^T; \quad V = \left(H^{((n-2).2)} \cdot ... \cdot H^{(1.2)} \right)^T$$

5.1.3 Вращения Гивенса

Очень важной частью “стандартного” алгоритма является эффективное вычисление элементов матриц вращений Гивенса. Более подробно описать задачу построения матриц Гивенса можно следующим образом: пусть имеется вектор $(f, g)^T$, необходимо образовать матрицу поворота, которая приведет вектор вида $(f, g)^T$ к виду $(r, 0)^T$:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad \left| \begin{array}{l} c := \cos\theta \\ s := \sin\theta \end{array} \right. \tag{5.1.1}$$

Предположим, что $f > g$. В таком случае:

$$t := \tan\theta = \frac{g}{f}.$$

Перепишем матричное уравнение (5.1.1) в виде системы алгебраических уравнений:

$$\begin{cases} cf - sg = r \\ sf + cg = 0 \Leftrightarrow g = -\dfrac{s}{c} f \end{cases} \tag{5.1.2}$$

Т.о. r вычисляется по формуле:

$$r = cf + \frac{s^2}{c} f = \frac{c^2 f + s^2 f}{c} = \frac{f}{c}.$$

Теперь остается вспомнить лишь тот факт, что $\cos\theta = \dfrac{1}{\sqrt{1 + \tan^2\theta}}$ и подставить в предыдущую формулу:

$$r = f\sqrt{1 + t^2}$$

При условии, что $g > f$ все рассуждения аналогичны за тем исключением, что элементы векторов $(f, g)^T$ и $(r, 0)^T$ записываются в обратном порядке:

$$\begin{bmatrix} f \\ g \end{bmatrix} \to \begin{bmatrix} g \\ f \end{bmatrix}; \quad \begin{bmatrix} r \\ 0 \end{bmatrix} \to \begin{bmatrix} 0 \\ r \end{bmatrix}$$

В работе авторов [6] более подробно описывается данный алгоритм, а также изучается погрешность и накопление ошибки.

5.1.4 "Стандартный" алгоритм

Сам "стандартный" алгоритм собирается как конструктор из вышеописанных частей. Сингулярное разложение разбивается на две большие независимые части:

1. Бидиагонализация матрицы.
   Замечание 1: "стандартный" алгоритм работает с вертикальными бидиагональными матрицами, т.е. матрицами $B_{m\times n}$, $m \geq n$. В противном случае перед бидиагонализацией исходную матрицу необходимо транспонировать.
   Замечание 2: возможна работа с горизонтальными матрицами, однако в таком случае необходимо привести матрицу к нижнему бидиагональному виду и соответствующим образом переопределить QR скачки.

2. Использование QR скачков для диагонализации бидиагональной матрицы.

Предположим, что $J^{(0)}$ – бидиагональная матрица. Разновидностью QR-алгоритма можно итерационно диагонализировать бидиагональную матрицу $J^{(0)}$:

$$J^{(0)} \to J^{(1)} \to ... \to J^{(n)} \to ... \to \Sigma, \quad J^{(i+1)} = S^{(i)^T} J^{(i)} T^{(i)},$$

где преобразования $S^{(i)}, T^{(i)}$ – ортонормированные. Матрицы $T^{(i)}$ выбираются таким образом, чтобы последовательность матриц $M^{(i)} = J^{(i)^T} J^{(i)}$ сходилась к диагональной матрице, в то время как матрицы $S^{(i)}$ выбираются такими, чтобы все $J^{(i)}$ были бидиагональными. Введем обозначения:

$$J \equiv J^{(i)}, \quad \bar{J} \equiv J^{(i+1)}, \quad S \equiv S^{(i)}, \quad T \equiv T^{(i)}, \quad M \equiv J^T J, \quad \bar{M} \equiv \bar{J}^T \bar{J}.$$

Преобразование $J \to \bar{J}$ достигается с помощью применения поворотов Гивенса к матрице J с обоих сторон:

$$\bar{J} = \underbrace{S_n S_{(n-1)} ... S_2}_{S} \, J \, \underbrace{T_2^T T_3^T ... T_n^T}_{T^T},$$

где S_i, T_i – матрицы поворота, задающиеся по определенному правилу:

T_2^T обнуляет элемент $J_{1,2}$; порождает элемент $J_{2,1}$
S_2 обнуляет элемент $J_{2,1}$; порождает элементы $J_{1,2}, J_{1,3}$
T_3^T обнуляет элементы $J_{1,3}, J_{2,3}$; порождает элемент $J_{3,2}$
. . . . . .
T_n^T обнуляет элементы $J_{n-2,n}, J_{n-1,n}$; порождает элемент $J_{n,n-1}$
S_n обнуляет элемент $J_{n,n-1}$; порождает элемент $J_{n-1,n}$

Рассмотрим качественно, что происходит в данных преобразованиях, на примере матрицы 4×4:

$$J = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}, \quad JT_2^T = \begin{bmatrix} x & 0 & 0 & 0 \\ + & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}, \quad S_2 JT_2^T = \begin{bmatrix} x & x & + & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix},$$

$$S_2 JT_2^T T_3^T = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & + & x & x \\ 0 & 0 & 0 & x \end{bmatrix}, \quad S_3 S_2 JT_2^T T_3^T = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & + \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix},$$

$$S_3S_2JT_2^TT_3^TT_4^T = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & + & x \end{bmatrix}, \qquad \bar{J} = S_4S_3S_2JT_2^TT_3^TT_4^T = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{bmatrix}$$

В [6] доказывается, что вышеприведенные преобразования итеративно приближают матрицу J к диагональному виду:

$$\lim_{n\to\infty} J^{(n)} = \Sigma.$$

Таким образом набор преобразований, диагонализирующий бидиагональную матрицу B, выглядит следующим образом:

$$\Sigma = \underbrace{S_n^{(N)} \cdot \ldots \cdot S_2^{(N)} \cdot \ldots \cdot S_n^{(1)} \cdot \ldots \cdot S_2^{(1)}}_{S} B \underbrace{T_2^{(1)^T} \cdot \ldots \cdot T_n^{(1)^T} \cdot \ldots \cdot T_2^{(N)^T} \cdot \ldots \cdot T_n^{(N)^T}}_{T^T}, \quad N \to \infty.$$

В (5.2.4) приведен исходный код, реализующий одну итерацию QR скачка. Однако в статье [6] приведен более эффективный метод реализации скачков, основанный на том, что работа происходит не со всей бидиагональной матрицей, а лишь с двумя ненулевыми диагоналями, которые задаются двумя векторами.

В данной работе рассматривался алгоритм с нулевым сдвигом. Алгоритм "стандартного" сингулярного разложения с ненулевым сдвигом подробно рассмотрен в работе [9].

Рассмотрим на примере построение матриц поворота $S_i,\ T_i$ с помощью разобранных в разделе (5.1.3) вращений Гивенса. Пусть дана исходная верхняя бидиагональная матрица $B_{0_{5\times4}}$:

$$B_0 = \begin{bmatrix} d_1 & u_1 & 0 & 0 \\ 0 & d_2 & u_2 & 0 \\ 0 & 0 & d_3 & u_3 \\ 0 & 0 & 0 & d_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \begin{aligned} d_i &- i\text{-ый диагональный элемент матрицы } B; \\ u_i &- i\text{-ый наддиагональный элемент матрицы } B; \\ l_i &- i\text{-ый поддиагональный элемент матрицы } B. \end{aligned}$$

Первой создается матрица Гивенса T_2. Матрица T_2 строится таким образом, чтобы обнулить элемент u_1 матрицы B_0. Для этого достаточно воспользоваться теорией из раздела (5.1.3):

$$\begin{bmatrix} d_1 & u_1 \end{bmatrix} \to \begin{bmatrix} d_1^* & 0 \end{bmatrix}. \tag{5.1.3}$$

Покажем, что задача (5.1.3) эквивалентна задаче, рассмотренной в (5.1.3).

$$\begin{bmatrix} d_1 & u_1 \end{bmatrix} \underbrace{\begin{bmatrix} c & -s \\ s & c \end{bmatrix}}_{T_2^T} = \left(\left(\begin{bmatrix} d_1 & u_1 \end{bmatrix} \underbrace{\begin{bmatrix} c & -s \\ s & c \end{bmatrix}}_{T_2^T} \right)^T \right)^T = \left(\underbrace{\begin{bmatrix} c & s \\ -s & c \end{bmatrix}}_{T_2} \begin{bmatrix} d_1 \\ u_1 \end{bmatrix} \right)^T \to \begin{bmatrix} d_1^* \\ 0 \end{bmatrix}^T = \begin{bmatrix} d_1^* & 0 \end{bmatrix}.$$

Таким образом находим значения $c := \cos\theta,\ s := \sin\theta$ и составляем матрицу поворота Гивенса T_2^T:

$$T_2^T = \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad B_1 = B_0T_2^T = \begin{bmatrix} d_1^* & 0 & 0 & 0 \\ l_1 & d_2 & u_2 & 0 \\ 0 & 0 & d_3 & u_3 \\ 0 & 0 & 0 & d_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Вторым шагом строится матрица Гивенса S_2. Матрица S_2 строится таким образом, чтобы обнулить элемент l_1 матрицы B_1.

$$\underbrace{\begin{bmatrix} c & s \\ -s & c \end{bmatrix}}_{S_2} \begin{bmatrix} d_1^* \\ l_2 \end{bmatrix} \to \begin{bmatrix} d_1^{**} \\ 0 \end{bmatrix}.$$

Составляем матрицу поворота Гивенса S_2:

$$S_2 = \begin{bmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_2 = S_2 B_1 T_2^T = \begin{bmatrix} d_1^{**} & u_1^* & t_1 & 0 \\ 0 & d_2 & u_2 & 0 \\ 0 & 0 & d_3 & u_3 \\ 0 & 0 & 0 & d_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

На третьем шаге составляется матрица Гивенса T_3^T, которая обнуляет элемент u_2:

$$\begin{bmatrix} d_2 & u_2 \end{bmatrix} \underbrace{\begin{bmatrix} c & -s \\ s & c \end{bmatrix}}_{T_3^T} \to \begin{bmatrix} d_2^* & 0 \end{bmatrix}.$$

Можно показать, что помимо обнуления u_2, обнуляется также элемент t_1. Составляем матрицу поворота Гивенса T_3^T:

$$T_3^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_3 = S_2 B_2 T_2^T T_3^T = \begin{bmatrix} d_1^{**} & u_1^* & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & l_2 & d_3 & u_3 \\ 0 & 0 & 0 & d_4 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Аналогично строятся последующие матрицы S_i, T_i.

5.2 Исходный код

5.2.1 Преобразование Хаусхолдера

```
% Householder transformation
function H = householder(x,k)
% Input: x(k:end) -- vector for reflection
% Output: H -- Householder matrix
   [n,~] = size(x); % get size of x
   u = zeros(n,1); % creation of a vector with respect to which the...
   u(k) = x(k) + sign(x(k)) * sqrt(sum(x(k:n) .^ 2)); % reflection is...
   u((k+1):n) = x((k+1):n); % performed
   H = eye(n) - 2 .* (u * u') / dot(u,u); % creation of a Hauseholder matrix
end
```

5.2.2 Бидиагонализация матрицы

```
% Bidiagonilization algorithm.
% Finds an upper bidiagonal matrix B so that A=U*B*V'.
```

```
function [U,B,V] = bidiag_reduction(A)
% Input: A is an original m-by-n matrix.
% Ouput: U,V are orthogonal matrices,
%        B is an upper bidiagonal matrix
% U,S,V are constructed in a such way that A = U * B * V'
   [m,n] = size(A);
   B = A;
   U = eye(m);
   V = eye(n);
   for k = 1:min(m,n)
     % setting to zero elements of a column
     H1 = householder(B(:,k),k); % creating Hausholder matrix
     B = H1 * B; % applying transformation
     U = U * H1'; % saving transformation
     if k < min(m,n)-1
       % setting to zero elements of a row
       H2 = householder(B(k,:)', k+1); % creating Hausholder matrix
       B = B * H2'; % applying transformation
       V = V * H2'; % saving transformation
     end
   end
end
```

5.2.3 Вращения Гивенса

```
% Effective computing of Given's rotation matrix
function [c,s,r] = rot(f,g)
% Input: [f;g] -- vector of two elements
% Ouput: c := cos(theta),
%        s := sin(theta),
%        [r;0] -- result vector, such that
% [c,-s;s,c]*[f;g] = [r;0].
    if f == 0 % trivial case
        c = 0; s = 1; r = g;
    elseif abs(f) > abs(g)
        t = g / f; t1 = sqrt(1 + t^2);
        c = 1 / t1; s = t * c; r = f * t1;
    else
        t = f / g; t1 = sqrt(1 + t^2);
        s = 1 / t1; c = t * s; r = g * t1;
    end
end
```

5.2.4 QR скачки

```
% QR-type sweeps for SVD
function [U,B,V,E] = qr_sweep(B)
% Input: B is a bidiagonal rectangular matrix
% Ouput: U,V are orthonormal matrices,
%        B` is a bidiagonal rectangular matrix,
%        E is the norm of B (new bidiagonal matrix)
% U,B`,V are constructed in a such way that B` = U * B * V'
    [m,n] = size(B);
    U = eye(m); % creating orthonormal matrices...
    V = eye(n); % of transformations
```

```
    M = min(m,n);
    for k = 1:M-1
        [c,s,~] = rot(B(k,k),B(k,k+1)); % calculate sin and cos
        Q = eye(n); % create matrix of...
        Q(k:k+1,k:k+1) = [c s;-s c]; % givence rotations
        B = B * Q'; % apply transformation
        V = V * Q'; % save transformation
        B((abs(B) < 1.e-13)) = 0; % remove infinitely small elements
        %spy(B); pause; % for debug
        [c,s,~] = rot(B(k,k),B(k+1,k)); % calculate sin and cos
        Q = eye(m); % create matrix of...
        Q(k:k+1,k:k+1) = [c s;-s c]; % givence rotations
        B = Q * B; % apply transformation
        U = U * Q'; % save transformation
        B((abs(B) < 1.e-13)) = 0; % remove infinitely small elements
        %spy(B); pause; % for debug
    end
    I = zeros(m,n); % creation of a temporary bidiagonal matrix...
    I(1:M,1:M) = diag(ones(1,M-1),1); % for computing the norm
    E = sum(sum(abs(I .* B))); % computing the norm (error)
end
```

5.2.5 “Стандартный” алгоритм

```
% "Standard" algorithm for SVD
function [U,S,V] = svd_standard(A)
% Input: A is an original rectangular m-by-n matrix with condition: m >= n.
% Ouput: U,V are orthonormal matrices,
%        S is a diagonal matrix
% U,S,V are constructed in a such way that A = U * S * V'
    [U1,B,V1] = bidiag_reduction(A); % first part is a bidiagonalization
    % A = U1 * B * V1'
    U2 = eye(size(U1)); % temporary variable
    V2 = eye(size(V1)); % temporary variable
    E = Inf; % norm of a bidiagonal matrix
    while E > 1e-8
        [U3,B,V3,E] = qr_sweep(B); % performing sweeps
        U2 = U2 * U3; % adding rotations
        V2 = V2 * V3; % adding rotations
    end
    U = U1 * U2; % adding rotations
    V = V1 * V2; % adding rotations
    S = B; % singular values (diagonal matrix)
end
```

6 Приложения сингулярного разложения

6.1 Вычисление псевдообратной матрицы

6.1.1 Теория

При вычислении обратной матрицы возникает 2 проблемы:

1. Вычисление обратной матрицы возможно только для квадратных матриц
2. Если определитель матрицы равен нулю, то при вычислении обратной матрицы происходит деление на ноль.

Эти проблемы можно обойти, используя сингулярное разложение и вычисляя псевдообратную матрицу.

Представим, что дана матрица $A_{m\times n},\ m \geq n$, причем $r = \mathrm{rank}\,(A),\ r = n$, тогда можно утверждать, что $\exists\ (A^TA)^{-1}$. Пусть имеется система уравнений:

$$Ax = b,$$

тогда, домножив слева на A^T, мы получим уравнение вида:

$$A^TAx = A^Tb.$$

Т.к. $\exists\ (A^TA)^{-1}$, то можно выразить x и тем самым определить псевдообратную матрицу:

$$x = (A^TA)^{-1}A^Tb \Rightarrow A^+ = (A^TA)^{-1}A^T. \tag{6.1.1}$$

В данном примере требовалось, чтобы $r = \mathrm{rank}\,(A),\ r = n$. Предположим теперь, что $r < n$. Вспомним, что "полное" сингулярное разложение имеет следующий вид:

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}. \tag{6.1.2}$$

Подставим (6.1.2) в (6.1.1) и произведем необходимые сокращения:

$$A^+ = (V\Sigma U^TU\Sigma^TV^T)^{-1}V\Sigma U^T = (V\Sigma^T\Sigma V^T)^{-1}V\Sigma^TU^T = V(\Sigma^T\Sigma)^{-1}\Sigma^TU^T = V\Sigma^+U^T.$$

$$A^+ = V_{n\times n}(\Sigma_{m\times n})^+U^T_{m\times m} = V_{n\times n}(\Sigma^+)_{n\times m}U^T_{m\times m}.$$

Основная проблема – вычислить Σ^+. Эта операция выполняется в два шага:

1. Матрица транспонируется
2. Все ненулевые элементы главной диагонали возводятся в степень -1.

Т.о. вычисление псевдообратной матрицы сводится к вычислению сингулярного разложения.

6.1.2 Исходный код

```
% Moore-Penrose pseudoinverse of a matrix using SVD decomposition
function X = p_inv(A)
% Input: A is an original matrix
% Ouput: X is a pseudoinverse of a matrix A,
% so A*X*A = A, X*A*X = X and A*X and X*A are Hermitian.
    [U,S,V] = svd(A);
    s = 1 ./ diag(S); % inverse of singular values
    n = size(s,1);
    S(1:n,1:n) = diag(s);
    X = V * S' * U';
end
```

6.2 Метод главных компонент

Метод Главных Компонент – один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации. Изобретен К.Пирсоном в 1901 г. Применяется во многих областях, таких как распознавание образов, компьютерное зрение, сжатие данных и т.п.

6.2.1 Теория

Существует две основные матричные нормы в численных методах линейной алгебры: норма Фробениуса – $||A||_F$ и 2-норма – $||A||_2$. Норма Фробениуса была подробно рассмотрена в (2.4), поэтому перейдем сразу к рассмотрению 2-нормы.

2-норма определяется следующим образом:

$$||A||_2 = \max_{||\mathbf{v}||=1} ||A\mathbf{v}|| \Rightarrow ||A||_2 = \sigma_1.$$

Представим матрицу размеров $n \times d$ как n точек $\{a^{(i)}\}_{i=1}^n$ в d-мерном пространстве. Норма Фробениуса матрицы A эквивалентна квадратному корню суммы квадратов расстояний от точек $\{a^{(i)}\}_{i=1}^n$ до начала координат. 2-норма матрицы A эквивалентна квадратному корню суммы квадратов расстояний точек $\{a^{(i)}\}_{i=1}^n$ до прямой, проходящей вдоль вектора $\mathbf{v_1}$.

Пусть

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}$$

сингулярное разложение матрицы A. Пусть для $k \in \{1, 2, ..., r\}$

$$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i^T}$$

является k-ой частичной суммой. В таком случае матрица A_k имеет ранг k, а также она является наилучшей аппроксимацией ранга k матрицы A, если ошибка измеряется с помощью 2-нормы или нормы Фробениуса.

Докажем лемму, утверждающую, что строки матрицы A_k являются проекциями строк матрицы A на подпространство V_k, образованное первыми k правыми сингулярными векторами $\{\mathbf{v_i}\}_{i=1}^k$ матрицы A.

Пусть $\mathbf{a}$ – произвольная вектор-строка. Т.к. векторы $\mathbf{v_i}$ ортонормированы, то проекция вектора $\mathbf{a}$ на подпространство V_k определяется следующим соотношением: $\sum_{i=1}^k (\mathbf{a} \cdot \mathbf{v_i}) \mathbf{v_i^T}$. Таким образом, матрица, строки которой являются проекциями строк матрицы A на подпространство V_k, задается выражением $\sum_{i=1}^k A \mathbf{v_i} \mathbf{v_i^T}$.

$$\sum_{i=1}^{k} A \mathbf{v_i} \mathbf{v_i^T} = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i^T} = A_k.$$

Т.к. строки матрицы A_k – проекции строк матрицы A на k-размерное подпространство V_k, то $\text{rank}\,(A_k) \leq k$.

Докажем теорему, утверждающую, что для любой матрицы B ранга не более, чем k

$$||A - A_k||_F \leq ||A - B||_F.$$

Пусть B минимизирует $||A - B||_F^2$ среди всех матриц ранга не больше, чем k. Пусть V – подпространство, порожденное векторами-строками матрицы B. Размерность подпространства V не превосходит k. Т.к. B минимизирует $||A - B||_F^2$, то отсюда следует, что

строки матрицы B – проекции строк матрицы A на подпространство V. Т.к. строки матрицы B являются проекциями соответствующих строк матрицы A, то $||A-B||_F^2$ является суммой квадратов расстояний строк матрицы A на подпространство V. Т.к. A_k минимизирует сумму квадратов расстояний строк матрицы A на подпространство размерности не больше k, то

$$||A-A_k||_F^2 \leq ||A-B||_F^2 \Rightarrow ||A-A_k||_F \leq ||A-B||_F.$$

Докажем лемму, утверждающую, что

$$||A-A_k||_2^2 = \sigma_{k+1}^2.$$

Пусть $A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}$ – сингулярное разложение матрицы A. В таком случае $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i^T}$ и

$$A - A_k = \sum_{i=k+1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T}$$

Вспомним, что

$$||A|| = \sup_{||x||=1} ||Ax||.$$

В таком случае

$$||A-A_k||_2 = \sup_{||\mathbf{x}||_2=1} ||(A-A_k)\mathbf{x}||_2 = \left|\left| \sum_{i=k+1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \mathbf{x} \right|\right|_2.$$

Представим вектор $\mathbf{x}$ в виде линейной комбинации векторов $\mathbf{v_1}, ..., \mathbf{v_r}$:

$$\mathbf{x} = \sum_{i=1}^{r} \alpha_i \mathbf{v_i}.$$

В таком случае

$$||(A-A_k)\mathbf{x}||_2 = \left|\left| \sum_{i=k+1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \sum_{i=1}^{r} \alpha_i \mathbf{v_i} \right|\right|_2 = \left|\left| \sum_{i=k+1}^{r} \alpha_i \sigma_i \mathbf{u_i} \mathbf{v_i^T} \mathbf{v_i} \right|\right|_2 = \left|\left| \sum_{i=k+1}^{r} \alpha_i \sigma_i \mathbf{u_i} \right|\right|_2.$$

Последние преобразования возможны благодаря тому, что

$$\mathbf{v_i^T} \mathbf{v_j} = \begin{cases} 1, \ i = j \\ 0, \ i \neq j \end{cases}$$

Таким образом получаем, что

$$||(A-A_k)\mathbf{x}||_2 = \left|\left| \sum_{i=k+1}^{r} \alpha_i \sigma_i \mathbf{u_i} \right|\right|_2 = \sqrt{\sum_{i=k+1}^{r} \alpha_i^2 \sigma_i^2},$$

т.к. получившееся выражение есть ни что иное, как разложение вектора по базису $\mathbf{u_i}$. Учитывая тот факт, что последовательность $\{\sigma_i\}_{i=1}^{r}$ не возрастает,

$$\left.\begin{array}{l} \mathbf{x} = (\underbrace{0, ..., 0}_{k}, 1, 0, ..., 0)^T, \ ||\mathbf{x}||_2 = 1 \\ \mathbf{x} = \underset{||\mathbf{x}||_2=1}{\operatorname{argmax}} ||(A-A_k)\mathbf{x}||_2 \end{array}\right\} \Rightarrow ||A-A_k||_2^2 = \sigma_{k+1}.$$

Докажем теорему, утверждающую, что для любой матрицы B такой, что $\text{rank}(B) \leq k$ выполняется неравенство:

$$||A - A_k||_2 \leq ||A - B||_2.$$

Если $\text{rank}(A) \leq k$, то теорема, очевидно, выполняется, т.к. $||A - A_k||_2 = 0$. Таким образом, предположим, что $\text{rank}(A) > k$. Используя вышедоказанную лемму, получим, что $||A - A_k||_2 = \sigma_{k+1}^2$. Теперь предположим, что $\exists B,\ \text{rank}(B) \leq k$, для которой

$$||A - B||_2 \leq ||A - A_k||_2 \ \Rightarrow\ ||A - B||_2 \leq \sigma_{k+1}^2.$$

Будем рассматривать матрицу B как линейное преобразование. В таком случае размерность ядра преобразования B:

$$\ker(B) = \{\mathbf{x} \mid B\mathbf{x} = 0\},$$

будет не меньше, чем $(d - k)$

$$\dim \ker(B) \geq (d - k).$$

Пусть векторы $\mathbf{v_1}, ..., \mathbf{v_{k+1}}$ – первые $k + 1$ сингулярные векторы матрицы A. Обозначим $d - k$ независимых векторов ядра преобразования B как $\mathbf{u_1}, ..., \mathbf{u_{d-k}}$. В таком случае $\mathbf{u_1}, ..., \mathbf{u_{d-k}}, \mathbf{v_1}, ..., \mathbf{v_{k+1}}$ – набор из $d + 1$ векторов в d-мерном пространстве, следовательно пространства $V_k = \text{span}\{\mathbf{v_1}, ..., \mathbf{v_{k+1}}\}$ и $\ker(B) = \text{span}\{\mathbf{u_1}, ..., \mathbf{u_{d-k}}\}$ обязательно пересекаются:

$$\dim\Big(\ker(B) \cap \text{span}\{\mathbf{v_1}, ..., \mathbf{v_{k+1}}\}\Big) \neq 0 \ \Rightarrow\ \exists \mathbf{z} \neq 0 :\ \mathbf{z} \in \Big(\ker(B) \cap \text{span}\{\mathbf{v_1}, ..., \mathbf{v_{k+1}}\}\Big).$$

Выберем $\mathbf{z}$ таким образом, чтобы $||\mathbf{z}||_2 = 1$. Вспомнив определение нормы матрицы, получим:

$$||A|| = \sup_{||\mathbf{x}||=1} ||A\mathbf{x}|| \Rightarrow ||A - B||_2^2 \geq ||(A - B)\mathbf{z}||_2^2.$$

Учитывая тот факт, что $\mathbf{z} \in \ker(B) :\ B\mathbf{z} = 0$, получаем, что

$$||A - B||_2^2 \geq ||(A - B)\mathbf{z}||_2^2 = ||A\mathbf{z}||_2^2.$$

Учитывая тот факт, что $\mathbf{z} \in V_k = \text{span}\{\mathbf{v_1}, ..., \mathbf{v_{k+1}}\}$, получаем, что

$$\begin{aligned} ||A\mathbf{z}||_2^2 &= \left|\left|\sum_{i=1}^{n} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \mathbf{z}\right|\right|_2^2 = \left|\left|\sum_{i=1}^{n} \sigma_i \mathbf{u_i} \mathbf{v_i^T} \left(\sum_{j=1}^{k+1} \alpha_j \mathbf{v_j}\right)\right|\right|_2^2 = \left|\left|\sum_{i=1}^{k+1} \sigma_i \alpha_i \mathbf{u_i}\right|\right|_2^2 \\ &= \sum_{i=1}^{k+1} \sigma_i^2 \alpha_i^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} \alpha_i^2 = \sigma_{k+1}^2. \end{aligned}$$

Таким образом получили выражение:

$$||A - B||_2^2 \geq \sigma_{k+1}^2,$$

что противоречит изначальному предположению о том, что $||A - B||_2 < \sigma_{k+1}$, следовательно

$$\forall\ B,\ \text{rank}(B) \leq k :\ ||A - A_k||_2 \leq ||A - B||_2.$$

6.2.2 Исходный код

```
% Low-rank approximation using SVD decomposition
function Ak = low_rank(A,k)
% Input: A is an original matrix, k - required rank, r = rank(A) >= k
% Ouput: Ak is a result matrix, size(Ak) = size(A), rank(Ak) = k
   assert(rank(A) > k, 'Rank of the matrix A is smaller than the required ...
       rank k!');
   [~,~,V] = svd(A);
   Ak = A * V(:,1:k) * V(:,1:k)';
end

% Low-rank approximation using SVD decomposition
function Ak = low_rank(A,k)
% Input: A is an original matrix, k - required rank, r = rank(A) >= k
% Ouput: Ak is a result matrix, size(Ak) = size(A), rank(Ak) = k
   assert(rank(A) > k, 'Rank of the matrix A is smaller than the required ...
       rank k!');
   [U,S,V] = svd(A);
  Ak = U(:,1:k) * S(1:k,1:k) * V(:,1:k)';
end
```

6.2.3 Пример работы

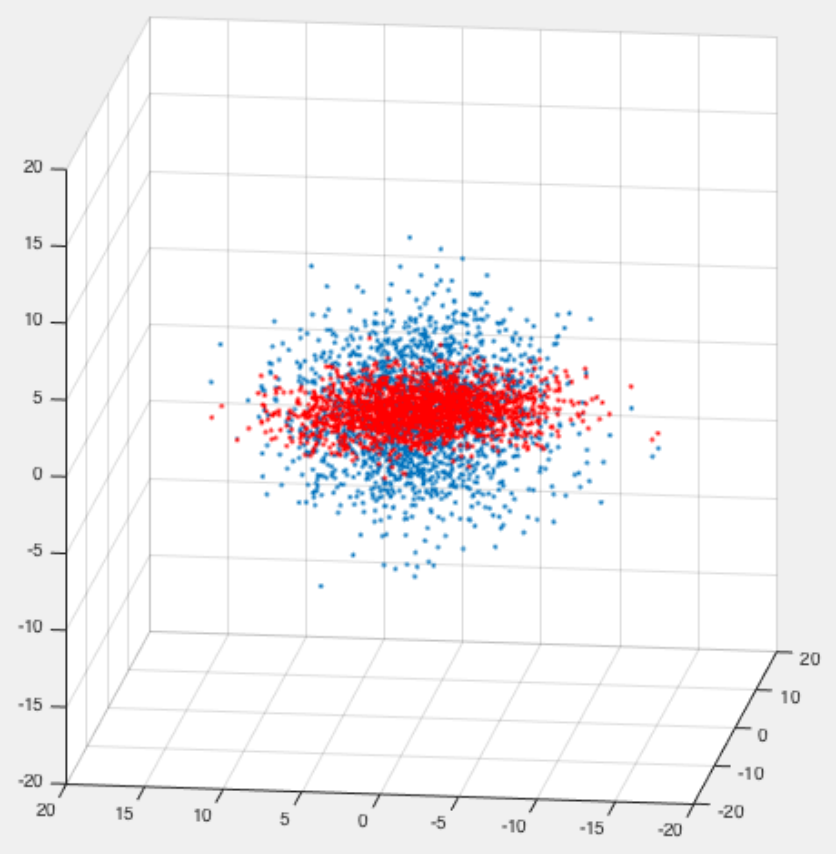


Рис. 11: Пример работы

На (Рис. 11) можно видеть, как точки, случайным образом разбросанные в трехмерном пространстве (синие) спроецировались на двумерное подпространство (красное).

6.3 Понижение размерности

6.3.1 Постановка задачи

Ярким примером задачи понижения размерности является задача из области машинного обучения.

Будем рассматривать определенный класс объектов, каждый из которых можно представить с помощью конечного набора количественных и качественных характеристик, которые образуют вектор $(x_1, ..., x_N)^T$. Рассмотрим набор, состоящий только из n количественных характеристик ($n \leq N$). Т.о. каждому объекту ставится в соответствие вектор действительных чисел $(x_{i_1}, ..., x_{i_n})^T$, $\{i_1, ..., i_n\} \subset \{1, ..., N\}$. Без ограничения общности будем предполагать, что $(i_1, ..., i_n) \equiv (1, ..., n)$.

Предположим, что обучающая выборка содержит M различных объектов, каждый из которых описывается вектором n действительных чисел. В таком случае каждый объект можно представить радиус-вектором n-мерного пространства, а всю обучающую выборку – матрицей $A_{n \times M}$.

Часто бывает так, что n достаточно велико и работать с матрицей обучающей выборки либо неудобно, либо тяжело. В таком случае есть смысл попробовать понизить размерность матрицы, потеряв при этом наименьшее количество информации: рассматривать вместо вектора $(x_1, ..., x_n)^T$, вектор $(y_1, ..., y_k)^T$, причем $k < n$.

Однако проблема стоит не только в том, чтобы понизить размерность матрицы обучающей выборки, но также разработать алгоритм перевода любого вектора $(x_1, ..., x_n)^T$ в соответствующий ему вектор $(y_1, ..., y_k)^T$.

6.3.2 Теория

В данном разделе будет приведен алгоритм понижения размерности набора свойств класса объектов:

$$(x_1, ..., x_n)^T \to (y_1, ..., y_k)^T.$$

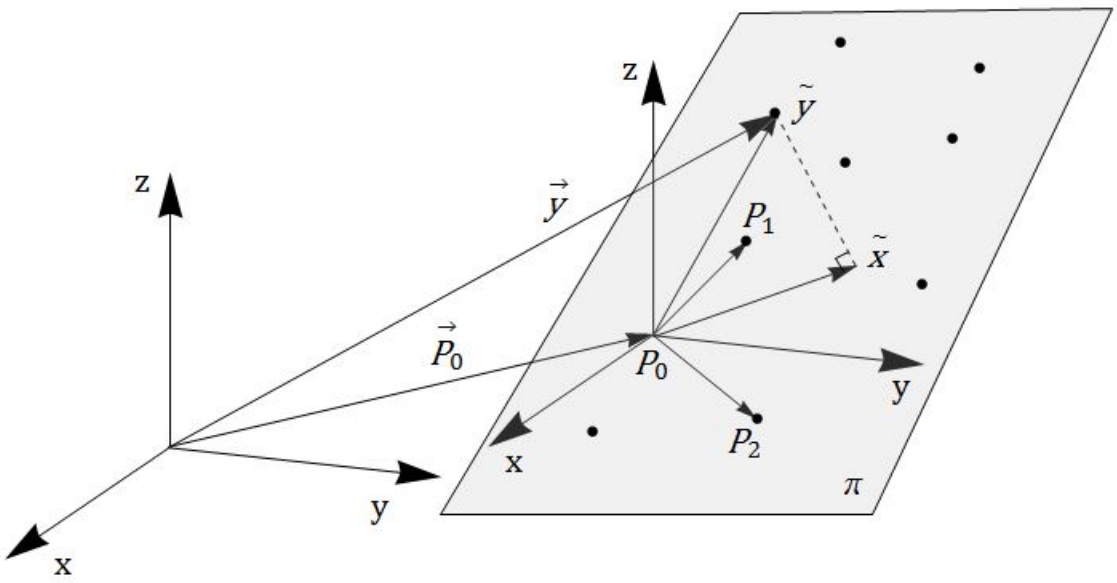


Рис. 12: Частный случай

– Первым делом необходимо понизить ранг матрицы $A_{n \times M}$ до $k < \min(n, M)$, воспользовавшись алгоритмом, описанным в (6.2).

После того, как матрица $A_{n\times M}$ понижена до ранга k, каждый столбец можно представить точкой на k-мерной гиперплоскости в n-мерном пространстве. Визуально представить эффект понижения порядка можно, посмотрев на (Рис. 11).

Разложим матрицу $A_{n\times M}$ с помощью QR-разложения. Полученные матрицы Q и R являются матрицами размерностей $n \times n$ и $n \times M$ соответственно. Однако, учитывая то, что ранг матрицы $A_{n\times M}$ равен $k < n$, верхнетреугольная матрица R имеет $n - k$ нулевых строк, т.е. $n - k$ характеристик обнулились. Необходимо запомнить преобразование Q, обнуляющее $n - k$ характеристик. Оно выглядит следующим образом:

$$Y := R = Q^T A.$$

– На втором шаге необходимо выбрать "опорную" точку P_0, представляющую из себя рандомный вектор-столбец матрицы A, пониженного ранга.

– Третим пунктом алгоритма является задание базиса k-мерной гиперплоскости П. Для этого необходимо случайным образом выбрать k точек $\{P_1, ..., P_k\}$ таким образом, чтобы

$$\forall\ i \neq j:\ P_i \neq P_j,\quad B_{n\times k} = \left[P_1 - P_0 \;\middle|\; ... \;\middle|\; P_k - P_0\right],\quad \operatorname{rank}(B) = k.$$

Фактически данный базис задает смещенную плоскость, проходящую через начало координат в точке P_0. Т.о. произошла неявная смена системы координат: $Oxyz \to P_0xyz$.

– Далее рассмотрим произвольную точку y, которую необходимо спроецировать на гиперплоскость П.

Изначально представим ее в новой, смещенной системе координат:

$$\tilde{y} = y - P_0.$$

Обозначим базисные векторы

$$P_i - P_0 = z_i = (z_i^1, ..., z_i^n)^T.$$

Т.о. матрица $B_{n\times k}$ примет следующий вид:

$$B = \begin{bmatrix} z_1 & \dots & z_k \end{bmatrix} = \begin{bmatrix} z_1^1 & \dots & z_k^1 \\ \dots & \dots & \dots \\ z_1^n & \dots & z_k^n \end{bmatrix}$$

Если бы вектор $\tilde{y}$ лежал в плоскости П, то $B\tilde{x} = \tilde{y}$. Однако в конкретном случае $\tilde{y}$ не обязан лежать в плоскости П, поэтому сперва необходимо спроецировать его на эту плоскость с помощью псевдообратной матрицы B^+:

$$\tilde{x} = B^+\tilde{y} = B^+(y - P_0).$$

Полученный вектор $\tilde{x}$ задан в базисе плоскости $\{z_1, ..., z_k\}$, поэтому необходимо перейти сперва к системе координат P_0xyz, а затем и в исходную $Oxyz$. Поэтому искомый вектор x примет окончательный вид:

$$x = B\tilde{x} + P_0 = BB^+\tilde{y} + P_0 = BB^+(y - P_0) + P_0 = BB^+y - BB^+P_0 + P_0 = BB^+y.$$

$$x = BB^+y.$$

Т.о. полностью проанализирована задача перехода от y к $\tilde{x}$ во всех системах координат.

6.4 Сжатие изображения

Сингулярное разложение оказывается очень удобным механизмом сжатия изображений с минимальной потерей информации. В данном разделе будет подробно разобран и запрограммирован простейший алгоритм сжатия серых изображений с использованием сингулярного разложения.

6.4.1 Принцип работы

Мы можем представить изображение как матрицу следующим образом. Предположим, что изображение имеет $n \times m$ пикселей. Оттенки серого кодируются одним числом, поэтому все изображение представляется обычной матрицей размерности $n \times m$.

У цветных изображений на каждый пиксель приходится три числовых значения, поэтому все изображение можно представить тремя матрицами размерностей $n \times m$, либо одной матрицей размерности $n \times 3m$, получающейся в результате конкатенации трех матриц, каждая из которых отвечает за свой цвет: $A = [A_{\text{red}} \mid A_{\text{green}} \mid A_{\text{blue}}]$.

Рассмотрим сперва случай квадратного изображения в градации серого, которое представляется матрицей I_0 размерности $n \times n$. В подавляющем большинстве случаев $\text{rank}(I_0) = n$.

Матрицу I_0 можно разложить с помощью сингулярного разложения:

$$I_0 = \sum_{i=1}^{n} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

Аппроксимируем матрицу I_0 матрицей

$$I^{(k)} = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i^T}.$$

Матрица $I^{(k)}$ имеет ранг k, т.е. линейно независимыми являются лишь k строк и столбцов.

Проанализируем полученный результат: мы аппроксимировали матрицу I_0 размерности $n \times n$ и ранга n матрицей $I^{(k)}$ размерности $n \times n$ и рангом k, причем погрешность аппроксимации составляет

$$\delta_I = \frac{||I_0 - I^{(k)}||_2}{||I_0||_2} \cdot 100\% \qquad \Delta_I = \frac{||I^{(k)}||_F}{||I_0||_F}.$$

Чем больше значение k, тем лучше $I^{(k)}$ аппроксимирует I_0. Из приведенных двух оценок аппроксимации принято использовать Δ_I, т.к. она дает более точную и логичную оценку. На (Рис. 17) показаны изменения погрешностей в зависимости от выбранного ранга k для изображения (Рис. 13).

На данный момент изображение потеряло часть информации (возможно, пропала четкость и появилось размытие), однако само изображение как кодировалось матрицей $n \times n$, так и продолжает кодироваться матрицей такого же размера. Необходимо каким-то образом уменьшить размерность матрицы. Т.к. матрица $I^{(k)}$ имеет ранг k, то это значит, что она содержит k линейно независимых столбцов. Следовательно, ортогонализовав матрицу $I^{(k)}$, мы получим матрицу с $n - k$ нулевыми строками или столбцами (в зависимости от того, как ортогонализуем).

Разложим матрицу $I^{(k)}$ с помощью QR-разложения. Полученные матрицы Q и R являются матрицами размерностей $n \times n$. Однако, учитывая то, что ранг матрицы $I^{(k)}$ равен

$k < n$, верхнетреугольная матрица R имеет $n - k$ нулевых строк, которые можно исключить из матрицы R. Исключив из R $n-k$ последних строк, необходимо также исключить из матрицы Q $n-k$ последних столбцов для согласованности матриц Q, R. Таким образом,

$$\left.\begin{array}{r} I^{(k)}_{n\times n} = Q_{n\times n}R_{n\times n} \\ Q_{n\times n} \to \widetilde{Q}_{n\times k} \\ R_{n\times n} \to \widetilde{R}_{k\times n} \end{array}\right\} \Rightarrow \; I^{(k)} = \widetilde{Q}\widetilde{R}. \tag{6.4.1}$$

Таким образом, мы закодировали матрицу $n \times n$ двумя матрицами $n \times k$.

Абсолютно аналогичными рассуждениями прямоугольная матрица $n \times m$ сводится к двум матрицам с размерностями $n \times k$ и $k \times m$.

6.4.2 Сжатие

Рассмотрим случай квадратной матрицы $I_{n\times n}$. Применив алгоритм, описанный в (6.4.1), мы свели матрицу $I_{n\times n}$ к двум матрицам $\widetilde{Q}_{n\times k},\ \widetilde{R}_{k\times n}$. Т.о. n^2 элементов теперь кодируется $2nk$ элементами. Проанализируем, в каком случае происходит сжатие объема требуемой памяти.

$$2nk < n^2 \;\Rightarrow\; 2k < n \;\Rightarrow\; k < \frac{n}{2}.$$

Т.е. сжатие изображения происходит, если выбираемый параметр $k < \dfrac{n}{2}$.

В таком случае целесообразно ввести коэффициент сжатия следующим образом:

$$\alpha = \frac{n}{2k}.$$

Рассмотрим теперь случай прямоугольной матрицы $I_{n\times m}$. Применив алгоритм, описанный в (6.4.1), мы свели матрицу $I_{n\times m}$ к двум матрицам $\widetilde{Q}_{n\times k},\ \widetilde{R}_{k\times m}$. Т.о. nm элементов теперь кодируется $(n+m)k$ элементами. Проанализируем, в каком случае происходит сжатие объема требуемой памяти.

$$(n+m)k < nm \;\Rightarrow\; k < \frac{nm}{n+m}.$$

Т.е. сжатие изображения происходит, если выбираемый параметр $k < \dfrac{nm}{n+m}$.

Введем коэффициент сжатия аналогично тому, как он был введен в случае с квадратной матрицей:

$$\alpha = \frac{nm}{(n+m)k}.$$

6.4.3 Исходный код

```
function image_compression(img_path)
I = imread(img_path); % loads image in integer format as a 3d-array
I2 = im2double(I); % transform to real values
I3 = I2(:,:,1); % for gray-scale images, we consider only the first ...
    matrix in the array
figure; imshow(I3); % show original image
Ic = svd_compress(I3,5); % compress image
figure; imshow(Ic); % show compressed image
Id = svd_decompress(Ic); % decompress image
figure; imshow(histeq(I3 - Id)); % histogram equalization of the ...
    difference matrix
```

```
10
11 function I_compressed = svd_compress(I,alpha) % compression function
12 % alpha -- compression parameter
13 k = floor(min(size(I)) / (2*alpha)); % rank needed to compress alpha times
14 I1 = low_rank(I,k); % low-rank approximation
15 figure; imshow(I1); % show approximation image
16 [Q,R] = qr(I1); % qr decomposition
17 C1 = Q(:,1:k); C2 = R(1:k,:); % removing last n-k columns in Q and last ...
       n-k rows in R
18 I_compressed = [C1';C2]; % concatenation (works only for squared matrices)
19
20 function I_decompressed = svd_decompress(I)% decompression function
21 [K,~] = size(I); k = K/2; % getting rank k
22 Q = I(1:k,:)'; R = I(k+1:end,:); % de-concatenation (works only for ...
       squared matrices)
23 I_decompressed = Q * R; % getting the original matrix
```

6.4.4 Пример работы

- Применим вышеприведенный код к изображению (Рис. 13).
- После аппроксимации матрицей меньшего ранга ($\alpha = 3$) получим изображение (Рис. 14).
- Сжав изображение, получим новое изображение (Рис. 15).
- На (Рис. 16) можно видеть разность оригинального изображения и его аппроксимации, к которой применен алгоритм “Histogram equalization”.

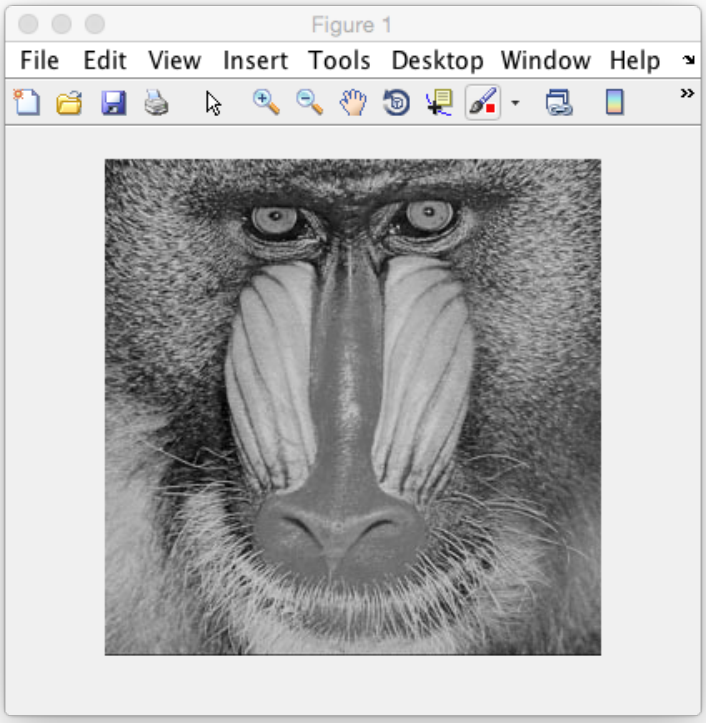


Рис. 13: Исходное изображение

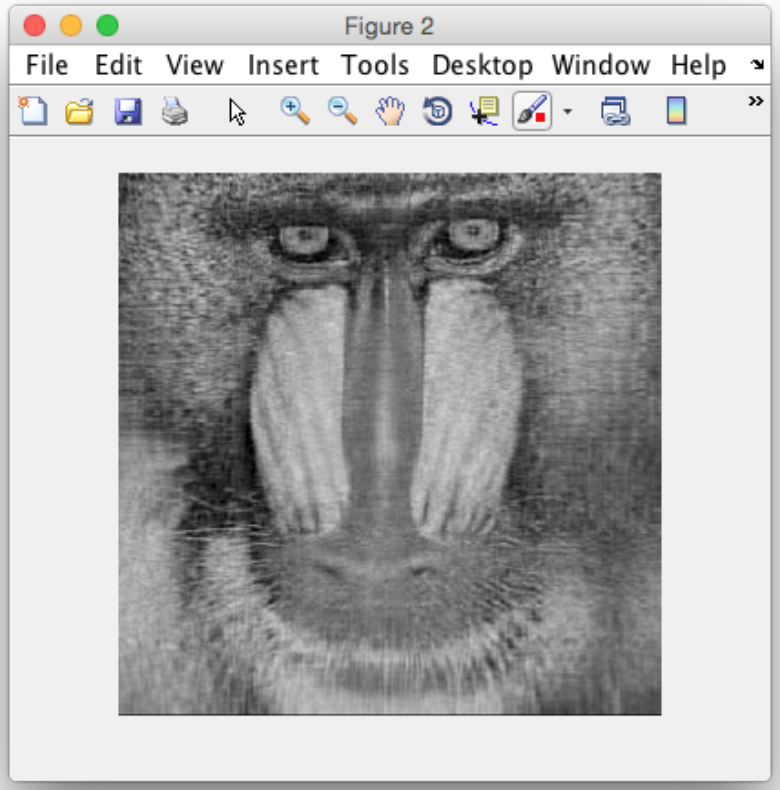


Рис. 14: Аппроксимация изображения

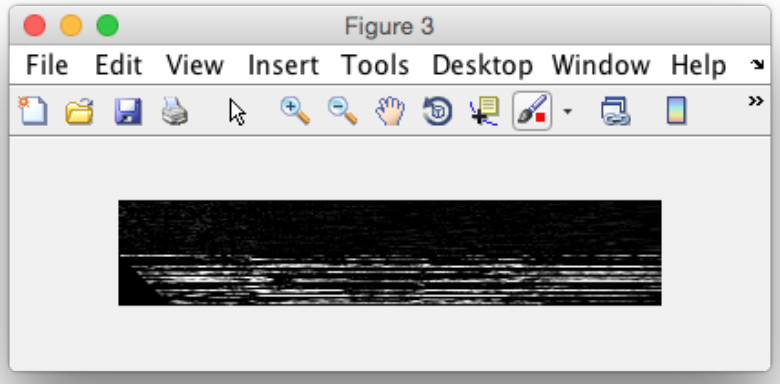


Рис. 15: Сжатое изображение (Рис. 14)

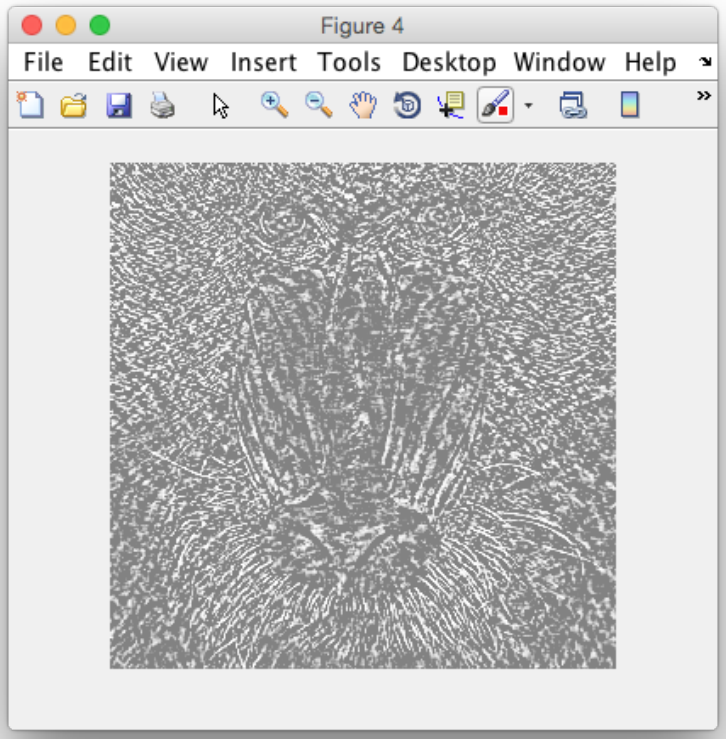


Рис. 16: Разность изображений

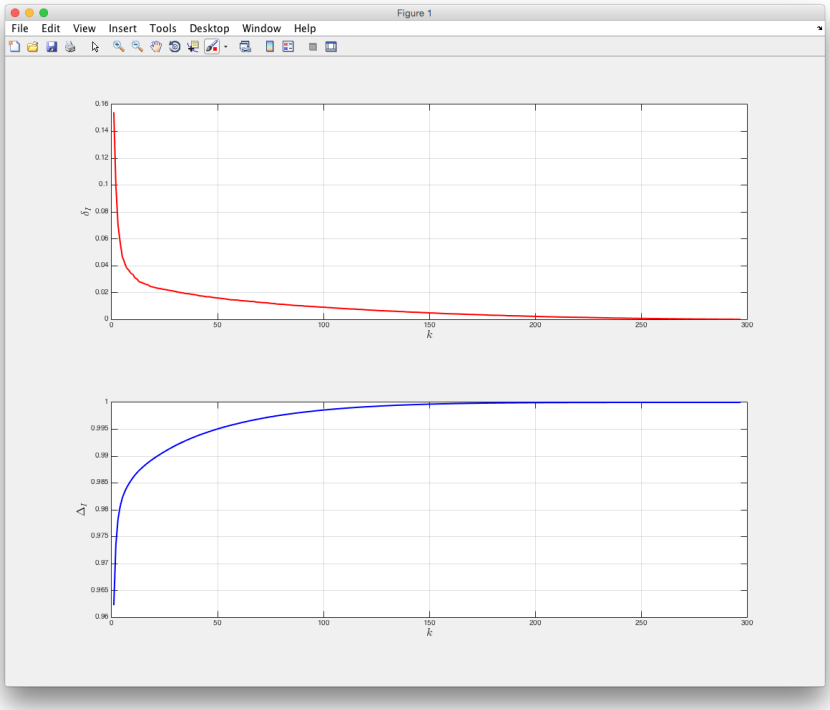


Рис. 17: Поведение ошибок изображений

7 Дополнение

7.1 Определения

Эрмитово-сопряжённая мтрица или сопряжённо-транспонрованная мтрица — это матрица A^H с комплексными элементами, полученная из исходной матрицы A транспонированием и заменой каждого элемента комплексно-сопряжённым ему. Эрмитово-сопряжённые матрицы во многом играют ту же роль при изучении комплексных векторных пространств, что и транспонированные матрицы в случае вещественных пространств.

Унитрная мтрица – квадратная матрица с комплексными элементами, результат умножения которой на эрмитово-сопряжённую равен единичной матрице: $U^\dagger U = UU^\dagger = I$. Другими словами, матрица унитарна тогда и только тогда, когда существует обратная к ней матрица, удовлетворяющая условию $U^{-1} = U^\dagger$. Унитарная матрица, элементы которой вещественны, является ортогональной.

Норма Фробениуса – субмультипликативная норма ($||AB||_F \leq ||A||_F||B||_F$), часто используемая в численных методах линейной алгебры из-за своих свойств: во-первых, она часто проще в вычислении, чем индуцированные нормы, а во-вторых, она инвариантна относительно поворотов $\forall R \in SO(n) : ||A||_F = ||AR||_F = ||RA||_F$.

7.2 Извлечение квадратного корня из матрицы

Извлечение квадратного корня из матрицы возможно для любой матрицы, имеющей полный набор собственных векторов и собственных значений, и у которой все собственные значения положительны.

Корень квадратный из матрицы – матрица, имеющая такой же набор собственных векторов, что и исходная матрица, а собственные значения которой являются квадратными корнями из собственных значений исходной матрицы.

Таким образом, чтобы посчитать $\sqrt{M}$, необходимо с помощью спектрального разложения представить матрицу M в виде $M = V\Lambda V^{-1}$ и тогда:

$$\sqrt{M} = V\sqrt{\Lambda}V^{-1}.$$

Список литературы

[1] J. Hopcroft, R. Kannan – Computer Science Theory for the Information Age, 2012.

[2] G. W. Stewart – On the Early History of the Singular Value Decomposition.

[3] J. Demmel, K. Veselic – Jacobi's method is more accurate than QR.

[4] J. Krishnamoorthy, K. Kocagoez – Singular Values using Cholesky Decomposition.

[5] N. J. Higham – Analysis of the Cholesky Decomposition of a Semi-Definite Matrix.

[6] J. Demmel и W. Kahan – Accurate Singular Values of Bidiagonal Matrices.

[7] M. Plesinger, Z. Strakos – Some remarks on bidiagonalization and its implementation.

[8] S. Kahu, R. Rahate – Image Compression using Singular Value Decomposition, International Journal of Advancements in Research & Technology, Volume 2, Issue 8, August-2013 244 ISSN 2278-7763.

[9] G. Golub и C. Reinsch – Singular Value Decomposition and Least Squares Solutions.