Computer Homework 4

Matrix and Tensor Decompositions

Dr. Sepideh Hajipour

Sharif University of Technology

Electrical Engineering

Nikoo Moradi

400101934

Date: July 2, 2024

Contents

1

This is the pseudo code of my implementation of the ALS algorithm for the CP decomposition of a third-order tensor.

1.A ALS Algorithm (cp_als)

Algorithm 1 cp_als(T, U1_0, U2_0, U3_0, maxIter, tol)

Initialize: U1 = U1_0, U2 = U2_0, U3 = U3_0
for iter = 1 to maxIter **do**
    Compute Khatri-Rao product kr_U3_U2 = kr(U3, U2)
    Unfold tensor T along mode-1, T1 = unfold(T, 1)
    Update U1: U1_new = (T1 × kr_U3_U2) / (kr_U3_U2' × kr_U3_U2)
    Compute Khatri-Rao product kr_U3_U1 = kr(U3, U1_new)
    Unfold tensor T along mode-2, T2 = unfold(T, 2)
    Update U2: U2_new = (T2 × kr_U3_U1) / (kr_U3_U1' × kr_U3_U1)
    Compute Khatri-Rao product kr_U2_U1 = kr(U2_new, U1_new)
    Unfold tensor T along mode-3, T3 = unfold(T, 3)
    Update U3: U3_new = (T3 × kr_U2_U1) / (kr_U2_U1' × kr_U2_U1)
    if norm(U1_new - U1, 'fro') < tol and norm(U2_new - U2, 'fro') < tol and norm(U3_new - U3, 'fro') < tol **then**
        break
    end if
    Update U1 = U1_new, U2 = U2_new, U3 = U3_new
end for
return {U1, U2, U3}

1.B Unfolding Tensor (unfold)

Algorithm 2 unfold(T, mode)

Input: Tensor T, mode
Output: Matrix T_unfold
sz = size(T)
if mode == 1 **then**
    T_unfold = reshape(permute(T, [1, 2, 3]), sz(1), [])
else if mode == 2 **then**
    T_unfold = reshape(permute(T, [2, 1, 3]), sz(2), [])
else if mode == 3 **then**
    T_unfold = reshape(permute(T, [3, 1, 2]), sz(3), [])
end if
return T_unfold

1.C Khatri-Rao Product (kr)

Algorithm 3 kr(A, B)

```
Input: Matrices A, B
Output: Matrix K
[I, R] = size(A)
[J, _] = size(B)
K = zeros(I * J, R)
for r = 1 to R do
    K(:, r) = kron(A(:, r), B(:, r))
end for
return K
```

To check the functionality, I tested my CP_ALS algorithm on a random tensor T, reconstruct T from the outputs of the function, and then measure the difference using the `TMSE function` which was provided.

- TMSE = 1.9 +- 0.2 which is good enough.

```
TMSE = 2.0102  TMSE = 1.7085  TMSE = 1.9128  TMSE = 2.052
```

Figure 1: TMSE for different attempts on using CP_ALS function

2

Part A and B

In this part, we apply different algorithms for CP decomposition (including my function from the previous question), and we compare them. The algorithms compared are:

1. ALS with TensorLab
2. Minf with TensorLab
3. SD with TensorLab
4. Custom CP ALS implementation

Thirty third-order tensors of dimensions $5 \times 5 \times 5$ were generated with random factor matrices $U_{\text{org}}^{(1)}$, $U_{\text{org}}^{(2)}$, and $U_{\text{org}}^{(3)}$. Gaussian noise was added to the tensors at SNR levels of 0, 20, 40, and 60 dB. The CP decomposition was performed using the four algorithms, both with random initialization and initialization using Higher Order Singular Value Decomposition (HOSVD). The Total Mean Squared Error (TMSE) was computed for each decomposition, and the results were averaged across the 30 tensors.

Results

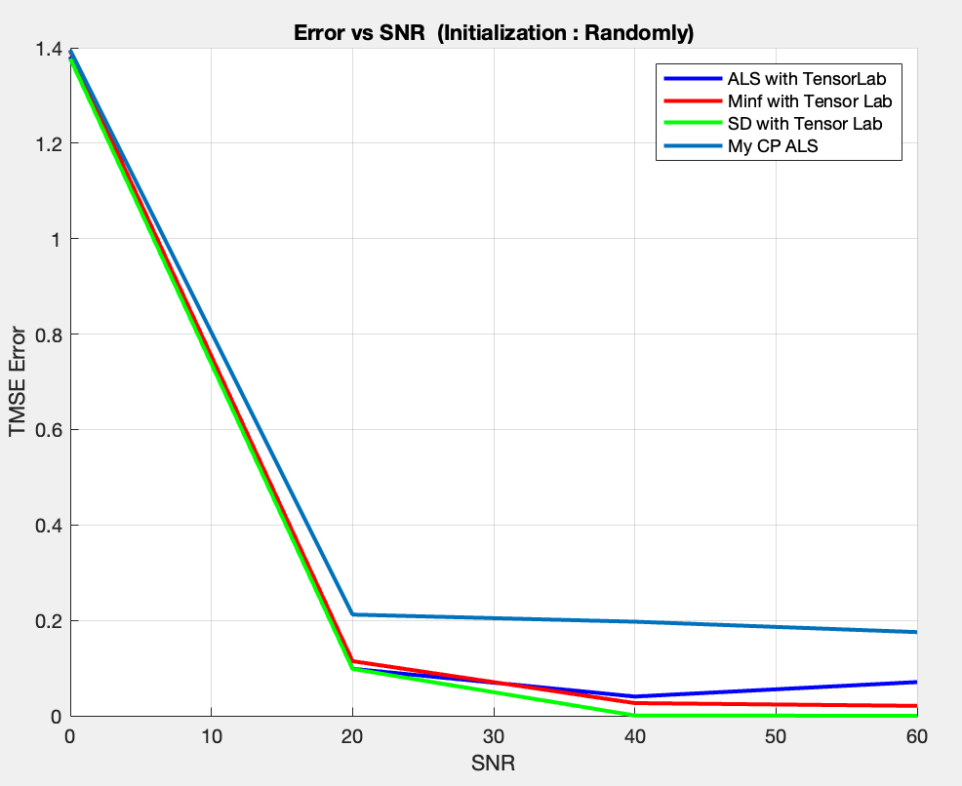

Figure 2: Error vs SNR (Initialization: Randomly)

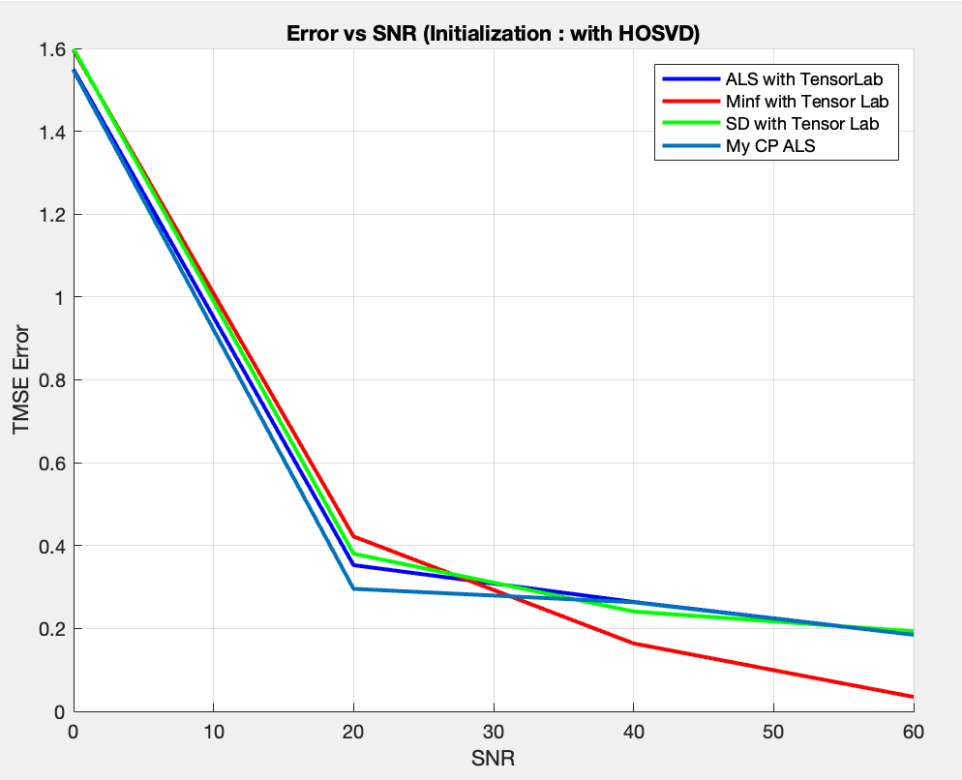

Figure 3: Error vs SNR (Initialization: with HOSVD)

```
Initialization randomly
SNR: 0 dB
CPD: TMSE = 1.38e+00 ± 4.58e-01
CPD_MINF: TMSE = 1.39e+00 ± 4.60e-01
CPD3_SD: TMSE = 1.38e+00 ± 3.62e-01
My CP ALS: TMSE = 1.40e+00 ± 4.38e-01

Initialization with HOSVD
SNR: 0 dB
CPD: TMSE = 1.55e+00 ± 4.02e-01
CPD_MINF: TMSE = 1.60e+00 ± 4.82e-01
CPD3_SD: TMSE = 1.60e+00 ± 4.25e-01
My CP ALS: TMSE = 1.55e+00 ± 4.13e-01
```

```
Initialization randomly
SNR: 20 dB
CPD: TMSE = 9.85e-02 ± 2.89e-01
CPD_MINF: TMSE = 1.14e-01 ± 2.82e-01
CPD3_SD: TMSE = 9.82e-02 ± 2.00e-01
My CP ALS: TMSE = 2.12e-01 ± 4.53e-01

Initialization with HOSVD
SNR: 20 dB
CPD: TMSE = 3.53e-01 ± 3.31e-01
CPD_MINF: TMSE = 4.22e-01 ± 3.85e-01
CPD3_SD: TMSE = 3.80e-01 ± 3.35e-01
My CP ALS: TMSE = 2.96e-01 ± 3.14e-01
```

Figure 4: TMSE for SNR 0 dB and SNR 20 dB

```
Initialization randomly
SNR: 40 dB
CPD: TMSE = 4.03e-02 ± 2.19e-01
CPD_MINF: TMSE = 2.66e-02 ± 1.43e-01
CPD3_SD: TMSE = 5.87e-04 ± 7.48e-04
My CP ALS: TMSE = 1.97e-01 ± 3.93e-01
```

```
Initialization randomly
SNR: 60 dB
CPD: TMSE = 7.07e-02 ± 2.69e-01
CPD_MINF: TMSE = 2.08e-02 ± 1.14e-01
CPD3_SD: TMSE = 5.26e-06 ± 6.51e-06
My CP ALS: TMSE = 1.75e-01 ± 3.40e-01
```

```
Initialization with HOSVD
SNR: 40 dB
CPD: TMSE = 2.64e-01 ± 2.70e-01
CPD_MINF: TMSE = 1.64e-01 ± 3.70e-01
CPD3_SD: TMSE = 2.41e-01 ± 2.76e-01
My CP ALS: TMSE = 2.63e-01 ± 2.69e-01
```

```
Initialization with HOSVD
SNR: 60 dB
CPD: TMSE = 1.85e-01 ± 2.73e-01
CPD_MINF: TMSE = 3.48e-02 ± 1.91e-01
CPD3_SD: TMSE = 1.93e-01 ± 2.28e-01
My CP ALS: TMSE = 1.85e-01 ± 2.73e-01
```

Figure 5: TMSE for SNR 40 dB and SNR 60 dB

- Initialization is important in the performance of the algorithms. HOSVD initialization provides better performance compared to random initialization.
- The TensorLab implementations generally outperform the custom CP ALS implementation, but the custom method shows competitive results with appropriate initialization.
- Randomly Initialized performance ranking:
  - Minf & SD
  - ALS
  - custom ALS
- HOSVD Initialized performance ranking:
  - Minf
  - SD & ALS & custom ALS

Part C and D

In this part, everything is pretty much the same but with only one difference that is $U_{\text{org}}^{(1)}$, $U_{\text{org}}^{(2)}$, and $U_{\text{org}}^{(3)}$ are not independent from each other, we generate them as follows:

Algorithm 4 Pseudocode for Matrix Operations

$u1\_1 \leftarrow \text{randn}(I, 1)$
$v1\_2 \leftarrow \text{randn}(I, 1)$
$u1\_2 \leftarrow u1\_1 + 0.5 \times v1\_2$
$u1\_3 \leftarrow \text{randn}(I, 1)$
$U1\_org \leftarrow [u1\_1, u1\_2, u1\_3]$
$U3\_org \leftarrow \text{randn}(K, R)$
$u2\_1 \leftarrow \text{randn}(J, 1)$
$v2\_2 \leftarrow \text{randn}(J, 1)$
$u2\_2 \leftarrow u2\_1 + 0.5 \times v2\_2$
$u2\_3 \leftarrow \text{randn}(J, 1)$
$U2\_org \leftarrow [u2\_1, u2\_2, u2\_3]$
$U\_org \leftarrow \{U1\_org, U2\_org, U3\_org\}$
$T \leftarrow \text{zeros}(I, J, K)$
for $r \leftarrow 1$ **to** R **do**
    $T \leftarrow T + \text{outer\_product}(U1\_org(:, r), U2\_org(:, r), U3\_org(:, r))$
end for

Results

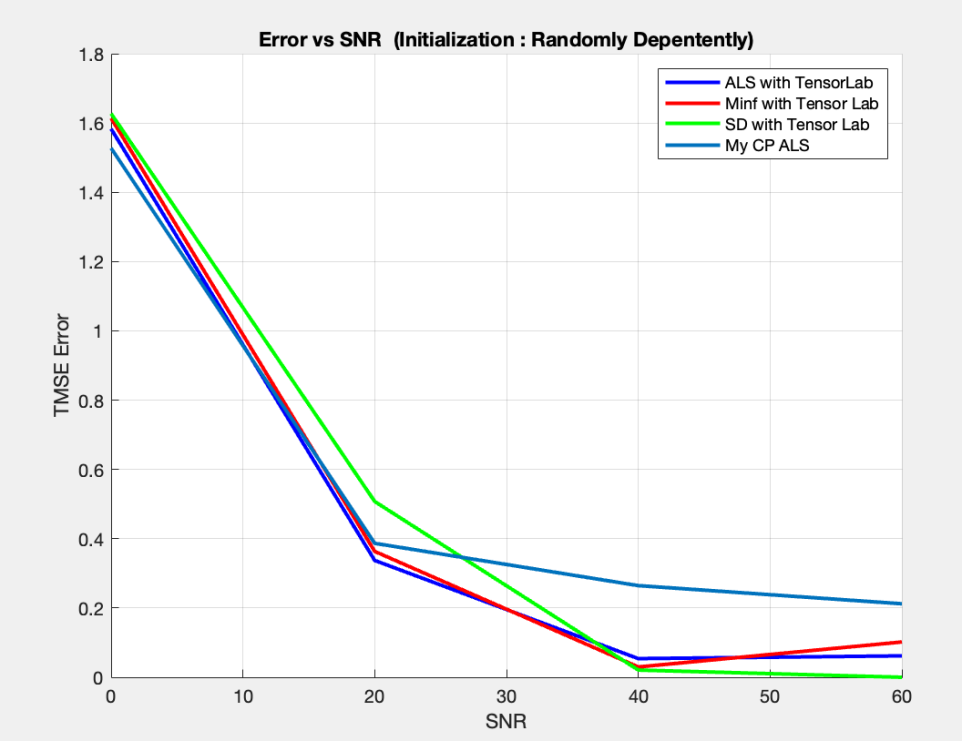

Figure 6: Error vs SNR (Initialization: Randomly, Depentantly)

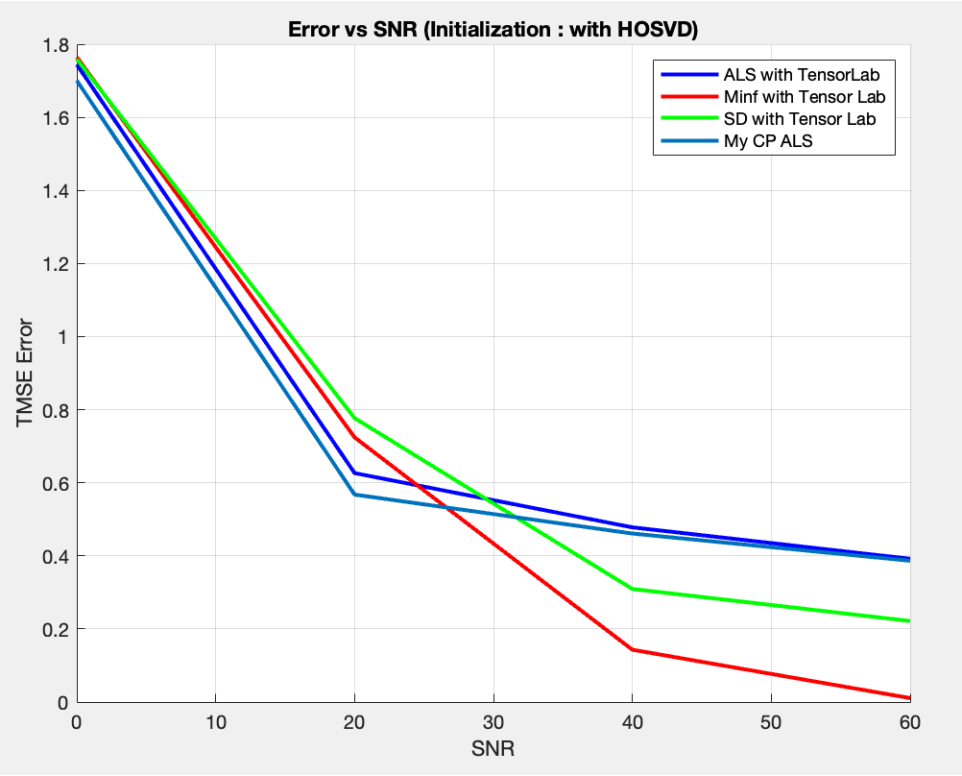

Figure 7: Error vs SNR (Initialization: with HOSVD)

```
Initialization randomly
SNR: 0 dB
CPD: TMSE = 1.38e+00 ± 4.58e-01
CPD_MINF: TMSE = 1.39e+00 ± 4.60e-01
CPD3_SD: TMSE = 1.38e+00 ± 3.62e-01
My CP ALS: TMSE = 1.40e+00 ± 4.38e-01
```

```
Initialization randomly dependently
SNR: 20 dB
CPD: TMSE = 3.37e-01 ± 4.03e-01
CPD_MINF: TMSE = 3.63e-01 ± 4.33e-01
CPD3_SD: TMSE = 5.07e-01 ± 4.23e-01
My CP ALS: TMSE = 3.87e-01 ± 3.81e-01
```

```
Initialization with HOSVD
SNR: 0 dB
CPD: TMSE = 1.55e+00 ± 4.02e-01
CPD_MINF: TMSE = 1.60e+00 ± 4.82e-01
CPD3_SD: TMSE = 1.60e+00 ± 4.25e-01
My CP ALS: TMSE = 1.55e+00 ± 4.13e-01
```

```
Initialization with HOSVD
SNR: 20 dB
CPD: TMSE = 6.26e-01 ± 4.44e-01
CPD_MINF: TMSE = 7.24e-01 ± 5.34e-01
CPD3_SD: TMSE = 7.77e-01 ± 3.99e-01
My CP ALS: TMSE = 5.68e-01 ± 3.77e-01
```

Figure 8: Enter Caption

```
Initialization randomly dependently
SNR: 40 dB
CPD: TMSE = 5.39e-02 ± 1.93e-01
CPD_MINF: TMSE = 2.98e-02 ± 7.99e-02
CPD3_SD: TMSE = 2.08e-02 ± 7.58e-02
My CP ALS: TMSE = 2.65e-01 ± 3.50e-01
```

```
Initialization randomly dependently
SNR: 60 dB
CPD: TMSE = 6.18e-02 ± 2.33e-01
CPD_MINF: TMSE = 1.02e-01 ± 2.66e-01
CPD3_SD: TMSE = 6.87e-05 ± 2.15e-04
My CP ALS: TMSE = 2.12e-01 ± 2.99e-01
```

```
Initialization with HOSVD
SNR: 40 dB
CPD: TMSE = 4.78e-01 ± 3.11e-01
CPD_MINF: TMSE = 1.43e-01 ± 2.83e-01
CPD3_SD: TMSE = 3.09e-01 ± 2.32e-01
My CP ALS: TMSE = 4.61e-01 ± 2.92e-01
```

```
Initialization with HOSVD
SNR: 60 dB
CPD: TMSE = 3.92e-01 ± 2.41e-01
CPD_MINF: TMSE = 1.07e-02 ± 5.84e-02
CPD3_SD: TMSE = 2.22e-01 ± 1.85e-01
My CP ALS: TMSE = 3.86e-01 ± 2.40e-01
```

Figure 9: Enter Caption

- In comparison to what we did in part A and B, we get higher TMSE which is because of the fact that $U^{(i)}_{\text{org}}$s are dependent.
- Randomly Initialized performance ranking(Dependably):
  - SD
  - ALS & Minf
  - custom ALS
- HOSVD Initialized performance ranking(Dependably):
  - Minf
  - SD
  - ALS & custom ALS

3

3.A

The algorithms compared are:

1. ALS with TensorLab
2. Minf with TensorLab
3. SD with TensorLab
4. Custom CP ALS implementation

- Rank 2: All algorithms detected 2 factors.

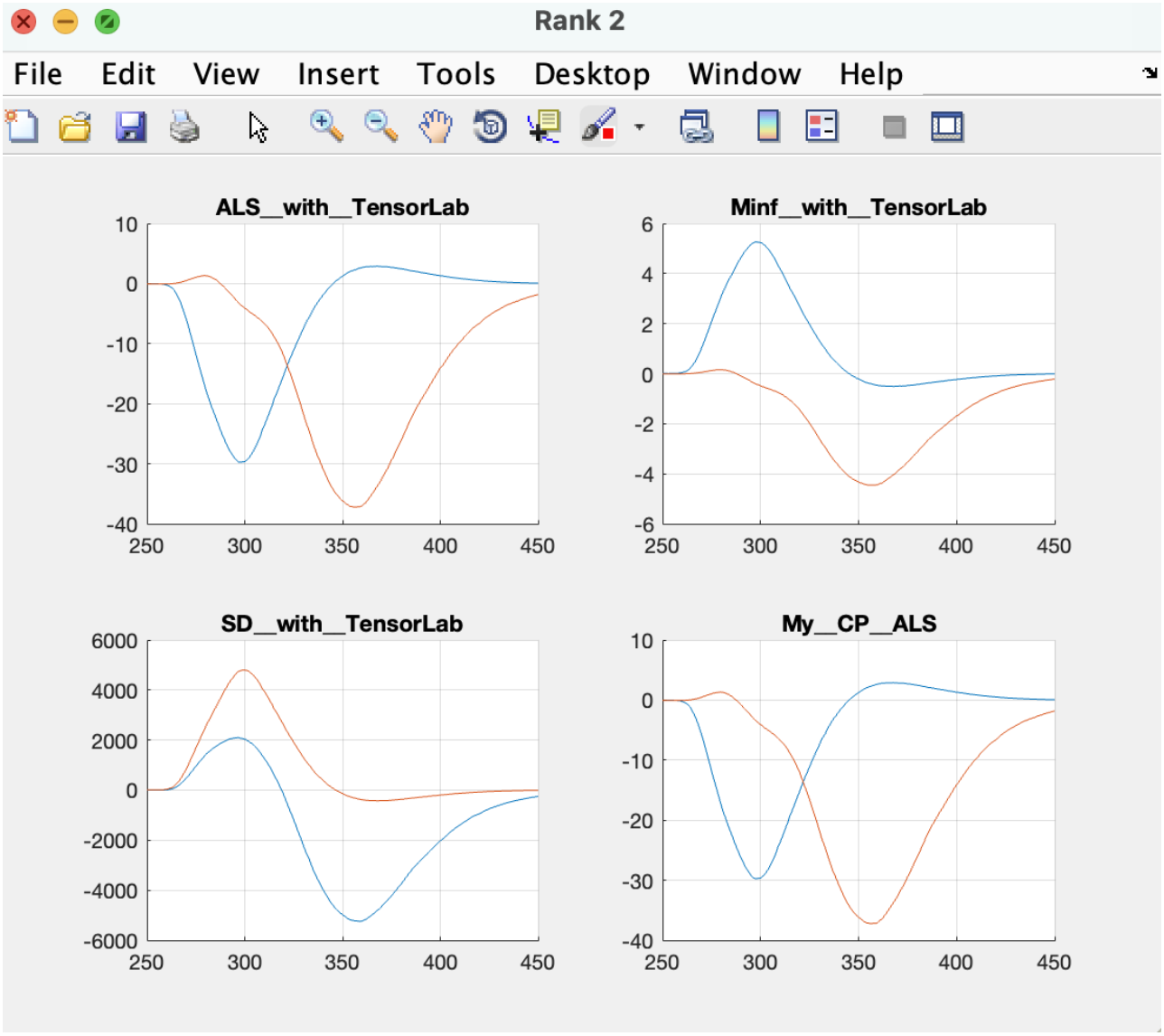

Figure 10: Rank 2

- Rank 3: All algorithms successfully detected 3 factors.

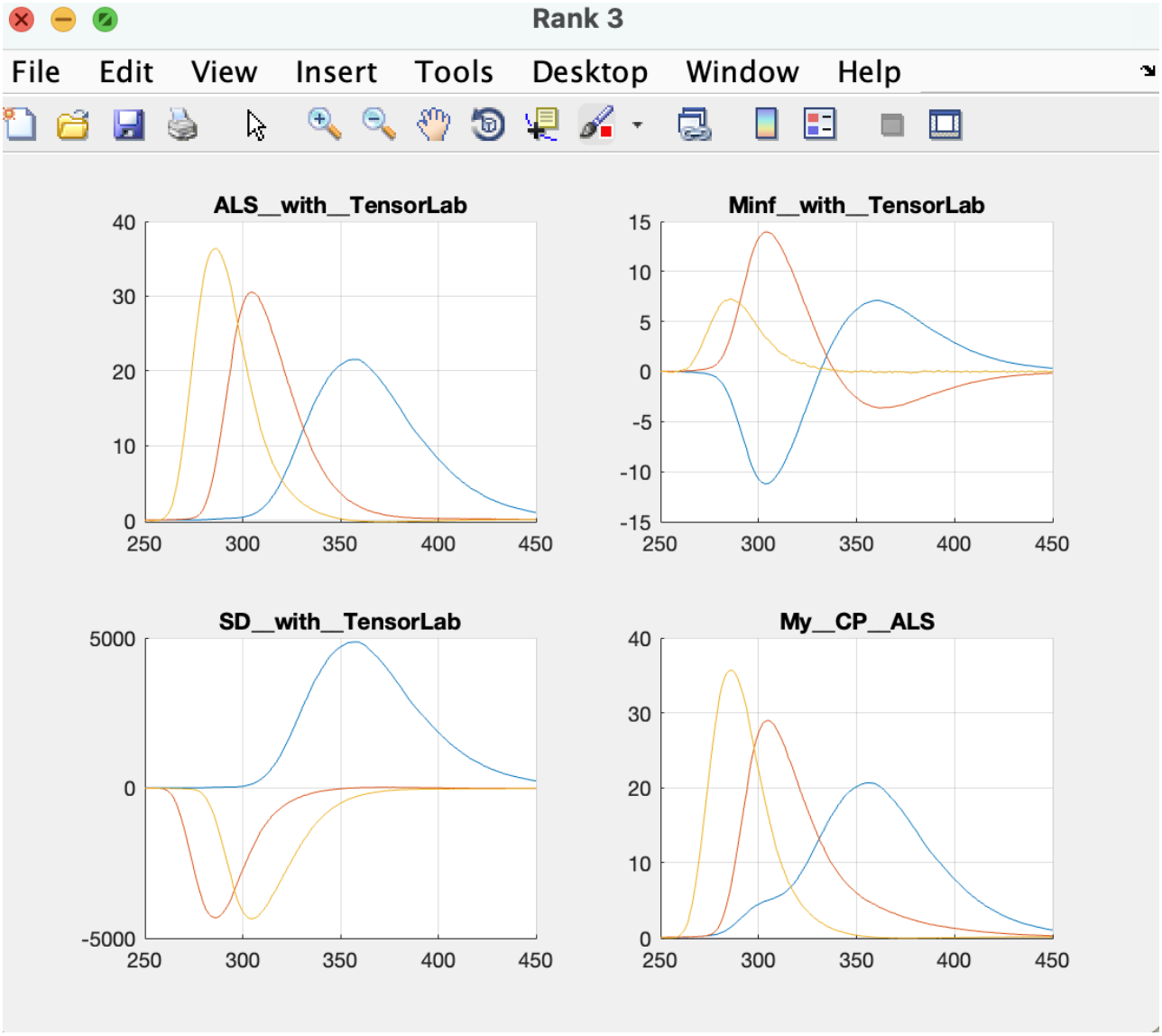

Figure 11: Rank 3

- Rank 4: Only SD algorithm detected 3 factors, others detected 4 factors which is wrong.

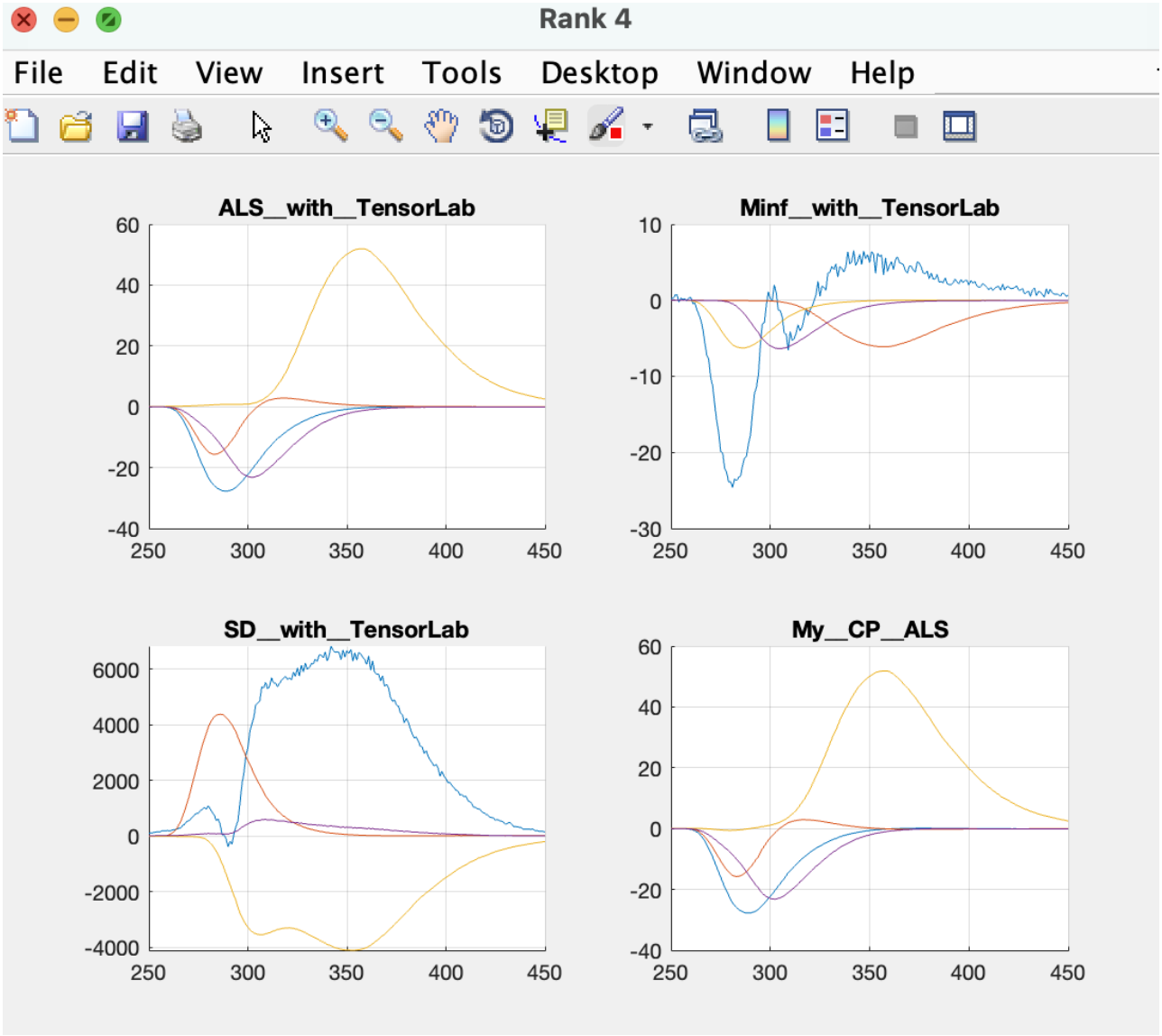

Figure 12: Rank 4

- Rank 5: Only SD algorithm detected 3 factors, others detected 5 factors which is wrong.

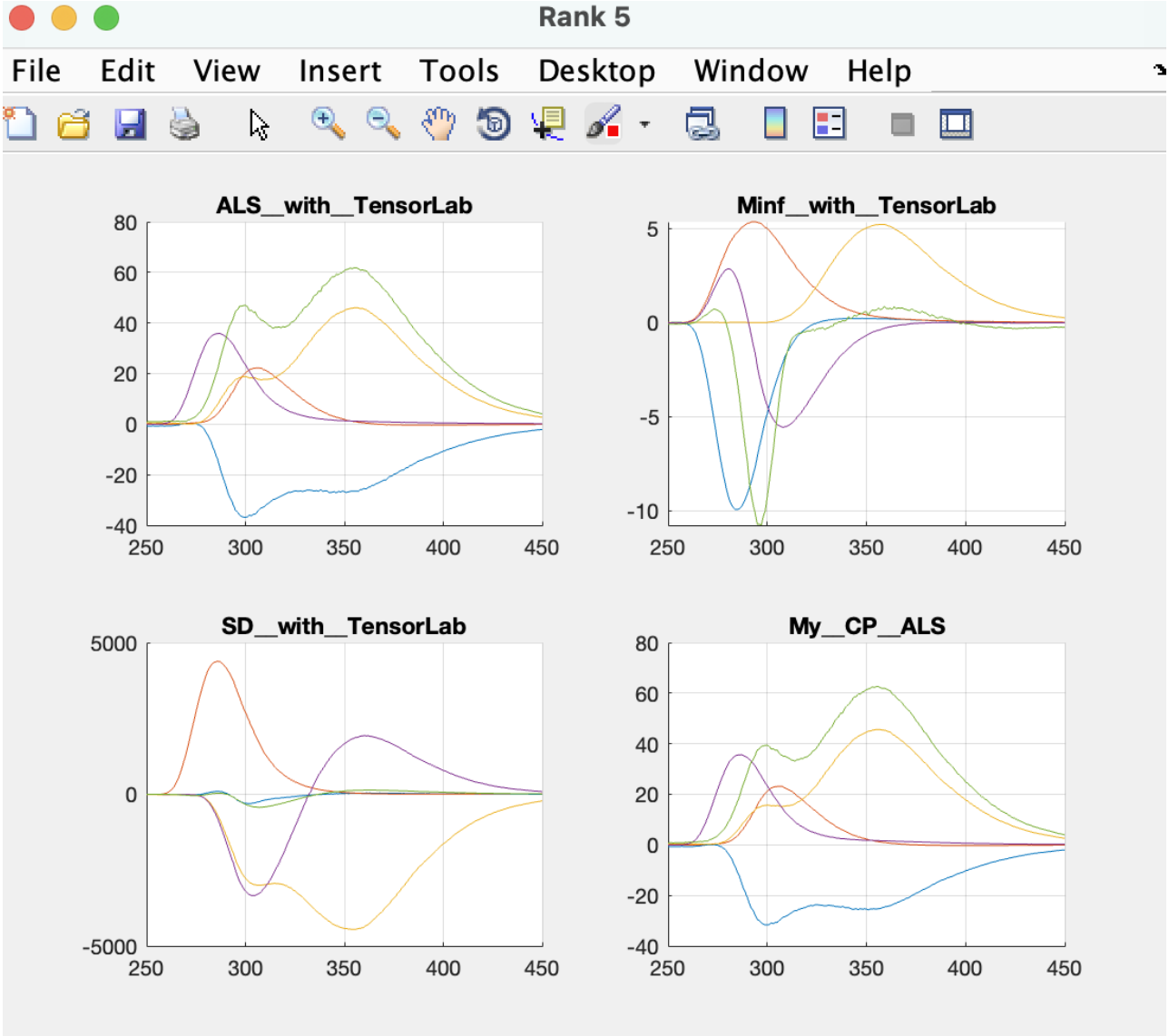

Figure 13: Rank 5

3.B

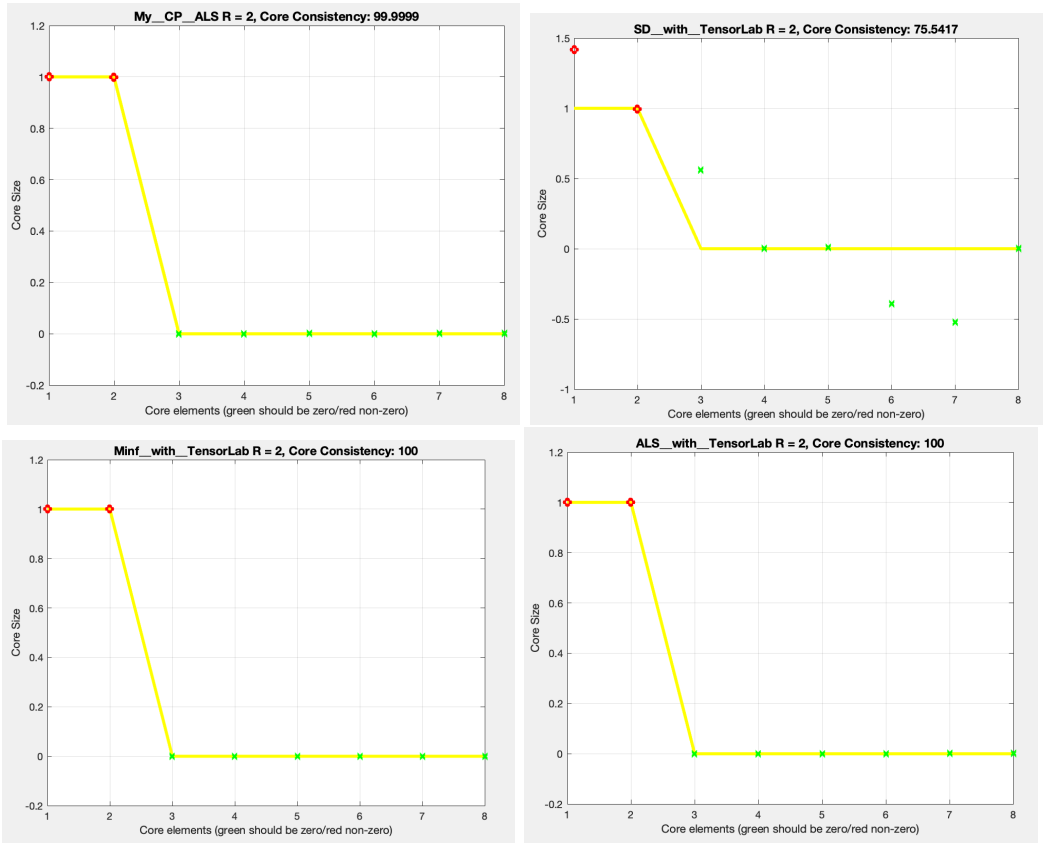

Figure 14: Rank 2

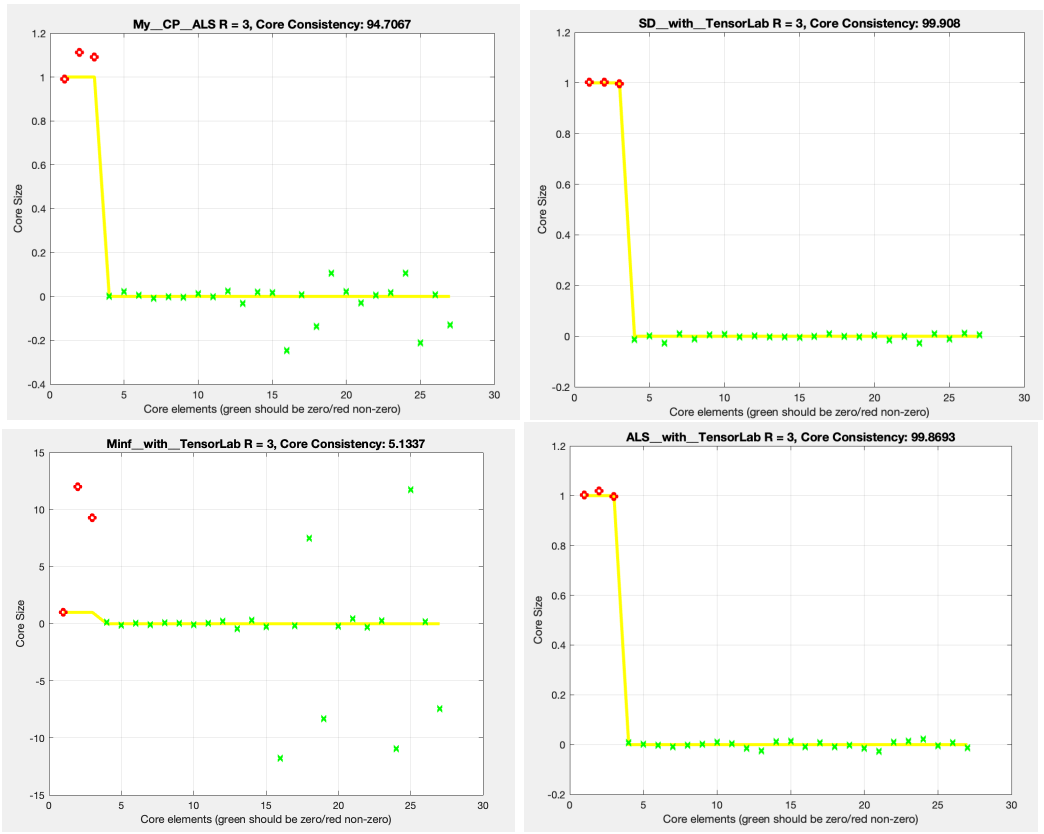

Figure 15: Rank 3

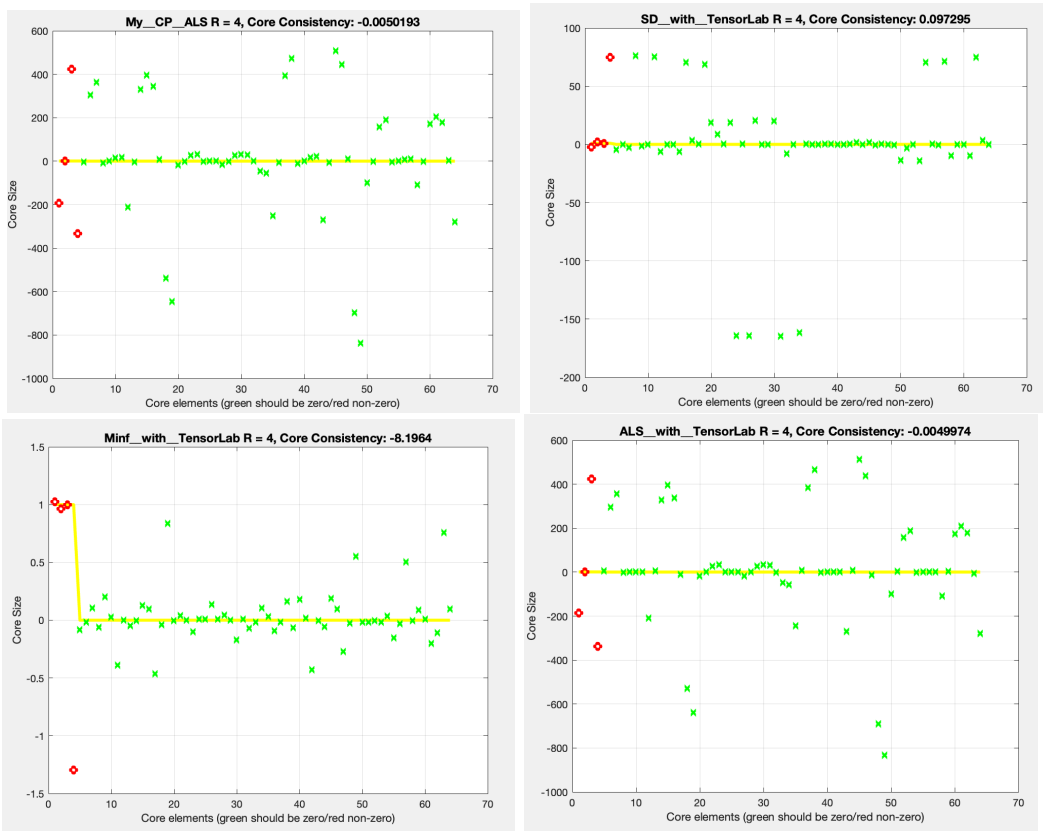

Figure 16: Rank 4

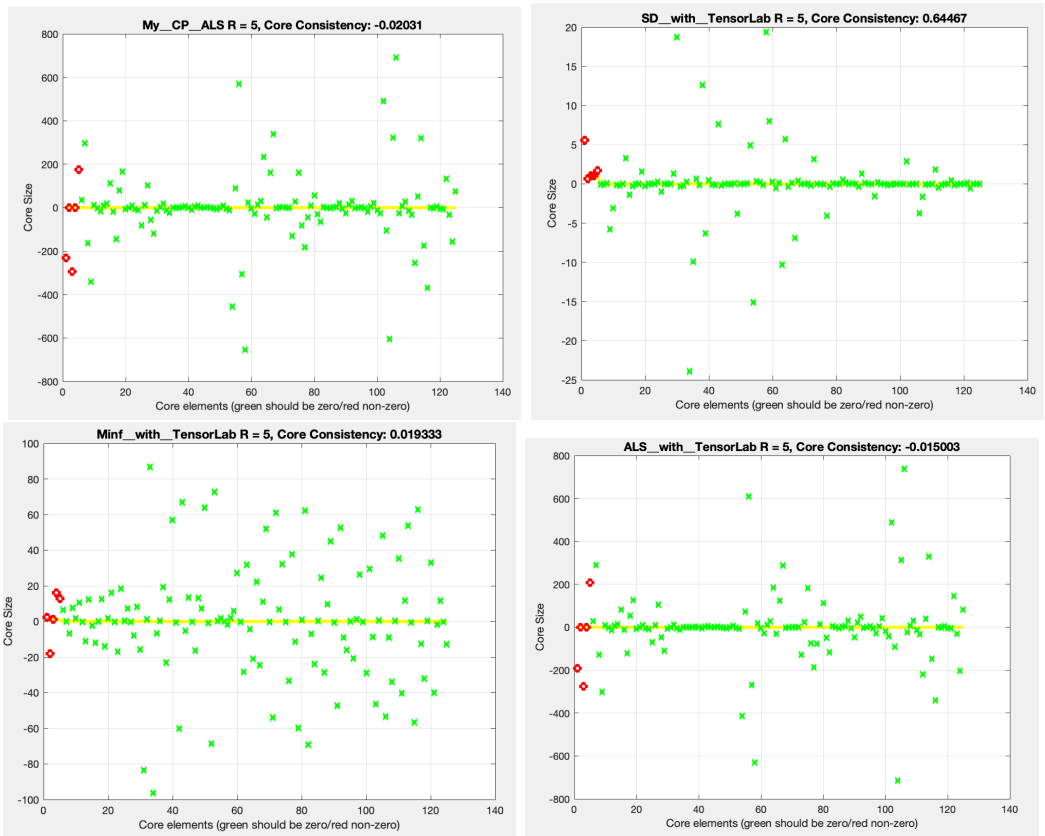

Figure 17: Enter Caption

- Core Consistency for Rank 2 & 3 ¿¿ Rank 4 & 5

- Only Minf for Rank = 3 has poor performance, but other algorithms performed pretty well with Rank = 2 & Rank = 3.
- All algorithms performed poorly with Ranks 4 & 5.