Selected Topics in Constrained Optimization

A Project report

submitted by

PARTH K. THAKER

in partial fulfilment of the requirment for the award of the degree

of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

July 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **Selected Topics in Constrained Optimization**, submitted by **Parth K. Thaker**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Radha Krishna Ganti
Research Guide
Professor
Dept. of Electrical Engneering
IIT-Madras, 600 036

Place: Chennai
Date: 4th July 2016

ACKNOWLEDGEMENTS

I would like to thank my guide Prof. Radha Krishna Ganti for giving me a chance to work with him on a topic of my interest. His constant support and freedom on the research topic was something which helped me get a peak into a researcher's life. Even in his busy schedules, he used to heard my thoughts and give comments on the same. My dual degree project period wouldn't have been this much fruitful without his support.

I would like to thank all of the current and past members of NSS lab whose constant support and intellectual discussion led me to a great experience throughout out my Dual degree project time.

ABSTRACT

KEYWORDS: Constrained optimization ; Projected gradient descent; Rank Preserving Flow; Matrix Optimization.

Recently, due to the increasing size of the data matrices, it has become more and more computationally expensive to operate of such matrices. We suggest some additional thoughts as to how to reduce this computational complexity.

First, we suggest the possibility of cutting down on the number of projection operation in a normal constrained optimization setting and thereby cutting down on some of the vary taxing operations on matrices. We also indicate the benefits of limitations of this approach

Second, we more to a specific notorious constraint i.e. rank constraint. We study the minimization of function $f(X)$ over the set of all $n \times m$ matrices when we are supposed to satisfy a constraint rank=r. We propose a algorithm on the lines of Factored Gradient descent (FGD) and show its theoretical convergence and simulation results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
SVD	Singular Value decomposition
ADMM	Alternate dimension method of multipliers
PSD	Positive Semi definite
PGD	Projected gradient descent
FGD	Factored gradient descent

NOTATION

$\pi_Q(.)$	Projection Operator on set Q, m
$\min_x$	minimization operation
χ	Constrained set
rank(.)	Rank of a matrix (number of non-zero eigenvalues)
X_{ij}	$(i,j)^{th}$ element of matrix X
$X \succeq 0$	PSD constraint (all eigenvalues are non-zero)
L^T	Transpose of a matrix L
L^*	Conjugate transpose of matrix L
$\mathbb{R}^{n \times m}$	Entire matrix space with dimension $n \times m$
$\propto$	Proportional
$O(.)$	Complexity notation
Q_U	Rotation matrix
X^+	Next update step
$\nabla(.)$	First order differentiation
$\nabla^2(.)$	Second order differentiation
$\|\|X\|\|_2$	2-norm of matrix X

CHAPTER 1

Introduction

Machine learning and computational statistics problems involving large datasets have proved to be a rich source of interesting and challenging optimization problems in recent years. The challenges arising from the complexity of these problems and the special requirements for their solutions have brought a wide range of optimization algorithms into play.

We start by briefly surveying the application space, outlining several important analysis and learning tasks, and describing the contexts in which such problems are posed.

We then take one optimization approaches that is proving to be relevant which deals with Constrained Optimization. We also discuss parallel variants of some of these approaches. Various areas where we face constrained Optimization are given below

1.1 Background concepts

The basic framework of Constrained Optimization is given by the following problem formualation,

$$\min_{x \in \chi} f(x)$$

where χ is the constrained set.
Another way of representing the same is as follows,

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & g_i(x) = c_i \ \text{for} i = 1, 2, \ldots, n \\ & h_i(x) \geq d_j \ \text{for} j = 1, 2, \ldots, m \end{aligned}$$

where $g_i(x) = c_i$ for $i = 1, \ldots, n$ and $h_j(x) \geq d_j$ for $j = 1, 2, \ldots, m$ are constraints that are required to be satisfied

1.1.1 Low Rank Matrix Completion

Often also known as the Netflix problem, one of the variants of the matrix completion problem is to find the lowest rank matrix X, which matches the matrix M, which we wish to recover, for all entries in the set E of observed entries.The mathematical formulation of this problem is as follows:

$$\begin{array}{rl} \min & \text{rank}(X) \\ \text{subject to} & X_{ij} = M_{ij} \ \ \forall i, j \in E \end{array}$$

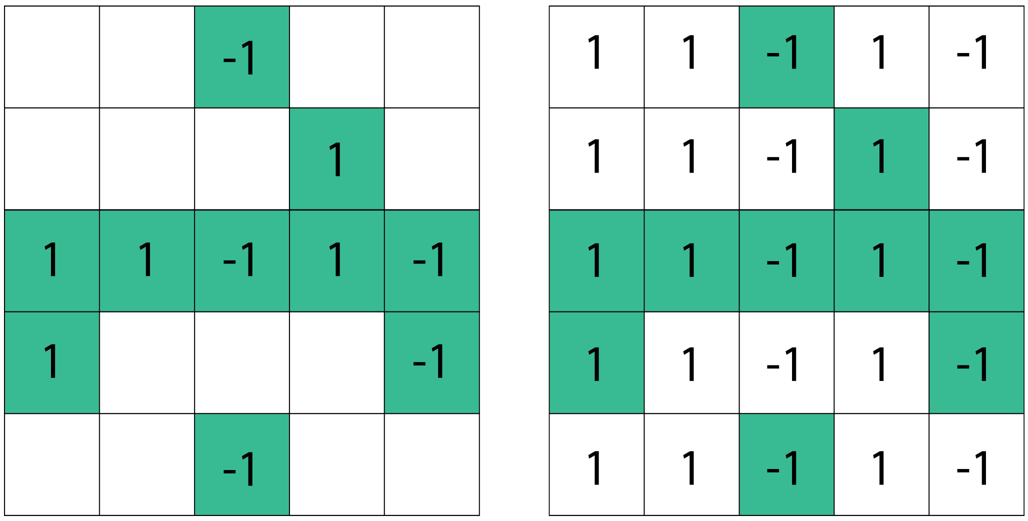


Figure 1.1: Low rank completion problem

1.1.2 Semi-definite Optimization

Sometimes in Matrix Optimization there are situation when one needs the optimal solution to have only non negative eigenvalues and thus we get the following constrained Optimization problem,

$$\begin{array}{rl} \min & f(X) \\ \text{subject to} & X \succeq 0 \end{array}$$

1.1.3 Cholesky factorization

The Cholesky decomposition of a Hermitian positive-definite matrix A is a decomposition of the form

$$A = LL^* \tag{1.1}$$

where L is a lower triangular matrix with real and positive diagonal entries, and L^* denotes the conjugate transpose of L. Every Hermitian positive-definite (PSD) matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition(Golub, 1996, p. 143), (Horn and Johnson, 1985, p.407).

If the matrix A is Hermitian and positive semi-definite, then it still has a decomposition of the form $A = LL^*$ if the diagonal entries of L are allowed to be zero(Golub, 1996, p. 147).

When A has real entries, L has real entries as well and the factorization may be written $A = LL^T$(Horn and Johnson, 1985, p. 407) The Cholesky decomposition is unique when A is positive definite; there is only one lower triangular matrix L with strictly positive diagonal entries such that $A = LL^*$. However, the decomposition need not be unique when A is positive semi-definite.

The converse holds trivially: if A can be written as LL^* for some invertible L, lower triangular or otherwise, then A is Hermitian and positive definite.

CHAPTER 2

Skipping in Projected Gradient Descent

In the case of constrained minimization problem ($\min f(x)$ where $x \in Q$), we need to consider the projection to the convex set Q. So let us understand the 'projection' first

2.1 Projection

Projection of a point x_0 to the set Q is defined as,

$$\pi_Q(x_0) = \text{argmin}\,||x - x_0||, x \in Q \tag{2.1}$$

i.e. projection of a point x_0 to the closed and convex set Q is the closed point, in the sense of the Euclidean norm, on the set Q.
Some of the interesting properties of projection operator are as follows:

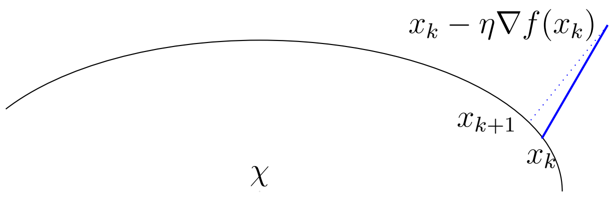
- Let Q be a closed and convex set. Then $\exists$ a unique projection $\pi_Q(x_0)$
- For any $x \in Q$, $||x - \pi_Q(x_0)||^2 + ||x_0 - \pi_Q(x_0)||^2 \leq ||x - x_0||^2$
- From the above point it follows that $||x - \pi_Q(x_0)|| \leq ||x - x_0||$, thus projection is a non-expansive operator.

2.2 Projected Gradient Descent

A big part of convex optimization is based on working out clever fixed-point equations verified by the minimizers and to then apply a fixed-point iteration. In the case of constrained optimization case it leads to the following algorithm.
The Projected Gradient Descent(PGD) generates iterative sequence $\{x_k\}$ by the following equation,

$$x_{k+1} = \pi_Q(x_k - \eta_k \nabla f(x_k)) \tag{2.2}$$

where π_Q is the projection operator on the constraint set Q and η_k is the step size.



Roughly speaking, assuming that the η_k are picked sensibly and under some regularity conditions on the problem, the method enjoys a rate $(f(x_k) - f(x^*)) = O(\sqrt{k})$.
This convergence with the square root of the number of iterations is actually a standard result for gradient-descent type methods.

2.2.1 Origin of problem

With the over increasing databases, the optimization operations need to become more and more efficient so as to counter the ever increasing size of the datasets.
If we look at the following constrained optimization problem

$$\begin{aligned} \min_{X\in\mathbb{R}^{n\times m}} \quad & f(X) \\ \text{s.t} \quad & \text{rank}(X) \leq k \end{aligned}$$

where $k \leq$min$\{m, n\}$

The most primitive approach for solving question of this form would be to implement a projection gradient descent algorithm.
For each step of the projection required in projected gradient descent, we need to do Singular Value Decomposition of matrix at each step in order to maintain the rank constraint.

Singular Value decomposition

the Singular Value Decomposition (SVD) is a factorization of a real or complex matrix. Formally, singular value decomposition of a $m \times n$ real or complex matrix M is a factorization of the form $U\Sigma V^*$, where U is a $m \times m$ real or complex unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is a $n \times n$ real or complex unitary matrix. The diagonal entries σ_i of Σ are known as singular values of M.
Optimal cost of calculating a SVD operation is $O(min\{m^2n, n^2m\})$. So as the size of the matrix increases, projected gradient descent takes more and more time to compute each step.

This is just one of the many examples as to where projected gradient descent becomes inadequate as the data matrix starts scaling.

2.2.2 Skipping between projections

Suppose the following happens:

$$\begin{aligned} x_k &\rightarrow x_k - \eta_k \nabla f(x_k) && \text{Time taken = 0.001s} \\ x_k - \eta_k \nabla f(x_k) &\rightarrow x_{k+1} = \pi_Q(x_k - \eta_k \nabla f(x_k)) && \text{Time taken = 10s} \end{aligned}$$

So, we what if we could skip a projection step. That is, what if we do projection every alternate step. The update equation will look something like this,

$$\begin{aligned} y_k &= x_k - \eta_k \nabla f(x_k) \\ x_{k+1} &= \pi_Q(y_k - \eta_k \nabla f(y_k)) \end{aligned}$$

Now we can continue this and have a projection step every three step. The update equation of the same will look something like this,

$$y_k = x_k - \eta_k \nabla f(x_k)$$
$$z_k = y_k - \eta_k \nabla f(y_k)$$
$$x_{k+1} = z_k - \eta_k \nabla f(z_k)$$

So we term the number of skips of the as Frequency.

2.2.3 Simulation 1

Objective Function

The function we used for simulation is a general elliptical function given by,

$$\min_{(x,y)\in\chi} \ 3x^2 + 5xy + 4y^2 - 2x - 4y + 27 \tag{2.3}$$

where χ was tried for two different sets $\chi : y = 0$ and $\chi : x^2 + y^2 = 1$.

Log Error Plot

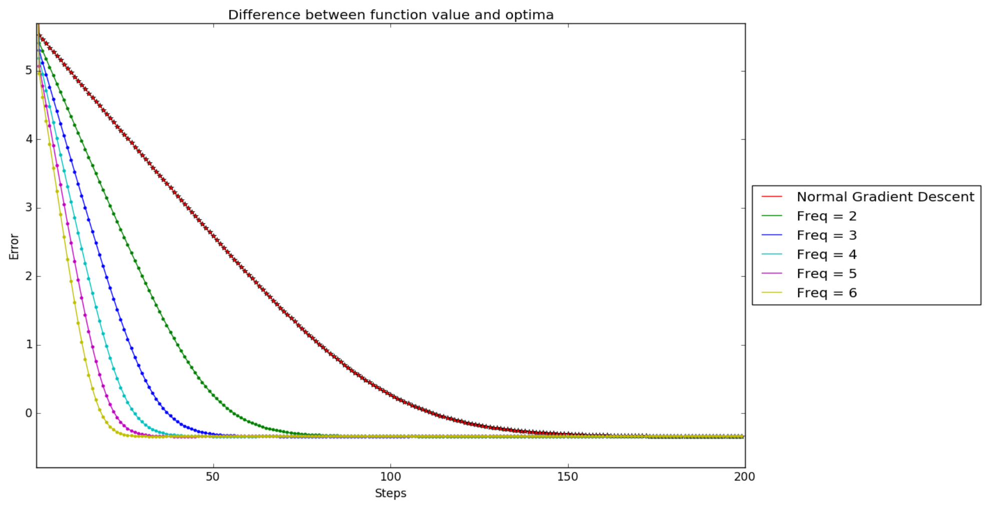


Figure 2.1: Log Error plot

In the above figure, the y axis represent the log of error where the error is defined as follows:

$$\text{Error}(x) = f(x) - f(x^*) \tag{2.4}$$

where $f(x^*)$ is the unconstrained minimum.

A closer look towards the tail of the above plot, we get the following plot, Thus there

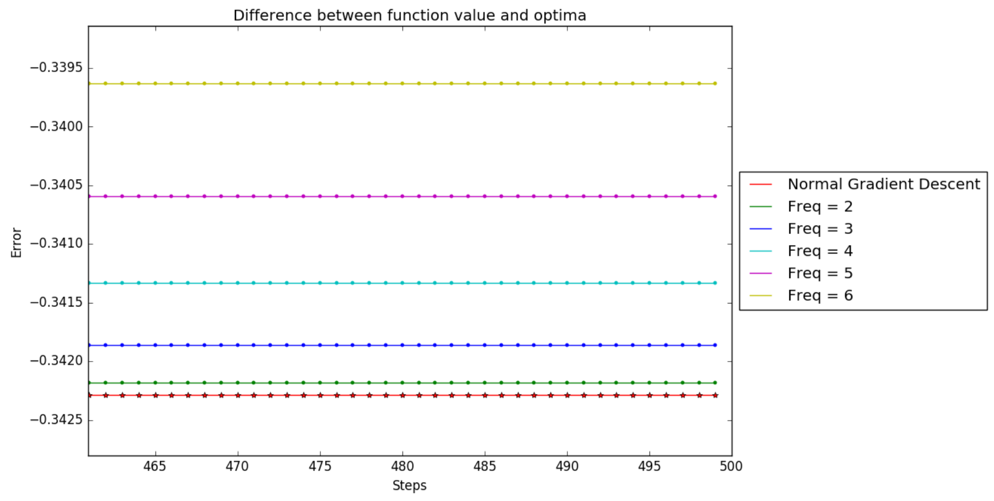


Figure 2.2: Log Error plot–Tail

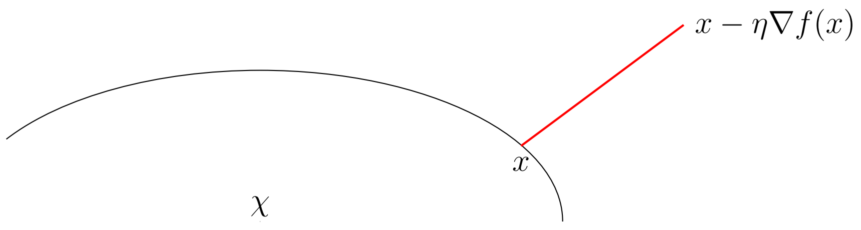
seems to be a trade-off between the number of skipping and the steady state error. The more the frequency of skipping, the more is the steady state error. Thus we can say that

$$\text{Steady state error} \propto \text{Frequency of skipping} \tag{2.5}$$

Inference

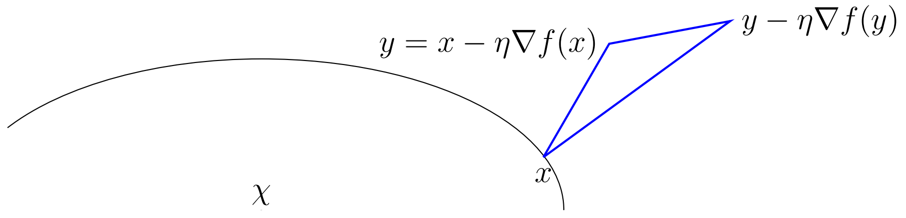
Since the unconstrained minimum x^* is out of the constrained set in this situation, the projected gradient descent method cannot reach and settle at the fixed point of the equation,

$$x = \pi_Q(x - \eta \nabla f(x))$$

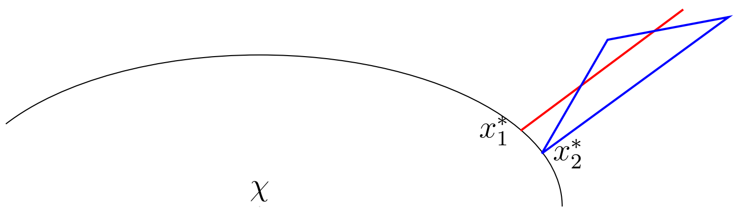


For alternate skipping method, or projected gradient descent with skip frequency =2, the fixed point equations will look like,

$$y = x - \eta \nabla f(x)$$
$$x = \pi_Q(y - \eta \nabla f(y))$$



We can derive similar fixed point equations for other skipping frequencies as well. Now if we overlap both above diagrams we get something as below,



As displayed in the above figure, the fixed point of normal projected gradient descent and projected gradient descent with 1 skip differs. This is the main reason for the steady state error difference seen in the Error plot.

2.2.4 Possible Workaround

Consider the comparison between the steady state fixed point equation of normal projected gradient descent and the projected gradient descent with skip frequency as 1,

Normal Projected gradient descent

$$x^* = \text{Pr}(x^* - \eta \nabla f(x^*)) \tag{2.6}$$

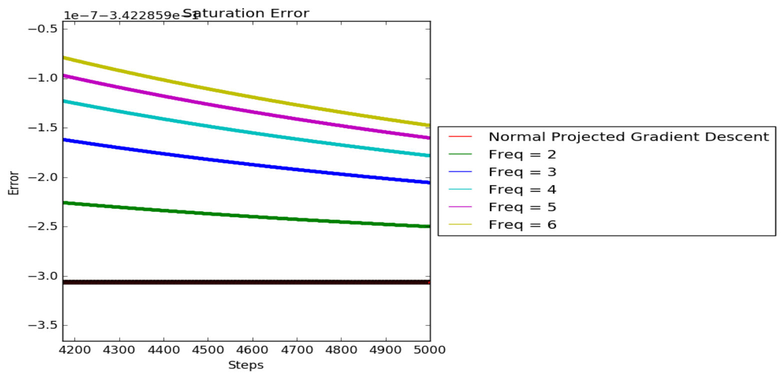
Projected Gradient descent with one Skip

$$x^* = \text{Pr}(x^* - \eta\nabla f(x^*) - \underbrace{\eta\nabla f(x^* - \eta\nabla f(x^*)))}_{\text{Error Cause}} \tag{2.7}$$

The steady state error difference is due to the excess $\eta\nabla f(x^* - \eta\nabla f(x^*))$ term in the fixed point equation of projected gradient descent with freq 1.

Reduction of Error term

We can reduce this error by making $\eta \to 0$ as $t \to 0$ still ensuring that the sequence of η's do not violate the gradient decent constraints.
Simulating the above condition we get the following tail plot,
which shows a $O(\frac{1}{t})$ convergence as $\eta_t = O(\frac{1}{t})$



2.3 Limitations

Even though this practice of skipping might become a bit practically relevant, in theory we do not get a provable time reduction as,

$$O\left(\frac{n}{c}\right) = O(n) \tag{2.8}$$

2.4 Possible Future Extensions

- **Error Bounds** :
  A closed form expression can be found for the trade-off between the number of

skipping and the resulting error.

- **Approximate Solution**:
  If we keep increasing the skipping frequency to ∞, which is equivalent to saying we get the solution of this thought algorithm as Pr($x^{\#}$). $x^{\#}$ denotes the unconstrained minimum. In other words, $x^{\#}$ is the minimizer of

  $$\min_{x} f(x) \tag{2.9}$$

  The question to be addressed here is, how good is the approximation Pr($x^{\#}$) as a solution of the constrained case. Are there some conditions which can quantify this thought?

- **Initialization**:
  As it is a known fact that Newton descent method has two phase convergence : Damped phase and quadratic convergence phase. A clever initialization can lead to the quadratic convergence phase. For further discussion on Newton's method please refer to Boyd and Vandenberghe (2004).

  From looking at a similar thought process, can initialization with Pr($x^{\#}$) lead to better convergence rate?

CHAPTER 3

Rank Preserving Flow

3.1 Symmetric matrices

Let S(N) denote the set of all $N \times N$ symmetric matrices. For integers $1 \leq n \leq N$ we have,

$$S(n, N) = \{X \in \mathbb{R}^{N \times N} | X^T = X,\ \text{rank } X = n\}, \tag{3.1}$$

denotes the set of all $N \times N$ symmetric matrices of rank n.

For proof of further proposition please refer to Helmke *et al.* (1993)

Proposition 1: $S(n, N)$ is a smooth manifold of dimension $\frac{1}{2}n(2N - n + 1)$ and has $n + 1$ connected components

$$S(p, q, N) = \{X \in S(n, N) | \text{signature } X = p - q\} \tag{3.2}$$

where $p, q \geq 0, p + q = n$.

3.1.1 Tangent Space

Consider the mapping $f : M(n) \to S(n)$ where $A \mapsto AA^T$. The tangent space at $x \in f^{-1}(y)$ is equal to ker(Df_x). So currently for the case of function f we have Df as

$$Df_A(B) = BA^T + AB^T \tag{3.3}$$

For a detailed explanation on tangent spaces do take a look at Absil *et al.* (2009)

The tangent space of S(n,N) at an element X is

$$T_X S(n, N) = \{\Delta X + X\Delta^T | \Delta \in \mathbb{R}^{N \times N}\} \tag{3.4}$$

3.1.2 Rank preserving flow

A differential equation $\dot{X} = F(X)$ evolving on the matrix space S(N) is said to eb rank preserving if rank of X(t) is constant as a function of t.The following lemma gives a simple characterization of rank preserving vector fields on S(N).

Lemma 1: Let $I \subset \mathbb{R}$ be an interval and let $A(t) \in \mathbb{R}^{N\times N}, t \in I$, be a continuous family of matrices, Then

$$\dot{X}(t) = A(t)X(t) + X(t)A(t)^T, \;\; X(0) \in S(N) \tag{3.5}$$

is a rank conserving flow on $S(N)$. Conversely, any rank preserving vector field on $S(N)$ is of this form.
For proof do refer to Helmke *et al.* (1993).

3.2 Ricatti Flow

3.2.1 Riccati – First form

We get good comparisons from Helmke *et al.* (1993) on the properties of the update equations. By Lemma 1, every rank preserving flow on $S(N)$ has the form,

$$\dot{X} = F(X)X + XF(X)^T \tag{3.6}$$

Theorem :

1. The Riccati Differential equation defined as,

$$\dot{X} = (A - X)X + X(A - X) \tag{3.7}$$

   where $A \in \mathbb{R}^{N\times N}$ is symmetric is rank preserving flow on $S(n, N)$

2. Assume A is invertible. Then solutions X(t) of (3.7) are given by

$$X(t) = e^{tA}X_0[I_N + A^{-1}(e^{2At} - I_N)X_0]^{-1}e^{tA} \tag{3.8}$$

3. For any positive semi-definite matrix initial condition $X(0) = X(0)^T \geq 0$, the solution of (3.7) exists for all $t \geq 0$ and is positive semi-definite.
4. Every positive semi-definite solution $X(t) \in S^+(n, N) = S(n, o, N)$ of (3.7) converges to the connected component of the set of equilibrium points, characterized by $(A - X_\infty)X_\infty = 0$. Also X_∞ is positive semi-definite and has rank $\leq n$. **If** A has distinct eigenvalues then every positive semi-definite solution $X(t)$ converges to an equilibrium point.

Now this gives a good intuition as to why the update equation in (**??**) is a rank preserving update.

If we see closely, the for of update equation (**??**) can be rewritten as,

$$X^+ = X - \eta(\nabla f(X)X\Lambda + \Lambda^T X \nabla f(X)), \tag{3.9}$$

where $\Lambda = I - \frac{\eta}{2} Q_U Q_U^T \nabla f(X)$.

This looks similar to the rank preserving flow, assuring the readers that the update equation in the X domain indeed retains the rank even after updating.

Riccati – Second form

Now due to some more stability properties we look at a different gradient flow like differential equation given as follows,

$$\dot{X} = (A - XX^T)X,\ X(0) \in \mathbb{R}^{N\times n}, \tag{3.10}$$

for $A = A^T \in \mathbb{R}^{N\times N}$ real symmetric.

The main motivation of studying flow (3.10) on $\mathbb{R}^{N\times n}$ is that it induces flow (3.7) on $S^+(n, N)$.

This can eb seen as, let $H = XX^T$, then

$$\dot{H} = \dot{X}X^T + X\dot{X}^T = (A - H)H + H(A - H) \tag{3.11}$$

Convergence properties of flow (3.10) are given in the next result

Theorem :

1. The differential equation in (3.10) is a rank preserving flow on $\mathbb{R}^{N\times n}$.
2. The solution for (3.10) exits for all $t \geq 0$ and converges for $t \to \infty$ to a connected component of the set of equilibrium points X_∞ of rank $\leq n$. Equilibrium components are characterized by $(A - X_\infty X_\infty^T)X_\infty = 0$ or equivalently by the condition that the columns of X_∞^T generate an A-invariant subspace of $\mathbb{R}^N$. If $n = 1$ and A has distinct eigenvalues, then every solution of $X(t)$ converges to the equilibrium point.
3. The function $V : \mathbb{R}^{N\times n} \to \mathbb{R}$, $V(X) := ||A - XX^T||^2$ is a Lyapunov function of (3.10)

This gives the intuition for the update equation (4.4). The update of the algorithm goes as,

$$U^+ = U - \eta \nabla f(UU^T)U \tag{3.12}$$

3.3 Possible Work Direction

The problem which seemed quite intriguing is explained as following, Consider the equation for Rank preserving flow as below

$$\dot{X}(t) = A(t)X(t) + X(t)A(t)^T, \;\; X(0) \in S(N) \tag{3.13}$$

For the above to be an update step of a Gradient descent flow, what should be the conditions on $A(t)$.

CHAPTER 4

Factored Gradient Descent

4.1 Problem Structure

Let $f : \mathbb{R}^{n\times n} \rightarrow \mathbb{R}$ be a convex function. Let us take the following optimization problem for further analysis,

$$\begin{array}{cc} \min\limits_{X\in\mathbb{R}^{n\times n}} & f(X) \\ \text{subject to} & X \succeq 0 \\ & \text{rank}(X) = k \end{array}$$

A standard method of approaching is to start as follows

$$\begin{aligned} Y_k &= X_k - \eta\nabla f(X_k) \\ X_{k+1} &= \text{Pr}(Y_k) \end{aligned}$$

where the operator Pr$(.)$ projects the operand to the space of PSD rank r matrices.

4.1.1 Major Problem

The projection step for the required algorithm requires computation of a singular value decomposition (SVD) at each step. Evaluating a SVD for matrix $X \in \mathbb{R}^{m\times n}$ requires $O(\min\{nm^2, n^2m\})$. Thus this algorithm quite time consuming when used with the current time data matrices with large dimensions.

4.2 Alternate approach

Factored gradient descent (FGD) provides an alternative way to the above problem. The PSD constraint matrices enables us to replace X by its cholesky decomposition as follows,

$$X = UU^T \tag{4.1}$$

where $X \in \mathbb{R}^{n \times n}, U \in \mathbb{R}^{n \times r}$ and $r << n$. Here we are using U^T as we are only considering real variables.

Thus the optimization problem becomes,

$$\begin{aligned} &\min_{U \in \mathbb{R}^{n \times r}} g(U) \\ &\text{where } r \leq n \end{aligned} \tag{4.2}$$

where $g(U) = f(UU^T)$.

4.2.1 Benefits

Instead of running the update steps in 'X' domain, we try to update the algorithm in the 'U' domain.

We can list the benefits of the above approach as follows,

- **In-built Positive semi-definiteness**:
  The update step in 'U' domain gets mapped back to the 'X' domain by the mapping $X = UU^T$ which ensures matrix X is a PSD matrix.
- **Rank preserving update**:
  The update step occurs is the 'U' domain which is a rank-r space. Thus the rank constraint of rank(X)$\leq r$ is always satisfied as $X = UU^T$ cannot have a rank more than r.

4.3 Update Algorithm

We try to define the update algorithm for the optimization problem

$$\begin{aligned} &\min_{U \in \mathbb{R}^{n \times r}} g(U) \\ &\text{where } r \leq n \end{aligned} \tag{4.3}$$

The standard update of the algorithm goes as,

$$U^+ = U - \eta \nabla g(U) \tag{4.4}$$

We can further evaluate $\nabla g(U)$ as,

$$\nabla g(U) = \nabla(f(UU^T)) = 2\nabla f(UU^T)U \tag{4.5}$$

Thus, we get the update equation as,

$$U^+ = U - \eta \nabla f(UU^T)U \tag{4.6}$$

which translates in the X domain as,

$$\begin{aligned} X^+ &= U^+(U^+)^T \\ &= (U - \eta\nabla f(UU^T)U)(U - \eta\nabla f(UU^T)U)^T \\ &= UU^T - \eta\nabla f(UU^T)UU^T - UU^T\eta\nabla f(UU^T)^T - \eta^2\nabla f(UU^T)UU^T\nabla f(UU^T) \\ &= X - \eta\nabla f(X)X - \eta X\nabla f(X) + \eta^2\nabla f(X)X\nabla f(X) \end{aligned}$$

Further studies on the convergence rate for various scenarios for the above update equation has been done in the paper Bhojanapalli *et al.* (2015).

4.4 Relation with Riccati Flow

Recalling the Ricatti flow of the first form (3.7), we have,

$$\dot{X} = F(X)X + XF(X)^T \tag{4.7}$$

Comparing it to our equation of update in 'X' domain above we get,

$$X^+ = X - \eta\nabla f(X)X - \eta X\nabla f(X) + \eta^2\nabla f(X)X\nabla f(X) \tag{4.8}$$

which can also be written as,

$$X^+ = X - \eta\nabla f(X)X\Lambda - \eta\Lambda X\nabla f(X) \tag{4.9}$$

where $\Lambda = I - \frac{\eta}{2}Q_U Q_U^T\nabla f(X)$.

Intuitively, as it fits the Riccati form, the above equation (4.9) is a rank preserving update equation.

CHAPTER 5

Generalised Factored Gradient Descent

5.1 Problem Structure

Consider a convex function $f : \mathbb{R}^{n\times m} \to \mathbb{R}$ which is to be optimized with respect to the variable $X \in \mathbb{R}^{n\times m}$

$$\begin{aligned} \min_{X\in\mathbb{R}^{n\times m}} \quad & f(X) \\ & \text{rank}(X) = r \end{aligned}$$

Taking inspiration from Bhojanapalli *et al.* (2015). We try to split the matrix into two factors $W \in \mathbb{R}^{n\times r}$ and $H \in \mathbb{R}^{m\times r}$ such that $X = WH$. After parametrizing the problem as $X = WH$, we try to convert the optimization problems in terms of W and H separately.
We have to solve the below problems,

W domain

When updating for **W** we have to solve,

$$\min_{W\in\mathbb{R}^{n\times r}} \quad g_H(W)$$

where $g_H(W) = f(WH)$ for a particular fixed H.

H domain

When updating for **H** we have to solve,

$$\min_{H\in\mathbb{R}^{r\times m}} \quad p_W(H)$$

where $p_W(H) = f(WH)$ for a certain fixed W.

5.2 Major Improvements

- Instead of the rigidity of the square matrix with dimension $n \times n$ in citesujay, we have relaxed to a general rectangular matrix of $n \times m$.
- Bhojanapalli *et al.* (2015) analysis and algorithm can be considered the special case of the current analysis.
- We cannot currently prove the PSD matrix constraint satisfaction for our case.
- We can guarantee the rank preserving flow for the algorithm. If one start with rank r matrix, it will stay rank r.
- Bhojanapalli *et al.* (2015) had to make the optimization scheme as non convex when going from one domain to another. But in out case we are maintaining the convexity at each step of our scheme.

5.3 Theoretical Analysis

W update

Here we try to optimize over the problem,

$$\min_{W \in \mathbb{R}^{n \times r}} \quad g_H(W) \tag{5.1}$$

where $g_H(W) = f(WH) = f(X)$. Thus our update equation should be of the form,

$$W^+ = \arg \min_{W \in \mathbb{R}^{n \times r}} g_H(W) \tag{5.2}$$

Thus after every update of such form, since the function $g_H(W)$ is convex in W, we have that,

$$\begin{aligned} g_H(W^+) &\leq g_H(W) \\ \Rightarrow f(W^+H) &\leq f(WH) \qquad \forall H \in \mathbb{R}^{r\times m} \end{aligned} \tag{5.3}$$

H update

Here we try to optimize over the problem,

$$\min_{H\in\mathbb{R}^{r\times m}} \quad p_W(H) \tag{5.4}$$

where $p_W(H) = f(WH) = f(X)$. Thus our update equation should be of the form,

$$H^+ = \arg\min_{H\in\mathbb{R}^{r\times m}} p_W(H) \tag{5.5}$$

Thus after every update of such form, since the function $p_W(H)$ is convex in H, we have that,

$$\begin{aligned} p_W(H^+) &\leq p_W(H) \\ \Rightarrow f(WH^+) &\leq f(WH) \qquad \forall W \in \mathbb{R}^{n\times r} \end{aligned} \tag{5.6}$$

X update

Recombining the above two different update steps to return in the X domain, we get the update step as

$$\begin{aligned} W_{n+1} &= \arg\min_{W \in \mathbb{R}^{n\times r}} g_{H_n}(W) \\ H_{n+1} &= \arg\min_{H \in \mathbb{R}^{r\times m}} p_{W_{n+1}}(H) \\ X_{n+1} &= W_{n+1}H_{n+1} \end{aligned}$$

After the first update of W_{n+1} we can say that,

$$f(W_{n+1}H_n) \leq f(W_nH_n)$$

After the update of H_{n+1} we have the update step as,

$$f(W_{n+1}H_{n+1}) \leq f(W_{n+1}H_n) \leq f(W_nH_n) \tag{5.7}$$

Thus,

$$f(X_{n+1}) \leq f(X_n) \tag{5.8}$$

Since the function is convex we cna say that,

$$\begin{gathered} f(X_0) \geq f(X_1) \geq \cdots \geq f(X_r^*) \\ \lim_{n\to\infty} f(X_n) \to f(X_r^*) \end{gathered}$$

where X_r^* is the optimal r-rank matrix. The convergence of the above statement is because of a lower bounded monotonically decreasing series.

5.4 Descent Algorithm

5.4.1 W update

Here we try to optimize over the problem,

$$\min_{W \in \mathbb{R}^{n \times r}} \quad g_H(W) \tag{5.9}$$

where $g_H(W) = f(WH) = f(X)$. Thus our update equation should be of the form,

$$W^+ = W - \eta_1 \nabla g_H(W) \tag{5.10}$$

where the gradient is w.r.t. W and the step size $\eta_1 < \frac{1}{||\nabla^2 g_H(W)||_2}$. Thue we try to evaluate $\nabla g_H(.)$ and $\nabla^2 g_H(.)$. We get the following,

$$\nabla g_H(W) = \nabla f(WH) H^T \tag{5.11}$$

For the special case of W and H being vectors and denoted by w and h respectively, we can say the following,

$$\nabla^2 g_H(w) = \left[\sum_j \left[\sum_l \frac{\partial^2 f(wh)}{\partial w_i h_j \partial w_k h_l} h_l \right] h_j \right] \tag{5.12}$$

which gives the indication that for L Lipschitz smooth function, we get

$$\eta_1 < \frac{1}{\nabla^2 g_H(W)} \propto \frac{1}{L||H^T H||} \text{(recheck)} \tag{5.13}$$

Thus our update equation becomes,

$$W^+ = W - \eta_1 \nabla f(WH) H^T \tag{5.14}$$

Convergence Results

Theorem 1 *The convergence rate of the update algorithm* (5.14) *in the W domain when H is kept constant is given by,*

$$g(W_{n+1}) - g(W^*) \leq \frac{||W_0 - W^*||^2}{n\beta}$$

where $\beta = \left(\frac{L||H^T H||\eta_1^2}{2} - \eta_1\right)$.

For proof, please refer to Appendix A

5.4.2 H update

Here we try to optimize over the problem,

$$\min_{H \in \mathbb{R}^{r \times m}} \quad p(H) \tag{5.15}$$

where $p(H) = f(WH) = f(X)$. Thus our update equation should be of the form,

$$H^+ = H - \eta_2 \nabla p(H) \tag{5.16}$$

where the gradient is w.r.t. W and the step size $\eta_2 < \frac{1}{||\nabla^2 p(H)||_2}$. Thus, we try to evaluate $\nabla p(.)$ and $\nabla^2 p(.)$. We get the following,

$$\nabla p(W) = W^T \nabla f(WH) \tag{5.17}$$

For the special case of W and H being vectors and denoted by w and h respectively, we can say the following,

$$\nabla^2 p(w) = \left[\sum_i \left[\sum_k \frac{\partial^2 f(wh)}{\partial w_i h_j \partial w_k h_l} w_k\right] w_i\right] \tag{5.18}$$

which gives the indication that for L Lipschitz smooth function, we get

$$\eta_2 < \frac{1}{\nabla^2 p(H)} \propto \frac{1}{L||W_n W_n^T||}\text{(recheck)} \tag{5.19}$$

Thus our update equation becomes,

$$H^+ = H - \eta_2 W^T \nabla f(WH) \tag{5.20}$$

Convergence Results

Theorem 2 *The convergence rate of the update algorithm* (5.20) *in the W domain when H is kept constant is given by,*

$$p_W(H_{n+1}) - p_W(H^*) \leq \frac{||H_0 - H^*||^2}{n\alpha}$$

where $\alpha = \left(\frac{L||W^T W||\eta_1^2}{2} - \eta_1\right)$.

5.4.3 X update

Taking both the update equation (5.14) and (5.20) into the X domain we get,

$$\begin{aligned} X_{n+1} &= (W_{n+1})(H_{n+1}) \qquad (5.21) \\ X_{n+1} &= X_n - \eta_2 W_n W_n^T \nabla f(X_n) - \eta_1 \nabla f(X_n) H_n^T H_n + \eta_2 \eta_1 \nabla f(X_n) X_n^T \nabla f(X_n) \end{aligned}$$

Convergence result

Theorem 3 *The convergence rate of the update algorithm in the X domain* (5.21) *is given by,*

$$X_{n+1} - X^* \leq \frac{1}{\sum_{i=0}^{n} \frac{\alpha_{n-i}}{\Pi_{j=0}^{i} c_{n-j}}}$$

where $\alpha_n = \left(\frac{L||H_n H_n^T||\eta_2^2}{2} - \eta_2\right)$ *and* $c_n = (1 + \alpha_n \Delta_{H_n})$.

For proof please refer to Appendix C

5.4.4 Simulation 2

Objective function

$$\min_{X \in \mathbb{R}^{n\times m}} ||X - av_1v_2^T + bv_3v_4^T||_2$$

where $a, b \in \mathbb{R}$ and v_1, v_2, v_3, v_4 chosen to fit the dimensionality constraint appropriately.

Convergence plot Rank plot

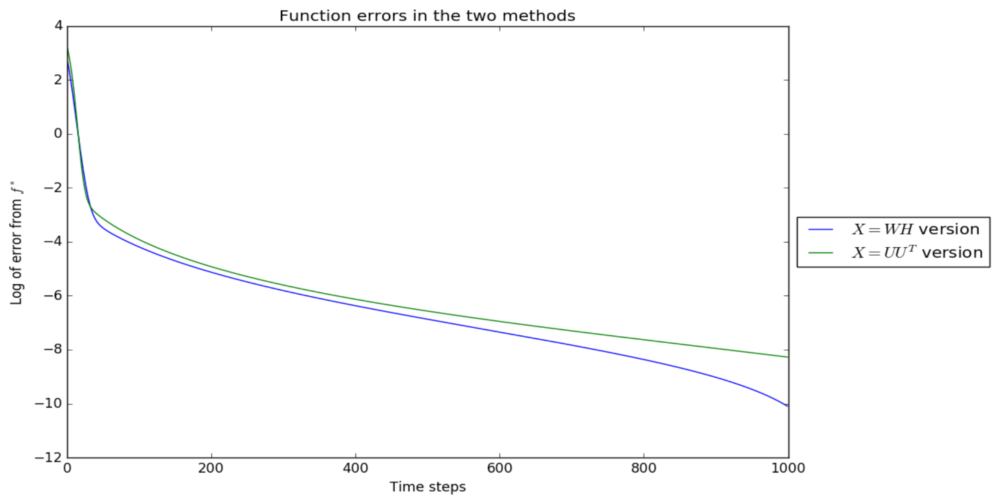


Figure 5.1: Plot of function value with the proposed algorithm

Norm plot

5.5 Recent advancement

Recently, a paper Tu *et al.* (2015) proved the standard convergence results for the algorithm.

$$\min_{U,V} f(UV^T) + h(U^TU - V^TV)$$

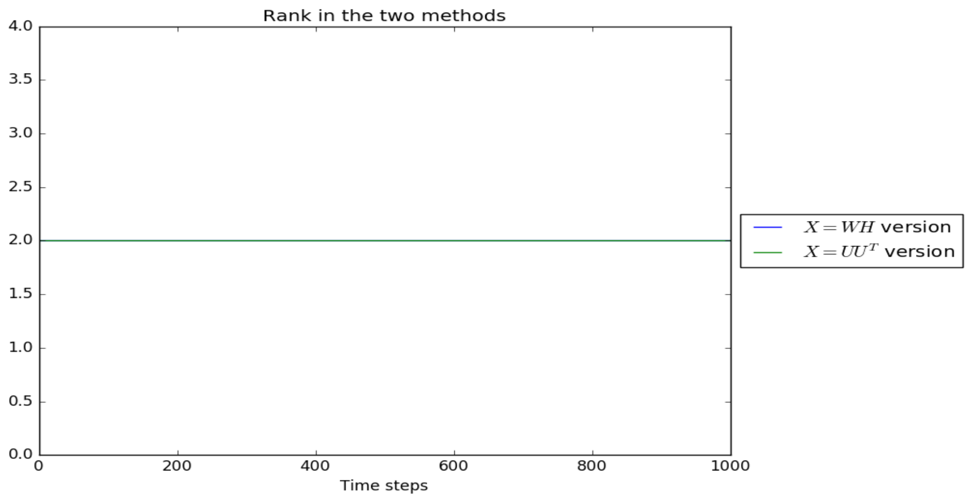


Figure 5.2: Plot of the rank of matrix throughout the simulation

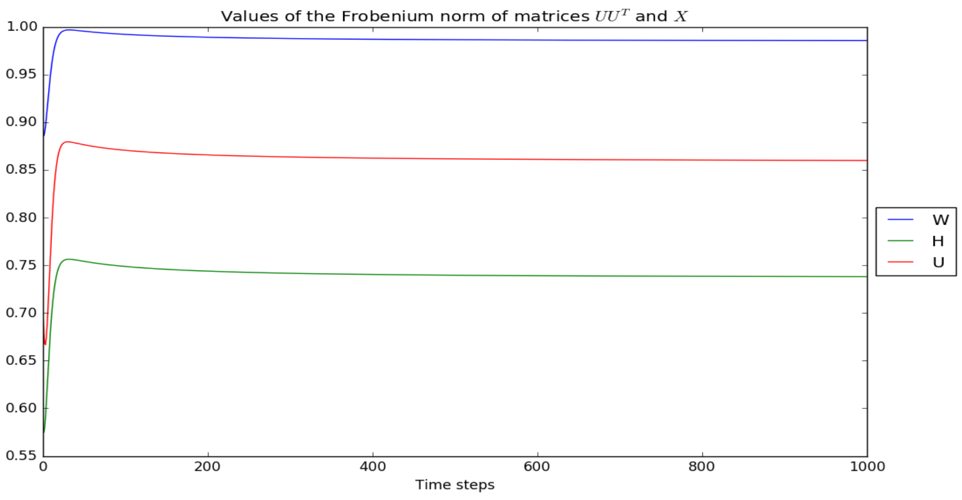


Figure 5.3: Plot of $||W||_2$ and $||H||_2$ throughout the simulation

where h is strongly smoothly convex with global minima at 0. They proved local linear convergence to neighbourhood of X_r^*.

$$d((U^+, V^+), X^*) \leq \gamma d((U, V), X^*) + \eta M||X^* - X_r^*||^2$$

where $\gamma < 1$.

5.6 Possible Future Extensions

- **ADMM-type approach for rank constrained optimization**:
  The current approach only focuses on the one constraint i.e. maintaining rank-r.

We can maybe have other linear constraints and develop a ADMM like optimization scheme which can converge provably.

- **Accelerated gradient approaches:**
  The basic case of convergence has been proved not long ago, so one can work on Accelerated version of the same in order to prove stronger convergence rate.
- **Generalized Descent proof**:
  Going to an even broader sense, a generalised result showing that an established descent algorithm for single variable can be applied in this scenario i.e. can be applied to either W domain update or H domain update or both the variables having the same descent algorithm or different descent algorithm and still can be proved to converge.

CHAPTER 6

Python Code for simulation 1

Listing 6.1: Rank maintaing simulation

```
import numpy as np
from matplotlib.pyplot import *
from numpy.linalg import inv
n = 5
n1 = 5
n2 = 5
k = 2
m = 4
s = (n,k)
a = abs(np.random.rand())+2.3
b = abs(np.random.rand())*0.0001
v1 = np.random.rand(n1, 1)
v2 = np.random.rand(n2, 1)
v3 = np.random.rand(n2, 1)
v4 = np.random.rand(n1, 1)
v1[-1][0] = 1.
v3[-1][0] = -np.dot(v2[:-1].T, v3[:-1])
v1 = v1/np.linalg.norm(v1, 'fro')
v1 = v3/np.linalg.norm(v3, 'fro')
v1 = v4/np.linalg.norm(v4, 'fro')
v2 = v2/np.linalg.norm(v2, 'fro')
W = np.random.rand(n1, k)
H = np.random.rand(k, n2)
U = np.random.rand(n, k)

def funct(W,H):
     return np.linalg.norm(np.dot(W,H) -a*a*np.dot(v1,v1.T) +\
      b*b*np.dot(v2, v2.T), 'fro')**2

def del_funct(W,H):
     return 2*(np.dot(W,H) -a*a*np.dot(v1,v1.T) +\
      b*b*np.dot(v2, v2.T) )



def eta():
     return 1./16/(a*a+2*b*b)

def norm_matrix(X):
     return np.linalg.norm(X, 'fro')

```

```
i=1
Wold = W
Hold = H
Uold = U
W_Values = []
W_ver = []
H_Values = []
H_ver = []
U_Values = []
U_ver = []
X_ver = []
X_Values = []

while(i < 1000):
     i=i+1
     U = Uold - eta()*np.dot(del_funct(Uold,Uold.T), Uold)
     W = Wold - eta()*np.dot(del_funct(Wold,Hold), Hold.T)
     H = Hold - eta()*np.dot(Wold.T, del_funct(Wold,Hold))
     Uold = U
     Wold = W
     Hold = H
Wstar = Wold
Hstar = Hold
Ustar = Uold
fstar = funct(Ustar,Ustar.T)


W = np.random.rand(n1, k)
H = np.random.rand(k, n2)
U = np.random.rand(n, k)

Wold = W
Hold = H
Uold = U

W_Values.append(funct(W,H) )
W_ver.append(norm_matrix(W))
U_Values.append(funct(U,U.T) )
U_ver.append(norm_matrix(U))
H_Values.append(funct(W,H) )
H_ver.append(norm_matrix(H))
Xold =np.dot(W,H)
X_Values.append(funct(W,H) )
X_ver.append(norm_matrix(np.dot(W,H)))
W_rank = [np.linalg.matrix_rank(Wold)]
H_rank =[np.linalg.matrix_rank(Hold)]
X_rank = [np.linalg.matrix_rank(Xold)]
U_rank = [np.linalg.matrix_rank(Uold)]
i=1

while(i < 1000):
        i=i+1
        U = Uold - eta()*np.dot(del_funct(Uold,Uold.T), Uold)
        W = Wold - eta()*np.dot(del_funct(Wold,Hold), Hold.T)
        H = Hold - eta()*np.dot(Wold.T, del_funct(Wold,Hold))
        W_Values.append((funct(W,H) - fstar))
        W_ver.append(norm_matrix(W))
        U_Values.append((funct(U,U.T) - fstar))
        U_ver.append(norm_matrix(U))
        H_Values.append((funct(W,H) - fstar))
        H_ver.append(norm_matrix(H))
        W_rank.append(np.linalg.matrix_rank(W))
        H_rank.append(np.linalg.matrix_rank(H))
        U_rank.append(np.linalg.matrix_rank(U))
        Uold = U
        Wold = W
        Hold = H

figure(1)
plot(range(i), np.log(W_ver), label = "W")
plot(range(i), np.log(H_ver), label = "H")
plot(range(i), np.log(U_ver), label = "U")
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))
title(r'Values of the Frobenium norm of matrices $UU^T$ and $X$')
xlabel("Time steps")
figure(2)
plot(range(i), np.log(H_Values), label = r"$X = WH$ version")
plot(range(i), np.log(U_Values), label = r"$X = UU^T$ version")
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))
title("Function errors in the two methods")
xlabel("Time steps")
ylabel("Log of error from $f^*$")
figure(3)
plot(range(i), H_rank, label = r"$X = WH$ version")
plot(range(i), U_rank, label = r"$X = UU^T$ version")
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))
title("Rank in the two methods")
xlabel("Time steps")
ylim([0,4])
show()
```

CHAPTER 7

Python Code for simulation 2

Listing 7.1: Skippings in Proceted Gradient descent

```
from matplotlib.pyplot import *
from numpy import *
from math import *

a = array([-127,64])
b = array([26,13])
c = (a+b)/2
c0 = array([-4./23, 14./23])
t = 100
def step(t):
     return 1./t

def f(x):
     return sqrt((x[0]-a[0])**2+(x[1]-a[1])**2) +\
      sqrt((x[0]-b[0])**2+(x[1]-b[1])**2) -\
      sqrt((c[0]-a[0])**2+(c[1]-a[1])**2) -\
       sqrt((c[0]-b[0])**2+(c[1]-b[1])**2)

def curr_t(t):
     return int(1.+4/t**2)
def f1(x):
     return 3*x[0]**2 + 5*x[0]*x[1] +\
      4*x[1]**2 - 2*x[0] - 4*x[1] + 27

def diff_f1(x):
     return array([6*x[0] + 5*x[1] -2,\
      5*x[0] +8*x[1] -4])
def proj(x):
     return array([x[0],0])
no = 5000
freq = range(1,7)
color1 = ['r-','g-','b-','c-','m-','y-','k-']
color2 = ['r*','g.','b.','c.','m.','y.','k.']
lam = 0.005
stats = []
for j in freq:
     t = 100
     x0 = array([10,10])
     xcurr1 = x0
     Ini_try=[]
```

```
    xarray = []
    yarray = []
    Ini_try.append(log(f1(xcurr1)-f1(array(c0))))
    print "Freq = "+str(j)
    for i in range(1,no*j):
        xarray.append(xcurr1[0])
        yarray.append(xcurr1[1])
        xcurr1 = xcurr1 - step(t)*diff_f1(xcurr1)
        t = t+1
        if(i%j == 0):

            xcurr1 = proj(xcurr1)
            Ini_try.append(log(f1(xcurr1)-f1(array(c0))))
    figure(1)
    if j == 1:
        plot(range(no), Ini_try, color1[j-1],\
         label = "Normal Projected Gradient Descent")
    else:
        plot(range(no), Ini_try, color1[j-1],\
         label = "Freq = "+str(j))
    plot(range(no), Ini_try, color2[j-1])
    figure(2)
    if j == 1:
        plot(xarray,yarray, color1[j-1],\
         label = "Normal Projected Gradient Descent")
    else:
        plot(xarray,yarray, color1[j-1],\
         label = "Freq = "+str(j))
    plot(xarray,yarray, color2[j-1])
    stats.append(Ini_try[1:])
figure(3)
z = 1
for i in stats[2:]:
    z = z+1

    plot(range(len(i)), [log(abs(i[j]-stats[1][j])) for j in range(le
     color1[z-1],  label = "Freq = "+str(z))

title("Errors between frequencies")
xlabel('Steps')
ylabel('Error')
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))

figure(2)
plot(c0[0],c0[1],color2[-2])
title("Points given by the algorithm")
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))
figure(1)
xlabel('Steps')
ylabel('Error')
title("Saturation Error")
```

```
legend(loc = 'center left', bbox_to_anchor = (1.0, 0.5))

delta = 0.25
x = np.arange(-10.0, 10.0, delta)
y = np.arange(-10.0, 10.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
Z = 10.0 * (Z2 - Z1)
for i in range(len(X[0])):
     for j in range(len(X)):
          Z[i][j] = f1([X[i][j], Y[i][j]])

figure(2)
CS = contour(X, Y, Z)
clabel(CS, inline=1, fontsize=10)
title('Simplest default with labels')
show()
```

APPENDIX A

Convergence in W domain

Theorem 1: The convergence rate of the update algorithm (5.14) in the W domain when H is kept constant is given by,

$$g(W_{n+1}) - g(W^*) \leq \frac{||W_0 - W^*||^2}{n\beta}$$

where $\beta = \left(\frac{L||H^TH||\eta_1^2}{2} - \eta_1\right)$.

Proof 1

$$g_H(Y) \leq g_H(X) + \nabla g_H(X)^T(Y-X) + \frac{L||H_n^TH_n||}{2}||Y-X||^2 \tag{A.1}$$

Thus for our case we have for the n^{th} step that,

$$\begin{aligned} g_H(W_{n+1}) - g_H(W_n) &\leq -\eta_1\nabla g(W)^T\nabla g(W) + \frac{L||H_n^TH_n||\eta_1^2}{2}||\nabla g(W)||^2 \\ &\leq \underbrace{\left(\frac{L||H^TH||\eta_1^2}{2} - \eta_1\right)}_{\beta}||\nabla g(W)||^2 \\ \delta_{n+1} - \delta_n &\leq \beta||\nabla g(W)||^2 \end{aligned}$$

Now we know that,

$$\begin{aligned} g_H(X_n) - g_H(X^*) &\leq \nabla g_H(X_n)^T(X_n - X^*) \\ \delta_n &\leq ||\nabla g_H(W_n)||||(W_n - W^*)|| \\ \frac{\delta_n}{||(W_n - W^*)||} &\leq ||\nabla g_H(W_n)|| \end{aligned} \tag{A.2}$$

For the case when $\beta \leq 0$, we have that,

$$\delta_{n+1} - \delta_n \leq \beta\left(\frac{\delta_n}{||W_n - W^*||}\right)^2$$

Since we have a uniform decrease with n. We have,

$$\delta_{n+1} - \delta_n \;\leq\; \beta \left(\frac{\delta_n}{||W_0 - W^*||} \right)^2$$

After manipulation we get,

$$\frac{1}{\delta_{n+1}} - \frac{1}{\delta_n} \geq \frac{\beta}{||W_0 - W^*||^2} \tag{A.3}$$

Thus summing the telescopic series we have,

$$\begin{aligned} \frac{1}{\delta_{n+1}} - \frac{1}{\delta_0} &\geq \frac{n\beta}{||W_0 - W^*||^2} \\ \frac{1}{\delta_{n+1}} &\geq \frac{n\beta}{||W_0 - W^*||^2} \\ \delta_{n+1} &\leq \frac{||W_0 - W^*||^2}{n\beta} \\ g_H(W_{n+1}) - g_H(W^*) &\leq \frac{||W_0 - W^*||^2}{n\beta} \end{aligned}$$

APPENDIX B

Convergence in H domain

Theorem **??**: The convergence rate of the update algorithm (5.20) in the W domain when H is kept constant is given by,

$$p_W(H_{n+1}) - p_W(H^*) \leq \frac{||H_0 - H^*||^2}{n\alpha}$$

where $\alpha = \left(\frac{L||W^TW||\eta_1^2}{2} - \eta_1\right)$.

Proof 2

$$p_W(Y) \leq p_W(X) + \nabla p_W(X)^T(Y-X) + \frac{L||WW^T||}{2}||Y-X||^2 \tag{B.1}$$

Thus for our case we have for the n^{th} step that,

$$\begin{aligned} p_W(H_{n+1}) - p_W(H_n) &\leq -\eta_2 \nabla p_W(H)^T \nabla p_W(H) + \frac{L||WW^T||\eta_2^2}{2}||\nabla p_W(H)||^2 \\ &\leq \underbrace{\left(\frac{L||WW^T||\eta_2^2}{2} - \eta_1\right)}_{\alpha} ||\nabla p_W(H)||^2 \\ \delta_{n+1} - \delta_n &\leq \alpha||\nabla p_W(H)||^2 \end{aligned}$$

Now we know that,

$$\begin{aligned} p_W(X_n) - p_W(X^*) &\leq \nabla p_W(X_n)^T(X_n - X^*) \\ \delta_n &\leq ||\nabla p_W(H_n)||||(H_n - H^*)|| \\ \frac{\delta_n}{||(H_n - H^*)||} &\leq ||\nabla p_W(H_n)|| \end{aligned} \tag{B.2}$$

For the case when $\alpha \leq 0$, we have that,

$$\delta_{n+1} - \delta_n \leq \alpha\left(\frac{\delta_n}{||H_n - H^*||}\right)^2$$

Since we have a uniform decrease with n. We have,

$$\delta_{n+1} - \delta_n \quad \leq \quad \alpha \left(\frac{\delta_n}{||H_0 - H^*||} \right)^2$$

After manipulation we get,

$$\frac{1}{\delta_{n+1}} - \frac{1}{\delta_n} \geq \frac{\alpha}{||H_0 - H^*||^2} \tag{B.3}$$

Thus summing the telescopic series we have,

$$\begin{aligned}
\frac{1}{\delta_{n+1}} - \frac{1}{\delta_0} &\geq \frac{n\alpha}{||H_0 - H^*||^2} \\
\frac{1}{\delta_{n+1}} &\geq \frac{n\alpha}{||H_0 - H^*||^2} \\
\delta_{n+1} &\leq \frac{||H_0 - H^*||^2}{n\alpha} \\
p_W(H_{n+1}) - p_W(H^*) &\leq \frac{||H_0 - H^*||^2}{n\alpha}
\end{aligned}$$

APPENDIX C

Convergence in X Domain

Theorem **??**: The convergence rate of the update algorithm in the X domain (5.21) is given by,

$$X_{n+1} - X^* \leq \frac{1}{\sum_{i=0}^{n} \frac{\alpha_{n-i}}{\Pi_{j=0}^{i} c_{n-j}}}$$

where $\alpha_n = \left(\frac{L||H_n H_n^T||\eta_2^2}{2} - \eta_2\right)$ and $c_n = (1 + \alpha_n \Delta_{H_n})$.

Proof 3

$$\begin{aligned} g(W_{n+1}) - g(W_n) &\leq \left(\frac{L||H_n^T H_n||\eta_1^2}{2} - \eta_1\right) ||\nabla g(W_n)||^2 \\ f(W_{n+1}H_n) - f(W_n H_n) &\leq \left(\frac{L||H_n^T H_n||\eta_1^2}{2} - \eta_1\right) ||\nabla f(W_n H_n) H_n^T||^2 \end{aligned}$$

and

$$\begin{aligned} p(H_{n+1}) - p(H_n) &\leq \left(\frac{L||W_n W_n^T||\eta_2^2}{2} - \eta_2\right) ||\nabla p(H_n)||^2 \\ f(W_n H_{n+1}) - f(W_n H_n) &\leq \left(\frac{L||W_n W_n^T||\eta_2^2}{2} - \eta_2\right) ||W_n \nabla f(W_n H_n)||^2 \\ f(W_{n+1}H_{n+1}) - f(W_{n+1}H_n) &\leq \left(\frac{L||W_{n+1} W_{n+1}^T||\eta_2^2}{2} - \eta_2\right) ||W_{n+1} \nabla f(W_{n+1} H_n)||^2 \end{aligned}$$

Thus,

$$f(W_{n+1}H_{n+1}) - f(W_n H_n) \leq \left(\frac{L||W_{n+1} W_{n+1}^T||\eta_2^2}{2} - \eta_2\right) ||W_{n+1} \nabla f(W_{n+1} H_n)||^2 + \left(\frac{L||H_n^T H_n||\eta_1^2}{2} -\right.$$

Or we can also go the other way,

$$f(W_{n+1}H_{n+1}) - f(W_n H_n) \leq \left(\frac{L||W_n W_n^T||\eta_2^2}{2} - \eta_2\right) ||W_n \nabla f(W_n H_n)||^2 + \left(\frac{L||H_{n+1}^T H_{n+1}||\eta_1^2}{2} - \eta_1\right)$$

We know that,

First

$$\begin{aligned} \frac{p(H_n) - p(H^*)}{||(H_n - H^*)||} &\leq ||\nabla p(H_n)|| \\ \frac{f(W_nH_n) - f(W_nH^*)}{||(H_n - H^*)||} &\leq ||W_n^T \nabla f(W_nH_n)|| \end{aligned} \tag{C.1}$$

Second

$$\begin{aligned} \frac{g(X_n) - g(X^*)}{||(W_n - W^*)||} &\leq ||\nabla g(W_n)|| \\ \frac{f(W_nH_n) - f(W^*H_n)}{||(W_n - W^*)||} &\leq ||\nabla f(W_nH_n)H_n^T|| \end{aligned} \tag{C.2}$$

Let

$$\Delta_{W_n} = f(W_nH^*) - f(W^*H^*)$$

$$\Delta_{H_n} = f(W^*H_n) - f(W^*H^*)$$

and

$$\delta_n = f(X_n) - f(X^*) = f(W_nH_n) - f(W^*X^*)$$

.

Thus from the above few equations we get,

$$\begin{aligned} \delta_{n+1} - \delta_n \leq \left(\frac{L||W_{n+1}W_{n+1}^T||\eta_2^2}{2} - \eta_2\right)(\delta_{n+1} - \Delta_{W_{n+1}})^2 + \left(\frac{L||H_n^TH_n||\eta_1^2}{2} - \eta_1\right)(\delta_n - \Delta_{H_n})^2 \\ \delta_{n+1} - \delta_n \leq \left(\frac{L||W_nW_n^T||\eta_2^2}{2} - \eta_2\right)(\delta_n - \Delta_{W_n})^2 + \left(\frac{L||H_{n+1}^TH_{n+1}||\eta_1^2}{2} - \eta_1\right)(\delta_{n+1} - \Delta_{H_{n+1}})^2 \end{aligned} \tag{C.3}$$

Let

$$\beta_n = -\left(\frac{L||W_nW_n^T||\eta_2^2}{2} - \eta_2\right)$$

$$\alpha_n = -\left(\frac{L||H_n^TH_n||\eta_1^2}{2} - \eta_1\right)$$

(Just to make the constants positive)

Thus for the telescopic series we have,

Telescopic Series 1

$$
\begin{aligned}
\delta_{n+1} - \delta_n &\leq -\beta_{n+1}(\delta_{n+1} - \Delta_{W_{n+1}})^2 - \alpha_n(\delta_n - \Delta_{H_n})^2 \\
&\leq -\beta_{n+1}(\delta_{n+1}^2 - 2\delta_{n+1}\Delta_{W_{n+1}} + \Delta_{W_{n+1}}^2) - \alpha_n(\delta_n^2 - 2\delta_n\Delta_H \\
(1 - 2\beta_{n+1}\Delta_{W_{n+1}})\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\beta_{n+1}(\delta_{n+1}^2 + \Delta_{W_{n+1}}^2) - \alpha_n(\delta_n^2 + \Delta_{H_n}^2) \\
(1 - 2\beta_{n+1}\Delta_{W_{n+1}})\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\beta_{n+1}\delta_{n+1}^2 - \alpha_n\delta_n^2 \\
(1 - 2\beta_{n+1}\Delta_{W_{n+1}})\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\alpha_n\delta_n^2
\end{aligned}
$$

Telescopic Series 2

$$
\begin{aligned}
\delta_{n+1} - \delta_n &\leq -\beta_{n+1}(\delta_{n+1} - \Delta_{W_{n+1}})^2 - \alpha_n(\delta_n - \Delta_{H_n})^2 \\
&\leq -\alpha_n(\delta_n^2 - 2\delta_n\Delta_{H_n} + \Delta_{H_n}^2) \\
\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\alpha_n(\delta_n^2 + \Delta_{H_n}^2) \\
\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\alpha_n\delta_n^2 \\
\delta_{n+1} - (1 + \alpha_n\Delta_{H_n})\delta_n &\leq -\alpha_n\delta_n^2
\end{aligned}
\tag{C.5}
$$

We can go further with telescopic series 2 as follows,

Let

$$c_n = (1 + \alpha_n\Delta_{H_n})$$

. Thus we have,

$$\begin{aligned}
\delta_{n+1} - c_n\delta_n &\leq -\alpha_n\delta_n^2 \\
\frac{1}{\delta_n} - \frac{c_n}{\delta_{n+1}} &\leq -\alpha_n\frac{\delta_n}{\delta_{n+1}} \\
\frac{1}{\delta_n} - \frac{c_n}{\delta_{n+1}} &\leq -\alpha_n \\
\frac{1}{\delta_{n+1}} - \frac{1}{c_n\delta_n} &\geq \frac{\alpha_n}{c_n} \\
\frac{1}{\delta_{n+1}} - \frac{1}{c_nc_{n-1}\delta_{n-1}} &\geq \frac{\alpha_n}{c_n} + \frac{\alpha_{n-1}}{c_nc_{n-1}} \\
\frac{1}{\delta_{n+1}} - \frac{1}{\delta_0\Pi_{i=0}^n c_i} &\geq \sum_{i=0}^{n}\frac{\alpha_{n-i}}{\Pi_{j=0}^i c_{n-j}} \\
\frac{1}{\delta_{n+1}} &\geq \sum_{i=0}^{n}\frac{\alpha_{n-i}}{\Pi_{j=0}^i c_{n-j}} \\
\delta_{n+1} &\leq \frac{1}{\sum_{i=0}^n \frac{\alpha_{n-i}}{\Pi_{j=0}^i c_{n-j}}}
\end{aligned} \tag{C.6}$$

REFERENCES

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

2. **Bhojanapalli, S.**, **A. Kyrillidis**, and **S. Sanghavi** (2015). Dropping convexity for faster semi-definite optimization. *arXiv preprint*.

3. **Boyd, S.** and **L. Vandenberghe**, *Convex optimization*. Cambridge university press, 2004.

4. **Golub, G. H.** (1996). Cf van loan matrix computations. *The Johns Hopkins*.

5. **Helmke, U.**, **M. Prechtel**, and **M. A. Shayman** (1993). Riccati-like flows and matrix approximations. *Kybernetika*, **29**(6), 563–582.

6. **Horn, R. A.** and **C. R. Johnson** (1985). Matrix analysis cambridge university press. *New York*.

7. **Tu, S.**, **R. Boczar**, **M. Soltanolkotabi**, and **B. Recht** (2015). Low-rank solutions of linear matrix equations via procrustes flow. *arXiv preprint arXiv:1507.03566*.