Lab 3: Nonnegative Matrix Factorization

Ting Lin, 1700010644

May 30, 2020

Contents

In this lab, we surveyed several methods on non-negative matrix factorization (NMF). Most of introduced algorithms are concerning NMF under the Euclidean(Frobenius) metric, however, some of them can be naturally extended to KL divergence. We introduce methods based on Multiplicative Update, Alternative Least Square, Alternative Non-negative Least Square, Alternative Direction of Multiplier Methods. Moreover, a new algorithm based on Nonlinear Least Square while subproblem is solved by ADMM is proposed. In the sections, we will analyze the convergence and test their performance, and finally test them in the popular ORL and Yale database.

1 Problem Setting

Nonnegative matrix factorization(NMF) considers the following optimization problem,

$$\begin{aligned} &\min \|V - WH\|_F^2 \\ &\text{s.t. } 0 < W \in \mathbb{R}^{m\times r}, \\ &\qquad 0 < H \in \mathbb{R}^{r\times n} \end{aligned} \tag{1}$$

or based on KL divergence alternatively,

$$\begin{aligned} &\min D(V||WH) \\ &\text{s.t. } 0 < W \in \mathbb{R}^{m\times r}, \\ &\qquad 0 < H \in \mathbb{R}^{r\times n} \end{aligned} \tag{2}$$

Here

$$D(A||B) = \sum_{i,j} A_{ij} \log \frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij},$$

which might behaves singular when the entry of A or B is near zero.

We always assume that V is nonnegative otherwise we can consider $\mathcal{P}(V)$. Here and throughout this paper we denote $\mathcal{P}(V)$ by the projection: $[\mathcal{P}(V)]_{ij} = \max(V_{ij}, 0)$.

We recall some basic properties according to [1]:

1. NMF does not in general have a unique solution (up to scaling and permutation).
2. NMF is not identifiable, in the sense that the task is challenging even in the clear artificial low rank data.
3. Clearly, it is smooth but non-convex.
4. The gradient of W is $\nabla W = W(HH') - VH'$, and $\nabla H = W'WH - W'V$

2 Methods based on MU

In this section we introduce MU method and its variants, like [2, 4, 3, 5].

2.1 MU and its convergence analysis

The basic idea of MU is simple, we choose a suitable stepsize and do gradient descent. Concretely, consider the gradient descent:

$$H_{ij} = H_{ij} + \eta_{ij}[\nabla H]_{ij}$$

Set $\eta_{ij} = H_{ij}/[W'WH]_{ij}$ we obtain the update rule of H,

$$H_{ij} = H_{ij} \frac{[W'V]_{ij}}{[W'WH]_{ij}}$$

Similar for W's update,

$$W_{ij} = W_{ij} \frac{[V'H]_{ij}}{[WHH']_{ij}}$$

Here we introduce a more compact notation, let $\otimes$ and $\oslash$ be the element-wise multiplication and division operator, we can rewrite it as

$$H = H \otimes (W'V) \oslash (W'WH), \qquad W = W \otimes (V'H) \oslash (WHH')$$

The implementation please see **nmf_mu.m**.

The implementation is a little complex in the KL version:

$$H_{ij} = H_{ij} \frac{\sum_k W_{ki} V_{kj} / [WH]_{kj}}{\sum_k W_{ki}}$$

$$H_{ij} = H_{ij} \frac{\sum_k H_{jk} V_{ik} / [WH]_{ik}}{\sum_k H_{jk}}$$

also, see **nmf_mu.m**

The update factor is chosen such that the residual $\|V - WH\|$ ($D(V||WH)$ resp) is nonincreasing. The advantage of MU method is that no explicit projection operation is needed.

Proposition 1. *$\|V - WH\|$ ($D(V||WH)$ resp) is nonincreasing after each update.*

We summarize it into the following algorithm [2].

Algorithm 1 MU

Require: V, k

1: Initialize W and H
2: **while** not convergence **do**
3:
$$W_{ij} = W_{ij} \frac{[V'H]_{ij}}{[WHH']_{ij}}$$
4:
$$H_{ij} = H_{ij} \frac{[W'V]_{ij}}{[W'WH]_{ij}}$$
5: **end while**
6: **return** W, H.

In practice, we set stop criterion as max iteration, and initialization step we use $W = rand(m, r); H = rand(r, n)$.

2.2 Modified and Accelerated MU

However, the original MU method is not efficient, and faced various problem. We introduce some techniques to alleviate them and make mu more powerful.

Modified MU In [3], two main difficulties of MU is declared:

1. The denominator of the step size may be zero.
2. If the numerator if zero, and the gradient is negative, then H_{ij}^k will not be changed. Hence the convergence analysis fails, and it often occurs in numerical results.

Therefore, the authors proposed the modified step size to

$$\bar{H} \oslash (W'WH + \delta)$$

where

$$\bar{H} = H \otimes [\nabla H \geq 0] + \max(H, \sigma) \otimes [\nabla H < 0].$$

The detailed algorithm is shown below.

Algorithm 2 Modified MU

Require: V, k

Initialize W and H

while not convergence **do**

$$\bar{H} = H \otimes [\nabla H \geq 0] + \max(H, \sigma) \otimes [\nabla H < 0].$$

$$\bar{W} = W \otimes [\nabla W \geq 0] + \max(W, \sigma) \otimes [\nabla W < 0].$$

$$H = H - \bar{H} \oslash (W'W\bar{H} + \delta) \otimes \nabla H$$

$$W = W - \bar{W} \oslash (\bar{W}HH' + \delta) \otimes \nabla W$$

normalize W and H, such that the column sum of W is one.

end while

return W, H.

In [3], several properties about this method is discussed.

Proposition 2. *If the initial value is nonnegative (strictly positive), then nonnegativity and strit positivity will be preserved after each update.*

Proposition 3. *The loss $\|V - WH\|$ is nonincreasing after each modified MU update. The sequence W^k, H^k is pre-compact, and each of its accumulated point satisfies KKT condition.*

2.3 Accelerated MU

The acceleration techniques is introduced and analyzed in [4, 5]. We mainly focus on the work in [4].

For any given ρ_W and ρ_H, we introduce the following algorithm:

Algorithm 3 Accelerated MU

Require: V, k, δ

Initialize W and H
while not convergence **do**
  $\varepsilon = 1, \gamma = 1$
  while iter¡$[1 + rho_W \alpha]$ and $\gamma > \delta\varepsilon$ **do**
    $W^- = W$
    $W = W \otimes (V'H) \oslash (WHH')$
    if iter==1 **then**
      $\varepsilon = \|W - W^-\|\|_F$
    end if
    $\gamma = \|W - W^-\|\|_F$
  end while
  $\varepsilon = 1, \gamma = 1$
  while iter¡$[1 + rho_H \alpha]$ and $\gamma > \delta\varepsilon$ **do**
    $H^- = H$
    $H = H \otimes (W'V) \oslash (W'WH)$
    if iter==1 **then**
      $\varepsilon = \|H - H^-\|\|_F$
    end if
    $\gamma = \|H - H^-\|\|_F$
  end while
end while
return W, H.

Here $\rho_W = 1 + \frac{mn+nr}{mr+m}$, $\rho_H = 1 + \frac{mn+mr}{nr+n}$. δ is chosen to control the stop criterion and is chosen as 0.1 in our code (see **mnf_muacc.m**). Notice that the method is just replace an alternative update to a new update strategy with condition, hence all the convergence analysis makes sense in the accelerated version.

3 Methods based on ALS

The idea of Alternative (Nonnegative) Least Square is instead of solving the original problem, we optimize two linear problem alternatively.

$$\min_{W>0} \|V - WH^*\| \tag{3}$$

$$\min_{H>0} \|V - W^*H\| \tag{4}$$

clearly, if we can solve the linear problem correctly, then the series W^k, H^k must be non-increasing. Two ways was attempted in the literature. We will discuss the methods based on solving subproblem with or without constraints. In this section we introduce solve ALS plus a suitable projection step, and more general algorithms in constraint linear programming is explored in

3.1 Naive ALS

In Naive ALS, we only need to compute the least square solution and project it into positive cone.

Algorithm 4 MU

Require: V, k

Initialize W and H

while not convergence **do**

  $H = (W'W)^{-1}(W'V)$

  $H = \mathcal{P}(H)$

  $W = VH'(HH')^{-1}$

  $W = \mathcal{P}(W)$

end while

return W, H.

3.2 Projected BB method based ALS

We solve the subproblem by a projected BB, which is first introduced in [7]. Which is slightly different from global BB method, since we have to project our result into positive cone after each line search.

Algorithm 5 PBBNLS

Require: $A, B, X, \nabla X, \rho$

Set $\gamma, M, \lambda_{max}, \lambda_{min}$

Set $Q(X) - (A'B, X) + 0.5(X, A'AX)$

$\lambda = 1/\|\nabla X\|_\infty$

for i = 1:iter **do**

  $\alpha = 1$

  $X^+ = X - \lambda \nabla X$

  $X^+ = \mathcal{P}(X^+)$

  $Q^+ = Q(X^+)$

  while $\lambda > \lambda_{min}$ and $Q^+ < \max_{i-M<k<i} Q(X_i) + \gamma\alpha(\nabla X, D)$ **do**

    $\alpha = \alpha/4$

    $X^+ = X - \alpha\lambda\nabla X$

    $X^+ = \mathcal{P}(X^+)$

    $Q^+ = Q(X^+)$

  end while

  $s = X^+ - X$

  $\nabla X^+ = A'AX - A'B$

  $y = \nabla X^+ - \nabla X$

  if $(s, y) < \varepsilon$ **then**

    $\lambda = \lambda_{\max}$

  else

    $\lambda = \min(\lambda_{\max}, \max(\lambda_{\min}, (s, s)/(s, y)))$

  end if

  $\nabla X = \nabla X^+, X_i = X = X^+$

end for

return $X, \nabla X$

The whole algorithm uses PBBNLS to update each steps.

Algorithm 6 APBB

Require: V, k
1. Initialize W and H
2. **while** not convergence **do**
3. $[W, \nabla W] = PBBNLS(H', V', W', \nabla W')$
4. $W = W'$
5. $\nabla W = \nabla W'$
6. $[H, \nabla H] = PBBNLS(W, V, H, \nabla H)$
7. **end while**
8. **return** W, H.

3.3 Projected Gradient Descent based ALS

[6] proposed a projected gradient descent method to solve the subproblem.

Algorithm 7 Projected Gradient Method

1. Given $V, W^0, H^0, M, k = 0, \sigma = 0.01, \beta = 0.1$
2. **while** $k < M$ **do**
3. $k = k + 1$
4. **for** $j = 1 : m$ **do**
5. $x_{k-1} = H^{k-1}(:, j), h = V(:, j)$
6. Compute $g_{k-1} = \nabla f(x_{k-1})$
7. Find the first nonnegative integer t that $f(x_k) - f(x_{k-1}) \leq -\sigma\beta^t \|g_{k-1}\|^2$,$x_k = P(x_{k-1} - \beta^t g_{k-1})$
8. $H^k(:, j) = P(x_{k-1} - \beta^t g_{k-1})$
9. **end for**
10. **for** $i = 1 : n$ **do**
11. $x_{k-1} = W^{k-1}(:, j), h = V(:, j)$
12. Compute $g_{k-1} = \nabla f(x_{k-1})$
13. Find the first nonnegative integer t that $f(x_k) - f(x_{k-1}) \leq -\sigma\beta^t \|g_{k-1}\|^2$,$x_k = P(x_{k-1} - \beta^t g_{k-1})$
14. $W^k(:, j) = P(x_{k-1} - \beta^t g_{k-1})$
15. **end for**
16. **if** W^k, H^k satisfy stopping criterion **then**
17. break
18. **end if**
19. **end while**

3.4 Hierarchical ALS and its acceleration

Hierarchical ALS [8] solves subproblem by LS with rank one modification. More precisely, we solve the linear problem column by column in W, and row by row in H in one epoch's update. Here is the algorithm, written in MATLAB format in order to make the idea of row/column by row/column more clear.

Algorithm 8 HALS

Require: V, k

Initialize W and H
while not convergence **do**
  $VtW = V'W, WtW = W'W$
  for $k = 1 : r$ **do**
    $tmp = VtW(:,k)' - (WtW(:,k) * H) + WtW(k,k) * H(k,:)$
    $H(k,:) = \mathcal{P}(tmp/WtW(k,k))$
  end for
  $VHt = VH', HHt = HH'$
  for $k = 1 : r$ **do**
    $tmp = (VHt(:,k) - (W * HHt(:,k)) + (W(:,k) * HHt(k,k)))$
    $W(:,k) = \mathcal{P}(tmp/HHt(k,k))$
  end for
end while
return W, H.

Also, an accelerated version is provided in [4] and implemented in this lab, see **nmf_halsacc.m**

4 Methods based on ADMM

In this section we introduce Alternative Direction of Multiplier Method, which is powerful in handling constraint problem and non-smooth problem.

4.1 a short introduction to ADMM framework

Suppose we aim to solve a function

$$min_X f(X)$$

where X is subjecting to some constraints. We introduce an auxiliary variable Z, and consider the following augmented Lagrangian

$$L(X, Z, \alpha) = g(X, Z) + (\alpha, X - Z) + \frac{\rho}{2}\|X - Z\|^2.$$

Here $g(X, Z)$ is a splitting of $f(X)$, i.e. $g(X, X) = X$.

The ADMM provide the following framework: Usually two arg min problem does not have the closed form solution,

Algorithm 9 General ADMM

Require: f, X

Set L be the augmented Lagrangian.
$Z = X$, $\alpha = 0$.
while not converge **do**
  $Z = \text{arg min}\, L(X, Z, \alpha)$
  $X = \text{arg min}\, L(X, Z, \alpha)$
  $\alpha = \alpha + \rho\mu(X - Z)$
end while

however, if there is a splitting such that both subproblem is easy to solve, then the algorithm might be very effective, at least in convex optimization. In practice, especially in nonconvex problem, such algorithm is also widely used while some rigorous convergence analysis is lacked.

4.2 Naive ADMM

We first apply ADMM naively, the algorithm is introduced in [**?**] and we use the form shown in [9], yielding the following algorithm.

The augmented Lagrangian function is denoted as

$$L(W,H,S,T,\Lambda,\Pi) = \frac{1}{2}\|X - WH\|_F^2 + (\Lambda, W - S) + (\Pi, V - T) + \frac{\rho}{2}\|U - S\|_F^2 + \frac{\rho}{2}\|V - T\|_F^2$$

Algorithm 10 Naive ADMM

Require: f, X

Set L be the augmented Lagrangian.
$Z = X$, $\alpha = 0$.
while not converge **do**
    $W = (VH' + \rho S - \Lambda)(HH' + \rho I)^{-1}$
    $H = (W'W + \rho I)^{-1}(W'V + \rho T - \Pi)$
    $S = \mathcal{P}(W + \Lambda/\rho)$
    $T = \mathcal{P}(H + \Pi/\rho)$
    $\Lambda = \Lambda + \rho(W - S)$
    $\Pi = \Pi + \rho(H - T)$
end while

4.3 Alternating Optimizing ADMM

AO-ADMM is use ADMM to solve the subproblem in ALS, introduced in [10]. We only introduce how to solve the subproblem by ADMM, in the following algorithm.

Algorithm 11 ADMM-LS-UPDATE

Require: Y, W, H, r

$G = W'W$
$Haux = H$, $\alpha = 0$
$\rho = tr(G)/k$
for $i = 1 : iter$ **do**
    $Haux = (G + \rho I)^{-1}(W'Y + \rho(H + \alpha))$
    $H = Haux - \alpha$
    $\alpha = \alpha + H - Haux$
end for

5 A new method: LMF-ADMM

In this section we propose a new method based on LMF method of Nonlinear Least Square, and use ADMM to solve the subproblem.

5.1 Trust region method

We first recall LMF method, regarding NMF into a nonlinear least square problem: Suppose we have x_k and $J_k = \nabla r(x_k)$. Here $r(x) : \mathbb{R}^m \to \mathbb{R}^n$ is the residual function and our goal is to minimize $r'r$

Then we solve the following question to obtain the next point:

$$\min \|J_k(x - x_k) - r_k\|_F^2 + \nu\|x - x_k\|_F^2$$

Without constraint, the minimizer has a closed form:

$$x_k + (J_k' J_k + \nu I)^{-1}(J_k' r_k)$$

That is equivalent to LMF method. Inspired by this, we proposed an ANLS approach of LMF method.

Algorithm 12 LMF1

```
R = V − WH
for i = 1 : iter do
    [W+, H+] = admm_update(W, H)
    R+ = V − W+H+
    dW = W+ − W, dH = H+ − H
    Δf = ‖R‖_F^2 − ‖R+‖_F^2
    if Δf > 0 then
        W = W+, H = H+, R = R+, μ = μ/2
        ∇W = WHH' − VH', ∇H = W'WH − W'V
    else
        μ = 2μ
    end if
end for
```

or a more complicated trust region update method

Algorithm 13 LMF2

```
R = V − WH
for i = 1 : iter do
    [W⁺, H⁺] = admm_update(W, H)
    R⁺ = V − W⁺H⁺
    dW = W⁺ − W, dH = H⁺ − H
    Δg = 1/2[(dW, νdW − ∇W) + (dH, νdH − ∇H)]
    if Δg < 0 then
        ν = 4ν
    end if
    Δf = ‖R‖²_F − ‖R⁺‖²_F
    if Δf > 0 then
        W = W⁺, H = H⁺, R = R⁺, μ = μ/2
        ∇W = WHH′ − VH′, ∇H = W′WH − W′V
    else
        μ = 2μ
    end if
end for
```

5.2 ADMM solving subproblem

We introduce ADMM for solving the subproblem, since direct projection based solver will behave bad in numeric. Consider the following Augmented Lagrangian.

$$L(x, z, \alpha, \rho) = \frac{1}{2}\|J(z-u) - d\|^2 + \frac{nu}{2}\|x-u\|^2 + (\alpha, x-z) + \frac{\rho}{2}\|x-z\|^2$$

Here $x = vec(W, H)$, transforming the ensemble of W and H into a global vector, $z = vec(Waux, Haux)$ is the vectorized auxiliary variable, $\alpha = vec(\alpha_W, \alpha_H)$ is vectorized multiplier. $u = vec(W_0, H_0)$ is the starting point.

Algorithm 14 LMF-ADMM

- $\min z : u + (J'J + rho)^{-1}(J'd + \rho(x-u) + \alpha)$
- $\min x : (\nu u + \rho z - \alpha)/(\nu + \rho)$
- $x = \mathcal{P}(x)$
- update $\alpha = \alpha + \mu(x - z)$

However, this form is not computable, since we do not know the form of J. We first try to understand the mechanism of $J, J', J'J$ at point W, H

Notice that

$$J(\tilde{W}, \tilde{H}) = W\tilde{H} + \tilde{W}H$$

$$J'(\tilde{V}) = [\tilde{V}H', W'\tilde{V}]$$

$$(J'J)(\tilde{W}, \tilde{H}) = [\tilde{W}HH' + W\tilde{H}H', W'\tilde{W}H + W'W\tilde{H}]$$

Hence we can use the mapping $[\tilde{W}, \tilde{H}] = (J'J)[\tilde{W}, \tilde{H}]$ and apply krylov subspace method to solve the problem.

6 Numerical Experiment (I): Synthesis Data

In this section we test aforementioned algorithms in the $m = n = 100, r = 5$, the initial is random given and we run 1000 iterations. We test the following case:

1. actually low rank: V_1 is a non-negative low rank matrix, generated by $rand(m, r) * rand(r, n)$
2. multiplicative noise: $V_2 = V_1. * (1 + 0.01 * rand(100))$
3. additive noise :$V_3 = V_1 + (0.01) * max(V_1) * rand(100)$
4. intermediate rank $V_4 = rand(100, 10) * rand(10, 100)$
5. totally full rank $V_5 = rand(100)$

The following algorithm will be tested: MU, MUmod, MUacc, ALS, HALS, HALSacc, ALSPGD, APBB, ANLSBPP, ANLSas, ANLSgroup, ADMM, AO-ADMM, and our LMF-ADMM. Here ANLS does not appear in this report, and the solver of subproblem is written by Jingu Kim. The other methods are all introduced in previous sections. We run for 300 iterations, and for subproblem we always do 10 iterations. The rest parameter setting please refer the code. The experiment here and next section runs on my laptop, with CPU inter i7-6700HQ.

We first show Case 1-5 in 100*100 matrices.

Table 1: Case 1 for 100*100 matrices

	10	50	100	300	CPUTIME	
MU	1.82E+01	1.20E+01	4.74E+00	1.65E+00	0.11	
MUmod	1.86E+01	1.31E+01	6.94E+00	1.77E+00	0.30	
MUacc	2.31E+00	1.32E+00	7.68E-01	2.74E-01	1.44	
ALS	6.23E+00	1.50E+00	7.46E-01	1.52E-01	0.33	
HALS	1.05E+01	1.89E+00	1.38E+00	6.82E-01	0.30	
HALSacc	1.02E+01	2.13E+00	1.26E+00	3.41E-01	0.53	
APBB	2.81E+00	1.41E+00	7.96E-01	1.25E-01	6.19	
ALSPGD	2.55E+00	9.35E-01	6.03E-01	1.45E-01	0.58	
ANLSas	2.67E+00	1.02E+00	2.41E-01	9.08E-03	1.56	
ANLSgp	2.67E+00	1.02E+00	2.41E-01	9.08E-03	7.97	
ANLSbpp	2.67E+00	1.02E+00	2.41E-01	9.08E-03	1.00	
ADMM	1.99E+00	9.44E-01	7.75E-01	5.06E-02	0.22	
AO-ADMM	1.89E+00	9.22E-01	6.94E-01	5.60E-01	0.77	
LMF-ADMM	4.92E+00	1.08E+00	2.56E-01	2.49E-03	58.95	

Table 2: Case 2 for 100*100 matrices

	10	50	100	300	CPUTIME	
MU	11.73354	9.43582	6.46684	4.90432	0.18750	
MUmod	19.63202	13.88192	8.24987	3.82733	0.21875	
MUacc	4.63789	3.74934	3.59990	3.52362	1.28125	
ALS	4.96628	3.98324	3.71764	3.52806	0.35938	
HALS	7.04732	3.76370	3.61449	3.52758	0.29688	
HALSacc	5.59787	3.72378	3.61851	3.52389	0.42188	
APBB	4.35276	3.83743	3.66441	3.52452	5.82813	
ALSPGD	3.72733	3.52501	3.51194	3.50714	1.60938	
ANLSas	3.78897	3.55926	3.52209	3.50764	1.14063	
ANLSgp	3.78897	3.55926	3.52209	3.50764	7.42188	
ANLSbpp	3.78897	3.55926	3.52209	3.50764	1.12500	
ADMM	3.60405	3.58949	3.52744	3.50635	0.23438	
AO-ADMM	3.90334	3.59783	3.53715	3.50812	1.10938	
LMF-ADMM	5.05952	3.52288	3.50702	3.50629	52.53125	

Table 3: Case 3 for 100*100 matrices

	10	50	100	300	CPUTIME
MU	13.52824	8.02069	6.19232	4.38261	0.21875
MUmod	18.39742	12.46458	8.33057	3.76292	0.34375
MUacc	3.87478	1.48928	1.11853	0.96705	1.40625
ALS	NaN	NaN	NaN	NaN	f
HALS	3.47972	1.74977	1.63396	1.30570	0.32813
HALSacc	3.47727	1.74606	1.58582	1.25207	0.51563
APBB	2.70701	1.82558	1.60890	0.96966	6.37500
ALSPGD	2.15294	1.41451	1.01194	0.93130	1.40625
ANLSas	1.69059	0.96913	0.93201	0.92897	1.14063
ANLSgp	1.69059	0.96913	0.93201	0.92897	7.96875
ANLSbpp	1.69059	0.96913	0.93201	0.92897	0.85938
ADMM	1.40970	1.31235	1.35918	0.93221	0.21875
AO-ADMM	1.66494	1.02083	0.93671	0.92899	0.78125
LMF-ADMM	3.55281	0.93506	0.92922	0.92889	50.93750

Table 4: Case 4 for 100*100 matrices

	10	50	100	300	CPUTIME
MU	13.57983	9.57117	7.88564	5.40414	0.20313
MUmod	18.43720	14.24323	9.12457	1.97093	0.21875
MUacc	4.85143	1.21040	1.01626	0.95890	1.23438
ALS	4.08123	1.89589	1.86719	1.47343	0.23438
HALS	7.83307	1.66307	1.39283	1.01988	0.34375
HALSacc	7.62153	2.06372	1.52949	0.97912	0.43750
APBB	2.60248	1.60447	1.52344	1.11161	6.23438
ALSPGD	1.53281	0.97816	0.93703	0.92972	1.01563
ANLSas	1.74547	1.05425	0.97569	0.93301	1.45313
ANLSgp	1.74547	1.05425	0.97569	0.93301	7.57813
ANLSbpp	1.74547	1.05425	0.97569	0.93301	0.87500
ADMM	1.33932	1.02559	0.95570	0.92964	0.21875
AO-ADMM	2.00599	1.12812	0.99042	0.94364	0.98438
LMF-ADMM	3.35264	0.93401	0.93020	0.92893	56.48438

Table 5: Case 5 for 100*100 matrices

	10	50	100	300	CPUTIME
MU	27.19286	26.90420	26.86254	26.74390	0.21875
MUmod	27.88722	26.83411	26.56678	26.40138	0.31250
MUacc	26.79257	26.41016	26.37401	26.36624	1.15625
ALS	26.43794	26.40644	26.40803	26.42186	0.35938
HALS	26.59060	26.37620	26.36408	26.36200	0.28125
HALSacc	26.61683	26.38218	26.36590	26.36232	0.42188
APBB	26.45687	26.37201	26.36953	26.36495	6.31250
ALSPGD	26.44448	26.36726	26.36561	26.36330	1.93750
ANLSas	26.39062	26.36580	26.36292	26.36125	1.79688
ANLSgp	26.39062	26.36580	26.36292	26.36125	8.10938
ANLSbpp	26.39062	26.36580	26.36292	26.36125	1.15625
ADMM	26.37509	26.34614	26.35163	26.35347	0.42188
AO-ADMM	26.39926	26.36603	26.36315	26.36128	0.93750
LMF-ADMM	26.65626	26.46923	26.46770	26.46701	53.78125

Next we run our algorithm in some 1000*100 matrices and see their performance

Table 6: Case 1 for 1000*1000 matrices

	10	50	100	300	CPUTIME
MU	196.78	161.34	81.09	17.78	18.67
MUmod	197.61	159.68	77.02	22.46	17.78
MUacc	23.69	4.97	2.52	0.90	29.33
ALS	47.72	30.63	8.05	0.71	18.36
HALS	131.97	24.24	23.58	22.11	16.78
HALSacc	112.31	24.58	22.61	7.67	17.64
APBB	23.26	17.19	15.42	1.27	50.92
ALSPGD	26.72	12.06	3.30	2.22	12.66
ANLSas	16.97	2.41	1.06	0.28	21.64
ANLSgp	27.85	12.22	2.61	0.38	102.27
ANLSbpp	24.41	10.53	2.51	0.44	22.02
ADMM	19.45	4.32	4.90	2.89	18.17
AO-ADMM	24.28	9.01	2.38	0.41	32.05
LMF-ADMM	31.84	1.58	0.37	0.05	211.63

Table 7: Case 2 for 1000*1000 matrices

	10	50	100	300	CPUTIME
MU	198.49	151.31	74.80	24.82	18.84
MUmod	198.40	157.87	71.48	24.54	20.20
MUacc	27.19	12.39	11.35	11.07	29.23
ALS	33.36	12.50	11.25	11.04	18.03
HALS	135.15	20.84	16.39	11.35	16.92
HALSacc	74.15	22.08	14.91	11.14	19.11
APBB	26.71	23.76	20.40	11.10	50.92
ALSPGD	28.41	20.70	11.65	11.07	19.56
ANLSas	29.68	20.59	16.80	11.04	22.63
ANLSgp	27.52	19.78	13.97	11.04	98.25
ANLSbpp	27.58	19.04	17.65	11.07	21.95
ADMM	21.61	11.87	12.03	11.51	17.95
AO-ADMM	26.57	19.03	12.59	11.04	28.13
LMF-ADMM	34.14	11.10	11.03	11.03	246.45

Table 8: Case 4 for 1000*1000 matrices

	10	50	100	300	CPUTIME
MU	325.47	258.97	236.80	184.70	18.63
MUmod	325.04	260.14	244.43	184.05	18.89
MUacc	182.82	177.82	177.63	177.54	31.23
ALS	186.42	178.40	178.13	177.84	20.52
HALS	234.62	191.31	179.44	177.89	16.86
HALSacc	201.10	178.53	177.80	177.59	19.66
APBB	202.02	179.42	178.73	177.57	54.25
ALSPGD	184.14	177.68	177.53	177.52	36.44
ANLSas	179.91	177.54	177.52	177.51	22.28
ANLSgp	180.40	177.57	177.53	177.51	97.50
ANLSbpp	181.41	177.67	177.61	177.57	19.83
ADMM	180.44	177.84	177.50	177.50	18.28
AO-ADMM	183.68	178.96	177.72	177.59	30.63
LMF-ADMM	186.29	180.38	178.41	177.64	206.25

Table 9: Case 5 for 1000*1000 matrices

	10	50	100	300	CPUTIME
MU	290.52	287.66	287.04	286.37	17.05
MUmod	290.45	287.62	287.00	286.37	17.86
MUacc	286.23	286.09	286.07	286.07	29.88
ALS	286.27	286.08	286.07	286.07	16.22
HALS	287.37	286.53	286.27	286.12	14.34
HALSacc	287.03	286.27	286.15	286.09	14.27
APBB	286.47	286.13	286.09	286.07	49.22
ALSPGD	286.33	286.09	286.08	286.08	9.86
ANLSas	286.21	286.09	286.07	286.07	20.73
ANLSgp	286.21	286.08	286.07	286.07	104.97
ANLSbpp	286.19	286.08	286.08	286.07	20.61
ADMM	286.22	286.08	286.07	286.07	16.61
AO-ADMM	286.35	286.09	286.07	286.07	26.75
LMF-ADMM	286.78	286.14	286.09	286.08	225.22

7 Numerical Experiment (II): ORL and YALE data

In this section we test our algorithm in some face recognition datebase. Some available database are

1. ORL database, 400*1024
2. YALE64 database, 165*1096

Table 10: Result for ORL, $r = 25$

	10	50	100	300	CPUTIME
MU	19440.65	15744.82	12926.04	10678.80	9.03
MUmod	19416.06	15847.19	13029.62	10643.24	8.73
MUacc	10580.54	10151.85	10093.71	10019.69	26.20
ALS	16430.72	17949.73	20445.01	15497.98	7.80
HALS	11498.96	10401.59	10192.43	10043.15	6.80
HALSacc	11129.77	10229.99	10114.94	10029.77	10.36
APBB	10742.71	10176.37	10083.67	10022.82	166.30
ALSPGD	10323.89	10080.27	10042.24	10003.47	91.63
ANLSas	10356.45	10103.21	10035.58	9990.92	31.11
ANLSgp	10325.57	10115.27	10062.05	10000.40	93.67
ANLSbpp	10314.33	10047.80	9999.50	9971.23	28.20
ADMM	9762.95	9753.69	9753.68	9753.72	9.02
AO-ADMM	10370.99	10102.02	10036.93	9983.37	22.52
LMF-ADMM	17030.63	11659.28	11389.58	11384.44	1057.09

Table 11: Result for ORL, $r = 5$

	10	50	100	300	CPUTIME
MU	20085.40	17308.75	15413.77	14732.42	8.23
MUmod	20067.81	17406.64	15562.47	14787.19	6.98
MUacc	14960.11	14661.48	14647.24	14643.68	10.19
ALS	14955.29	15206.21	14928.43	14756.34	6.16
HALS	15583.35	14757.70	14691.18	14672.70	5.59
HALSacc	15888.10	14778.31	14682.50	14641.19	6.16
APBB	14881.16	14669.87	14647.22	14639.27	40.44
ALSPGD	14837.19	14756.62	14667.70	14638.19	29.34
ANLSas	14699.73	14655.41	14639.05	14634.60	9.98
ANLSgp	14801.67	14649.31	14641.85	14632.81	55.14
ANLSbpp	14774.92	14651.12	14639.58	14634.50	9.67
ADMM	14611.43	14611.36	14611.36	14611.37	6.55
AO-ADMM	14822.87	14667.14	14644.08	14634.95	12.64
LMF-ADMM	17450.14	14745.60	14702.23	14695.00	130.97

Table 12: Result for YALE64, $r = 15$

	10	50	100	300	CPUTIME
MU	34990.52	23017.40	20527.59	19485.95	11.98
MUmod	34936.58	22877.60	20558.53	19575.53	12.19
MUacc	19537.92	19110.92	19026.61	18961.13	22.02
ALS	26244.53	24230.16	23395.79	24250.73	10.38
HALS	20885.16	19223.12	19014.83	18912.83	12.31
HALSacc	20029.15	19139.15	18999.31	18822.64	14.50
APBB	19894.14	19066.42	18898.15	18810.85	344.34
ALSPGD	19466.01	19008.36	18940.59	18852.89	152.98
ANLSas	19288.68	18936.94	18857.79	18784.20	30.31
ANLSgp	19388.38	18994.20	18862.99	18835.54	184.73
ANLSbpp	19409.33	18869.82	18830.48	18821.90	27.78
ADMM	17805.45	17774.16	17763.52	17763.47	11.11
AO-ADMM	19444.01	18984.55	18921.18	18855.77	31.44
LMF-ADMM	26481.25	26200.66	26200.66	26200.66	1517.94

Table 13: Result for YALE64, $r = 5$

	10	50	100	300	CPUTIME
MU	35590.77	26896.94	25999.42	25548.82	10.64
MUmod	35549.14	27302.08	26293.49	25859.91	10.33
MUacc	25746.54	25485.63	25478.12	25477.43	15.78
ALS	26646.58	25770.85	25753.79	25754.79	8.27
HALS	26400.63	25519.81	25477.96	25477.39	8.91
HALSacc	26341.70	25525.97	25483.92	25477.39	10.25
APBB	25846.32	25682.42	25674.44	25477.39	121.55
ALSPGD	25854.18	25503.74	25478.70	25477.40	23.58
ANLSas	25708.88	25498.49	25477.40	25477.39	15.47
ANLSgp	25776.14	25482.30	25477.43	25477.39	143.20
ANLSbpp	25647.57	25478.36	25477.40	25477.39	13.86
ADMM	25312.44	25274.40	25274.49	25274.52	9.28
AO-ADMM	25769.20	25483.20	25477.41	25477.39	19.38
LMF-ADMM	28969.72	27905.80	27905.80	27905.80	326.61

We next the result also on the normalized matrices.

Table 14: Result for YALE64/256, $r = 5$

	10	50	100	300	CPUTIME
MU	140.31	125.42	124.36	124.08	9.88
MUmod	139.84	126.54	124.81	124.08	9.19
MUacc	124.33	124.06	124.06	124.06	13.22
ALS	124.34	124.14	124.14	124.14	8.88
HALS	129.00	125.47	124.06	124.06	7.92
HALSacc	126.32	124.07	124.06	124.06	9.28
APBB	78.41	74.29	74.01	73.44	196.33
ALSPGD	78.18	77.13	77.13	77.13	3.59
ANLSas	76.19	74.07	73.60	73.38	28.48
ANLSgp	75.69	73.96	73.82	73.76	191.20
ANLSbpp	76.37	74.18	73.57	73.38	29.59
ADMM	70.75	69.44	69.44	69.49	10.69
AO-ADMM	76.28	74.08	73.73	73.49	30.44
LMF-ADMM	157.33	157.33	157.33	157.33	801.42

Net we show the performance on TR11. We first normalize the data.

Table 15: Result for TR11, $r = 10$

	10	50	100	300	CPUTIME
MU	0.55	0.42	0.41	0.40	43.06
MUmod	0.50	0.40	0.40	0.40	42.42
MUacc	0.40	0.37	0.36	0.36	75.27
ALS	0.37	0.36	0.36	0.36	41.00
HALS	0.40	0.37	0.37	0.36	42.83
HALSacc	0.39	0.37	0.36	0.36	51.14
APBB	1.84	1.84	1.84	1.84	1.33
ALSPGD	1.81	1.81	1.81	1.81	0.84
ANLSas	0.38	0.38	0.36	0.36	68.22
ANLSgp	0.37	0.36	0.36	0.36	393.92
ANLSbpp	0.37	0.36	0.36	0.36	65.41
ADMM	1.81	1.76	1.27	1.00	49.47
AO-ADMM	1.81	0.45	0.37	0.36	89.28
LMF-ADMM	16.52	1.82	1.82	1.82	564.94

8 Discussion and Improvement

8.1 Discussion

From the experiment we can find that if the rank is small or the residual is small, then LMF-ADMM method behaves well, since in the small residual case, the LMF is a good choice of NLS. However, if the rank is too large, the LMF behaves very poor. As the experiment indicates, in 100*100 case our proposed method is powerful, but things changed in 1000*1000 case and real datebase.

8.2 Possible Improvement of LMF-ADMM

Several improvement can be proposed to alleviate the small-residual-dependence. For example, we can use sequencing optimization. Suppose $r = r_1 r_2$, then we can do r_1 times of rank r_2 LMF-ADMM. Practically it can reach the performance of MU and ALS method in YALE64.

References

[1] Gillis N. Nonnegative matrix factorization: Complexity, algorithms and applications[J]. Unpublished doctoral dissertation, Universit catholique de Louvain. Louvain-La-Neuve: CORE, 2011.

[2] Lee D D, Seung H S. Algorithms for non-negative matrix factorization[C]//Advances in neural information processing systems. 2001: 556-562.

[3] Lin C J. On the convergence of multiplicative update algorithms for nonnegative matrix factorization[J]. IEEE Transactions on Neural Networks, 2007, 18(6): 1589-1596.

[4] N. Gillis and F. Glineur,"Accelerated Multiplicative Updates and hierarchical ALS Algorithms for Nonnegative Matrix Factorization"

[5] Gonzalez E F, Zhang Y. Accelerating the Lee-Seung algorithm for nonnegative matrix factorization[R]. 2005.

[6] Lin C J. Projected gradient methods for nonnegative matrix factorization[J]. Neural computation, 2007, 19(10): 2756-2779.

[7] Han, Lixing Neumann, Michael Prasad, Upendra. (2010). Alternating projected Barzilai-Borwein methods for Nonnegative Matrix Factorization. Electronic transactions on numerical analysis ETNA. 36. 54-82.

[8] Cichocki A, Phan A H. Fast local algorithms for large scale nonnegative matrix and tensor factorizations[J]. IEICE transactions on fundamentals of electronics, communications and computer sciences, 2009, 92(3): 708-721.

[9] Song D, Meyer D A, Min M R. Fast nonnegative matrix factorization with rank-one admm[C]//NIPS 2014 Workshop on Optimization for Machine Learning (OPT2014). 2014.

[10] Huang K, Sidiropoulos N D, Liavas A P. A flexible and efficient algorithmic framework for constrained matrix and tensor factorization[J]. IEEE Transactions on Signal Processing, 2016, 64(19): 5052-5065.