Solving Nonnegative Principal Component Analysis Using the Frank-Wolfe and Manifold Proximal Gradient Methods

Jiayi Lei
SID: 915016668
Email: cjlei@ucdavis.edu

Undergraduate Thesis Writing
Mentor: Shiqian Ma
Spring Quarter 2019

Abstract

Millions of data points are generated at an exponential speed nowadays. Extracting essential information from this pile of data is an active area of research. Principal Component Analysis (PCA) is a fundamental tool for this purpose. In some applications, the principal vectors are expected to lie in the non-negative orthant, which leads to the non-negative Principal Component Analysis (NNPCA). The objective function for NNPCA is non-convex and the non-negative constraint makes it even harder to solve. In this paper, we propose the Frank-Wolfe algorithm (FW) and Manifold Proximal Gradient Method (ManPG) to solve NNPCA. Through intensive numerical study on many examples, it is shown that ManPG is better than FW with less number of iterations and faster cpu time.

1 Introduction

Principal Component Analysis (PCA) is often used to reduce the dimensionality of high-dimensional data points. Extracting the most important features of the data set is very important in analyzing data. However in many applications like in hyperspectral imaging [1] , non-negative PCs are preferable due to their physical meanings. This leads to the non-negative PCA (NNPCA) problem. Applications of NNPCA arise in many fields,

such as gene expression, signal processing and non-negative matrix factorization [2].

$$\begin{array}{rrl} \max & \sum_{i=1}^{n} < v_i, x >^2 & \text{Classical PCA} \\ \text{s.t.} & ||x||_2 = 1 & \\ \max & \sum_{i=1}^{n} < v_i, x >^2 & \text{Nonnegative PCA} \\ \text{s.t.} & ||x||_2 = 1 & \\ & x \geq 0 & \end{array} \tag{1}$$

Classical PCA can be solved by the power method efficiently. However, adding the non-negative constraints makes the NNPCA problem much more difficult, mainly due to the following two reasons: (i) The problem is non-convex. It is known that non-convex problems are usually much harder to solve than convex problems. (ii) The problem is non-smooth due to the non-negative constraints. In this paper, the Frank-Wolfe (FW) algorithm, Manifold Gradient Descent (ManGD) method, and the Manifold Proximal Gradient Method (ManPG) [3] methods will be used to solve NNPCA. Also the comparison of these three methods when solving NNPCA will be covered.
We use the following notation throughout this paper.

- $X \in \mathbb{R}^{n \times p}$ the matrix with rows $v_1, v_2, v_3, ..., v_n$.
- $F(x) := x^T A x$, where A is a symmetric positive definite matrix

The objective function in NNPCA can be transformed as

$$\begin{array}{rl} & \max \quad \sum_{i=1}^{n} < v_i, x >^2 \\ = & \max \quad ||Vx||_2^2 \\ = & \max \quad (Vx)^T (Vx) \\ = & \max \quad x^T V^T V x \\ = & \max \quad x^T A x, \end{array} \tag{2}$$

where $A := V^T V$ is a symmetric positive definite matrix.
We have the following theorem:

Theorem 1. *If $A \in \mathbb{R}^{n \times n}$ is symmetric, then A has exactly n orthogonal eigenvectors $x_1, x_2, x_3, ..., x_n$, with possibly repeated eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. In other words, there exists an orthogonal matrix X of eigenvectors and a diagonal matrix D of eigenvalues such that $D = X^T A X$. This algorithm is called Spectral Theorem.*

2 Frank-Wolfe Method

2.1 Introduction of Frank Wolfe Method

The classical Frank-Wolfe method was proposed by Marguerite Frank and Philip Wolfe in 1956 [7]. The algorithm is used to solve the general constrained convex optimization problems of the form

$$\min_{x \in \mathbb{D}} f(x)$$

where $f(x)$ is convex and continuously differentiable, and the domain $\mathbb{D}$ is a compact convex subset of any vector space. The Frank-Wolfe method is also known as the *conditional gradient method*.

Algorithm 1 Classical Frank-Wolfe algorithm

1: **Input:** Initial point $x_0 \in \mathbb{M}$
2: **for** k = 0 to K **do**
3: solve $x_k^+ = \arg\min_{x \in \mathbb{M}} < x, \nabla f(x_k) >$
4: solve $x^{k+1} := (1-\gamma)x_k + \gamma x_k^+$, for $\gamma := \frac{2}{k+2}$
5: **end for**

There are many variants of Frank-Wolfe algorithm corresponding to different situations. The Frank-Wolfe algorithm to solve non-convex functions is given below [3].

Algorithm 2 Frank-Wolfe algorithm (non-convex variant)

1: **Input:** Initial point $x_0 \in \mathbb{M}$ threshold *tol*
2: **for** k = 0 to K **do**
3: solve $x_k^+ = \arg\min_{x \in \mathbb{M}} < x, \nabla f(x_k) >$
4: direction $d_k := x_k^+ - x_k$
5: FW-gap $g_t := -\nabla f(x_k)^T d_k$
6: **if** $g_k \leq tol$ **then return** x_k
7: **end if**
8: $\gamma_k := \min\{g_k/C, 1\} \forall C > C_f$
9: Updae $x_{k+1} = x_k + \gamma d_k$
10: **end for**
11: **return** x_k

In this paper, for the sake of simplicity, we let $\gamma = 1$, thus $x_{k+1} = x_k + (x_k^+ - x_k)$, and then we normalize x_{k+1} because of the constraint of $||x|| = 1$

Since $F(x)$ is not convex, the classical Frank-Wolfe algorithm is not suitable to solve the NNPCA problem. However, many variants of Frank-Wolfe algorithm were proposed. One of them is to solve non-convex optimization problem [3].

$$\begin{aligned} \min_{x} \quad & < x, b > \\ \text{s.t.} \quad & ||x||_2 = 1 \\ & x \geq 0 \end{aligned} \tag{3}$$

Lemma: Define $b^- = -\min\{b, 0\}$, the elements are taken element-wise, the solution for the equation (3) is as follow: [4]

$$x^* = \begin{cases} \frac{b^-}{||b^-||}, & \text{if } b^- \neq 0 \\ e_i, & \text{otherwise} \end{cases} \tag{4}$$

2.2 Using Frank-Wolfe to solve NNPCA

$\nabla f(x) = -2Ax$, we randomize an initial point x_0 and solve $x_k^+ = \arg\min_{x \in \mathbb{M}} < x, \nabla f(x_k) >$ by using the lemma above. For simplicity, we let $\gamma_l = 1$, thus $x_{k+1} = x_k + \gamma d_k = x_k + (x_k^+ - x_k) = x_k^+$, we then normalize x_{k+1} to bring it back to the feasible region. Repeat this step until the difference between x_k and x_{k+1} is smaller than some ε.

$$\begin{aligned} \min \quad & < x, \nabla f(x_k) > \\ \text{s.t.} \quad & ||x||_2 = 1 \\ & x \geq 0 \end{aligned} \tag{5}$$

3 Manifold Gradient Descent method

Manifold Gradient Descent Method is a type of method that aims to minimize functions define on a manifold domain. Algorithms for solving manifold optimization with smooth objective were well developed, including conjugate gradient methods, Newton-type methods and trust region methods [4] .
Definition: A manifold is a topological space that locally resembles Euclidean space near each point.
Definition: A smooth function is a function that has continuous derivatives up to some desired order over some domain.

3.1 Introduction of Manifold Proximal Gradient Method(ManPG)

Development of ManPG

Manifold Proximal Gradient Method(ManPG) is an algorithm proposed by Chen, Ma, So and Zhang in 2018 [5]. The algorithm aims to solve manifold optimization problems with non-smooth objectives. It is based on the proximal gradient method with a retraction operation to keep the iterates feasible with respect to the manifold constraint. The form of manifold optimization problem that ManPG is targeting is:

$$\min F(X) := f(X) + h(X), s.t., X \in \mathbb{M}$$

where f is smooth, possibly non-convex, h is convex, possibly non-smooth, $\mathbb{M}$ is a Riemannian manifold embedded in the Euclidean space.

Algorithm 3 Manifold Proximal Gradient Method (ManPG)

1: **Input:** Initial point $x_0 \in \mathbb{M}, \delta \in (0,1), \gamma \in (0,1)$, Lipschitz constant L for $\nabla f(x)$
2: **for** k = 0 to K **do**
3: Obtain d_k by solving problem (1) with $t \in (0, 1/L]$;
4: Set $\alpha = 1$
5: **while** $F(Retr_{x_k}(\alpha d_k)) > F(x_k) - \delta\alpha||d_k||_F^2$ **do**
6: $\alpha = \gamma\alpha$
7: **end while**
8: Set $x_{k+1} = Retr_{x_k}(\alpha d_k)$
9: **end for**

In the above psudocode, $Retr_{x_k} = x_k + \alpha d_k$, $d_k = x_k^+ - x_k$
Definition: Given an open set $\mathbb{B} \subseteq \mathbb{R}^m$, we say f is Lipschitz-continuous on the open subset $\mathbb{B}$ if there exists a constant $L \in \mathbb{R}^+$, which is called the Lipschitz constant of f on $\mathbb{B}$, such that:

$$||f(x) - f(y)|| \leq L||x - y|| \qquad \forall x, y \in \mathbb{B}$$

Noticed that the norm ||.|| can be any norm. However, once a norm has been chosen one should stick to that single norm, as the Lipschitz constant L depends on the particular choice of this norm.

$$\begin{aligned} \min_{d_k \in \mathbb{T}_{x_k}\mathbb{M}} \quad & < \nabla f(x_k), d_k > + \tfrac{1}{2t}||d_k||_F^2 + h(x_k + d_k) \\ \text{s.t.} \quad & d_k \in \mathbb{T}_{x_k}\mathbb{M} \\ & x \geq 0 \end{aligned} \tag{6}$$

3.2 Use ManPG to Solve NNPCA

In NNPCA, the objective function is

$$F(x) := f(x) = -x^T A x$$

where $f(x)$ is non-convex and non-smooth in the region where $||x|| = 1$, and $x \geq 0$. Notice that in NNPCA problem, $h(x) = 0$
By the Manifold Proximal Gradient method, we have the subproblem of:

$$\min < \text{grad} f(x_k), x - x_k > + \frac{1}{2t}||x - x_k||_F^2$$

$$s.t. x^T x^k = 1$$

$$||x^k||^2 = 1$$

where $t > 0$ is the step size, x is in the tangent space to the domain at the point x^k, and grad f is the Riemannian gradient. Also note that grad $f(x)$ is the orthogonal projection of $\nabla f(x)$ onto the tangent space.
Following the definition of grad f, we have:

$$\langle \text{grad} f(x^k), x \rangle = \langle \nabla f(x^k), x \rangle \qquad \forall x \in T_{x_k}\mathbb{M}$$

We can rewrite the above objective function as:

$$\min_x \langle \text{grad} f(x_k), x \rangle + \frac{1}{2t}||x - x_k||_F^2$$

$$\min_x x^T \nabla f(x^k) + \frac{1}{2t}(x^T x - 2t x^T x_k + x_k^T x_k)$$

Which is equivalent to the objective function below, since $x^T x^k = 1, ||x^k||^2 = 1$

$$\min x^T \nabla f(x^k) + \frac{1}{2t} x^T x$$

Similarly, since $(t\|\nabla f(x_k)\|)^2$ is not affected by the choice of x, the above objective function is equivalent to the equation below:

$$\begin{aligned} \min \quad & \tfrac{1}{2t}(x^T x + 2t x^T \nabla f(x_k) + (t\|\nabla f(x_k)\|)^2) \\ \text{equivalent to: } \min \quad & ||x + t\nabla f(x_k)||^2. \end{aligned} \tag{7}$$

The objective function can be solved by solving the subproblem below with $c = -t\nabla f(x_k) = 2tAx_k$.
SUBPROBLEM

$$\begin{aligned} \min \quad & ||x - c||^2 \\ \text{s.t.} \quad & a^T x = 1 \\ & x \geq 0 \end{aligned} \tag{8}$$

Assume the minimizer is x^*.

$$x^* := P_{[0,\infty)}(c - \lambda a)$$

where c and a are both vectors with the same dimention of n and λ is a scalar. Also, a is a non-negative vector. λ is a solution to $a^T x^* = 1$ [6] Thus we have:

$$\sum_{i=1}^{n} a_i * x_i^* = 1$$

since

$$x_i^* = \max\{0, c_i - \lambda a_i\}$$

$x_i^* = c_i - \lambda a_i$ if and only if when $\lambda < \frac{c_i}{a_i}$

Case 1. $x_i^* = 0, \forall i$, in which case $a^T x = 1$ can not be achieved.

Case 2. $x_i^* = c_i - \lambda a_i, \forall i$, in which case, $x^* = c - \lambda a$. Since $a^T x^* = 1$, we have $a^T c - \lambda a^T a = 1$, and since c and a are given, it is easy to get λ. After getting λ, check whether the lowest ratio $\frac{c_i}{a_i}$ is greater or equal to the chosen λ.

Case 3. For some $i \in L$, $x_i^* = 0$; for $i \in N \setminus L$, $x_i^* = c_i - \lambda a_i$. N is an index set of n integers, L is a subset of set N. In this case, we need to figure out the index set of L, such that $\lambda < \frac{c_i}{a_i}, \forall i \in N \setminus L$ and $\sum_{i=1}^{n} a_i * x_i^* = 1$.
Notice that when $a_i = 0, x_i^* = \max\{c_i, 0\}$, when considering λ, we can ignore the the index i when $a_i = 0$, since $a_i * x_i^* = 0$ no matter what x_i^* is. We store those index and a_c pairs in set $\mathbb{J}$ Noticed that $\mathbb{J} \subset N \setminus L$, and store the corresponding ratios in set $\mathbb{S}_0$.
For $a_i > 0$, sort the value of $\frac{c_i}{a_i}, \forall i \in N \setminus \mathbb{J}$ in the ascending order. Sorted indexes are stored in set $\mathbb{S}$, the corroponding indexes in vector a and vector c are stored in $\mathbb{I}$
After showing that Case 2 is not possible, we can start "muting" the a_c pair from the lowest ratio until we find the λ that satisfies all requirements. That means while letting $x_i = 0$ we puting index i into set $\mathbb{L}$, and set the rest elements in x as $x_i = c_i - \lambda a_i$. Since all c_i and a_i are known, by using equation that $a^T x^* = 1$, and $x_i^* = \max\{0, c_i - \lambda a_i\}$, we can find λ easily. After that we can check whether λ is less than all other ratios, if yes, that is the λ we want, if not, then continue to mute the next a _ c pair and repeat the process above until we find the right λ and thus x^*.

ManPG is a better algorithm because we do not need to choose a good step size t and α in order to let the algorithm converge, we can choose random γ and δ as inputs. In contrast, when using the *cvx* package from Matlab, two parameters t and α need to be selected manually and the choice of

Algorithm 4 Quadratic Norm Minimizer

procedure QUADRATIC NORM MINIMIZER
  $\mathbb{E} \leftarrow \{i | a_i = 0\}$
  $[\mathbb{R}, \mathbb{I}] \leftarrow \text{sort} \frac{c_i}{a_i}, \forall i \in \mathbb{N} \setminus \mathbb{E}$
  $\mathbb{U} \leftarrow []$
  $j \leftarrow 0$
  do
    $\lambda = \frac{\sum_i a_i c_i - 1}{\sum_i a_i a_i}, \forall i \in \mathbb{I} \setminus \mathbb{U}$
    $j \leftarrow j + 1$
    $\mathbb{U} \leftarrow [\mathbb{U}, I(j)]$
  while $\lambda \geq R[j]$
  $x_i \leftarrow 0, \forall i \in \mathbb{U}$
  $x_i \leftarrow max\{c_i - \lambda a_i\}, \forall i \in \mathbb{N} \setminus \mathbb{U}$
end procedure

these two parameters have huge impact on the performance of the Matlab built in function, sometimes the algorithm does not even converge. This has been confirmed by intensive experiments.
To find the Lipschitz constant L for $\nabla f(x)$, we have:

$$\nabla f(x) = -2Ax$$
$$|| - 2Ax - (-2Ay)|| \leq L||x - y||, \forall x, y \in \mathbb{M}$$
$$|| - 2A(x - y)|| \leq L||x - y||$$
$$||2A(x - y)|| \leq 2||A||||x - y|| \leq L||x - y||$$
$$\text{thus } L = 2||A||$$

Notice that we can use either the Euclidean norm or the Frobenius norm here. But a lot of experience, we observed that it does not make a big difference in the result about whether we use Euclidean norm or Frobenius norm. Details about the comparisoin of these two ways to measure the Lipschitz number is attached at the Appendix. In this paper we can will use Euclidean norm.
When using ManPG to solve NNPCA, we set $\delta = 0.5, \gamma = 0.5, t = 1/L$. By experiment, we can see that compared to setting t to be a random number $\in (0, L]$, when setting $t = 1/L$, ManPG converges in less iterations and uses slightly less CPU time. The table of comparison is attached in the appendix.

4 Numereical Results: Compare ManPG and FW

In order to compare the performance of ManPG and FW, we try to avoid the difference result from problems themselves. Hence 60 symmetric positive definite matrices of different sizes n of 10, 20, 50, 100, 500 and 1000, 10 matrices for each size, are pre-generated along with the dominate non-negative eigenvector stored. But intensive experiments, the rank of the matrix does not affect the performance gradient descent type algorithm finding dominate non-negative eigenvector, hence we set the rank of all matrices to be 20, except for matrices of size 10, the rank is 10. Pesudocode about how to generate the matrix is attached in the Appendix.

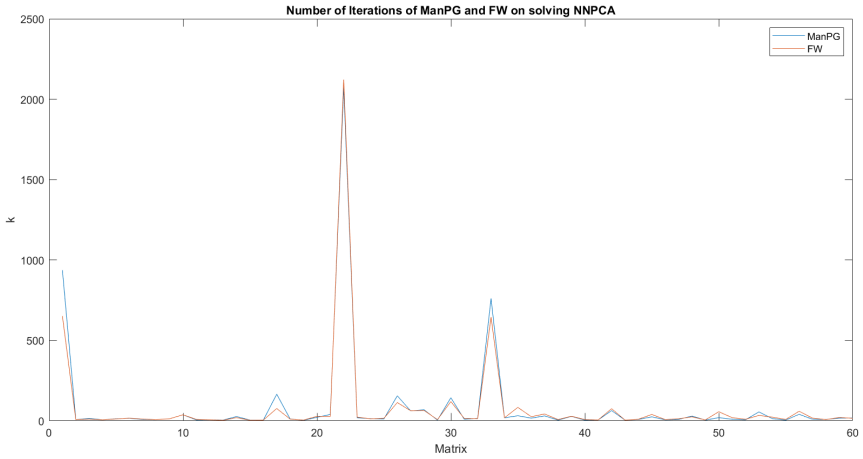

Figure 1: Comparision of Iteration Number of ManPG and FW

In Figure 1 and Figure 2, even though the two lines are not always overlapping, i.e the same k for each matrix, the overall pattern is similar. ManPG and FW have similar behavior in terms of the number of iterations for the 60 sample problems. It takes more iterations for "harder" problems and less iterations for "easier" problems.
For CPU time, the two lines also have a similar pattern, even though the similarity is lower than the similarity for the iteration numbers. However, since CPU time will be affected by not only the efficiency of the algorithms themselves but also the computer itself and the workload of the computer. The result of CPU time might not reflect the efficiency of the algorithms completely.
Details of number of iteration and CPU time for each matrix is attached at

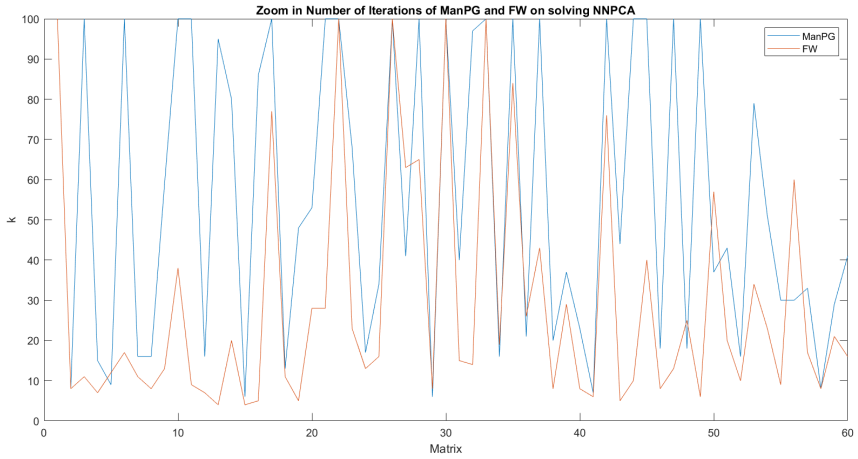

Figure 2: Zoom in Comparision of Iteration Number of ManPG and FW

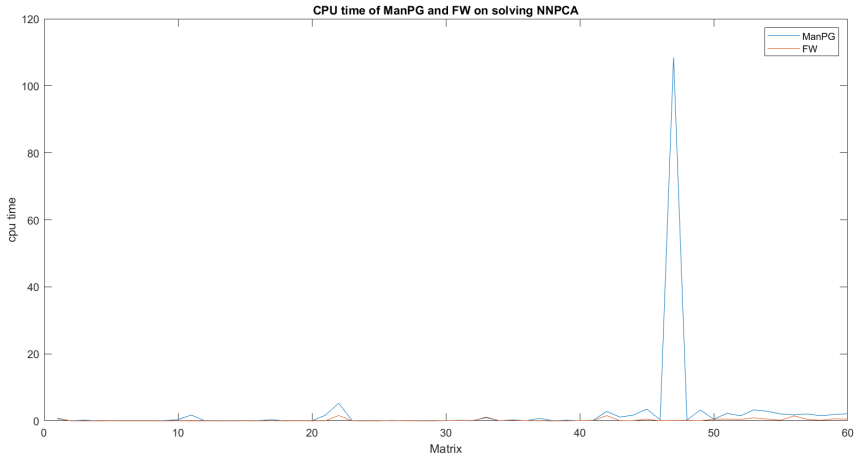

Figure 3: Comparision of CPU Time of ManPG and FW

Appendix.
We suspect whether the condition number affects the performance of different algorithms, i.e. maybe one algorithm performs better on matrices with higher condition number and maybe other algorithms perform better on matrices with a lower condition number. Thus we have the graphs below to investigate this question. The horizontal axis are condition numbers for those 60 pre-generated matrices and the vertical axis is the difference of number of iteration or CPU time between any two algorithms. It is hard to see any pattern in these graphs and thus there is no conclusion about

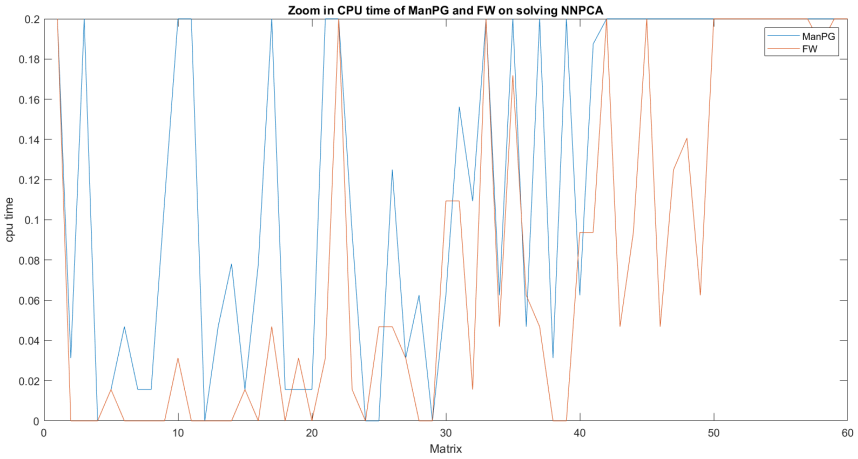

Figure 4: Zoom in Comparision of CPU Time of ManPG and FW

how condition number of the matrix affects the performance of ManPG and FW over each other.

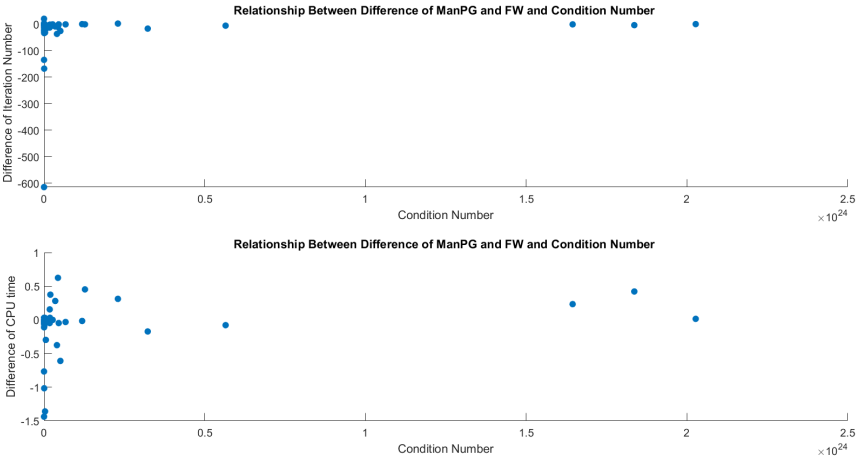

Figure 5: Relationship Between Condition Number and Difference in Performance of Different Algorithms

We take the average of all number of iteration and all CPU time by using ManPG and FW respectively for 10 matrices of the same size and analyze the effect of the size of matrix to the performance of both algorithms. While it is natural that it takes longer CPU time to solve matrices with bigger size, it is surprising that the size of matrix does not affect the number of iteration.

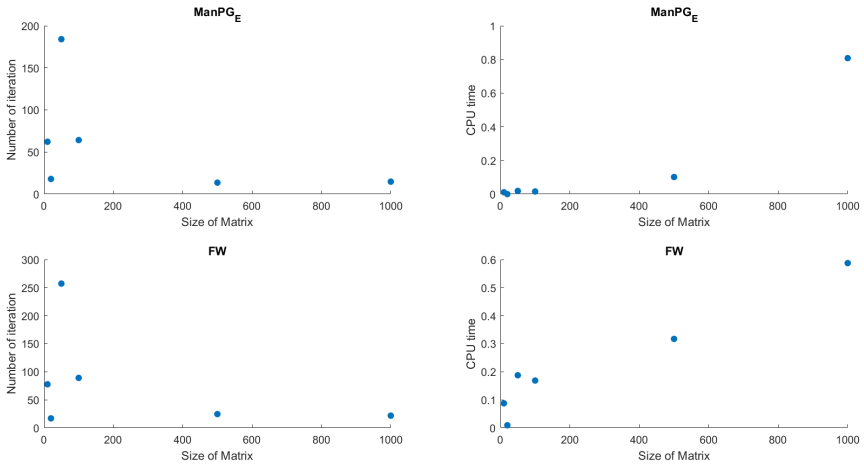

Figure 6: Relationship Between Matrix size and Difference in Performance of Different Algorithms

5 Conclusion

Nonnegative principal analysis is a hard problem to solve due to its nonsmooth nonconvex feature. In this paper, we solved the nonegative principal analysis (NNPCA) problem by using Frank Wolfe (FW) algorithm and Manifold Proximal Gradient Method (ManPG). We compared the numerical result from both algorithms and found that ManPG is better than FW with less number of iterations and slightly less CPU time. But overall, their performance on different matrices are similar. Our experiment suggested the size of matrix will affect the computatoin cpu time but not the number of iteration. Also, when we set $t = 1/L$, where L is the Lipschitze constant of $\nabla f(x)$, ManPG performs the better than setting t to be a random number between 0 and $1/L$. We believe that ManPG has greater potential in solving nonsmooth optimization problem than FW.

References

[1] Bajorski, P *Application of nonnegative principal component analysis in hyperspectral imaging* Proc. SPIE 6302, Imaging Spectrometry XI, 63020G (1 September 2006); doi: 10.1117/12.677375

[2] Montanari, A. and Richard, E. (2016) *Non-Negative Principal Component Analysis: Message Passing Algorithms and Sharp Asymptotics.*. IEEE Transactions on Information Theory, 62(3), pp.1458-1484.

[3] Zhao, H. and Gordon, G. (2018) *Frank-Wolfe Optimization for Symmetric-NMF under Simplicial Constraint* (German) [online] arXiv.org. Available at: https://arxiv.org/abs/1706.06348 [Accessed 15 May 2019].

[4] Frank, Marguerite and Wolfe, Philip (1956) *An Algorithm for Quadratic Programming* Naval Research Logistics, 3: 95-110. doi:10.1002/nav.3800030109

[5] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds.* Princeton University Press, 2009.

[6] Zhang, J., Ma, S. and Zhang, S. (2019). *Primal-Dual Optimization Algorithms over Riemannian Manifolds.* an Iteration Complexity Analysis. [online] arXiv.org. Available at: https://arxiv.org/abs/1710.02236 [Accessed 15 May 2019].

[7] Parikh, N. and Boyd, S. (2014) *Proximal Algorithms.*. Foundations and Trends in Optimization, 1(3), pp.127 - 239.

Appendix

Analyzing the effect from size of matrix on number of iteration			
n	ManPG_E	ManPG_Fro	FW
10	62.2	106.4	77.7
20	18	25.9	17
50	184.1	259.4	257.2
100	64.2	92.4	89.1
500	13.6	18	24.6
1000	14.8	18.7	21.8

Analyzing the effect from size of matrix on cpu time			
n	ManPG_E	ManPG_Fro	FW
10	0.010938	0.010938	0.0875
20	0	0.00625	0.009375
50	0.01875	0.021875	0.1875
100	0.015625	0.017188	0.16875
500	0.101563	0.057813	0.317188
1000	0.807813	0.1875	0.5875

In bit table, we analyze the effect from choosing t and L in ManPG algorithm. Where ManPG_E means using Euclidean norm in the algorithm and ManPG_F means using Frobenius norm. k means the number of iteartion,

	ManPG with $t \in (0, 1/l]$				ManPG with t = 1/L				Difference			
	ManPG_Fro		ManPG_E		ManPG_Fro		ManPG_E		ManPG_Fro		ManPG_E	
matrix	k	time	k	time	k	time	k	time	k	time	k	time
A_10_10_1	5396	0.953125	1250	0.5625	517	0.0625	937	0.109375	4879	0.890625	313	0.453125
A_10_10_2	8	0	9	0	6	0	8	0	2	0	1	0
A_10_10_3	19	0	22	0	14	0	15	0	5	0	7	0
A_10_10_4	142	0.015625	20	0	6	0.03125	7	0	136	-0.01563	13	0
A_10_10_5	654	0.03125	51	0.015625	10	0.015625	13	0	644	0.015625	38	0.015625
A_10_10_6	126	0.015625	32	0.03125	14	0	16	0	112	0.015625	16	0.03125
A_10_10_7	37	0	24	0	9	0	9	0	28	0	15	0
A_10_10_8	16	0	194	0.015625	8	0	8	0	8	0	186	0.015625
A_10_10_9	63	0	35	0	11	0	13	0	52	0	22	0
A_10_10_10	111	0	39	0	27	0	38	0	84	0	1	0
A_20_20_1	163	0.015625	17	0	6	0	6	0	157	0.015625	11	0
A_20_20_2	23	0	120	0.015625	6	0	7	0	17	0	113	0.015625
A_20_20_3	14	0	9	0	5	0	5	0.03125	9	0	4	-0.03125
A_20_20_4	22	0	47	0	20	0	27	0	2	0	20	0
A_20_20_5	46	0	36	0	5	0	5	0	41	0	31	0
A_20_20_6	7	0	15	0	5	0	5	0	2	0	10	0
A_20_20_7	172	0	173	0.015625	97	0	166	0.03125	75	0	7	-0.01563
A_20_20_8	18	0	87	0.015625	9	0	12	0	9	0	75	0.015625
A_20_20_9	6	0	15	0	5	0	4	0	1	0	11	0
A_20_20_10	770	0.265625	86	0.015625	22	0	22	0	748	0.265625	64	0.015625
A_50_20_1	62	0	30	0	27	0	40	0	35	0	-10	0
A_50_20_2	7095	0.5625	2617	0.3125	1507	0.15625	2073	0.15625	5588	0.40625	544	0.15625
A_50_20_3	35	0	332	0.03125	15	0	19	0	20	0	313	0.03125
A_50_20_4	29	0	10	0.015625	11	0	14	0	18	0	-4	0.015625
A_50_20_5	17	0	21	0.015625	11	0	12	0	6	0	9	0.015625
A_50_20_6	205	0.015625	95	0	106	0	156	0.03125	99	0.015625	-61	-0.03125
A_50_20_7	124	0.015625	55	0	38	0.03125	62	0	86	-0.01563	-7	0
A_50_20_8	298	0.015625	80	0.015625	31	0	69	0.03125	267	0.015625	11	-0.01563
A_50_20_9	97	0.015625	102	0.015625	5	0	5	0	92	0.015625	97	0.015625
A_50_20_10	305	0.03125	108	0.015625	90	0	144	0	215	0.03125	-36	0.015625
A_100_20_1	11	0	14	0	10	0	10	0	1	0	4	0
A_100_20_2	25	0	45	0.015625	11	0	16	0	14	0	29	0.015625
A_100_20_3	820	0.5	4295	0.84375	477	0.125	761	0.125	343	0.375	3534	0.71875
A_100_20_4	58	0	695	0.125	13	0	19	0	45	0	676	0.125
A_100_20_5	543	0.234375	266	0.046875	67	0	32	0	476	0.234375	234	0.046875
A_100_20_6	112	0.046875	18	0	7	0	17	0.046875	105	0.046875	1	-0.04688
A_100_20_7	43	0	30	0	23	0	31	0	20	0	-1	0
A_100_20_8	71	0.046875	32	0	5	0.03125	5	0	66	0.015625	27	0
A_100_20_9	27	0	20	0	24	0	28	0	3	0	-8	0
A_100_20_10	7	0	6	0	5	0	5	0	2	0	1	0
A_500_20_1	56	0.328125	6	0.0625	6	0.078125	6	0	50	0.25	0	0.0625
A_500_20_2	68	0.390625	46	0.203125	44	0.21875	64	0.203125	24	0.171875	-18	0
A_500_20_3	5	0.03125	61	0.28125	5	0.0625	5	0	0	-0.03125	56	0.28125
A_500_20_4	12	0.046875	13	0.078125	9	0.0625	9	0.046875	3	-0.01563	4	0.03125
A_500_20_5	111	0.765625	19	0.109375	22	0.203125	25	0.09375	89	0.5625	-6	0.015625
A_500_20_6	8	0.046875	12	0.09375	6	0.078125	7	0.046875	2	-0.03125	5	0.046875
A_500_20_7	64	0.484375	32	0.1875	7	0.046875	9	0.03125	57	0.4375	23	0.15625
A_500_20_8	38	0.21875	158	0.640625	12	0.09375	30	0.09375	26	0.125	128	0.546875
A_500_20_9	17	0.0625	32	0.140625	5	0.0625	5	0	12	0	27	0.140625
A_500_20_10	197	1.40625	607	2.3125	20	0.109375	20	0.0625	177	1.296875	587	2.25
A_1000_20_1	30	0.578125	20	0.828125	9	0.6875	10	0.078125	21	-0.10938	10	0.75
A_1000_20_2	52	1.15625	1453	16.46875	12	0.75	7	0.09375	40	0.40625	1446	16.375
A_1000_20_3	48	0.875	61	1.25	33	0.84375	56	0.5625	15	0.03125	5	0.6875
A_1000_20_4	85	1.640625	20	0.765625	12	1.15625	14	0.109375	73	0.484375	6	0.65625
A_1000_20_5	66	1.359375	7	0.765625	5	0.625	5	0.046875	61	0.734375	2	0.71875
A_1000_20_6	39	0.796875	145	2.328125	34	0.890625	41	0.40625	5	-0.09375	104	1.921875
A_1000_20_7	31	0.84375	21	0.90625	8	0.71875	10	0.09375	23	0.125	11	0.8125
A_1000_20_8	22	0.390625	16	0.921875	7	0.640625	9	0.078125	15	-0.25	7	0.84375
A_1000_20_9	62	1.296875	15	0.78125	13	1	17	0.21875	49	0.296875	-2	0.5625
A_1000_20_10	244	5.015625	22	1.15625	15	0.765625	18	0.1875	229	4.25	4	0.96875
								number of nonegative number:	58	26	48	35
								percentage:	96.67%	43.33%	80.00%	58.33%

Algorithm 5 Generate Matrix

1: **Input:** size of matrix n, rank of matrix r
2: Generate a vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_r)'$, where $\lambda_1 \geq \lambda_2 \geq \lambda_3 ... \geq \lambda_r$
3: Generate a n by 1 nonnegative vector u_1
4: Compute the other $r-1$ orthogonal vectors $u_2, u_3, ...u_r$
5: Generate matrix $A := \sum_{i=1}^{r} u_i \lambda_i u_i^T$
6: **Output:** matrix A and dominate singular vector u_1

time measures the cpu time of the algorithm. Difference in the last four columns are calculated by subtracting data at the 5-8 column from corresponding data at 1-4 column. The conclusion is that in terms of number of iteration, setting $t = 1/L$ will leads to less number of iterations, since the percentages of positive number in the difference column are both above 80% for both ways of choosing the Lipschtiz number. However, in terms of cpu time, the way of choosing t does not matter. We can also conclude that the way of choosing the Lipschtiz number does not make significant difference in the result.

Acknowledgement

I would like to thank Professor Shiqian Ma for providing me the opportunity to do this research with him and all the motivation and guidance through the research process. Without his encouragement and help, I wouldn't have finished the undergraduate thesis.