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

Homework 3

Least-Squares Khatri-Rao Factorizarion (LSKRF)

1 Implementation

Generate $\mathbf{X} = \mathbf{A} \diamond \mathbf{B} \in C^{24\times 2}$, for randomly chosen $\mathbf{A} \in C^{4\times 2}$ and $\mathbf{B} \in C^{6\times 2}$. Then implement the Least-Squares Khatri-Rao Factorization (LSKRF) algorithm that estimate $\mathbf{A}$ and $\mathbf{B}$ by solving the following problem

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \min_{\mathbf{A},\mathbf{B}} \|\mathbf{X} - \mathbf{A} \diamond \mathbf{B}\|_F^2$$

Compare the estimated matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ with the original ones. What can you conclude? Explain the results.

Results

The implementation of the Khatri-Rao product used in this work is listed in appendix 1. The matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{X}$ were generated using the code in Listing 1. The *np.random.seed()* function allows us to set a seed so that every time this code is ran, the same random matrices are generated. The *assert* statement guarantees that the Khatri-Rao product used returns a valid shape.

```
np.random.seed(0) # set a seed so that random matrices fixed

A = randn(4, 2)
B = randn(6, 2)

X = khatri_rao(A, B)

assert X.shape == (24, 2) # guarantee that the shape is correct
```

Listing 1: Generation of matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{X}$

The numbers in matrix $\mathbf{A}$ and $\mathbf{B}$ are drawn from a normal distribution.

$$A = \begin{bmatrix} 1.7641 + 1.6243j & 0.4002 - 0.6118j \\ 0.9787 - 0.5282j & 2.2409 - 1.073j \\ 1.8676 + 0.8654j & -0.9773 - 2.3015j \\ 0.9501 + 1.7448j & -0.1514 - 0.7612j \end{bmatrix} \quad B = \begin{bmatrix} -0.1032 + 0.319j & 0.4106 - 0.2494j \\ 0.144 + 1.4621j & 1.4543 - 2.0601j \\ 0.761 - 0.3224j & 0.1217 - 0.3841j \\ 0.4439 + 1.1338j & 0.3337 - 1.0999j \\ 1.4941 - 0.1724j & -0.2052 - 0.8779j \\ 0.3131 + 0.0422j & -0.8541 + 0.5828j \end{bmatrix}$$

The LSKRF implementation code is on Listing 2. In this factorization, for each of the columns $\mathbf{x}_p$ of input matrix $\mathbf{X}$, we calculate it's truncated SVD to obtain a rank-1 approximation of the matrix $\mathbf{X}_p$, which is the unvectorized form of $\mathbf{x}_p$. The first component of the truncated SVD provides us with and estimate of p-th column of $\mathbf{A}$ and $\mathbf{B}$ as

$$\hat{\mathbf{a}}_p = \sqrt{\sigma_1} \cdot \mathbf{v}_1 \qquad \text{and} \qquad \hat{\mathbf{b}}_p = \sqrt{\sigma_1} \cdot \mathbf{u}_1$$

```python
A_hat = []
B_hat = []

for i in range(X.shape[1]):
    X_p = X[:,i].reshape(4, 6).T
    U, S, Vh = svd(X_p)

    ap = np.sqrt(S[0])*Vh[0,:]
    bp = np.sqrt(S[0])*U[:,0]

    A_hat.append(ap)
    B_hat.append(bp)

A_hat = np.array(A_hat).T
B_hat = np.array(B_hat).T
```

Listing 2: LSKRF implementation

The estimated matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ have a noticeable amount of error compared to $\mathbf{A}$ and $\mathbf{B}$. Some of the values seem close, others not. One noticeable feature of the estimated matrix $\hat{\mathbf{A}}$ is that the imaginary part of the values of the first row is set to 0.

$$\hat{\mathbf{A}} = \begin{bmatrix} 1.962 + 0j & -0.6766 + 0j \\ 0.2964 - 0.8603j & -1.9666 - 1.1922j \\ 1.6037 - 0.5142j & -1.2876 + 1.9232j \\ 1.5388 + 0.5236j & -0.513 + 0.5029j \end{bmatrix} \quad \hat{\mathbf{B}} = \begin{bmatrix} -0.3569 + 0.2014j & -0.0174 + 0.5187j \\ -1.081 + 1.4339j & 1.0026 + 2.5332j \\ 0.9512 + 0.3402j & 0.2753 + 0.3371j \\ -0.5396 + 1.3869j & 0.7971 + 0.9521j \\ 1.4861 + 1.0819j & 0.915 + 0.3337j \\ 0.2465 + 0.2971j & -0.0218 - 1.1169j \end{bmatrix}$$

But when applying the Khatri-Rao Product $\hat{\mathbf{X}} = \hat{\mathbf{A}} \diamond \hat{\mathbf{B}}$ we can see that $\hat{\mathbf{X}} = \mathbf{X}$, and we conclude that the estimations are a valid factorization of $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} -0.7003+0.3951j & 0.0118-0.351j \\ -2.1209+2.8132j & -0.6784-1.714j \\ 1.8662+0.6674j & -0.1863-0.2281j \\ -1.0586+2.721j & -0.5393-0.6443j \\ 2.9157+2.1227j & -0.6191-0.2258j \\ 0.4837+0.583j & 0.0148+0.7557j \\ 0.0675+0.3668j & 0.6525-0.9994j \\ 0.9132+1.3549j & 1.0484-6.1769j \\ 0.5746-0.7175j & -0.1394-0.9912j \\ 1.0333+0.8752j & -0.4324-2.8228j \\ 1.3712-0.9579j & -1.4017-1.7471j \\ 0.3287-0.124j & -1.2886+2.2224j \\ -0.4689+0.5065j & -0.9752-0.7013j \\ -0.9963+2.8552j & -6.1627-1.3337j \\ 1.7003+0.0565j & -1.0028+0.0953j \\ -0.1522+2.5015j & -2.8575+0.3069j \\ 2.9395+0.971j & -1.8199+1.3301j \\ 0.5481+0.3498j & 2.1761+1.3962j \\ -0.6547+0.123j & -0.252-0.2748j \\ -2.4142+1.6405j & -1.7883-0.7952j \\ 1.2856+1.0215j & -0.3108-0.0345j \\ -1.5565+1.8516j & -0.8877-0.0875j \\ 1.7204+2.4431j & -0.6372+0.289j \\ 0.2238+0.5864j & 0.5729+0.5619j \end{bmatrix} \quad \hat{\mathbf{X}} = \begin{bmatrix} -0.7003+0.3951j & 0.0118-0.351j \\ -2.1209+2.8132j & -0.6784-1.714j \\ 1.8662+0.6674j & -0.1863-0.2281j \\ -1.0586+2.721j & -0.5393-0.6443j \\ 2.9157+2.1227j & -0.6191-0.2258j \\ 0.4837+0.583j & 0.0148+0.7557j \\ 0.0675+0.3668j & 0.6525-0.9994j \\ 0.9132+1.3549j & 1.0484-6.1769j \\ 0.5746-0.7175j & -0.1394-0.9912j \\ 1.0333+0.8752j & -0.4324-2.8228j \\ 1.3712-0.9579j & -1.4017-1.7471j \\ 0.3287-0.124j & -1.2886+2.2224j \\ -0.4689+0.5065j & -0.9752-0.7013j \\ -0.9963+2.8552j & -6.1627-1.3337j \\ 1.7003+0.0565j & -1.0028+0.0953j \\ -0.1522+2.5015j & -2.8575+0.3069j \\ 2.9395+0.971j & -1.8199+1.3301j \\ 0.5481+0.3498j & 2.1761+1.3962j \\ -0.6547+0.123j & -0.252-0.2748j \\ -2.4142+1.6405j & -1.7883-0.7952j \\ 1.2856+1.0215j & -0.3108-0.0345j \\ -1.5565+1.8516j & -0.8877-0.0875j \\ 1.7204+2.4431j & -0.6372+0.289j \\ 0.2238+0.5864j & 0.5729+0.5619j \end{bmatrix}$$

2 Experiments

Assuming 1000 Monte Carlo experiments, generate $\mathbf{X}_0 = \mathbf{A} \diamond \mathbf{B} \in C^{IJ \times R}$, for randomly chosen $\mathbf{A} \in C^{I \times R}$ and $\mathbf{B} \in C^{J \times R}$, with R = 4, whose elements are drawn from a normal distribution. Let $\mathbf{X} = \mathbf{X}_0 + \alpha\mathbf{V}$ be a noisy version of $\mathbf{X}_0$, where $\mathbf{V}$ 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

$$SNR_{dB} = 10log_{10}\left(\frac{\|\mathbf{X}_0\|_F^2}{\|\alpha\mathbf{V}\|_F^2}\right) \tag{1}$$

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

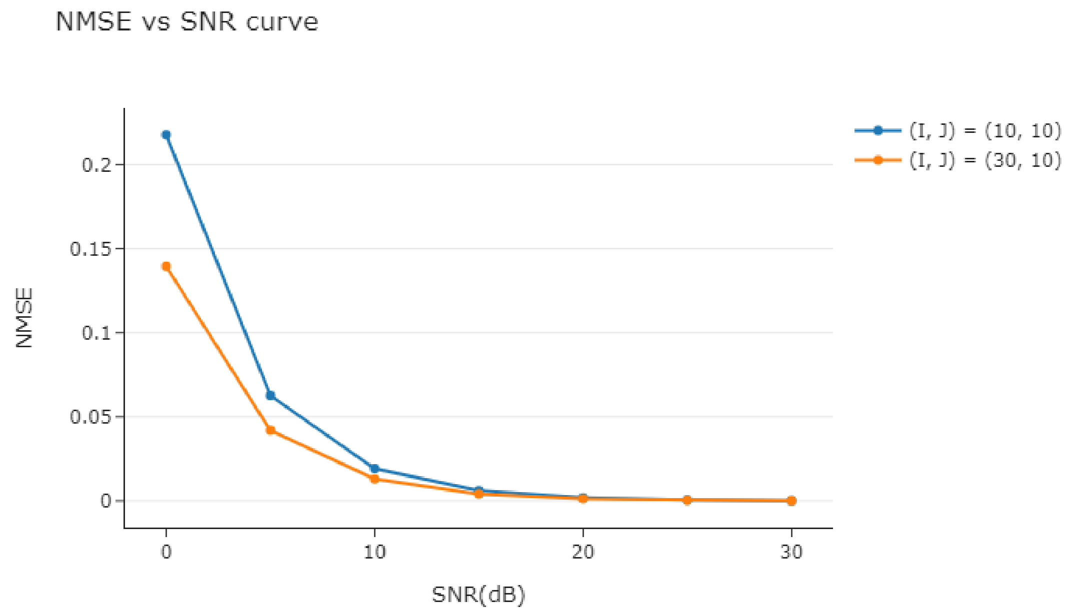
$$NMSE(\mathbf{X}_0) = \frac{1}{1000}\sum_{i=1}^{1000}\frac{\|\hat{\mathbf{X}}_0(i) - \mathbf{X}_0(i)\|_F^2}{\|\mathbf{X}_0(i)\|_F^2}, \tag{2}$$

where $\mathbf{X}_0(i)$ and $\hat{\mathbf{X}}_0(i)$ represent the original data matrix and the reconstructed one at the ith experiment, respectively. For each SNR value and configuration, plot the NMSE vs. SNR curve. Discuss the obtained results.

Results

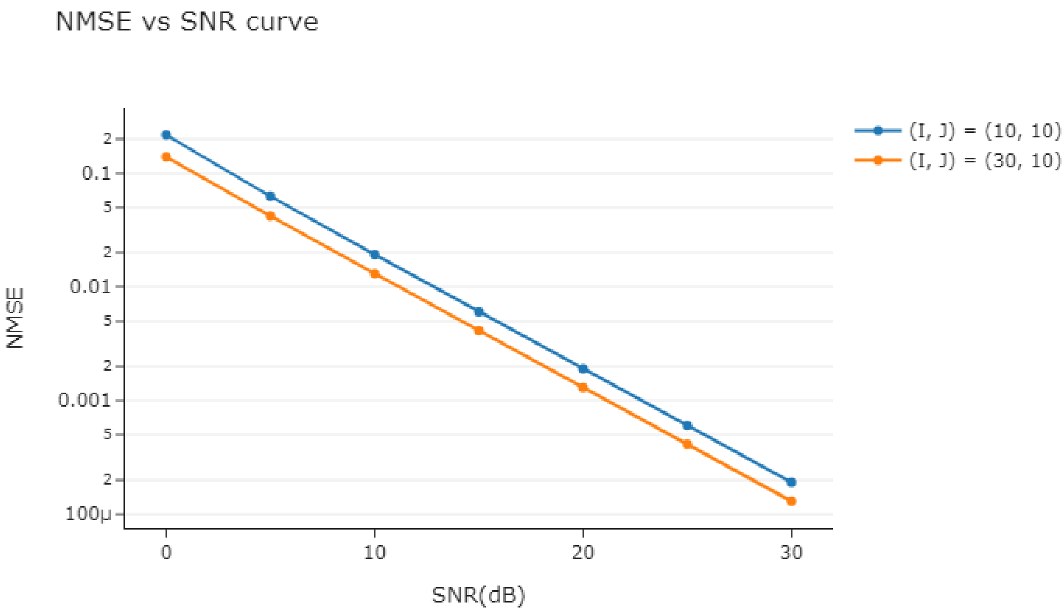
Listing 3 contains the code implementation of the experiment. Parameters I and J control the matrix configuration of inputs $\mathbf{A}$ and $\mathbf{B}$, which are randomly generated every Monte Carlo experiment. Every experiment gets it's error saved to a list, which gets its values summed and then divided by 1000, calculating the NMSE value for each SNR_{dB} value.

Figure 1: NSME vs. SNRdB curve



The results can be seen in graphic 1. A downward slope behavior is expected, since with every increment of SNR_{dB} we get less noise present in the input matrix and the estimation matrix is more precise. Figure 2 shows the same data in log scale.

Figure 2: NSME vs. SNRdB curve



There is a notable gap between the matrix configurations, with (I, J) = (10, 10) having a higher error, but both configurations descend into near-zero error. A possible explanation for the lower error in the (I, J) = (30, 10) configuration is the fact that its has more rows. The column-wise Kronecker products characteristic of the Khatri-Rao product creates subsequent rows that are linear combinations of the first rows, that makes the rank-1 truncated SVD approximation less imprecise.

```python
I, J = 30, 10
R = 4
SNRdb_range = [0, 5, 10, 15, 20, 25, 30]
No_experiments = 1000

# randn method gets random floats sampled from a univariate
# “normal” (Gaussian) distribution of mean 0 and variance 1

NMSE_3010 = []

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

    for i in range(No_experiments):

        # generating X_0 and V (noise matrix)
        A = randn(I, R)
        B = randn(J, R)
        V = randn(I*J, R)

        A = A + randn(I, R)*1j
        B = B + randn(J, R)*1j
        V = V + randn(I*J, R)*1j

        X_0 = khatri_rao(A, B)

        assert X_0.shape == (I*J, R) # guarantee that the shape is correct

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

        # LSKRF on X to estimate A_hat and B_hat
        A_hat = []
        B_hat = []

        for i in range(X.shape[1]):
            X_p = X[:,i].reshape(I, J).T
            U, S, Vh = svd(X_p)

            ap = np.sqrt(S[0])*Vh[0,:]
            bp = np.sqrt(S[0])*U[:,0]

            A_hat.append(ap)
            B_hat.append(bp)

        A_hat = np.array(A_hat).T
        B_hat = np.array(B_hat).T

        # calculating X_0_hat based on estimations
        X_0_hat = khatri_rao(A_hat, B_hat)

        # calculating error of X_0_hat estimation
        exp_errors.append(error(X_0, X_0_hat))

    # saving NMSE for this SNRdb value
    NMSE_3010.append(np.sum(exp_errors)/No_experiments)
```

Listing 3: LSKRF experiment implementation

Appendix 1: Code

```python
def khatri_rao(A, B):
    '''
    Returns the Khatri-Rao product between matrices A and B
    _______________________________________
    Inputs:
        A: Matrix with n number of columns
        B: Matrix with n number of columns
    _______________________________________
    Outputs:
        X: Khatri-Rao product of A and B
    '''
    # Verify that matrices have the same number of columns
    assert A.shape[1] == B.shape[1]

    # transposing matrices cos looping through rows is simpler in Python
    A = A.T
    B = B.T

    result = []

    for Ai in range(0, len(A)):
        for Bi in range(0, len(B)):
            if Ai == Bi:
                result.append(np.kron(A[Ai], B[Bi]))

    return np.array(result).T # transposing back to column format
```

Listing 4: Khatri-Rao Product implementation