Approximate Mini-ALS for Tensor Completion

This repository provides a simplified Python implementation of the **Approximate Mini-ALS (Alternating Least Squares)** algorithm for **tensor completion** problems. The algorithm estimates missing values in a partially observed tensor by solving a sequence of least-squares subproblems.

💡 Mathematical Background

The tensor completion problem can be formulated as follows:

Given:

- A partially observed tensor $\mathcal{X}$.
- A set of observed entries Ω.
- A low-rank factorization assumption.

The objective is to minimize:

$$\min_x \|Px - q\|_2^2$$

where:

- $P \in \mathbb{R}^{|\Omega| \times R}$ is the subsampled design matrix corresponding to observed entries.
- $q \in \mathbb{R}^{|\Omega|}$ is the observed values vector.
- R is the rank parameter.
- $x \in \mathbb{R}^R$ is the solution vector we want to estimate.

To handle missing entries, we introduce a *lifted* matrix $A \in \mathbb{R}^{I \times R}$ such that:

$$A = \begin{bmatrix} P \\ P_{\text{missing}} \end{bmatrix}$$

with $I \geq |\Omega|$. The problem becomes:

$$\min_x \|Ax - b\|_2^2$$

where b is the *lifted* vector combining known and missing values.

🔍 Algorithm Overview

The **Approximate Mini-ALS** algorithm iteratively estimates x by:

1. **Lifting the Problem:**

   Constructing a *lifted* vector b: $b = \begin{bmatrix} q \\ b_{\text{missing}} \end{bmatrix}$ where b_{missing} is initialized and updated during iterations.
2. **Solving Least-Squares Subproblems:**

   Each iteration solves: $x^{(k)} = \left(A^\top A\right)^{-1} A^\top b$ using the **normal equation.** This provides a closed-form least-squares solution.
3. **Acceleration Step:**
   To improve convergence speed, the following acceleration is applied:

   $\alpha^{(k)} = \dfrac{\|x^{(k)} - x^{(k-1)}\|_2^2}{\|x^{(k-1)} - x^{(k-2)}\|_2^2}$ Then, the accelerated solution is updated as:

   $$x^{(k)} = x^{(k-1)} + \frac{1}{1 - \alpha^{(k)}} \left(x^{(k)} - x^{(k-1)}\right)$$

4. **Convergence Check:**

   The iterations stop when: $\|x^{(k)} - x^{(k-1)}\|_2 < \epsilon$ where ϵ is a small threshold.

🏃 Code Walkthrough

⚙️ Function Definition

```
def approximate_mini_als(A, P, q, R, max_iter=10, epsilon=1e-6):
```

- **Inputs:**
  - A: The *lifted* design matrix $(I \times R)$.
  - P: The subsampled matrix corresponding to observed entries $(|\Omega| \times R)$.
  - q: Observed values vector $(|\Omega|)$.
  - R: Rank parameter.
  - `max_iter`: Maximum number of iterations.
  - `epsilon`: Convergence threshold.
- **Output:**
  - x: Approximate solution vector (R).

🔄 Core Steps

1. **Initialization:**

   ```
   x = np.zeros(R)
   b_missing = np.zeros(I - O)
   ATA_inv = np.linalg.inv(A.T @ A)
   ```

   - Initializes the solution vector x and missing values b_{missing}.
   - Precomputes $(A^\top A)^{-1}$ for efficiency.
2. **Iterative Updates:**

   ```
   for k in range(max_iter):
       b = np.concatenate([q, b_missing])
       x_new = ATA_inv @ (A.T @ b)
   ```

   - Constructs the *lifted* vector b.
   - Solves the normal equation for x.
3. **Acceleration:**

   ```
   if k > 1:
       alpha = np.linalg.norm(x_new - x_old)**2 / np.linalg.norm(x_old - x_prev)**2
   ```

```
    x = x_old + (1 / (1 - alpha)) * (x_new - x_old)
else:
    x = x_new
```

- Uses the acceleration formula for faster convergence.

4. **Convergence Check:**

```
if np.linalg.norm(x - x_old) < epsilon:
    break
```

- Terminates iteration if convergence is reached.

📈 Example Usage

```
I = 100  # Lifted dimension
R = 10   # Rank parameter
O = 50   # Observed samples
A = np.random.rand(I, R)
P = A[:O, :]
q = np.random.rand(O)
x_approx = approximate_mini_als(A, P, q, R)
estimated_E = P @ x_approx
print("Estimated solution:", estimated_E[:10])
print("Length of estimated solution:", len(estimated_E))
```

✨ Key Highlights

- **Acceleration technique** for faster convergence.
- **Closed-form least-squares solution** at each iteration.
- **Easily extensible** to more complex tensor completion scenarios.

📜 References

- Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*
- **ALS techniques** for matrix and tensor factorization in recommender systems.