

Московский государственный университет имени М. В. Ломоносова



факультет вычислительной математики и кибернетики
кафедра ВТМ

Курсовая работа студента 303 группы

Каноническая аппроксимация тензоров и ее реализация на Python

Выполнил:
студент 3 курса 303 группы
Кузнецов Максим Алексеевич
Научный руководитель:
к.ф.-м.н., с.н.с
Оселедец Иван Валерьевич

Москва, 2012

1 Введение

Тензоры широко используются в физике, дифференциальной геометрии, а также в теории анизотропных сред (которые могут быть анизотропны изначально, как кристаллы низкой симметрии, или вследствие своего движения или напряжений, как текущая жидкость или газ, или как деформированное твердое тело). Тензорные методы активно используются в многофакторном анализе, в психометрике и хемометрике. Подробный обзор методов представления тензоров и их применения можно найти в обзоре [6]. Одним из важнейших представлений является каноническое разложение тензора. Одному из методов построения такого разложения и посвящена данная работа.

1.1 Цель курсовой работы

Целью курсовой работы является изучение метода переменных направлений для канонической аппроксимации тензора, и написание его эффективной реализации. К программе предъявляются следующие требования:

1. Она должна работать для любой размерности тензора.
2. Реализация на Python

Интерес к пункту 1) обусловлен тем, что многомерные массивы в памяти должны быть представлены как одномерные, и для выполнения операций с большим числом индексов необходимо предпринимать специальные усилия.

Язык Python выбран потому, что он является удобной средой для прототипирования сложных вычислительных алгоритмов, однако его использование добавляет дополнительные сложности: циклы в этом языке являются достаточно медленными, и задачу необходимо свести к операциям с матрицами.

В настоящее время автору неизвестны программы, реализующие алгоритм построения канонической аппроксимации тензора произвольной размерности, написанные на Python.

2 Основные определения и понятия

Введем основные определения, необходимые для дальнейшего изложения:

Определение

Тензором A размерности d назовем многомерный массив, элементы которого $A(i_1, i_2, \dots, i_d)$ имеют d индексов. $1 \leq i_k \leq n_k$; n_k называются модовыми размерами (размерами мод)

Определение

Каноническим разложением многомерного массива (*тензора*) называется представление вида

$$A(i_1, i_2, \dots, i_d) = \sum_{\alpha=1}^r U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha), \quad (1)$$

где U_k называются *факторами* канонического разложения, а r — каноническим рангом.

Уравнение (1) является основным.

2.1 О ранге тензора

Рангом r тензора A называется наименьшее r , которое в выражении (1) дает искомый тензор. К сожалению, не существует конечного алгоритма поиска тензорного ранга [6], с чем сопряжены определенные трудности в построении канонического разложения и канонической аппроксимации.

Для тестирования алгоритмов в данной работе была реализована процедура, строящая тензор требуемого ранга из набора случайных компонент (см. “Реализация на Python”)

3 Алгоритм ALS

Пусть задан тензор A с элементами $A_{i_1 \dots i_d}$. Задача состоит в том, чтобы найти его каноническое приближение, а именно найти такие матрицы U_1, \dots, U_d

$$A_{i_1, \dots, i_d} \approx \sum_{\alpha=1}^r U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha). \quad (2)$$

Математическая постановка задачи состоит в том, чтобы решить задачу (11) в смысле наименьших квадратов

$$\sum_{i_1, \dots, i_d} \left(A(i_1, \dots, i_d) - \sum_{\alpha=1}^r U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha) \right)^2 \rightarrow \min. \quad (3)$$

Будем решать вариационную задачу поиска аппроксимации тензора с помощью алгоритма ALS (Alternating Least Squares), подробное изложение которого можно найти в статье [4]. Основная идея алгоритма, состоит в том, чтобы фиксировать все факторы, кроме одного, канонического разложения и искать минимум функционала только по нему. Путем циклических перестановок, используя уже полученные факторы, строятся последующие, до тех пор, пока не будет достигнута требуемая точность аппроксимации или, пока не сработают другие критерии остановки алгоритма (превышение максимального количества итераций, превышение времени выполнения программы). Построим один шаг алгоритма ALS.

3.1 Описание алгоритма ALS

Для упрощения расчетных формул проведем построение метода отыскания минимума для тензора размерности 3 (в d -мерном случае рассуждения аналогичны). Введем функционал F ,

$$F = \sum_{i,j,k} \left(A_{ijk} - \sum_{\alpha=1}^r U_{i\alpha} V_{j\alpha} W_{k\alpha} \right)^2. \quad (4)$$

Найдем частную производную функционала F по $U_{i\hat{\alpha}}$ и приравняем ее к 0:

$$\frac{\partial F}{\partial U_{i\hat{\alpha}}} = 2 \left(\sum_{i,j,k} (A_{ijk} - \sum_{\alpha} U_{i\alpha} V_{j\alpha} W_{k\alpha}) \right) \left(- \sum_{\hat{\alpha}} (V_{j\hat{\alpha}} W_{k\hat{\alpha}}) \frac{\partial U_{i\hat{\alpha}}}{\partial U_{i\hat{\alpha}}} \right) = 0;$$

$$\frac{\partial U_{i\hat{\alpha}}}{\partial U_{i\hat{\alpha}}} = \delta_{i,\hat{i}} \delta_{\hat{\alpha},\hat{\alpha}};$$

Отсюда,

$$- \sum_{i,j,k,\hat{\alpha}} A_{ijk} \delta_{i,\hat{i}} \delta_{\hat{\alpha},\hat{\alpha}} V_{j\hat{\alpha}} W_{k\hat{\alpha}} + \sum_{i,j,k,\alpha,\hat{\alpha}} U_{i\alpha} V_{j\alpha} \delta_{i,\hat{i}} \delta_{\hat{\alpha},\hat{\alpha}} V_{j,\hat{\alpha}} W_{k\hat{\alpha}} = 0;$$

Окончательно, получаем следующие соотношения:

$$\sum_{j,k} A_{ijk} V_{j\hat{\alpha}} W_{k\hat{\alpha}} = \sum_{j,k,\alpha} U_{i\alpha} V_{j\alpha} W_{k\alpha} V_{j\hat{\alpha}} W_{k,\hat{\alpha}},$$

где

$$\sum_{j,k,\alpha} U_{i,\alpha} V_{j,\alpha} W_{k,\alpha} V_{j,\hat{\alpha}} W_{k,\hat{\alpha}} = \sum_{\alpha} U_{i,\alpha} \left(\sum_j V_{j,\alpha} V_{j,\hat{\alpha}} \right) \left(\sum_k W_{k,\alpha} W_{k,\hat{\alpha}} \right);$$

Обозначим через $M_{\alpha\hat{\alpha}}$ матрицу с элементами

$$M_{\alpha,\hat{\alpha}} = \left(\sum_j V_{j,\alpha} V_{j,\hat{\alpha}} \right) \left(\sum_k W_{k\alpha} W_{k\hat{\alpha}} \right); \quad (5)$$

тогда

$$\sum_{\alpha} U_{i,\alpha} M_{\alpha,\hat{\alpha}} = \sum_{j,k} A_{i,j,k} V_{j,\hat{\alpha}} W_{k,\hat{\alpha}}; \quad (6)$$

Через $F_{i\hat{\alpha}}$ обозначим правую часть. Тогда, имеем

$$\sum_{\alpha} U_{i\alpha} M_{\alpha\hat{\alpha}} = F_{i\hat{\alpha}}. \quad (7)$$

или в виде системы линейных уравнений

$$UM = F. \quad (8)$$

где $M \in \mathbb{R}^{r \times r}$.

Путем циклических перестановок аналогичные соотношения получаем для V и W . В методе ALS гарантировано убывание невязки, однако до последнего времени не было известно даже теорем о локальной сходимости. Тем не менее, метод ALS является простым, и часто наиболее эффективным, методом канонической аппроксимации тензоров.

3.2 Оценка сложности алгоритма ALS

Предположим, что заданный тензор A имеет размеры мод n_1, n_2, n_3 и ранг r . Тогда матрицы $U \in \mathbb{R}^{n_1 \times r}$, $V \in \mathbb{R}^{n_2 \times r}$, $W \in \mathbb{R}^{n_3 \times r}$.

Простейшая программа для вычисления $U_{i\alpha}$ может быть написана с помощью помощью вложенных циклов. С помощью формул (5) и (6) вычислим сложность вычисления одного шага алгоритма. Так как индексы i, j, k меняются в пределах $1 \dots n_1, n_2, n_3$ соответственно, а α в пределах $1 \dots r$, получим следующие соотношения:

1. Сложность вычисления левой части системы для одной матрицы U пропорциональна $O((n_2 + n_3)r^2)$; аналогично для V и W : $O((n_3 + n_1)r^2)$; $O((n_1 + n_2)r^2)$;
2. Сложность вычисления правой части $O(n_1 n_2 n_3 r)$;

Сравнительную характеристику алгоритма ALS можно найти в статье [5]

4 Реализация на Python

Поставленная задача реализации алгоритма ALS на Python предполагает:

- Реализацию алгоритма в виде единой процедуры для любой размерности
- Реализацию функций вычисления правой и левой частей системы (8),

используя математические ухищрения и возможности Python, для того чтобы обойти проблему неопределенной размерности, так как предыдущий пункт эту проблему ставит.

- Ограничение инструментария стандартными функциями библиотек (довольно богатых),

чтобы избежать потерь в скорости, так как Python интерпретируемый скриптовый язык.

В ходе написание программы были реализованы следующие функции:

- Функция вычисления правой части (rights)
- Функция вычисления левой части (lefts)
- Функция получения случайного тензора известного ранга (randomtensor)
- Функция получения тензора по заданным факторам разложения (gettensor)

4.1 О языке Python

Python [1] — высокоуровневый язык программирования общего назначения, ориентированный на производительность разработчика и читаемость кода. Python является интерпретируемым языком, и эталонной реализацией интерпритатора считается CPython. Благодаря большому количеству библиотек и специальных модулей, решение большого количества задач упрощается с помощью их использования, более того сокращается время как разработки, так и выполнения программ. в ходе написания данной работы, автором использовались следующие библиотеки:

- NumPy[2] — библиотека для работы с матрицами и многомерными массивами, включающая высокоуровневые математические функции для операций с ними.
- SciPy[3] — библиотека, содержащая большое количество математических функций, а также средства для визуализации полученных результатов.
- стандартные библиотеки Time и прочие.

Разработчики Python придерживаются определенной философии программирования “Zen of Python” (“Дзен Питона”), автором которой является Тим Пейтерс, и которая выводится единожды за сеанс, по команде

```
import this
```

4.2 Текст философии:

Красивое лучше, чем уродливое.

Явное лучше, чем неявное.

Простое лучше, чем сложное.

Сложное лучше, чем запутанное.

Плоское лучше, чем вложенное.

Разреженное лучше, чем плотное.

Читаемость имеет значение.

Особые случаи не настолько особые. чтобы нарушать правила.

При этом практичность важнее безупречности.

Ошибки никогда не должны замалчиваться.

Если не замалчиваются явно.

Встретив двусмысленность, отбрось искушение угадать.

Должен существовать один — и, желательно, только один — очевидный способ сделать это.

Хотя он поначалу может быть и не очевиден, если вы не голландец.

Сейчас лучше, чем никогда.

Хотя никогда зачастую лучше, чем прямо сейчас.

Если реализацию сложно объяснить — идея плоха.

Если реализацию легко объяснить — идея, возможно, хороша.

Пространства имён — отличная штука! Будем делать их побольше!

4.3 Функция вычисления правой части (rights)

Вычисление правой части системы (8) представляет некоторую сложность, при попытке реализации с помощью циклов (в силу переменности размерности тензора), поэтому прибегнем к математическим преобразованиям, с целью получить матрично-матричные произведения. Для простоты изложения, вновь ограничимся размерностью тензора $d = 3$

$$F_{i\alpha} = \sum_{j,k} A_{ijk} V_{j\alpha} W_{k\alpha} = \sum_{j,k,\beta} A_{ijk} V_{j\alpha} \delta(\alpha, \beta) W_{k\beta} = \sum_{j,\beta} V_{j\alpha} \delta(\alpha, \beta) \sum_k A_{ijk} W_{k\beta};$$

заметим здесь матричные перемножения:

$$F_{i\alpha} = W_\alpha(V_\alpha A_i)$$

Данное выражение обобщается и на случай произвольной размерности тензора. Поясним эту запись для произвольной размерности тензора.

$$F_{i\alpha} = U_{d\alpha} U_{d-1\alpha} \dots U_{k+1\alpha} U_{k-1\alpha} \dots U_{1\alpha} A_i \quad (9)$$

Данная формула дает нам схему вычисления элемента матрицы $F_{i\alpha}$. Однако прямо воспользоваться этой формулой не получится, это скорее некоторая формальная запись. На самом деле A_i — срез тензора по k — оси, причем размер ее приведен к размеру вектора $U_{j\alpha}$, где $j \neq k, j = 1 \dots d$ — размерность тензора так, чтобы было возможно умножение на него справа. Номер k соответствует вычисляемому фактору U_k . Результат каждого умножения называем матрицей S . “Свертка” формулы (9) (то есть умножение вектора на матрицу) происходит справа налево, каждый раз меняя размерность полученной матрицы, чтобы умножение на следующий вектор было возможным. То есть:

$$S = U_{j\alpha} S$$

На k -ом шаге размер матрицы S равен $n_{k+1} \times n_{k+2} \dots n_d$, при $k = d$ S становится числом. Перемножив таким образом все известные факторы U_j найдем элемент $F_{i\alpha}$. Для вычисления же всей матрицы F потребуется $i \alpha$ таких “элементарных” операций. Чтобы окончательно понять, каков алгоритм получения правой части, приведем код функции:

Входные данные: a — тензор, u — список известных (фиксированных) факторов, d — размерность тензора, r — ранг тензора, k — номер вычисляемого фактора.

Выходные данные: правая часть системы, f

```
def rights(a,u,dimension,d,r,k):
    f=zeros((dimension[k],r))

    for i in range (0,dimension[k]):
        for alf in range(0,r):
            kol=0
            for j in range(0,d):
                if (j<>k):
                    if (kol<>1):
                        s=dot(u[j][:,alf],a.take([i],axis=k).reshape(size(u[j][:,alf]),
size(a.take([i],axis=k))/size(u[j][:,alf]),order='F'))

                        kol=1
                    else:
                        s=s.reshape(size(u[j][:,alf]),size(s)/size(u[j][:,alf]),order='F')
                        s=dot(u[j][:,alf],s)
                f[i,alf]=s
    return f
```

4.4 Функция вычисления левой части (lefts)

Левая часть системы (8) может быть вычислена по формуле:

$$(U_1^T U_1) \circ (U_2^T U_2) \circ \dots \circ (U_d^T U_d), \quad (10)$$

где под символом ‘ \circ ’ подразумевается поэлементное произведение, а в скобках матричное. Обе эти операции реализованы с помощью стандартных функций Python.

Реализация на Python такова:

```

def lefts(u,k,d,r):
    m=ones((r))
    for i in range(0,d):
        if (i<>k):
            m=m*dot(u[i].transpose(),u[i])
    return m

```

Входные данные: u — список факторов, k — номер вычисляемого фактора, d — размерность тензора, r — ранг. Выходные данные: m — левая часть системы.

4.5 Функция получения случайного тензора

$$A_{i_1, \dots, i_d} = \sum_{\alpha=1}^r U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha). \quad (11)$$

для получения здесь матрично матричных перемножений применим хитрость:

$$A_{i_1, \dots, i_d} = \sum_{\alpha=1}^r U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha) = \sum_{\alpha_1}^r U_1(i_1, \alpha_1) \delta(\alpha_1, \alpha_2) \sum_{\alpha_2}^r U_2(i_2, \alpha_2) \dots \sum_{\alpha_d}^r U_d(i_d, \alpha_d); \quad (12)$$

в итоге получим:

$$A = U_1 \hat{U}_2 \dots \hat{U}_{d-1} U_d;$$

где \hat{U}_j имеет вид:

$$\begin{pmatrix} U_j(1) & 0 & 0 & \dots & 0 \\ 0 & U_j(2) & 0 & \dots & 0 \\ & & \dots & & \\ 0 & \dots & 0 & 0 & U_j(r) \end{pmatrix};$$

$U_j(k)$ — столбец матрицы U_j , $j = 2, \dots, d-1$, $k = 1, \dots, r$. $\hat{U}_j \in \mathbb{R}^{r, rn}$

Код процедуры:

```

def randomtensor(r,dimension,d):
    u=list(arange(d))
    for i in range(0,d):
        u[i]=randn(dimension[i],r)
    u0=[x.copy() for x in u]
    s=1
    tr=u[0]
    temp=list(arange(d-2))
    for j in range(0,d-2):
        temp=zeros((r,r*dimension[j+1]))
        for i in range(0,r):
            temp[i,i*dimension[j+1]:i*dimension[j+1]+dimension[j+1]]=u[j+1][:,i].transpose()
        u[j+1]=temp.transpose()
    for i in range(0,d-1):
        s=size(tr)/r
        tr=tr.reshape(s,r,order='F')
        tr=dot(tr,u[i+1].transpose())
    tr=tr.reshape(dimension,order='F')
    return tr,u0

```

Входные данные: ранг, размеры мод, размерность тензора Выходные данные: построенный тензор, факторы тензорного разложения.

4.6 Функция получения тензора по заданным факторам

Идея преобразований аналогична изложенной в предыдущем пункте, поэтому ограничимся только предоставлением реализации:

```
def gettensor(u1,r,dimension,d):
    u=[x.copy() for x in u1]
    s=1
    tr=u\footnotemark[1]
    temp=list(arange(d-2))
    for j in range(0,d-2):
        temp=zeros((r,r*dimension[j+1]))
        for i in range(0,r):
            temp[i,i*dimension[j+1]:i*dimension[j+1]+dimension[j+1]]=u[j+1][:,i].transpose()
        u[j+1]=temp.transpose()
    for i in range(0,d-1):
        s=size(tr)/r
        tr=tr.reshape(s,r,order='F')
        tr=dot(tr,u[i+1].transpose())
        tr=tr.reshape(dimension,order='F')
    return tr
```

Входные данные: факторы, ранг, размерности мод, размерность тензора. Выходные данные: построенный тензор.

4.7 Критерий остановки

Критерием остановки служат несколько параметров:

1. $\|A - \hat{A}\|_2 < \varepsilon$ где A — заданный тензор, \hat{A} — аппроксимация. Точность ε задается пользователем.
2. Алгоритм ALS формально ищет локальный минимум, вследствие чего критерий 1)

может быть выполнен за большое время, если алгоритм попал в область локального минимума функционала. В связи с этим дополнительно считается

$$\frac{\|U_{inew} - U_i\|_2}{\|U_i\|_2} < \varepsilon_2,$$

где точность ε_2 зависит от заданной пользователем ε

1. Превышение допустимого количества итераций (допустимым по умолчанию считается 45000 итераций)

5 Численные эксперименты

В данном параграфе будут изложены в графическом виде результаты работы программы, реализующей метод ALS. В качестве входных данных подавались:

- Размерность тензора $d = 3$
- Ранг r переменный
- Размерности мод n_i переменные

5.1 Численные эксперименты для случайных тензоров

В качестве входного тензора подается тензор, случайным образом полученный программно (с помощью процедуры `gettensor`) наперед заданного ранга и размерностей мод.

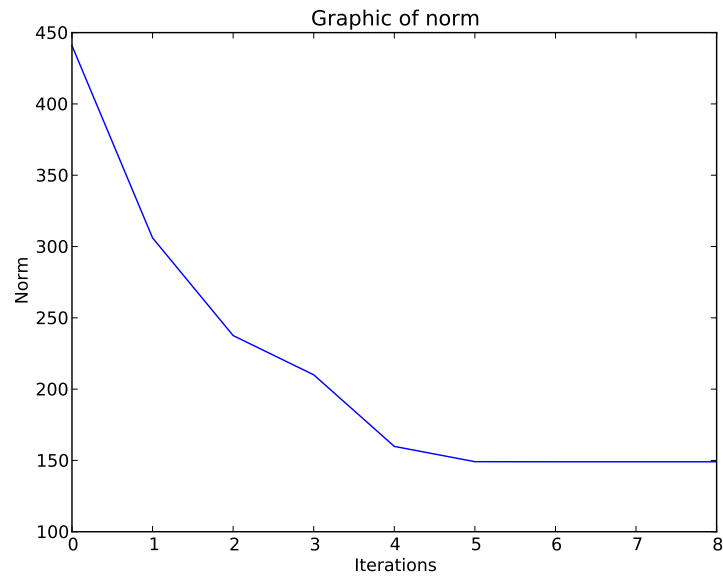
Первый цикл экспериментов призван был установить характер поведения нормы невязки

$$\|A(i_1, i_2, i_3) - \text{Approximation}(i_1, i_2, i_3)\|_2 \quad (13)$$

где $\text{Approximation}(i_1, i_2, i_3)$ — аппроксимация заданного тензора, построенная с помощью алгоритма ALS, реализованного на Python.

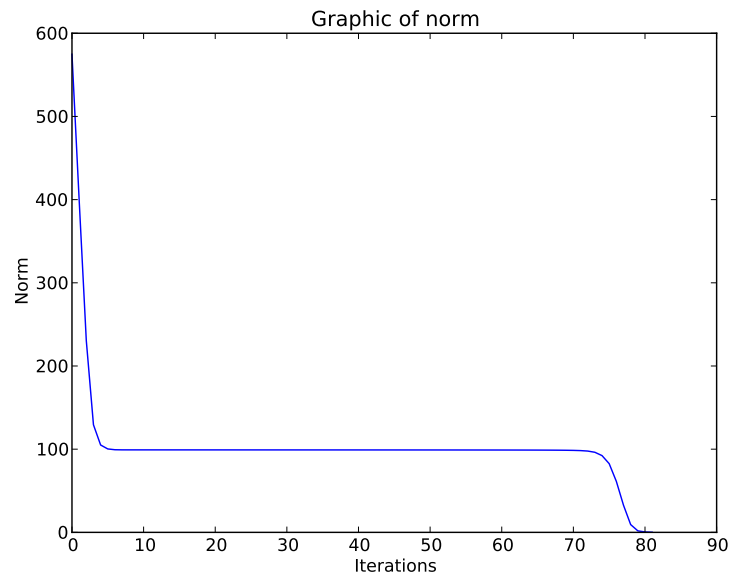
Ниже приводятся графики поведения нормы невязки в зависимости от числа итераций.

- Для случайного тензора ранга $r = 5$

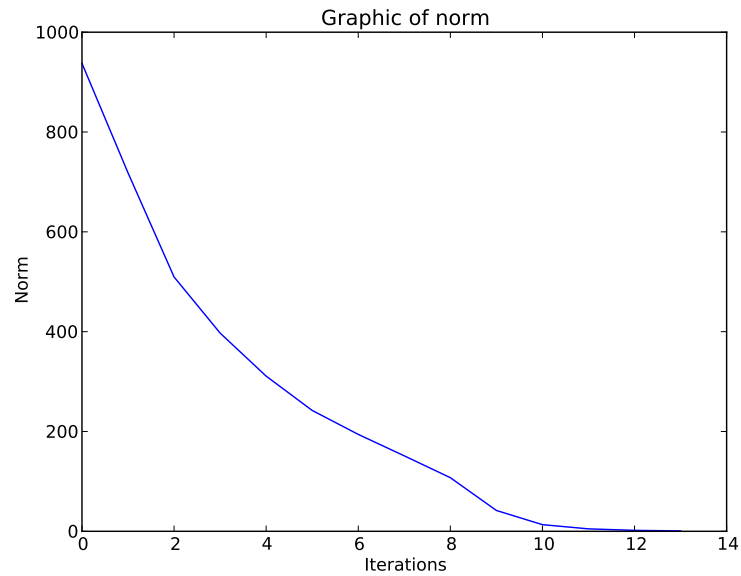


- Для случайного тензора ранга $r = 10$

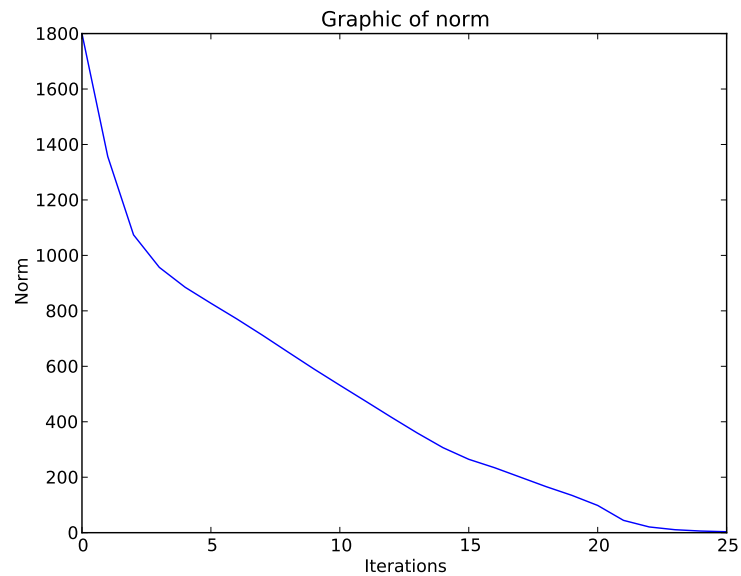
На этом примере метод попал в локальный минимум функционала (13), вследствие чего невязка убывает медленно почти на всем протяжении времени работы алгоритма. Однако миновав локальный минимум, метод сошелся очень быстро.



- Для случайного тензора ранга $r = 25$



- Для случайного тензора ранга $r = 100$



Несмотря на то, что скорость убывания невязки может варьироваться в зависимости от ранга и начального приближения, невязка убывает монотонно.

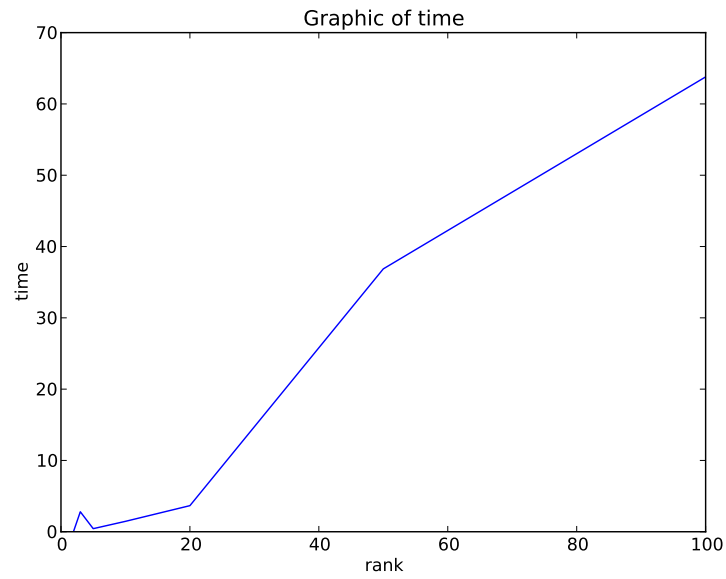
Следующая серия экспериментов показывает графическую зависимость времени выполнения программы от:

- ранга r при фиксированных размерностях тензора

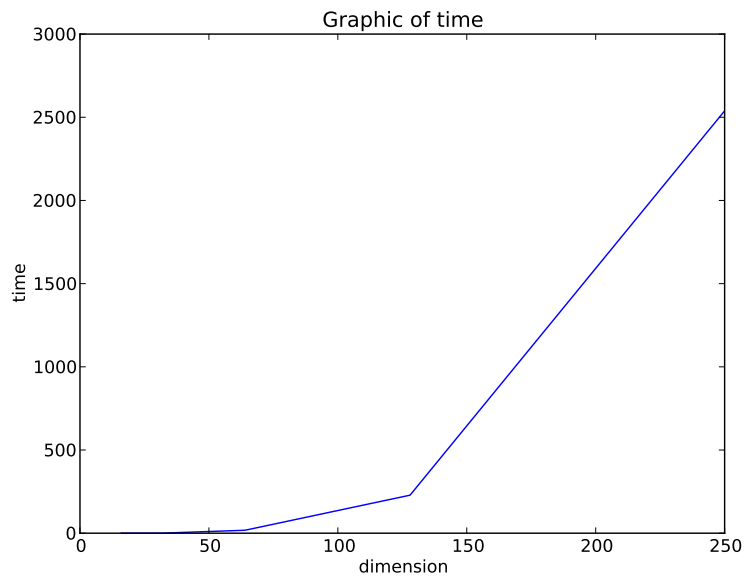
в ходе этого эксперимента размерности мод $dimension_i$ брались равными между собой и равными 32 а ранг менялся $r = 2, 3, 5, 10, 25, 50, 100$. Исходя из графика, можно сделать вывод, что время зависит от ранга как $O(r)$

- размерностей тензора n_i ($i = 1, 2, 3$) при фиксированном ранге

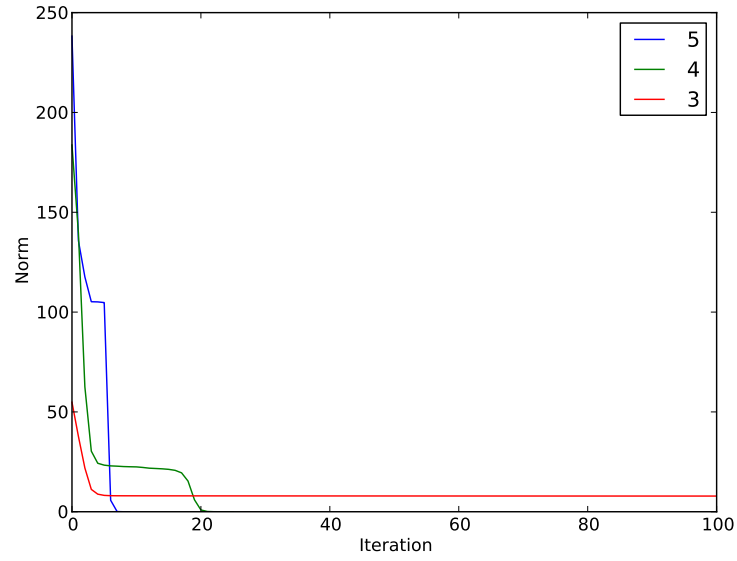
Эта серия экспериментов проводилась с целью изучения зависимости времени выполнения программы от размерностей мод $n_i = 32, 64, 128, 250, 500$ и ранге $r = 5$.



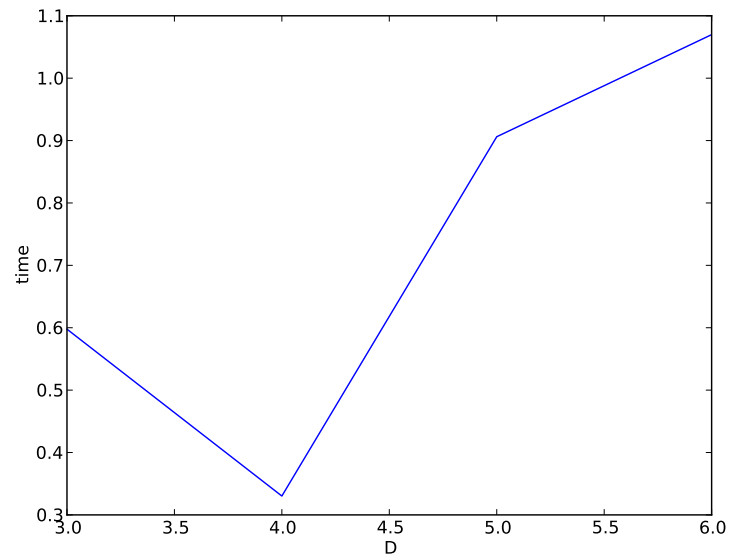
Логичным является проверить зависимость времени выполнения программы от различных размерностей мод. В данном эксперименте $dimension_i$ брались равными: 16, 32, 64, 128, 250, — а ранг r равным 5. Получена следующая зависимость:



Построим сравнительный график убывания невязки при разных размерностях тензора $d = 3, 4, 5$



И зависимость времени выполнения программы от размерности тензора d

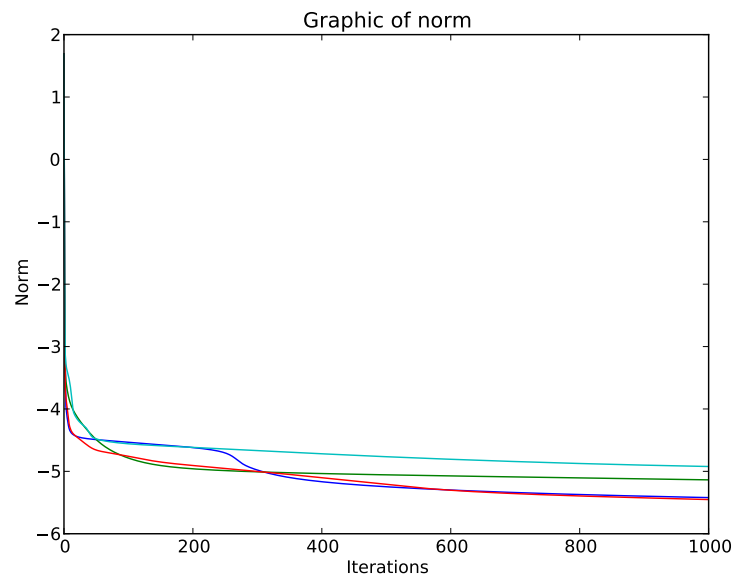


5.2 Эксперименты над неслучайными тензорами

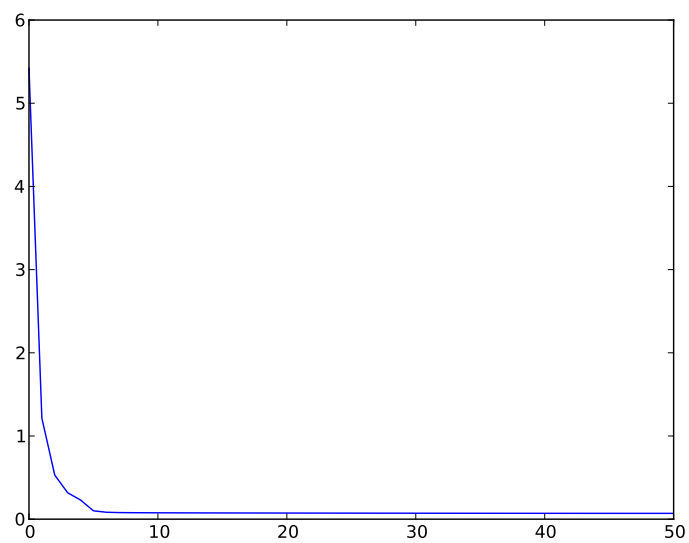
В ходе этой серии экспериментов на вход подавался тензор размерности $d = 3$ вида:

$$A[i, j, k] = \frac{1}{i + j + k + 1}, i, j, k = 1, 2, \dots, n - 1$$

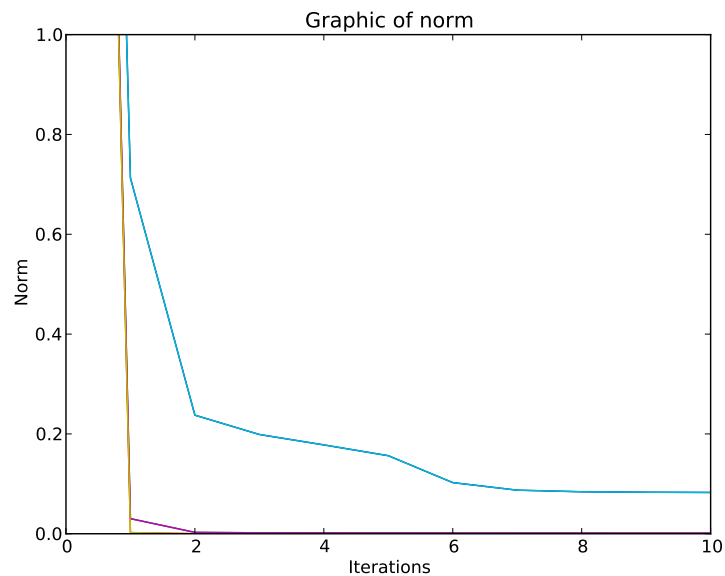
График убывания логарифма невязки с разных стартов выглядит следующим образом:



Метод сошелся за 35000 итераций (была достигнута точность 10^{-6}), однако большую часть времени он находился в области локального минимума, а невязка быстро убывала только в начале. Увеличенный график приводится ниже:

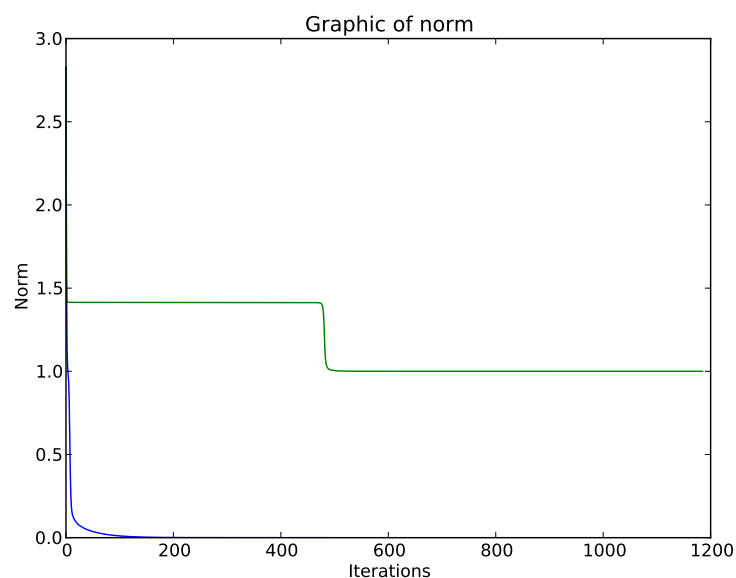


Проследим изменение скорости убывания невязки при изменении ранга. $r = 5, 20, 37$. Желтая линия соответствует рангу 37, лиловая 20, а голубая 5.



5.3 Эксперименты с тензором матричного умножения

В ходе этих экспериментов на вход подавался тензор $4 \times 4 \times 4$, получаемый из алгоритма Штрассена быстрого перемножения матриц. Как и ожидалось, метод сапроксимировал тензор при ранге $r = 7$, а на ранге 6 построить аппроксимацию не удалось. Нижеприведенный график иллюстрирует поведение невязки в ходе выполнения программы.



6 Заключение

В ходе выполнения работы была получена реализация алгоритма ALS, удовлетворяющая требованиям:

1. Независимости от размерности тензора
2. Реализации с помощью библиотек и стандартных средств языка Python

Результаты расчетов по реализации программы на Python согласуются с известными результатами.

В дальнейшем полученный опыт планируется распространить на другие форматы и функционалы: TT (TensorTrain) и на задачу интерполяции многомерных функций

Список литературы

- [1] <http://www.python.org/>.
- [2] <http://numpy.scipy.org/>.
- [3] <http://www.scipy.org/>.
- [4] J.D. Carroll and J.J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition. *Psychometrika*, 35(3):283–319, 1970.
- [5] N.K.M. Faber, R. Bro, and P.K. Hopke. Recent developments in candecomp/parafac algorithms: a critical review. *Chemometrics and Intelligent Laboratory Systems*, 65(1):119–137, 2003.
- [6] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455, 2009.