Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгебраические вычисления тензоры и оптимизация

Под редакцией Е.Е. Тыртышникова

Учебно-методическое пособие
для студентов магистратуры филиала МГУ в г. Сарове

Москва
МАКС Пресс
2023

ББК 22.143
УДК 512.6+514.1
А45

Печатается по решению редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ имени М. В. Ломоносова

Коллектив авторов:

С. А. Матвеев, Е. Е. Тыртышников

А45 **Алгебраические вычисления тензоры и оптимизаций:** учебно-методическое пособие / под ред. Е.Е. Тыртышникова. — М.: Издательский отдел факультета ВМК МГУ имени М. В. Ломоносова; МАКС Пресс, 2023. — 42 с.

ISBN XXX-X-XXXXX-XXX-X

Настоящее пособие содержит методические материалы, сопровождающие учебный процесс по курсу "Алгебраические вычисления, тензоры и оптимизация"для студентов филиала МГУ в г. Сарове. В качестве введения в курсе формулируются необходимые базовые сведения об алгоритмах построения малоранговых матричных разложений (LU, QR, SVD, адаптивная крестовая аппроксимация, рандомизированные методы линейной алгебры), крайне важных для дальнейшей работы студентов с тензорными разложениями. Дополнительно слушателям курса рассказывается о применениях матричных и численных методов в задачах теории графов, решения систем кинетических уравнений типа Смолуховского и глобального анализа чувствительности многомерных функций.

Основное содержание курса посвящено трём популярным видам тензорных разложений (каноническому, Таккера и тензорному поезду) и их применениям в современных задачах вычислительной математики и анализа данных.

В пособии можно найти программу курса, план тем для семинарских занятий, образцы задач практикума, зачётных заданий, учебные заметки по отдельным темам курса, списки рекомендованной литературы для более подробного ознакомления с отдельными разделами курса.

ББК 22.143
УДК 512.6+514.1

ISBN XXX-X-XXXXX-XXX-X

Содержание

1 Предисловие.

В 2021 году в истории всего Московского университета и, в частности, факультета ВМК произошло важное событие: при национальном центре физики и математики был открыт филиал МГУ в г. Сарове. В этом филиале на момент открытия были запущены несколько программ магистратуры по направлениям, связанным с актуальными для страны задачами по прикладной математике и физике. Одна из таких программ была названа “Вычислительные методы и методика моделирования”.

В рамках учебного плана этой программы академику Е. Е. Тыртышникову и преподавателям кафедры вычислительных технологий и моделирования (ВТМ) предложили сформировать курс лекций “Алгебраические вычисления, тензоры и оптимизация” (36 акад. часов), предусматривающий проведение семинарских занятий (18 акад. часов). Курс был сформирован и впервые состоялся в 2021 году. По итогам 2022-2023 учебного года года авторам курса и руководству кафедры математики стала ясной необходимость подготовки актуальных учебно-методических материалов для студентов. Несмотря на то, что похожий курс под названием “Матрицы, тензоры, вычисления” уже достаточно давно входит в обязательную программу кафедры ВТМ факультета ВМК, не существовало учебно-методического пособия для сопровождения данного курса.

Причина отсутствия такого пособия (хотя его заготовку легко найти в интернете) лежит на поверхности – на кафедре ВТМ в Москве учится сравнительно немного студентов, поэтому у преподавателей всегда есть возможность выстроить с ними индивидуальное взаимодействие. В случае филиала МГУ в г. Сарове число слушателей, для которых курс является обязательным, значительно возросло, поэтому подготовка этого пособия, с учётом специфики новых учебных программ, стала неизбежной необходимостью.

Авторы благодарны за многочисленные обсуждения материалов разных тем, связанных с этим курсом, своим коллегам по факультету ВМК и ИВМ РАН Н. Л. Замарашкину, А. И. Осинскому, Д. А. Желткову, И. В. Оселедцу, М. С. Смирнову, А. В. Зылю, С. С. Морозову и всем студентам филиала МГУ в г. Сарове и кафедры ВТМ факультета ВМК.

2 Программа курса.

1. Напоминание о нормах и метриках в конечномерных пространствах.
2. Нормальные матрицы, унитарные матрицы, теорема Шура.
3. Ранги матриц и тензоров и разделение переменных.
4. Сингулярное разложение матрицы.
5. Малоранговые приближения матриц и тензоров.
6. Приниципп максимального объема при построении малоранговых приближений.
7. Крестовые методы построения приближений по неполной информации.
8. Тензорные разложения: сумма разложимых тензоров, разложение Таккера, тензорный поезд.
9. Алгоритм ALS (Alternating Least Squares) построения канонического тензорного разложения.
10. Применения канонического разложения в конкретных задачах.
11. Сингулярное разложение Таккера.
12. Алгоритм TTSVD для построения тензорного поезда.
13. Рандомизированное сингулярное разложение и его применение для поиска тензорных разложений.
14. QTT-формат и тензоризация векторов.
15. Редукция рангов тензорного поезда.
16. Циркулянтные матрицы, тёплицевы матрицы, спектральная теорема циркулянта.
17. Быстрое вычисление апериодической свёртки, её применения.
18. Теорема Перрона-Фробениуса, матрица связности, агоритмы Pagerank и Simrank.
19. Метод глобальной чувствительности Соболя, редукция размерности через разложение ANOVA.

3 Вопросы для программы государственного экзамена.

1. Сингулярное разложение матрицы, задача о наилучшем приближении матрицы малоранговой, задача о наименьших квадратах, нормальное псевдорешение.
2. Основные определения для трёх видов тензорных разложений: разложение Таккера, тензорный поезд, каноническое разложение.
3. Каноническое разложение тензора алгоритма матричного умножения, алгоритм Штрассена, процедура ALS для получения приближенного канонического разложения.
4. Разложение Таккера, алгоритм HOSVD и его вычислительная сложность. Тензорный поезд, алгоритм TTSVD, тензоризация для векторов (QTT-формат).
5. Циркулянтные и тёплицевы матрицы, их свойства.

4 Примерные темы семинарских занятий.

В рамках курса предусмотрено 18 академических часов для проведения семинарских занятий. Примерный список тем этих занятий приводится ниже. По некоторым отдельным темам далее будут приведены примеры реализации конкретных алгоритмов и сформулированы упражнения для домашней работы студентов.

1. Напоминание о необходимом минимуме фактов по вычислительной линейной алгебре: ранг матрицы, LU-разложение, нормальные матрицы, нормы и расстояния в конечномерных пространствах, теорема Шура, сингулярное разложение матрицы.

2. Напоминание о необходимом минимуме фактов по вычислительной линейной алгебре: матричные и операторные нормы, разложение Холецкого, QR-разложение (методы вращений, отражений и Грама-Шмидта).

3. Напоминание о необходимом минимуме фактов по вычислительной линейной алгебре: решение линейной задачи о наименьших квадратах, псевдообратная матрица, сжатие данных, понятие об адаптивной крестовой аппроксимации.

4. Каноническое тензорное разложение: сжатие данных при его использовании, понятие канонического тензорного ранга, алгоритм метод чередующихся наименьших квадратов (ALS – alternating least squares), построение эффективных алгоритмов суммирования с использованием канонического формата.

5. Каноническое тензорное разложение: применения канонического формата при понижении сложности матричного умножения, обработке спутниковых снимков, численном решении уравнений математической физики.

6. Разложение Таккера: определение, отличие от канонического разложения, матрицы развертки и связь их рангов с минимальными рангами Таккера, алгоритмы HOSVD и st-HOSVD.

7. Разложение Таккера: упражнения по вычислению рангов функционально-порожденных тензоров, использование разложения Таккера для построения эффективных алгоритмов суммирования, программный практикум.

8. Тензорный поезд: определение, связь с разложением Таккера, отличие от двух других форматов, другие матрицы развёртки и связь их рангов с рангами тензорного поезда. Количество ячеек памяти, необходимых для хранения тензорного поезда, алгоритм TTSVD.

9. Тензорный поезд: арифметические операции в TT-формате, построение эффективных алгоритмов на основе тензорных поездов, проблема роста рангов.

10. Тензорный поезд: алгоритмы редукции рангов тензорных поездов, тензоризация векторов и QTT-формат.

11. Циркулянтные и Тёплицивы матрицы и их свойства. Матрица дискретного преобразования Фурье, собственные значения и базис из собственных векторов для циркулянтных матриц. Использование циркулянтных и Тёплицивых матриц для вычисления апериодической свёртки.

5 Методические материалы по отдельным темам.

5.1 Необходимые сведения о матричных разложениях.

5.1.1 Сингулярное разложение.

Для любой вещественной/комплексной матрицы существует наилучшее приближение ранга не выше r, которое может быть построено при помощи усечённого сингулярного разложения (SVD – singular value decomposition) в случае унитарно-инвариантных норм. Важными примерами таких норм являются спектральная норма

$$||A||_2 = \max_{x \neq 0} \frac{||Ax||_2}{||x||_2} = \max_{||x||_2 = 1} ||Ax||_2$$

и норма Фробениуса

$$||A||_F = \sqrt{\mathrm{tr}(AA^\top)} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2}.$$

В более общем случае читатели могут ознакомиться, например, с нормами Шаттена. Для простоты изложения ограничимся записью сингулярного разложения для вещественной матрицы

$$A = U\Sigma V, \quad A \in \mathbb{R}^{M \times N}, \quad U \in \mathbb{R}^{M \times M}, \quad V \in \mathbb{R}^{N \times N}, \quad \Sigma \in \mathbb{R}^{M \times N},$$

где U состоит из ортонормированных столбцов, V из ортонормированных строк, а матрица Σ является диагональной матрицей с неотрицательными элементами на диагонали, упорядоченными в порядке невозрастания. Диагональные элементы матрицы Σ называются сингулярными числами. Напомним, что число ненулевых сингулярных чисел $\sigma_\alpha > 0$ в точности равно рангу исходной матрицы A, поэтому всегда можно записать усечённый вид сингулярного разложения

$$A = U_r \Sigma_r V_r, \quad A \in \mathbb{R}^{M \times N}, \quad U_r \in \mathbb{R}^{M \times r}, \quad V_r \in \mathbb{R}^{r \times N}, \quad \Sigma_r \in \mathbb{R}^{r \times r},$$

где U_r состоит из первых ортонормированных столбцов матрицы, V_r из первых ортонормированных строк, а матрица Σ_r является ведущей диагональной подматрицей матрицы Σ порядка r с положительными элементами на диагонали, упорядоченными в порядке невозрастания.

Построение сингулярного разложения сводится к вычислению собственных значений и собственных векторов симметричной матрицы $A^\top A \in \mathbb{R}^{n \times n}$. На практике, для этого используют адаптации алгоритмов для задач на поиск собственных значений, которые явным образом не строят $A^\top A$.

Два основных типа алгоритмов:

- бидиагонализация $A = U_B B V_B^T$ с факторами $U_B \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times n}$, $V_B \in \mathbb{R}^{n \times n}$

  $$U_B^T U_B = I_n, \quad V_B^T V_B = I_n, \quad b(i,j) = 0,\ j - i \notin \{0, 1\}$$

  и вычисление SVD для двухдиагональной B,
- методы Якоби.

Отметим, что сингулярное разложение является базовым инструментом для сжатия черно-белых изображений, метода главных компонент, решения классической линейной задачи о наименьших квадратах $||Ax - b||_2^2 \to \min\limits_x$ и построения псевдообратной матрицы $A^\dagger$.

Построение полного сингулярного разложения требует $O\left(\min(M \cdot N^2,\ N \cdot M^2)\right)$ действий. Его усечённая запись позволяет построить матричное скелетное разложение/приближение из R слагаемых с разделяемыми переменными i и j

$$A(i,j) = \sum_{\alpha=1}^{R} u_\alpha(i) \sigma_\alpha v_\alpha(j).$$

Подробнее данные темы разобраны в учебниках [1-2] из списка литературы в конце данного раздела, особенно рекомендуется разобрать задачи в конце глав 1 и 2 из учебника [2].

5.1.2 Идея рандомизированного SVD.

Предположим $M \leq N$, и $R \ll \min(M, N)$, дополнительно зафиксируем константу *избыточности* p. Рандомизированное сингулярное разложение можно построить, если необходимо получить квазиоптимальную аппроксимацию матрицы ранга R (напомним условие $R \ll M$) быстрее, чем за полное время SVD. Оно может быть получено при предварительном домножении оригинальной матрицы на случайную матрицу $\Omega \in \mathbb{R}^{N\times(R+p)}$. Такое умножение позволяет получить матрицу зарисовки (или скетча) B по следующему правилу

$$B = (A \cdot \Omega) \in \mathbb{R}^{M\times(R+p)}.$$

Сложность такого матричного умножения составит $O(MN(R+p))$ операций. Так как матрица B содержит гораздо меньше столбцов, для неё можно получить QR-разложение (например, через ортогонализацию Грама-Шмидта для столбцов) за $O((R+p)^2M)$ действий. Таким образом получаем:

$$B = QR.$$

Далее отметим, что сгенерированная матрица Q состоит из ортонормированных столбцов $Q \in \mathbb{R}^{M\times(R+p)}$, то есть $Q^\top Q = I_{R+p}$. С другой стороны, $Q \cdot Q^\top$ задаёт *неплохой* оператор проектирования для пространства столбцов исходной матрицы (данный факт прост в формулировке, однако его доказательство мы не приводим, так как оно потребует значительных усилий и использования леммы Джонсона-Линденштраусса, оригинальное доказательство которой доступно в [8]). Далее получаем

$$A_R = Q(Q^\top A) \approx QU_0\Sigma_0V_0.$$

Следовательно, восстанавливаем аппроксимацию для факторов усечённого сингулярного разложения матрицы A через следующее разложение ранга R:

$$U_R \approx QU_0, \Sigma_R \approx \Sigma_0, V_R \approx V_0.$$

Дополнительное матричное умножение $Q \cdot U_0$ потребует ещё $O(R^2M)$ действий. В результате, вычисление усечённого сингулярного разложения с использованием рандомизированного скетча занимает $O(MN(R+p) + M(R+p)^2 + N(R+p)^2)$ действий. В случае $R \ll M$, $R \ll N$ получается существенное снижение вычислительных затрат.

Теория, позволяющая проверить дополнительную погрешность, возникающую из-за случайной структуры матрицы скетча, а также из-за потери точности в используемых уравнениях, достаточно сложна, и выходит за рамки данного курса. Выбор параметра избыточности может быть скорректирован с помощью оценок точности такой версии квазиоптимальных приближений (см., например, обзор [9]). На практике часто принимают $p = 8$.

Программные реализации алгоритмов построения сингулярного разложения и его рандомизированного аналога доступны в библиотеках для многих языков программирования, в частности, `Python` в библиотеках `numpy` и `sklearn`. Отметим, что модуль `numpy` является лишь интерфейсом к классическим библиотекам BLAS (Basic Linear Algebra Subroutines) и LAPACK (Linear Algebra PACKage), реализованных на языке программирования Fortran.

Ниже мы приведём примеры небольших программ, с реализацией вычислительных примеров использования упомянутых алгоритмов.

Импортируем необходимые библиотеки:

```
import numpy as np
from sklearn.utils.extmath import randomized_svd as rsvd
from matplotlib import pyplot as plt
import time
```

Далее задаём параметры запуска учебной программы, реализацию SVD будем тестирвоать на конкретном примере матрицы Гильберта:

```
def A(i, j):
    return 1.0/(i+j+1)
```

Для матрицы Гильберта зафиксируем размеры 1500×1024, ранг $r = 20$ и заполним её:

```
M = 1500 # number of rows
N = 1024 # number of columns
x = np.linspace(1,N,N) # grid for drawings
r = 20 # target rank
# fill the matrix Ma
Ma=np.fromfunction(A, [M, N])
```

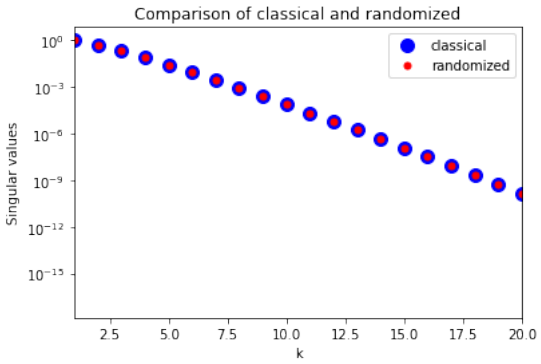
Далее измерим времена построения усечённого сингулярного разложения и его рандомизированного аналога

```
# Estimate the time , does not save the data
start = time.time()
U, s, V = np.linalg.svd(Ma)
## Truncation of SVD
Ut = U[:,:r] # m \times r
st = s[:r] # r
Vt = V[:r,:] # # r \times n
np.linalg.norm(Ma - Ut @ np.diag(st) @ Vt)
print("Classical ", time.time() - start, " sec")
start = time.time()
Ur, sr, Vr = rsvd(Ma, n_components=r, random_state=np.random.randint(1))
print("Randomized ", time.time() - start, " sec")
```

Так как аппроксимации ранга R нами уже построены, оценим их качество в норме Фробениуса,

```
# Check the error for singular values
plt.title("Comparison of classical and randomized")
plt.plot(x, s/s[0], 'bo', markersize=10, label='classical')
plt.plot(x[:r], sr/s[0], 'ro', markersize=5, label='randomized')
plt.ylabel('Singular values')
plt.xlabel('k')
plt.xlim(1,r)
plt.legend()
plt.yscale('log')
print("Relative error of sigmas=", np.linalg.norm(s[:r] - sr)/np.linalg.norm(s[:r]))
print("Classical relative approximation error = ", np.linalg.norm(s[r:])/np.linalg.
    norm(s))
print("Randomized straight-forward error = ", np.linalg.norm(Ma - (Ur @ np.diag(sr) @
     Vr)) / np.linalg.norm(s))
print("Initial storage", M*N, " memory cells")
print("Compressed storage", (1+M+N)*r, " memory cells")
```

При вызове такого участка программного кода получим график сингулярных чисел



Из графика легко увидеть, что результаты рандомизированного сингулярного разложения очень близки к оригинальным (оптимальным с точки зрения точности аппроксимаций) сингулярным значениям. В то же время мы получим при печати следующие результаты:

```
Relative error of sigmas =  6.015800151775415e-16
Classical relative approximation error =  3.322658295342276e-11
Randomized straight-forward error =  3.322660547744596e-11
Initial storage 1536000  memory cells
Compressed storage 50500  memory cells
```

Данные результаты наглядно демонстрируют уровень относительной ошибки в норме Фробениуса (порядка 10^{-11}) для построенных аппроксимаций и полученный уровень сжатия от исходных 1536000 ячеек памяти до 50500.

5.1.3 Матричный крестовый метод.

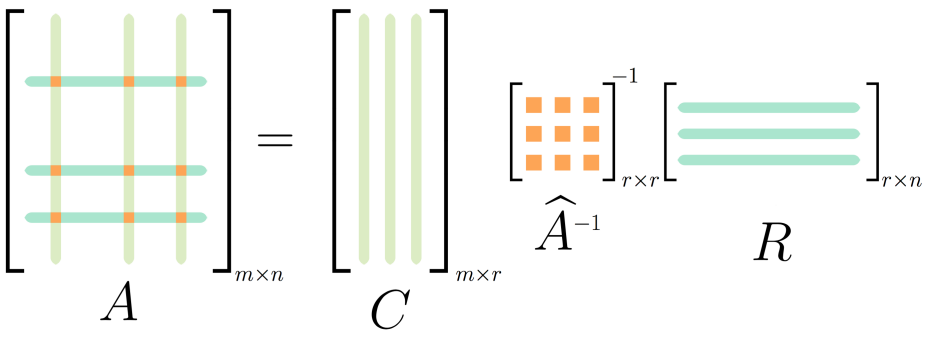
Описание матричного крестового метода мы приводим, повторяя текст статьи [3] из списка литературы в конце данного подраздела. В основе идеи крестового метода для матриц лежит следующий известный факт. Пусть

- $A \in \mathbb{R}^{m\times n}$, $\operatorname{rank} A = r$,
- $\hat{A} \in \mathbb{R}^{r\times r}$ – подматрица матрицы A, $\det \hat{A} \neq 0$,
- $C \in \mathbb{R}^{m\times r}$ – столбцы матрицы A, содержащие $\hat{A}$,
- $R \in \mathbb{R}^{r\times n}$ – строки матрицы A, содержащие $\hat{A}$.

Тогда

$$A = C\hat{A}^{-1}R$$

(фактически, это хорошо известная теорема о базисном миноре). Проиллюстрировать данную идею можно при помощи следующей картинки:



Пусть теперь A полного ранга, но существует матрица $E \in \mathbb{R}^{m \times n}$, такая что её спектральная или фробениусовская норма близка к нулю, и $\mathrm{rank}(A - E) = k$. Известно, что если в качестве $\hat{A}$ выбрать подматрицу матрицы A максимального объёма (то есть обладающую наибольшим по модулю определителем), то и $A - C\hat{A}^{-1}R$ также будет близка к нулю в смысле спектральной, фробениусовской и C-нормы ($\|A\|_C = max_{i,j}|A_{i,j}|$) [4]. Поиск подматрицы максимального объёма – вычислительно сложная задача (строго говоря, NP-трудная) [5-7], поэтому на практике следует ограничиться достаточно большим, а не максимальным объёмом. В этом случае $A - C\hat{A}^{-1}R$ также будет близка к нулю. Поэтому для отбора подматриц используется алгоритм, схема которого представлена ниже:

Algorithm 5.1: Схема крестового метода(m, n, A, k)

$I \leftarrow \{1, \ldots, m\}$
$J \leftarrow \{1, \ldots, n\}$
$r \leftarrow 0$
repeat
  $r \leftarrow r + 1$
  $maxvol(A, k, I, J, I_*, J_*)$
  $C_r \leftarrow A[:, J_*]$ выбор столбцов по множеству J_*
  $R_r \leftarrow A[I_*, :]$ выбор строк по множеству I_*
  $\hat{A}_i \leftarrow A[I_*, J_*]$
  $A \leftarrow A - C_r\hat{A}_r^{-1}R_r$
  $I \leftarrow I \setminus I_*$
  $J \leftarrow J \setminus J_*$
until $stoppingcriteria$

В этой схеме процедура $maxvol(A, k, I, J, I_*, J_*)$ позволяет выполнить поиск подматрицы порядка k достаточно большого объёма в подматрице исходной матрицы A, содержащейся в строках I и столбцах J, номера строк и столбцов содержащих найденную подматрицу записываются в I_* и J_*. $stoppingcriteria$ – некоторый критерий остановки.

В результате работы этого алгоритма матрица $\sum_{i=1}^{r} C_i\hat{A}_i^{-1}R_i$, состоящая из суммы одноранговых матриц, будет приближать исходную матрицу A. Наиболее простым для реализации кажется крестовый метод с $k = 1$, т. е. $maxvol$ осуществляет поиск большого по модулю элемента. В таком случае C_i, $\hat{A}_i$ и R_i являются, соответственно, вектором-столбцом, числом и вектором строкой. Поэтому, обозначим:

$$u_i = \frac{C_i}{\sqrt{|\hat{A}_i|}}, v_i = \frac{R_i\sqrt{|\hat{A}_i|}}{\hat{A}_i},$$

$$U_k = \begin{bmatrix} u_1 & u_2 & \dots & u_k \end{bmatrix}, V_k = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

Тогда

$$A \approx UV$$

На практике часто используется следующий критерий остановки:

$$|(A - UV)_{ij}| \cdot \sqrt{(m-r)(n-r)} < \varepsilon \|UV\|_F,$$

где ε – параметр, отвечающий за точность аппроксимации, i и j – позиции максимального найденного элемента текущей погрешности. Этот критерий хорош тем, что с одной стороны является достаточно надёжным, так как для того, чтобы была найдена аппроксимация точности, не ниже требуемой, достаточно, чтобы найденный нами элемент по модулю был не меньше среднего квадратичного, а на практике это, как правило, так. С другой стороны, он достаточно точный, и на практике ранг приближения несильно отличается от минимального ранга, необходимого крестовому методу. С учётом данного выбора алгоритм выглядит следующим образом:

Algorithm 5.2: Крестовый метод(m, n, A)

$I \leftarrow \{1, \dots, m\}$
$J \leftarrow \{1, \dots, n\}$
$r \leftarrow 0$
while true
do
- $r \leftarrow r + 1$
- $i \leftarrow \arg\max_{i \in I} |A_{iJ(1)} - [UV]_{iJ(1)}|$
- $j \leftarrow \arg\max_{j \in J} |A_{ij} - [UV]_{ij}|$
- **if** $|(A - UV)_{ij}| \cdot \sqrt{(m-r)(n-r)} < \varepsilon \|UV\|_F$
  - **then**
    - $r \leftarrow r - 1$
    - **break**
- $u \leftarrow \frac{A^j - [UV]^j}{\sqrt{|(A-UV)_{ij}|}}$
- $v \leftarrow \frac{(A_i - [UV]_i)\sqrt{|(A-UV)_{ij}|}}{(A-UV)_{ij}}$
- $U \leftarrow \begin{bmatrix} U & u \end{bmatrix}$
- $V \leftarrow \begin{bmatrix} V \\ v \end{bmatrix}$
- $I \leftarrow I \setminus \{i\}$
- $J \leftarrow J \setminus \{j\}$

return (U, V, r)

Отметим, что вычисление $\|UV\|_F$ напрямую сложно и требует $O(mn)$ операций. Однако,

$$\|UV\|_F^2 = \mathrm{tr}(V^T U^T U V) = \mathrm{tr}((U^T U)(V V^T)).$$

Пользуясь этим, вычисление данной нормы можно осуществлять за $O((m+n)r^2)$ операций. Тем не менее, вычисление нормы и таким образом слишком затратно, так как приведёт алгоритм к сложности

$O((m+n)r^3)$. Воспользуемся тем, что матрица является одноранговым обновлением матрицы, норма которой нам уже известна:

$$\| \begin{bmatrix} U u \end{bmatrix} \begin{bmatrix} V \\ v \end{bmatrix} \|_F^2 = \|UV\|_F^2 + (U^T u, Vv^T) + (Vv^T, U^T u) + \|u\|_2^2 \|v\|_2^2$$

Таким образом пересчёт нормы можно осуществлять за $O((m+n)r)$. Общая сложность представленного алгоритма – $O((m+n)r^2)$ операций и $O((m+n)r)$ вычислений функции.

Упражнения:

1. Реализуйте матричный крестовый метод с $k = 1$ программно на любом удобном языке программирования.
2. Для матрицы $A(i,j) = i^{1/3}j^{-1/3} + i^{-1/3}j^{1/3}$ размеров 8192×8192 исследуйте время построения её точного разложения ранга 2 при помощи сингулярного разложения, рандомизированного сингулярного разложения и матричного крестового метода.
3. Постройте численные аппроксимации матрицы $A(i,j) = i^{1/3}j^{-1/3} + i^{-1/3}j^{1/3}$ размеров 8192×8192, используя критерии для остановки $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}$. Какие при этом получаются ранги? Сравните ранги после применения процедуры крестовой аппроксимации с рангами сингулярного разложения при таком же критерии усечения.

Литература:

1. *Тыртышников Е.Е.* Матричный анализ и линейная алгебра. М.: Физматлит, 2007. 480 с.
2. *Тыртышников Е.Е.* Методы численного анализа. М.: Издательский центр “Академия”, 2007. 320 с.
3. Желтков Д. А., Тыртышников Е. Е. Параллельная реализация матричного крестового метода //вычислительные методы и программирование. – 2015. – Т. 16. – С. 369-375.
4. Goreinov S. A., Tyrtyshnikov E. E., Zamarashkin N. L. A theory of pseudoskeleton approximations //Linear algebra and its applications. – 1997. – Т. 261. – №. 1-3. – С. 1-21.
5. Welch W. J. Algorithmic complexity: three NP-hard problems in computational statistics // Journal of Statistical Computation and Simulation. – 1982. – Т. 15. – №. 1. – С. 17-25.
6. Goreinov S. A. et al. How to find a good submatrix //Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub. – 2010. – С. 247-256.
7. Савостьянов Д. В. Быстрая полилинейная аппроксимация матриц и интегральные уравнения. // Кандидатская диссертация – 2006.
8. *Halko N.*, *Martinsson, P.-G.*, *Tropp, J. A.* Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2), 217–288 2011.
9. *Nakatsukasa Y.* Fast and stable randomized low-rank matrix approximation //arXiv preprint arXiv:2009.11392. – 2020.
10. Веб-страница автора Gregory Gundersen, https://gregorygundersen.com/blog/2019/01/17/randomized-svd/

5.2 Примеры применения матриц малого ранга при вычислениях.

5.2.1 Простейший пример: сжатие матрицы и ускорение её умножения на вектор.

Малоранговая структура матрицы $A \in \mathbb{R}^{M\times N}$ позволяет уменьшить сложность важнейшей операции линейной алгебры – вычисления умножения матрицы на вектор Ax. Действительно,

$$Ax = U_R \Sigma_R V_R x = U_R(\Sigma_R(V_R x))$$

. Таким образом, необходимо выполнить:

1. $y = V_r x, V_r \in \mathbb{R}^{R\times N}$ – потребует $O(NR)$ действий.
2. $z = \Sigma_r y$ – потребует $O(R)$ действий, так как Σ_r – диагональная матрица.
3. $v = U_r z = Ax, U_r \in \mathbb{R}^{N\times R}$ – потребует $O(M \cdot R)$ действий.

В конце концов, получаем $O((M + N + 1)R)$ действий для вычисления $A \cdot x$ вместо классических $O(M\cdot N)$. Если $R \ll M$, $R \ll N$, то выполнение операции матрично-векторного умножения существенно ускоряется.

Упражнения:

1. На примере матрицы Гильберта реализуйте эффективную реализацию операции умножения её аппроксимации ранга $R = 20$ на случайный вектор. Сравните время исполнения этого алгоритма с классическим умножением полной матрицы на такой же вектор.
2. С использованием языка `Python` реализуйте процедуру получения усечённого сингулярного разложения с адаптацией ранга по значению целевой точности приближения ε в спектральной и Фробениусовской нормах.
3. Загрузите в память любое черно-белое изображение в форме массива для модуля `numpy` и проверьте его SVD-сжатие, меняя ранг усечения.
4. Реализуйте эффективное ε-усечение по точности на основе SVD для произвольного скелетного малорангового разложения матрицы $A = U \cdot V$ без построения полной исходной матрицы A.
5. Реализуйте эффективную операцию вычисления всех элементов малоранговой матрицы.

5.2.2 Быстрые вычисления для решения уравнения коагуляции.

Рассмотрим классические уравнения коагуляции Смолуховского [3-4]:

$$\frac{dc_k}{dt} = \frac{1}{2}\sum_{i=1}^{k-1} K_{i,k-i}c_i c_{k-i} - c_k \sum_{i=1}^{\infty} K_{k,i}c_i$$

Для того, чтобы считать задачу поставленной, зададим также типичные для области применения этих уравнений начальные условия:

$$c_1(t = 0) = 1,$$
$$c_k(t = 0) = 0, \quad k > 1.$$

Предположим, что матрица коэффициентов является малоранговой, то есть $K_{i,j} = \sum_{\alpha=1}^{R} U_\alpha(i)V_\alpha(j)$.

Тогда известен алгоритм вычисления правой части при использовании N уравнений за $O(RN\log N)$ действий вместо исходных $O(N^2)$ операций.

Рассмотрим дискретную функцию

$$f_1(k) = \sum_{i+j=k} K_{i,j}c_ic_j \equiv \sum_{i=1}^{k-1} K_{i,k-i}c_ic_{k-i}, k = 1,\dots,N\,.$$

С помощью разложения для $K_{i,j}$ функцию $f_1(k)$ можно представить в виде

$$\begin{aligned} f_1(k) = \sum_{i=1}^{k-1} K_{i,j}c_ic_{k-i} &\approx \sum_{i=1}^{k-1}\sum_{\alpha=1}^{R} U_\alpha(i)V_\alpha(k-i)c_ic_{k-i} \\ &= \sum_{i=1}^{k-1}\sum_{\alpha=1}^{R} \widehat{U_\alpha}(i)\widehat{V_\alpha}(k-i), \\ \text{где}\quad \widehat{U_\alpha}(i) \equiv U_\alpha(i)c_i,&\quad \widehat{V_\alpha}(i) \equiv V_\alpha(i)c_i. \end{aligned}$$

Таким образом, мы имеем сумму из R нижнетреугольных дискретных свёрток массивов $\widehat{U_\alpha}(i)$ и $\widehat{V_\alpha}(j)$, каждую из которых можно вычислить за $O(N\log N)$ арифметических операций одновременно при всех значениях $k = 1,\dots,N$ с применением быстрого преобразования Фурье (это обсуждается более детально в книге В. Н. Чугунова [5]). В результате, сложность вычисления первой суммы при $k = 1,\dots,N$ снизится с $O(N^2)$ до $O(RN\log N)$ арифметических операций.

В то же время, на уровне матрично-векторных операций вычисление каждого из слагаемых в конечной сумме для $f_1(k)$ требует решения следующей задачи:

$$g(k) = \sum_{i=0}^{k} a_ib_{k-i}, k = \overline{0, M-1}.$$

Вычисление $g(k)$ равносильно задаче умножения треугольной тёплицевой матрицы

$$T(a) = \begin{bmatrix} a(\nu_0) & 0 & 0 & \dots & 0 \\ a(\nu_1) & a(\nu_0) & 0 & 0 & 0 \\ a(\nu_2) & a(\nu_1) & a(\nu_0) & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a(\nu_N) & a(\nu_{N-1}) & \dots & \dots & a(\nu_0) \end{bmatrix}$$

на вектор значений b. Заметим, что матрицу $T(a)$ можно представить в виде суммы обычной и "косой" циркулянтных матриц:

$$\begin{aligned} &\frac{1}{2}\begin{bmatrix} a(\nu_0) & a(\nu_N) & a(\nu_{N-1}) & \dots & a(\nu_1) \\ a(\nu_1) & a(\nu_0) & a(\nu_N) & \dots & a(\nu_2) \\ \dots & \dots & \dots & \dots & \dots \\ a(\nu_N) & a(\nu_{N-1}) & \dots & \dots & a(\nu_0) \end{bmatrix} \\ +\frac{1}{2}&\begin{bmatrix} a(\nu_0) & -a(\nu_N) & -a(\nu_{N-1}) & \dots & -a(\nu_1) \\ a(\nu_1) & a(\nu_0) & -a(\nu_N) & \dots & -a(\nu_2) \\ \dots & \dots & \dots & \dots & \dots \\ a(\nu_N) & a(\nu_{N-1}) & \dots & \dots & a(\nu_0) \end{bmatrix} \\ &= \frac{1}{2}(C^+ + C^-) = T(a). \end{aligned}$$

Следовательно, $T(a)b = \frac{1}{2}(C^{+}b + C^{-}b)$. Быстрое умножение каждого из циркулянтов на вектор основывается на использовании быстрого дискретного преобразования Фурье (на языке Python данная операция доступна, например в библиотеке numpy в модуле fft). Подробно изучить утверждения о тёплицевых и циркулянтных матрицах на русском языке можно в монографии В. Н. Чугунова [5] из списка литературы в конце данного раздела.

Далее рассмотрим дискретную функцию

$$f_2(k) = \sum_{i=1}^{N} K_{i,k} c_i, k = 1, \ldots, N .$$

Вычисление значений $f_2(k)$ при $k = 1, \ldots, N$ напрямую потребует $O(N^2)$ арифметических операций. Заметим, что эта операция эквивалентна операции умножения матрицы K на вектор значений c_i. Таким образом, сложность выполнения этой операции тоже снижается до $O(NR)$ операций. В результате, сложность вычислений правой части для уравнений Смолуховского снижается до $O(NR \log N)$ действий с исходных $O(N^2)$, где R – ранг матрицы коэффициентов коагуляции K, для которой обычно известна функция вычисления её элементов.

Далее продемострируем небольшую программу с реализацией вычислений по данной схеме, предполагая, что непосредственно скелетное разложение заранее известно:

```
# Operator for the Smoluchowski coagulation equation
def SmolOper(c_k, Uc, Vc):
  hat_U = np.zeros(Uc.shape)
  hat_V = np.zeros(Vc.shape)
  for i in range(Uc.shape[1]):
    hat_U[:,i] = Uc[:,i] * c_k
    hat_V[:,i] = Vc[:,i] * c_k
  # technical zeros for the convolution operator
  zzeros = np.zeros([Uc.shape[0]-1, Uc.shape[1]])
  hat_U = np.concatenate((hat_U, zzeros))
  hat_V = np.concatenate((hat_V, zzeros))
  # vector for accumulation of the convolution within the rank summation
  conv_part = np.zeros(2*Uc.shape[0]-1)
  # compute the convolutions and accumulate the sums
  for i in range(Uc.shape[1]):
    conv_part=conv_part+np.fft.ifft(np.fft.fft(hat_U[:,i])*np.fft.fft(hat_V[:,i]))
  # truncate and roll the result in order to get original concentrations
  conv_part = np.abs(conv_part[:Uc.shape[0]])
  conv_part = np.roll(conv_part, 1)
  conv_part[0] = 0.0
  #print "conv_part", conv_part[:10]
  # compute the matvec part of the Smoluchowski operator
  matvec_part = Uc.dot( Vc.T.dot(c_k))
  #print "matvec_part", matvec_part[:10]
  # return the final result of the Smoluchowski operator
  return 0.5 * conv_part - c_k * matvec_part
```

Более подробное изложение этого алгоритма можно найти в работах [1-2] из списка литературы данного раздела. Для построения малорангового разложения матрицы коэффициентов (которая обычно задаётся конкретной формулой) можно использовать алгоритмы из предыдущего раздела:

1. усеченное сингулярное разложение (в случае относительно небольших матриц),
2. рандомизированное сингулярное разлоежение (если заранее есть гипотеза о значении ранга),

3. алгоритм адаптивной крестовой аппроксимации (если матрица большая).

Далее можно привести три классических примера:

1. $K_{i,j} \equiv 1$ – постоянная матрица с известным значением ранга, равным единице.
2. $K_{i,j} = \left(\frac{i}{j}\right)^{1/3} + \left(\frac{j}{i}\right)^{1/3} + 2$ – Броуновское ядро коагуляции с известным значением ранга, равным трём.
3. $K_{i,j} = (i^{1/3} + j^{1/3})^2 \cdot \sqrt{\frac{1}{i} + \frac{1}{j}}$ – баллистическое (или свободно-молекулярное) ядро, являющееся формально матрицей полного ранга, но с известной логарифмической асимптотической зависимостью ранга от точности приближений.

Численно решать уравнения Смолуховского можно при помощи простейшей схемы интегрирования по времени

$$\frac{c_k(t + \Delta t) - c_k(t)}{\Delta t} = \frac{1}{2} \sum_{i=1}^{k-1} K_{i,k-i} c_i(t) c_{k-1}(t) - c_k(t) \sum_{i=1}^{\infty} K_{k,i} c_i(t),$$

Следовательно,

$$c_k(t + \Delta t) = c_k(t) + \Delta t \left(\frac{1}{2} \sum_{i=1}^{k-1} K_{i,k-i} c_i(t) c_{k-1}(t) - c_k(t) \sum_{i=1}^{\infty} K_{k,i} c_i(t) \right).$$

В обозначениях языка Python мы проводим вызов функции `SmolOper` с аргументом c_k и используем заранее разложенное ядро коагуляции для реализации эффективных вычислений. Таким образом, получаем

$$c_{new} = c_{old} + \Delta t \cdot \texttt{SmolOper}(c_{old})$$

и можем продолжать шаги по времени до тех пор, пока не достигнем целевого финального момента времени t_{finish}.

Упражнения:

1. Докажите, что для монодисперсных начальных условий и постоянного ядра $\frac{dc}{dt} = \frac{2}{2+t}$ (полная плотность убывает), $\frac{dm}{dt} = 0$(полная масса похраняется), где $c(t) = \sum\limits_{k=1}^{\infty} c_k(t)$, $m(t) = \sum\limits_{k=1}^{\infty} k c_k(t)$.
2. (*) Для $K_{i,j} = 1$ и монодисперсных начальных условий найдите аналитическое решение уравнений Смолуховского.
3. Для $K_{i,j} = i \cdot j$ закон сохранения массы не выполняется. Используя литературу, найдите необходимые требования к ядру коагуляции, при которых закон сохранения массы справедлив. См., например, книги:
   (a) Крапивский П., Реднер С., Бен-Наим Э. Кинетический взгляд на статистическую физику // (Пер. с анг.) Москва: Научный мир. – 2012.
   (b) Галкин В. А. Уравнение Смолуховского. М.: Физматлит // Москва. – 2002.
4. Реализуйте явную схему интегрирования по времени Рунге-Кутты второго порядка для той же задачи. Наблюдается ли рост точности вычислений?
5. Реализуйте явную схему интегрирования по времени Рунге-Кутты четвёртого порядка для той же задачи. Наблюдается ли рост точности вычислений?

6. Реализуйте схему адаптивного выбора шага для упомянутых выше явных методов Рунге-Кутты.

7. Почему для данной задачи сложно реализовать неявную схему интегрирования по времени?

8. Реализуйте процедуру вычисления оператора Смолуховского без использования малоранговой структуры ядра.

9. Сравните времена, необходимые для проведения вычислений при помощи обычного алгоритма численного решения уравнений Смолуховского и с использованием малорангового разложения ядра коагуляции.

Литература:

1. *Матвеев С. А.*, *Тыртышников Е. Е.*, *Смирнов А. П.*, *Бриллиантов Н. В.* Быстрый метод решения уравнений агрегационно-фрагментационной кинетики типа уравнений Смолуховского //Вычислительные методы и программирование. – 2014. – Т. 15. – №. 1. – С. 1-8.

2. *Matveev S. A.*, *Smirnov A. P.*, *Tyrtyshnikov E. E.* A fast numerical method for the Cauchy problem for the Smoluchowski equation //Journal of Computational Physics. – 2015. – Т. 282. – С. 23-32.

3. *Крапивский П. Л.*, *Реднер С.*, *Бен-Наим Э.* Кинетический взгляд на статистическую физику //Пер. с анг.) Москва: Научный мир. – 2012.

4. *Галкин В. А.* Уравнение Смолуховского. М.: Физматлит // Москва. – 2002.

5. *Чугунов В. Н.*, Нормальные и перестановочные теплицевы и ганкелевы матрицы, Москва, Наука, 2017.

5.3 Минимальные необходимые сведения о тензорных разложениях.

Рассмотрим трехмерный массив

$$A(i,j,k) \in \mathbb{R}^{N\times N\times N}$$

далее будем называть многомерные массивы тензорами. Для его сохранения в памяти потребуется N^3 ячеек памяти. Иногда данные обладают структурой, позволяющей записать их малопараметрическое представление, требующее меньше ячеек в памяти компьютера.

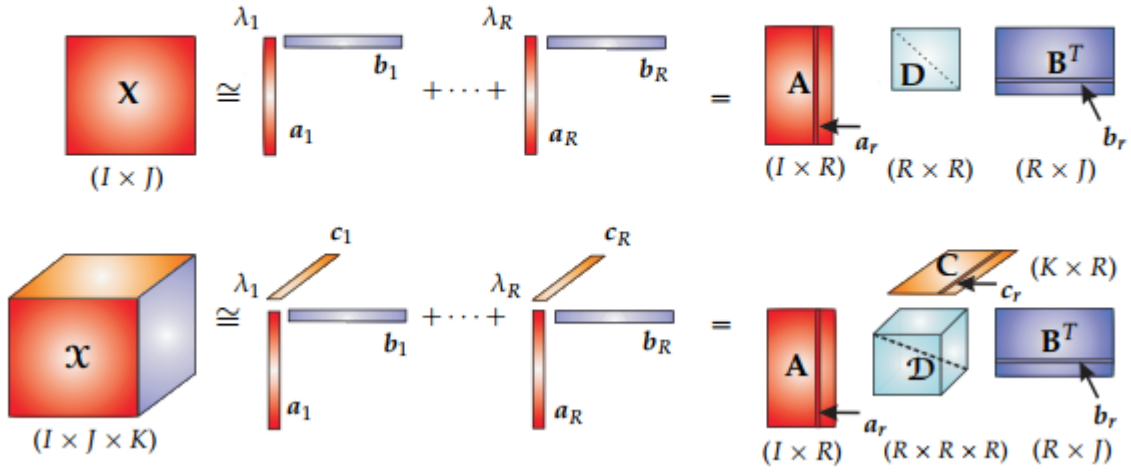
Например, можно показать, что любой трехмерный массив представим в виде одного из трёх базовых тензорных разложений (известно, впрочем, больше способов, но это выходит за рамки программы нашего курса): канонический полилинейный формат, формат Таккера и тензорный поезд.

5.3.1 Канонический формат.

Каонический полилинейный формат (в англоязычных источниках его называют canonical-polyadic format или CP-format) является естественным обобщением формулы матричного скелетного разложения. В данном формате для вычисления произвольного элемента тензора A можно записать следующую формулу:

$$A(i,j,k) = \sum_{\alpha=1}^{R} u_\alpha(i)v_\alpha(j)w_\alpha(k) = \sum_{\alpha=1}^{R} u(\alpha,i)v(\alpha,j)w(\alpha,k),$$

которая позволяет нам представить массив $N \times N \times N$ при помощи $3NR$ параметров. Очевидно, что в предположении $R \ll N$ можно наблюдать потрясающее сжатие исходных данных. Ниже мы приводим картинку из учебника А. Чихоцкого и соавторов (см. ниже в списке литературы), иллюстрирующую наглядно, как работает представленная формула; верхняя из двух картинок иллюстрирует матричное скелетное разложение, а нижняя представляет каноническое полилинейное разложение:



Минимальное число слагаемых R, при котором в уравнении наблюдается точное равенство называется каноническим тензором рангом тензора A и обозначается $\mathrm{trank}\, A$. Важно отметить, что любой тензор A возможно представить при помощи тривиального (и вычислительно бесполезного) канонического

разложения в виде разложения в одноранговые тензоры по всем ненулевым элементам:

$$\mathbf{A} = \sum_{A(i,j,k)\neq 0} e_j \otimes e_j \otimes e_k \cdot A(i,j,k).$$

Хотя такое представление тензора не приводит к сжатию данных, оно мгновенно доказывает существование CP-разложения конечной длины для произвольного тензора. Значит, и канонический тензорный ранг – величина конечная и осмысленная для любого тензора. Отметим, что знак $\otimes$ соответствует классической операции кронекеровского произведения. Минимальные необходимые сведения об этой операции и её свойствах есть в разделе 1.11 источника [3] в конце данного раздела, ознакомиться с ними *очень* рекомендуется читателям. Векторы e_i соответствуют каноническим векторам, единственный ненулевой элемент которых равен единице и находится в позиции i.

Существует большое количество тензоров и функционально-порожденных массивов, обладающих малоранговыми каноническими разложениями. Например, в случае $A(i,j,k) = \sin(i+j+k)$ достаточно лишь открыть скобки, следуя базовым свойствам тригонометрических функий sin, cos, и получить шесть слагаемых. Таким образом, можно увидеть, что канонический ранг этого тензора всегда не превосходит шести. Чуть больше усилий потребуется, если читатель сможет вспомнить запись функции с использованием комплексных чисел

$$\sin(i+j+k) = \frac{e^{I\cdot(i+j+k)} - e^{-I\cdot(i+j+k)}}{2I} = \frac{1}{2I}\left(e^{I\cdot i}\cdot e^{I\cdot j}\cdot e^{I\cdot k} - e^{-I\cdot i}\cdot e^{-I\cdot j}\cdot e^{-I\cdot k}\right),$$

где I соответствует мнимой единице. В таком случае, число слагаемых для таких функционально-порождённых тензоров сократится до двух. Из нашего простого примера видно также, что для разных числовых полей ($\mathbb{C}$, $\mathbb{R}$ и др.), как значения канонического ранга, так и сами разложения для "одного и того же" тензора могут отличаться.

Основным недостатком CP-формата является огромная трудность вычисления точного значения канонического ранга $R = \mathrm{trank}\ A$, которая, как правило, приводит к необходимости решать NP-трудные или NP-полные задачи. На практике обычно фиксируют целевое знаение ранга R и используют метод переменных наименьших квадратов, чтобы получить приближённое каноническое разложение.

Algorithm 5.3: Схема процедуры переменных наименьших квадратов(A, R)

$U(\alpha,i), V(\alpha,j), W(\alpha,k) \leftarrow$ инициализация случайными значениями
repeat
1. Зафиксируем значения $V(\alpha,j), W(\alpha,k)$,
$U(\alpha,i) \leftarrow$ решим задачу линейных наименьших квадратов $\| \sum_{\alpha=1}^{R} u(\alpha,i)v(\alpha,j)w(\alpha,k) - A\|_F$.
2. Зафиксируем значения $U(\alpha,i), W(\alpha,k)$,
$V(\alpha,j) \leftarrow$ решим задачу линейных наименьших квадратов $\| \sum_{\alpha=1}^{R} u(\alpha,i)v(\alpha,j)w(\alpha,k) - A\|_F$.
3. Зафиксируем значения $U(\alpha,i), V(\alpha,j)$,
$W(\alpha,k) \leftarrow$ решим задачу линейных наименьших квадратов $\| \sum_{\alpha=1}^{R} u(\alpha,i)v(\alpha,j)w(\alpha,k) - A\|_F$.
until stoppingcriteria

Можно использовать самые разные критерии остановки в данном итерационном алгоритме, в частности, часто используется точность получаемого приближения по норме Фробениуса и достижение максимального допустимого количества итераций (что, фактически, эквивалентно ограничению сверху на допустимое время вычислений). К сожалению, про алгоритм переменных наименьших квадратов нет доказанных теорем, гарантирующих его сходимость в общем случае, что не мешает его широкому применению на практике.

Несмотря на трудности вычисления точного значения канонического ранга, использование канонического разложения крайне полезно для построения эффективных алгоритмов умножения матриц. Запишем этот хорошо известный алгоритм:

$$A \in R^{n_1 \times n_2}, B \in R^{n_2 \times n_3}; \quad C = AB$$

$$C_{i,j} = \sum_{k=1}^{n_2} A_{i,k} B_{k,j} \quad \{\text{запишем более формально}\}$$

$$= \sum_{k,l} \sum_{m,n} T_{ij,kl,mn} A_{k,l} B_{m,n}$$

{элементы тензора T равны 0 или 1, всего в нем $n_1 \cdot n_2 \cdot n_3$ ненулевых элементов}

$$= \sum_{\alpha=1}^{R} u_1(\alpha, \overline{i,j}) \left(\sum_{\overline{k,l}} u_2(\alpha, \overline{k,l}) A_{k,l} \right) \cdot \left(\sum_{\overline{m,n}} u_3(\alpha, \overline{m.n}) B_{m,n} \right).$$

Если $R < n_1 n_2 n_3$, тогда можно наблюдать сокращение числа умножений. Тензор T в точности соответствует алгоритму умножения матриц, а его тензорный ранг минимальному возможному количеству операций умножения элементов матриц A и B. Если $n_1 = n_2 = n_3 = 2$, то известно, что $R = 7$ и это приводит к знаменитому алгоритму Штрассена (отметим, что в классической реализации потребовалось бы 8 умножений). Один из наиболее впечатляющих недавних успехов в области исследования формальной алгоритмический сложности операции умножения матриц можно найти в источнике [2] в списке литературы в конце данного раздела. Задачу поиска рангов тензоров умножения матриц разных размеров авторы данной работы решали при помощи комбинации алгоритма переменных наименьших квадратов, машинного обучения и целого ряда эвристических наблюдений.

Более того, даже зная точное или целевое значение канонического ранга (для точного или приближённого представления исходного тензора), пользователю будет необходимо решить задачу заполнения факторов разложения $u_\alpha(i)v_\alpha(j)w_\alpha(k)$, что является не менее трудной задачей. Обычно для решения этой задачи используются разнообразные реализации итерационного алгоритма чередующихся наименьших квадратов, которые часто приводят к результату, но могут обладать существенными проблемами в скорости сходимости и обосновании конечных получаемых результатов. Таким образом, сообщество нуждается в получении SVD-подобных робастных процедур сжатия тензоров.

Использование малоранговых канонических тензорных разложений тех или иных тензоров часто приводит к возможностям построения быстрых вычислительных процедур. Например, наличие малоранговых представлений для тензоров вида $\sin(ai+bj+ck)$ может привести к эффективным процедурам вычисления средних значений, свёрток и тд. Несмотря на важность, канонического формата, мы сконцентрируем основное внимание на разложениях Таккера и тензорного поезда, а заинтересовавшимся читателям для более подробного знакомства с CP-форматом оставим несколько источников.

5.3.2 Формат Таккера.

Формат Таккера соответствует следующей формуле для поэлементного представления тензора:

$$A(i,j,k) = \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \sum_{\alpha_3=1}^{r_3} G(\alpha_1, \alpha_2, \alpha_3) U_1(i, \alpha_1) U_2(j, \alpha_2) U_3(k, \alpha_3).$$

В данном случае необходимо ввести понятие мультилинейных Таккеровских рангов (r_1, r_2, r_3), которые существуют и соотвествуют минимальным значениям r_1, r_2, r_3, при которых наблюдается поэлементное равенство между исходным тензором A и формулой Таккера. В данном случае уже необходимо хранить так называемое ядро Таккера $G(\alpha_1, \alpha_2, \alpha_3) \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ и Таккеровские факторы

$U_1(i,\alpha_1) \in \mathbb{R}^{N\times r_1}$, $U_2(i,\alpha_2) \in \mathbb{R}^{N\times r_2}$, $U_3(i,\alpha_3) \in \mathbb{R}^{N\times r_3}$. Нетрудно видеть, что их хранение потребует $r_1r_2r_3 + N(r_1+r_2+r_3)$ ячеек памяти вместо исходных N^3.

Дополнительное хранение ядра Таккера при помощи $r_1r_2r_3$ ячеек является недостатком такого формата в сравнении с каноническим тензорным разложеним. Однако, величайшим преимуществом этого разложения является наличие алгоритма его построения, в литературе называемого higher order SVD (HOSVD), а также его алгоритмического эквивалента (если существует точное разложение) – последовательно округляемого HOSVD (st-HOSVD). Для обеих процедур можно оценить сложность построения разложения Таккера через $O(N^4)$ операций, но алгоритм st-HOSVD потребует в 2-3 раза меньше действий.

Минимальные значения Таккеровских рангов связаны с рангами специальных матриц развертки, которые формируются в результата операций `reshape` исходного тензора $A(i,j,k)$:

1. $A_1 = A(i,\overline{j,k})$ в Python: `reshape(A, [N, N*N])`, далее SVD потребует $O(N^4)$ действий.
2. $A_2 = A(j,\overline{i,k})$ в Python: `transpose(A, [1,0,2]), reshape(A, [N, N*N])`, далее SVD потребует $O(N^4)$ действий.
3. $A_3 = A(k,\overline{i,j})$ в Python: `transpose(A, [2,0,1]), reshape(A, [N, N*N])`, далее SVD потребует $O(N^4)$ действий.

Значения рангов этих трёх матриц развертки в точности соответствуют и формируют минимальные мультилинейные Таккеровские ранги (r_1,r_2,r_3). Будем считать далее, что $R = \max_i r_i$. Предположим, что $A_i = U_i\Sigma_i V_i$, тогда можно сформировать тензор

$$G(\alpha_1,\alpha_2,\alpha_3) = U_1^\top U_2^\top U_3^\top A.$$

Данная формула – это и есть классический алгоритм HOSVD. Алгоритм stHOSVD получается при последовательной редукции рангов в скелетных разложениях матриц развёрток (например, при помощи SVD) и более аккуратном использовании операций `reshape` и `transpose`. Реализуется данный алгоритм следующим образом:

$$A(i,j,k) = A(i,\overline{j,k}) = \sum_{\alpha_1}^{r_1} U_1(i_1,\alpha_1)\tilde{G}_{23}(\alpha_1,\overline{j,k}) = \sum_{\alpha_1}^{r_1} U_1(i_1,\alpha_1)\tilde{G}_{23}(\alpha_1,\overline{j,k}) =$$
$$\sum_{\alpha_1}^{r_1} U_1(i_1,\alpha_1)\tilde{G}_{23}(j,\overline{\alpha_1,k}) = \sum_{\alpha_1}^{r_1}\sum_{\alpha_2}^{r_2} U_1(i,\alpha_1)U_2(j,\alpha_2)\tilde{G}_3(\alpha_2,\overline{\alpha_1,k}) =$$
$$\sum_{\alpha_1}^{r_1}\sum_{\alpha_2}^{r_2} U_1(i,\alpha_1)U_2(j,\alpha_2)\tilde{G}_3(k,\overline{\alpha_1,\alpha_2}) = \sum_{\alpha_1}^{r_1}\sum_{\alpha_2}^{r_2}\sum_{\alpha_3}^{r_3} U_1(i,\alpha_1)U_2(j,\alpha_2)U_3(k,\alpha_3)\tilde{G}(\alpha_3,\overline{\alpha_1,\alpha_2})$$

Нетрудно видеть, что после финальных операций `reshape` и `transpose` мы получаем искомую формулу разложения Таккера:

$$\sum_{\alpha_1}^{r_1}\sum_{\alpha_2}^{r_2}\sum_{\alpha_3}^{r_3} U_1(i,\alpha_1)U_2(j,\alpha_2)U_3(k,\alpha_3)G(\alpha_1,\alpha_2,\alpha_3).$$

В процессе реализации этого алгоритма нам необходимо было строить скелетные разложения для следующих матриц:

1. $A(i,\overline{j,k})$ – потребует $O(N^2\cdot N^2) = O(N^4)$ действий.
2. $\tilde{G}_{23}(j,\overline{\alpha_1,k})$ – потребует $O(N^2\cdot NR) = O(N^3R)$ действий.

3. $\tilde{G}_3(k, \overline{\alpha_1, \alpha_2})$ – потребует $O(\min(N^2R^2, NR^4))$ действий.

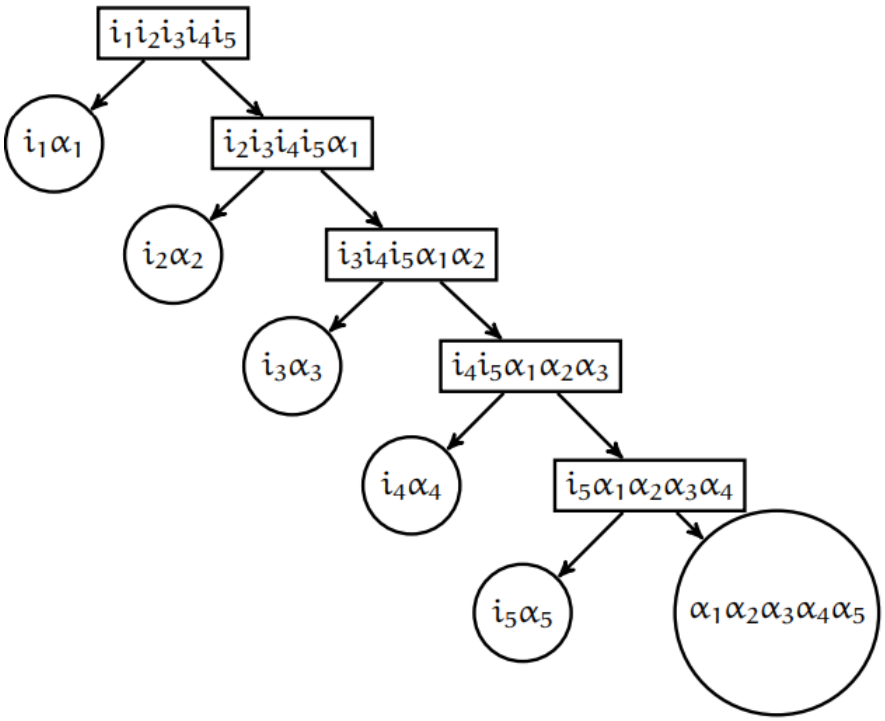
В результате полная сложность алгоритма составит $O(N^4 + N^3R + \min(N^2R^2, NR^4))$ действий вместо исходных $O(3N^4)$, соотвествующих классическому алгоритму HOSVD. Проиллюстрируем данный алгоритм на языке Python (процедура stHOSVD_3d):

```
def stHOSVD_3d(A, r1, r2, r3):
    # Construct Tucker decomposition of 3D tensor via stHOSVD with ranks [r1, r2, r3]
    N1,N2,N3 = A.shape
    A1=A.reshape(N1, N2*N3)
    U, s, V = np.linalg.svd(A1, full_matrices=False)
    V1 = U[:,:r1]
    V1 = V1.T
    A1 = np.diag(s[:r1]) @ V[:r1,:] # r1 x N2*N3
    A1 = A1.reshape(r1, N2, N3)
    A1 = A1.transpose([0,2,1])
    A1 = A1.reshape(r1 * N3, N2)# r1 * N3 x N2
    U, s, V = np.linalg.svd(A1, full_matrices=False)
    V2 =  V[:r2,:] # r2 x N2
    U2 = U[:,:r2] @ np.diag(s[:r2]) # r1*N3 x r2
    U2 = U2.reshape(r1, N3, r2) # r1 x N3 x r2
    U2 = U2.transpose([0,2,1]) # r1 x r2 x N3
    U2 = U2.reshape(r1 * r2, N3) # r1*r2 x N3
    U, s, V = np.linalg.svd(U2, full_matrices=False)
    V3 = V[:r3,:] # U3(r3,j)
    G = U[:,:r3] @ np.diag(s[:r3])# r1 * r2 x r3
    G = G.reshape(r1, r2, r3) # main core
    return G, V1, V2, V3
```

Замечательное свойство разложения Таккера состоит в следующем: использование усеченного сингулярного разложения в алгоритмах HOSVD и stHOSVD позволяет получить *приближенное* разложение Таккера, удовлетвряющее всем необходимым теоремам о существовании и устойчивости. На практике данная теорема в применении к приближенной процедуре HOSVD состоит в следующем:

$$\|A - B_t\|_F \leq \sqrt{d}\|A - B_*\|_F,$$

где B_t – результат усеченного по рангам алгоритма HOSVD или stHOSVD, а B_* – наилучшее приближение с такими же фиксированными рангами Таккера. В случае более высокой размерности, алгоритмы HOSVD и stHOSVD наглядно могут быть описаны с помощью следующей картинки (картинка взята из докторской диссертации И. В. Оселедца):



Окончательное представление d-мерного тензора в формате Таккера соответствует уравнению:

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \ldots \sum_{\alpha_d=1}^{r_d} G(\alpha_1, \alpha_2, \ldots \alpha_d) U_1(i_1, \alpha_1) U_2(i_2, \alpha_2) \ldots U_d(i_d, \alpha_d).$$

Нетрудно заметить, что для хранения ядра разложения Таккера $G(\alpha_1, \alpha_2, \ldots \alpha_d)$ потребуется экспоненциально растущее с размерностью количество $r_1 \cdot r_2 \cdot \ldots \cdot r_d$ ячеек памяти. В таком случае, если для простоты положить все ранги равными $r_i = R$, легко заметить, что для хранения ядра потребуется R^d ячеек, что существенно снижает привлекательность такого способа представления данных в случае их высокой размерности d.

5.3.3 Тензорный поезд.

В случае трехмерных данных тензорный поезд соответствует следующей форме представления массива:

$$A(i, j, k) = \sum_{\alpha_1}^{r_1} \sum_{\alpha_2}^{r_2} G_1(i, \alpha_1) G_2(\alpha_1, j, \alpha_2) G_3(\alpha_2, k).$$

В каком-то смысле мы можем сказать, что это представление является “не вполне законченным” разложением Таккера, потребует $Nr_1r_2 + Nr_1 + Nr_2$ ячеек памяти и вообще не кажется хорошей идеей для сжатия данных. Однако замечательные свойства этого формата проявляются в случае данных высокой размерности, так как он в явной форме не содержит факторов, имеющих экспоненциальные требования по памяти с ростом размерности d, как это мы видели на примере разложения Таккера.

К счастью, данное разложение тоже может быть вычислено при помощи процедуры, основанной на сингулярном разложении. Эта процедура известна под названием TTSVD и записывается следующим

образом в случае трехмерных массивов:

$$A(i,\overline{i,k}) = \sum_{\alpha_1=1}^{r_1} G_1(i,\alpha_1)G_{2,3}(\alpha_1,\overline{j,k}) =$$
$$\sum_{\alpha_1=1}^{r_1} G_1(i,\alpha_1)G_{2,3}(\overline{\alpha_1,j},k) = \sum_{\alpha_1=1}^{r_1}\sum_{\alpha_2=1}^{r_2} G_1(i,\alpha_1)G_2(\alpha_1,j,\alpha_2)G_3(\alpha_2,k).$$

Из проведенных действий ясно, что сложность получения такого разложения составит $O(N^4)$ действий. На языке Python эти действия можно реализовать в виде следующей процедуры:

```
def TTSVD_3d(A, r1, r2):
    N1,N2,N3 = A.shape
    A1=A.reshape(N1, N2*N3)
    U, s, V = np.linalg.svd(A1, full_matrices=False)
    G1 = U[:,:r1] #N1 x r1
    V1 = np.diag(s[:r1]) @ V[:r1, :] # r1 x N2*N3
    #print("V1 shape", V1.shape)
    V1 = V1.reshape(r1 * N2, N3)
    U, s, V = np.linalg.svd(V1, full_matrices=False)
    G2 = U[:,:r2]  # r1*N2 x r2
    G2 = G2.reshape(r1, N2, r2)# r1 x N2 x r2
    G3 = np.diag(s[:r2]) @V[:r2,:] # r2 x N3
    return G1, G2, G3
```

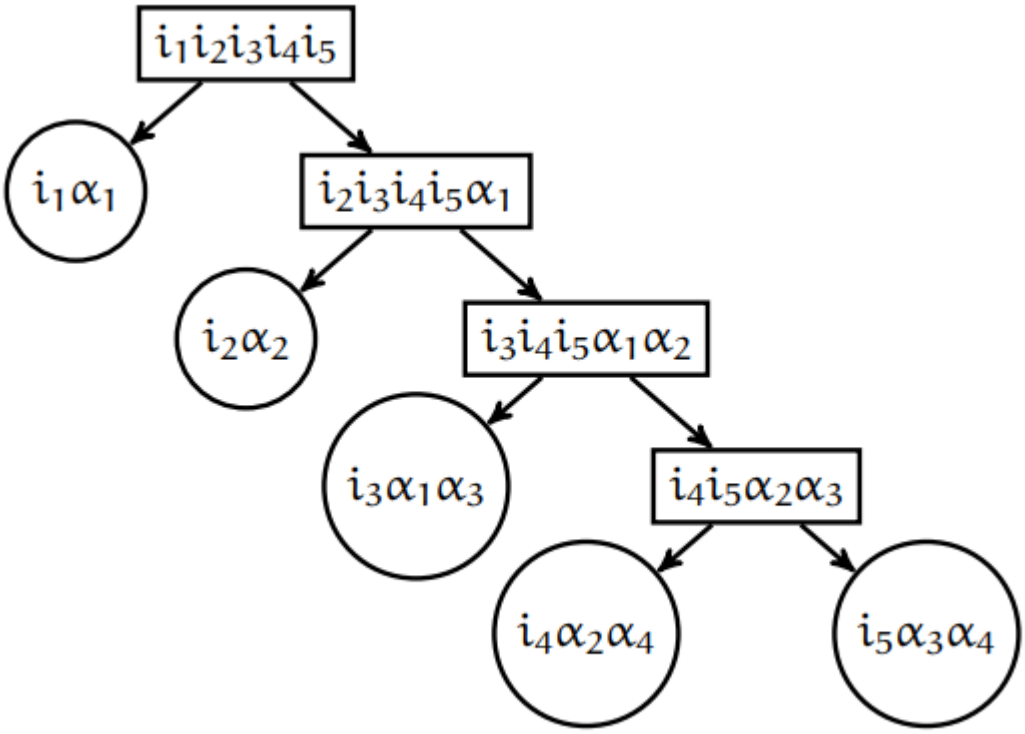
Минимальные возможные значения TT-рангов соответствуют значениям рангов матриц развертки (несколько иных, чем в случае разложения Таккера):

$$A_1 = A(i,\overline{j,k}) \text{ и } A_2 = A(\overline{i,j},k).$$

К счастью, приближенное усечение рангов в процедуре TTSVD имеет аналогичные с HOSVD свойства квазиоптимальности получаемых приближений:

$$\|A - B_t\|_F \leq \sqrt{d}\|A - B_*\|_F,$$

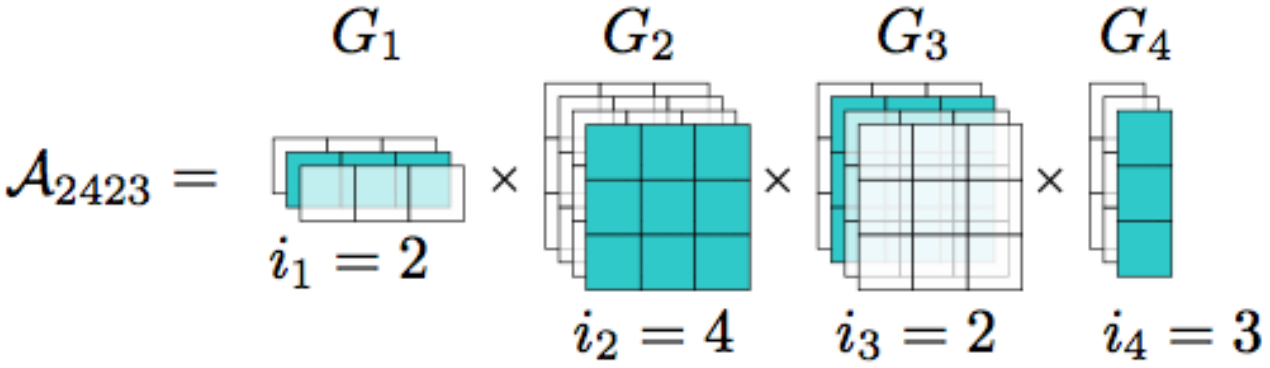
где B_t – результат TTSVD, а B_* – наилучшая возможная аппроксимация в TT-формате с такими же фиксированными рангами. Пример реализации разделения индексов при процедуре TTSVD приведен на картинке ниже (взята из докторской диссертации И. В. Оселедца):



Итоговая запись для тензорного поезда в d-мерном случае выглядит следующим образом:

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_1}^{r_1} \sum_{\alpha_2}^{r_2} \ldots \sum_{\alpha_{d-1}}^{r_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d).$$

Нетрудно показать, что такое представление существует для любого d-мерного массива и в альтернативных источниках литературы (в основном, связанных с исследованиями по теоретической физике) известно под названием *состояния матричного произведения* (matrix product state). Такое название кажется вполне обоснованны, так как вычисление элемента массива $A(i_1, i_2, \ldots, i_d)$, представленного в виде тензорного поезда может быть выполнено через $(d-2)$ операции умножения матрицы на вектор и ещё одно финальное скалярное произведение. Это свойство для тензорных поездов проиллюстрировано ниже на картинке.



Упражнения:

1. Реализуйте оптимизационный алгоритм ALS для тензора алгоритма умножения матриц 2×2 с целевым рангом $R = 7$. Сходится ли данная процедура? Совпадает ли полученное разложение с оригинальным результатом Штрассена?

2. Пусть для тензора $A(i, j, k)$ известно каноническое разложение ранга R. С использованием этого разложения получите алгоритм вычисления среднего значения всех его элементов.

3. Пусть для тензора $A(i, j, k)$ известно разложение Таккера с рангами r_1, r_2, r_3, не большими чем R. С использованием этого разложениея получите алгоритм вычисления среднего значения всех его элементов.

4. Пусть для тензора $A(i, j, k)$ известно разложение тензорный поезд с рангами r_1, r_2, не большими чем R. С использованием этого разложениея получите алгоритм вычисления среднего значения всех его элементов.

5. Пусть для тензора $A(i, j, k) \in \mathbb{R}^{M \times N \times K}$ и матрицы $B(j, k) \in \mathbb{R}^{N \times K}$ известно разложение Таккера и малоранговое скелетное разложение соответственно. Ранги разложений не выше, чем R, оцените сложность вычисления вектора $c(i) = \sum\limits_{j=1}^{N} \sum\limits_{k=1}^{K} A(i, j, k) \cdot B(j, k)$.

6. Пусть для тензора $A(i, j, k) \in \mathbb{R}^{M \times N \times K}$ и матрицы $B(j, k) \in \mathbb{R}^{N \times K}$ известно каноническое разложение и малоранговое скелетное разложение соответственно. Ранги разложений не выше, чем R, оцените сложность вычисления вектора $c(i) = \sum\limits_{j=1}^{N} \sum\limits_{k=1}^{K} A(i, j, k) \cdot B(j, k)$.

7. Пусть для тензора $A(i, j, k) \in \mathbb{R}^{M \times N \times K}$ и матрицы $B(j, k) \in \mathbb{R}^{N \times K}$ известно разложение в тензорный поезд малоранговое скелетное разложение соответственно. Ранги разложений не выше, чем R. Оцените сложность вычисления вектора $c(i) = \sum_{j=1}^{N} \sum\limits_{k=1}^{K} A(i, j, k) \cdot B(j, k)$.

8. Докажите, что сложность алгоритма TTSVD действительно составляет $O(N^4)$ действий.

9. Сформулируйте рандомизированные версии алгоритмов HOSVD, st-HOSVD, TTSVD. Снизится ли сложность полученных процедур?

10. Докажите, что ранги Таккера для функционально-порожденных тензоров $\sin(i + j + k)$ равны 2.

11. Докажите, что TT-ранги для $\sin(i + j + k)$ и $i + j + k$ равны 2.

12. Обобщите программные реализации процедур HOSVD и TTSVD на случай произвольной размерности d.

13. Реализуйте алгоритмы переменного проектирования для получения неотрицательных тензорных разложений, следуя статье: Sultonov A., Matveev S., Budzinskiy S. Low-rank nonnegative tensor approximation via alternating projections and sketching //Computational and Applied Mathematics. – 2023. – Т. 42. – №. 2. – С. 68 (препринт доступен arXiv:2209.02060)..

14. Докажите, что сложность вычисления элемента d-мерного массива, представленного в виде тензорного поезда с максимальным из TT-рангов равным R, составит $O(dR^2)$ действий.

15. Получите эффективные процедуры вычисления поэлементной суммы $A + B$ и произведения $A \cdot B$ для многомерных массивов, представленных в TT-формате.

16. Реализуйте алгоритмы HOSVD и TTSVD с адаптивным по целевой точности выбором рангов.

17. Реализуйте программно рандомизированные версии алгоритмов HOSVD и TTSVD.

Литература:

1. Badeau R., Boyer R. (2008). Fast multilinear singular value decomposition for structured tensors. SIAM Journal on Matrix Analysis and Applications, 30(3), 1008-1021.

2. Fawzi, A., et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 610(7930), 47-53.

3. Тыртышников, Е. Е.. Матрицы, тензоры, вычисления. М.: МГУ им. М. В. Ломоносова, 2013.

4. Oseledets I. V., Tyrtyshnikov E. E. (2009). Breaking the curse of dimensionality, or how to use SVD in many dimensions. SIAM Journal on Scientific Computing, 31(5), 3744-3759.

5. Zhang C., Jeckelmann E., White S. R. (1998). Density matrix approach to local Hilbert space reduction. Physical review letters, 80(12), 2661.

6. Оселедец, И. В. Вычислительные тензорные методы и их применения. *докторская диссертация*, М.: ИВМ РАН, 2012.

7. Sultonov A., Matveev S., Budzinskiy S. Low-rank nonnegative tensor approximation via alternating projections and sketching //Computational and Applied Mathematics. – 2023. – Т. 42. – №. 2. – С. 68 (arXiv:2209.02060).

8. Cichocki A. et al. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1 //arXiv preprint arXiv:1609.00893. – 2016.

9. Cichocki A. et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives //Foundations and Trends® in Machine Learning. – 2017. – Т. 9. – №. 6. – С. 431-673.

5.4 Применения матричных методов в задачах теории графов.

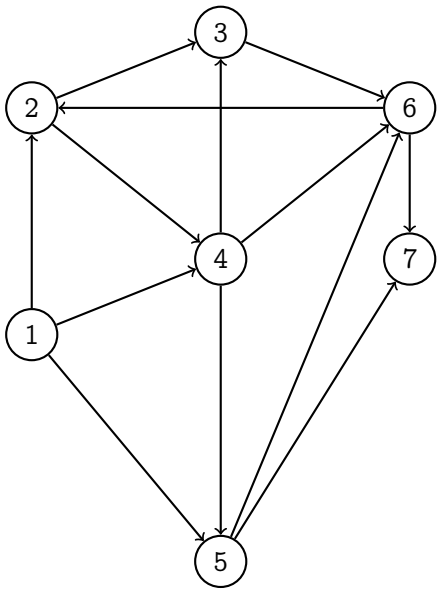
Существует огромное количество применений линейной алгебры в задачах, связанных с анализом данных, в частности, представленных в виде графов. Теория графов – хорошо развитый раздел математики, познакомиться с которым можно при помощи учебника Фрэнка Харрари [1]. Под графом мы понимаем набор вершин V_i и связывающих их ребёр $E_{i,j}$. В случае взвешенного графа с каждым ребром ассоциируется его вес $w_{i,j} \geq 0$. Совершенно естественным образом любому граму (взвешенному или обычному) ставится в соответствие его матрица связности A, для которой

$$A_{i,j} = w_{i,j}, \qquad \text{если вес ребра к вершине } V_i \text{ от вершины } V_j \text{ равен } w_{i,j}.$$

В случае обыкновенного графа, матрица связности состоит из 0 и 1,

$$A_{i,j} = \begin{cases} 1, & \text{если существует ребро к вершине } V_i \text{ от вершины } V_j, \\ 0, & \text{иначе.} \end{cases}$$

Рассмотрим в качестве примера следующий граф



Его матрица связности будет следующей (с учётом упорядочения вершин)

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Если бы рёбрам этого графа соответствовали некоторые веса, то элементы матрицы $a_{i,j}$ соответствовали бы значениям этих весов.

Легко отметить, что для ненаправленного графа матрица связности обязательно будет симметричной. В случае матрицы связности ненаправленного графа без весов на рёбрах можно доказать, что элементы матриц $\{A^m\}_{i,j}$ равны в точности количеству маршрутов длины m из вершины V_i к вершине

V_j. В случае сильно связного графа (то есть графа, в котором существует путь от любой вершины к другой) можно доказать, что матрица связности будет неприводимой по отношению к перестановкам (то есть она не может быть представлена в блочном виде $\begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix}$ при помощи преобразования вида $P^\top AP$, где P – матрица перестановки, а блоки A_1, A_3 – квадратные).

Сконцентрируемся для начала на обыкновенных графах. Во-первых, матрица связности графа G с N вершинами будет квадратной порядка N. Так как эта матрица является квадратной, она обладает N собственными значениями λ_i. Во-вторых, для изоморфных ему графов $\hat{G}$ (отличающихся лишь переупорядочением вершин) матрицы связности будут связаны соотношением подобия

$$A_G = PA_{\hat{G}}P^\top,$$

где P – матрица перестановки (она ортогональная), возникающая при переупорядочении вершин от графа G к графу $\tilde{G}$. Так как матрицы A_G и $A_{\hat{G}}$ подобны, у них одинаковые наборы собственных значений. В-третьих, эти матрицы состоят из неотрицательных элементов. Из этого на основании теоремы Перрона-Фробениуса можно заключить, что, как минимум, максимальное по модулю собственное значение для любой матрицы связности тоже является вещественным неотрицательным числом, а соответствующий ему собственный вектор состоит из вещественных неотрицательных чисел. Теорема Перрона-Фробениуса состоит в следующем:

Теорема. Пусть A – квадратная матрица со строго положительными элементами (или неприводимая неотрицательная). Тогда

- наибольшее по модулю собственное значение $\lambda \in R$, $\lambda > 0$,
- наибольшее по модулю собственное значение является простым корнем характеристического многочлена,
- соответствующий ему собственной вектор имеет вещественные строго положительные координаты
- это собственное значение удовлетворяет неравенствам $\min_i \sum_j a_{ij} \le \lambda \le \max_i \sum_j a_{ij}$.

Последние неравенства можно записать и в столбцовом варианте $\min_j \sum_i a_{ij} \le \lambda \le \max_j \sum_i a_{ij}$ (так как характеристические многочлены матриц A и $A^\top$ совпадают). Таким образом, для всех изоморфных графов старшая собственная пара является инвариантом, который обладает весьма важным для приложений смыслом из-за неотрицательности элементов старшего собственного вектора ν.

Прикладной смысл в этот собственный вектор можно вложить (догадались об этом, видимо, Сергей Брин и Лоуренс Пейдж в работа [2]), если провести его нормировку в норме $\hat{\nu} = \nu/\|\nu\|_1$ (напомним, что $\|\nu\|_1 = \sum_{i=1}^{N} |\nu_i|$). В таком случае, сумма элементов (из-за их знакопостоянности) вектора $\hat{\nu}$ станет равна единице, а значениям отдельных $\hat{\nu}_i$ можно придать смысл *важности* отдельных вершин в графе. Понятие *важности* страниц возникает при рассмотрении следующей модели случайного блуждания *пользователя* по графу, вершины которого представляет собой интернет-страницы (или научные документы), а рёбра – гиперссылки между этими страницами.

Предположим, что на странице V_j размещено l_j исходящих ссылок. Если одна из этих ссылок ведет на страницу V_i, то V_j передаст $1/l_j$ своей *важности* странице V_i. Этот механизм возникает из гипотезы, что модельный пользователь случайно равновероятно переходит по одной из исходящих ссылок из текущей вершины в следующую и продолжает такое блуждание. В таком случае *важность* страницы соответствует средней доле *попаданий* пользователя на эту страницу в процессе случайных

переходов. Тогда уровень важности (в оригинальной работе $PR(V_i)$) страницы V_i – это сумма всех таких значений со всех входящих ссылок. Если представить набор страниц, ссылающихся на страницу V_i, как B_i, то *важность* V_i вычисляется по следующей формуле

$$PR(V_i) = \sum_{V_j \in B_i} \frac{PR(V_j)}{l_j}.$$

Таким образом для значений вектора $\vec{p}_i = PR(V_i)$ мы получаем систему линейных алгебраических уравнений, коэффициенты которой формируются при помощи матрицы связности исходного графа, для столбцов которых применяется нормировка по норме $\|\cdot\|_1$ (чтобы сумма значений элементов по столбцам была равна единице). Обозначим эту матрицу $\hat{A}$. Тогда уравнение на поиск вектора значений $\vec{p}$ является задачей о поиске собственного вектора матрицы $\hat{A}$, соответствующего собственному значению $\lambda = 1$.

Матрица $\hat{A}$ состоит из неотрицательных элементов (предположим, что она соответствует графу, в котором существует путь между любой парой вершин), следовательно, пользуясь теоремой Перрона-Фробенуиуса можно заключить, что её максимальное собственное значение $\lambda = 1$ является простым, а единственный вектор значений авторитетностей страниц $\vec{p}$ вещественным знакопостоянным (не ограничивая общности будем считать его поэлементно положительным). Таким образом, решая задачу

$$A \cdot \vec{p} = \vec{p},$$

получаем прикладной результат. На практике графы не всегда оказываются сильно связными и могут возникнуть сложности при формальном вычислении вектора *важности* вершин. Первая трудность возникает в случае графов, с *висячими* (на нашей картинке это вершина с номером 1) или *тупиковымы* вершинами (вершина номер 7). То есть с такими вершинами, для которых нет входящих или исходящих рёбер, значит им будут соответствовать нулевые строки (для *висячих*) вершин или столбцы (для *тупиковых*). В случае тупиковых вершин, невозможно провести нормировку значений элементов соответствующих им столбцов при построении матрицы $\hat{A}$, поэтому оставим их нулевыми. Авторитетность проблемных *висячих* страниц в любом случае окажется нулевой, а граф не будет сильно-связным, поэтому теоремой Перрона-Фробениуса воспользоваться нельзя.

Для решения этих трудностей обычно пользуются модификацией исходной модели поведения пользователя. Предполагается, что с небольшой вероятностью $\frac{\varepsilon}{N}$ пользователь, воспользовавшись адресной строкой браузера, может перейти на любую из страниц, и с вероятностью $1-\varepsilon$ придерживаться исходной схемы поведения (на практике часто используют значения ε порядка 0.1). Модификация "модели" поведения пользователя, с учётом условия $\sum_i p_i = 1$, приводит к следующей системе уравнений для значений $PR(V_i)$

$$PR(V_i) = \frac{\varepsilon}{N} + (1-\varepsilon) \sum_{V_j \in B_i} \frac{PR(V_j)}{l_j}.$$

На уровне матриц это приводит к следующей задаче

$$\left((1-\varepsilon)\hat{A} + \frac{\varepsilon}{N} E\right) \vec{p} = \vec{p},$$

где все элементы E равны единице. Матрица $\left((1-\varepsilon)\hat{A} + \frac{\varepsilon}{N} E\right)$ содержит уже строго-положительные элементы и удовлетворяет условиям теоремы Перрона-Фробениуса (причём *почти* все её стобцовые суммы окажутся равными 1, кроме висячих вершин, сумма для которых окажется равной ε), поэтому её максимальное по модулю собственное значение будет вещественным положительным и простым (впрочем, не обязательно равным 1), а соответствующий собственный вектор будет вещественным

и знакопостоянным. Следовательно, Вектор важности вершин $\vec{p}$ можно получить, решая задачу о собственном векторе

$$\left((1-\varepsilon)\hat{A} + \frac{\varepsilon}{N}E\right)\vec{p} = \lambda_{max}\vec{p},$$

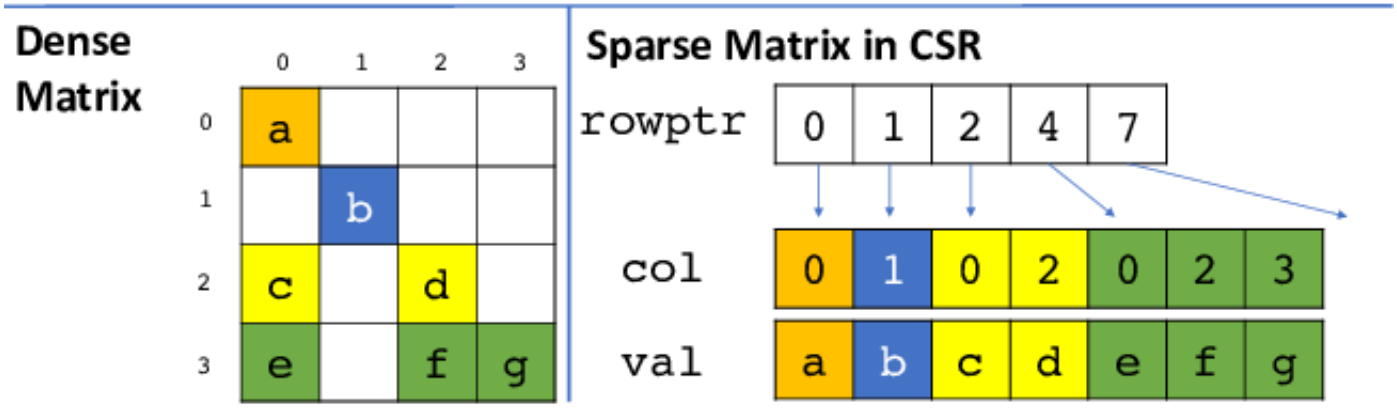
применяя степенной метод с нормировкой результатов,

$$\left((1-\varepsilon)\hat{A} + \frac{\varepsilon}{N}E\right)\cdot x_i = y_i,$$
$$x_{i+1} = \frac{y_i}{\|y_i\|_1},$$
$$y_0 = \begin{bmatrix}1/N & 1/N & \dots & 1/N\end{bmatrix}.$$

Более подробное исследование влияния значений параметра ε и возможных обобщений этого приёма можно найти в работе [3] из списка литературы в конце данного раздела. Обоснование степенного метода можно найти в уже упоминавшемся учебнике Е. Е. Тыртышникова [4]. Важно отметить, что матрица E полностью состоит из единиц (то есть это матрица ранга 1), и умножение данной матрицы на вектор можно провести за дополнительные $O(N)$ операций (то есть модификация исходной "модели" поведения пользователя – крайне простая). Матрицы ссылок $\hat{A}$ на практике обычно сильно разреженные, поэтому для их хранения и реализации операции их умножения на вектор используют специальный формат разреженный по строкам формат хранения: CSR – Compressed Sparse Row format. Идейно, данный формат описан на картинке ниже:

SpMV

```
// loop over rows
for (i=0; i<N; i++) {
  y[i] = 0;
  for (k=rowptr[i]; k<rowptr[i+1]; k++) {
    y[i] += val[k]*x[col[k]]; }
}
```



Процедуры по хранению и работе с матрицами в CSR-формате реализованы в большом количестве открытых и коммерческих библиотек, в частности, Intel MKL и SPARSKIT2.

Другой интересной возможностью применения матричных методов для задач, связанных с анализом графов, является задача о поиске матрицы *схожести* вершин графа, сформулированная в [5].

В этой задаче необходимо вычислить величину $0 \leq s(V_i, V_j) \leq 1$ (элементы матрицы S), характеризующую схожесть вершин V_i и V_j, следуя модели

$$s(V_i, V_j) = \begin{cases} 1, & \text{если } V_i = V_j, \\ 0, & \text{если } |I_i| = 0 \text{ или } |I_j| = 0, \\ \dfrac{c}{|I_i| \cdot |I_j|} \cdot \sum\limits_{k \in I_i} \sum\limits_{l \in I_j} s(V_k, V_l), & \text{иначе,} \end{cases}$$

где I_i – множество вершин-соседей вершины V_i с ребрами направленными к ней, а $0 \leq c \leq 1$ – некоторая константа, которую выбирают на практике. В экстремальных случаях $c = 1$ все вершины графа, для которых есть входящие рёбра, оказываются абсолютно схожи между собой (для них $s(V_i, V_k) = 1$). В другом крайнем случае $c = 0$ все вершины становятся полностью различны (то есть схожи только с самими собой). Отсюда становится ясно, что на практике необходимо подбирать значение $0 < c < 1$. Данная модель известна в литературе под названием SimRank (Similarity Rank). Для нас она интересна тем, что элементы матрицы S входят в уравнения линейным образом

$$S = cA^\top SA - c \cdot \mathrm{diag}\left(A^\top SA\right) + I.$$

Заметим, что для матрицы S возникает задача о неподвижной точке

$$S = \mathbf{F}(S) = \underbrace{cA^\top SA - c \cdot \mathrm{diag}\left(A^\top SA\right) + I.}_{\mathbf{F(S)}}$$

Для решения этой задачи можно использовать аналогичный метод простой итерации, стартуя с единичной матрицы,

$$S_{k+1} = \mathbf{F}(S_k) = \underbrace{cA^\top S_k A - c \cdot \mathrm{diag}\left(A^\top S_k A\right) + I}_{\mathbf{F(S)}}, S_0 = I.$$

Ускорение вычислений в данной итерационной схеме при помощи матричных методов предложено в работе [6]. Более подробный обзор данного алгоритма и класса похожих математических моделей можно найти в работе [7].

Упражнения:

1. Проведите расчёт вектора $\vec{p}$ для графа, представленного на картинке.
2. Докажите, что матрица связности сильно-связного графа будет неприводимой.
3. Разберитесь с обоснованием сходимости степенного метода для задачи поиска вектора $\vec{p}$.
4. Приведите обоснование, что $\{A^m\}_{i,j}$ соответствуют числу маршрутов между вершинами i и j длины m.
5. Найдите данные и подготовьте матрицу связности для графа, представляющего карту московского метро, произведите расчёт вектора *важностей* вершин в этом графе.
6. Проведите расчёт вектора *важностей* вершин в каком-либо графе с использованием CSR-формата представления разреженных матриц.
7. Для графа, представленного на картинке, реализуйте итерационный процесс по вычислению матрицы схожести вершин, следуя описанию модели SimRank.

Литература:

1. *Харари Ф.* Теория графов., М.: Мир, 1973.
2. *Brin S.*, *Page L.* The anatomy of a large-scale hypertextual web search engine // Computer networks and ISDN systems. – 1998. – Т. 30. – №. 1-7. – С. 107-117.
3. *Поляк Б. Т.*, *Тремба А. А.* Решение задачи PageRank для больших матриц с помощью регуляризации // Автоматика и телемеханика. – 2012. – №. 11. – С. 144-166.
4. *Тыртышников Е.Е.* Методы численного анализа. М.: Издательский центр "Академия", 2007. 320 с.
5. *Jeh G.*, *Widom J.* Simrank: a measure of structural-context similarity //Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. – 2002. – С. 538-543.
6. *Oseledets I. V.*, *Ovchinnikov G. V.*, *Katrutsa A. M.* Fast, memory-efficient low-rank approximation of SimRank // Journal of Complex Networks. – 2017. – Т. 5. – №. 1. – С. 111-126.
7. *Велихов П. Е.* Меры семантической близости статей Википедии и их применение к обработке текстов // Информационные технологии и вычислительные системы. – 2009. – №. 1. – С. 23-37.

5.5 Метод глобальной чувствительности Соболя.

Метод глобальной чувствительности Соболя разработан с целью получить упорядочение параметров скалярной многомерной функции с точки зрения *важности* их влияния на её значения. Исходно метод разработан для интегрируемых функций, определенных в многомерном d-мерном кубе:

$$y = f(x_1, \dots, x_d) = f(\mathbf{x}), x_i \in [0, 1].$$

Если зафиксировать значения аргументов $x_i = x_i^*$, то упорядочить их "влияние" на итоговый результат y можно через вычисление вектора-градиента (или численного приращения) в этой точке. Анализ чувствительности с использованием градиента или его аналогов значений функции f к изменениям своих аргументов принято называть *локальным*. Если же нас интересует значимость параметров, в целом по всей области, то задачу об оценке их значимости принято называть *глобальным анализом чувствительности*. Существуют различные методы решения этой задачи (векторы Шепли, метод Морриса, метод моментов, RBD-FAST и другие), но мы сконцентрируемся на единственном заявленном в этом разделе.

Для начала заметим, что любая многомерная скалярная функция допускает представление

$$f(x_1, \dots, x_d) = f_0 + \sum_i^d f_i + \sum_i^d \sum_{j>i}^d f_{ij} + \dots + f_{1\dots d},$$

где f_0 – постоянный член, а $f_i = f_i(x_i)$ – одномерные функции от соответствующих переменных, $f_{ij} = f_{ij}(x_i, x_j)$ – двумерные и так далее. В литературе данное разложение известно под названием разложения Соболя-Гёффдинга (и также ANOVA – ANAlysis of VAriances), существует и единственно, если потребовать попарной ортогональности слагаемых данного разложения, пользуясь определением скалярного произведения через интегрирование сомножителей по соответствующим переменным.

В конкретных формулах эта запись для слагаемых разложения выражается следующим образом:

$$f_0 = \int_0^1 f(x_1, \dots, x_d) dx_1 \dots dx_d,$$

то есть f_0 – это среднее значение для f. Далее

$$\int f(\mathbf{x}) \prod_{k \neq i} dx_k = f_0 + f_i(x_i),$$

$$\int f(\mathbf{x}) \prod_{k \neq i,j} dx_k = f_0 + f_i(x_i) + f_j(x_j) + f_{i,j}(x_i, x_j),$$

$$\dots$$

и так далее. Всего получается необходимо реккурентным образом вычислить 2^n слагаемых. По мере роста размерности слагаемых $f_{i_1,\dots,i_s}$ можно заметить, что снижается размерность интегралов, которые нужно при этом вычислять.

Чаще всего для глобального анализа чувствительности по Соболю при вычислении интегральных операторов используется метод Монте-Карло с использованием квазислучайных числовых последовательностей Соболя. Такие последовательности обеспечивают оптимальную скорость численной сходимости (обоснование этого весьма непросто и выходит далеко за рамки учебного курса). Нетрудно заметить, что для каждого слагаемого, кроме первого, верно свойство о нулевом среднем значении

$$\int_0^1 f_{i_1,\dots,i_s}(x_{i_1}, \dots, x_{i_s}) dx_k = 0 \text{ для } k \in i_1, \dots, i_s.$$

Из-за этого на уровне дисперсии можно заметить, что выполнено разложение

$$\int f^2(\mathbf{x}) \prod_{i=1}^{d} dx_i - f_0^2 = \sum_{s=1}^{n} \sum_{i_1<\ldots<i_s}^{n} \int f^2_{i_1,\ldots,i_s}(x_{i_1},\ldots,x_{i_s}) dx_{i_1} \ldots dx_{i_s}.$$

Значения

$$D_{i_1,\ldots,i_s} = \int f^2_{i_1,\ldots,i_s}(x_{i_1},\ldots,x_{i_s}) dx_{i_1} \ldots dx_{i_s}$$

принято называть частичными дисперсиями. Если предположить, что исходная функция не равна константе, а дисперсия для неё равна D, то можно ввести следующие значения

$$S_{i_1,\ldots,i_s} = \frac{D_{i_1,\ldots,i_s}}{D}.$$

Данные значения называются глобальными индексами чувствительности Соболя и описывают важность групп переменных $x_{i_1},\ldots,x_{i_s}$ с точки зрения их влияния на изменение результата $y = f(x_1,\ldots,x_s)$. Отметим простейшие свойства индексов Соболя:

1. $0 \le S_{i_1,\ldots,i_s} \le 1$,
2. Если $S_{i_k} = 1$, то функция $f(\mathbf{x})$ зависит только от переменной x_k.

Для каждой переменной (группы переменных) нужно также ввести и полные индексы Соболя

$$S^{tot}_{i_1,\ldots,i_s} = \sum_{k=1}^{s} \sum_{i_1<\ldots<i_k} S_{i_1,\ldots,i_k},$$

являющиеся суммой всех отдельных индексов Соболя, в которые входит хотя бы одна из переменных x_{i_k} с номером i_k из закфисированного набора $i_1,\ldots,i_s$. Отсюда следуют ещё два свойства индексов Соболя:

1. $0 \le S_{i_1,\ldots,i_s} \le S^{tot}_{i_1,\ldots,i_s} \le 1$,
2. Если $S^{tot}_{i_1,\ldots,i_s} = 0$, то функция $f(\mathbf{x})$ НЕ зависит группы переменных $x_{i_1},\ldots x_{i_s}$.

Более неформально можно сказать, что индексы Соболя позволяют решить несколько задач:

1. Упорядочить переменные или группы переменных функции по значимости,
2. Отказаться от некоторых переменных,
3. Избавиться от слагаемых высоких порядков в разложении Гёффдинга-Соболя.

Решение первой задачи выглядит понятной – важнее те переменные (или группы переменных), чьи полные индексы Соболя больше. Отметим, что полный индекс чувствительности для группы переменных вводится по аналогии с полным индексом чувствительности для одной переменной

$$S^{tot}_{i_1,\ldots,i_s} = \sum_{k=1}^{s} \sum_{j_1,\ldots j_k \in i_1,\ldots,i_s} S_{j_1,\ldots,j_k}.$$

Далее на интуитивном уровне можно отметить, что малое значение полного индекса чувствительности $S_{i_1,\ldots,i_s} = \varepsilon \ll 1$ соответствует “слабой” зависимости исходной функции от группы своих переменных с такими номерами. Отсюда возникает желание понизить размерность этой функции, отказавшись от

рассмотрения этой группы переменных. Наиболее естественным способом здесь кажется фиксирование случайным образом значений этих переменных $x^*_{i_1}, \ldots, x^*_{i_s}$. В таком случае известна теоретическая оценка, речь о которой пойдёт ниже.

Введем для начала величину погрешности приближения функции $f(\mathbf{x})$ при помощи функции $h(x)$ как

$$\delta(f,h) = \frac{1}{D}\int [f(\mathbf{x}) - h(\mathbf{x})]^2 \prod_{i=1}^{d} dx_i,$$

где D – значение дисперсии исходной функции $f(x)$. В наиболее грубом случае $h(\mathbf{x}) \equiv \mathrm{const}$ и можно заметить, что тогда $\delta(f,h) = 1$, следовательно нас будут интересовать такие $h(\mathbf{x})$, чтобы выполнялось $\delta(f,h) \ll 1$. Без потери общности будем считать, что мы фиксируем подряд идущие переменные $x_k, \ldots x_d$, а их полный индекс чувствительности мал.

В таком случае можно выбрать аппроксимацию $h(\mathbf{x}) = f(x_1, \ldots x_{k-1}, x^*_k, \ldots, x^*_d)$, выбрав значения $x^*_k, \ldots, x^*_d$ случайным образом. Тогда верно следующее утверждение:

Теорема.

1. $\delta(f,h) \geq S^{tot}_{k,\ldots,d}$, то есть возможная среднеквадратичная ошибка ограничена снизу полным индеском чувствительности.
2. $\forall \varepsilon > 0$ $P\left(\delta(f,h) < (1+\frac{1}{\varepsilon})S^{tot}_{k,\ldots,d}\right) \geq 1-\varepsilon$, вероятность удачного случайного выбора с небольшой погрешностью существенно ненулевая.

Так, например, если $S_{k,\ldots,d} = 10^{-4}$, а $\varepsilon = \frac{1}{10}$, можно наблюдать

$$P\left(\delta(f,h) < 11 \cdot 10^{-5}\right) \geq \frac{9}{10}.$$

То есть с вероятностью 90% погрешность приближения меньшей размерности

$$h(x_1, \ldots, x_{k-1}) = f(x_1, \ldots, x_{k-1}, x^*_k, \ldots, x^*_d)$$

при помощи "фиксации" случайно выбранных точек $x^*_k, \ldots, x^*_d$ не превысит $11 \cdot 10^{-5}$. Доказательство данной теоремы выходит за рамки целей исходного методического пособия, но его можно посмотреть в оригинальных работах И. М. Соболя из списка рекомендованной литературы в конце этого раздела.

Надёжности построенной таким способом аппроксимации можно добавить путём N повторений таких случайных выборов и взятия усреднения из них в качестве итоговой аппроксимации (фактически, возникает дополнительный *уровень* для методики Монте-Карло).

Несмотря на простоту и сравнительно большой возраст описанной идеи, глобальный анализ чувствительности при помощи метода Соболя популярен в современной литературе, а на языке программирования Python в рамках библиотеки по анализу чувствительности SALib (Sensitivity Analysis Library) реализован модуль sobol, пример использования которого для конкретной функции, мы приводим ниже.

Сначала проведем подгрузку нужных библиотек:

```
import SALib
from SALib.sample import saltelli
from SALib.analyze import sobol
from math import *
import numpy as np
```

Опишем далее целевую функцию и интервалы значений переменных:

```
a = 0.0001
b = 0.3
# target function
def f(x):
    return sin(x[0]) + a * sin(x[1])*sin(x[1]) + b * sin(x[0]) * x[2]**3
# problem description structure
problem = {
    'num_vars': 3,
    'names': ['x1', 'x2', 'x3'],
    'bounds': [[-3.14159265359, 3.14159265359],
               [-3.14159265359, 3.14159265359],
               [-3.14159265359, 3.14159265359]]
}
```

Подготовим точки для интегрирования методом Монте-Карло и вычислим значения целевой функции:

```
# number of samples for Monte Carlo integration
n = 32768
param_values = saltelli.sample(problem, n)
#print(param_values)
# prepare the numpy array for the results
Y = np.zeros([param_values.shape[0]])
#evaluate the target function at the sample points
for i, X in enumerate(param_values):
    Y[i] = f(X)
#print(Y)
```

В конце концов, вызываем вычисление индексов Соболя на основе сформированных данных:

```
# Call of the Sobol sensitivity analysis procedure
Si = sobol.analyze(problem, Y)
# first order indices
print(Si['S1'])
# total sensitivity indices
print(Si['ST'])
# second order indices
print("x1-x2:", Si['S2'][0,1])
print("x1-x3:", Si['S2'][0,2])
print("x2-x3:", Si['S2'][1,2])
```

Из приведённого примера видно, что анализ чувствительности методом Соболя допускает использование параллельных вычислений на этапе подготовки выборки квазислучайных точек многомерного пространстка и подготовки выборки значений целевой функции для анализа чувствительности.

Отметим, что этап вычисления индексов чувствительности в библиотеке SALib выполняется последовательно. Необходимо отметить, что использование массивно-параллельных вычислений возможно в том числе и на этом этапе. В случае существования достаточно точной малоранговой аппроксимации для целевой многомерной функции $f(x_1, \ldots, x_d)$ в формате тензорного поезда возможна специальная высокоэффективная организация процедуры по вычислению индексов чувствительности Соболя из источника [6] списка литературы данного раздела.

Упражнения:

1. Докажите, что при $S_1 + S_2 + \ldots + S_d = 1$, функция $f(x_1, \ldots, x_n)$ представима в виде суммы одномерных функций.

2. При помощи замены переменных сведите задачу о глобальной чувствительности интегрируемой функции $F(X_1, \ldots, X_d)$, $X_i \in [a_i, b_i]$ к задаче о глобальной чувствительности для функции $f(x_1, \ldots, x_d)$, $x_i \in [0, 1]$.

3. Для функции $f(x_1, x_2, x_3, x_4)$ запишите полные формулы для S_1^{tot}, $S_{1,2}^{tot}$, $S_{2,3,4}^{tot}$. Чему соответствует значение $1 - S_1^{tot}$ и можно ли его связать с другими индексами чувствительности?

4. Вычислите индексы чувствительности Соболя функции $f(x_1, x_2, x_3) = x_1 + 10^{-2}x_2 + \sin(x_3 + x_1)$.

5. (*) С какой скоростью будет уменьшаться дисперсия ошибки $\delta(f, h)$ при усреднениях N повторных случайных реализаций?

6. С использованием библиотеки `SALib` вычислите $f(x_1, x_2, x_3) = x_1 + 10^{-2}x_2 + \sin(x_3 + x_1)$ приближения для индексов чувствительности с использованием $2^{15}, 2^{16}, 2^{17}$ точек. Проведите измерения времени работы отдельных этапов анализа чусвтвительности и оцените скорость-качество сходимости приближенных индексов чусвтвительности к точным значениям.

7. Для функции $f(x, y) = (100 + \sin(x)) \cdot y^2 + (100 + \cos(x)) \cdot y^3 + 2x$ при $x \in [-\pi, \pi]$, $y \in [-\pi, \pi]$ проведите вычисление индексов Соболя и вычислите на нескольких равномерных сетках сингулярные числа таких функционально-порождённых матриц. Что можно сказать о значимости переменных в каждом из двух подходов?

Литература:

1. Sobol I. M. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates //Mathematics and computers in simulation. – 2001. – Т. 55. – №. 1-3. – С. 271-280.

2. Соболь И. М. (1990). Об оценке чувствительности нелинейных математических моделей. Математическое моделирование, 2(1), 112-118.

3. Соболь И. М. Многомерные квадратурные формулы и функции Хаара //М.: Наука. – 1969.

4. Gasanov M., Petrovskaia, A., Nikitin A., Matveev S., Tregubova P., Pukalchik M., Oseledets I. (2020, June). Sensitivity analysis of soil parameters in crop model supported with high-throughput computing. In International Conference on Computational Science (pp. 731-741). Cham: Springer International Publishing.

5. Документация библиотеки `SALib`:
`https://salib.readthedocs.io/en/latest/user_guide/getting-started.html`

6. Chertkov A., Ryzhakov G., Oseledets I. Black box approximation in the tensor train format initialized by ANOVA decomposition //SIAM Journal on Scientific Computing. – 2023. – Т. 45. – №. 4. – С. A2101-A2118.

Список литературы

[1] *Тыртышников Е.Е.* Матричный анализ и линейная алгебра. М.: Физматлит, 2007. 480 с.

[2] *Тыртышников Е.Е.* Методы численного анализа. М.: Издательский центр “Академия”, 2007. 320 с.

[3] *Желтков Д. А.*, *Тыртышников Е. Е.* Параллельная реализация матричного крестового метода //вычислительные методы и программирование. – 2015. – Т. 16. – С. 369-375.

[4] *Goreinov S. A.*, *Tyrtyshnikov E. E.*, *Zamarashkin N. L.* A theory of pseudoskeleton approximations //Linear algebra and its applications. – 1997. – Т. 261. – №. 1-3. – С. 1-21.

[5] *Welch W. J.* Algorithmic complexity: three NP-hard problems in computational statistics //Journal of Statistical Computation and Simulation. – 1982. – Т. 15. – №. 1. – С. 17-25.

[6] *Goreinov S. A. et al.* How to find a good submatrix // Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub. – 2010. – С. 247-256.

[7] *Савостьянов Д. В.* Быстрая полилинейная аппроксимация матриц и интегральные уравнения. // Кандидатская диссертация – 2006

[8] *Halko N.*, *Martinsson, P.-G.*, *Tropp, J. A.* Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2), 217–288 2011.

[9] *Nakatsukasa Y.* Fast and stable randomized low-rank matrix approximation //arXiv preprint arXiv:2009.11392. – 2020.

[10] Веб-страница автора Gregory Gundersen, https://gregorygundersen.com/blog/2019/01/17/randomized-svd/

[11] *Матвеев С. А.*, *Тыртышников Е. Е.*, *Смирнов А. П.*, *Бриллиантов Н. В.* Быстрый метод решения уравнений агрегационно-фрагментационной кинетики типа уравнений Смолуховского // Вычислительные методы и программирование. – 2014. – Т. 15. – №. 1. – С. 1-8.

[12] *Matveev S. A.*, *Smirnov A. P.*, *Tyrtyshnikov E. E.* A fast numerical method for the Cauchy problem for the Smoluchowski equation // Journal of Computational Physics. – 2015. – Т. 282. – С. 23-32.

[13] *Крапивский П. Л.*, *Реднер С.*, *Бен-Наим Э.* Кинетический взгляд на статистическую физику // (Пер. с англ.) Москва: Научный мир. – 2012.

[14] Галкин В. А. Уравнение Смолуховского. М.: Физматлит // Москва. – 2002.

[15] *Чугунов В. Н.*, Нормальные и перестановочные теплицевы и ганкелевы матрицы, Москва, Наука, 2017.

[16] *Badeau R.*, *Boyer R.* (2008). Fast multilinear singular value decomposition for structured tensors. SIAM Journal on Matrix Analysis and Applications, 30(3), 1008-1021.

[17] *Fawzi, A., et al.* (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 610(7930), 47-53.

[18] *Тыртышников, Е. Е.*. Матрицы, тензоры, вычисления. М.: МГУ им. М. В. Ломоносова, 2013.

[19] *Oseledets I. V.*, *Tyrtyshnikov E. E.* (2009). Breaking the curse of dimensionality, or how to use SVD in many dimensions. SIAM Journal on Scientific Computing, 31(5), 3744-3759.

[20] *Zhang C.*, *Jeckelmann E.*, *White S. R.* (1998). Density matrix approach to local Hilbert space reduction. Physical review letters, 80(12), 2661.

[21] *Оселедец И. В.* Вычислительные тензорные методы и их применения. *докторская диссертация*, М.: ИВМ РАН, 2012.

[22] *Sultonov A.*, *Matveev S.*, *Budzinskiy S.* Low-rank nonnegative tensor approximation via alternating projections and sketching // Computational and Applied Mathematics. – 2023. – Т. 42. – №. 2. – С. 68.

[23] *Cichocki A. et al.* Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1 // arXiv preprint arXiv:1609.00893. – 2016.

[24] *Cichocki A. et al.* Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives // Foundations and Trends® in Machine Learning. – 2017. – Т. 9. – №. 6. – С. 431-673.

[25] *Харари Ф.* Теория графов., М.: Мир, 1973.

[26] *Brin S.*, *Page L.* The anatomy of a large-scale hypertextual web search engine // Computer networks and ISDN systems. – 1998. – Т. 30. – №. 1-7. – С. 107-117.

[27] *Поляк Б. Т.*, *Тремба А. А.* Решение задачи PageRank для больших матриц с помощью регуляризации // Автоматика и телемеханика. – 2012. – №. 11. – С. 144-166.

[28] *Jeh G.*, *Widom J.* Simrank: a measure of structural-context similarity // Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. – 2002. – С. 538-543.

[29] *Oseledets I. V.*, *Ovchinnikov G. V.*, *Katrutsa A. M.* Fast, memory-efficient low-rank approximation of SimRank // Journal of Complex Networks. – 2017. – Т. 5. – №. 1. – С. 111-126.

[30] *Велихов П. Е.* Меры семантической близости статей Википедии и их применение к обработке текстов // Информационные технологии и вычислительные системы. – 2009. – №. 1. – С. 23-37.

Учебно-методическое пособие

Алгебраические вычисления, тензоры и оптимизация