Non-negative CP Decomposition (NN-CP) of Order-3 Tensors: Mathematical Principles, Numerical Examples, and a General Rank-R Algorithm

Abstract

CANDECOMP/PARAFAC (CP) decomposition is a classical multilinear model that represents a higher-order tensor as a sum of "rank-1 tensors" (outer products of three vectors). When every factor matrix is additionally required to be non-negative, the model is called non-negative CP decomposition (NN-CP) — a natural three-way (and higher-order) generalization of non-negative matrix factorization (NMF). It is widely used for additive, interpretable modeling of chemometric data, signal processing data, and multi-factor coupled material data (e.g., temperature–strain–strain-rate flow-stress data). This report systematically derives the mathematical principles of NN-CP decomposition for order-3 tensors, including the outer-product tensor model, matricization (unfolding) and the Khatri-Rao product, the KKT conditions of the non-negative least squares (NNLS) optimization problem, and the alternating non-negative least squares (NN-ALS) algorithm based on multiplicative updates. Explicit matrix-form derivations and hand-verifiable numerical examples are given separately for Rank-1, Rank-2, and Rank-3, followed by a general algorithm applicable to an arbitrary rank R (companion Python code in Code/nncp_decomposition.py; all tensor data are synthetically generated within the script and no ready-made dataset is used).

Keywords: tensor decomposition; CP/PARAFAC; non-negativity constraint; alternating least squares; Khatri-Rao product

1 Introduction

Many experimental or simulated datasets naturally have a three-way (or higher) multi-factor structure — for example, a flow-stress tensor measured over "temperature × strain × strain rate," or fluorescence spectroscopy data over "wavelength × time × space" [1]. Simply flattening (unfolding) such data into a matrix and applying principal component analysis (PCA) or non-negative matrix factorization (NMF) discards the coupling structure among the multiple factors, and matrix-factorization factors are generally non-unique (extra constraints such as orthogonality are needed to fix them). CP decomposition, independently proposed by Harshman [2] and Carroll & Chang [3], performs low-rank additive modeling directly on the tensor's original three-way structure; under mild conditions its decomposition is inherently

unique (no orthogonality assumption is required) [4], making it better suited to explain the data-generating mechanism through physically meaningful "components." When the data itself is non-negative (e.g., concentration, intensity, stress, and other physical quantities), further imposing a non-negativity constraint (NN-CP) yields additive, interpretable factors free of cancelling negative components — an advantage analogous to that of NMF over PCA [5]; this three-way generalization of the non-negativity constraint is itself the direct motivation behind early work such as Welling & Weber [6].

The remainder of this report is organized as follows. Section 2 gives the general mathematical definition of the CP model, outer-product notation, matricization, and the Khatri-Rao product. Section 3 formulates NN-CP decomposition as a non-negativity-constrained least squares optimization problem and states its KKT optimality conditions. Section 4 derives the alternating non-negative least squares algorithm based on multiplicative updates in detail, writes out explicit matrix forms for Rank-1, Rank-2, and Rank-3, and finally gives the general algorithm for arbitrary Rank-R. Section 5 briefly reviews Kruskal's uniqueness condition for CP decomposition. Section 6 gives hand-verifiable numerical examples for Rank-1, Rank-2, and Rank-3. Section 7 describes the structure of the companion code and discusses convergence and complexity. Section 8 concludes.

2 Basic Definitions of CP Decomposition

2.1 Order-3 Tensors and the Outer Product

Let $\mathcal{X} \in \mathbb{R}_{\geq 0}^{I \times J \times K}$ be an order-3 (three-way) non-negative tensor with entries x_{ijk}, $i = 1, \ldots, I$, $j = 1, \ldots, J$, $k = 1, \ldots, K$, corresponding respectively to the tensor's 1st, 2nd, and 3rd "modes." Given three vectors $a \in \mathbb{R}^I$, $b \in \mathbb{R}^J$, $c \in \mathbb{R}^K$, their outer product is defined as an $I \times J \times K$ **rank-1 tensor**:

$$(a \circ b \circ c)_{ijk} = a_i \, b_j \, c_k \tag{1}$$

2.2 The General Form of the CP Model (Rank-R)

CP decomposition approximates $\mathcal{X}$ as a weighted sum of R rank-1 tensors:

$$\mathcal{X} \approx [\![\lambda; A, B, C]\!] = \sum_{r=1}^{R} \lambda_r \, a_r \circ b_r \circ c_r, \qquad x_{ijk} \approx \sum_{r=1}^{R} \lambda_r \, a_{ir} \, b_{jr} \, c_{kr} \tag{2}$$

where $A = [a_1, \dots, a_R] \in \mathbb{R}^{I \times R}$, $B = [b_1, \dots, b_R] \in \mathbb{R}^{J \times R}$, $C = [c_1, \dots, c_R] \in \mathbb{R}^{K \times R}$ are called the mode-1, mode-2, and mode-3 factor matrices, and $\lambda = (\lambda_1, \dots, \lambda_R)$ is the weight vector. When R is taken to be the smallest value for which $\mathcal{X}$ can be represented exactly, R is the (non-negative) rank of the tensor. **Non-negative CP decomposition (NN-CP)** requires

$$A \geq 0, \quad B \geq 0, \quad C \geq 0, \quad \lambda \geq 0.$$

Without loss of generality, an implementation may absorb λ_r directly into the columns of a factor matrix (e.g., into the columns of A), and separate λ out again after decomposition by column normalization (see cp_normalize in the code of Section 6).

2.3 Matricization (Unfolding) and the Khatri-Rao Product

To turn the trilinear model of Eq. (2) into a least squares problem that can be handled with matrix algebra, we introduce the tensor's **matricization (unfolding)**. Unfolding $\mathcal{X}$ along mode-1, mode-2, and mode-3 gives matrices denoted $X_{(1)} \in \mathbb{R}^{I \times JK}$, $X_{(2)} \in \mathbb{R}^{J \times IK}$, $X_{(3)} \in \mathbb{R}^{K \times IJ}$, respectively.

Next, define the **Khatri-Rao product** (column-wise Kronecker product) of two matrices $U \in \mathbb{R}^{I \times R}$, $V \in \mathbb{R}^{J \times R}$ as $U \odot V \in \mathbb{R}^{IJ \times R}$, whose r-th column is $u_r \otimes v_r$ (the Kronecker product of the column vectors). Using the Khatri-Rao product, the three matricized forms of the CP model can be written (following the convention of Kolda & Bader [4]) as:

$$X_{(1)} = A\,(C \odot B)^{\mathsf{T}}, \qquad X_{(2)} = B\,(C \odot A)^{\mathsf{T}}, \qquad X_{(3)} = C\,(B \odot A)^{\mathsf{T}} \qquad (3)$$

The Khatri-Rao product satisfies a key identity that makes the algorithm efficient: for any U, V with the same number of columns,

$$(U \odot V)^{\mathsf{T}}(U \odot V) = (U^{\mathsf{T}}U) * (V^{\mathsf{T}}V) \qquad (4)$$

where $*$ denotes the elementwise (Hadamard) product. This holds because $\langle u_r \otimes v_r,\, u_s \otimes v_s \rangle = \langle u_r, u_s \rangle \langle v_r, v_s \rangle$. Equation (4) reduces an inner-product computation of size $IJ \times IJ$ to the Hadamard product of two small $R \times R$ matrices, which is the core reason the alternating least squares algorithm below can be implemented efficiently.

3 The Optimization Problem Under Non-negativity Constraints

Expressing Eq. (2) as a least squares objective under the Frobenius norm, with non-negativity constraints imposed:

$$\min_{A,B,C \geq 0} f(A,B,C) = 1/2 \, \| \mathcal{X} - [\![A,B,C]\!] \|_F^2 \tag{5}$$

Equation (5) is non-convex jointly in the three factor matrices, but **fixing any two of them and solving only for the third** is a convex non-negative least squares (NNLS) problem — this is exactly the origin of the alternating least squares (ALS) idea. Taking the case of fixing B, C and solving for A as an example, Eq. (5) is rewritten in matrix form using Eq. (3):

$$f(A) = 1/2 \left\| X_{(1)} - A \, (C \odot B)^{\mathsf{T}} \right\|_F^2, \qquad A \geq 0 \tag{6}$$

This is a standard (matrix) non-negative least squares problem. Its KKT optimality conditions are: the gradient $\nabla_A f \geq 0$ (elementwise), and the complementary slackness condition $A_{ir} \, [\nabla_A f]_{ir} = 0$ holds for all (i, r). The gradient of Eq. (6) with respect to A is:

$$\nabla_A f = A[(C^{\mathsf{T}} C) * (B^{\mathsf{T}} B)] - X_{(1)}(C \odot B) \tag{7}$$

where the identity of Eq. (4) has been used to reduce $(C \odot B)^{\mathsf{T}}(C \odot B)$ to an $R \times R$ Hadamard product, thereby avoiding explicit construction of a large matrix product of size $JK \times R$.

4 Derivation of the Alternating Non-negative Least Squares (NN-ALS) Algorithm

4.1 Derivation of the Multiplicative Update Rule

The multiplicative update technique proposed by Lee & Seung [5,7] for non-negative matrix factorization can be directly generalized to Eq. (7). Split the gradient into two non-negative parts:

$$\nabla_A f = \nabla_A^+ - \nabla_A^-, \qquad \nabla_A^+ = A[(C^\top C) * (B^\top B)] \geq 0, \qquad \nabla_A^- = X_{(1)}(C \odot B) \geq 0 \tag{8}$$

(both terms are elementwise non-negative whenever $A, B, C, \mathcal{X} \geq 0$). Define the multiplicative update:

$$A \leftarrow A \circledast \frac{\nabla_A^-}{\nabla_A^+} = A \circledast \frac{X_{(1)}(C \odot B)}{A[(C^\top C) * (B^\top B)]} \tag{9}$$

where $\circledast$ and the division are both elementwise operations (a small positive ε is added to the denominator to avoid division by zero). Equation (9) has two important properties:

1. **Non-negativity is preserved automatically**: as long as the initial $A^{(0)} \geq 0$, the ratio $\nabla_A^-/\nabla_A^+ \geq 0$, so the product remains non-negative — no extra projection or clipping step is needed.
2. **Fixed points satisfy the KKT conditions**: if the update converges to a fixed point $A = A \circledast (\nabla_A^-/\nabla_A^+)$, then for every (i, r), $A_{ir}\nabla_{A,ir}^+ = A_{ir}\nabla_{A,ir}^-$, i.e., $A_{ir}[\nabla_A f]_{ir} = 0$ — exactly the complementary slackness condition of the NNLS problem. It can also be shown, via an auxiliary-function argument, that this update monotonically decreases the objective in Eq. (5) (the proof follows the same reasoning as Lee & Seung's original NMF multiplicative update and is omitted here; see the references).

The updates for B and C follow directly by symmetry (based on the matricizations $X_{(2)}, X_{(3)}$, respectively):

$$B \leftarrow B \circledast \frac{X_{(2)}(C \odot A)}{B[(C^\top C) * (A^\top A)]}, \qquad C \leftarrow C \circledast \frac{X_{(3)}(B \odot A)}{C[(B^\top B) * (A^\top A)]} \tag{10}$$

In each "outer sweep," the algorithm performs the three updates of Eqs. (9) and (10) in turn, then recomputes the reconstructed tensor $\widehat{\mathcal{X}} = [\![A, B, C]\!]$ and the relative error $\|\mathcal{X} - \widehat{\mathcal{X}}\|_F / \|\mathcal{X}\|_F$, until the change in this error falls below a threshold or the maximum number of iterations is reached.

Below, the $R \times R$ Gram-matrix portion of Eqs. (9)–(10) is expanded explicitly for $R = 1,2,3$, to show what the "abstract rank-R formula" looks like concretely for small ranks.

4.2 Explicit Derivation for Rank-1 ($R = 1$)

When $R = 1$, $A = a$, $B = b$, $C = c$ degenerate to column vectors, $C^\top C = \|c\|^2$ and $B^\top B = \|b\|^2$ are both scalars, and the Khatri-Rao product $C \odot B$ degenerates to the ordinary Kronecker product (of vectors) $c \otimes b$. Equation (9) simplifies to an elementwise scalar update:

$$a \leftarrow a \circledast \frac{X_{(1)}(c \otimes b)}{\|b\|^2 \, \|c\|^2 \, a} \tag{11}$$

That is, $a_i \leftarrow \frac{\sum_{j,k} x_{ijk} \, b_j c_k}{\|b\|^2 \|c\|^2}$ (the numerator is the bilinear contraction of tensor $\mathcal{X}$ with b, c over modes 2 and 3; the denominator is the product of the two squared norms). This is exactly the form of "tensor power iteration" for updating a: for an exactly rank-1, non-negative $\mathcal{X}$, Perron-Frobenius theory guarantees the existence of non-negative a, b, c such that the iteration converges to the exact solution, and in principle a single alternating update (together with normalization) suffices to converge, with no "crosstalk" between components to handle as in the $R > 1$ case.

4.3 Explicit Derivation for Rank-2 ($R = 2$)

When $R = 2$, $C^\top C$ and $B^\top B$ are 2×2 symmetric matrices:

$$C^\top C = \begin{bmatrix} c_1 \cdot c_1 & c_1 \cdot c_2 \\ c_2 \cdot c_1 & c_2 \cdot c_2 \end{bmatrix}, \quad B^\top B = \begin{bmatrix} b_1 \cdot b_1 & b_1 \cdot b_2 \\ b_2 \cdot b_1 & b_2 \cdot b_2 \end{bmatrix} \tag{12}$$

The 2×2 Hadamard product inside the denominator matrix $A[(C^\top C) * (B^\top B)]$ is

$$(C^\top C) * (B^\top B) = \begin{bmatrix} (c_1 \cdot c_1)(b_1 \cdot b_1) & (c_1 \cdot c_2)(b_1 \cdot b_2) \\ (c_2 \cdot c_1)(b_2 \cdot b_1) & (c_2 \cdot c_2)(b_2 \cdot b_2) \end{bmatrix} \tag{13}$$

Its off-diagonal entry $(c_1 \cdot c_2)(b_1 \cdot b_2)$ is precisely the source of "crosstalk" between the two rank-1 components: when the b, c factors of the two components are mutually orthogonal (zero inner product), this matrix degenerates to a diagonal matrix and Eq. (9) reduces to two independent Rank-1 updates as in Eq. (11); in general the off-diagonal entry is non-zero, and the two components are coupled during the iteration, requiring many alternations to converge

(corresponding to the slower convergence than Rank-1 observed in the numerical example of Section 6.2).

4.4 Explicit Derivation for Rank-3 ($R = 3$)

For $R = 3$, the Gram matrices above extend to 3×3:

$$C^{\mathsf{T}}C = \begin{bmatrix} c_1 \cdot c_1 & c_1 \cdot c_2 & c_1 \cdot c_3 \\ c_2 \cdot c_1 & c_2 \cdot c_2 & c_2 \cdot c_3 \\ c_3 \cdot c_1 & c_3 \cdot c_2 & c_3 \cdot c_3 \end{bmatrix}, \qquad B^{\mathsf{T}}B = \begin{bmatrix} b_1 \cdot b_1 & b_1 \cdot b_2 & b_1 \cdot b_3 \\ b_2 \cdot b_1 & b_2 \cdot b_2 & b_2 \cdot b_3 \\ b_3 \cdot b_1 & b_3 \cdot b_2 & b_3 \cdot b_3 \end{bmatrix} \quad (14)$$

The 3×3 Hadamard product $(C^{\mathsf{T}}C) * (B^{\mathsf{T}}B)$ in the denominator of Eq. (9) now has six independent off-diagonal entries (the matrix is symmetric), each corresponding to the coupling strength between one pair of components. Computationally, relative to Rank-2 and Rank-1, only the Gram-matrix size grows from $1 \times 1 \to 2 \times 2 \to 3 \times 3$; the cost of the MTTKRP (matricized-tensor-times-Khatri-Rao-product, i.e., the term $X_{(1)}(C \odot B)$) grows linearly with R, so the ALS algorithm's good scalability with rank R is preserved overall.

4.5 The General Rank-R Algorithm

Combining Eqs. (9)–(10), the NN-CP alternating non-negative least squares algorithm for an arbitrary rank R can be summarized in the following pseudocode:

```
Input: non-negative tensor X in R_{>=0}^{I x J x K}, target rank R, max iterations N, tolerance tol
Initialize: A in R_{>=0}^{I x R}, B in R_{>=0}^{J x R}, C in R_{>=0}^{K x R}  (random non-nega
tive init)
Precompute: X_(1), X_(2), X_(3)  (matricizations along the three modes)
for it = 1 ... N:
   A <- A (*) [ X_(1)(C(.)B) ] / [ A((C^T C)*(B^T B)) + eps ]
   B <- B (*) [ X_(2)(C(.)A) ] / [ B((C^T C)*(A^T A)) + eps ]
   C <- C (*) [ X_(3)(B(.)A) ] / [ C((B^T B)*(A^T A)) + eps ]
   compute relative error  err = ||X - [[A,B,C]]||_F / ||X||_F
   if |err_{it-1} - err_it| < tol, break
Output: A, B, C, and the weight vector lambda separated out by column normalization
```

(Here (*) denotes the elementwise product and (.) denotes the Khatri-Rao product.)

This pseudocode corresponds one-to-one with the function nncp_als(X, rank, ...) in the companion code Code/nncp_decomposition.py; the rank parameter is exactly R, and applies to $R = 1,2,3$ or any larger R without a separate implementation for each rank.

5 Uniqueness: Kruskal's Condition (Brief)

Unlike matrix decompositions (e.g., SVD, NMF), CP decomposition is essentially unique under mild conditions (up to permutation and per-component rescaling), without requiring extra constraints such as orthogonality. The classical sufficient condition of Kruskal [8] is:

$$k_A + k_B + k_C \ \geq \ 2R + 2 \tag{15}$$

where k_M denotes the **Kruskal rank** (k-rank) of matrix M: the largest integer k such that every k columns of M are linearly independent. When Eq. (15) is satisfied, if $[\![A, B, C]\!] = [\![A', B', C']\!]$, then there must exist a permutation matrix Π and diagonal scaling matrices D_1, D_2, D_3 (satisfying $D_1 D_2 D_3 = I$) such that $A' = A\Pi D_1$, $B' = B\Pi D_2$, $C' = C\Pi D_3$. This property is the most important theoretical advantage of CP decomposition over matrix decomposition: when Eq. (15) holds, the decomposed "components" are determined by the data-generating mechanism itself, rather than by an algorithmic or artificially chosen constraint. The "component-order permutation" phenomenon observed in the Rank-3 numerical example of Section 6.3 is a direct manifestation of this "permutation indeterminacy" in uniqueness.

6 Numerical Examples

The following three examples are all generated and verified using the functions make_rank_r_tensor (constructs an exact rank-R tensor from given non-negative vectors) and nncp_als (non-negative ALS decomposition) in the companion code Code/nncp_decomposition.py. All tensors are constructed by hand/synthetically within the script and do not rely on any external database or dataset.

6.1 Rank-1 Example

Take $I = J = K = 2$ and construct the vectors

$$a = (1,2), \qquad b = (2,1), \qquad c = (1,3)$$

Using Eq. (1) to construct the exact rank-1 tensor $\mathcal{X} = a \circ b \circ c$, its two "slices" (fixing k along mode-3) are:

$$\mathcal{X}_{:,:,1} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}, \qquad \mathcal{X}_{:,:,2} = \begin{bmatrix} 6 & 3 \\ 12 & 6 \end{bmatrix}$$

Iterating Eq. (11) from a random non-negative initialization of a, b, c (nncp_als(X, rank=1)), after convergence (column-normalized, with weight denoted $\hat{\lambda}$) we obtain:

Quantity	True value (normalized)	Recovered value
$\hat{a}$	$(0.4472,\ 0.8944)$	$(0.4472,\ 0.8944)$
$\hat{b}$	$(0.8944,\ 0.4472)$	$(0.8944,\ 0.4472)$
$\hat{c}$	$(0.3162,\ 0.9487)$	$(0.3162,\ 0.9487)$
$\hat{\lambda}$	$\|a\|\|b\|\|c\| = 15.8114$	15.8114

The final relative reconstruction error is $\left\|\mathcal{X} - \widehat{\mathcal{X}}\right\|_F / \|\mathcal{X}\|_F \approx 2.4 \times 10^{-14}$, i.e., exact recovery to floating-point precision — consistent with the theoretical analysis of Section 4.2: decomposing an exactly rank-1 non-negative tensor is essentially a power-iteration problem, which converges quickly and uniquely (no permutation ambiguity between components, since there is only one component).

6.2 Rank-2 Example

Take $I = 3, J = 2, K = 2$, with the vectors of the two rank-1 components given by

$$a_1 = (1,0,2),\ b_1 = (1,1),\ c_1 = (2,0); \qquad a_2 = (0,1,1),\ b_2 = (2,0),\ c_2 = (1,1)$$

(both component weights taken as 1). Constructing $\mathcal{X} = a_1 \circ b_1 \circ c_1 + a_2 \circ b_2 \circ c_2$, its two mode-3 slices are:

$$\mathcal{X}_{:,:,1} = \begin{bmatrix} 2 & 0 \\ 2 & 2 \\ 6 & 4 \end{bmatrix}, \qquad \mathcal{X}_{:,:,2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 0 \end{bmatrix}$$

After 8000 iterations of nncp_als(X, rank=2), the recovered results (component-aligned, column-normalized) are:

Component r	$\hat{\lambda}_r$	True value $\lambda_r = \|a_r\|\|b_r\|\|c_r\|$	$\hat{a}_r$	$\hat{b}_r$	$\hat{c}_r$
1	6.3244	$\sqrt{5} \cdot \sqrt{2} \cdot 2 = 6.3246$	(0.4472,0,0.8	(0.7071,0.707	(1,0)
2	4.0000	$\sqrt{2} \cdot 2 \cdot \sqrt{2} = 4.0000$	(0,0.7071,0.7(	(1,0)	(0.7071,0.707

The recovered values agree fully with the true (normalized) values (numerical noise $< 10^{-4}$), with a final relative reconstruction error of about 4.1×10^{-5}. Compared with the Rank-1 example, the number of iterations required for convergence is markedly larger, consistent with the "crosstalk between components" (non-zero off-diagonal Hadamard terms) discussed via Eq. (13) in Section 4.3, which slows coupled convergence.

6.3 Rank-3 Example

Take $I = 4, J = 3, K = 3$, with three mutually non-orthogonal rank-1 components carrying different weights 2.0, 1.5, 1.0:

$$a_1 = (1,0,0,1), b_1 = (1,1,0), c_1 = (1,0,1)$$
$$a_2 = (0,1,0,1), b_2 = (0,1,1), c_2 = (1,1,0)$$
$$a_3 = (0,0,1,1), b_3 = (1,0,1), c_3 = (0,1,1)$$

After 15000 iterations, nncp_als(X, rank=3) converges with a relative reconstruction error of about 2.7×10^{-5}. The recovered weights are $\hat{\lambda} = (2.8284,\ 5.6568,\ 4.2426)$, and the internal component ordering produced by the algorithm differs from the construction order — as discussed via the "permutation indeterminacy" in the Kruskal uniqueness discussion of Section 5, the actual correspondence is:

Algorithm output component	Corresponding true component	$\hat{\lambda}$	True value $\lambda = w \cdot \|a\|\|b\|\|c\|$

Algorithm output component	Corresponding true component	$\hat{\lambda}$	True value $\lambda = w \cdot \|\|a\|\|\|\|b\|\|\|\|c\|\|$
Output #1	True component 3 (weight 1.0)	2.8284	$1.0 \times (\sqrt{2})^3 = 2.8284$
Output #2	True component 1 (weight 2.0)	5.6568	$2.0 \times (\sqrt{2})^3 = 5.6569$
Output #3	True component 2 (weight 1.5)	4.2426	$1.5 \times (\sqrt{2})^3 = 4.2426$

Checking the three sets of normalized factor vectors $(\hat{a}, \hat{b}, \hat{c})$ one by one confirms numerical agreement with the corresponding true components (difference $< 10^{-4}$). This "components reordered but numerically exactly recovered" phenomenon is a direct manifestation, in a numerical experiment, of the theoretical property that CP decomposition is "unique only up to permutation and scaling"; it also shows that comparing two decomposition results requires component alignment first (e.g., sorting by $\hat{\lambda}$ or matching by inter-factor correlation), rather than comparing directly by component index.

7 Implementation and Convergence Discussion

The companion code Code/nncp_decomposition.py (a pure-NumPy implementation, relying on no dedicated tensor-decomposition library and loading no ready-made dataset) contains the following parts:

- khatri_rao: the Khatri-Rao product of an arbitrary number of matrices;
- unfold: matricization of an order-3 tensor along mode-1/2/3;
- cp_to_tensor: reconstructs the full tensor $[\![A, B, C]\!]$ from the factor matrices;
- cp_normalize: absorbs column norms into the weight λ, normalizing the factor matrices to unit column norm;
- nncp_als(X, rank, n_iter, tol, eps, random_state): a direct implementation of the pseudocode in Section 4.5, with the rank parameter supporting any $R \geq 1$;
- make_rank_r_tensor / demo_rank: synthesizes an exact rank-R tensor from given vectors and calls nncp_als to verify the decomposition result — the three examples in Section 6 and one additional $R = 6$ example are all generated by this function.

Convergence and complexity: the multiplicative updates of Eqs. (9)–(10) can be shown to monotonically decrease the objective in Eq. (5) (via a Lee-Seung-type auxiliary-function argument), but the convergence rate is first-order (linear), generally slower than implementations based on exact non-negative least squares (e.g., the Lawson-Hanson active-set method [10]) or hierarchical alternating least squares (HALS [9]) — consistent with the phenomenon in the Rank-2 and Rank-3 examples of Sections 6.2–6.3, where thousands of sweeps are needed to reach a relative error of 10^{-4}–10^{-5} (whereas Rank-1 converges to machine precision in effectively one sweep). The dominant computational cost of each sweep is the product of a matricized tensor with a Khatri-Rao product (i.e., MTTKRP, matricized-tensor-times-Khatri-Rao-product), with complexity about $O(IJK \cdot R)$, growing linearly with the total number of tensor elements and the rank R; the cost of the $R \times R$ Gram-matrix products is negligible by comparison. For faster convergence, the multiplicative updates in this report can be replaced with HALS or a projected-gradient/active-set NNLS subproblem solver, while the overall algorithmic framework (matricization + alternating sweeps) remains unchanged.

Conclusions

1. CP decomposition of an order-3 tensor represents the data as a sum of rank-1 tensors (outer products of three vectors); its matricized forms such as $X_{(1)} = A(C \odot B)^{\mathsf{T}}$, together with the Khatri-Rao Gram identity $(U \odot V)^{\mathsf{T}}(U \odot V) = (U^{\mathsf{T}}U) * (V^{\mathsf{T}}V)$, turn the trilinear optimization problem into a sequence of efficiently solvable (non-negative) least squares subproblems.
2. Under the non-negativity constraint, Lee-Seung-type multiplicative updates (Eqs. 9, 10) monotonically decrease the reconstruction error while guaranteeing non-negativity of the factor matrices, and the fixed points satisfy exactly the KKT complementary slackness conditions of the NNLS problem; this update rule has a unified form for any rank R, with Rank-1, Rank-2, and Rank-3 being special cases that differ only in the size of the $R \times R$ Gram matrix.
3. The numerical examples show that an exact rank-1 tensor is recovered in effectively one convergence step to floating-point precision; as the rank increases, off-diagonal Gram terms couple the components, and the number of iterations needed for convergence increases significantly; the recovered component order may differ from the construction order, which is a direct manifestation of the theoretical property that

CP decomposition is “unique only up to permutation and scaling” (Kruskal’s condition, Eq. 15).

4. The companion nncp_decomposition.py provides a minimal, dependency-free NN-CP implementation that relies on no dedicated library and no ready-made dataset; the rank parameter of the nncp_als function can be set directly to any positive integer, covering decomposition needs from Rank-1 to any Rank-R.

References

1. Bro, R. (1997). PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2), 149–171.
2. Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an explanatory multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16, 1–84.
3. Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an N-way generalization of the Eckart-Young decomposition. *Psychometrika*, 35(3), 283–319.
4. Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3), 455–500.
5. Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755), 788–791.
6. Welling, M., & Weber, M. (2001). Positive tensor factorization. *Pattern Recognition Letters*, 22(12), 1255–1261.
7. Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13, 556–562.
8. Kruskal, J. B. (1977). Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and Its Applications*, 18(2), 95–138.
9. Cichocki, A., & Phan, A. H. (2009). Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E92-A(3), 708–721.
10. Lawson, C. L., & Hanson, R. J. (1974). *Solving Least Squares Problems*. Prentice-Hall.