Tensor Algebra
University Federal do Ceará
PPGETI - 2020.2
Saulo Mendes de Melo

Homework 10

Alternating Least Squares (ALS) Algorithm

1 Problem 1

For the third-order tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$ provided in the file "cpd_tensor.mat", implement the plain-vanilla Alternating Least Squares (ALS) algorithm that estimates the factor matrices $\mathbf{A} \in \mathbb{C}^{I \times R}$, $\mathbf{B} \in \mathbb{C}^{J \times R}$ and $\mathbf{C} \in \mathbb{C}^{K \times R}$ by solving the following problem

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}) = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \left\| \mathcal{X} - \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \right\|_F^2$$

where $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_R]$, $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_R]$, $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_R]$. Considering a sucessful run, compare the estimated matrices $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ and $\hat{\mathbf{C}}$ with the original ones. Explain the results.
Hint: An error measure at the i-th iteration can be calculated from the following formula:

$$e_{(i)} = \left\| [\mathcal{X}]_{(1)} - \hat{\mathbf{A}}_{(i)} \big(\hat{\mathbf{C}}_{(i)} \diamond \hat{\mathbf{B}}_{(i)} \big)^T \right\|_F \tag{1}$$

The convergence at the i-th iteration can be declared when $e_{(i-1)} - e_{(i)} < \delta$, where δ is a prescribed threshold value (e.g., $\delta = 10^{-6}$).

Solution

The Listing 1 has the source code of the Alternating Least Squares algorithm in Python. This function implements the standard iterative analytic solution to ALS. First we set a iterator $i = 0$ and initialize $\hat{\mathbf{A}}_0^{(2)}$ and $\hat{\mathbf{A}}_0^{(3)}$. At each step we estimate the factors according to the equations below until convergence, considering a minimum threshold of $\delta = 10^{-6}$.

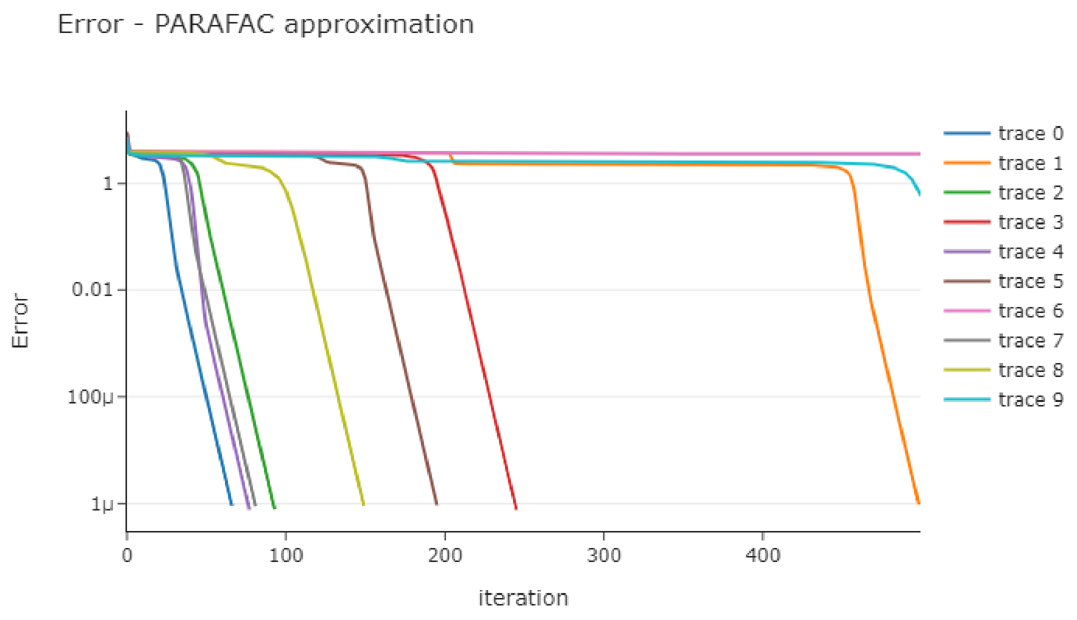
$$\begin{aligned}
\hat{\mathbf{A}}_i^{(1)} &= [\mathcal{X}]_{(1)} \big[\big(\hat{\mathbf{A}}_{i-1}^{(3)} \diamond \hat{\mathbf{A}}_{i-1}^{(2)} \big)^T \big]^\dagger \\
\hat{\mathbf{A}}_i^{(2)} &= [\mathcal{X}]_{(2)} \big[\big(\hat{\mathbf{A}}_{i-1}^{(3)} \diamond \hat{\mathbf{A}}_{i}^{(1)} \big)^T \big]^\dagger \\
\hat{\mathbf{A}}_i^{(3)} &= [\mathcal{X}]_{(3)} \big[\big(\hat{\mathbf{A}}_{i}^{(2)} \diamond \hat{\mathbf{A}}_{i}^{(1)} \big)^T \big]^\dagger
\end{aligned} \tag{2}$$

Testing with the tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$ provided in the file "cpd_tensor.mat", the ALS function has a convergence behavior that can be seen in Figure 1. We can see that for most of the tests the error plateaus for a undetermined number of iterations before descending to a reasonable threshold.

```python
def PARAFAC_ALS(X, R, max_iterations=100, thresh=1e-6):
    A_hat_2 = randn(X.shape[2], R)
    A_hat_3 = randn(X.shape[0], R)

    i = 0
    errs = []
    while i <= max_iterations:
        # A_hat_1
        A_hat_1 = np.matmul(tl.unfold(X, 1), pinv(khatri_rao(A_hat_3, A_hat_2).T))

        # A_hat_2
        A_hat_2 = np.matmul(tl.unfold(X, 2), pinv(khatri_rao(A_hat_3, A_hat_1).T))

        # A_hat_3
        A_hat_3 = np.matmul(tl.unfold(X, 0), pinv(khatri_rao(A_hat_1, A_hat_2).T))
        i += 1

        err = norm(tl.unfold(X, 1) - np.matmul(A_hat_1, khatri_rao(A_hat_3, A_hat_2).T), 'fro')
        errs.append(err)

        if err <= thresh:
            break

    return A_hat_1, A_hat_2, A_hat_3, errs
```

Listing 1: ALS function definition

Figure 1: Convergence of the error rate for ALS function



Comparing side by side the matrices $\mathbf{B}$ and $\hat{\mathbf{B}}$ from a given ALS estimation we can see that despite yielding a correct tensor estimation, they are not equal.

$$\mathbf{B} = \begin{bmatrix} 0.332 & 0.762 & 0.629 \\ -1.365 & 0.271 & 2.869 \\ 0.554 & 1.835 & -1.17 \\ -0.636 & 0.087 & -0.669 \end{bmatrix} \qquad \hat{\mathbf{B}} = \begin{bmatrix} -0.259 & 0.275 & 0.237 \\ -1.181 & 0.098 & -0.974 \\ 0.481 & 0.663 & 0.395 \\ 0.275 & 0.031 & -0.454 \end{bmatrix} \tag{3}$$

From Kruskal's condition for PARAFAC Uniqueness, we have that any triplet $(\bar{\mathbf{A}}^{(1)}, \bar{\mathbf{A}}^{(2)}, \bar{\mathbf{A}}^{(3)})$ is related to the true triplet $(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)})$ up to permutation and scaling of it's columns.

As a demonstration of this occurrence, the columns 1 and 3 from $\hat{\mathbf{B}}$ have been swapped and computing $\frac{\mathbf{B}}{\hat{\mathbf{B}}^*}$ with the swapped columns matrix $\hat{\mathbf{B}}^*$ we have a matrix that demonstrates clearly the relationship stated by Kruskal's condition. (This example was done by direct inspection of the matrix)

$$\frac{\mathbf{B}}{\hat{\mathbf{B}}^*} = \begin{bmatrix} 1.401 & 2.769 & -2.429 \\ 1.401 & 2.769 & -2.429 \\ 1.401 & 2.769 & -2.429 \\ 1.401 & 2.769 & -2.429 \end{bmatrix} \tag{4}$$

2 Problem 2

Assuming 1000 Monte Carlo experiments, generate a tensor $\mathcal{X}_0 = CPD(\mathbf{A}, \mathbf{B}, \mathbf{C})$, where $\mathbf{A} \in \mathbb{C}^{I\times R}$, $\mathbf{B} \in \mathbb{C}^{J\times R}$ and $\mathbf{C} \in \mathbb{C}^{K\times R}$ have unit norm columns with elements randomly drawn from a Normal distribution, with $R = 3$. Let $\mathcal{X} = \mathcal{X}_0 + \alpha\mathcal{V}$ be a noisy version of $\mathcal{X}_0$, where $\mathcal{V}$ is the additive noise term, whose elements are drawn from a Normal distribution. The parameter α controls the power (variance) of the noise term, and is defined as a function of the signal to noise ratio (SNR), in dB, as follows

$$SNR_{dB} = 10log_{10}\left(\frac{\|\mathcal{X}_0\|_F^2}{\|\alpha\mathcal{V}\|_F^2}\right) \tag{5}$$

Assuming the SNR range [0, 5, 10, 15, 20, 25, 30] dB, find the estimates $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ and $\hat{\mathbf{C}}$ obtained with the ALS algorithm for $(I, J, K) = (10, 4, 2)$. Let us define the normalized mean square error (NMSE) measure as follows

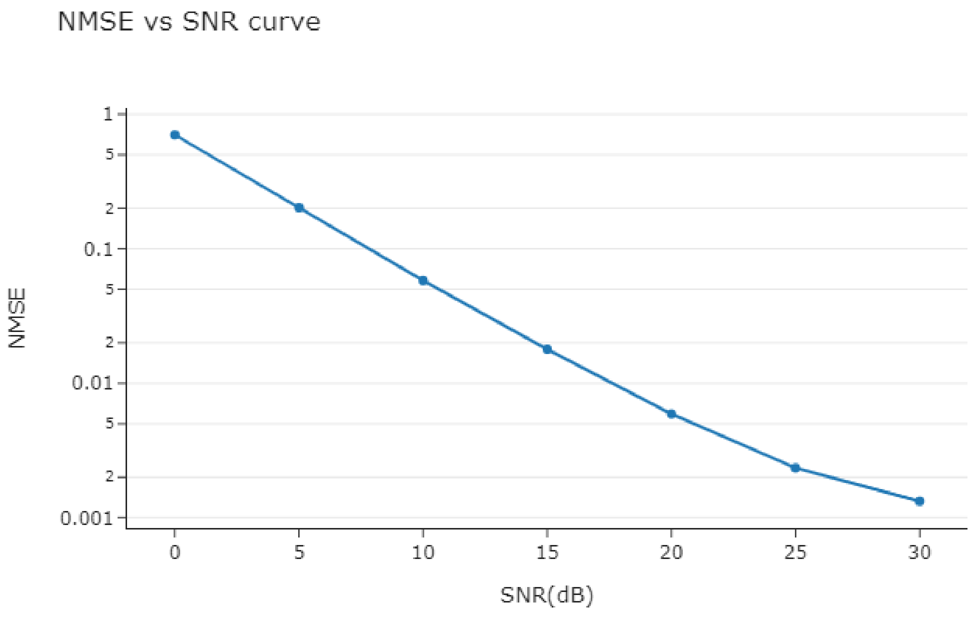
$$NMSE(\mathbf{Q}) = \frac{1}{1000}\sum_{i=1}^{1000}\frac{\|\hat{\mathbf{Q}}(i) - \mathbf{Q}(i)\|_F^2}{\|\mathbf{Q}(i)\|_F^2}, \tag{6}$$

where $\mathbf{Q}(i)$ and $\hat{\mathbf{Q}}(i)$ represent the original data matrix and the reconstructed one at the ith Monte Carlo experiment, respectively. For each SNR value plot $NMSE(\mathbf{A})$, $NMSE(\mathbf{B})$ and $NMSE(\mathbf{C})$ as a function of the SNR. Discuss the obtained results.

Solution

Listing 2 contains the implementation of the experiments.
As a result of this experiment, the graph 2 show the NMSE for each SNR value. There is a clear logarithmic decreasing trend as the noise becomes smaller and the tensor estimation becomes more correct.

Figure 2: NMSE vs. SNR Curve

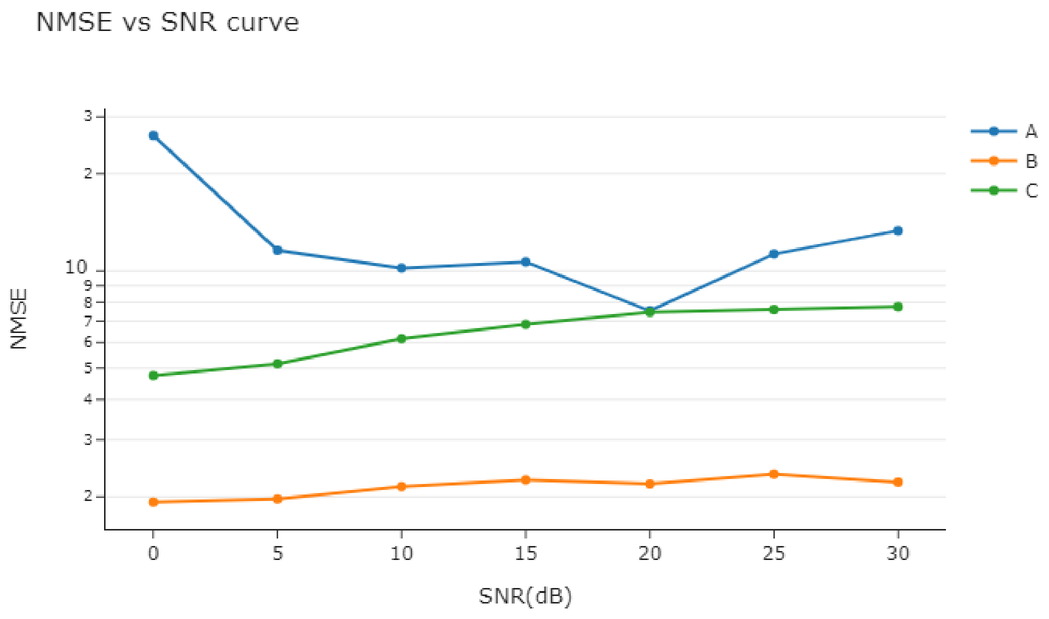


```python
I, J, K = 10, 4, 2
R = 3

SNRdb_range = [0, 5, 10, 15, 20, 25, 30]
No_experiments = 1000

NMSE = []

for SNRdb in SNRdb_range:
    exp_errors = [] # list of errors for the experiments

    for i in range(No_experiments):
        print(f"Experiment {i} with {SNRdb} SNRdb starting...")
        A = randn(I, R) + randn(I, R)*1j
        B = randn(J, R) + randn(J, R)*1j
        C = randn(K, R) + randn(K, R)*1j

        X_0 = np.matmul(A, khatri_rao(B, C).T)

        V = randn(I, J*K) + randn(I, J*K)*1j

        # calculate alpha and X
        alpha = np.sqrt((1/10**(SNRdb/10))*(norm(X_0, 'fro')**2/norm(V, 'fro')**2))
        X = X_0 + alpha*V

        # Estimante via MLS-KRF
        An, Bn, Cn, _ = PARAFAC_ALS(tl.fold(X, 0, (I, J, K)), R, max_iterations=500)

        X_0 = tl.fold(X_0, 0, (I, J, K))

        X_0_hat = np.matmul(An, khatri_rao(Cn, Bn).T)

        exp_errors.append(error(tl.unfold(X_0, 1), X_0_hat))
    print('done')

    NMSE.append(np.sum(exp_errors)/No_experiments)
```

Listing 2: Definition of other functions used in this work

We can see on Figure 3 a NMSE comparison between the original matrices $(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \bar{\mathbf{C}})$ and the estimated factors $(\mathbf{A}, \mathbf{B}, \mathbf{C})$.

Figure 3: NMSE vs. SNR Curve



There is a seemingly random error involved. This is due to the fact that the estimated factors are not guaranteed to be equal to the original ones, they are only bounded by Kruskal's condition. For $\bar{\mathbf{A}}^{(1)} = \mathbf{A}^{(1)}\mathbf{\Pi}\mathbf{\Delta}_1$, $\mathbf{\Pi}$ and $\mathbf{\Delta}_1$ can be determined by manual inspection, but no procedural way of computing this was found, and thus the factor estimations end up being quite different.