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

Homework 4

Least-Squares Kronecker Product Factorizarion (LSKronF)

1 Implementation

Generate $\mathbf{X} = \mathbf{A} \otimes \mathbf{B} \in C^{24\times 6}$, for randomly chosen $\mathbf{A} \in C^{4\times 2}$ and $\mathbf{B} \in C^{6\times 3}$. Then implement the Least-Squares Kronecker Product Factorization (LSKronF) algorithm that estimate $\mathbf{A}$ and $\mathbf{B}$ by solving the following problem

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

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

Results

The 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 Kronecker product used returns a valid shape.

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

I, P = 4, 2
J, Q = 6, 3

A = randn(I, P)
B = randn(J, Q)

np.random.seed(1)

A = A + randn(I, P)*1j
B = B + randn(J, Q)*1j

X = kron(A, B)
```

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

The real and imaginary parts of the numbers in matrix $\mathbf{A}$ and $\mathbf{B}$ are both drawn from different normal distributions.

$$A = \begin{bmatrix} 1.7641 + 1.6243j & 0.4002 - 0.6118j \\ 0.9787 - 0.5282j & 2.2409 - 1.073j \\ 1.8676 + 0.8654j & -0.9773 - 2.3015j \\ 0.9501 + 1.7448j & -0.1514 - 0.7612j \end{bmatrix}$$

$$B = \begin{bmatrix} -0.1032 + 0.319j & 0.4106 - 0.2494j & 0.144 + 1.4621j \\ 1.4543 - 2.0601j & 0.761 - 0.3224j & 0.1217 - 0.3841j \\ 0.4439 + 1.1338j & 0.3337 - 1.0999j & 1.4941 - 0.1724j \\ -0.2052 - 0.8779j & 0.3131 + 0.0422j & -0.8541 + 0.5828j \\ -2.553 - 1.1006j & 0.6536 + 1.1447j & 0.8644 + 0.9016j \\ -0.7422 + 0.5025j & 2.2698 + 0.9009j & -1.4544 - 0.6837j \end{bmatrix}$$

The LSKronF implementation code is on Listing 2. In this factorization, the input matrix $\mathbf{X}$ has to be firstly reshaped. This reshaping is done in by separating blocks of $\mathbf{X}$ corresponding to the $a_{ij} \times \mathbf{B}$ multiplications of $\mathbf{A} \otimes \mathbf{B}$ and vectorizing them into single columns of a matrix $\tilde{\mathbf{X}}$. In Listing 2, the first calculation is done using the formula 1, and the second is done simply by splitting the matrix into vectorized blocks corresponding to the multiplications of $\mathbf{A} \otimes \mathbf{B}$. This was done merely as a way to verify if the calculation is correct, as in a real application $\mathbf{A}$ and $\mathbf{B}$ are not known.

$$\tilde{\mathbf{X}} = vec(\mathbf{B})vec(\mathbf{A})^T \tag{1}$$

This is a rank-1 matrix and it can be best approximated by truncated SVD.

$$vec(\tilde{\mathbf{A}}) = \sqrt{\sigma_1} \cdot \mathbf{v}_1 \qquad \text{and} \qquad vec(\tilde{\mathbf{B}}) = \sqrt{\sigma_1} \cdot \mathbf{u}_1$$

```python
######### X-tilde calculation ###############
# Naive version for correctness verification using vec(B).vec(A)T

X_til_naive = np.outer(B.T.reshape(1, -1), A.T.reshape(1, -1))

######### X-tilde calculation ###############
# Actual method

X_til = []

for n_matrix in np.hsplit(X, P):
    for a_B in np.vsplit(n_matrix, I):
        X_til.append(a_B.T.reshape(1, -1)[0])

X_til = np.array(X_til).T

# assert correctness
assert norm(X_til - X_til_naive) == 0


########## LSKronF ################

U, S, Vh = svd(X_til)

vec_A_hat = np.sqrt(S[0])*Vh[0,:]
vec_B_hat = np.sqrt(S[0])*U[:,0]
```

Listing 2: LSKronF implementation

The estimated matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are quite different compared to $\mathbf{A}$ and $\mathbf{B}$. This is due to the fact that there is no unique solution to this optimization problem, and it may find different matrices that satisfy it. In fact, when calculating the error $\|\tilde{\mathbf{X}} - \hat{\mathbf{A}} \otimes \hat{\mathbf{B}}\|$ we have a value of 2.85e-14, which is very near zero.

$$\hat{\mathbf{A}} = \begin{bmatrix} 2.5567 + 0j & 0.3862 - 1.1211j \\ 2.0898 - 0.67j & 2.0053 + 0.6823j \\ -0.128 - 0.7688j & 0.9827 - 2.46j \\ -2.4287 - 1.0994j & -0.6685 - 0.4877j \end{bmatrix}$$

$$\hat{\mathbf{B}} = \begin{bmatrix} -0.2739 + 0.1545j & 2.3123 - 0.4975j & -0.4141 + 1.0643j \\ 0.4162 - 0.736j & -1.0622 - 2.3814j & -0.8313 - 0.1248j \\ 0.4417 + 0.0888j & 0.7299 + 0.261j & 0.929 - 0.5469j \\ 0.1892 + 0.228j & -0.2763 + 1.2051j & 0.9937 + 2.0636j \\ -0.8295 + 1.1003j & 0.3279 - 0.1877j & 1.1404 + 0.8303j \\ -0.9596 - 0.1405j & 0.0236 + 1.1713j & -0.5691 - 1.3957j \end{bmatrix}$$

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

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

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 LSKronF algorithm for the configurations I.$(I, J) = (2, 4)$, $(P, Q) = (3, 5)$ and II.$(I, J) = (4, 8)$, $(P, Q) = (3, 5)$. Let us define the normalized mean square error (NMSE) measure as follows

$$NMSE(\mathbf{X}_0) = \frac{1}{1000}\sum_{i=1}^{1000}\frac{\|\hat{\mathbf{X}}_0(i) - \mathbf{X}_0(i)\|_F^2}{\|\mathbf{X}_0(i)\|_F^2}, \tag{3}$$

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.

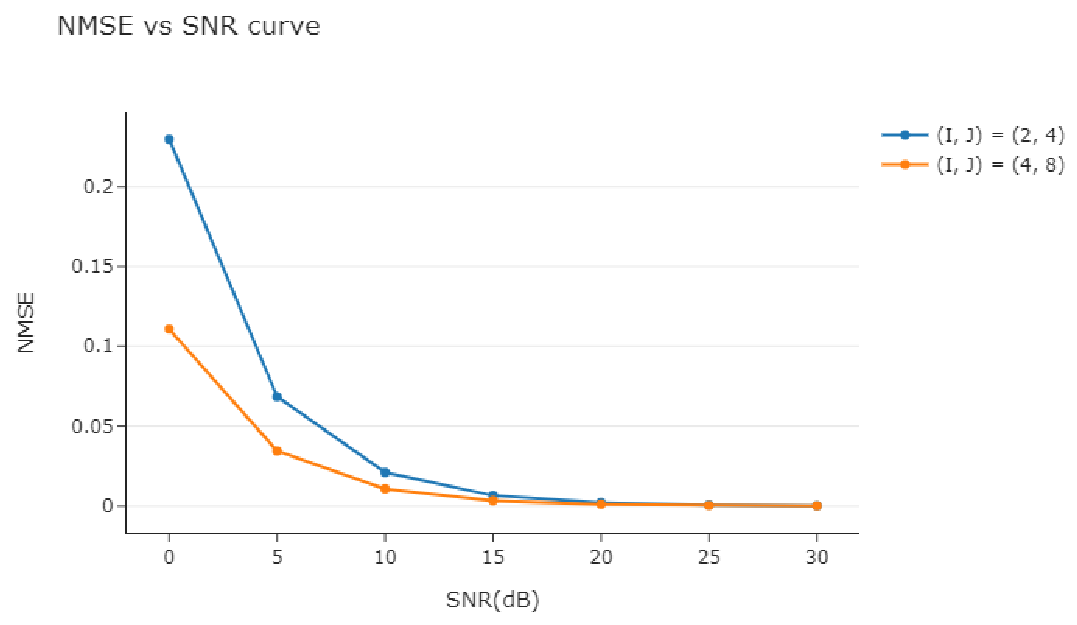
Results

With some algebraic manipulation, we can derive formula 4 for α.

$$\alpha = \frac{1}{10^{\frac{SNR_{dB}}{10}}}\frac{\|\mathbf{X}_0\|_F^2}{\|\mathbf{V}\|_F^2} \tag{4}$$

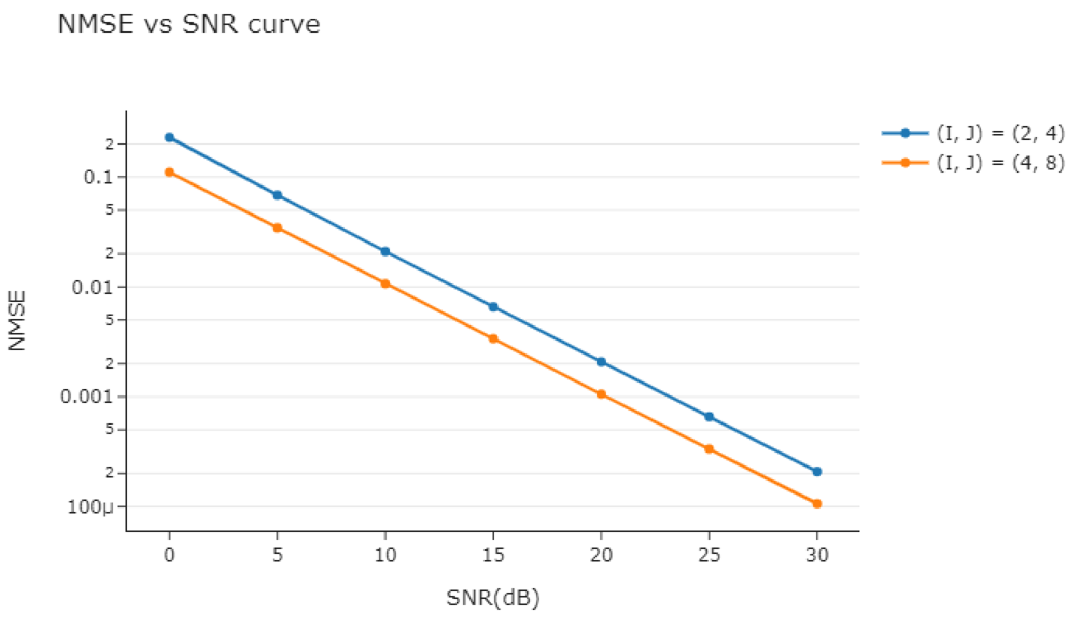
Listing 3 contains the code implementation of the experiment. Parameters I, J, P, Q control the matrix configuration of inputs $\mathbf{A}$ and $\mathbf{B}$, which are randomly generated every Monte Carlo experiment. Every experiment gets it's error saved to a list, which gets its values summed and then 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) = (2, 4) having a higher error, but both configurations descend into near-zero error. A possible explanation for the lower error in the (I, J) = (4, 8) configuration is the fact that with a bigger "**A**" matrix (the first term of the Kronecker product), the $\tilde{\mathbf{X}}$ matrix has more columns which are simply linear combinations of the form $a_{ij} \times vec(\mathbf{B})$. This means that the rank-1 approximation given by truncated SVD is precise for a bigger matrix, giving a smaller error in general.

```python
I, P = 2, 3
J, Q = 4, 5

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_2435 = []

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, P)
        B = randn(J, Q)
        V = randn(J*Q, I*P)

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

        X_0 = kron(A, B)

        # reshaping the matrix in vec(A)vec(B)^T form
        X_til = []

        for n_matrix in np.hsplit(X_0, P):
            for a_B in np.vsplit(n_matrix, I):
                X_til.append(a_B.T.reshape(1, -1)[0])

        X_til = np.array(X_til).T

        assert X_til.shape == (J*Q, I*P) # guarantee that the shape is correct

        X_0 = X_til

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

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

        U, S, Vh = svd(X)

        vec_A_hat = np.sqrt(S[0])*Vh[0,:]
        vec_B_hat = np.sqrt(S[0])*U[:,0]

        # calculating X_0_hat based on estimations
        X_0_hat = np.outer(vec_B_hat.T.reshape(1, -1), vec_A_hat.T.reshape(1, -1))

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

    # saving NMSE for this SNRdb value
    NMSE_2435.append(np.sum(exp_errors)/No_experiments)
```

Listing 3: LSKRF experiment implementation