Московский государственный университет
Факультет вычислительной математики
и кибернетики

Методы оптимизации для низкоранговых матриц

Автор: Фролов Антон

Москва
2021

Оглавление

Глава 1

Введение

1.1 О задаче

Оптимизация — это задача нахождения экстремума (минимума или максимума) целевой функции в некоторой области конечномерного пространства, ограниченной набором линейных и/или нелинейных равенств и/или неравенств.

В нашем случае, мы имеем дело с задачей оптимизации с ограничением на ранг или задачей RCOP (Rank Constrained Optimization Problem). Она заключается в оптимизации выпуклой целевой функции от прямоугольной матрицы заданного размера с учетом ограничения на её ранг.

Эта задача привлекла к себе большое внимание в связи с широким применением в обработке сигналов, редукции моделей и идентификации систем. Несмотря на то что некоторые из задач такого рода могут быть решены аналитически, в большинстве случаев они являются NP трудными.

Существующие подходы к решению RCOP в основном сосредоточены на методах на основе попеременного проецирования и комбинированных алгоритмах линеаризации и факторизации с применением факторного анализа. Однако эти итерационные подходы зависят от первоначального предположения, и их быстрая сходимость не может быть гарантирована. Кроме того, был предложен метод, подобный методу Ньютона, применяемый в задаче стабилизации с обратной связью. Метод оптимизации с помощью римановых многообразий был применен для решения матричных уравнений Ляпунова большой размерности путем нахождения приближения низкого ранга.

1.2 Формулировка задачи

Задача оптимизации (миннимизации) с ограничением на ранг может быть сформулирована так

$$\begin{aligned} &\min_X f(X) \\ &s.t.\ X \in \mathbb{R}^{m\times n},\ rank(X) \le r \end{aligned} \tag{1.1}$$

где $f(X)$ выпуклая функция и $\mathbb{R}^{m\times n}$ – множество вещественных матриц размера m на n. Не ограничивая общности, будем считать, что $m \le n$.

Так как один из приведенных ниже методов для решения задачи RCOP требует от матрицы свойства положительной полуопределенности, необходимо перевести ограничения на ранг прямоугольной матрицы в ограничения на ранг положительно полуопределенной матрицы.

Лемма 1. *Пусть дана матрица $X \in \mathbb{R}^{m\times n}$. Тогда неравенство $rank(X) \le r$ выполнено тогда и только тогда, когда существуют матрицы $Y = Y^T \in R^{m\times m}$ и $Z = Z^T \in R^{n\times n}$ такие, что*

$$rank(Y) + rank(Z) \le 2r, \begin{bmatrix} Y & X \\ X^T & Z \end{bmatrix} \succeq 0 \tag{1.2}$$

Утверждение 2. *Равенство $Z = X^T X$ эквивалентно неравенству $rank\left(\begin{bmatrix} I_m & X \\ X^T & Z \end{bmatrix}\right) \le m$, где $Z \in \mathbb{S}^n, X \in \mathbb{R}^{m\times n}$ и $I_m \in \mathbb{R}^{m\times m}$ единичная матрица.*

Доказательство: С учетом того, что ранг симметричной блочной матрцы равен рангу блока, стоящего на диагонали, плюс ранг его дополнения Шура (Guttman rank additivity formula), имеем следующее отношение, $rank\left(\begin{bmatrix} Y & X \\ X^T & Z \end{bmatrix}\right) \le m \Leftrightarrow rank(I_m) + rank(Z - X^T X) \le m \Leftrightarrow m + rank(Z - X^T X) \le m \Leftrightarrow rank(Z - X^T X) = 0 \Leftrightarrow Z = X^T X$.

Далее приводится расширенная лемма о полупопределенном вложении.

Лемма 3. *Пусть дана матрица $X \in \mathbb{R}^{m\times n}$. Тогда неравенство $rank(X) \le r$ выполнено тогда и только тогда, когда существует матрица $Z \in \mathbb{S}^n$ такая, что*

$$rank(Z) \le r\ \&\ rank\left(\begin{bmatrix} I_m & X \\ X^T & Z \end{bmatrix}\right) \le m \tag{1.3}$$

Доаказательство: Из утверждения (2), $rank\left(\begin{bmatrix} I_m & X \\ X^T & Z \end{bmatrix}\right) \le m \Leftrightarrow Z = X^T X$. Известно, что $rank(Z) = rank(X)$, когда $Z = X^T X$. Таким образом, доказательство завершено.

Следовательно, основываясь на Лемме 3 задача 1.1 с ограничением на ранг прямоугольной матрицы может быть эквивалентным образом переведена в следующую формулировку с ограничениями на ранги положительно полуопределенных матриц

$$\begin{aligned} &\min_{X,Z} f(X) \\ &X \in \mathbb{R}^{m\times n}, rank(Z) \leq r \\ &rank\left(\begin{bmatrix} I_m & X \\ X^T & Z \end{bmatrix}\right) \leq m \end{aligned} \tag{1.4}$$

где $Z \in \mathbb{S}^n$ - дополнительно введенная матрица.

1.3 Сингулярное разложение

Сингулярным разложением или SVD (Singular Vector Decomposition) матрицы M порядка $m \times n$ называется разложение следующего вида

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

где Σ – матрица размера $m \times n$ с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми), а матрицы U (порядка m) и V (порядка n) – это две унитарные матрицы, состоящие из левых и правых сингулярных векторов соответственно (а V^* — это сопряжённо-транспонированная матрица к V).

В некоторых практических задачах требуется приближать заданную матрицу M некоторой другой матрицей M_k с заранее заданным рангом k. Известна следующая теорема, которую иногда называют теоремой Эккарта — Янга.

Если потребовать, чтобы такое приближение было наилучшим в том смысле, что норма Фробениуса разности матриц M и M_k минимальна, при ограничении $rank(M_k) = k$, то оказывается, что наилучшая такая матрица M_k получается из сингулярного разложения матрицы M по формуле:

$$M_k = U\Sigma_k V^*$$

где Σ_K – матрица Σ, в которой заменили нулями все диагональные элементы, кроме k наибольших элементов. Если элементы матрицы Σ упорядочены по невозрастанию, то выражение для матрицы M_k можно переписать в такой форме:

$$M_k = U_k\Sigma_k V_k^*$$

где матрицы U_k, Σ_k и V_k получаются из соответствующих матриц в сингулярном разложении матрицы M обрезанием до ровно k первых столбцов.

Таким образом видно, что приближая матрицу M матрицей меньшего ранга, мы выполняем своего рода сжатие информации, содержащейся в M: матрица M размера $m \times n$ заменяется меньшими матрицами размеров $m \times k$ и $k \times n$ и диагональной матрицей с k элементами. При этом сжатие происходит с потерями — в приближении сохраняется лишь наиболее существенная часть матрицы M.

Во многом благодаря этому свойству сингулярное разложение и находит широкое практическое применение: в сжатии данных, обработке сигналов, численных итерационных методах для работы с матрицами, методе главных компонент, латентно-семантическом анализе и прочих областях.

Сингулярное разложение и теорема SVD понадобятся в дальнейшем при рассмотрении одного из методов оптимизации.

Глава 2

Методы оптимизации

2.1 Попеременная оптимизация

В подходе попеременной минимизации (AltMin – Alternating Minimization) искомая матрица X низкого ранга записывается в билинейной форме, то есть $X = UV^T$. Матрицы U и V имеют r столбцов и m и n строк соответсвенно, поэтому ранг матрицы UV^T не превосходит r.

Тогда задачу можно переписать в виде

$$\begin{aligned} &\min_{U,V} f(UV^T) \\ &s.t.\ U \in \mathbb{R}^{m\times r},\ V \in \mathbb{R}^{n\times r} \end{aligned} \tag{2.1}$$

Алгоритм начинается с инициализации матриц U_0 и V_0 и постепенно минимизирует целевую функцию, фиксируя одну из матриц и минимизируя функцию по второй.

Основным преимуществом попеременной минимизации перед другими методами является то, что каждый шаг является дешевым с точки зрения вычислений и требует небольшого объема памяти, поскольку нужно отслеживать только $2k$ векторов.

2.2 IRM

Далее рассмотрим алгоритм итеративной минимизации ранга (IRM), также основанный на попеременной минимизации.

Хотя IRM в первую очередь разработан для RCOP с ранговыми ограничениями на положительно полуопределенные матрицы, была введена лемма о полуопределенном вложении для расширения алгоритма IRM на RCOP с ранговыми ограничениями на обычные прямоугольные матрицы. Более того, метод IRM применим к RCOP с ограничениями на ранг как сверху, так и снизу, а также с равенствами.

Algorithm 1: Алгоритм попеременной минимизации (AltMin)

Input : T – количество итераций, r – ограничение на ранг
Output: $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{n\times r}$
begin

1. **Initialize** Инициализировать матрицу U_0
2. **for** $t = 1, 2, \ldots, T$:
3. $V_t = \text{argmin}_{V\in\mathbb{R}^{n\times r}} f(U_{t-1}V^T)$
4. $U_t = \text{argmin}_{U\in\mathbb{R}^{m\times r}} f(UV_t^T)$
5. **end for**

end

Сходимость метода IRM доказывается с помощью теории двойственности и условий Каруша-Куна-Таккера, доказательство опустим.

Учитывая вышесказанное, сформулируем задачу RCOP следующим образом,

$$\begin{aligned} &\min_X f(X) \\ &s.t.\ X \in \mathbb{S}_+^n,\ rank(X) \le r \end{aligned} \tag{2.2}$$

Количество ненулевых собственных значений матрицы совпадает с ее рангом. Для неизвестной квадратной матрицы $U \in \mathbb{S}_+^n$ невозможно найти ее собственные значения пока она не определена. Обратим внимание на тот факт, что при ранге матрицы r она имеет r ненулевых собственных значений. Поэтому вместо ограничения ранга мы сосредоточимся на ограничении собственных значений U таким образом, чтобы все $n - r$ собственных значений U были нулями. Далее приведем необходимые утверждения и леммы, которые будут использоваться в дальнейшем.

Утверждение 4. *$(r + 1)e$ по величине собственное значение λ_{n-r} матрицы $U \in S_+^n$ не больше, чем e тогда и только тогда, когда $eI_{n-r} - V^TUV \succeq 0$, где I_{n-r} единичная матрица размера $n - r$, $V \in R^{n\times(n-r)}$ – матрица из собственных векторов, отвечающих $n - r$ наименьшим собственным значениям U.*

Доказательство: Предположим, что собственные значения U отсортированы по убыванию в виде $[\lambda_n, \lambda_{n-1}, \ldots, \lambda_1]$. Так как отношение Рэлея собственного вектора это собственное значение, которому он отвечает, то $eI_{n-r} - V^TUV$ диагональная матрица с диагональными элементами $[e - \lambda_{n-r}, e - \lambda_{n-r-1}, \ldots, e - \lambda_1]$. Следовательно, $e \ge \lambda_{n-r}$ тогда и только тогда, когда $eI_{n-r} - V^TUV \succeq 0$.

Следствие 5. *Если $e = 0$ и U положительно полуопределенная матрица, неравенство $rank(U) \le r$ выполнено тогда и только тогда, когда $eI_{n-r} - V^TUV \succeq 0$.*

Однако для задачи 2.2 до нахождения матрицы X мы не можем получить мат-

рицу V, т.е. собственные векторы X. Поэтому предлагается итерационный метод решения 2.2 путем постепенного приближения к заданному рангу. На каждом шаге k мы будем решать следующую задачу полуопределенного программирования, сформулированную ниже

$$\begin{aligned} \min_{X_k, e_k} \; & f(X_k) + \omega_k e_k \\ X_k & \in \mathbb{S}_+^n \\ e_k I_{n-r} - V_{k-1}^T X_k V_{k-1} & \succeq 0 \\ e_k & \leq e_{k-1} \end{aligned} \tag{2.3}$$

где $w_k = w_0 t^k$ - весовой коэффициент на итерации k. w_k растет с увеличением k, когда заданы параметры $t > 1$ и $w_0 > 0$. $V_{k-1} \in R^{n \times (n-r)}$ – ортонормированные собственные векторы, отвечающие $n-r$ наименьшим собственным значениям матрицы X_{k-1}, найденной на предыдущей итерации $k-1$. На первой итерации, когда $k = 1$, e_0 является $(n-r)$-м наименьшим собственным значением матрицы X_0, описанной ниже. Известно, что решатель задач полуопределенного программирования (SDP – Semi-Definite Programming), основанный на методе внутренней точки, имеет вычислительную временную сложность порядка $O(m(n^2 m + n^3))$ для задачи SDP с m линейными ограничениями и линейным матричным неравенством размерности n. В результате дополнительное линейное матричное неравенство $e_k I_{n-r} - V_{k-1}^T X_k V_{k-1} \succeq 0$ в подзадаче приводит к дополнительным линейным ограничениям порядка выше $O(n^2)$. Следовательно, временная сложность каждой итерации ограничена снизу величиной $O(n^6)$.

На каждом шаге мы пытаемся оптимизировать исходную целевую функцию и в то же время минимизировать параметр e, чтобы при $e = 0$ выполнялось ограничение на ранг матрицы X. Весовой коэффициент, w_k, действует как коэффициент регуляризации, и увеличение его значения на каждом шаге приводит к постепенному уменьшению $(r+1)$-го наибольшего собственного значения до нуля. Благодаря этому коэффициенту регуляризации наш алгоритм не просто переключается между поиском неизвестной матрицы и ее собственными векторами, он также приводит к быстрому уменьшению фиктивной переменной e_k до нуля. Кроме того, дополнительное ограничение $e_k \leq e_{k-1}$ гарантирует, что e_k монотонно убывает. Вышеупомянутый подход повторяется до тех пор, пока не будет выполнено $e \leq \varepsilon$, где ε - малый порог для критерия останова. Каждая итерация решается с помощью решателя задач SDP на основе метода внутренней точки, который применим к задачам SDP малого и среднего размера. Несложно расширить 2.3 на задачи с многоранговыми ограничениями. Для краткости здесь описана простейшая версия с ограничением одного ранга.

Кроме того, на первой итерации $k = 1$ требуется начальная матрица V_0. Интуитивно понятно что, нужно использовать решение задачи 2.3 с ослабленными ограничениями, отбросив последнее ограничение и штрафной член целевой функ-

ции. При этом предположении X_0 находится через

$$\begin{aligned} &\min_{X_0} f(X_0) \\ &s.t.\ X_0 \in \mathbb{S}_+^n \end{aligned} \tag{2.4}$$

Тогда V_0 – собственные векторы, соответствующие $n-r$ наименьшим собственным значениям X_0.

Algorithm 2: Последовательная минимизация ранга для решения задачи 2.2

Input : $w_0, t, \varepsilon_1, \varepsilon_2, k_{max}$
Output: матрица X^*, на которой достигается минимум
begin

1. **Initialize:** Положить $k = 0$, решить задачу 2.4 для получения матрицы V из X через спектральное разложение. Положить $k = k + 1$
2. **while** $k \leq k_{max}$ & $e_k \geq \varepsilon_1$ || $|(f(X_k) - f(X_{k-1}))/f(X_{k-1})| \geq \varepsilon_2$
3. Решить последовательную задачу 2.3 и найти X_k, e_k
4. Обновить V_k из X_k с помощью спектрального разложения и положить $k = k + 1$
5. Обновить w_k с помощью формулы $w_k = w_{k-1} * t$
6. **end while**

end

2.3 Алгоритм GECO

В этом разделе рассмотрим жадный алгоритм покомпонентной оптимизации или GECO (Greedy Efficient Component Optimization).

Далее будем искать решение задачи оптимизации вида:

$$\begin{aligned} &\min_{A} R(A) \\ &s.t.\ rank(A) \leq r \end{aligned} \tag{2.5}$$

где $R : \mathbb{R}^{m \times n} \to \mathbb{R}$ - выпуклая и гладкая функция.

Алгоритм GECO основан на простом, но действенном наблюдении: вместо представления матрицы A с использованием чисел $m \times n$ мы представляем ее с помощью бесконечномерного вектора λ, индексированного всеми парами (u, v)

взятыми из единичных сфер $\mathbb{R}^m$ и $\mathbb{R}^n$ соответственно. В этом представлении низкий ранг соответствует разреженности вектора λ.

Таким образом, мы можем свести задачу, заданную в уравнении 2.5, к задаче минимизации вектор-функции $f(\lambda)$ по множеству разреженных векторов, $\|\lambda\|_0 \leq r$. Основываясь на этой редукции, мы применяем алгоритм жадной аппроксимации для минимизации выпуклой векторной функции с учетом ограничения разреженности. На первый взгляд, прямое применение этой редукции кажется невозможным, поскольку λ - бесконечномерный вектор, и на каждой итерации жадного алгоритма нужно искать по бесконечному набору координат λ. Однако эту задачу поиска можно представить как задачу нахождения первых ведущих правых и левых сингулярных векторов матрицы.

Пусть $A \in R^{m \times n}$ – матрица размера m на n, и, не ограничивая общности, предположим, что $m \leq n$. Теорема SVD утверждает, что матрицу A можно записать в виде $A = \sum_{i=1}^{m} \lambda_i u_i v_i^T$, где $u_1, \ldots, u_m$ лежат во множестве $\mathcal{U} = \{u \in R^m : \|u\| = 1\}$, $v_1, \ldots, v_m$ – во множестве $\mathcal{V} = \{v \in R^n : \|v\| = 1\}$, а $\lambda_1, \ldots, \lambda_m$ - скаляры. Чтобы упростить представление, мы предполагаем, что каждое действительное число представлено с использованием конечного числа битов, поэтому наборы $\mathcal{U}$ и $\mathcal{V}$ являются конечными наборами. Отсюда следует, что мы также можем записать A в виде $A = \sum_{(u,v) \in \mathcal{U} \times \mathcal{V}} \lambda_{u,v} u v^T$, где $\lambda \in R^{|\mathcal{U} \times \mathcal{V}|}$, и мы индексируем элементы λ, используя пары $(u, v) \in \mathcal{U} \times \mathcal{V}$. Обратим внимание, что представление A с помощью вектора λ не единственно, но из теоремы SVD всегда существует представление A, для которого количество ненулевых элементов λ не больше чем m, т.е. $\|\lambda\|_0 \leq m$, где $\|\lambda\|_0 = |\{(u, v) : \lambda_{u,v} \neq 0\}|$. Кроме того, если $rank(A) \leq r$, то существует представление A с помощью вектора λ, для которого $\|\lambda\|_0 \leq r$

Для (разреженного) вектора $\lambda \in R^{|\mathcal{U} \times \mathcal{V}|}$ мы определяем соответствующую матрицу как

$$A(\lambda) = \sum_{(u,v) \in \mathcal{U} \times \mathcal{V}} \lambda_{u,v} u v^T \tag{2.6}$$

Обратим внимание, что $A(\lambda)$ – линейное отображение. Для функции $R : R^{m \times n} \to R$ мы определяем функцию

$$f(\lambda) = R(A(\lambda)) = R\left(\sum_{(u,v) \in \mathcal{U} \times \mathcal{V}} \lambda_{u,v} u v^T \right) \tag{2.7}$$

Легко проверить, что если R – выпуклая функция над $R^{m \times n}$, то f выпукла над $R^{|\mathcal{U} \times \mathcal{V}|}$ (поскольку f – композиция R над линейным отображением). Таким образом, мы можем свести задачу, заданную в уравнении 2.5, к задаче

$$\begin{aligned} &\min_{\lambda} f(\lambda) \\ &s.t.\ \lambda \in \mathbb{R}^{|U \times R|}, \|\lambda\|_0 \leq r \end{aligned} \tag{2.8}$$

Хотя задача оптимизации, заданная в уравнении 2.8, относится к произвольному большому пространству, прямая жадная процедура выбора может быть реализована эффективно. В начале жадного алгоритма полагается $\lambda = (0, \ldots, 0)$. На каждой итерации мы сначала находим векторы (u, v), которые максимизируют величину частной производной $f(\lambda)$ по $\lambda_{u,v}$. Предполагая, что R дифференцируема, и используя цепное правило, получаем:

$$\frac{\partial f(\lambda)}{\partial \lambda_{u,v}} = \langle \nabla R(A(\lambda)), uv^T \rangle = u^T \nabla R(A(\lambda)) v \tag{2.9}$$

где $\nabla R(A(\lambda))$ - матрица размера $m \times n$ частных производных R по элементам матрицы $A(\lambda)$. Векторы u, v, которые максимизируют величину вышеуказанного выражения, являются левым и правым сингулярными векторами, отвечающими наибольшему сингулярному значению $\nabla R(A(\lambda))$. Следовательно, даже несмотря на то, что количество элементов в $\mathcal{U} \times \mathcal{V}$ очень велико, мы все же можем эффективно выполнить жадный выбор одной пары $(u, v) \in \mathcal{U} \times \mathcal{V}$.

В некоторых ситуациях даже вычисление ведущих сингулярных векторов может оказаться слишком дорогостоящим. Поэтому допустима приближенная максимизация. Обозначим через ApproxSV $(\nabla R(A(\lambda)), \tau)$ функцию, которая возвращает векторы, для которых

$$u^T \nabla R(A(\lambda)) v \geq (1 - \tau) \max_{p,q} p^T \nabla R(A(\lambda)) q \tag{2.10}$$

Пусть U и V - матрицы, столбцы которых содержат векторы u и v, которые мы получили до этого момента. Второй шаг каждой итерации алгоритма устанавливает λ как решение следующей задачи оптимизации:

$$\min_{\lambda \in \mathbb{R}^{|\mathcal{U} \times \mathcal{V}|}} f(\lambda) \text{ s.t. } \operatorname{supp}(\lambda) \subseteq \operatorname{span}(U) \times \operatorname{span}(V) \tag{2.11}$$

где $\operatorname{supp}(\lambda) = \{(u, v) : \lambda_{u,v} \neq 0\}$ и $\operatorname{span}(U)$, $\operatorname{span}(V)$ - линейные оболочки столбцов U и V соответственно.

Далее опишем, как решить уравнение 2.11. Пусть s будет количеством столбцов в U и V. Обратим внимание, что любой вектор $u \in \operatorname{span}(U)$ можно записать как Ub_u, где $b_u \in \mathbb{R}^s$, и аналогично любой $v \in \operatorname{span}(V)$ можно записать как Vb_v. Следовательно, если носитель функции λ находится в $\operatorname{span}(U) \times \operatorname{span}(V)$, получаем, что $A(\lambda)$ может быть записано как

$$A(\lambda) = \sum_{(u,v) \in \operatorname{supp}(\lambda)} \lambda_{u,v} (Ub_u)(Vb_v)^T = U \left(\sum_{(u,v) \in \operatorname{supp}(\lambda)} \lambda_{u,v} b_u b_v^T \right) V^T \tag{2.12}$$

Таким образом, любая λ, носитель функции которой находится в $\operatorname{span}(U) \times \operatorname{span}(V)$, дает матрицу $B(\lambda) = \sum_{u,v} \lambda_{u,v} b_u b_v^T$. Теорема SVD говорит, что обратное

утверждение также верно, а именно, для любого $B \in \mathbb{R}^{s\times s}$ существует λ, носитель функции которого находится в $\text{span}(U) \times \text{span}(V)$, который порождает B (а также UBV^T). Обозначим $\tilde{R}(B) = R(UBV^T)$, из этого следует, что уравнение 2.11 эквивалентно следующей задаче оптимизации без ограничений $\min_{B\in\mathbb{R}^{s\times s}} \tilde{R}(B)$. Легко проверить, что $\tilde{R}$ - выпуклая функция, и поэтому ее можно эффективно минимизировать. Как только мы получим матрицу B, которая минимизирует $\tilde{R}(B)$, мы можем использовать ее SVD для нахождения соответствующей λ.

На практике нам не нужно подстраивать λ, а лишь подстраивать такие матрицы U, V, что $A(\lambda) = UV^T$.

Algorithm 3: GECO

1: **Ввод:** Выпукло-гладкая функция $R : R^{m\times n} \to R$; ограничение на ранг r; константа $\tau \in [0, 1/2]$
2: **Initialize:** $U = [], V = []$
3: **for** i=1,...,r **do**
4: (u, v)=ApproxSV$(\nabla R(UV^T), \tau)$
5: Положить $U = [U, u]$ and $V = [V, v]$
6: Положить $B = \text{argmin}_{B:\in\mathbb{R}^{i\times i}} R(UBV^T)$
7: Вычислить SVD: $B = PDQ^T$
8: Обновить: $U = UPD, V = VQ$
9: **end for**

Время выполнения алгоритма следующее. Шаг 4 можно выполнить за время $O(N \log(n)/\tau)$, где N - количество ненулевых элементов $\nabla R(UV^T)$, используя степенной метод. Поскольку анализ алгоритма позволяет τ быть константой (например, $1/2$), это означает, что время выполнения равно $O(N \log(n))$. Время выполнения шага 6 зависит от структуры функции R. Наконец, время выполнения шага 7 составляет не более r^3, шаг 8 можно выполнить за время $O(r^2(m + n))$.

Глава 3

Сравнение методов

Методы оптимизации для низкоранговых матриц применимы для многих задач, например:

- Приближение матрицей низкого ранга
- Заполнение матрицы
- Идентификация систем
- Обработка сигналов
- Редукция моделей

Протестируем первые два метода на второй задаче, то есть на задаче заполнения матрицы.

Эта задача может быть сформулирована в виде:

$$\begin{aligned} &\min_X \sum_{(i,j)\in E} (Y_{i,j} - X_{i,j})^2 \\ &s.t.\ rank(X) \leq r \end{aligned} \tag{3.1}$$

где E – множество пар индексов известных значений данной матрицы $Y \in \mathbb{R}^{m\times n}$.

Для решения задачи минимизации в алгоритме AltMin будем использовать функцию minimize модуля optimize библиотеки scipy.

Для реализации выпуклого программирования в алгоритме IRM воспользуемся библиотекой cvxpy. В качестве решателя возьмём решатель MOSEK, который проявил себя лучше других в ходе исследования.

В качестве заполняемой матрицы возьмем случайную матрицу Y размера m на n, состоящую из целых чисел от 0 до 10. Для того, чтобы решить, какие значения мы знаем, возьмем булевскую матрицу того же размера, в которой на каждой из позиций стоит 1 с вероятностью p и 0 с вероятностью $1 - p$.

Проведем эксперименты для матрицы размера 15 на 20, которую будем приводить к рангу 10, при $p = 0.8$ и $p = 0.9$ и для матрицы размера 30 на 45, которую будем приводить к рангу 20, при $p = 0.8$.

Для первого и второго эксперимента положим параметры ω_0 и t алгоритма IRM равными 0.1 и 1.5 соответственно. Для третьего эксперимента положим $\omega_0 = 1$, $t = 1.1$.

В ходе исследования будем отслеживать средний квадрат ошибки и время выполнения итераций.

В случае алгоритма IRM также будем отслеживать величину e_k, которая ограничивает сверху $n - r$-е наименьшее собственное значение.

Метод AltMin устроен так, что с первой итерации его решение имеет заданный ранг, поэтому приводить для него $n - r$-е собственное значение не имеет смысла.

Iteration	MSE	Time	Iteration	MSE	Time
1	0.4877	2.21	1	1.0135	1.67
2	0.1795	2.23	2	0.4895	1.53
3	0.1161	2.03	3	0.4135	1.77
4	0.0919	2.06	4	0.3719	1.44
5	0.0786	1.96	5	0.3385	1.31
6	0.0697	1.81	6	0.3068	1.43
7	0.0632	1.66	7	0.2777	1.38
8	0.0580	1.63	8	0.2538	1.39
9	0.0537	2.01	9	0.2361	1.78
10	0.0500	2.09	10	0.2241	1.68
11	0.0468	2.06	11	0.2163	1.66
12	0.0440	1.92	12	0.2113	1.67
13	0.0417	1.98	13	0.2081	1.62
14	0.0397	1.91	14	0.2059	1.64
15	0.0381	1.64	15	0.2043	1.62

Таблица 3.1: AltMin (m = 15, n = 20, rank = 10 p = 0.8, 0,9)

На рисунках 3.1 и 3.2 видно, что, как и предполагалось, метод IRM постепенно уменьшает $n - r$-е наименьшее собственное значение. Достаточно малым для достижения заданного ранга оно становится на 7 итерации для матрицы 15 на 20 и на 5 итерации для матрицы 30 на 45. Также из графиков можно сделать вывод, что алгоритм AltMin постепенно уменьшает средний квадрат ошибки, причем в третьем эксперименте время выполнения каждой последующей операции уменьшается, что уменьшает время работы в целом.

В первом эксперименте (левые части таблиц 3.1 и 3.2) при более разреженной матрице алгоритм AltMin имеет значительно лучшую точность, по сравнению с IRM. IRM же был немного быстрее, однако стоит учитывать, что этот метод должен продолжать работу, пока не будет достигнут заданный ранг.

При $p = 0.9$ (правые части таблиц 3.1 и 3.2) точность методов и время их работы схожи.

Iteration	e_k	MSE	Time	Iteration	e_k	MSE	Time
0	-	4.32e-28	0.011	0	-	4.36e-28	0.012
1	0.9179	0.2179	1.54	1	0.8986	0.2565	1.55
2	0.2830	0.2333	1.85	2	0.2864	0.2881	1.64
3	0.1122	0.2324	1.52	3	0.1230	0.2982	1.50
4	0.0482	0.2303	1.44	4	0.0555	0.3039	1.57
5	0.0210	0.2290	1.63	5	0.0245	0.3081	1.91
6	0.0097	0.2278	1.27	6	0.0112	0.3106	1.35
7	0.0043	0.2271	1.67	7	0.0048	0.3127	1.27
8	0.0019	0.2268	1.25	8	0.0023	0.3137	1.30
9	0.000867	0.2265	1.47	9	0.001043	0.3146	1.66
10	0.000401	0.2263	1.26	10	0.000542	0.3150	1.18
11	0.000198	0.2262	1.62	11	0.000202	0.3156	1.60
12	7.6425e-5	0.2262	1.43	12	0.000191	0.3153	1.19
13	3.8299e-5	0.2261	1.38	13	9.8068e-5	0.3155	1.62
14	1.9896e-5	0.226070	1.72	14	2.4713e-5	0.3159	1.65
15	1.8754e-5	0.226018	1.86	15	1.0938e-5	0.3160	1.66

Таблица 3.2: IRM (m = 15, n = 20, rank = 10 p = 0.8, 0,9)

Во третьем эксперименте (таблица 3.3) используется матрица большего размера. В нем лучше себя показал метод AltMin, дав более высокую точность и скорость выполнения.

В ходе сравнения заметно лучше показал себя метод попеременной минимизации. В целом он дал более точное решение и показал более быструю сходимость. К тому же метод AltMin с первой итерации дает решение заданного ранга, то есть его $n - r$ наименьших собственных значений гарантированно равны 0.

Сравним полученное время работы методов с теоретической оценкой. При рассмотрении метода IRM приводилась оценка времени выполнения очередной итерации $O(n^6)$. В случае прямоугольной матрицы мы дополнительно приводим квадратную матрицу размера $m + n$ к рангу m. Поэтому оценкой в нашем случае является величина $O((m + n)^6)$. Для метода AltMin оценка определяется временем работы функции minimize, в которой используется алгоритм BFGS (Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно). Этот алгоритм имеет оценку $O(n^2)$. Так как на вход функции подается максимум nr переменных, оценкой времени выполнения итерации нашего метода будет величина $O((nr)^2)$

Для сравнения возьмем 1 и 3 эксперименты (матрицы одинаково разреженны и разного размера) Вместо второго эксперимента проведем новый с тем же $p = 0.8$ и матрицей размера 20 на 30, которую будем приводить к рангу 15.

Для метода AltMin среднее время выполнения итерации в первом эксперименте составило 1.95, во втором – 10.504, в третьем – 49.43.

Для метода IRM – 1.53, 7.808 и 89.52 соответственно (не считая 0-ю итерацию).

Iteration	MSE	Time
1	0.6345	68.09
2	0.2117	56.78
3	0.10748	48.32
4	0.0682	44.02
5	0.0482	39.21
6	0.0359	40.14

Iteration	e_k	MSE	Time
0	-	7.92e-23	0.063
1	0.8517	0.3419	86.92
2	0.1547	0.3760	100.83
3	0.0371	0.3853	107.84
4	0.0101	0.3893	79.91
5	0.0033	0.3904	77.42
6	0.0022	0.3892	84.20

Таблица 3.3: AltMin, IRM (m = 30, n = 45, rank = 20, p = 0.8)

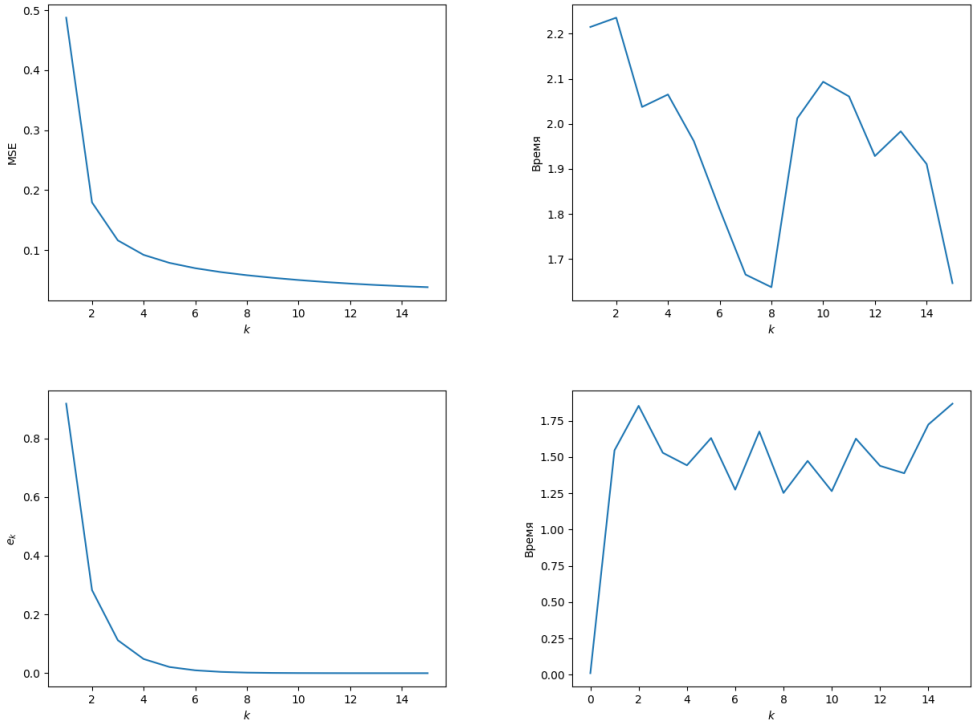

Рис. 3.1: AltMin, IRM (m = 15, n = 20, rank = 10, p = 0.8)

Постоим графики оценок и посмотрим насколько они совпадают с полученными значениями. В качестве коэффициента перед $(m+n)^6$ возьмем отношение среднего времени выполнения итерации алгоритма IRM к значению функции $f(x) = x^6$ в точке $m+n$ для 2 эксперимента. В качестве коэффициента перед $(nr)^2$ возьмем отношение среднего времени выполнения итерации алгоритма AltMin к значению функции $f(x) = x^2$ в точке nr для 2 эксперимента.

Из графиков на рисунке 3.3 видно, что полученные результаты совпадают с теоретической оценкой.

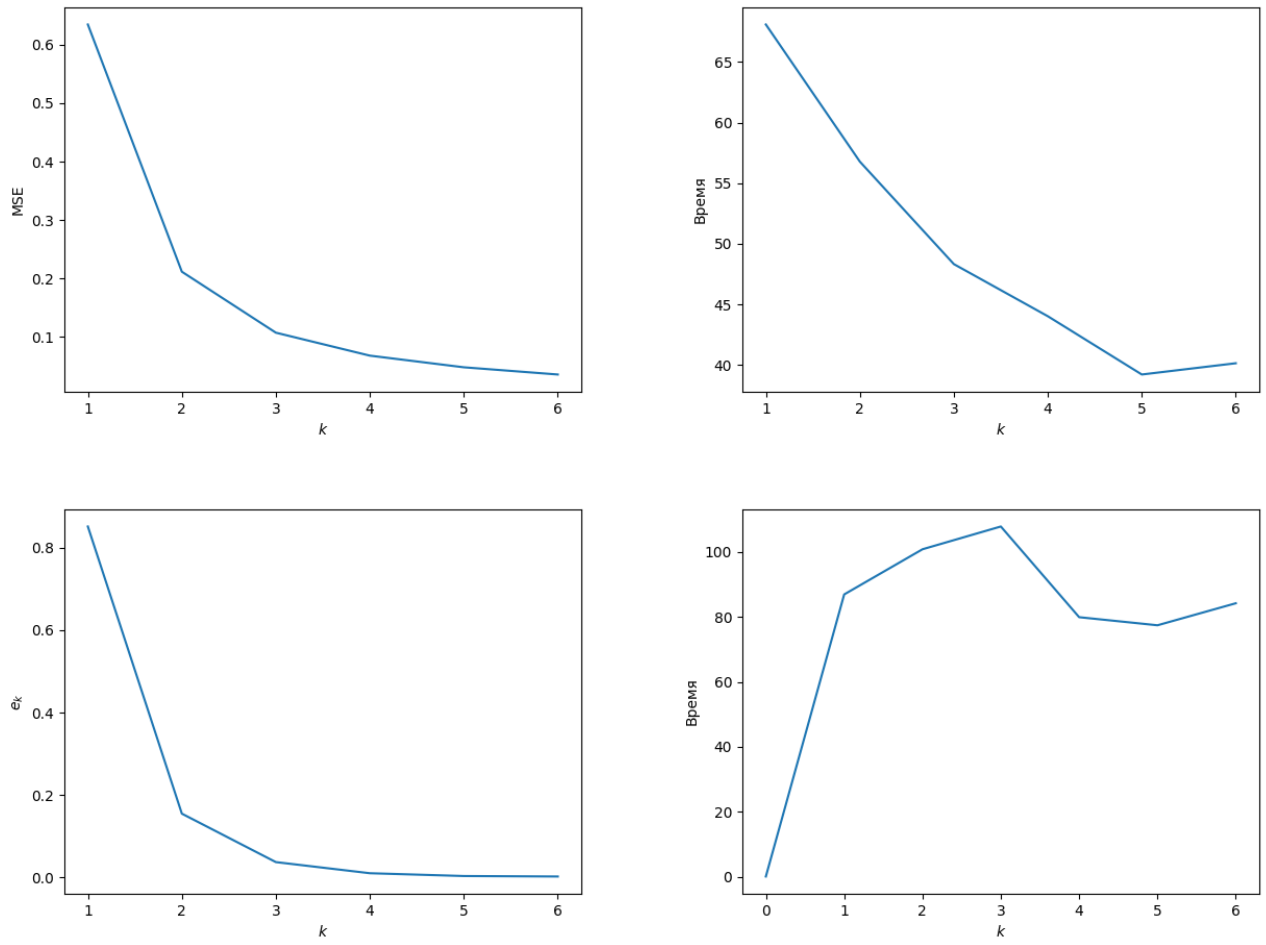

Рис. 3.2: AltMin, IRM (m = 30, n = 45, rank = 20, p = 0.8)

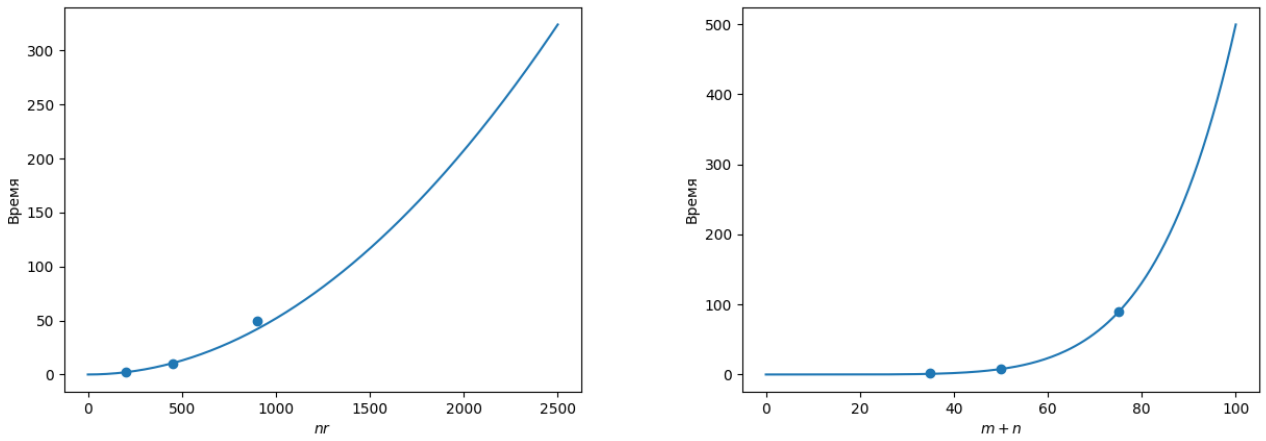

Рис. 3.3: AltMin, IRM

Глава 4

Применение

4.1 Заполнение матрицы

Заполнение матрицы - это задача предсказания элементов некоторой неизвестной искомой матрицы $Y \in R^{m \times n}$ на основе случайного подмножества наблюдаемых элементов $E \subset [m] \times [n]$. Например, в известной задаче Netflix m представляет количество пользователей, n представляет количество фильмов, а $Y_{i,j}$ - рейтинг, который пользователь i дает фильму j. Один из подходов к заполнению матрицы Y - найти матрицу A низкого ранга, которая приблизительно совпадает с Y на элементах E (в смысле среднего квадрата ошибки).

Тогда можно сформулировать задачу:

$$\begin{aligned} &\min_A \frac{1}{|E|} \sum_{(i,j) \in E} (A_{i,j} - Y_{i,j})^2 \\ &s.t.\ rank(A) \leq r \end{aligned} \tag{4.1}$$

4.2 Аппроксимация матрицей низкого ранга

Очень распространенная задача при анализе данных – найти матрицу A низкого ранга, которая аппроксимирует данную матрицу Y, а именно решение

$$\begin{aligned} &\min_A d(A, Y) \\ &s.t.\ rank(A) \leq r \end{aligned} \tag{4.2}$$

где d – некоторая мера несоответствия. Для простоты предположим, что $Y \in R^{n \times n}$. Когда $d(A, V)$ – нормированная норма Фробениуса $d(A, V) = \frac{1}{n^2} \sum_{i,j} (A_{i,j} - Y_{i,j})^2$, эта задача решается эффективно через SVD. Однако хорошо известно, что из-за использования нормы Фробениуса эта процедура чувствительна к выбросам.

4.3 Задача идентификации систем

Рассматривая линейную инвариантную во времени систему с входом $u \in \mathbb{R}^m$ и выходом $y \in \mathbb{R}^p$, в этой задаче требуется идентифицировать линейную систему через T выборок $w_d := (u_d, y_d) \in (\mathbb{R}^m \times \mathbb{R}^p)^T$. Такая задача идентификации эквивалентна поиску полной матрицы рангов строк $R = [R_0, R_1, ..., R_l]$ такой, что

$$R_0 w(t) + R_1 w(t+1) + \ldots + R_l w(t+l) = 0, \ \forall t = 1, 2, \ldots, T - l \tag{4.3}$$

где l - отставание идентифицированной модели. Выражение в 4.3 можно переписать в виде $R\mathscr{H}_{l+1,T-l}(w) = 0$, где $\mathscr{H}_{l+1,T-l}(w) = \begin{bmatrix} w(1) & w(2) & \ldots & w(T-l) \\ w(2) & w(3) & \ldots & w(T-l+1) \\ \vdots & \vdots & \ldots & \vdots \\ w(2) & w(3) & \ldots & w(T-l+1) \end{bmatrix}$

Поскольку размерность строки R равна количеству выходов p модели, ранг $\mathscr{H}_{l+1,T-l}(w)$ ограничен p. В результате проблема идентификации эквивалентна следующей задаче аппроксимации при условии низкого ранга:

$$\begin{aligned} &\min_{w} \|w - w_d\|_F^2 \\ &s.t.\ rank(\mathscr{H}_{l+1,T-l}(w)) \leq r \end{aligned} \tag{4.4}$$

где $r = (m+p)(l+1) - p$.

Литература

[1] Chuangchuang Sun, Ran Dai *Rank-Constrained Optimization: Algorithms and Applications*, 2018

[2] Shai Shalev-Shwartz, Alon Gonen, Ohad Shamir *Large-Scale Convex Minimization with a Low-Rank Constraint* , June 2011

[3] Prateek Jain, Praneeth Netrapalli, Sujay Sanghavi *Low Rank Matrix Completion using Alternating Minimization*, November 2016

[4] Prateek Jain, Praneeth Netrapalli, Sujay Sanghavi *Low-rank Matrix Completion using Alternating Minimization*, December 4, 2012

[5] Moritz Hardt *Understanding Alternating Minimization for Matrix Completion*, May 15, 2014