Assignments

Report Jennrich's Algorithm Nhat-Nam Nguyen

Jennrich's algorithm is an efficient approach to decompose a third-order tensor $\mathcal{X}$ under the assumption that the factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are all of full column rank. The tensor decomposition is represented as follows:

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \, \boldsymbol{a}_r \bigotimes \boldsymbol{b}_r \bigotimes \boldsymbol{c}_r$$

where:

- - R is the rank of the tensor.
- - λ_r are scalar coefficients.
- - $\boldsymbol{a}_r$, $\boldsymbol{b}_r$, and $\boldsymbol{c}_r$ are the r-th columns of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ respectively.
- - $\bigotimes$ denotes the outer product, forming a rank-1 tensor from vectors.

The core idea behind Jennrich's algorithm is to exploit the independence of factor matrices by applying random projections to reduce the tensor to a matrix form that retains the essential information of the original tensor. Firstly, we choose random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ and hit the tensor $\mathcal{X}$ with these vectors. The resulting matrices encode the factor matrix structure and allow us to use eigendecomposition to isolate each matrix component.
1. Choose two random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.
2. Contract the tensor $\mathcal{X}$ along the third mode with $\boldsymbol{x}$ and $\boldsymbol{y}$, resulting in two matrices M_x and M_y:

$$M_x = \sum_{r=1}^{R} \langle \boldsymbol{c}_r, \boldsymbol{x} \rangle \, \boldsymbol{a}_r \bigotimes \boldsymbol{b}_r = \boldsymbol{A} D_x \boldsymbol{B}^T$$

$$M_y = \sum_{r=1}^{R} \langle \boldsymbol{c}_r, \boldsymbol{y} \rangle \, \boldsymbol{a}_r \bigotimes \boldsymbol{b}_r = \boldsymbol{A} D_y \boldsymbol{B}^T$$

where D_x and D_y are diagonal matrices with diagonal entries $\langle \boldsymbol{c}_1, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{c}_r, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{c}_1, \boldsymbol{y} \rangle, \ldots, \langle \boldsymbol{c}_r, \boldsymbol{y} \rangle$, respectively. Here, $\boldsymbol{A}$ has columns $\boldsymbol{A}_r$ and $\boldsymbol{B}$ has columns $\boldsymbol{b}_r$.
3. Since M_y is not necessarily full rank, we use the Moore-Penrose pseudoinverse M_y^+, which has the property that:

$$M_y M_y^+ = I$$

This allows us to handle non-invertible matrices effectively. The pseudoinverse of M_y can be expressed as:

$$M_y^+ = (\boldsymbol{B}^T)^+ D_y^+ \boldsymbol{A}^+$$

where D_y^+ is the pseudoinverse of the diagonal matrix D_y, and $(\boldsymbol{B}^T)^+$ and $\boldsymbol{A}^+$ are the pseudoinverses of $\boldsymbol{B}^T$ and $\boldsymbol{A}$, respectively.
4. To exploit the structure of the matrices, we compute the product $M_x M_y^+$ to find $\boldsymbol{A}$:

$$M_x M_y^+ = \boldsymbol{A} D_x \boldsymbol{B}^T \left(\boldsymbol{B}^T\right)^+ D_y^+ \boldsymbol{A}^+ = \boldsymbol{A} D \boldsymbol{A}^+$$

where D is a diagonal matrix with entries $D_{rr} = \frac{\langle \boldsymbol{c}_r, \boldsymbol{x} \rangle}{\langle \boldsymbol{c}_r, \boldsymbol{y} \rangle}$. The reason for choosing $\boldsymbol{x}$ and $\boldsymbol{y}$ as random vectors is to ensure, with high probability, that the values D_{rr} are distinct and well-defined. Respectively we can find $\boldsymbol{B}$:

$$M_x^+ M_y = \boldsymbol{B} D_x \boldsymbol{A}^T \left(\boldsymbol{A}^T\right)^+ D_y^+ \boldsymbol{B}^+ = \boldsymbol{B} D \boldsymbol{B}^+$$

5. When we have $\boldsymbol{A}$ and $\boldsymbol{B}$ we can solve thea linear system to find $\boldsymbol{C}$.

$$(\mathbf{B} \odot \mathbf{A})\mathbf{C}^\top = \mathcal{X}_{(3)}$$

and

$$\boldsymbol{C} = \boldsymbol{X}_{(3)} \left[(\boldsymbol{B} \odot \boldsymbol{A})^T \right]^\dagger = \boldsymbol{X}_{(3)} (\boldsymbol{B} \odot \boldsymbol{A}) \left(\boldsymbol{B}^T \boldsymbol{B} * \boldsymbol{A}^T \boldsymbol{A} \right)^\dagger$$

Below is the MATLAB code implementing Jennrich's algorithm

```
function [A_hat, B_hat, C_hat, lambda] = jennrich(X, R)
    % Jennrich decomposition function
    % Input:
    %   X - input tensor of size (I, J, K)
    %   R - target rank
    % Output:
    %   A_hat, B_hat, C_hat - estimated factor matrices
    %   lambda - scaling factors

    % Get dimensions of the tensor
    I = size(X, 1);
    J = size(X, 2);
    K = size(X, 3);

    % Generate random vectors x and y of size K
    x = rand(K, 1);
    y = rand(K, 1);

    % Initialize matrices Xx and Xy
    Xx = zeros(I, J);
    Xy = zeros(I, J);

    % Step 1: Contract the tensor along the third mode with x and y
    for i = 1:R
        Xx = Xx + (C(:, i)' * x) * (A(:, i) * B(:, i)');
        Xy = Xy + (C(:, i)' * y) * (A(:, i) * B(:, i)');
    end

    % Step 2: Eigendecomposition to find A_hat
    [Vx, Dx] = eig(Xx * Xx');
    A_hat = Vx;

    % Step 3: Eigendecomposition to find B_hat
    [Vy, Dy] = eig(Xy' * Xy);
    B_hat = Vy;

    % Step 4: Compute C_hat
    V3 = (B_hat' * B_hat) .* (A_hat' * A_hat);
    X3 = ndim_unfold(X, 3); % Unfold X along the third mode
    C_hat = X3 * khatrirao_prod(B_hat, A_hat) * pinv(V3);

    % Step 5: Calculate lambda as scaling factors
    lambda = zeros(1, R);
    for r = 1:R
        lambda(r) = norm(A_hat(:, r)) * norm(B_hat(:, r)) * norm(C_hat(:, r));

        % Normalize A_hat, B_hat, and C_hat
        A_hat(:, r) = A_hat(:, r) / norm(A_hat(:, r));
        B_hat(:, r) = B_hat(:, r) / norm(B_hat(:, r));
        C_hat(:, r) = C_hat(:, r) / norm(C_hat(:, r));
    end
end
```

Func 1: Jennrich's Algorithm MATLAB Implementation