

# Каноническая аппроксимация тензоров и ее реализация на Python

Кузнецов М.А.

## 1 Введение

Аппроксимация многомерных массивов играет важную роль в приложениях. Однако вместо заданного многомерного массива часто нужно пользоваться его приближением, свойства которого известны, возможно, в отличие от заданного. Такие аппроксимации удобно строить используя следующее представление многомерного массива (*тензора*)

*Определение*

Тензором  $A$  размерности  $d$  назовем многомерный массив, элементы которого  $A(i_1, i_2, \dots, i_d)$  имеют  $d$  индексов.

*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 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  $dimension_i$  переменные

## 2.1 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.

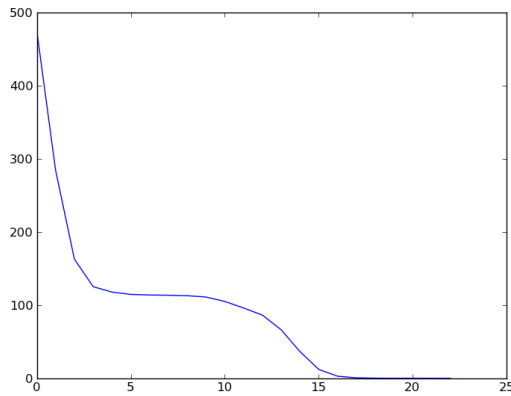
Первый цикл экспериментов призван был установить характер поведения нормы невязки

$$\max |A(i_1, i_2, i_3) - \text{Approximation}(i_1, i_2, i_3)| \quad (2)$$

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

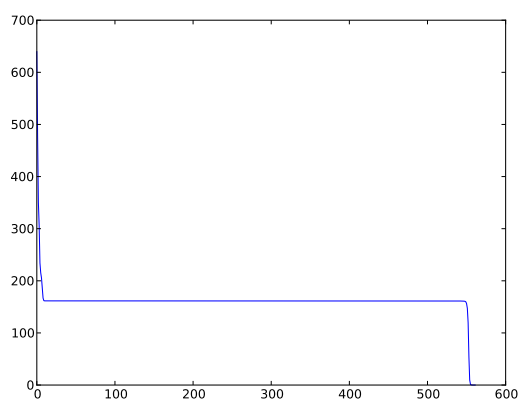
Ниже приводятся графики поведения нормы невязки (2) в зависимости от числа итераций.

- Для случайного тензора ранга  $r = 5$

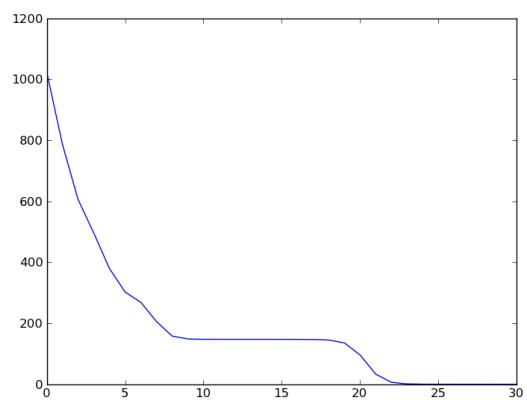


- Для случайного тензора ранга  $r = 10$

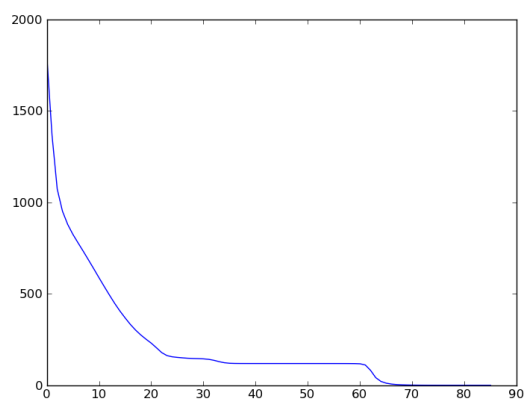
На этом примере метод попал в локальный минимум функционала (2), вследствие чего невязка убывает медленно почти на всем протяжении времени работы алгоритма. Однако миновав локальный минимум, метод сошелся очень быстро.



- Для случайного тензора ранга  $r = 25$



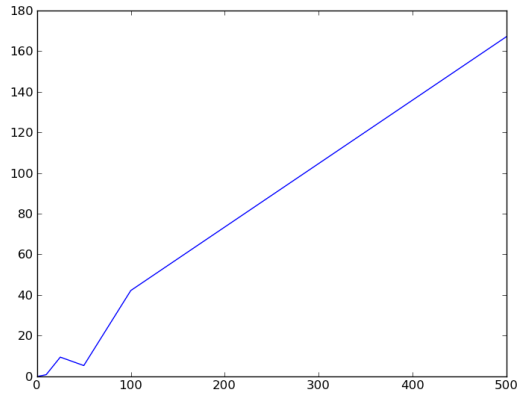
- Для случайного тензора ранга  $r = 100$



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  $r = 2, 3, 5, 10, 25, 50, 100, 500$ . Исходя из графика, можно сделать вывод, что время зависит от ранга как  $O(r)$

- размерностей тензора  $dimension_i$  ( $i = 1, \dots, 3$ ) при фиксированном ранге